Modeling Spatiotemporal Patterns of PM 2.5 at the Sub-Neighborhood Scale Using Low-Cost Sensor Networks

Philip Jeffrey Orlando
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Environmental Sciences Commons

Let us know how access to this document benefits you.

Recommended Citation

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDX Scholar. For more information, please contact pdxscholar@pdx.edu.
Modeling Spatiotemporal Patterns of PM$_{2.5}$ at the Sub-Neighborhood Scale Using Low-Cost Sensor Networks

by

Philip Jeffrey Orlando

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Environmental Science and Management

Thesis Committee:
Linda George, Chair
Meenakshi Rao
Kelly Gleason
Lynne Messer

Portland State University
2019
Epidemiological research has demonstrated an adverse relationship between fine particulate matter (PM$_{2.5}$) and human health. While PM$_{2.5}$ continues to pose a significant global health risk, there is still the need to further characterize exposures at the intra-urban scale. Land use regression is a statistical modeling technique which is used to predict air pollution concentrations at high resolution from a limited number of monitoring sites. However, the existing regulatory monitoring networks are typically not dense enough to apply these techniques. We explored the potential of using low-cost PM$_{2.5}$ sensor networks to overcome the limitations of the existing regulatory monitoring infrastructure, and identified the need to determine sensor-specific correction factors based on the local PM$_{2.5}$ source profile. Once calibrated, a land use regression model ($R^2 = 0.89$) was developed using the low-cost sensor network ($n \approx 20$), alongside several land use and meteorological variables, to predict daily particulate matter concentrations at a 50 m spatial resolution during a two year period within Portland, Oregon. From this model, we assessed the relative strengths of expected sources and sinks of fine particulate matter, focusing specifically on the role that the urban canopy may play in mitigating PM$_{2.5}$ exposure. This model showed a modest but observable spatial pattern in PM$_{2.5}$, but attributed the majority of PM$_{2.5}$ variation to temporal predictors (e.g. ambient background PM$_{2.5}$, wind speed, temperature). Neither proxies for traffic-related sources, or vegetation-related sinks were identified as significant predictors of PM$_{2.5}$. Our research also demonstrated the importance of sensor placement, as a considerably different set of predictors was selected after the inclusion of four additional monitoring
sites. Future work will apply this method to four cities with a varying degree of canopy cover to assess differences in intra-urban gradients of PM$_{2.5}$ and to further characterize the influence of vegetation.
Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor Dr. Linda George for her continuous patience, guidance, and expertise throughout my graduate study and research. I never would have imagined pursuing a graduate degree without her inspiring wisdom and dedication to science.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Meenakshi Rao, Dr. Kelly Gleason, and Dr. Lynne Messer for their time, encouragement, and insightful feedback.

My sincere thanks also goes to the remaining members of the Canopy Continuum project: Dr. Vivek Shandas and Dr. Todd Rosenstiel for their helpful comments and support.

Another special thanks goes to Meenakshi Rao, as well as, Dr. Yangdong Pan and Jackson Voelkel for cultivating my appreciation of computer science and for teaching me how to harness it to tackle environmental problems.

I thank my fellow labmates in the Sustainable Atmospheres Research lab: Devin Wilde, Dat Vo, Kirsten Sarle, Modi Raduma, Patrick Léon Gross, and Kellen McInerney for their field and laboratory assistance.

I would like to thank my partner, Judit Takács, for her continual moral support and patience while I was often absorbed by my graduate work.
Last but not least, I would like to thank my parents: Shannon and Jeff Orlando for their endless encouragement and support throughout my entire life.
Contents

Abstract

Acknowledgements

List of Tables

List of Figures

1 Introduction

1.1 Background

1.1.1 Particulate Matter and Human Health

1.1.2 Characterizing Particulate Matter Pollution

1.1.3 Urban Air Quality and Traffic

1.1.4 Influence of Vegetation on Air Quality

1.2 Modeling Intra-Urban PM

1.2.1 The Lack of Spatially Resolved PM

1.2.2 Land Use Regression Modeling of PM

1.2.3 Opportunities and Challenges of Low-Cost Sensors

1.3 Research Objectives

2 Methodology

2.1 Low-Cost Sensor Selection

2.1.1 Sensor Performance

2.1.2 PurpleAir Device Ownership and Open Data Philosophy

2.2 Laboratory Evaluations

2.3 PurpleAir Data Acquisition and Storage

2.4 Opportunistic Ambient Calibration of PurpleAir Networks

2.5 Data Recovery and Sensor Lifetime

2.6 Sensor Allocation

2.7 Data Compilation

2.7.1 Spatial and Temporal Predictors of PM

15

1

1

2

4

5

7

7

9

12

14

15

15

17

17

19

21

23

23

31

33
List of Tables

2.1 Table of Predictors ............................................. 46
3.1 Lab Evaluations Summary Statistics ......................... 49
3.2 VIF Scores ....................................................... 60
3.3 LOOCV Summary Table ......................................... 63
3.4 Beta Coefficients of Final Model .............................. 75
3.5 Beta Coefficients of Additional Model ....................... 82
List of Figures

2.1 PurpleAir $R^2$ as a Function of Distance ........................................ 25
2.2 Albuquerque Site Map ................................................................. 27
2.3 Boise Site Map ................................................................................. 28
2.4 Sacramento Site Map ........................................................................ 29
2.5 Tacoma Site Map .............................................................................. 30
2.6 Workflow Diagram ............................................................................. 33
2.7 Focal Buffers of RLIS Streets ............................................................. 41

3.1 $R^2$ Distributions from Lab Evaluations ........................................ 49
3.2 Slope Distributions from Lab Evaluations ........................................ 50
3.3 Scatter Plot of PurpleAir Primary Sensor and DustTrak .................... 51
3.4 Scatter Plot of PurpleAir Secondary Sensor and DustTrak ............... 52
3.5 $R^2$ Distributions from Field Evaluations ........................................ 55
3.6 Slope Distributions from Field Evaluations ...................................... 56
3.7 Scatter Plot of PurpleAir Primary Sensor and Nephelometer ........... 57
3.8 Scatter Plot of PurpleAir Secondary Sensor and Nephelometer ....... 58
3.9 LOOCV Observation Bar Plot ........................................................... 62
3.10 Scatter Plot Comparing Test Data and LUR Predictions .................. 66
3.11 LUR Test Assumptions ................................................................. 67
3.12 Scatter Plot Comparing Test Data and RF Predictions ..................... 68
3.13 Map of LUR Standard Error .......................................................... 77
3.14 PM$_{2.5}$ from both LUR Model Predictions ...................................... 87
3.15 Scatter Plot Comparing LUR Model Predictions .............................. 88
3.16 PM$_{2.5}$ from both Random Forest Model Predictions ..................... 89
3.17 Predictions During the Wildfire Season .......................................... 90
3.18 Predictions During the Winter Season ............................................ 91
Chapter 1

Introduction

1.1 Background

1.1.1 Particulate Matter and Human Health

Epidemiological studies have demonstrated an adverse relationship between ambient particulate matter (PM) exposure and human health (Kloog et al., 2013; Brunekreef and Holgate, 2002; Dockery et al., 1993). Adverse health effects from PM exposure include cardiovascular and respiratory morbidity and mortality (Le Tertre et al., 2002; Schwartz, 1994; Pope and Dockery, 2006). The Global Burden of Disease (GBD) comparative risk assessment conducted by the World Health Organization (WHO) attributed approximately 3.2 million year-2010 deaths to ambient PM$_{2.5}$ (PM with a diameter of 2.5 µm or smaller) exposure, and ranked this pollutant as the sixth largest overall risk factor for shortened life expectancy globally (Lim et al., 2012; Apte et al., 2015; Fann et al., 2012; Cohen et al., 2005). To provide further context, the global health burden from ambient PM$_{2.5}$ is greater than the cumulative health risks of other well-established global health threats including HIV-AIDS and malaria which contributed 1.5 million and 1.2 million year-2010 deaths respectively (Lim et al.,
While ambient PM continues to pose a significant global health burden, there is the need to further characterize higher granularity PM exposure gradients, specifically at the intra-urban scale. Health studies typically rely on inter-city estimates from a limited number of monitoring sites to develop a single exposure estimate for a given city. These single estimates are then assigned to an entire metropolitan area, despite the potential for intra-city heterogeneity in PM sinks and sources. This likely contributes to the underestimation of PM-related health effects in the vicinity of sources (Ross et al., 2007; Jerrett et al., 2005; Hoek et al., 2002).

### 1.1.2 Characterizing Particulate Matter Pollution

PM is a mixture of solid and liquid particles suspended in the air that vary in shape, number, chemical composition and size. PM is often described by three major size categories, the largest being coarse respirable PM, or PM$_{10}$. These are particles with a diameter less than 10 $\mu$m but greater than 2.5 $\mu$m, and are mostly derived from biogenic or geogenic sources including the suspension or resuspension of dust, soil, windstorms, volcanoes, pollen, mold, spores, and sea salt. PM$_{10}$ is also attributed to anthropogenic sources including agricultural activities and mining (Pope and Dockery, 2006). Smaller than coarse PM are fine PM, or PM$_{2.5}$. These are particles that have a diameter less than or equal to 2.5 $\mu$m. Fine PM are derived primarily from anthropogenic sources, usually the direct result from some form of combustion. This includes vehicle emissions, wood burning, coal burning, and industrial processes (steel mills, cement plants, paper mills). In addition to primary sources of fine PM, there are also
secondary sources, where particles form as the result of some chemical transformation in the atmosphere. Secondary fine PM is often the result of nitrogen and sulfur oxides transforming into nitrates and sulfates, or through the formation of secondary organic aerosols (SOAs) from volatile organic compound emissions (VOCs) (Pope and Dockery, 2006). Even smaller than fine PM are the ultrafine PM. While their definition appears to range slightly throughout the literature, these are typically particles which have an aerodynamic diameter less than 0.1 µm (Pope and Dockery, 2006; Oberdorster, Oberdorster, and Oberdorster, 2005; EPA, 2004). Similar to fine PM, ultrafines are primarily derived from some source of combustion. However, ultrafines have an extremely short lifetime (minutes to hours), and will rapidly coagulate or condense to form larger PM\(_{2.5}\) particles, which can remain suspended in the atmosphere and have much longer residence time (days to weeks). Due to these properties, ultrafine PM are often indicative of freshly-emitted PM from local sources.

The human health impacts from PM exposure have been shown to vary depending on the size of the particle. Generally, the smaller the particles, the deeper into the lungs they can infiltrate, leading to more severe health effects. While ultrafine PM may be more capable than fine PM of transferring from the lung to the blood and other parts of the body, there has been a focus on monitoring only fine and coarse PM throughout regulatory agencies due to the cost and reproducibility of ultrafine measurements. Fine PM, in contrast to larger particles, can remain suspended in the atmosphere for longer periods of time, can be breathed more deeply into the lungs, and are more capable of infiltrating indoor environments. In addition, PM\(_{10}\) levels throughout the US are generally
within compliance, whereas PM$_{2.5}$ remains a regulatory challenge.

1.1.3 Urban Air Quality and Traffic

Air pollution can vary across spatiotemporal scales (Gilliland Frank et al., 2005; Beckerman et al., 2013). Spatially, local-scale variations are the result of primary sources, whereas regional-scale variations occur from secondary reactions and transport mechanisms. The majority of temporal variation is the result of diurnal traffic patterns or meteorology (Beckerman et al., 2013). Generally, pollutants directly emitted from mobile sources (traffic-related emissions), or indirectly via photochemical reactions, still dominate the urban environment despite significant improvements in fuel, engine, and emission control technology over the last few decades (Fenger, 1999; Hoek et al., 2000; Nyberg et al., 2000). It is imperative that we characterize air pollution gradients within cities as population density and traffic emissions are high within these environments and exposure is not expected to be experienced evenly (Vardoulakis et al., 2003; Rao et al., 2014). The spatial variability of primary air pollution is higher near major intersections and busy streets within the built environment, where traffic emissions are higher and ventilation is reduced due to the local topography (Harrison, Jones, and Barrowcliffe, 2004; Vardoulakis et al., 2003). In contrast, the spatial variability of secondary pollutants is driven by meteorology and generally regional in scale. Many studies have identified an increase in respiratory and cardiovascular problems from living near major roadways (Jerrett et al., 2008; Brugge, Durant, and Rioux, 2007; Zhou and Levy, 2007). Identifying intra-urban pollution gradients will become increasingly relevant as rising rates
Chapter 1. Introduction

of urbanization and high density development result in greater human exposure to near-road environments. Currently, roughly 80% of the US population currently live within metropolitan areas (and approximately half of the global population), and the global urban population is expected to increase to approximately 68% by the year 2050 (Bureau, 2012; United Nations, 2018).

A meta-analysis conducted by Karner, Eisinger, and Niemeier, 2010 found that many air pollutants within cities have strong pollution gradients, decaying within 150 m - 200 m from the source, and reaching background levels between 200 m - 1000 m (Karner, Eisinger, and Niemeier, 2010). These fine-scale gradients from road sources are well documented for NO, NO\textsubscript{2}, PNC, CO, and PM\textsubscript{10} (Karner, Eisinger, and Niemeier, 2010; Rao et al., 2014; Zhou and Levy, 2007). However, patterns in PM\textsubscript{2.5} mass concentration as distance from road sources increases are less pronounced. While PM\textsubscript{2.5} is a component of traffic-related emissions, mass-based measurements do not always capture this influence. The spatial patterns of PM\textsubscript{2.5} are mixed, either decreasing very gradually from traffic sources, or showing no clear trend at all (Karner, Eisinger, and Niemeier, 2010; Kendrick, Koonce, and George, 2015).

1.1.4 Influence of Vegetation on Air Quality

There is an increasing body of evidence that suggests that access to greenery can improve human health. A pioneering study conducted by Ulrich showed that hospital patients recovering from surgery were discharged sooner and required less pain medication if they had a view of greenspace from their hospital bed. This was in comparison to patients whose hospital room window
only provided a view of a brick wall (Ulrich, 1984). Subsequent studies have shown that access to greenspace is associated with reduced morbidity (Maas et al., 2009), mortality (Mitchell and Popham, 2007), and lower obesity rates (Bell, Wilson, and Liu, 2008). In addition, recent studies have shown an association between urban greenery (specifically trees) and air quality, suggesting that the urban canopy has a small yet significant effect on reducing pollutants including NO$_2$, PM$_{10}$, O$_3$, and SO$_2$ (Donovan et al., 2011; Rao et al., 2014; Nowak, Crane, and Stevens, 2006). While some studies found that vegetation may only improve the average air quality by $<$1-2%, this statistic accounts for multiple pollutants, and the actual magnitude of pollution mitigation can be substantial, typically on the order of $10^2$ - $10^3$ metric tons per year in a given city (Nowak, Crane, and Stevens, 2006). Even when focusing on just a single pollutant, for example NO$_2$, a recent study in Portland, Oregon attributed pollution removal by trees to providing a $7$ million USD annual benefit in reduced occurrences of respiratory complications (Rao et al., 2014). Given that trees are a fundamental component of the urban environment, with a 35% average tree cover in US cities (Nowak and Greenfield, 2012), and that rates of urbanization are expected to rapidly increase over the next several decades, it is crucial that we continue to assess the potential role the urban canopy may play in reducing even more harmful air pollutants such as PM$_{2.5}$.

Trees can affect atmospheric PM concentrations by removal (Beckett, Freer-Smith, and Taylor, 2000) and emission (e.g. pollen). Particles that physically deposit on the leaf surface can also be resuspended, often by precipitation or when leaves, twigs, and branches fall to the ground. While some of the particles
can be absorbed into the tree, most are confined to the plant surface and are ultimately resuspended. As a result, trees are considered only temporary sinks for many atmospheric particles (Nowak et al., 2013). Most of the studies related to PM and urban trees have focused on PM$_{10}$ (Nowak et al., 2013), and while some studies have evaluated the removal rate and suspension of PM$_{2.5}$ by specific tree species (Beckett, Freer-Smith, and Taylor, 2000; Freer-Smith, El-Khatib, and Taylor, 2004; Freer-Smith, Beckett, and Taylor, 2005; Pullman, 2008), few have estimated the effect of trees on PM$_{2.5}$ concentrations at the city-scale (Nowak et al., 2013). There is some evidence suggesting PM$_{2.5}$ mitigation by urban trees is considerably lower relative to PM$_{10}$ in terms of mass, but the health benefits are significantly higher (Nowak, Crane, and Stevens, 2006; Nowak et al., 2013). The benefits the urban canopy provides with PM$_{2.5}$ removal is still not well understood, and only a limited number of studies have attempted to evaluate this effect at the intra-urban scale (Nowak et al., 2013; Jeanjean, Monks, and Leigh, 2016; Tallis et al., 2011). This is due, in part, to the challenges inherent to creating spatially resolved PM$_{2.5}$ exposures surfaces at this resolution.

1.2 Modeling Intra-Urban PM$_{2.5}$

1.2.1 The Lack of Spatially Resolved PM$_{2.5}$

The lack of spatially resolved PM$_{2.5}$ at the city-scale is mostly due to the practical constraints of air quality monitoring networks. Due to the high cost, cumbersome design, and the skilled labor required to establish and maintain research-grade air quality instrumentation, the current distribution of monitoring stations is confined to a limited number of near-road and urban background sites.
within a given city (Vardoulakis, Solazzo, and Lumbreras, 2011). For example, within our Portland, Oregon study area, approximately 4-6 PM$_{2.5}$ monitoring stations were operational during the two year study period. This network, maintained by Oregon Department of Environmental Quality (ODEQ) consisted of a couple rural ambient background sites, urban ambient background sites, and a single near-road site. Reference monitoring stations, typically maintained by local or regional air quality management agencies, are primarily established to assess regulatory compliance and develop a regional air quality index. However, the use of these monitors in epidemiological research, specifically their use in modeling spatially resolved PM surfaces at sub-neighborhood scales (0.01 - 1.0 km), is largely an afterthought. This focus on compliance ultimately results in the potential lack of representativeness of fixed air quality monitoring sites in epidemiological studies. In order to account for the spatial heterogeneity of air pollution sources in urban areas, several modeling techniques have been developed to spatially interpolate PM$_{2.5}$ in order to better estimate the human health effects from PM$_{2.5}$ exposure (Vardoulakis, Solazzo, and Lumbreras, 2011). These include land use regression (LUR) and dispersion modeling techniques (Briggs et al., 2000; Vardoulakis, Solazzo, and Lumbreras, 2011; Hoek et al., 2008; Zhou and Levy, 2008). For the purpose of this study, we applied LUR techniques to develop spatially resolved PM$_{2.5}$ exposures necessary for assessing the relative power of land use sources and sinks, with a focus on the influence of vegetation.
1.2.2 Land Use Regression Modeling of PM$_{2.5}$

Land use regression (LUR) is a statistical modeling approach which is used to spatially predict continuous air pollution concentrations at high resolution from a limited number of monitoring sites (Briggs et al., 2000; Rao et al., 2014). Land-use and land-cover (LULC) variables are extracted within an appropriate buffer distance from each monitoring location using spatial analysis software. A multiple linear regression model is then developed using the monitored air pollution data as a response variable. The model is then assessed to see how well it meets its test assumptions. These assumptions include (1) a linear relationship exists between the response variable and predictors, (2) model residuals are normally distributed, (3) variation of observations around the line of best fit is constant (homoscedasticity), (4) and strong multicollinearity among predictors is not observed. This method is rooted in the principles that the environmental conditions for the air pollutant response variable of interest can be determined from a limited number of readily available predictors; and that the relationship between the response variable and predictors can be evaluated using a small sample of ‘training’ data (Briggs et al., 2000). After validation, the LUR model can be used to make spatial predictions of air pollution in between and beyond the original monitoring network used during the training phase. Briefly, LUR models of air pollutants typically rely on predictors including population density, land use, and traffic-related variables to characterize mean pollutant concentrations within several buffer distances at specific monitoring sites. Typical buffer sizes range between 50 m - 3000 m (Eeftens et al., 2012; Jerrett et al., 2007; Rao et al., 2014; Liu et al., 2016). Once the model performance has been evaluated, this
technique can be used to make air quality predictions without the presence of a monitoring station, essentially extrapolating pollutant exposure throughout the remaining study area by relying on the patterns observed within the predictor variables.

While land use regression models are capable of providing spatially resolved air pollutant surfaces at fine scales, they still rely on observational data from a limited number of monitoring sites for model development, calibration, and validation (Jerrett et al., 2005; Vardoulakis, Solazzo, and Lumbreras, 2011). As a result, the performance of LUR models depends on monitoring locations which provide a representative sample for the remaining study area that is to be predicted. In addition, regression mapping techniques are based on assumptions regarding their predictors, or independent variables.

Numerous LUR studies have been conducted to assess fine-scale spatially resolved NO$_2$ exposures because low-cost passive sampler technology has enabled the deployment of high-density monitoring networks (Rao et al., 2014; Briggs et al., 2000; Eeftens et al., 2012; Hoek et al., 2008). In contrast, LUR models for particulate matter are less numerous because they necessitate a more intensive monitoring campaign, despite evidence suggesting that adverse health effects are tied more closely to PM exposure than nitrogen oxides (Eeftens et al., 2012; Hoek et al., 2008; Pope et al., 2002; Sarnat et al., 2001). The ESCAPE Project (European Study of Cohorts for Air Pollution Effects), a pioneering LUR study focusing on PM$_{2.5}$, produced one of the largest spatially resolved PM databases in Europe, and relied on a total of 20 - 40 monitoring sites within
each of the 20 European cities examined (Eeftens et al., 2012). However, existing regulatory monitoring networks throughout our five US cities of interest within this study are not dense enough to develop LUR models predicting intra-urban PM$_{2.5}$ (e.g. Portland n = 4, Boise n = 2, Albuquerque n = 5, Tacoma n = 2, and Sacramento n = 7) (according to the EPA’s AirData Active PM$_{2.5}$ Map) (US EPA, 2016). Due to the aforementioned limitations of deploying high-density networks with research-grade monitoring equipment, we opted to explore the potential of commercially available low-cost PM$_{2.5}$ sensors to achieve the recommended network density of $\geq 20$ per city.

Previously developed LUR models of PM$_{2.5}$ within the ESCAPE project have shown moderate to good explained variance, ranging from $R^2 = 35\%$, to $R^2 = 94\%$, with a median of $R^2 = 71\%$ (Eeftens et al., 2012). This range in variance explained is partially due to limited availability of relevant GIS predictor variables across cities. As a result, no single model could be applied to each city. Instead, area-specific predictors were selected for each city’s model due to these differences in available predictors. Better performing models were observed to have local traffic data available, specifically traffic intensity, and models without local or limited traffic data performed below the median variance explained (71%). Traffic variables were selected in 18 of the 20 models developed. Less important predictors included residential land use, population density, industrial land use and ports, and natural land use (Eeftens et al., 2012). The differences between final model $R^2$ and Leave-One-Out Cross-Validation (LOOCV) $R^2$ was less than 15% for most models, indicating overall model stability (Eeftens et al., 2012).
1.2.3 Opportunities and Challenges of Low-Cost Sensors

Optical Particle Counters (OPCs), also referred to as photometers, are commonly used for particle monitoring because of their ability to provide continuous data, ease of operation, small size, and relatively low cost. This is in contrast to gravimetric techniques which, despite being the most accurate form of measurement, do not provide continuous data and are costly to routinely operate and maintain. Briefly, gravimetric PM samplers actively pull an air sample at a constant flow rate through a series of impactors to collect size-selected PM (e.g. PM$_{2.5}$ or PM$_{10}$) on a filter paper over a sample period of $\geq$24 hours. Once a measurable amount of PM$_{2.5}$ has been collected after this sample period, the filters are processed and weighed in a lab. The total PM$_{2.5}$ mass is divided by the total volume of air that was sampled throughout the entire sample interval to determine a mean mass density (e.g. $10 \mu g m^{-3}$). This method requires a technician to routinely install and collect filters at the beginning and end of each sample period. In contrast to collecting and weighing PM samples, OPCs rely on a laser beam and light receptor (usually a photodiode) to measure the light scattered from an individual illuminated aerosol particle due to Mie scattering (Chun et al., 2001; Kim, 2004; Kohli and Mittal, 2015). The particle number concentration (PNC) is then determined by counting the number of scattered light pulses within a specific time interval. The particle diameter is also determined by measuring the height of these light pulses. The PNC is typically discretized into regulatory or epidemiologically relevant size bins (e.g. PM$_{2.5}$, PM$_{10}$). Once discretized, PNC can be used to approximate PM mass density, but these standalone values should be interpreted carefully. OPCs cannot be used to
determine aerosol type and some concessions must be made before PNC can be converted into a mass density measurement. For example, OPCs are not able to account for several aerosol properties, including particle density, shape, refractive index, and absorption. As a result, assumptions are made regarding each of these properties, as OPCs report light-scattering equivalent diameters instead of the true physical diameter and mass density (Kobayashi et al., 2014; Kohli and Mittal, 2015). Examples of some of these assumptions include aerosol particles having an aerodynamic diameter (particles are assumed to be spherical), with a constant density of 1 g cm$^{-3}$ (density of water), and a constant refractive index or absorption. As a result, OPCs may be more adept at approximating PM mass of certain sources than others, which often depends on source of PM used during the factory calibration. This information is often not disclosed by OPC manufacturers, as the exact algorithms used to convert PNC to particle mass are proprietary.

Despite the challenges of using OPCs to derive PM$_{2.5}$ mass density, the potential of these low-cost sensor networks in producing spatially resolved exposure maps at finer scales relative to the existing monitoring infrastructure is promising. If individual correction factors can be established which correspond to the ambient PM$_{2.5}$ source profile of a specific urban area, then low-cost sensor networks may provide an advantage over traditional monitoring techniques for LUR modeling.
Chapter 1. Introduction

1.3 Research Objectives

This thesis focuses on the development of (a) an opportunistic ambient calibration method for correcting low-cost PM$_{2.5}$ sensor networks in order to use them for intra-urban modeling and (b) a LUR modeling technique which relies on this corrected sensor network data to predict spatially resolved PM$_{2.5}$ throughout Portland, Oregon in order to assess the drivers of PM$_{2.5}$ intra-urban variability, with focus on vegetation. This thesis is part of a larger epidemiological study entitled the Canopy Continuum Project, funded by the USFS. The method developed within this thesis will be subsequently applied to four additional cities within the USFS Canopy Continuum study (Albuquerque, New Mexico, Boise, Idaho, Sacramento, California, and Tacoma, Washington).

The analytical goal of this study is to develop a predictive model to produce spatially resolved PM$_{2.5}$ surfaces at a spatial resolution of 50 m and a daily temporal resolution throughout a 2 year period. These fine-scale exposures will be developed for later use in an epidemiological study which characterizes the effects of several environmental stressors, including urban heat, NO$_2$, and PM$_{2.5}$ on prenatal health, and assesses the extent the urban canopy may play in helping mitigate prenatal exposure to these environmental stressors.
Chapter 2

Methodology

2.1 Low-Cost Sensor Selection

2.1.1 Sensor Performance

We selected the PurpleAir-II SD (PurpleAir) for this study due to their previously demonstrated performance and relatively low-cost ($250). The South Coast Air Quality Management District’s (SCAQMD) Air Quality Sensor Performance Evaluation Center (AQ-SPEC) has created both lab and field evaluation reports comparing several PurpleAir monitors against Federal Equivalent Methods (FEM). They observed good correlations with FEMs during lab ($R^2 \approx 0.99$) and field ($R^2 \approx 0.95$) evaluations of PM$_1$ and PM$_{2.5}$ (AQ-SPEC South Coast AQMD, 2017b; AQ-SPEC South Coast AQMD, 2017a). AQ-SPEC also observed moderate to good accuracy within the 0-250 $\mu g m^{-3}$ range from the PurpleAir monitors across multiple temperature and relative humidity profiles. The PurpleAir monitors outperformed competing sensors that cost 2-4 times more (MetOne E-Sampler, Alphasense OPC-N2, Dylos). The low-cost and good accuracy of the PurpleAir monitors allowed us to installed a higher density network in each of the five cities. The PurpleAir monitors are also WiFi capable,
and transmit data to the ThingSpeak IoT cloud platform every minute. This enabled the real-time monitoring of the network via a Python script which notified us when a specific sensor lost WiFi connection or power, and data collection was interrupted (Appendix A: purpleair_watchdog.py). We could then notify the appropriate volunteer monitor hosts to troubleshoot the device and establish a connection, improving long-term data recovery. The PurpleAir-II SD model also includes local storage in the form of a micro-SD card, providing a backup if the device loses connection to its WiFi network, or was not available at a specific location altogether. While this appeared to be a promising feature, the current firmware version (2.50i) of the PurpleAir-II SD had issues with data logging, and the microcontroller would intermittently lose connection with the SD card module. This produced data gaps and error messages were logged within the main data file, creating further data processing issues during analysis. Upon this discovery, we modified our sensor hosting requirements to require a WiFi connection to avoid local storage issues.

Each PurpleAir monitor contains two Plantower PMS5003 sensors. These are low-cost sensors ($20-$40) that approximate PM mass using an optical scattering technique. The sensors record PM mass density concentrations for 1.0 µm, 2.5 µm, 5.0 µm, and 10 µm size categories using a proprietary algorithm and come pre-calibrated from factory. The PMS5003 has additional fields that report number density concentrations of particles ranging in size from 0.3 to 10.0 µm. In addition to the two Plantower PMS5003 sensors, the PurpleAir monitors contain a Arduino ESP8266 microcontroller for wireless data communication and logging, a BME280 pressure, temperature, and humidity sensor, an SD card
reader, and a real time clock to keep time while disconnected from the main power supply. The devices also have an IP68 waterproof rating, and are designed for long-term outdoor use.

2.1.2 PurpleAir Device Ownership and Open Data Philosophy

PurpleAir upholds an open data philosophy that is not matched by competing low-cost air quality sensor manufacturers. Many sensor companies consider data as a service, renting monitors and providing data access via monthly subscription (AirAdvice, Apis Inc, MetOne, Alphasense OPC-N2, etc.). While this may benefit the use-cases of some consumers (device troubleshooting/maintenance is often included within this service), this was disadvantageous for our research purposes. In contrast, any data collected from PurpleAir monitors is freely available to all users, independent of device ownership. Data from third party monitors are accessible via their application program interface (API). After field calibration, this allowed us to incorporate data from additional third party PurpleAir monitors into our existing network.

2.2 Laboratory Evaluations

We conducted laboratory evaluations of 65 PurpleAir monitors within a 25 °C temperature controlled chamber at ambient humidity, and measured their response to multiple PM sources. The low-cost PurpleAir monitors were compared against a TSI DustTrak DRX 8433 Aerosol Monitor (also an optical scattering instrument). PurpleAir monitors were exposed to PM generated from smoke generated from lit matches, and smoke created by snuffing a candle
within the test chamber. Only one specific PM source was evaluated during a given test. Approximately 8-10 PurpleAir monitors were suspended from the lid of the 160L stainless steel test chamber using baling wire. The DustTrak reference monitor was placed on the floor of the chamber to ensure adequate ventilation between itself and the sensors hanging above. During the candle and match smoke tests, a 100 mL beaker was placed on the floor of the chamber, opposite to the sampling inlet of the DustTrak monitor. A match or candle was lit and snuffed inside of the beaker and the chamber lid was immediately closed. The chamber would rapidly saturate with PM well above the maximum effective range reported by the low-cost sensor (500 µgm$^{-3}$) manufacturer. The PM would then slowly decay to zero as it was ventilated with particle-free air (zero-air pushed through a 0.3 µm particle filter) at a constant flow-rate (inlet pressure of 30 psi). During this time, the PM levels within the chamber were monitored using the analog port on the DustTrak, an analog to digital convert (A/D converter), and a Python script which provided a real-time 1-second estimate of the PM$_{2.5}$ mass density. The entire process, from PM saturation to below-ambient concentration, required approximately 1-2 hours per test. The slow and steady decay of the PM within the chamber allowed us to compare each PurpleAir monitor to the DustTrak reference across a wide range of PM concentrations, throughout the entirety of the low-cost sensors’ effective range (0 - 500 µgm$^{-3}$). However, since we were primarily interested in sensor performance across a environmentally relevant range, we developed individual linear models for each sensor by comparing 1-minute averages against the reference
instrument between 0 - 150 $\mu g m^{-3}$. This method allowed us to capture a near-continuous concentration range instead of just a few discrete points that a typical span calibration would provide. Comparing the low-cost sensors across a continuous PM range from multiple PM sources helped characterize the linearity of the low-cost sensors against our reference monitor.

2.3 PurpleAir Data Acquisition and Storage

PurpleAir provides a lightweight JSON API (http://www.purpleair.com/json) for requesting the most recent observation from each monitor, but accessing historical data is not as straightforward. While historical data is stored by PurpleAir and made available via a wagtail which requires a manual download (http://www.purpleair.com/sensorlist), they do not provide a programmatic interface for downloading historical data from their platform. Instead, historical data is accessible through the ThingSpeak cloud platform. Programmatically downloading historical data is still free and open, but it is significantly more challenging than accessing the real-time data from the PurpleAir JSON API.

This added difficulty can be explained by two main factors. The primary factor is the API rate limit set by ThingSpeak. The size of each request is limited to a maximum of 8000 observations. This equates to about one week’s worth of data from a single node per request. Therefore, accessing historical data from multiple devices over a long period of time (especially in the case of initial database population) required a dynamic method of scheduling a sequence of API requests based on a given start and end date. To overcome this obstacle, we developed a suite of R/Python/Bash/SQL software which dynamically identified
new sensors as they are installed within our study areas, and included them into our data acquisition queue. Our system then iteratively made requests no larger than 8000 observations across the entire lifetime of each monitor within the network. The results from each request were then processed and stored in our PostgreSQL database with the PostGIS extension. This preserved the spatial coordinates associated with each PurpleAir monitor, and allowed us to perform spatial SQL queries throughout analysis. Once our initial database was populated, a similar script was executed from a Linux server (Xubuntu 18.04) each day at 1:00 AM using the ‘cron’ task-scheduling software. This script would upload the previous day’s data into the database, including any new sensors that were installed throughout the network. It is worth noting that there are no API rate limits in terms of request frequency (requests per unit time), and we found success through multi-threading our pull requests. So long as each request was no larger than 8000 observations, we could pull ThingSpeak data from every available processor core simultaneously. This reduced our initial database population time by 1/8\textsuperscript{th} - 1/12\textsuperscript{th} depending on the available hardware (8 cores versus 12 cores).

In addition to the API request size limit, another challenge we encountered was the field encoding and total field limit of the PurpleAir data when stored on the ThingSpeak platform. Relations within the ThingSpeak API are limited to a total of eight fields, and their names are encoded (e.g. ‘field1’ instead of ‘PM1.0 (CF=ATM) ug/m3’). However, the PurpleAir monitors record 32 fields. Therefore, requesting all of the available data from a given PurpleAir monitor required four separate API requests, achieved through a combination of
Chapter 2. Methodology

ThingSpeak primary and secondary API keys included within the PurpleAir JSON API. These four subsets were combined through SQL-like table joins using an R script. A key provided by PurpleAir’s technical support team was required before these data could be fully processed (Appendix B). Once these fields were decoded, our suite of R/Python/Bash/SQL software automated the acquisition, processing, and storage of historical PurpleAir data (Appendix A: purpleair_id_key.py; create_observation.sql; thingspeakCollect.R; uploadDailyThingspeak.R; uploadHistoricalThingspeak.R).

2.4 Opportunistic Ambient Calibration of PurpleAir Networks

Since the response of each PMS5003 sensor is dependent on the PM source (in addition to the intra-model variation observed), an opportunistic ambient calibration method was developed with the goal of applying regionally-specific correction factors to each individual PMS5003 sensor within the PurpleAir network (Portland, OR). Given that PM$_{2.5}$ is not just a single pollutant, but a complex mixture of particles from multiple primary and secondary sources, an ambient field calibration method was required to best reflect the local PM source profile throughout each city. To determine locally relevant correction factors for each PMS5003 sensor in the Portland network, we compiled 5-minute nephelometer-derived PM$_{2.5}$ data provided by the Oregon Department of Environmental Quality (ODEQ) throughout the duration of our study period (September 2017 - March 2019). Nephelometers, being an optical scattering instrument themselves, initially provide only a Bscat value (the extinction of light due to back
scattering), and are subject to the same response issues as any other optical-based PM measurement (e.g. PurpleAir). However, a linear model was developed by ODEQ to correct these Bscat values. Nephelometers and 24 hour PM$_{2.5}$ gravimetric samplers (FRM) were co-located to develop these correction factors. We used the 5-minute PM$_{2.5}$ data provided by the ODEQ’s FRM-corrected nephelometers to compare with our low-cost sensor network as a means of quality scaffolding. Depending on the day, there were 4-6 active nephelometers throughout the entire study period. We used these nephelometers to identify periods of time where PM$_{2.5}$ levels were regionally homogeneous. This consisted of 15-minute periods (a total of three 5-minute observations per nephelometer) where the relative standard deviation (RSD) of the ODEQ reference monitors was less than or equal to 10%. Once windows of regionally homogeneous PM were identified, we compared the atmospheric PM$_{2.5}$ channels (‘pm2_5_atm’ as opposed to ‘pm2_5_cf_1’ which is recommended for indoor use) from both PMS5003 sensors within a given PurpleAir monitor against the ambient background concentration (the mean of the active nephelometers) using an Ordinary Least Squares regression (OLS) to determine sensor-specific correction factors. These correction factors were stored as a new relation in our Postgres database to be queried during the modeling phase. Sensors with an R$^2$ value below 0.90 were deemed faulty or unreliable, and were excluded from the modeling phase. This method incorporated periods of time throughout the entire study duration, allowing us to capture PM across a range of seasonal sources (wildfires, wood heating, traffic, temperature inversions, etc.) and develop comprehensive correction factors that account for all regionally relevant sources.
2.5 Data Recovery and Sensor Lifetime

Between July 2017 and December 2018 data quality issues from 7% of the PurpleAir network were observed. While this was an improvement over the Shinyei sensors (AirAdvice monitors) we have used during previous studies, the PMS5003 within the PurpleAir are not immune to optical contamination and laser diode degradation. PurpleAir’s technical support team recommended cleaning the optics with compressed air and/or applying a vacuum to the sample inlets. Compressed air revived one out of three faulty sensors. Data recovery over this period was greater than 90% and the remaining network has shown no signs of further degradation. Plantower now provides a dual-laser model (PMS6003) which has effectively doubled the lifetime of the sensor by alternating diodes between measurements. Modifications to the sensor chassis and sampling path design have also demonstrated improvements to mitigating dust accumulation in the optics in the previous models (PMS1003, PMS3003).

2.6 Sensor Allocation

The Portland, Oregon sensor network was the first in the Canopy Continuum study, and PurpleAir monitors were placed opportunistically. A sensor allocation and volunteer recruitment method was developed for the remaining four cities (Albuquerque, New Mexico, Boise, Idaho, Tacoma, Washington, and Sacramento, California). These cities were previously selected within the Canopy Continuum project because they represent a varying degree of canopy
cover necessary to characterize the potential role of vegetation in mitigating environmental stressors. Our sensor allocation approach first characterized the spatial autocorrelation of the existing Portland PurpleAir network to identify the minimum distance necessary to capture spatially heterogeneous PM$_{2.5}$. In other words, we plotted the $R^2$ between every sensor pair as a function of distance and observed when the $R^2$ began to drop below 0.75. We identified a minimum placement distance of 2 km from these results (Figure 2.1). Sensor allocation in the remaining cities relied on a random sampling approach, stratifying by population density. For each city, a population density surface with a spatial resolution of 2 km$^2$ was derived via areal interpolation of American Community Survey (ACS) data at the census block group level. Census data was retrieved using the tidycensus R package (Walker, 2019). In short, a 2 km$^2$ grid was created using the spatial extent defined by the bounding box of a given city’s Metropolitan/Micropolitan Statistical Area shapefile (US Census Bureau, 2015). A spatial intersection was applied to each grid cell and its corresponding census block group(s) to determine area-based weights. For example, if a grid cell encompassed 50% of a given block group, the weight for that block group would be 0.5. If a grid cell was completely within a single block group, the weight would be 1.0. These weights were then applied to each block group’s population count, assuming that the population density within a given block group was evenly distributed. The sum of these weighted population counts, determined by the relative area of each block group within that cell, was divided by the total area of the cell (2 km$^2$) to calculate its population density. We then iterated through each grid cell/block group pair to create population density
surfaces throughout each city.

*Figure 2.1: Scatter plot of the Portland PurpleAir network spatial autocorrelation as a function of distance.*

These gridded population density layers consisted of approximately 200-500 individual 2 km\(^2\) grid cells depending on the study area. With only a limited number of PurpleAir sensors available to deploy in each city (Albuquerque \(n = 30\); Tacoma \(n = 25\); Boise and Sacramento \(n = 20\)), a random sampling approach was employed to prioritize areas of interest during network installation. These randomly assigned target areas were stratified by population density deciles.
Given that the primary goal of this study was to generate PM exposure maps for future analysis in an epidemiological study, the majority of the target areas were randomly assigned to the most densely populated deciles. In other words, we attempted to recruit more volunteers to host PurpleAir monitors who lived in more densely populated neighborhoods, and fewer volunteers from suburban and rural neighborhoods. The total number of monitors deployed in a given city was determined by the size of its Metropolitan Statistical Area (MSA), as well as, the number of pre-existing 3rd party PurpleAir monitors. After the random stratified selection process, target areas were manually adjusted for adjacency to other target areas, as well as, any existing 3rd party PurpleAir monitors.

While this method provided a quantifiable approach to sensor allocation, the structure of each city’s network was ultimately limited by volunteer availability. We adopted the ‘some data is better than no data’ philosophy, and placed the majority of our sensors wherever there were willing and able participants. Sensors were typically installed 6-8 ft above ground in the front or backyard of a volunteer residence. Occasionally, sensors were sited outside commercial or government offices. This was especially the case in target areas with relatively low population densities. Sensors were placed at least 30 ft away from hyper-local sources including barbeques, laundry vents, and fireplaces.

Web maps with the gridded population density and sensor target areas (outlined in bold) were developed using leaflet R package (Cheng, Karambelkar, and Xie, 2018). Static maps are included below and hyperlinks to the original web maps are included within the figure captions (Figure 2.2; Figure 2.3; Figure 2.4; Figure 2.5).
FIGURE 2.2: Sensor allocation map developed for Albuquerque, New Mexico. Areas prioritized for sensor allocation are highlighted in bold. The web map can be viewed here: http://web.pdx.edu/~porlando/albuquerque.html
Figure 2.3: Sensor allocation map developed for Boise, Idaho. Areas prioritized for sensor allocation are highlighted in bold. The web map can be viewed here: http://web.pdx.edu/~porlando/boise.html
Figure 2.4: Sensor allocation map developed for Sacramento, California. Areas prioritized for sensor allocation are highlighted in bold. The web map can be viewed here: http://web.pdx.edu/~porlando/sacramento.html
Figure 2.5: Sensor allocation map developed for Tacoma, Washington. Areas prioritized for sensor allocation are highlighted in bold. The web map can be viewed here: http://web.pdx.edu/~porlando/tacoma.html
2.7 Data Compilation

All data compilation and spatial analysis were performed using the R Statistical Programming Language (R Core Team, 2019), and its corresponding spatial libraries. These include ‘sp’ (Pebesma and Bivand, 2005), ‘sf’ (Pebesma, 2018), ‘raster’ (Hijmans, 2019), ‘rgdal’ (Bivand, Keitt, and Rowlingson, 2019), and ‘rgeos’ (Bivand and Rundel, 2019). The following workflow diagram highlights the various data acquisition and processing scripts necessary to develop LUR models with corrected PurpleAir network data (Figure 2.6).
Chapter 2. Methodology
Figure 2.6: A workflow diagram highlighting the various data compilation scripts necessary to develop a LUR model with corrected PurpleAir data. All relevant source code is included within Appendix A.

2.7.1 Spatial and Temporal Predictors of PM$_{2.5}$

Several land-use (LU) and meteorological variables were evaluated to predict daily PM$_{2.5}$ surfaces at a spatial resolution of 50 m. Land-use variables were included as surrogates for potential PM sources in attempt to characterize the spatial heterogeneity of PM within the study area. This included proxies of traffic-related PM such as road (freeways, major arterials, streets, etc.) and rail density. Other spatial predictors were included to account for potential area sources (population density, LULC), or sinks (NDVI, EVI). Meteorological variables were included to characterize the temporal variation of PM. For example, PM concentrations typically increase as the planetary boundary layer (PBL) decreases as the effective mixing volume for PM dilution decreases during low boundary height scenarios. In addition, wind speed has been shown to affect PM concentrations. As wind speed increases, PM generated from urban sources
is readily removed, diluting overall PM levels throughout the urban core (Chudnovsky et al., 2014). The network of ODEQ PM monitors (nephelometers) were also included as a temporal scaling factor of PM. This allowed us to capture temporal patterns in PM that would otherwise be unaccounted for by meteorological variables alone (e.g. wildfires).

Road and Rail Density

Road and rail data were obtained from Metro’s Regional Land Information System (RLIS) data resource center (Oregon Metro, 2018). This included GIS shapefiles of the ‘Streets’, ‘Arterials’ ‘Major Arterials’, ‘Freeways’, ‘Railroads’, and ‘Railyards’ feature classes. Originally vector data, each feature class was discretized into a new raster layer with a spatial resolution of 50 m. These data were also reprojected to match the Coordinate Reference System (CRS) of the study area, and clipped by its extent (Appendix A: processRoads.R). For Portland, the bounding box was defined by Metro’s urban growth boundary extent (UGB). RLIS data is not available for the other study areas, and a national-scale data sources should be used instead during future modeling endeavors.

NDVI and EVI

Satellite-derived Normalized Difference Vegetation Index (NDVI) and Enhanced Vegetation Index (EVI) data were obtained from the 250 m MODIS 16-day product (MOD13Q1) (Didan, 2015b). These data were programmatically downloaded using the MODIS R package (Mattiuzzi and Detsch, 2019) (Appendix A: downloadNDVI.R). The NDVI was determined using red and near
infrared surface reflectances, which has been shown to provide a temporal indication of vegetation cover and its phenological state (Tucker, 1979; Tucker and Sellers, 1986). The EVI layer is more sensitive over densely vegetated areas, explaining further variation in vegetation density beyond the range of the NDVI. Both of these layers are derived from bi-directional surface reflectances, with masking algorithms for water, clouds (and their shadows), and aerosols (Didan, 2015b). Each MODIS layer was reprojected to match the CRS of the study area, clipped by the study extent, and resampled via bilinear interpolation to match the 50 m spatial resolution of the model. Originally, we explored a temporal cubic spline interpolation method to derive daily NDVI/EVI observations from each 16-day MODIS overpass (Fritsch and Carlson, 1980). While this technique produced promising results with the 1 km resolution MODIS product (MOD13A2), (Didan, 2015a), it did not scale to the 250 m resolution product, and our processing script unexpectedly terminated without explanation (Appendix A: `interpolateNDVI.R`). This was likely due to issues with cloud masking and excessive missing values at this resolution. Instead of interpolating daily NDVI/EVI from the 250 m product, we opted to develop annual aggregates (Appendix A: `annualNDVI.R`).

Elevation data were retrieved from NASA’s Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) digital elevation map (DEM) product. This is a global DEM with a spatial resolution of 30 m provided in GeoTiff format (NASA, 2009). This layer was reprojected to match the CRS of our study area, clipped by our study extent, and resampled from 30 m to 50 m (Appendix A: `processElevation.R`).
Population Density

Population density data was obtained from NASA’s Gridded Population of the World (GPW) collection (version 4) (Doxsey-Whitfield et al., 2015). This product models the distribution of the global human population on a continuous raster layer for use in social, economic, and Earth science disciplines. This is a 30-arc second resolution product (approximately 1 km) provided in GeoTIFF format. This global raster layer was reprojected to match the CRS of our study area, and then clipped by its extent. It was also resampled from 1 km to 50 m via bilinear interpolation (Appendix A: `processPopDensity.R`).

GridMET Meteorological Variables

Meteorological data was obtained from the University of Idaho’s Gridded Surface Meteorological Data (GridMET). The GridMET data is a combination of high-resolution spatial data (4 km) from the Parameter-elevation Regressions on Independent Slopes Model (PRISM) and the high-temporal resolution data from the National Land Data Assimilation System (NLDAS) which generates spatially and temporally continuous fields of several meteorological variables (Abatzoglou, 2013; Daly, 2006; Mesinger et al., 2006). The GridMET product provides estimates of daily precipitation totals, minimum and maximum air temperature and relative humidity, specific humidity, mean wind direction and mean wind speed. However, the GridMET product was not designed to capture microclimates that occur at more granular resolutions than its native resolution of the two parent data sources (Abatzoglou, 2013). The wind variables are limited to a 32 km spatial resolution and are not capable of capturing any influences
Chapter 2. Methodology

of terrain on wind fields (Abatzoglou, 2013). An R script was developed to programmatically download daily GridMET data via wget calls to a HTTP server (Appendix A: downloadGridMET.R). As with the previous predictors, the Grid-MET data was reprojected to match the CRS of our study area, clipped by our study extent, and resampled to 50 m to match the resolution of our modeled surfaces. A bilinear interpolation was applied to all scalar variables, however, wind direction was resampled using the nearest neighbor method because the bilinear method is incapable of vector averaging (Appendix A: processGridMET.R).

Land Use and Land Cover

Land Use and Land Cover (LULC) data was obtained from the USGS’s National Land Cover Database (NLCD) 2011 product. The NLCD provides a 30 m spatial resolution categorical raster layer with fifteen different LULC attributes. This includes open water, low, medium, and high intensity developed land, deciduous, evergreen, and mixed forest lands (Homer et al., 2015). This raster was reprojected to match the CRS of our study area, clipped by its extent, and resampled from 30 m to 50 m using the nearest neighbor method. This single categorical layer was decomposed into individual presence/absence rasters for each LULC attribute via one-hot encoding. For example, if the ‘open water’ LULC was observed within a raster cell, a value of 1 would be assigned to this cell. If a LULC was not observed within a cell, then a value of 0 would be assigned. One-hot encoding the NLCD raster into individual layers was necessary for creating focal statistics (Appendix A: processNLCD.R).
Planetary Boundary Layer Height

Planetary boundary layer height (PBL) data was obtained from NOAA’s NARR product (Mesinger et al., 2006). An R script was used to programmatically download 3-hourly PBL netCDF data throughout the study duration (September 2017 - March 2019) via wget calls to NOAA’s FTP server (Appendix A: downloadPBL.R). The NARR PBL grid resolution is approximately 0.3 degrees, which corresponds to 32 km at the lowest latitudes. Each 3-hourly grid was aggregated into a single mean value throughout the study area. These 3-hour means were then used to determine the daily maximum and minimum PBL height values. The minimum and maximum PBL values were used as temporal scaling factors, and were extrapolated into continuous 50 m resolution spatial grids corresponding to the extent and CRS of our study area (Appendix A: processPBL.R).

Nephelometer PM$_{2.5}$

Nephelometer-derived PM$_{2.5}$ data with a 5-minute temporal resolution were provided by ODEQ. In addition to the use of these data for determining correction factors for our PurpleAir sensor network, these data were also included as a temporal scaling factor to account for changes in the ambient background concentration due to regional sources (e.g. wildfires). The 5-minute PM$_{2.5}$ values from 4-6 active nephelometers were aggregated into daily mean and median values (Appendix A: processNeph.R). Similar to the PBL compilation, these values were extrapolated into a continuous 50 m resolution surface corresponding to the extent and CRS of our study area (Appendix A: interpolateNeph.R).
Latitude and Longitude

Continuous fields for both latitude and longitude were generated using R and its spatial libraries (raster, rgdal, rgeos, sp, sf, etc.). These variables were included to account for the potential effects of spatial autocorrelation described by Tobler’s 1st law of geography (Tobler, 1970).

2.7.2 Focal Statistics

Focal statistics were determined for each spatial (and temporal-spatial) predictor at 50 m, 100 m, 300 m, 500 m, and 1000 m buffer sizes (Appendix A: buffer.R). These buffer distances were adopted from the LUR methods developed by the ESCAPE Project (Eeftens et al., 2012). The population density variable also included 1250 m, 1500 m, 2000 m, and 2500 m focal buffer distances. Briefly, this method iterates through each cell within a given 50 m resolution spatial grid and determines the mean value of all adjacent cells within a prescribed radius (neighborhood). The target cell is then assigned the mean value of this neighborhood. This process is then repeated across the sequence of desired buffer distances. For a graphical representation of the new layers created from applying focal statistics to the ‘Streets’ layer, see Figure 2.7. For the predictors which are purely spatial (NLCD, Roads and Rail, Elevation) focal statistics at each neighborhood size were performed only once. However, for the temporal spatial-predictors (NDVI, EVI, GridMET), this operation was repeated for each daily observation. This resulted in a total of 33 thousand individual raster layers, stored as GeoTIFFs, which consumed approximately 100 GB of local disk.
storage. Despite being rasterized, this method was also computationally expensive. Determining thousands of focal buffers required the use of a high-memory compute server provided by PSU Research Computing, and consumed \( \geq 500 \) GB of memory while running on all 24 available processor cores. Processing time was on the order of 4-6 hours depending on the total number of predictors selected for modeling.

The temporal predictors lacking any spatial component (e.g. scaling factors including PBL and Nephelometer PM) were not included in the focal statistics pipeline, as these layers only consisted of a single mean value for each day.

Focal statistics for the GridMET wind direction data were handled differently than the other variables. This layer, with the inclusion of wind speed, was converted from degrees to radians, and then transposed into its respective \( u \) (zonal velocity) and \( v \) (meridional velocity) components. Focal statistics were then performed on the \( u \) and \( v \) components (unit-vector averaging), and converting back into degrees using the arc-tangent (Appendix A: buffer.R).
2.7.3 Data Extraction

After generating focal statistics across our sequence of buffer distances, variables, and days, we extracted the grid cells values that spatially intersected with the corrected PurpleAir response variable (Appendix A: `extract.R`). Iterating the raster extraction process across each day within our study duration, the end result was a single large matrix (data frame) with the corrected PurpleAir PM$_{2.5}$ response variable and all relevant predictors at each buffer size. This matrix
consisted of 211 variables and 11,546 observations. Additional categorical variables were added to this matrix including ‘Weekday/Weekend’, ‘Day of Week’, ‘Season’, and ‘Month’. This matrix was used during the modeling phase. The study duration consisted of 575 consecutive days and, given the 11,546 unique observations, the average number of active PurpleAir monitors on a given day was approximately 20. This is comparable to the network sizes within the ESCAPE Project (Eeftens et al., 2012). The Portland network did grow overtime as PurpleAir monitors were incrementally added by our research group as well as other 3rd parties. The network size was closer to 10 active nodes during the first few months of the study, and grew to more than 35 nodes by its end. Some of these were active for a short period of time, but most remained active once initially established.

2.8 Predictor Selection

To avoid issues of multicollinearity, only a single focal buffer distance was selected for each predictor variable during the modeling phase. A correlation matrix was produced for each variable against the response data and the focal buffer with the strongest correlation with the response variable was then selected for modeling. If multiple focal buffers had the same correlation coefficient, then the finest spatial resolution buffer was chosen to capture as much spatial heterogeneity as possible. Variables with nonintuitive correlations, e.g. expected sources yielding a negative correlation, or expected sinks yielding a positive correlation, were excluded from the model.
An additional correlation matrix which compared all of the previously selected predictors was generated to explore groups which covaried. A complete linkage hierarchical clustering algorithm, a built-in parameter within the R ‘corrplot’ library (Wei and Simko, 2017), was used to identify clustered predictors (Defays, 1977). Once these groups were identified, only a single predictor within each group was selected for the modeling phase. For example, if the NLCD-derived ‘Deciduous Forest’, ‘Mixed Forest’, ‘Evergreen Forest’ were clustered with ‘NDVI’, then the most generalizeable predictor was selected for the final model (e.g. the NDVI layer can be used to explain all of the other vegetation layers).

These predictor selection criteria were established to meet the multicollinearity assumption of a traditional multiple linear regression. We relied on a Variance Inflation Factor (VIF) below 5 to meet this assumption. VIF scores were determined using the ‘car’ R library to assess multicollinearity (Fox et al., 2007). Although the multicollinearity assumption only applies to multiple linear regression model, the same variable selection procedure was applied to the Random Forest model for direct comparison. Model variables were also tested for significance, and only predictor coefficient’s p-value less than 0.1 were selected for the final model (Appendix A: modeling.R).

The initial predictors selected for the modeling phase are included within Table 2.1. This table details the variable description, abbreviation, and respective data sources. The expected source and sink proxies for each predictor are also included.
2.9 Modeling and Validation

For this study, we compared the performance of a traditional LUR model (multiple linear regression) against a decision-tree-based machine learning model (Random Forest) (Liaw and Wiener, 2002; Breiman, 2001). Of the 11,546 total observations, we randomly subset 70% during the training phase for each model. The remaining 30%, or ‘test’ data, was reserved for model validation. In addition to this 70/30 holdout approach, we also employed a leave-one-out cross-validation (LOOCV) method which has been widely used in previous LUR studies (Yang et al., 2017; Eeftens et al., 2012; Saucy et al., 2018). With this approach, a single PurpleAir node was excluded from the training data and a model was developed using the $n - 1$ remaining sites. The subsequent model was then used to predict all of the observations for this single node, and the model performance was evaluated by observing the cross validation (CV) $R^2$ and CV RMSE. This process was then repeated for each individual node, excluding nodes with fewer than 100 observations. The LOOCV method allowed us to evaluate how well a given model predicted for a specific sensor over a consecutive period of time, instead of predicting a random subset of days and sensors simultaneously. In addition, this method allowed us to compare an entire ensemble of predictions (a new model was developed for each node that was left out), and identify any locations that a model was consistently under/over predicting.

An ensemble method was used during the LUR model prediction phase. Specifically, each of the $n - 1$ models developed during the LOOCV phase were
used to predict an individual PM$_{2.5}$ surface. This ensemble of predictions was then aggregated to determine a single mean PM$_{2.5}$ surface each day. The Random Forest method was too computationally expensive to rely on the LOOCV technique when making predictions. Instead, the results from the single model which included all available monitoring sites generated from the 70/30 holdout process was used during the prediction phase.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Data Source</th>
<th>Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm2.5.median</td>
<td>Nephelometer PM$_{2.5}$ (median)</td>
<td>ODEQ 2018</td>
<td>Regional temporal variation</td>
</tr>
<tr>
<td>fwy</td>
<td>Freeways (length)</td>
<td>RLIS 2019</td>
<td>Traffic emissions</td>
</tr>
<tr>
<td>maj_art</td>
<td>Major Arterials (length)</td>
<td>RLIS 2019</td>
<td>Traffic emissions</td>
</tr>
<tr>
<td>arterial</td>
<td>Arterials (length)</td>
<td>RLIS 2019</td>
<td>Traffic emissions</td>
</tr>
<tr>
<td>streets</td>
<td>Streets (length)</td>
<td>RLIS 2019</td>
<td>Traffic emissions</td>
</tr>
<tr>
<td>railroad</td>
<td>Railroad (length)</td>
<td>RLIS 2019</td>
<td>Railroad emissions</td>
</tr>
<tr>
<td>railyards</td>
<td>Railyards (area)</td>
<td>RLIS 2019</td>
<td>Railyard emissions</td>
</tr>
<tr>
<td>elev</td>
<td>Elevation (m)</td>
<td>ASTER DEM 2009</td>
<td>Potential sink (windflow)</td>
</tr>
<tr>
<td>pop_dens</td>
<td>Population (density)</td>
<td>NASA 2015</td>
<td>Area sources</td>
</tr>
<tr>
<td>PBL</td>
<td>Planetary Boundary Layer Height</td>
<td>NOAA 2019</td>
<td>Sink (dilution)</td>
</tr>
<tr>
<td>pr</td>
<td>Precipitation (daily total)</td>
<td>GridMET 2019</td>
<td>Potential sink (wet deposition)</td>
</tr>
<tr>
<td>th</td>
<td>Wind Direction (degrees)</td>
<td>GridMET 2019</td>
<td>Potential sink (dilution)</td>
</tr>
<tr>
<td>vs</td>
<td>Wind Speed (m/s)</td>
<td>GridMET 2019</td>
<td>Potential sink (dilution)</td>
</tr>
<tr>
<td>tmmx</td>
<td>Maximum Temperature</td>
<td>GridMET 2019</td>
<td>Potential SOA source</td>
</tr>
<tr>
<td>tmmn</td>
<td>Minimum Temperature</td>
<td>GridMET 2019</td>
<td>Potential SOA source</td>
</tr>
<tr>
<td>rmax</td>
<td>Maximum Relative Humidity</td>
<td>GridMET 2019</td>
<td>Potential source (hygroscopic growth)</td>
</tr>
<tr>
<td>rmin</td>
<td>Minimum Relative Humidity</td>
<td>GridMET 2019</td>
<td>Potential source (hygroscopic growth)</td>
</tr>
<tr>
<td>ndvi</td>
<td>NDVI</td>
<td>MODIS 2019</td>
<td>Potential sink (deposition)</td>
</tr>
<tr>
<td>evi</td>
<td>EVI</td>
<td>MODIS 2019</td>
<td>Potential sink (deposition)</td>
</tr>
<tr>
<td>low_dev</td>
<td>Developed Low Intensity LULC (density)</td>
<td>NLCD 2011</td>
<td>Area sources</td>
</tr>
<tr>
<td>med_dev</td>
<td>Developed Medium Intensity LULC (density)</td>
<td>NLCD 2011</td>
<td>Area sources</td>
</tr>
<tr>
<td>high_dev</td>
<td>Developed High Intensity LULC (density)</td>
<td>NLCD 2011</td>
<td>Area sources</td>
</tr>
<tr>
<td>deciduous_forest</td>
<td>Deciduous Forest Cover (density)</td>
<td>NLCD 2011</td>
<td>Potential sink (deposition)</td>
</tr>
<tr>
<td>evergreen_forest</td>
<td>Evergreen Forest Cover (density)</td>
<td>NLCD 2011</td>
<td>Potential sink (deposition)</td>
</tr>
<tr>
<td>mixed_forest</td>
<td>Mixed Forest Cover (density)</td>
<td>NLCD 2011</td>
<td>Potential sink (deposition)</td>
</tr>
<tr>
<td>lat</td>
<td>Latitude</td>
<td>PurpleAir</td>
<td>Spatial variability of sinks and sources</td>
</tr>
<tr>
<td>lon</td>
<td>Longitude</td>
<td>PurpleAir</td>
<td>Spatial variability of sinks and sources</td>
</tr>
<tr>
<td>season</td>
<td>Season (1,2,3,4)</td>
<td>PurpleAir</td>
<td>Temporal variability of sinks and sources</td>
</tr>
<tr>
<td>weekend</td>
<td>Weekday/Weekend (0,1)</td>
<td>PurpleAir</td>
<td>Temporal variability of sinks and sources</td>
</tr>
</tbody>
</table>

**Table 2.1:** Descriptions of the predictor variables selected during LUR model development with their respective data sources and PM$_{2.5}$ proxies.
Chapter 3

Results and Discussion

3.1 Laboratory Evaluations

Strong linear relationships between the low-cost PurpleAir monitors and the reference DustTrak DRX Aerosol Monitor Model 8533 were observed for both the match and candle smoke tests. The mean $R^2$ of the candle smoke evaluations was $0.958 \pm 0.14$, whereas the mean $R^2$ of the match smoke evaluations was $0.942 \pm 0.17$ Table 3.1. Despite following the same protocol between tests, some sensors did not perform linearly with the reference instrument (Figure 3.1). However, these cases occurred infrequently and meaningful calibrations with $R^2$ values greater than 0.90 occurred for more than 80% of all test periods. While the strength of DustTrak-PurpleAir relationship was independent of the PM source, the slopes differed depending on the PM source. The mean slope of the models generated from the match smoke tests was $1.35 \pm 0.64$. This was approximately half the value observed from the candle smoke tests which yielded a mean slope of $2.66 \pm 0.75$ (Figure 3.2). These results indicate two challenges with low-cost PM sensors for LUR. The first was that intra-model
variation does occur with the Plantower sensors, despite the manufacturer’s ef-
fort to pre-calibrate each sensor. This means that two sensors, when exposed
to the same PM sample, will produce a slightly different response. The slopes
from the match smoke tests are more narrowly distributed, while the candle
smoke tests have a wider distribution ranging from 2-3 and appear bimodal.
This was somewhat unexpected because multiple different types of matches
were used throughout these laboratory evaluations, whereas only a single can-
dle was used. This could potentially be explained by match smoke PM having
similar physical properties as Plantower’s PM source during factory calibra-
tion (e.g. absorption, profile, shape, etc.). The second challenge presented by
these results is that the Plantower/PurpleAir response is dependent on the PM
source. This suggests that laboratory calibrations alone, while a reliable screen-
ing tool for faulty sensors, are not enough to determine meaningful correction
factors for field-deployed sensors necessary to produce epidemiologically rel-
evant PM$_{2.5}$ concentrations. These results support the development of an am-
bient calibration method of PurpleAir networks for determining field-specific
correction factors. Two scatter plots were included to highlight the PurpleAir’s
response across the entire PM range of a typical test for both the primary and
secondary sensors (A and B sensors) (Figure 3.3; Figure 3.4).
Table 3.1: Linear model summary statistics of both laboratory and field evaluations. Lab evaluations are comparing the DustTrak against PurpleAir monitors grouping by the PM source (candle and match smoke). The field evaluations are comparing the mean nephelometer PM$_{2.5}$ against PurpleAir monitors. SD = standard deviation.

<table>
<thead>
<tr>
<th>PM Source</th>
<th>$R^2$</th>
<th>$R^2$ SD</th>
<th>Slope</th>
<th>Slope SD</th>
<th>Intercept</th>
<th>Intercept SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candle</td>
<td>0.958</td>
<td>± 0.14</td>
<td>2.7</td>
<td>± 0.78</td>
<td>0.34</td>
<td>± 2.83</td>
</tr>
<tr>
<td>Match</td>
<td>0.942</td>
<td>± 0.17</td>
<td>1.35</td>
<td>± 0.64</td>
<td>3.98</td>
<td>± 5.24</td>
</tr>
<tr>
<td>Field</td>
<td>0.958</td>
<td>± 0.02</td>
<td>0.46</td>
<td>± 0.06</td>
<td>1.63</td>
<td>± 0.31</td>
</tr>
</tbody>
</table>

Figure 3.1: Distribution of $R^2$ values from linear models comparing DustTrak reference instrument against PurpleAir monitors during laboratory evaluations.
FIGURE 3.2: Distribution of Slope values from linear models comparing DustTrak reference instrument against PurpleAir monitors during laboratory evaluations.
Figure 3.3: Scatter plot of laboratory evaluation comparing Dust-Trak reference instrument against PurpleAir primary sensor (Device ID: 68:C6:8E:A3:CA). The "pm2_5_atm" label refers to the PM$_{2.5}$ mass-density channel of the PurpleAir.
Figure 3.4: Scatter plot of laboratory evaluation comparing DustTrak reference instrument against PurpleAir secondary sensor (Device ID: 68:C6:8E:A3:CA). The "pm2_5_atm" label refers to the PM$_{2.5}$ mass-density channel of the PurpleAir.
3.2 Opportunistic Ambient Calibration

Based on the laboratory calibration findings, we sought a way to calibrate sensors in-situ with ambient particulate matter. Based on other studies (Eeftens et al., 2012; Zhou and Levy, 2007), PM$_{2.5}$ levels can be mainly driven by regional sources thereby limiting small-scale variability. We tested the idea of calibrating the sensors during regional homogeneity events. Homogeneity was assessed using the 4-6 ODEQ nephelometers in the Portland metro area. These nephelometers have been calibrated against the federal reference method. Originally 5-minute resolution data, the nephelometers were aggregated to 15-minute means to identify periods when PM$_{2.5}$ was determined to be regionally homogeneous (the relative standard deviation of all active nephelometers within the Portland metropolitan area was $\leq 10\%$). Once these periods were identified, linear models were generated for each sensor, and those with an $R^2 < 0.90$, or those with $< 100$ observations were excluded from the modeling phase (Appendix A: `homogeneousPM.R`). Violin plots were created to graphically assess the distribution of $R^2$ and slope values during the field calibrations. The $R^2$ values from the field evaluations were not as narrowly distributed below 1.0 relative to the laboratory evaluations, and considerable left-skew of the $R^2$ distribution was observed. This suggests that the some sensors were affected by local PM sources that were otherwise not captured by the network of nephelometers. Figure 3.5 shows the distribution of $R^2$ for the 138 sensors with the nephelometers during homogeneity events. We chose an $R^2$ of 0.90 as our quality filter. After applying the quality filter to the 138 unique sensors observed throughout
the Portland network, a total of 61 (48%) sensors were considered viable for the modeling phase. These 61 sensors corresponded to 35 individual locations. There were 26 nodes which both the primary and secondary sensors passed our quality standards, and there were 9 remaining nodes which only had a single passing sensor. The 35 site total was accumulated throughout the entire 2-year study period, but, on average, there were 20 active sites on any given day. The slopes of the 61 sensors were then graphically assessed via violin plot. The slopes from the field evaluations ranged between 0.31 and 0.80, with a mean of 0.46 (Figure 3.6; Table 3.1). These coefficients are comparable to those determined by other field evaluations, where PurpleAir monitors were co-located with FRM/FEM instruments (AQ-SPEC South Coast AQMD, 2017a).

Scatter plots illustrating the relationship between primary and secondary sensors from a select PurpleAir node against the homogeneous mean PM$_{2.5}$ ambient background concentration were included to highlight the observed intra-model variation in slopes (Figure 3.6). The ‘STAR Lab Creston-Kenilworth’ monitor was an extreme example, and exhibited a 30% difference in slopes between the primary and secondary sensors, despite both sensors producing relatively high $R^2$ values. The primary sensor produced an $R^2$ of 0.970, a slope of 0.45, and a RMSE of 1.19. A mean PM$_{2.5}$ range of 0-50 $\mu$gm$^{-3}$ was observed from the nephelometer network during the evaluation period, with the majority of observations limited to 0-25 $\mu$gm$^{-3}$. The secondary sensor yielded an $R^2$ of 0.967, a slope of 0.31, and a RMSE of 1.25, and encompassed a similar range in nephelometer PM as the primary sensor. Model intercepts were 1.3 and 1.2 $\mu$gm$^{-3}$ for the primary and secondary sensor respectively (Figure 3.7; Figure
Figure 3.5: Distribution of $R^2$ values from linear models comparing PurpleAir network to ODEQ nephelometer-PM during 15-minute periods when PM was regionally homogeneous (Relative Standard Deviation $\leq 10\%$)
FIGURE 3.6: Distribution of slope values from linear models comparing PurpleAir network to ODEQ nephelometer-PM during 15-minute periods when PM was regionally homogeneous (Relative Standard Deviation $\leq 10\%$. Only the PurpleAir monitors with an $R^2 \geq 0.90$ were included within this plot.
FIGURE 3.7: Scatter plot of a select field evaluation comparing mean regional Nephelometer PM$_{2.5}$ against PurpleAir primary sensor (STAR Lab Creston-Kenilworth). The relative point density is captured by the colorscale.
FIGURE 3.8: Scatter plot of a select field evaluation comparing mean regional Nephelometer PM$_{2.5}$ against PurpleAir secondary sensor (STAR Lab Creston-Kenilworth). The relative point density is captured by the colorscale.
3.3 Predictor Selection

The goal of the modeling effort is to predict PM$_{2.5}$ at high spatial resolution and identify the land-use or physical drivers of variability. The next step for model development was to identify variables for the inclusion in the model. Purposeful variable selection was performed partly by software and partly by hand. We relied on previous literature and our understanding of the physical properties of PM$_{2.5}$ to derive our set of proxies for expected sources and sinks (Eeftens et al., 2012; Rao et al., 2014; Brokamp et al., 2017; Saucy et al., 2018; Wang and Ogawa, 2015; Chudnovsky et al., 2014). A list of all initial predictors that were considered are outlined in Table 2.1. Only a single buffer for each spatial predictor was selected for the subsequent modeling phases to avoid issues regarding multicollinearity. We then narrowed down our selected predictors by exploring clusters which covaried. Finally, a stepwise Akaike Information Criterion (AIC) reduction method was employed to further narrow the set of predictors into a final parsimonious model, referred to as the ‘final model’. We relied on the hybrid selection method, which is a combination of both forward and backward selection techniques (Venables and Ripley, 2002). Of the 28 predictors initially considered, only 9 were included within our final explanatory model. It is worth noting that the spatial and spatiotemporal variables each had multiple buffer distances, and the total number of initial predictors was considerably higher than 28 prior to the selection of a single buffer per variable. The 9 final predictors were selected based on correlation strength with the response variable, VIF scores below 5, intuitive signs of model coefficients (i.e. potential sinks
have negative coefficients, and potential sources have positive coefficients), and coefficient p-values below 0.10. The majority of the VIF scores of the predictors within the final model were below 2. The ‘Season’ variable exhibited the highest VIF score of 3.50. This was followed by a score of 2.93 for the maximum daily temperature (‘tmmx.50m’) and 2.16 for daily mean wind speed (‘vs.50m’).

**Table 3.2:** Summary table of the VIF scores from the final model predictors.

<table>
<thead>
<tr>
<th>Predictor</th>
<th>GVIF</th>
<th>Df</th>
<th>GVIF(^{1/(2*Df)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Nephelometer PM(_{2.5})</td>
<td>1.163</td>
<td>1</td>
<td>1.078</td>
</tr>
<tr>
<td>Elevation (50 m)</td>
<td>1.824</td>
<td>1</td>
<td>1.351</td>
</tr>
<tr>
<td>Minimum PBL Height</td>
<td>1.923</td>
<td>1</td>
<td>1.387</td>
</tr>
<tr>
<td>Maximum Relative Humidity (50 m)</td>
<td>1.466</td>
<td>1</td>
<td>1.211</td>
</tr>
<tr>
<td>Maximum Temperature (50 m)</td>
<td>2.934</td>
<td>1</td>
<td>1.713</td>
</tr>
<tr>
<td>Mean Wind Speed (50 m)</td>
<td>2.160</td>
<td>1</td>
<td>1.470</td>
</tr>
<tr>
<td>Developed Medium Intensity (100 m)</td>
<td>1.900</td>
<td>1</td>
<td>1.378</td>
</tr>
<tr>
<td>Season</td>
<td>3.497</td>
<td>3</td>
<td>1.232</td>
</tr>
<tr>
<td>Longitude</td>
<td>1.167</td>
<td>1</td>
<td>1.081</td>
</tr>
</tbody>
</table>

### 3.4 Leave-One-Out-Cross-Validation

Before conducting the LOOCV with the LUR model, the total number of observations for each PurpleAir node (both primary and secondary sensors) was determined. Nodes with fewer than 100 observations were excluded from the LOOCV modeling phase to prevent individual model performance from being unduly influenced by outliers. Overall, 32 of the 34 PurpleAir nodes had more than 100 observations and were used for LOOCV (the primary and secondary sensor data from each node were aggregated). There were 20 nodes which were
established earlier on in the study and captured more than 200 daily observations (Figure 3.9).

The mean $R^2$ from these 32 LOOCV models was 0.87, and ranged between 0.73 and 0.97. The interquartile range in $R^2$ was between 0.84 and 0.90. Model RMSE ranged between 1.31 and 3.71 $\mu g m^{-3}$. Model slopes ranged between 0.67 and 1.06 with a mean of 0.87, and the mean intercept was approximately 0 for most models (Table 3.3). The two nodes which produced LOOCV $R^2$ below 0.80 (‘Haig St’, ‘Upper Multnomah Village’) each had fewer than 200 observations. Differences between LOOCV $R^2$ and final model $R^2$ were typically below 10%, indicating overall model stability.
Chapter 3. Results and Discussion

3.5 Model Validation

Individual predictions from each of the \( n - 1 \) LUR models developed during the LOOCV phase were aggregated to produce a mean ensemble prediction. This ensemble prediction was then compared against the ‘test’ data (the 30% random subset of the corrected PurpleAir data held out during the model training.
Table 3.3: Summary statistics from the LOOCV phase. A validation regression was generated with each LOOCV model and the ‘test’ data to evaluate model stability.

<table>
<thead>
<tr>
<th>Stat</th>
<th>$R^2$</th>
<th>RMSE</th>
<th>Slope</th>
<th>Intercept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.73</td>
<td>1.31</td>
<td>0.67</td>
<td>-1.23</td>
</tr>
<tr>
<td>1st Quartile</td>
<td>0.84</td>
<td>1.64</td>
<td>0.82</td>
<td>0.73</td>
</tr>
<tr>
<td>Median</td>
<td>0.88</td>
<td>2.00</td>
<td>0.88</td>
<td>0.91</td>
</tr>
<tr>
<td>Mean</td>
<td>0.87</td>
<td>2.04</td>
<td>0.87</td>
<td>0.95</td>
</tr>
<tr>
<td>3rd Quartile</td>
<td>0.90</td>
<td>2.29</td>
<td>0.94</td>
<td>1.30</td>
</tr>
<tr>
<td>Max</td>
<td>0.97</td>
<td>3.71</td>
<td>1.06</td>
<td>2.64</td>
</tr>
</tbody>
</table>

Phase (LOOCV phase) to assess overall model performance. Specifically, a validation regression was generated, modeling the observed PM$_{2.5}$ from the ‘test’ data as a function of the LUR predictions. The LUR model explained 0.89% of the variance in the observed data and had a root mean squared error (RMSE) of ± 2.39 $\mu g m^{-3}$. The slope of this model was 0.98, which suggests that the model slightly underpredicts the corrected PurpleAir ‘test’ data. Despite the good model fit, a graphical assessment of the residuals shows that the model appears to perform better at lower PM concentrations, where more training data was available. While a disproportionate number of days with low pollution should generally be considered a benefit, for modeling purposes this left room for improvement when predicting PM$_{2.5}$ above the 20 $\mu g m^{-3}$ threshold. At lower concentrations, between 0-20 $\mu g m^{03}$, the residuals were within the ± 5 $\mu g m^{-3}$ range. However, LUR model performance appeared to decline as the predicted range exceeded this threshold. There are instances where the model both overpredicts and underpredicts PM$_{2.5}$, and this was only exacerbated when concentrations are above 20 $\mu g m^{-3}$ (Figure 3.10). Instances when the observed PM$_{2.5}$ levels were much
higher than the predicted levels suggest days when the regional monitoring network was unable to capture some local PM source at a given PurpleAir node.

The test assumptions of the LUR model were graphically assessed and can be viewed in Figure 3.11. The ‘Residuals vs Fitted’ panel shows the red trendline appears to approximate the zero line, which indicates that the linearity assumption is being met. The ‘Normal Q-Q’ panel shows that the residuals are normally distributed between the -2 and 2 theoretical quantiles, but the normality assumption is not met across the entire range of model residuals. The ‘Scale-Location’ panel shows the red trendline increasing as the fitted values increase. This means that the LUR model does not meet the homoscedascity assumption. However, since we were mostly interested in our ability to predict PM$_{2.5}$ concentrations that typically occur within the urban environment (0 - 20 $\mu$gm$^{-3}$), we were willing to move forward with the model prediction phase, acknowledging that the spatial variability of exceptionally high PM events (e.g. forest fires) must be interpreted with caution. Finally, the ‘Residuals vs Leverage’ panel is used to evaluate potentially influential outliers that located beyond Cook’s distance. Despite our lack of normally distributed residuals, no data points were observed to exceed this distance.

In addition to the LUR model, we explored using a random forest (RF) model for its expectedly higher predictive power and its ability to account for some of the limitations inherent to LUR. Land use regression models have difficulty capturing non-linear relationships and complex interactions, but random forests (RF) are resistant to these problems. Random forests are often preferred
Chapter 3. Results and Discussion

in prediction analyses due to their increased accuracy and resistance to multicollinearity and complex interactions relative to linear regression (Brokamp et al., 2017; Hastie et al., 2005). Our RF model performed slightly better than the LUR model, despite the computational limitations which prohibited an ensemble approach. This single model predicted 0.93% of the variance in the observed data and had a RMSE of $\pm 1.84 \mu g m^{-3}$, or 23% lower than the LUR model (Figure 3.12). The slope of this CV model was 1.0, and the intercept was near zero (-0.07). The residuals from the RF model also appear to be more tightly grouped along the 1:1 line relative to the LUR model. There were still some persistent outliers observed in the residuals of this model. Despite the promising summary statistics generated during the random forest model validation phase, we encountered some limitations while developing spatial predictions and with overall model interpretability that prevented us from relying on this technique instead of LUR. The limitations that we experienced with RF will be discussed in Section 3.9.
FIGURE 3.10: Scatter plot comparing the observed 30% holdout data as a function of the LUR mean ensemble predicted PM$_{2.5}$. 

$\text{R}^2 = 0.8908$  \hspace{0.5cm} \text{Intercept} = 0.18  \hspace{0.5cm} \text{Slope} = 0.98 \hspace{0.5cm} p = 0 \hspace{0.5cm} \text{RMSE} = 2.39$
FIGURE 3.11: A graphical assessment of the LUR model’s ability to meet multiple linear regression test assumptions.
Chapter 3. Results and Discussion

Figure 3.12: Scatter plot comparing the observed 30% holdout data as a function of the Random Forest predicted PM$_{2.5}$

R$^2 = 0.9355$  Intercept = -0.066  Slope = 1  $p = 0$  RMSE = 1.84
3.6 Standardized Regression Coefficients

3.6.1 Temporal Predictors

Nephelometer PM$_{2.5}$

The final model included the median PM$_{2.5}$ of the nephelometer network (the ODEQ regional monitoring network), elevation, the minimum planetary boundary layer height, the maximum relative humidity and temperature, mean wind speed, developed medium intensity LULC, season, and longitude. The standardized regression coefficients (betas) of the final LUR model show that the median nephelometer PM$_{2.5}$ value (‘pm2.5.median’) had the strongest association with PM$_{2.5}$ concentrations observed from the PurpleAir network. A one standard deviation increase in median nephelometer PM$_{2.5}$ corresponded to a 0.908 increase in standard deviation of the PurpleAir network. The unstandardized coefficient for the median nephelometer PM$_{2.5}$ indicates a 0.834 increase in PurpleAir PM$_{2.5}$ for every 1.0 $\mu g m^{-3}$ increase in the ambient background concentration. The daily median nephelometer PM$_{2.5}$ concentration can be interpreted as a temporal scaling factor, and the variation in the PurpleAir network can largely be explained simply by the variation in the ambient background concentration. This was not surprising, as Karner et al. previously identified little to no spatial variation with intra-urban PM$_{2.5}$ (Karner, Eisinger, and Niemeier, 2010). Instead, meteorology mostly drives the temporal variation, while the spatial variability of this pollutant typically occurs at regional scales (Table 3.4). The association of the remaining 8 predictors and PM$_{2.5}$ was much weaker, each with beta values below 0.10.
Planetary Boundary Layer Height

The minimum planetary boundary layer height (‘pbl.min’) was shown to have the 7th highest beta value. An increase in one standard deviation in minimum PBL height resulted in a 0.015 decrease in the standard deviation of our response network. This variable corresponds to the minimum atmospheric volume available for dilution of urban sources. PBL was primarily included to capture days when meteorological conditions were stable, and temperature inversions were expected to trap surface-emitted PM$_{2.5}$, resulting in elevated concentrations during the cold season (Table 3.4). It was surprising that this variable did not have as strong an association with PM$_{2.5}$ as some of the other temporal predictors (e.g. wind speed and maximum daily temperature).

Season

A categorical ‘Season’ variable was included to account for recurring seasonal patterns in PM$_{2.5}$ variation over the 2 year study period. While the the fall season had the fourth highest beta coefficient in the final model, the other two seasons (spring and summer) had some of the lowest associations with the PurpleAir response network (Table 3.4).

Another categorical variable was considered during the initial predictor selection phase to account for patterns in human activity (traffic, area sources) on weekends versus weekdays. This variable was also included to account for the lack of traffic intensity data that has been widely used throughout previous LUR studies. Weekdays were expected to have higher PM$_{2.5}$ levels due to increase traffic intensity during the work week commute relative to weekends.
However, this variable was not determined to be significant during the LUR model development.

### 3.6.2 Spatiotemporal Predictors

**Temperature**

The daily maximum temperature at a 50 m resolution had the second highest beta value. A one standard deviation increase in maximum daily temperature at 50 m (‘tmmx.50m’) resulted in a 0.073 decrease in standard deviation of the PM$_{2.5}$ levels from our response network. Several potential proxies were considered with the inclusion of temperature data in the final LUR model. On the one hand, higher temperatures correspond with a larger mixing volume, providing a surrogate for increased dilution. However, on the other hand, higher temperatures also enable higher rates of secondary organic aerosol formation, potentially leading to higher PM$_{2.5}$ levels in our response network (Wang and Ogawa, 2015). Additionally, higher temperatures are also associated with the wild fire season within the Portland, Oregon, and some of the most polluted days observed within the training data occurred during the warm season. Despite the uncertainties regarding which proxy would potentially drive the association between temperature and PM$_{2.5}$, we chose to include ‘tmmx.50m’ in the final LUR model due to its coefficient’s significance. Here, we can interpret this negative beta coefficient to be due to the influence from wildfires, and perhaps slightly by SOA formation. It is worth noting that although temperature was the second most important predictor its beta was only 8% of the nephelometer-derived PM$_{2.5}$ beta (Table 3.4). In other words, the majority of the variation in
our PurpleAir data was explained by the ambient background concentration of the nephelometers.

**Wind Speed**

Wind speed was identified to be somewhat associated with PM$_{2.5}$, and had the 3$^{\text{rd}}$ highest beta coefficient. This predictor was included as a surrogate for dilution, as PM$_{2.5}$ from local sources are readily removed from the urban environment as wind speeds increase. As this predictor increased by one standard deviation, the response variable was observed to decrease by 0.063 standard deviations (Table 3.4). During data compilation, wind direction was determined to be too coarse of a spatial resolution and was excluded from the modeling phase.

**Relative Humidity**

The maximum relative humidity at 50 m (‘rmax’) was shown to be one of the least associated predictors with the PurpleAir PM$_{2.5}$. An increase of one standard deviation of this variable was associated with only a 0.011 decrease in the response network. Similar to the maximum temperature layer, relative humidity was included to account for multiple potential affects on PM$_{2.5}$ concentrations. Generally, when relative humidity increases, PM$_{2.5}$ mass density is observed to also increase due to hygroscopic growth. However, after a certain relative humidity threshold, PM$_{2.5}$ levels have been observed to decline as hygroscopic growth produces particles which are too large to remain suspended in
the air, ultimately depositing (Wang and Ogawa, 2015). Despite these uncertainties, this variable was included within the final LUR model due to its significant p-value (Table 3.4).

3.6.3 Spatial Predictors

Elevation

Elevation at a 50 m resolution (‘elev.50m’) was shown to be the most influential spatial predictor within the final LUR model. Most of the spatial variation observed in the model predictions approximated this elevation layer. This predictor had the 5th highest beta value, and an increase by one standard deviation in elevation resulted in a 0.03 decrease in standard deviation of our response network. This variable was a proxy for dilution, as wind speeds are typically higher at higher elevations and less shielded by surface roughness. The inclusion of this variable was an effort to capture some of the finer-scale topographical effects on PM$_{2.5}$ that were unaccounted for by the GridMET wind speed product. The majority of primary urban PM$_{2.5}$ is emitted near the surface, and can become trapped at lower elevations during stable meteorological conditions.

Developed Medium Intensity LULC

The developed medium intensity LULC variable at the 100 m buffer size (‘med_dev’ or ‘developed_medium_intensity.100m’) was shown to have the 6th highest beta. A one standard deviation increase in medium developed LULC corresponded to a 0.023 increase in standard deviation of the response network. In other words, the more medium developed LULC observed within 100 m of
a given PurpleAir site, the more likely PM$_{2.5}$ levels were to be slightly higher. This variable was included as a proxy for area sources in effort to characterize the magnitude, however limited, of fine-scale PM$_{2.5}$ gradients. While the ASTER population density layer was originally prioritized over the NLCD predictors as general proxy for all area sources, its coefficient was determined to be insignificant. The inclusion of medium density development LULC was considered only after the population density layer was determined to no longer be a significant predictor. Surprisingly, during the beginning of the predictor selection phase, high density developed land use was not positively associated with PM$_{2.5}$ across the range of buffer sizes, despite the expectation that more area sources would occur within this LULC class. Also, low density developed land use did initially have a positive correlation with our response network, but this variable did not yield a significant coefficient p-value during the first round of modeling. (Table 3.4). While no traffic-related predictors were included within the final model, graphically speaking, the medium density LULC layer appears to approximate the density of major roads throughout the region.

**Longitude**

Longitude (‘lon’) was the second least associated with the PurpleAir response network. A one standard deviation increase in longitude resulted in a 0.013 increase in the standard deviation of PM$_{2.5}$ (Table 3.4). This suggests that the variability of various PM$_{2.5}$ sources and sinks have only a weak association with longitude throughout the Portland, Oregon study area.
### Table 3.4: Coefficients (1) and standardized (beta) coefficients (2) of the final model with standard error in parentheses

<table>
<thead>
<tr>
<th>Dependent variable:</th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Nephelometer PM$_{2.5}$</td>
<td>0.834***</td>
<td>0.908***</td>
</tr>
<tr>
<td></td>
<td>(0.004)</td>
<td>(0.004)</td>
</tr>
<tr>
<td>Elevation (50 m)</td>
<td>−0.003***</td>
<td>−0.030***</td>
</tr>
<tr>
<td></td>
<td>(0.001)</td>
<td>(0.001)</td>
</tr>
<tr>
<td>Minimum PBL Height</td>
<td>−0.0003***</td>
<td>−0.015***</td>
</tr>
<tr>
<td></td>
<td>(0.0001)</td>
<td>(0.0001)</td>
</tr>
<tr>
<td>Maximum Relative Humidity (50 m)</td>
<td>0.007**</td>
<td>0.011***</td>
</tr>
<tr>
<td></td>
<td>(0.003)</td>
<td>(0.003)</td>
</tr>
<tr>
<td>Maximum Temperature (50 m)</td>
<td>−0.058***</td>
<td>−0.073***</td>
</tr>
<tr>
<td></td>
<td>(0.005)</td>
<td>(0.005)</td>
</tr>
<tr>
<td>Mean Wind Speed (50m)</td>
<td>−0.242***</td>
<td>−0.063***</td>
</tr>
<tr>
<td></td>
<td>(0.022)</td>
<td>(0.022)</td>
</tr>
<tr>
<td>Developed Medium Intensity (100 m)</td>
<td>0.555***</td>
<td>0.023</td>
</tr>
<tr>
<td></td>
<td>(0.132)</td>
<td>(0.132)</td>
</tr>
<tr>
<td>Season (Spring)</td>
<td>−0.186**</td>
<td>−0.011</td>
</tr>
<tr>
<td></td>
<td>(0.090)</td>
<td>(0.090)</td>
</tr>
<tr>
<td>Season (Summer)</td>
<td>−0.232*</td>
<td>−0.014</td>
</tr>
<tr>
<td></td>
<td>(0.131)</td>
<td>(0.131)</td>
</tr>
<tr>
<td>Season (Fall)</td>
<td>0.718***</td>
<td>0.051</td>
</tr>
<tr>
<td></td>
<td>(0.073)</td>
<td>(0.073)</td>
</tr>
<tr>
<td>Longitude</td>
<td>0.000001***</td>
<td>0.013***</td>
</tr>
<tr>
<td></td>
<td>(0.00000)</td>
<td>(0.00000)</td>
</tr>
</tbody>
</table>
Chapter 3. Results and Discussion

3.7 Spatial Patterns of Model Residuals

The global Moran’s I of the residuals from each predicted raster was evaluated for spatial autocorrelation. On average, this statistic was $0.99 \pm 0.0005$, indicating a lack of complete spatial randomness in model residuals. Instead, the model error was significantly spatially clustered (a Moran’s I value of 1.0 indicates perfect clustering of similar values). Therefore, some predictors are contributing to the spatially consistent overprediction/underprediction of PM$_{2.5}$ throughout the Portland study area. Specifically, high model standard error was observed at both the global maximum and minimum elevations. Model standard error can also be seen increasing towards the Eastern and Western extents of the study area, suggesting potential edge effects, or an undesirable influence from including longitude as a predictor. Areas with consistently low modeled standard error were observed within the developed medium intensity LULC (Figure 3.13). Most of the PurpleAir monitors were located within this LULC which may explain the lower residuals within the urban core. Overall, the LUR model residuals ranged between 0.05 and 0.25 $\mu g m^{-3}$.
While the Global Moran’s I coefficient of 0.99 suggests the omission of specific variables that may contribute to spatially clustered error, or the inclusion of additional variables that may better explain the variation in intra-urban PM$_{2.5}$, the range in the residuals from the spatially resolved PM$_{2.5}$ predictions were well below the RMSE of the model (2.39). Therefore, we did not account for spatial autocorrelation of LUR model residuals beyond its initial characterization.

![Figure 3.13: A map highlighting the spatially clustered standard error from a single LUR model prediction (January 1st, 2018). All remaining days during the study period exhibited similar patterns in residuals.](image)
3.8 The Importance of Sensor Placement

While we were primarily focused on the anthropogenic drivers of fine-scale PM$_{2.5}$ spatial gradients, and the influence that vegetation may have on mitigating these exposures, ultimately no clear patterns between traffic or vegetation variables could be established. Our assessment of the relative strengths of the various sources and sinks of PM$_{2.5}$ has shown that the median ambient background concentration provided by the ODEQ nephelometer network had the strongest association with PM$_{2.5}$, and the remaining variables were only weakly associated. When comparing the relative strengths of the remaining variables, the effect size of the temporal and spatiotemporal predictors was relatively higher compared to the spatial variables, indicating that the spatial variation of urban PM$_{2.5}$ is primarily driven by meteorology and regional sources at the 50 m scale.

In addition to the observed spatial clustering in our model error, we found the LUR model development to be extremely sensitive to the initial placement of the PurpleAir response network. During the opportunistic ambient calibration phase, we explored several different time aggregations of the nephelometer network in order to identify periods when PM$_{2.5}$ was regionally homogeneous. While we initially selected a 15-minute time resolution to evaluate the RSD of the nephelometer network, we also explored a 1-hour resolution, which produced another set of reasonable correction factors. By implementing a coarser time resolution during the opportunistic ambient calibration, we observed an additional 9 sensors which passed our quality control criterion ($R^2 > 0.90$ and
This corresponded to approximately 5 additional locations (each node contains both a primary and secondary sensor) and an increase in the total number of sensors from 61 to 70. The number of observations used during model development subsequently rose from 11,546 to 12,388 during model development. A new LUR model was developed with these data, and a different set of predictors was selected (despite following the same predictor selection protocol as with the previous model) (Table 3.5). For example, an additional proxy for an anthropogenic source, railroad density, was determined to be a statistically significant predictor with the new data. The log of the railroad density at 500 m produced a beta of 0.015. In addition, a vegetation variable, deciduous forest cover, was also identified as a significant predictor. The deciduous forest predictor had a beta of 0.008, which was the lowest in this model. The relationship between many of the spatial predictors and PM$_{2.5}$ were weak, often with beta coefficients below 0.05. Due to the spatial predictors being weakly associated with PM$_{2.5}$, the addition or removal of a small subset of our response network can lead to a major difference in the predictors selected for the final model. When there are only a few sensors at each LULC to begin with, it becomes difficult to effectively capture the entire variation in each proxy, and model bias increases. The correlations for many of the proxies for anthropogenic sources, e.g. freeways, major arterials, railroads, developed LULC, and PM$_{2.5}$ were teetering above and below 0. The sign of the correlation coefficients for some predictors were observed to flip depending on whether the 15-minute or 1-hour RSD nephelometer data were used to correct the response network. For example, freeways at the 50 m buffer size flipped from a correlation coefficient of 0.01
for the 15-minute data, to -0.01 for the 1-hour data. In reality, this relationship is effectively null.

Patterns in PM$_{2.5}$ were predominately explained by temporal factors over spatial ones. Spatially, PM$_{2.5}$ only varied by 0-5 $\mu g m^{-3}$. Temporally, however, PM$_{2.5}$ levels were observed to increase by tens of micrograms per cubic meter from day to day, especially during wildfire PM events in the summer months, and temperature inversions in the winter months.

If there is a lack of representativeness with the air quality network used to train the model, then model bias for predictions in unrepresented areas will be higher. For example, if only a single sensor is located near railroad that has elevated PM$_{2.5}$ levels, then the model will predict that locations with a similar railroad density will also be elevated, regardless of the train volume on each particular rail.

In addition to the observed spatial clustering in our model error, we found the LUR model development to be extremely sensitive to the initial placement of the PurpleAir response network. During the opportunistic ambient calibration phase, we explored several different time resolutions of the nephelometer network in order to identify periods when PM$_{2.5}$ was regionally homogeneous. While we initially selected a 15-minute time resolution to evaluate the RSD of the nephelometer network, we also explored a 1-hour resolution, which produced another reasonable set of correction factors. By implementing this coarser 1-hour resolution during the opportunistic ambient calibration, we actually observed an additional 9 sensors which passed our quality control criterion ($R^2 > 0.90$ and $n \geq 50$). This corresponded to approximately 5 additional locations.
(each node contains both a primary and secondary sensor) and an increase in the total number of observations from 11,546 to 12,388 during model development. With these data, a new LUR model was developed, and a different set of predictors was selected (following the same predictor selection protocol as with the previous model). For example, an additional proxy for anthropogenic emissions, railroad density, was determined to be a statistically significant predictor with this new data. In addition, a vegetation variable, deciduous forest cover, was also identified as a significant predictor. However, the correlations for many of the spatial predictors with PM$_{2.5}$ were non-existent, often with correlation coefficients below 0.05. Therefore, the addition or removal of a small subset of our response network led to a major difference in the predictors selected for the final model. When there are only a few sensors at each LULC to begin with, it becomes difficult to capture the entire extent of the variation with each proxy, and model bias increases. The correlations for many of the proxies for anthropogenic sources, e.g. freeways, major arterials, railroads, developed LULC, and PM$_{2.5}$ were teetering above and below zero. The sign of the correlation coefficient for some predictors was observed to flip depending on whether the 15 minute or 1 hour RSD nephelometer data were used to correct the response network. For example, freeways at the 50 m buffer size flipped from a correlation coefficient of 0.01 for the 15-minute data, to -0.01 for the 1 hour data, when in reality, this is effectively a null correlation.
TABLE 3.5: Coefficients (1) and standardized (beta) coefficients (2) of a second LUR model, with the standard error included within parentheses. This model was developed using the PurpleAir correction factors data developed with hourly nephelometer data (opposed to the 15-minute data).

<table>
<thead>
<tr>
<th>Dependent variable:</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(2)</td>
</tr>
<tr>
<td>Median Nephelometer PM$_{2.5}$</td>
<td>0.983$^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.004)</td>
</tr>
<tr>
<td>Elevation (50 m)</td>
<td>$-0.003^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.001)</td>
</tr>
<tr>
<td>Minimum PBL Height</td>
<td>$-0.0004^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.0001)</td>
</tr>
<tr>
<td>Log Railroad (500 m)</td>
<td>0.018$^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.006)</td>
</tr>
<tr>
<td>Maximum Relative Humidity (50 m)</td>
<td>0.009$^{**}$</td>
</tr>
<tr>
<td></td>
<td>(0.004)</td>
</tr>
<tr>
<td>Maximum Temperature (50 m)</td>
<td>$-0.062^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.006)</td>
</tr>
<tr>
<td>Mean Wind Speed (50 m)</td>
<td>$-0.219^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.027)</td>
</tr>
<tr>
<td>Season (Spring)</td>
<td>$-0.306^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.108)</td>
</tr>
<tr>
<td>Season (Summer)</td>
<td>$-0.311^{**}$</td>
</tr>
<tr>
<td></td>
<td>(0.158)</td>
</tr>
<tr>
<td>Season (Fall)</td>
<td>0.822$^{***}$</td>
</tr>
<tr>
<td></td>
<td>(0.089)</td>
</tr>
</tbody>
</table>
Chapter 3. Results and Discussion

<table>
<thead>
<tr>
<th>Deciduous Forest (300 m)</th>
<th>$-3.661^*$</th>
<th>$-0.008$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1.933)</td>
<td>(1.933)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Developed Medium Intensity (300 m)</th>
<th>$0.967^{***}$</th>
<th>$0.023$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0.255)</td>
<td>(0.255)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Longitude</th>
<th>$-0.00001^{**}$</th>
<th>$-0.009^{***}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0.00001)</td>
<td>(0.00001)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latitude</th>
<th>$0.00001^{**}$</th>
<th>$0.009^{***}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0.00001)</td>
<td>(0.00001)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>$-32.782$</th>
<th>$0.000$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(26.091)</td>
<td>(26.091)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observations</th>
<th>8,671</th>
<th>8,671</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^2$</td>
<td>0.879</td>
<td>0.879</td>
</tr>
<tr>
<td>Adjusted $R^2$</td>
<td>0.879</td>
<td>0.879</td>
</tr>
<tr>
<td>Residual Std. Error (df = 8656)</td>
<td>2.878</td>
<td>2.878</td>
</tr>
<tr>
<td>F Statistic (df = 14; 8656)</td>
<td>4,500.152^{***}</td>
<td>4,500.152^{***}</td>
</tr>
</tbody>
</table>

*Note:* $^*p<0.1; ^{**}p<0.05; ^{***}p<0.01$

3.9 Predictions

3.9.1 Land Use Regression

Predictions of a select day, October 10th, 2018, were mapped in order to characterize the differences between the 1-hour RSD network and 15-minute RSD network for both LUR and RF models. The spatial patterns from the 15-minute RSD LUR model appear to approximate the underlying elevation and developed medium intensity LULC predictors, despite the betas of these variables being ranked only the 5th and 6th most associated with PM$_{2.5}$. Since the general range
of PM$_{2.5}$ levels for any given day is primarily governed by the median nephelometer values, and other temporal scaling factors such as wind speed and temperature, any remaining spatial heterogeneity is typically confined within a ± 5 µgm$^{-3}$ range from the median. For example, the exact range in predicted PM$_{2.5}$ on this day was 6.25 - 8.62 µgm$^{-3}$, a difference of only 2.37 µgm$^{-3}$ from the most to least polluted location. The 1-hour RSD LUR model captures similar patterns in elevation as the previous model, but the railroad and deciduous forest variables suggest different hot spots and cold spots. For example, the Western extent of the study area shows elevated PM$_{2.5}$ levels due to the presence of railroads, whereas the previous model indicated this region should have lower levels. In addition, small pockets of low PM$_{2.5}$ concentrations scattered throughout the region correspond to the presence of deciduous forest cover. From the addition of these two spatial variables, the predicted range in PM$_{2.5}$ concentration grew to 5.37 - 9.62 µgm$^{-3}$, a 4.25 µgm$^{-3}$ difference between most and least polluted locations (Figure 3.14). The spatial heterogeneity in PM$_{2.5}$ on a given day is further limited when the model RMSE is taken into consideration. The 15-minute RSD model, previously referred to as the ‘final model’, had a RMSE of 2.39. When the RMSE is applied to the range in predicted PM$_{2.5}$, it becomes difficult to interpret most spatial patterns. For example, the cold spots observed on October 10$^{th}$, 2018 could potentially overlap with the hotspots, and vice versa (e.g. $6.25 + 2.39 \approx 8.62 - 2.39$). In addition to the graphical assessment, we regressed the 1-hour RSD model against the 15-minute RSD model to further assess the relationship between the two models (Figure 3.15). The 15-minute RSD model (‘final model’) explained less than 1% of the variation in
the 1-hour RSD model, indicating that the inclusion of 4 PurpleAir sites has resulted a significantly different set of predictions. Each point within Figure 3.15 represents the two predicted PM$_{2.5}$ values of a single 50 m by 50 m cell within our raster, itself consisting of $1.031e^6$ total cells. The areas where the 1-hour RSD model predicts considerably lower than the 15-minute RSD model are due to the inclusion of the deciduous forest cover LULC. The 1-hour RSD model can also be seen predicting higher than the 15-minute model within the 500 m railroad buffer.

3.9.2 Random Forest

Exposure maps of the random forest (RF) predictions illustrate some of the limitations of using machine learning techniques over traditional LUR models. The range in PM$_{2.5}$ from the 15-minute and 1-hour RSD RF models was 6.63 - 10.78 µgm$^{-3}$ and 6.74 - 10.55 µgm$^{-3}$ respectively (Figure 3.16). Although the RF model appeared to perform marginally better during the validation phase, with higher validation regression $R^2$ and lower RMSE, the RF predictions show unrealistic gradients, and appear to overfit on the latitude and longitude variables. Specifically, explicit vertical thresholds in both the 15-minute, and 1-hour RSD models can be seen due to the inclusion of the longitude predictor. A horizontal threshold in the 1-hour RSD model predictions, while less pronounced, is also evident because this model found latitude to be a significant predictor. These RF results stress the importance of understanding the physical processes that drive PM$_{2.5}$ variation, as relying solely on the statistical measures of model performance may be misleading. Unlike the traditional LUR approach, RF models are
essentially a black box and do not provide model coefficients necessary for interpreting the relative strengths of sources and sinks. As with many other machine learning algorithms, random forest is a ‘data hungry’ technique, and relies on a substantial training set in order to develop a robust stable model. While our training set included more than 10,000 observations, these were generated from a limited number \((n \approx 20)\) of physical locations which repeated over a 2-year period. In comparison, the study area in which we are attempting to predict includes over \(1.031 \times 10^6\) individual grid cells at the 50 m resolution. We are likely not capturing enough variation in each of our spatial predictors necessary to develop a meaningful RF model for predicting intra-urban \(\text{PM}_{2.5}\).
**Figure 3.14:** Spatially resolved PM$_{2.5}$ predictions of Portland, Oregon on October 10$^{th}$, 2018 from two LUR models that were developed using 15-minute and 1-hour resolution RSD to correct the PurpleAir network. The 1-hour RSD model includes rail density and deciduous forest density predictors that were not selected in the 15-minute model.
Figure 3.15: Scatter plot comparing the predicted PM$_{2.5}$ on 2018-10-10 from the 1-hour RSD model as a function of the 15-minute RSD model.
Chapter 3. Results and Discussion

3.9.3 Final Model Predictions Over Time

We used the ‘final model’ to predict 573 consecutive days (September 2017 - March 2019) of PM$_{2.5}$ throughout the Portland metropolitan area. An animated figure which cycles through each daily prediction can be viewed at the following hyperlink: http://web.pdx.edu/~porlando/animated-pm.gif. This animation allowed us to identify periods of interest, including a forest fire haze event that occurred during August, 2018 and periods with stable meteorology and wood
burning in December, 2017. The spatial pattern appears to be relatively consistent even on the most polluted days (Figures 3.17, 3.18).

**Figure 3.17:** Spatially resolved PM$_{2.5}$ predictions of Portland, Oregon capturing a forest fire haze event that occurred between August 17$^{th}$ - 25$^{th}$, 2018 from the ‘final model’.
**Figure 3.18:** Spatially resolved PM$_{2.5}$ predictions of Portland, Oregon during stable meteorological conditions and higher rates of wood-burning between December 14$^{th}$ - 22$^{nd}$, 2017 from the ‘final model’.
Chapter 4

Summary and Future Work

4.1 Summary

Spatially resolved PM$_{2.5}$ concentrations at the intra-urban scale are not well defined throughout US cities. This is due to the practical constraints of deploying high-density networks using costly research-grade monitoring equipment, and the focus on compliance at the regional scale from regulatory agencies. Monitoring networks for PM$_{2.5}$ are scarce, typically between 4-6 monitors per city, and are often deliberately placed away from local sources in order to assess the ambient background concentration. The focus on compliance ultimately results in the potential lack of representativeness of fixed air quality monitoring sites for epidemiological studies, and limits studies which aim to characterize the potential role vegetation may play in mitigating PM$_{2.5}$ exposure. Land use regression techniques have been developed to account for the spatial heterogeneity of air pollution sources within urban areas, in an effort to provide spatially resolved exposure maps and identify the relative strengths of various sources and sinks. This thesis explored the potential of using low-cost sensors for modeling spatially resolved PM$_{2.5}$ exposures within Portland, Oregon at a 50 m resolution.
during a 2 year period. The low-cost sensor network increased the density of the existing monitoring network by a factor of 5, enabling us to better characterize the potential patterns in sources and sinks throughout the region necessary for producing spatially resolved PM$_{2.5}$ exposures. However, we observed the spatial predictors of PM$_{2.5}$ to be only weakly associated with our response network, relative to temporal scaling factors. This was not entirely surprising based on the lack of fine-scale spatial variability in PM$_{2.5}$ observed by previous studies (Karner, Eisinger, and Niemeier, 2010). In addition, the predicted range of PM$_{2.5}$ on a given day was typically within the RMSE of the LUR model. While many air pollutants within cities have strong pollution gradients, typically decaying within 150 m - 200 m from the source, spatial patterns of PM$_{2.5}$ are not as distinct. As identified by Karner et al., gradients of PM$_{2.5}$ with increasing distance from traffic-related sources are mixed, and either decrease very gradually or show no clear trend altogether (Karner, Eisinger, and Niemeier, 2010; Kendrick, Koonce, and George, 2015). We could not identify a significant association between our PurpleAir response network and any of the proxies for traffic sources. The only anthropogenic source associated with PM$_{2.5}$ was the medium intensity developed LULC. This predictor was observed in both the 1-hour and 15-minute RSD models, which may be a sign of developed land use as a proxy for area sources. In contrast, no clear associations between proxies for vegetation and PM$_{2.5}$ were established during these initial modeling efforts, and only a single vegetation LULC was identified during the additional 1-hour RSD model. We characterized the sensitivity to initial sensor placement when developing LUR models, as an addition of 4 sensors resulted in a considerably
different set of predictions throughout the study period. Despite the challenges inherent to modeling spatially resolved PM$_{2.5}$ with low-cost sensor data, this thesis has developed a method for modeling intra-urban PM$_{2.5}$ that can be generalized to many other cities, and produced a set of exposure maps which are a notable improvement over the alternative exposures generated from spatially interpolating the existing regulatory monitoring sites.

4.2 Future Work

We have developed a method for modeling spatially resolved PM$_{2.5}$ at the sub-neighborhood scale that can be applied to the remaining four cities within the Canopy Continuum project. Future work will explore the associations between a similar set of land use and meteorological predictors with each of the corresponding low-cost PM$_{2.5}$ networks deployed throughout Albuquerque, New Mexico, Boise, Idaho, Sacramento, California, and Tacoma, Washington. Since these cities exhibit a varying degree of urban canopy cover, the development of additional LUR models, and the resulting inter-city comparison may provide further insights regarding the potential role of vegetation as a sink for PM$_{2.5}$. The inclusion of some additional key predictors should also be considered with future models. As outlined by the ESCAPE project, traffic intensity data was shown to be a crucial variable when predicting spatially resolved PM$_{2.5}$ across many European cities, where PM emissions are higher due to higher levels of diesel engines, and study areas without local traffic intensity resulted in poorer models. While we incorporated road density data of several road classes (streets, arterials, freeways, etc.), the average annual daily traffic volume of
Chapter 4. Summary and Future Work

each road segment was not accounted for. In addition to traffic-intensity data, the performance of ESCAPE models for predicting PM absorbance, a marker of black carbon, were generally higher than that of PM$_{2.5}$. This suggests that additional monitoring networks may be required in order to characterize anthropogenic sources. While a substantial fraction of European private cars, and nearly all middle and heavy duty vehicles, use diesel instead of gasoline, the inclusion of PM absorbance within our LUR models may better reflect the impact that mobile emissions have on fine-scale PM exposures within the urban environment (Eeftens et al., 2012). The inclusion of particle number concentration, specifically of ultrafine PM, should also be considered in future modeling efforts, as this pollutant is another proxy for emissions from local sources. The two Plantower sensors inside each PurpleAir monitor provide additional mass density measurements for PM$_1$ and PM$_{10}$. Plantower sensors are also capable of measuring the number density of particles as small as 300 nm in diameter (PM$_{0.3}$). Future work should consider the inclusion of the various different PM channels recorded by the PurpleAir network in effort to further characterize the land-use and physical drivers of intra-urban PM variability (Appendix B).

Sensor placement, as we previously demonstrated, can have a cascading effect during model development. Two areas of improvement have been identified from these results: (1) a stratified sensor allocation method and (2) an alternative field calibration technique. A stratified sensor allocation method would better approximate the true variation in each of the relevant spatial predictors. The opportunistic placement of the PurpleAir sensors in Portland resulted in a network which overrepresented urban areas near small streets, but lacked data
for both heavily trafficked and heavily vegetated areas, and did not fully capture the extent of the urban-rural gradient. A previous study which developed fine-scale exposure maps of NO$_2$ throughout Portland, Oregon relied on a stratified random network of 144 passive samplers (Rao et al., 2014). Unfortunately, a PM$_{2.5}$ monitoring campaign of this caliber still poses many logistical challenges even for low-cost sensors. The PurpleAir monitors are still limited by cost (a PurpleNetwork with 144 sites would cost $36,000), power and telecommunication. Identifying remote locations that still provide access to an AC power source and a WiFi connection would likely be prohibitive with a stratified random sensor allocation. Our opportunistic ambient calibration method, while a major improvement over using the uncorrected sensor data, recovered only 48% of the PurpleAir network. An alternative field calibration method, for example a pre- and post-study calibration of each individual sensor against a co-located FRM in the field may return a higher data recovery rate, while enabling the assessment of sensor drift overtime. This is especially important as we demonstrated that the inclusion of a just a few sensors can have a significant impact on the predicted spatial variability.
Bibliography


Beckerman, Bernardo S., Michael Jerrett, Marc Serre, Randall V. Martin, Seung-Jae Lee, Aaron van Donkelaar, Zev Ross, Jason Su, and Richard T. Burnett


Bibliography


Didan, K (2015a). MOD13A2 MODIS/Terra Vegetation Indices 16-Day L3 Global 1km SIN Grid V006, NASA EOSDIS LP DAAC.


Eeftens, Marloes, Rob Beelen, Kees de Hoogh, Tom Bellander, Giulia Cesaroni, Marta Cirach, Christophe Declercq, Audrius Dédélé, Evi Dons, Audrey de Nazelle, Konstantina Dimakopoulou, Kirsten Eriksen, Grégoire Falq, Paul Fischer, Claudia Galassi, Regina Gražulevičienė, Joachim Heinrich, Barbara Hoffmann, Michael Jerrett, Dirk Keidel, Michal Korek, Timo Lanki, Sarah Lindley, Christian Madsen, Anna Mölter, Gizella Nádor, Mark Nieuwenhuijzen, Michael Nonnemacher, Xanthi Pedeli, Ole Raaschou-Nielsen, Evridiki Patelarou, Ulrich Quass, Andrea Ranzi, Christian Schindler, Morgane Stempfelet, Euripides Stephanou, Dorothea Sugiri, Ming-Yi Tsai, Tarja Yli-Tuomi, Mihály J Varró, Danielle Vienneau, Stephanie von Klot, Kathrin Wolf, Bert Brunekreef, and Gerard Hoek (Oct. 16, 2012). “Development of Land Use Regression Models for PM2.5, PM2.5 Absorbance, PM10 and PMcoarse in 20 European Study Areas; Results of the ESCAPE Project”. In: Environmental Science & Technology 46.20, pp. 11195–11205. ISSN: 0013-936X. DOI:


Bibliography


Jerrett, Michael, Shankardass Ketan, Berhane Kiros, Gauderman W. James, Künzli Nino, Avol Edward, Gilliland Frank, Lurmann Fred, Molitor Jassy N., Molitor John T., Thomas Duncan C., Peters John, and McConnell Rob
Bibliography


Kloog, I., B. Ridgway, P. Koutrakis, B. A. Coull, and J. D. Schwartz (July 2013). “Long- and Short-Term Exposure to PM2.5 and Mortality”. In: *Epidemiology*


Mattiuzzi, Matteo and Florian Detsch (2019). MODIS: Acquisition and Processing of MODIS Products. URL: https://CRAN.R-project.org/package=MODIS.


Pullman, Marcie (Aug. 21, 2008). “Conifer PM2.5 Deposition and Resuspension in Wind and Rain Events”. In: URL: https://ecommons.cornell.edu/handle/1813/11239 (visited on 07/04/2019).


Bibliography


Ulrich, R. S. (Apr. 27, 1984). “View through a window may influence recovery from surgery”. In: Science 224.4647, pp. 420–421. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.6143402. URL: https://science.sciencemag.org/content/224/4647/420 (visited on 07/04/2019).


Appendix A

Source Code

Source/animateStack.R

```r
# load the necessary packages

if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(raster, rasterVis, viridis, animation, classInt, pbapply, parallel)

mods <- c("lm", "rf", "xgb")
pblapply(mods, function(mod) {
  s <- stack(list.files("./data/predicted_raster", paste0(mod, "\tif$"), full.names = T))
  r <- s[[1]]

  # breaks <- seq(0,100,5) # focus on lower level variation
  breaks <- c(seq(0, 50, 1), Inf)
  # cols <- colorRampPalette(c("darkgreen", "lightgreen", "yellow", "orange", "red", "maroon"))(length(breaks)-1)
  # cols <- magma(length(breaks)-1)
  cols <- colorRampPalette(c("darkgreen", "forestgreen", "yellow", "orange", "red", 
                              "A62A2A"))(length(breaks)-1)
```

# sequence <- seq(0, 100, 5)
# n <- length(sequence)

# Match EPA PM2.5 AQI Palette
# pal <- c("#01CC01", "#66CC00", "#FFFF01", "#FE9900", "#FC0200", "#A62A2A")
# n <- length(pal)
# sequence <- c(-Inf, 31, 61, 91, 121, 250, 1000)
# classes <- classIntervals(values(r)
# ,n = n
# ,style = "fixed", precision = 2
# ,fixedBreaks = sequence
# ,fixedBreaks = c(5, seq(0, 100, 10))
# )

# brks <- classes$brks
# brks <- round(brks, 2)
# cols <- plasma(n)
# cols <- pal

saveGIF{

# ani.options(ani.height = 6
# ,ani.width = 8
# ,ani.res = 300
# ,ani.units = "in"),

for(i in c(1:nlayers(s))){

r <- s[[i]]

r[is.na(r[])] <- 0
r[r<0] <- 0

l <- levelplot(r
,at = breaks
,col.regions=cols
# ,colorkey=list(
# #at=breaks
# # at = seq(0,50,10)
# # ,labels=c(as.character(breaks
# )) # too many labels
# ,labels=c(as.character(seq(0,
# 50, 10))))
# )

# ,colorkey=list(title = expression(mu*g*m^-3),
# space = "top")

# ,colorkey=list(space = "left")
# ,margin=FALSE
Appendix A. Source Code

```r
, main = gsub("^X", "", names(s[[i]]))
  #, ylab = expression(mu * g/m^3)
)

# l <- plot(s[[i]]
# , breaks = brks
# , col = magma(256)
# , main = gsub("^X", "", names(s[[i]]))
# , ylab = "ug/m3"
#
plot(l)
rm(r)
}
}
}, interval = 0.2, movie.name = paste0("/home/porlando/Documents/purpleairlur/figures/animations/", mod, "_test.gif"))
dev.off()
}
# , cl = detectCores() - 1
)
#
# # Use the right pattern for each model type (lm, rf, xgb, se, etc.)
# s <- stack(list.files("./data/predicted_raster", "lm\.tif\$", full.names = T))
#
# # This is taking forever!
# # classes <- classIntervals(values(s), n = 10, style = "fisher", precision = 3)
#
# r <- s[[1]]
# classes <- classIntervals(values(r), n = 10, style = "fixed", precision = 2
# , fixedBreaks = c(0, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100)
#
# brks <- classes$brks
# brks <- round(brks, 2)
#
# # Absolute path required by ImageMagick?
# saveGIF()
# for(i in c(1:nlayers(s))){
# 1 <- levelplot(s[[i]], colorkey = list(at = brks, labels = c(as.character(brks)))
#  , margin = FALSE
#  , main = gsub("^X", "", names(s[[i]]))
#  , ylab = "ug/m3"
```
Appendix A. Source Code

```r
# # 1 <- plot(s[[i]])
# # , breaks = brks
# # , col = magma(256)
# # , main = gsub("^X", ",", names(s[[i]]))
# # , ylab = "ug/m3"
# # )
# plot(l)
# }
# } , interval = 0.2, movie.name="/home/porlando/Documents/purpleairlur/
# figures/animations/test.gif")
#
# # pbapply doesn't work...
# # saveGIF(
# # pbapply(1:layers(s), function(i) {
# # 1 <- levelplot(s[[i]], colorkey=list(at=brks, labels=c(as.
# character(brks))), margin=FALSE)
# # plot(l)
# # }, cl=detectCores() - 1), interval = 0.2, movie.name="/home/porlando/
# /Documents/purpleairlur/figures/animations/test.gif")

Source/annualNDVI.R

```
Appendix A. Source Code

26 ndvi <- stack(list.files(path = "./output/ndvi/", pattern = ".tif$", full.names = T))
27 # levelplot(ndvi)
28
ev <- stack(list.files(path = "./output/evi/", pattern = ".tif$", full.names = T))
29
gDates <- function(stack) {
  names <- names(stack)
  dates <- gsub("^.\.*\\\.", "\", names)
  dates <- gsub("\ \.", "-", dates) %>% as.Date()
  return(dates)
}
31
gVars <- function(stack) {
  names <- names(stack)
  # extract variable name (characters before date)
  vars <- gsub("\\\..*$", "\", names)
  return(vars)
}
35
# Calculate annual median raster for NDVI and EVI
annualNDVI <- function(stack, proj = epsg_26910) {
  # 16 day observations
  bi_month <- gDates(stack)
  vars <- gVars(stack)
  # generate daily sequence
  dates <- seq.Date(from = min(bi_month), to = max(bi_month), by = "day")
  dates <- gsub("-", "\\.", dates)
  daily_names <- paste0(vars, ".", dates)
  # Grab the years across our date range
  years <- seq.Date(from = min(bi_month), to = max(bi_month), by = "year") %>% lubridate::year()
  # Compute annual rasters
  pblapply(years, function(y) {
    r <- subset(stack, grep(y, names(stack), value = T)) %>% calc(
      median, na.rm = T)
    annual_median <- raster::projectRaster(from = r, res = sp_res,
      method = "bilinear", crs = proj)
    names(annual_median) <- paste0(vars[1], ".", y)
    annual_names <- grep(y, daily_names, value = T)
    # This isn’t efficient, but I don’t want to rewrite my buffer/extract scripts ...
  })
}
s <- lapply(annual_names, function(d) {
  r <- annual_median
  names(r) <- d
  return(r)
}) %>% stack
writeRaster(s, file.path('~/data', unique(vars), paste0(names(s), ".tif")))
  ,overwrite=T
  ,bylayer=T
  ,format="GTiff"
}
,cl = detectCores()-1)
}
# Call our annual processing function
pbapply(c(ndvi, evi), annualNDVI
  ,cl = detectCores()-1)

# ndvi2 <- stack(list.files("~/data/ndvi", full.names = T))
# ndvi2 <- readRDS("~/data/ndvi_daily_composite.RDS")
# names(ndvi2)
# plot(ndvi2)
# evi2 <- stack(list.files("~/data/evi", full.names = T))
# evi2 <- readRDS("~/data/evi_daily_composite.RDS")
# names(evi2)
# test <- evi2[[17:32]]
# plot(test)
#
# lapply(1:nlayers(ndvi), function(x) {
#  plot(ndvi[[x]], main = paste(names(ndvi[[x]]))))
#  Sys.sleep(0.3)
# })
#
# lapply(1:nlayers(ndvi2), function(x) {
#  plot(ndvi2[[x]], main = paste(names(ndvi2[[x]]))))
#  Sys.sleep(0.3)
# })
#
# lapply(1:nlayers(evi), function(x) {
#  plot(evi[[x]], main = paste(names(evi[[x]]))))
#  Sys.sleep(0.5)
# })
#
# lapply(1:nlayers(evi2), function(x) {
#  plot(evi2[[x]], main = paste(names(evi2[[x]]))))
# Compare CF from 1 hour and 15 minute data

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(dplyr, ggplot2)

pm_1h <- readRDS("./data/homogenousPM/lm_results_1-hour.RDS") %>%
  dplyr::select(r.squared, slope, id, n_obs, sigma)

pm_15m <- readRDS("./data/homogenousPM/lm_results_15-min.RDS'") %>%
  dplyr::select(r.squared, slope, id, n_obs, sigma)

pm_wide <- full_join(pm_1h, pm_15m, by = "id", suffix = c(".1h", ".15m"))

pm_wide$slope_diff <- abs(pm_wide$slope.1h - pm_wide$slope.15m)

ggplot(filter(pm_wide, r.squared.1h >= 0.9 & r.squared.15m >= 0.90),
  aes(x = as.factor(id), y = slope_diff)) + geom_col()

pm_wide %>% filter(r.squared.1h >= 0.9 & r.squared.15m >= 0.90) %>%
  dplyr::select(slope_diff) %>% boxplot()

pm_wide %>% filter(r.squared.1h >= 0.9 & r.squared.15m >= 0.90) %>%
  dplyr::select(slope_diff) %>% summary()

boxplot(pm_wide$slope_diff)

# Tidy format

pm_1h$time_res <- "1_hour"

pm_15m$time_res <- "15_min"

pm <- bind_rows(pm_1h, pm_15m)

pm_group_summary <- pm %>% filter(r.squared >= 0.90) %>% group_by(
  time_res) %>% summarise(min_r2 = min(r.squared),
  q1_r2 = quantile(r.squared, probs = 0.25),
  med_r2 = median(r.squared),
  mean_r2 = mean(r.squared),
  q3_r2 = quantile(r.squared, probs = 0.75)
\[ \max_r2 = \max(r.squared) \]
\[ q1_slope = \text{quantile}(slope, \text{probs} = 0.25) \]
\[ med_slope = \text{median}(slope) \]
\[ mean_slope = \text{mean}(slope) \]
\[ q3_slope = \text{quantile}(slope, \text{probs} = 0.75) \]
\[ max_slope = \max(slope) \]
\[ mean_rmse = \text{mean}(sigma) \]

```
pm_group_summary
saveRDS(pm_group_summary, "./data/homogenousPM/pm_summary_by_time_res.RDS")
# pm %>% na.omit() %>% ggplot(aes(x = as.factor(id), y = slope, fill = time_res)) +
  # geom_boxplot()
```

**Source/**downloadGridMET.R

```
library(raster)

# Load global project variables
source("./src/init.R")

# downloads met data for 1 year for 1 variable
getGridMetVar <- function(dir = dir
  ,year = year
  ,var = var) {
  file_name <- paste0(var, "_", year, ".nc")
  file_path <- paste0("./", dir, "/", file_name)

  # check if file exists
  if(!exists(file_path)) {
    print(paste0("Overwriting ", file_path))
    file.remove(file_path)
    url <- paste0("https://www.northwestknowledge.net/metdata/data/", file_name)
    system(paste0("wget --directory-prefix="!, dir
               , ", load-cookies ~/.cookies --save-cookies ~/
               , cookies --no-parent ", url))
  } else {
    ```
Appendix A. Source Code

```r
print(paste0("Downloading ", file_path))
url <- paste0("https://www.northwestknowledge.net/metdata/data/",
    file_name)
system(paste0("wget --directory-prefix=", dir
    , " --load-cookies ~/.cookies --save-cookies ~/.cookies
    --no-parent ", url))
```

```r
getGridMetVars <- function(dir, vars, year) {
lapply(vars, function(x) getGridMetVar(dir = dir
    ,year = year
    ,var = x))
}
```

```r
getGridMetYears <- function(dir, vars, years) {
lapply(years, function(x) getGridMetVars(dir = dir
    ,year = x
    ,vars = vars))
}
```

```r
getGridMetYears(dir = "gridMET/nc",
    vars = c("pr" # precip
    ,"sph" # specific humidity
    ,"rmin" # min rh
    ,"rmax" # max rh
    ,"vs" # wind speed 10m
    ,"th" # wind direction 10m
    ,"tmmn" # min temp
    ,"tmmx") # max temp
    ,years = c(seq(2017, 2019, 1))
    ,years = c(seq(lubridate::year(start_date), lubridate::year(end_date), 1)))
```

Source/downloadNDVI.R

```r
# load the necessary packages
if (!require(pacman)) {
install.packages("pacman")
library(pacman)
}
```
source("./src/functions.R")
source("./src/init.R")

# set MODIS options (these can also be passed to `...` in getHdf())
MODISoptions(localArcPath = "./", quiet = FALSE, outDirPath = "./"

# download the data 1km
# tfS <- runGdal(product = "MOD13A2", collection = "006"
#                , begin = str_replace_all(start_date, ";","."))
#                , end = str_replace_all(end_date, ";","."))
#                , job = "temporalComposite"
#                , SDSstring = "100000000010"
#                , overwrite = TRUE)

# 250 m Terra (Aqua is MYD13Q1)
getProduct("MOD13Q1")
getProduct("MYD13Q1")
print("begin download")
tfs <- runGdal(product = "MOD13Q1", collection = "006"
              , tileH = 9, tileV = 4
              , begin = "2010.01.01"

getProduct("MOD13Q1")
getProduct("MYD13Q1")
print("begin download")
tfs <- runGdal(product = "MOD13Q1", collection = "006"
              , tileH = 9, tileV = 4
              , begin = "2010.01.01"
53, begin = str_replace_all(start_date, "−", ".")
54, end = str_replace_all(end_date, "−", ".")
55, #, end = "2019.01.01"
56, #, job = "temporalComposite" # prevents new directory
57, with timestamp from being created
58, #, SDSstring = "100000000010"
59, #, outDirPath = "./MODIS/MOD13Q1.006"
60, overwrite = TRUE)
61
62
63
64
65
66
67 # rLandsat...
68 # if(!require(rLandsat)) {
69 #   devtools::install_github("socialcopsdev/rLandsat", upgrade_ 
70 #     dependencies=T)
71 #   library(rLandsat)
72 # }
73 #
74 # devtools::install_github(’azvolef/wrspathrow’, upgrade_ 
75 #     dependencies=T)
76 #
77 # result <- landsat_search(min_date = start_date, max_date = end_date 
78 #   ,row_master = c(28) # had to google this
79 #   ,path_master = c(46)
80 # )
81 #
82 # result %>% filter(cloudCover <= 10)
83 #
84 # row_id <- 28
85 # path_id <- 46
86 #
87 # downloadLandsat <- function(start_date, end_date, row_id, path_id, 
88 #   cloud_cover, dir) {
89 #   # Create output directory if it doesn’t exist yet
90 #   if(!dir.exists(dir)) {
91 #     dir.create(dir)
92 #   }
93 #
94 #   # Create list of download links via rLandsat
95 #   links <- rLandsat::landsat_search(min_date = start_date, max_date 
96 #     = end_date
97 #   )
# , row_master = row_id # had to google this

... 
# , path_master = path_id ) %>% filter( 
cloudCover <= cloud_cover )
# # links <- links [1 , ]
# pblapply(1:nrow(links), function(i) {
# urls <- links$download_links.google[i] %%
# unlist() %%
# grep(pattern = '.*_B2.TIF|.*_B4.TIF|.*_B5.TIF', value=T) # NIR and Red bands
# pblapply(urls, function(url) {
# system(paste0("wget --directory-prefix=", dir
# "", --load-cookies ~/.cookies --save-cookies ~/.
# cookies ", url))
# }
# 
# , cl = detectCores()-1
# )
# }
# 
# , cl = detectCores()-1
# )
# }
# }

# downloadLandsat(start_date = start_date, end_date = end_date, row_id = c(28), path_id = c(46),
# cloud_cover = 10, dir = "/data/landsat")

# # Is this because of the cookies argument?
# 
# # espa_creds("porlando", scan("batteries.espa", what=""))
# # prods = espa_products(result$product_id)
# # prods = prods$master
# #
# # result_order = espa_order(result$product_id
# # , product = c("sr", "sr_ndvi")
# # , projection="lonlat"
# #)
# #
# #
# # # get all the product IDs for India, alternatively can define path and row
# # result = landsat_search(min_date = "2018-01-01", max_date =
# "2018-01-16", country = "India")
# #
# # # inputting espa creds
# # espa_creds("porlando", scan("batteries.espa", what=""))
# #
# # # getting available products
# # prods = espa_products(result$product_id)
# # prods = prods$master
# #
# # # placing an espa order
# # result_order = espa_order(result$product_id, product = c("sr","sr
# # ndvi"),
# #     projection = "lonlat",
# #     order_note = "All India Jan 2018"
# # )
# # order_id = result_order$order_details$orderid
# #
# # # getting order status
# # durl = espa_status(order_id = order_id, getSize = TRUE)
# # downurl = durl$order_details
# #
# # # download; after the order is complete
# # landsat_download(download_url = downurl$product_dload_url, dest_file = ".\test")
# #
# # # Download dependencies
# # # system("sudo apt-get update")
# # # system("sudo apt-get install python-pip python-numpy python-scipy libgdal-dev libatlas-base-dev gfortran libfreetype6-dev")
# # # system("pip install landsat-util")
# #
# # # Extract coordinates for each corner of our study extent
# # study_area <- readRDS("./study_area/study_area.RDS") %>%
# #     spTransform(wgs_84)
# # min_lon <- study_area@bbox[1]
# # min_lat <- study_area@bbox[2]
# # max_lon <- study_area@bbox[3]
# # max_lat <- study_area@bbox[4]
# #
# # # landsat-utilities method, 0 items were returned!
# # system(paste0("landsat search --cloud 4"
# #     , " --lat ", min_lat
# #     , " --lon ", min_lon
# #     , " --start ", start_date
# #     , " --end ", end_date
# # ))
# #
# # landsat_search(collection="landsat-8", cloud_cover=c(0,20), limit = 3)$features
# #
# # "http://landsat-pds.s3.amazonaws.com/L8/003/017/
# # LC80030172015001LGN00/index.html"
Source/downloadPBL.R

```r
# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal,
   rgeos,
   sp,
   raster,
   dplyr,
   magrittr,
   pbapply,
   parallel,
   snow)

source("./src/init.R")

# create a year sequence based on our init start/end data
# years <- seq.Date(as.Date(start_date), as.Date(end_date), by = "year") %>% lubridate::year()
years <- seq(lubridate::year(start_date), lubridate::year(end_date), by = 1)

getPBL <- function(dir = dir,
                   year = year,
                   var = var) {
                 var, ".", year, ".nc")
  system(paste0("wget -c --directory-prefix="dir,
                 " --load-cookies ~/.cookies --save-cookies ~/.cookies ",
                 url))
}

getPBLYears <- function(dir, var, years) {
  # Overwrite flag added to wget (have not confirmed that it works 
  # Remove existing files
  # f <- list.files(path = dir, pattern = paste0("*.nc"), full.names = T)
```
Appendix A. Source Code

```r
# lapply(f, file.remove)

# Add in new files
pblapply(years, function(x) getPBL(dir = dir
  ,year = x
  ,var = var))

getPBLYears(dir = ".:/data/PBL/nc"
  ,var = c("hpbl")
  ,years = years)
```

`Source/edaPredictors.R`

```r
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2019-02-25
# Compare extract values to remainder of the study area for each predictor

# needed for nvim-R
if (basename(getwd()) == "src") {
  setwd("..
/")
  getwd()
}

# load the necessary packages

if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr
  ,dplyr
  ,tidyr
  ,magrittr
  ,tools
  ,rgdal
  ,raster
  ,corrplot
  ,sp
  ,rgeos
  ,ggplot2
  ,stringr
  ,RColorBrewer
  ,pbapply
```
# Appendix A. Source Code

```
, parallel
, randomForest
, ranger
, sf
, rasterVis
  #, circular
, scales
, tmap
, shinyjs
, tmapttools
, gridExtra
  #, ggslabel
  #, gghighlight
  , ggrepel
)

# this script is used to compare against a specific modeling session
  by its clean_time

# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

clean_time <- readRDS("./models/clean_time/clean_time.RDS")

# The following functions are a mirror of the extractBuffer script

# create a list of daily predictor variables (abbreviations)
vars <- c(
  "pbl.min",
  "pbl.max"
  #, "pr"
  #, "vs"
  #, "th"
  #, "tmmn"
  #, "tmmx"
  #, "rmin"
  #, "rmax"
  #, "sph"
  , "ndvi"
  , "evi"
  #, "pm2.5.mean"
  #, "pm2.5.max"
  #, "pm2.5.min"
  #, "pm2.5.sd"
```
Appendix A. Source Code

```
, "pm2.5.median"

# list of static predictors
static <- c(
  "fwy",
  "streets",
  "maj_art",
  "arterial",
  "railroad",
  "railyards",
  "elev"
  #, "population"
  #, "point"
  #, "nonpoint"
  #, "onroad"
  #, "nonroad"
)

# Define read functions for static and dynamic variables

# read functions for daily and static predictors
readVar <- function(var, date, study_area) {
  r <- raster(list.files("./data", pattern = paste0("^", var, "\.",
  date, "\.	if$"),
    full.names = T,
    recursive = T)) # resample(study_area,
  method = 'ngb')
}

readStatic <- function(var, study_area) {
  r <- raster(list.files("./data", pattern = paste0("^", var, "\.	if $"))
    full.names = T,
    recursive = T)) # resample(study_area,
  method = 'ngb')
}

# Focal Buffer functions

# focal buffers credited to Jackson Voelkel @ SUPR
# Buffer sequence
# buffers <- c(seq(500, 5000, 500), seq(6000, 10000, 1000))
# buffers <- c(1000, seq(0, 10000, 2500)) # 1km buffer isn’t the same
# as 0km!

# Read in buffers that matched their creation in extractBuffer.R
```
buffers <- readRDS(paste0("./data/buffers/", clean_time, ":buffers.RDS"))

# Run a single moving window
focalBufferRaster <- function(rast, bufferDist) {
  # skip wind direction and PBL during buffering
  if (!grepl("^th|pbl.*|pm2.5\s.*", names(rast))) {
    buf <- focal(rast,
      focalWeight(rast, bufferDist, type="circle")
    # assigns NA instead of 0 to edge effects
    , na.rm = T
    )
  }
  names(buf) <- paste0(names(rast), ":", bufferDist, "m")
  return(buf)
}

# Wind Vector Averaging Function
focalBufferWind <- function(stack, bufferDist) {
  # grab the wind direction and speed layers from stack
  wd <- raster::subset(stack, grep("^th\.*", names(stack), value = T))
  ws <- raster::subset(stack, grep("^vs\.*", names(stack), value = T))

  # determine u & v components
  u <- ws * sin(2*pi*wd/360)*(-1)
  v <- ws * cos(2*pi*wd/360)*(-1)

  # fun focal on each component
  mean_u <- focal(u, focalWeight(u, bufferDist, type = "circle"))
  mean_v <- focal(v, focalWeight(v, bufferDist, type = "circle"))

  # convert back to degrees
  mean wd <- (atan2(mean_u, mean_v) * 360/2/pi) + 180

  # assign wd buffer name to output
  names(mean wd) <- paste0(names(wd), ":", bufferDist, "m")
  return(mean wd)
}

# apply focalBufferRaster to entire stack
focalBufferStack <- function(stack, bufferDist, date) {
  s <- pblapply(1:nlayers(stack), function(x) {
    focalBufferRaster(rast = stack[[x]], bufferDist)
    #, cl = floor((detectCores() - 1)/2)
  }) %>% unlist %>% stack

  # wind direction buffering (requires wd and ws)
  if ("^th\.*" %in% names(stack) & "^vs\.*" %in% names(stack)) {
    wd <- focalBufferWind(stack, bufferDist)
    s %>% stack(wd)
  }

  return(s)

  # stackApply(stack, indices = 1:nlayers(stack), fun = function(x) {
  #    focalBufferRaster(rast = x, bufferDist = bufferDist)
  # })

  # function resamples and combines all GIS predictors into single
  # raster stack __________
  stackPredictors <- function(date, crs, study_area, static, vars) {
    # Study Area Extent Raster used for resampling...
    a <- readRDS(study_area) %>% spTransform(CRSobj = crs) %>% raster(
      res = sp_res, vals = 0)

    # read daily predictors
    r <- lapply(vars, function(x) {
      readVar(var = x, date = date, study_area = a)
    })
    #, cl = detectCores() - 1
    ) %>% stack()

    # read static predictors
    y <- pblapply(static, function(x) {
      readStatic(var = x, study_area = a)
    })
    #, cl = detectCores() - 1
    ) %>% raster::stack()

    # combine them
    s <- raster::stack(r, y)

    return(s)
    gc()
  }
# End mirror

# Run all of the focal buffers at once!

createBuffers <- function(stack, buffers, date) {
  # clear buf dir before writing new files?
  # try(system(paste0("rm ./data/buf/", date, "*", bufferDist, "m.tif")), silent=T)
  # file.remove(paste0("./data/buf/", date, "*", bufferDist, "m.tif"), showWarnings=F)
  # file.remove(paste0("./data/buf/", date, ".*", bufferDist, "m\.*tif"), showWarnings=F)
  # f <- list.files(path = "/.data/buf", pattern = paste0(date,".*tif"), full.names = T)
  # lapply(f, file.remove)
  pblapply(buffers
    ,function(x) {
      focalBufferStack(stack, bufferDist = x, date = date)
    }
    #,cl = ceiling((detectCores()-1)/2)
  )
}

# Raster processing functions

# Combine all predictors for a given day
extractPredictors <- function(date, crs, study_area, bb, vars, static, pm_df, scale_neph = scale_neph) {
  # Study Area Extent Raster used for resampling...
  a <- readRDS(study_area) %>% spTransform(CRSojb = crs) %>% raster(res = sp_res, vals = 0)
  # read PurpleAir data
  pa <- readRDS(pm_df) %>% spTransform(CRSojb = crs)
  # Final bounding box
  b <- readRDS(bb) %>% spTransform(CRSojb = crs) %>% raster(res = sp_res, vals = 0)
  # Create Raster Stack
  s <- stackPredictors(date = date,
    ,crs = crs
  )
,study_area = study_area
,vars = vars
,static = static)

# Buffering

s_buf <- createBuffers(s, buffers = buffers, date = date) %>%
  unlist %>% stack

# Add PBL and Neph data (not buffered)
pbl <- raster::subset(s, grep('^pbl\.*', names(s), value = T))
# neph <- raster::subset(s, grep('^pm2\.*', names(s), value = T))
neph <- raster::subset(s, grep('^pm2\.*median*', names(s),
  value = T))
s_buf %>% raster::stack(pbl, neph)

# strip date from daily predictor layer names
names(s_buf) <- gsub(x = names(s_buf), pattern = paste0("\"\.", date
  ), replacement = "")

# Rename 0m buffer to match non-buffered layers
names(s_buf) <- gsub(x = names(s_buf), pattern = paste0("\".0m\$"),
  replacement = "")

# Crop extent to original bounding box (10 times smaller than
  buffer boundary)
s_buf %>% raster::crop(b, snap = 'near') %>%
  resample(b, method = 'ngb')
s_buf %>% raster::mask(b)

# Extract PurpleAir data

pa <- pa[which(pa$date == gsub("\"\.", "\"--", date)),]

# Simple extract grabs only the intersecting cell value
s_extract <- raster::extract(s_buf, pa, method = 'bilinear', sp = T
  , na.rm = T)

# Convert to data.frame

# df <- raster::as.data.frame(s_extract) %>% na.omit()
df_network <- s_extract %>%
  st_as_sf() %>%
Appendix A. Source Code

```r
Appendix A. Source Code

279  st_drop_geometry() %>%
280  dplyr::select(-node) %>%
281  gather(-date, key = "variable", value = "value")
282  df_network$extent <- "network"
283
284  # Convert entire raster stack to a new dataframe
285  df_region <- as.data.frame(as.matrix(s_buf))
286  # Add the date variable
287  df_region$date <- as.Date(hyphenDate(date))
288  df_region %<>%
289  gather(-date, key = "variable", value = "value")
290  df_region$extent <- "region"
291
292  # Combine each tidy dataset
293  df <- bind_rows(df_network, df_region)
294
295  # Create a variable class vector
296  df$variable_group <- sub("\\\\d+m$", "", df$variable)
297  return(df)
298  gc()
299 }
300
301  # wrap our stacking function into a date sequence generator
302  
303  extractDailyPredictors <- function(start_date
304    ,end_date
305    ,crs
306    ,study_area
307    ,bb
308    ,vars # list of vars
309    ,static # list of static vars
310    ,pm_df
311    ,scale_neph
312 ) {
313     dates <- sequenceDates(start_date, end_date)
314     df <- Reduce(rbind, pblapply(dates, function(x) {
315       extractPredictors(date = x
316       ,study_area = study_area
317       ,bb = bb
318       ,crs = crs
319       ,vars = vars
320       ,static = static
321       ,pm_df = pm_df
322       ,scale_neph = scale_neph))
323     # parallelization takes place within each loop (faster)
324   
325 
326 }
327```
# Running our function on multiple days

```r
if (!file.exists(paste0("./data/df_eda_predictors/", clean_time, "_df_eda_predictors.RDS"))) {

  df <- extractDailyPredictors(start_date = start_date, end_date = end_date, #,end_date =
                              start_date # for quick profiling,
                              crs = epsg_26910,
                              study_area = study_area,
                              ,bb = bb,
                              ,vars = vars,
                              ,static =
                              pm_df = pm_
                              df
                              ,scale_neph =
  print(paste0("Creating ", clean_time, "df_eda_predictors.RDS"))
  saveRDS(df, paste0("./data/df_eda_predictors/", clean_time, "_df_eda_predictors.RDS"))
} else {
  df <- readRDS(paste0("./data/df_eda_predictors/", clean_time, "_df_eda_predictors.RDS"))
}
```

## Explore spatial heterogeneity of predictors

```r
# Create a violin plot comparing the distribution of each predictor across the network and region
pblapply(unique(df$variable), function(x) {
  p <- df %>% dplyr::filter(variable == x) %>%
       ggplot(aes(x = extent, y = value, fill = extent)) +
       geom_violin() +
```

Appendix A. Source Code

```r
labs(y = paste0(x)) +
theme_minimal() +
theme(axis.text.x = element_blank(),
axis.title.x = element_blank())
ggsave(filename = paste0("./figures/violin_plots/", x, "_violin_plot.png"),
plot = p,
width = 8,
height = 6,
units = "in",
dpi = 300)
rm(p)
gc()
}

# Create a violin plot comparing the distribution of each predictor group across the network and region
pblapply(unique(df$variable_group), function(x) {
  p <- df %>% dplyr::filter(variable_group == x) %>%
      mutate(variable = factor(variable, levels = c(paste0(x),
                                   paste0(x, ".1000m"),
                                   paste0(x, ".2500m"),
                                   paste0(x, ".5000m"),
                                   paste0(x, ".7500m"),
                                   paste0(x, ".10000m")) ) %>%
      ggplot(aes(x = variable, y = value, fill = extent)) +
    geom_violin() +
    labs(y = paste0(x)) +
    theme_minimal() +
    theme(axis.title.x = element_blank())
ggsave(filename = paste0("./figures/violin_plots/", x, "_group_violin_plot.png"),
plot = p,
width = 8,
height = 6,
units = "in",
dpi = 300)
rm(p)
gc()
})
```
c1 = detectCores() - 1

Source/functions.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018-09-10
# functions used to download, process, and visualize MAIAC AOD data

# Miscellaneous

getHostName <− function() {
  hostname <− Sys.info()
  return(hostname["nodename"])
}

# Set custom raster tempdir for XPS or Tower (hardcode new hostname for new machine)
setRasterTempDir <− function() {
  if (getHostName() == "XPS-15-9570") {
    dir <− "/tmp/
  } else if (getHostName() == "Precision-Tower-5810") {
    dir <− "/media/porlando/My Book/
  } else if (getHostName() == "hecate.rc.pdx.edu") {
    dir <− "/workspace/porlando/tmp"
  } else if (grepl("himem.*|phi.*|compute.*cluster|login1.*cluster|login2.*cluster", getHostName()))
    dir <− "/scratch/porlando/tmp"
  else {
    print(paste0(getHostName(), " is not a valid hostname! Modify setRasterTempDir() to accommodate new host."))
  }
  if (!dir.exists(dir)) {
    dir.create(dir)
  }
  rasterOptions(tmpdir = dir)
  print(paste0("Setting raster tempdir ", dir, " for ", getHostName()))
}

# Emulate ggplot default color palette
gg_color_hue <− function(n) {
  hues = seq(15, 375, length = n + 1)
  hcl(h = hues, l = 65, c = 100)[1:n]
```r
# capture object name as a character
obj2char <- function(object) {
  deparse(substitute(object))
}

# remove geometry from sf object, convert to dataframe with only the elements/attributes
st_drop_geometry <- function(x) {
  if (inherits(x,"sf")) {
    x <- st_set_geometry(x, NULL)
    class(x) <- 'data.frame'
  }
  return(x)
}

# convert sf c to x, y coordinates
sfc_as_cols <- function(x, names = c("x","y")) {
  stopifnot(inherits(x,"sf") & & inherits(sf::st_geometry(x),"sfc::POINT"))
  ret <- sf::st_coordinates(x)
  ret <- tibble::as_tibble(ret)
  stopifnot(length(names) == ncol(ret))
  x <- x[, !names(x) %in% names]
  ret <- setNames(ret ,names)
  dplyr::bind_cols(x,ret)
}

print_clean_time <- function() {
  str_replace_all(lubridate::now(), " |": " _")
}

# Grab p-value from model object
lmp <- function (modelobject) {
  if (class(modelobject) != "lm") stop("Not an object of class 'lm'")
  f <- summary(modelobject)$fstatistic
  p <- pf(f[1],f[2],f[3],lower.tail=F)
  attributes(p) <- NULL
  return(p)
}

# plot density in ggplot
get_density <- function(x, y, n = 100) {
  dens <- MASS::kde2d(x = x, y = y, n = n)
```

Appendix A. Source Code

142
ix ← findInterval(x, dens$x)
iy ← findInterval(y, dens$y)
ii ← cbind(ix, iy)
return(dens$z[ii])
}

# creating a custom not in function
%! in' ← function(x, y) { !( %in% (x, y)) }

sequenceDates ← function(start_date, end_date, interval = "day") {
dates ← seq.Date(as.Date(start_date), as.Date(end_date), by =
    interval) %>%
    as.character() %>%
gsub(pattern = "\-", replacement = "\.")
return(dates)
}

# Hyphenate dates that are seperated by "."
hyphenDate ← function(date) {
gsub("\.", "\-", date)
}

# Season generator function!
getSeason ← function(DATES) {
    WS ← as.Date("2012-12-15", format = "%Y-%m-%d") # Winter Solstice
    SE ← as.Date("2012-3-15", format = "%Y-%m-%d")    # Spring Equinox
    SS ← as.Date("2012-6-15", format = "%Y-%m-%d")    # Summer Solstice
    FE ← as.Date("2012-9-15", format = "%Y-%m-%d")    # Fall Equinox

    # Convert dates from any year to 2012 dates
d ← as.Date(strftime(DATES, format="2012-%m-%d"))
    ifelse (d >= WS | d < SE, 1, # Winter
        ifelse (d >= SE & d < SS, 2, # Spring
            ifelse (d >= SS & d < FE, 3, 4)) # Summer and Fall
    }
}

# downloading script

# trying to download MAIAC data from datapool via R
getAOD ← function(user = 'porlando',
    pw = scan("./batteries.dppss", what = ""),
    dir = 'MAIAC',
    product = 'MCD19A2')
Appendix A. Source Code

```
, date = '2018.08.28'
, tile = 'h09v04'
, collection = "006") {

  # parse date and julian date for url building
  Date <- lubridate::ymd(date)

  julianDate <- paste0(lubridate::year(Date)
                         , lubridate::yday(Date))

  # check if file exists
  folder <- paste0("./MAIAC/e4ftl01.cr.usgs.gov/MOTA/MCD19A2.006/",
                   date)

  # download file if it doesn't exist yet
  if(length(dir(folder, all.files = TRUE, pattern = "*.hdf")) == 0) {
    # extracts hdf files for a given tile across all observation times
    pattern <- paste0(product, ".A", julianDate, ".", tile, ".",
                       collection, "*.hdf")

    # base url for wget
    url <- paste0("https://e4ftl01.cr.usgs.gov/MOTA/", product, ".",
                  collection, "/", date, "/")

    # call wget from command line
    system(paste0("wget -r -A", pattern, " -L --user=", user
                  , " --password=", pw
                  , " --directory-prefix=", dir
                  , " --load-cookies ~/.cookies --save-cookies ~/.cookies
                  cookies --no-parent ", url))
  }
}

# create a list of correctly formatted dates
getDates <- function(start_date = "2017-07-01", end_date = Sys.Date()
               , by = "days") {
  days <- seq.Date(from = as.Date(start_date), to = as.Date(end_date)
                   , by = by)
  stringr::str_replace_all(days, "-", ".")
}

# delete all hdf files and subfolders CAREFUL!
purgeAOD <- function(parent_dir = "/MAIAC") {
  system(paste0("rm -rf ", parent_dir, "*/"))
```

170 
171 # processing script
172 # process hdf and create raster
173 processAOD <- function(file_path, study_area = portl_buf) {
174  cat("\n")
175  # extract date from file_path
176  date <- stringr::str_extract(file_path, '([0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9].[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9])')
177  date_clean <- stringr::str_replace_all(date, "\.", "-")
178  # read in hdf
179  sds <- get_subdatasets(file_path)
180  # Grab AOD layers
181  Optical_Depth_047 = sds[grepl("grid1km:Optical_Depth_047", sds)] %>% readGDAL %>% stack
182  Optical_Depth_055 = sds[grepl("grid1km:Optical_Depth_055", sds)] %>% readGDAL %>% stack
183  # More variables...
184  # AOD_Uncertainty = sds[grepl("grid1km:AOD_Uncertainty", sds)] %>% readGDAL %>% stack
185  # FineModeFraction = sds[grepl("grid1km:FineModeFraction", sds)] %>% stack
186  # Column_WV = sds[grepl("grid1km:Column_WV", sds)] %>% stack
187  # AOD_QA = sds[grepl("grid1km:AOD_QA", sds)] %>% readGDAL %>% stack
188  # AOD_MODEL = sds[grepl("grid1km:AOD_MODEL", sds)] %>% readGDAL %>% stack
189  # Injection_Height = sds[grepl("grid1km:Injection_Height", sds)] %>% readGDAL %>% stack
190  # Note that some variables such as 'RelAZ' are on a different resolution and must be disaggregated to match –
191  # cosSZA = sds[grepl("grid5km:cosSZA", sds)] %>% readGDAL %>% stack %>% disaggregate(5)
192  # cosVZA = sds[grepl("grid5km:cosVZA", sds)] %>% readGDAL %>% stack %>% disaggregate(5)
193  # RelAZ = sds[grepl("grid5km:RelAZ", sds)] %>% readGDAL %>% stack %>% disaggregate(5)
194  # Scattering_Angle = sds[grepl("grid5km:Scattering_Angle", sds)] %>% readGDAL %>% stack %>% disaggregate(5)
195  # Glint_Angle = sds[grepl("grid5km:Glint_Angle", sds)] %>% readGDAL %>% stack %>% disaggregate(5)
# aggregate each satellite overpass into a single band (mean, max)
Optical_Depth_047_mean = stackApply(Optical_Depth_047, indices = rep(1, nlayers(Optical_Depth_047)), fun = "mean", na.rm = TRUE)

Optical_Depth_047_max = stackApply(Optical_Depth_047, indices = rep(1, nlayers(Optical_Depth_047)), fun = "max", na.rm = TRUE)

Optical_Depth_055_mean = stackApply(Optical_Depth_055, indices = rep(1, nlayers(Optical_Depth_055)), fun = "mean", na.rm = TRUE)

Optical_Depth_055_max = stackApply(Optical_Depth_055, indices = rep(1, nlayers(Optical_Depth_055)), fun = "max", na.rm = TRUE)

# Combine to multi-band raster with ‘row’ and ‘col’ and AOD values
r = stack(Optical_Depth_047_mean, Optical_Depth_047_max, Optical_Depth_055_mean, Optical_Depth_055_max)

names(r) = c("Optical_Depth_047_mean", "Optical_Depth_047_max", "Optical_Depth_055_mean", "Optical_Depth_055_max")

# transform raster to pdx CRS
r <- projectRaster(r, crs = wgs_84)

# transform our study area
study_area <- spTransform(study_area, CRSobj = wgs_84)

if (identical(crs(study_area), crs(r))){
    # extract raster cells that intersect with our study area
    # s <- trim(r) # what does this do?
    # x <- extract(r, study_area) # what does this do?

    m <- mask(r, study_area)
    cr <- raster::crop(m, study_area)

    # date sequence (needed for rts)
Appendix A. Source Code

247 dates ← lubridate::ymd(rep(date, nlayers(cr)))
248
249 # convert to RasterBrickTS (breaks filterAOD)
250 # cr %>% rts(dates)
251
252 # add dates to Z values instead of using RTS (for now)
253 cr %>% setZ(dates)
254
255 # od47 ← cr[["Optical_Depth_047"]]
256 # leaflet(portl_buf) %>
257 # addTiles() %>
258 # addRasterImage(od47) %>
259 # addPolygons()
260
261 # dynamically assigning output folder and file name breaks in parallel/snow argh!
262 # location ← deparse(match.call())$study_area
263 location ← "portl_buf"
264 # determine if the data intersects
265 if (!is.null(cr[[1]])) {
266   # print(paste0("Returning overlapping AOD data for ", date))
267   writeRaster(cr, filename = paste0("./output/aod/", location,"/
268   , date_clean, ".", location, ".aod.tif")
269   ,format = "GTiff"
270   ,overwrite = TRUE)
271   return(cr)
272 }
273 else {
274   # print(paste0("AOD data does not overlap with study area on ",
275   date))
276   return(NULL)
277 }
278
279 }
280
281 # aggregate raster stack by week
282 # weekly_aod ← function(brick) {
283 #
284 # }
285
286 # retrieve AOD grid cells within study area
287
288 filterAOD ← function(aod_list) {
289   r ← unlist(aod_list)
# Appendix A. Source Code

```r
# works for rasterBrickTS but not rasterBrick
df <- as.data.frame(r@raster@data@values) 

df <- as.data.frame(r@data@values) %>% 
dplyr::filter(!is.na(Optical_Depth_047_mean) | 
  !is.na(Optical_Depth_047_max) | 
  !is.na(Optical_Depth_055_mean) | 
  !is.na(Optical_Depth_055_max))
return(df)
```

```r
# get modis tiles for a given shapefile’s extent (broken)
getTiles <- function(destfile = "./tiles/modis_sin.kmz") {
  if(!file.exists(destfile)) {
    url <- paste0("https://modis.ornl.gov/files/", basename(destfile))
    download.file(url, destfile = destfile)
  } else {
    tiles <- readOGR(destfile, "Features")
  }
  return(tiles)
}
```

```r
# EDA script

useable_days <- function(input = nrow_day, percent = 75) {
  p_days <- nrow_day %>% dplyr::filter(p_coverage >= percent) %>% 
nrow() / nrow(nrow_day) * 100
  return(p_days)
}
```

```r
# quickly inspect/visualize the aod coverage for each day
rPlot <- function(aod_layer) {
  plot(aod_layer)
  Sys.sleep(1)
}
```

```r
# Pan’s correlation matrix function
cor.matrix<-function(x, data=NA, cor.method="pearson", add1to1=F,...) {
  # panel.hist function adds the histogram
  options(warning=F)
  panel.hist <- function(x)
  {
    usr <- par("usr")
    on.exit(par(usr))
  }
```
```r
par(usr = c(usr[1:2], 0, 1.5))
h <- hist(x, plot = FALSE, breaks = 30)
breaks <- h$breaks
nB <- length(breaks)
y <- h$counts
y <- y/max(y)
rect(breaks[-nB], 0, breaks[-1], y, col = "lightblue")
box()
}
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor) {
  usr <- par("usr")
on.exit(par(usr))
par(usr = c(0, 1, 0, 1))
r <- cor(x, y, method = "pearson")
txt <- format(c(r, 0.123456789), digits = digits)[1]
txt <- paste(prefix, txt, sep = "")
if (missing(cex.cor)) cex <- 0.8/strwidth(txt)
text(0.5, 0.5, txt, cex = cex * abs(r))
}
panel.smooth2 <- function(x, y, bg = NA, pch = par("pch"), cex = 1,
col.smooth = "red", span = 2/3, iter = 3, plot1to1 = add1to1) {
  points(x, y, pch = pch, cex = cex, 
  ok <- is.finite(x) & is.finite(y)
  if (any(ok)) {
    lines(stats::lowess(x[ok], y[ok], f = span, iter = iter), col =
    col.smooth)
  }
  if (plot1to1) {
    mx <- min(c(x, y), na.rm = T)
    mx <- max(c(x, y), na.rm = T)
    new.x <- seq(min(mx, length.out = 10)
    points(new.x, new.x, type = "l", col = "blue", lwd = 2)
  }
  if (class(x) == "formula") {
    x <- model.frame(x, data = data)
  }
  pairs(x, upper.panel = panel.smooth2, lower.panel = panel.cor, diag.panel =
    panel.hist,
    cex.labels = 1, font.labels = 2)
  options(warning = T)
}
```

Source/ggRegression.R
# Define Test Regression Plot

def scaleFUN(x):
    return format(x, "%.2f")

# This plot is used to compare test observations and predictions from
# ranger/randomForest models
# Provide the names of x and y vectors as strings

def testRegression(df, x, y, outdir, reduced=False, clean_time=False):
    # Create lm model objects
    mod = lm(df[y] ~ df[x])
    mod_sum = summary(mod)

    # Compute RMSE on test data versus predictions
    rmse = sqrt(mean((df[y] - df[x])^2))

    # Create color palette to display point density
    d = grDevices:densCols(df[x], df[y], colramp=colorRampPalette(rev(rainbow(10, end=4/6))))

    # Build our regression plot
    p = ggplot(df, aes_string(x=x, y=y)) +
        ylab("Predicted PM2.5 (ug/m3)") +
        xlab("Observed PM2.5 (ug/m3)") +
        ylab(expression(~Observed~ PM[2.5]~mu~g~m^-3)) +
        xlab(expression(~Predicted~ PM[2.5]~mu~g~m^-3)) +
        annotate("text", label = paste("R2 = ", round(modr$r.squared, 4)), x = 25, y = max(c(cvr$x, cvr$y)) - 10) +
        annotate("text", label = paste("p-value = ", round(modr$coefficients[2], 2)), x = 25, y = max(c(cvr$x, cvr$y)) - 20) +
        annotate("text", label = paste("Slope = ", round(modr$coefficients[2], 2)), x = 25, y = max(c(cvr$x, cvr$y)) - 20) +
        annotate("text", label = paste("Intercept = ", round(modr$coefficients[1], 2)), x = 25, y = max(c(cvr$x, cvr$y)) - 30) +
        geom_point(col = d, alpha=0.5) +
        labs(subtitle = paste("Adj R2 = ", signif(mod_sum$adj.r.squared, 4),
                   "Intercept = ", signif(mod_sum$coef[1], 2),
                   "Slope = ", signif(mod_sum$coef[2], 2),
                   "p-value = ", signif(mod$r.squared, 4)))
"Slope =", \texttt{signif(mod_sum$coef[[2]], 2)}, \\
"p =", \texttt{signif(as.numeric(broom::glance(mod))$p.value), 3)}, \\
"RMSE =", \texttt{signif(rmse, 3))} + \\
\texttt{geom_smooth(se=F, method = "lm", col = "firebrick").} + \\
\texttt{geom_abline(intercept = mod$coef[[1]]}} \\
\texttt{,slope = mod$coef[[2]]} \\
\texttt{,color = "firebrick").} + \\
\texttt{geom_abline(intercept = 0}} \\
\texttt{,slope = 1} \\
\texttt{,color = "black"} \\
\texttt{,linetype = "dashed"} \\
\texttt{)} + \\
\texttt{theme_bw() +} \\
\texttt{theme(aspect.ratio = 1}} \\
\texttt{,plot.title = element_text(hjust = 0.5, size=12, face = "bold").} \\
\texttt{,plot.subtitle = element_text(hjust = 0.5, size=12, face = "bold")} \\
\texttt{,axis.text = element_text(size=rel(1.1), face = "bold",}} \\
\texttt{colour = "black")} \\
\texttt{,axis.title = element_text(size=15)} \\
\texttt{)} \\
\texttt{# Assign file name for full/reduced RF model} \\
\texttt{if (reduced) {} \\
\texttt{  outfile <- paste0(outdir, clean_time, "-obs-pred-lm-reduced.png")} \\
\texttt{} else { \\
\texttt{  outfile <- paste0(outdir, clean_time, "-obs-pred-lm.png")} \\
\texttt{}} \\
\texttt{print(paste0("Saving plot to ", outfile))} \\
\texttt{ggsave(plot = p}} \\
\texttt{  ,filename = outfile} \\
\texttt{  ,width = 8} \\
\texttt{  ,height = 8} \\
\texttt{  ,units = "in"} \\
\texttt{)} \\
\texttt{return(p)} \\
\texttt{}} \\
\texttt{# Same function as above, but adds a date to the title} \\
\texttt{dailyRegression <- function(df, x, y, outdir, reduced = F, date) { \\
\texttt{# Create lm model objects}
Appendix A. Source Code

mod <- lm(df[[y]] ~ df[[x]])
mod_sum <- summary(mod)

# Compute RMSE on test data versus predictions
rmse <- sqrt(mean((df[[y]] - df[[x]])^2))

# Create color palette to display point density
d = grDevices::densCols(df[[x]], df[[y]], colramp = colorRampPalette(rev(rainbow(10, end = 4/6))))

# Build our regression plot
p <- ggplot(df, aes_string(x = x, y = y)) +
  # ylab("Predicted PM2.5 (ug/m3)") +
  # xlab("Observed PM2.5 (ug/m3)") +
  xlab(expression(-{\text{Observed}} PM[2.5] - \mu_{g \cdot m^{-3}})) +
  ylab(expression(-{\text{Predicted}} PM[2.5] - \mu_{g \cdot m^{-3}})) +
  # annotate("text", label = paste("R2 =", round(mod$r.squared, 4), x = 25, y = max(c(cvr$x, cvr$y)) - 10) +
  # annotate("text", label = paste("p-value =", round(mod$coefficients[2], 2), x = 25, y = max(c(cvr$x, cvr$y)) - 20) +
  # annotate("text", label = paste("Slope =", round(mod$coefficients[2], 2), x = 25, y = max(c(cvr$x, cvr$y)) - 20) +
  # annotate("text", label = paste("Intercept =", round(mod$coefficients[1], 2), x = 25, y = max(c(cvr$x, cvr$y)) - 30) +
  # geom_point(col = d, alpha=0.5) +
  # geom_smooth(se=F, method = "lm", col = "firebrick") +
  geom_abline(intercept = mod$coefficients[[1]],
              slope = mod$coefficients[[2]],
              color = "firebrick") +
  geom_abline(intercept = 0, slope = 1, color = "black", linetype = "dashed") +
  geom_point(alpha=0.5) +
  geom_text_repel(point.padding = 0.3, aes(label = sprintf("%.2f", signif(df[[y]] - df[[x]], 2)))) +
  coord_fixed() +
  scale_x_continuous(labels = scaleFUN, limits = c(min(c(df[[x]], df[[y]])), max(c(df[[x]], df[[y]]))) +
  scale_y_continuous(labels = scaleFUN, limits = c(min(c(df[[x]], df[[y]])), max(c(df[[x]], df[[y]]))) +
  # scale_x_continuous(limits = c(0, max(c(cvr$x, cvr$y)))) +
  # scale_y_continuous(limits = c(0, max(c(cvr$x, cvr$y)))) +
  labs(caption = paste0(date),
       subtitle = paste("Adj R2 = ", sprintf("%.4f", signif(mod_sum$adj.r.squared, 4)))),
"Intercept =", sprintf("%.2f", signif(mod_sum$coef[[1]], 2)),
" Slope =", sprintf("%.2f", signif(mod_sum$coef[[2]], 2)),
" p =", sprintf("%.3f", signif(as.numeric(broom::glance(mod)$p.value), 3)),
" RMSE =", sprintf("%.3f", signif(rmse, 3))
}) +
theme_bw() +
theme(aspect.ratio = 1,
  plot.title = element_text(hjust = 0.5, size=12, face = "bold"),
  plot.subtitle = element_text(hjust = 0.5, size=12, face = "bold"),
  axis.text = element_text(size=rel(1.1), face = "bold", colour = "black"),
  axis.title = element_text(size=15))

# Assign file name for full/reduced RF model
if (reduced) {
  outfile <- paste0(outdir, str_replace_all(date, ".","-"), "-obs-pred-lm-reduced.png")
} else {
  outfile <- paste0(outdir, str_replace_all(date, ".","-"), "-obs-pred-lm.png")
}
print(paste0("Saving plot to ", outfile))
ggsave(plot = p,
  filename = outfile,
  width = 8,
  height = 8,
  units = "in")
return(p)

# Same function but with different labels ---------------
nephRegression <- function(df, x, y, outdir, reduced = F, clean_time)
{

  # Create lm model objects
  mod <- lm(df[[y]] ~ df[[x]])
  mod_sum <- summary(mod)

  # Compute RMSE on test data versus predictions
  rmse <- sqrt(mean((df[[y]] - df[[x]])^2)) # 1:1 RMSE
rmse <- glance(mod)$sigma  # Model RMSE
# Create color palette to display point density
d = grDevices::densCols(df[[x]], df[[y]], colramp =
colorRampPalette(rev(rainbow(10, end = 4/6))))

# Build our regression plot
p <- ggplot(df, aes_string(x = x, y = y)) +
  # ylab("Predicted PM2.5 (ug/m3)") +
  # xlab("Measured PM2.5 (ug/m3)") +
  xlab(expression(~PurpleAir~ PM[2.5] ~mu*g*m^-3)) +
  ylab(expression(~Nephelometer~ PM[2.5] ~mu*g*m^-3)) +
  # annotate("text", label = paste("R2 =", round(mod$r.squared, 4)
  , x = 25, y = max(c(cvr$x, cvr$y))−10) +
  # annotate("text", label = paste("p-value =", round(mod$coefficients[2], 2)), x = 25, y = max(c(cvr$x, cvr$y))−20) +
  # annotate("text", label = paste("Slope =", round(mod$coefficients[2], 2)), x = 25, y = max(c(cvr$x, cvr$y))−20) +
  # annotate("text", label = paste("Intercept =", round(mod$coefficients[1], 2)), x = 25, y = max(c(cvr$x, cvr$y))−30) +
  geom_point(col = d, alpha=0.5) +
  #coord_fixed() +
  scale_x_continuous(limits = c(min(c(df[[x]], df[[y]])), max(c(df[[
x]], df[[y]])))) +
  scale_y_continuous(limits = c(min(c(df[[x]], df[[y]])), max(c(df[[
x]], df[[y]])))) +
  # scale_x_continuous(limits = c(0, max(c(cvr$x, cvr$y)))) +
  # scale_y_continuous(limits = c(0, max(c(cvr$x, cvr$y)))) +
  labs(subtitle = paste("R2 =", signif(mod_sum$adj.r.squared, 4),
        " Intercept =", signif(mod_sum$coef[[1]], 2),
        " Slope =", signif(mod_sum$coef[[2]], 2),
        " p =", signif(as.numeric(broom::glance(mod
))$p.value), 3),
        " RMSE =", signif(rmse, 3)) +
  # geom_smooth(se=F, method = "lm", col = "firebrick") +
  geom_abline(intercept = mod$coef[[1]]
  ,slope = mod$coef[[2]]
  ,color = "firebrick") +
  geom_abline(intercept = 0
  ,slope = 1
  ,color = "black"
  ,linetype = "dashed"
) +
theme_bw() +
theme(aspect.ratio = 1
  ,plot.title = element_text(hjust = 0.5, size=12, face = "
bold")
  ,plot.subtitle = element_text(hjust = 0.5, size=12, face = "bold")
Appendix A. Source Code

```
, axis.text = element_text(size = rel(1.1), face = "bold",
        colour = "black")
, axis.title = element_text(size = 15)
)

outfile <- paste0(outdir, clean_time, "−obs−pred−lm.png")

# Assign file name for full/reduced RF model
if (reduced) {
  outfile <- paste0(outdir, clean_time, "_obs_pred−lm−reduced.png")
} else {
  outfile <- paste0(outdir, clean_time, "−obs−pred−lm.png")
}

print(paste0("Saving plot to ", outfile))
ggsave(plot = p
        , filename = outfile
        , width = 8
        , height = 8
        , units = "in"
)
return(p)
```

Source/homogeneousPM.R

```
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018−09−17
# extract regionally homogenous neph PM (+/− 10% of network mean)

# set up environment

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(readr
        , readxl
        , xlsx
        , ggplot2
        , raster
        , plyr
```
Appendix A. Source Code

```
# define global variables

source("./src/ggRegression.R")
source("./src/init.R")

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
```
Sys.timezone()
Sys.setenv(TZ="GMT")
Sys.timezone()

## connecting to local db
host <- 'pgsql120.rc.pdx.edu'
db <- 'canopycontinuum'
user <- 'porlando'
port <- 5433
pw <- scan("./batteries.pgpss", what = "") # in parent dir

# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres()
  ,dbname = db
  ,host = 'pgsql102.rc.pdx.edu' # not sure why object host isn't working...
  ,port = port
  ,password = pw
  ,user = user)

# create unanimous time resolution for all data
time_resolution <- "1 sec"

# time zone applied to all data
time_zone <- "GMT"

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs "

# Oregon North NAD83 HARN meters
epsg_2838 <- "+proj=lcc +lat_1=46 +lat_2=44.33333333333334 +lat_0=43.66666666666666 +lon_0=-120.5 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs "

epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs "

# Function definition

# creating a custom not-in function
"%! in%' <- function(x, y){"%in%"(x,y)}

# plot density in ggplot
generic_density <- function(x, y, n = 100) {
dens <- MASS::kde2d(x = x, y = y, n = n)
ix <- findInterval(x, dens$x)

iy <- findInterval(y, dens$y)

ii <- cbind(ix, iy)

return(dens$z[ii])

set.seed(1)

# data processing

# start_date <- "2017-07-01"
# end_date <- "2018-09-30"
# inherit start/end date from init script in modeling project
# this should work once I update my pdx table
# pa <- st_read(dsn = con
# , geom_column = "geom" # deprecated!
# , EWKB = TRUE
# , query = paste0("SELECT DISTINCT * FROM pdx WHERE
# created_at >= ", lubridate::ymd(start_date) + hours(8)
# , ", lubridate::ymd(end_date) + hours(8),
# ");")
#)
# saveRDS(pa, ".data/frm/purpleAir5min.RDS")
# pa <- readRDS(".data/frm/purpleAir5min.RDS")

# hourly group by with sql

pa <- st_read(dsn = con
#, geom_column = "geom" # deprecated!
# , EWKB = TRUE
#, query = paste0("SELECT date(a.created_at) , date_part('hour', a.created_at) as hour , a.id , a.sensor , a.label , avg(a.pm2_5_atm) pm2_5_atm , a.geom
# created_at >= "
# , lubridate::ymd(start_date) + hours(8)
# , lubridate::ymd(end_date) + hours(8), "
# ) a
# FROM (SELECT DISTINCT * FROM pdx WHERE
# created_at >= "
# , lubridate::ymd(start_date) + hours(8)
# , lubridate::ymd(end_date) + hours(8), "
# ) a
# GROUP BY a.id , a.sensor , a.label , a.
# geom, date(a.created_at) , date_part('hour', a.created_at);")
)
pa$created_at <- paste(pa$date, pa$hour)
pa$created_at <- ymd_h(pa$created_at)
pa$created_at <- pa$created_at - hours(8)
naes(pa)[naes(pa) == "created_at"] <- "datetime"

# 15 minute average PurpleAir

# hourly group by with sql
pa <- st_read(dsn = con
   #.geom_column = "geom" # depreceated!
   #.EWKB = TRUE
   #.query = paste0( "SELECT date_trunc('hour', a.created_at + date_part('minute', a.created_at)::int / 15 * interval '15 min at atm) pm2_5_atm, a.geom
   (SELECT DISTINCT * FROM pdx WHERE
      lubridate::ymd(start_date) + hours(8)
      lubridate::ymd(end_date) + hours(8), "
    ) a
      GROUP BY a.id, a.sensor, a.label, a.geom
      ,date_trunc('hour', a.created_at) +
      date_part('minute', a.created_at)::int / 15 * interval '15 min';")
)

### pa$created_at <- paste(pa$date, pa$hour, pa$fifteen)
pa$created_at <- ymd_hms(pa$created_at)
pa$created_at <- pa$created_at - hours(8)
naes(pa)[naes(pa) == "created_at"] <- "datetime"

# End 15-minute read PA

# lookup table for sensor geom
pa_points <- st_read(dsn = con
   #.geom_column = "geom"
   #.EWKB = TRUE
   #.query = "SELECT DISTINCT * FROM pdx_local_slope;"
   #.query = "SELECT DISTINCT * FROM pdx_slope;"
   # compare PurpleAir to DEQ during regional PM event (wildfire)
Appendix A. Source Code

```
,query = "SELECT DISTINCT id, label, geom FROM pdx GROUP BY id, label, geom;"

# define projection
st_crs(pa) <- epsg_26910
# st_crs(pa_hourly) <- epsg_26910
st_crs(pa_points) <- epsg_26910

# Relative standard deviation calc

# Reading in 5-minute neph data
# neph <- readRDS("./data/deq/neph5min.RDS")
frm <- readRDS("./data/deq/neph5min.RDS")
# frm <- readRDS("./data/deq/neph1hour.RDS")

rsd <- function(sd, mean) {
  sd/mean
}

# 15-minute only
frm$datetime <- floor_date(frm$datetime, "15 minutes")

frm %<>% dplyr::select(-date) %>% st_drop_geometry() %>%
  group_by(datetime, site_id) %>% summarise(pm2_5_ref = mean(pm2_5_ref))

# 15-minute only

# aggregate by 15 minutes (or 1 hour)
frm_mean <- frm %>% group_by(datetime) %>% summarise(pm2_5_mean =
  mean(pm2_5_ref),
  n_mean = n(),
  pm2_5_rsd = rsd(sd(pm2_5_ref), mean(pm2_5_ref)))

# Reduce minimum sites to 4 (2017 data missing 1 site - Cully)
rsd_threshold <- 0.10
# n_min <- 4 # 1-hour
n_min <- 5 # 15-min
time_res <- "15-min"

# For 1-hour data
# frm_mean %>% filter(pm2_5_rsd <= 0.10) %>% filter(n_mean >= n_min &
# n_mean <= 6) %>% st_drop_geometry() %>% nrow()
# frm_mean %>% filter(pm2_5_rsd <= 0.05) %>% filter(n_mean >= n_min &
# n_mean <= 6) %>% st_drop_geometry()
```
# For 15-minute data
frm_mean %>% filter(pm2_5_rsd <= rsd_threshold) %>% filter(n_mean >= n_min & n_mean <= 6) %>% st_drop_geometry() %>% nrow()

frm_mean %>% filter(pm2_5_rsd <= rsd_threshold) %>% filter(n_mean >= n_min & n_mean <= 6) %>% st_drop_geometry()

# Calc percent difference
# frm %>% inner_join(frm_mean)
# summary(frm)
#
# # DEQ 24-h datetime is introducing duplicates around midnight?
# frm$hour <- hour(frm$datetime)
# frm %>% filter(hour != 0)
# frm %>% filter(n_mean >= 4 & n_mean <= 6)
#
# # percent difference function
# pct_diff <- function(v1, v2) {
# abs(v1-v2)/(v1+v2)/2
# }
# # Find hours with low variation among 5–6 nephelometers
# frm$pct_diff <- pct_diff(frm$pm2_5_ref, frm$pm2_5_mean)
#
# frm_homo <- frm %>% filter(pct_diff <= 0.10) %>% arrange(datetime)
#
# # Join mean pct diff with mean pm by datetime
# frm_mean_homo <- frm_homo %>% group_by(datetime) %>%
# summarise(pct_diff_mean = mean(pct_diff), n_pct = n()) %>% filter(n_pct >= 5) %>% left_join(frm_mean)
#
# # Join mean pct diff with mean pm by datetime and by site id!
# frm_site_homo <- frm_homo %>% group_by(datetime, stn_id) %>%
# summarise(pct_diff_mean = mean(pct_diff), n_pct = n()) %>% filter(n_pct >= 5) %>% left_join(frm)
#
# frm %>% st_as_sf(coords = c("lon","lat"), crs = wgs_84)
# frm <- frm_mean_homo # mean % difference of mean
# frm <- frm_mean # RSD

# Join DEQ with PA
#
# matching the sel file with the 5stn id
# frm$ site_id <- ifelse(frm$site_id == "PM2 5L_Est", "Portland SE Lafayette", frm$site_id)
str(frm)
str(pa)
# str(pa_hourly)

# check the time zone of pa data
attr(pa$datetime, "tzone")
# attr(pa_hourly$datetime, "tzone")
attr(frm$datetime, "tzone")

# Convert DEQ data to UTC!
# frm$datetime <- with_tz(frm$datetime, tzone = "UTC")
# attr(frm$datetime, "tzone")

# DEQ datetime isn’t in US/Pacific, it’s UTC–8 (they don’t observe DST)
# Convert tz without changing datetime
# pa$datetime <- force_tz(pa$datetime, "US/Pacific")
# pa$datetime <- force_tz(pa$datetime, "GMT")
attr(pa$datetime, "tzone")

# pa_hourly$datetime <- force_tz(pa_hourly$datetime, "US/Pacific")
# attr(pa_hourly$datetime, "tzone")

# Join mean neph data with PA

df <- inner_join(pa, frm)
# df <- inner_join(pa_hourly, frm)
df$year <- lubridate::year(df$datetime)
df$month <- lubridate::month(df$datetime, label = TRUE)

# Exclude outliers in PurpleAir and Neph
df %>% filter(pm2_5_atm <= max_pm_range) # global variable stored in init.R
saveRDS(df, paste0("./data/homogenousPM/df_", time_res, ".RDS"))

# Loop through each sensor and run regression
sensors <- unique(df$id) # use id here instead of label b/c it has duplicates (e.g. Miller)
system("rm ./figures/pa_neph_cal/*/*.png")

r_squared_threshold <- 0.90
n_obs_threshold <- 50

lm_results <- Reduce(rbind, pblapply(sensors, function(n) {
# Select specific node

def_sensor <- filter(df, id %in% n)
label <- def_sensor$label[1] %>% gsub(pattern = '\s|\\/|\.\s', replacement = '_') %>% tolower()

# Run models

mod <- lm(pm2_5_mean~pm2_5_atm, df_sensor)
summary_lm <- broom::glance(mod)
summary_lm$slope <- mod$coef[[2]]
summary_lm$intercept <- mod$coef[[1]]
# summary_lm$model_type <- "lm"
summary_lm$n_obs <- nrow(df_sensor)
summary_lm$id <- n

# Visualize CV

try{
  if(summary_lm$r.squared >= r_squared_threshold & summary_lm$n_obs >= n_obs_threshold) {
    nephRegression(df = df_sensor, y = "pm2_5_mean", x = "pm2_5_atm",
                   outdir = ".//figures/pa_neph_cal/pass/",
                   clean_time = paste0("lm_", label), reduced = F)
  } else {
    nephRegression(df = df_sensor, y = "pm2_5_mean", x = "pm2_5_atm",
                   outdir = ".//figures/pa_neph_cal/fail/",
                   clean_time = paste0("lm_", label), reduced = F)
  }
}
return(summary_lm)

# , cl = detectCores() - 1

saveRDS(lm_results, paste0("./data/homogenousPM/lm_results_", time_res, ".RDS"))
system(paste0("zip -r ./figures/lm_rsd10_", time_res, ".zip ./figures/pa_neph_cal/"))

# see ./src/compareCF.R for 15-min vs. 1-hour comparison

lm_df <- lm_results %>% filter(r.squared >= r_squared_threshold) %>%
      filter(n_obs >= n_obs_threshold)

r2_violin <- ggplot(lm_results, aes(x = id, y = r.squared)) +
             geom_violin(fill = "cornflowerblue") +
Appendix A. Source Code

```r
geom_hline(yintercept = r_squared_threshold, color = "firebrick", lineType = 2) +
ylab(expression(R^2)) +
# ggtitle(paste0(expression(Distribution of R^2), " (Nephelometer-PurpleAir) " )) +
ggtitle(bquote('Distribution of R^2~(Nephelometer-PurpleAir)~.(time_res)')) +
theme_bw() +
theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
      axis.ticks.x = element_blank())

r2_violin

png(filename = paste0("./figures/pa_neph_violins/r2_", time_res, ".png"),
     width = 8,
     height = 6,
     units = "in",
     res = 300)
r2_violin
dev.off()

slope_violin <- ggplot(lm_df, aes(x = id, y = slope)) +
geom_violin(fill = "forestgreen") +
ylab(expression(beta)) +
ggtitle(paste0("Distribution of Slopes (Nephelometer-PurpleAir) ",
                time_res)) +
theme_bw() +
theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
      axis.ticks.x = element_blank())
slope_violin

png(filename = paste0("./figures/pa_neph_violins/slope_", time_res, ".png"),
     width = 8,
     height = 6,
     units = "in",
     res = 300)
slope_violin
dev.off()

# I'm scrapping this idea for now. Might revisit 15-minute CF later if needed...
# Salvage any "failed" sensors by hand
# id_pass <- c()```
# Connect CF to each PurpleAir and upload to DB

# Checkpoint our homogenous PM results for our CF script
saveRDS(lm_df, ".data/cf/homogenous_pm_cf.RDS")

# Apply best method from EDA to pdx_cf
# Do this after storing data in DB (store even bad sensors CF for now ...)
# pdx_cf %>%
#   group_by(id, label) %>%
#   slice(which.max(r.squared)) %>
#   filter(r.squared >= 0.80) %>% arrange(label)

# Append geometry and convert to simplefeatures
pdx_cf <- inner_join(lm_df, pa_points) %>% st_as_sf()

# Prepare colnames for postgres (remove periods)
names(pdx_cf) <- gsub(x = names(pdx_cf), pattern = "\./", ".")

# names(pdx_cf)[names(pdx_cf) == "id"] <- "label"

# HIMS Roof was returning Inf and NaN
pdx_cf %>% na.omit()

saveRDS(pdx_cf, ".data/cf/pdx_cf.RDS")

# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres()
  , dbname = db
  , host = 'pgsql102.rc.pdx.edu' # not sure why object host isn't working...
  , port = port
  , password = pw
  , user = user)

# Find a better way to do this... append = TRUE is not ideal, but append = FALSE and overwrite = TRUE converts numeric to real datatype!?
# Write to db as lookup table

# clear table before writing new data
res <- dbSendQuery(con, statement = "delete from pdx_cf;")
dbClearResult(res)
res <- dbSendQuery(con, statement = "vacuum full pdx_cf;")
dbClearResult(res)
# Write the new CF data to the pdx_cf table in canopycontinuum

```
st_write(obj = pdx_cf,
          dsn = con,
          layer_options = "OVERWRITE=true",
          drop_table = FALSE,
          try_drop = FALSE,
          debug = TRUE,
          append = TRUE,
          overwrite = FALSE,
          dataset_options = "GEOMETRY=AS_WKT")
```

# TODO create new pdx_cf table with appropriate attribute names!

```
# cf <- st_read(dsn = con
#   # ,geom_column = "geom"
#   # ,EWKB = TRUE
#   # , query = "SELECT DISTINCT * FROM pdx_local_slope;"
#   # , query = "SELECT DISTINCT * FROM pdx ;"
#   # compare PurpleAir to DEQ during regional PM event (wildfire)
#   # ,query = "SELECT * FROM pdx_cf ;"
# )
```

# INSERT FLAG DATA SCRIPT HERE

```
# added to processPurpleAir in aod_modeling instead ...
# pdx_flag <- st_read(dsn = con
#   # ,geom_column = "geom"
#   #,query = "SELECT DISTINCT * FROM pdx_flag;")
# ```

---

### Source/init.R

```r
library(raster)
library(pbapply)
source("./src/functions.R")

# Global Variable definition

# Oregon North NAD83 Meters UTM Zone 10
epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

# Portland Study Area
```
study_area <- "./study_area/study_area.RDS"
bb <- "./study_area/bb.RDS"

# Study start and end date
start_date = "2017-09-01"
# end_date = "2018-09-30" # note, download/processGridMET must be re-ran if more recent end date is required!
# end_date = "2019-05-29"
end_date = "2019-03-31" # GridMet and PBL limited to ~April

crs = epsg_26910
study_area = "./study_area/study_area.RDS"
pm_df = "./data/purpleair/pm_df_24h.RDS"

# for testing functions
date <- "2017.10.10"

# Minimum Spatial Resolution for all variables
sp_res <- 50 # meters

# Max PM range (used for portlandCF.R, and processPurpleAir.R)
max_pm_range <- 250

# Use nephelometer data as a scaling factor for PurpleAir network?
scale_neph = F

# Set tmpdir for raster data
setRasterTempDir()

# Raster black white color palette
bw <- colorRampPalette(c('white', 'black'))(50)

# Print progressbar to CL
pbo = pbOptions(type="timer")

Source/interpolateNDVI.R
# Library (#, rasterVis, zoo, magrittr, parallel)

```r
# sourceforge repo
```

```r
library(raster); library(rasterVis); library(zoo)
```

```r
# r1 = raster(matrix(c(NA, rep(1,8)), 3, 3))
# r2 = raster(matrix(c(NA, rep(2,8)), 3, 3))
# r3 = raster(matrix(c(NA, rep(3,8)), 3, 3))
# b1 = brick(list(r1, r2, r3))
# levelplot(b1)
```

```r
# # Fill in empty time steps
# b2 = brick(lapply(1:9, function(x) raster(matrix(NA, 3, 3))))
# old.layers = c(1, 4, 7)
# for (i in 1:nlayers(b1)) {
#   b2[[old.layers[i]]] = b1[[i]]
# }
# levelplot(b2, layout=c(9, 1))
```

```r
# for (cell in 1:ncell(b2)) {
#   if (all(is.na(as.vector(b2[cell])))) {
#     new.values = na.spline(as.vector(b2[cell]), na.rm = F)
#   } else {
#     new.values = na.spline(as.vector(b2[cell]), na.rm = F)
#   }
#   b2[cell][] = new.values
# }
# levelplot(b2, layout=c(9, 1))
```

```r
ndvi <- stack(list.files(path = "./output/ndvi/", pattern = ".tif$", full.names = T))
```

```r
evi <- stack(list.files(path = "./output/evi/", pattern = ".tif$", full.names = T))
```

```r
# Graphically assess layers with cloud interference
# cloud_days <- 
#   "2017.08.29 | 2017.11.01 | 2018.01.17 | 2018.02.18 | 2018.03.22"
```

```r
# Filter out bad Landsat scenes
```
# Appendix A. Source Code

```r
57 # ndvi <- subset(ndvi, grep(cloud_days, names(ndvi), value = T, invert=T))
58 # evi <- subset(evi, grep(cloud_days, names(evi), value = T, invert=T))
59 # ndvi <- subset(ndvi, grep("2017.09.14|2017.09.30", names(ndvi), value = T, invert=F))
60
61 getDates <- function(stack) {
62 names <- names(stack)
63 dates <- gsub("^.*\\.\", "", names)
64 dates <- gsub("\\\.", "-", dates) %>% as.Date()
65 return(dates)
66 }
67
68 getVars <- function(stack) {
69 names <- names(stack)
70 # extract variable name (characters before date)
71 vars <- gsub("\\..*$", "", names)
72 return(vars)
73 }
74
75 interpolateNDVI <- function(stack) {
76 # 16 day observations
77 bi_month <- getDates(stack)
78 vars <- getVars(stack)
79
80 # generate daily sequence
81 dates <- seq.Date(from = min(bi_month), to = max(bi_month), by = "day")
82 dates <- gsub("-", "\\\.", dates)
83 daily_names <- paste0(vars, ".", dates)
84
85 # create empty rasters for missing days
86 daily_stack <- pblapply(dates, function(x) {
87 raster(
88 matrix(NA, nrow(stack), ncol(stack)),
89 ,crs = crs(stack)
90 #nrows = nrow(stack)
91 #,ncols = ncol(stack)
92 #,ext = extent(stack)
93 #,values = 0
94 # ,xmin = xmin(stack)
95 # ,ymin = ymin(stack)
96 # ,xmax = xmax(stack)
97 # ,ymax = ymax(stack)
98 )
99 #values(r) <- NA
```
Appendix A. Source Code

101  
102  #, cl = detectCores() - 1 
103  ) 
104  
105  s <- stack(daily_stack) 
106  extent(s) <- extent(stack) 
107  values(s) <- NA 
108  names(s) <- daily_names 
109  
110  # fill in 16 day observations 
111  daily_stack <- pblapply(names(s), function(x) { 
112    if(x %in% names(stack) & x %in% names(s)) { 
113      s[[x]] <- stack[[x]] 
114    } else { 
115      s[[x]] 
116    } 
117  } 
118  #, cl = detectCores() - 1 
119  ) 
120  
121  s <- stack(daily_stack) 
122  
123  # this works, but isn’t parallel 
124  # for (cell in 1:ncell(s)) { 
125  #   if (all(is.na(as.vector(s[cell])))) { 
126  #     new.values = as.numeric(rep(NA, dim(s)[3])) 
127  #   } else { 
128  #     new.values = na.spline(as.vector(s[cell]), na.rm = F) 
129  #   } 
130  #   s[cell][] = new.values 
131  # } 
132  
133  # parallel version of above for loop (this takes 2 hours on hecate! ) 
134  # daily_stack <- pblapply(1:ncell(s), function(cell) { 
135  #   if (all(is.na(as.vector(s[cell])))) { 
136  #     new_values <- as.numeric(rep(NA, dim(s)[3])) 
137  #   } else { 
138  #     new_values <- na.spline(as.vector(s[cell]), na.rm = F) 
139  #   } 
140  #   s[cell][] <- new_values 
141  # }, cl = detectCores() - 1) 
142  
143  # using raster calc is much faster but I don’t understand what’s 
144  # happening at all! 
145  # create empty vector for cell-wise raster processing 
146  layers <- nlayers(s) 
147  empty_layer <- rep(NA, layers)
# get the layer index of the 16-day observations
s_index <- names(s) %in% names(stack) %>% which()

# define our raster calc interpolation
rasterSpline <- function(x) {
  # don’t interpolate global NA cells
  if(all(is.na(x))) {
    empty_layer
  }

  # polynomial fitting daily raster cell to 16-day obs
  else {
    na.spline((empty_layer[c(s_index)] <- x), xout = 1:layers, na.rm = F)
  }
}

# perform cell-level temporal interpolation on stack
# raster::beginCluster(n = detectCores() - 1)
s <- raster::calc(s, rasterSpline)
names(s) <- daily_names

# raster::endCluster()
writeRaster(s, file.path(’./data’, unique(vars), paste0(names(s), ”.tif”)),
            overwrite=T,
            bylayer=T,
            format=”GTiff”)

interpolateNDVI(ndvi)
in interpolateNDVI(evi)

##
# ndvi2 <- stack(list.files(’./data/ndvi’, full.names = T))
# ndvi2 <- readRDS(’./data/ndvi_daily_composite.RDS”)
# names(ndvi2)
# plot(ndvi2)
# evi2 <- stack(list.files(’./data/evi’, full.names = T))
# evi2 <- readRDS(’./data/evi_daily_composite.RDS”)
# names(evi2)
# test <- evi2[[17:32]]
# plot(test)
#
Source/interpolateNeph.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018–11–27
# IDW interpolation of DEQ nephelometer data

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal,
  ,plyr
  ,dplyr
  ,magrittr
  ,gdalUtils
  ,raster
  ,rasterVis
  ,sp
  ,sf
  ,rgeos
  ,leaflet
  ,lubridate
  #,MODIS

# lapply (1:nlayers(ndvi), function(x) {
#  plot(ndvi[[x]], main = paste(names(ndvi[[x]])))
#  Sys.sleep(0.3)
# })
#
# lapply (1:nlayers(ndvi2), function(x) {
#  plot(ndvi2[[x]], main = paste(names(ndvi2[[x]])))
#  Sys.sleep(0.3)
# })
#
# lapply (1:nlayers(evi), function(x) {
#  plot(evi[[x]], main = paste(names(evi[[x]])))
#  Sys.sleep(0.5)
# })
#
# lapply (1:nlayers(evi2), function(x) {
#  plot(evi2[[x]], main = paste(names(evi2[[x]])))
#  Sys.sleep(0.5)
# })
Appendix A. Source Code

173

# , bowerbird
, gstat
, parallel
, snow
, pbapply
, ggplot2
, rts
, tidyR
, RPostgres
, ggmap
, gridExtra
, rlang )

source("./src/init.R")
source("./src/functions.R")

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.timezone()
Sys.setenv(TZ="GMT")
Sys.timezone()

# Read in data

neph <- readRDS("./data/deq/neph1day.RDS")
study_area <- readRDS("./study_area/study_area.RDS")

# Create spatial interpolation functions

# call gstat on each PM variable
apply_gstat <- function(col_name, pm_data, study_area, day, outdir) {
  pm_df <- dplyr::filter(pm_data, date >= day & date <= day) %>%
    dplyr::select(date, col_name, geometry) %>% na.omit()
  pm_df %>% as("Spatial")
  pm_df %>% spTransform(CRSobj = crs(study_area))
  gs <- gstat(formula = as.formula(paste(col_name, "~1"))
              ,locations = pm_df)
  r <- raster(study_area, crs = crs, res = sp_res, vals = 0)
  idw <- raster::interpolate(r, gs)
  idwr <- mask(idw, study_area)
  layer_name <- paste0(col_name, ".", gsub("-", "\.", day))
  names(idwr) <- layer_name
  #plot(idwr)
writeRaster(idwr, paste0(outdir, "/", layer_name, ".tif"), format = "GTiff", overwrite = T)

neph_mean <- function(col_name, pm_data, study_area, day, outdir) {
  pm_df <- dplyr::filter(pm_data, date >= day & date <= day) %>%
    dplyr::select(date, col_name, geometry) %>% na.omit() %>%
    st_drop_geometry()
  #pm_df %>% as("Spatial")
  #pm_df %>% spTransform(CRSGeo = CRS(study_area))
  daily_mean <- dplyr::summarise(pm_df, mean = mean(eval(parse(text =
    (col_name))), na.rm = T))
  r <- raster(study_area, crs = CRS, res = sp_res, vals = daily_mean
    [[1]])
  layer_name <- paste0(col_name, ".", gsub("-", "\.", day))
  names(r) <- layer_name
  #plot(r)
  writeRaster(r, paste0(outdir, "/", layer_name, ".tif"), format = "GTiff", overwrite = T)
}

# Call our ground-station nephelometer interpolation across each day
# (function wrapper)
interpolate_neph <- function(pm_data, day, vars, study_area, outdir) {
  pblapply(vars, function(x) {
    # IDW Interpolation
    # apply_gstat(pm_data = pm_data
    # ,day = day
    # ,study_area = study_area
    # ,col_name = x
    # ,outdir = outdir)
    # Daily Mean Nephelometer
    neph_mean(pm_data = pm_data
      ,day = day
      ,study_area = study_area
      ,col_name = x
      ,outdir = outdir)
    #,cl = detectCores()-1
  })
}
Appendix A. Source Code

110 }
111
112 # apply interpolation across each variable (fix start and end date later...)
113 # create date sequence
114 days <- seq.Date(as.Date(start_date), as.Date(end_date), by="days")
115 # create list of variables (only one in this case.. but kept dynamci for future PM data?)
116 vars <- c("pm2.5.mean", "pm2.5.min", "pm2.5.max", "pm2.5.sd", "pm2.5.median")
117
118 # Running the interpolation function
119 pblapply(days, function(x) {
120   interpolate_neph(pm_data = neph
121     , day = x
122     , vars = vars
123     , outdir = "./data/neph"
124     , study_area = study_area)
125   , cl = detectCores() - 1
126 })
127
128 # Checking to see if it worked
129 # Create RDS stacks
130 # pblapply(1:nlayers(r), function(x) {
131 #   plot(r[[x]], main = paste(names(r[[x]]))
132 #   , breaks = c(5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 80, 90)
133 #   , col = terrain.colors(14))
134 #   plot(ugb, add = T)
135 # })
136 # pblapply(1:nlayers(r), function(x) {
137 #   plot(r[[x]], main = "2018-07-04 Max PM2.5")
138 #   plot(ugb, add = T)
139 #})
Appendix A. Source Code

# plot(r[["pm2.5.mean.2018.07.04"]], col = terrain.colors(10), main = "2018−07−04 Mean PM2.5")
# plot(ugb, add = T)
# plot(r[["pm2.5.max.2018.01.16"]], col = terrain.colors(10))
# plot(ugb, add = T)

Source/purpleairQAQC.R

#!/usr/bin/env Rscript
args = commandArgs(trailingOnly=TRUE)

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2019−01−10
# Identify outliers and flag faulty PurpleAir sensors

# set up environment

# test if there is at least one argument: if not, return an error
if (length(args)==0) {
  purge_arg <- F
  print("Setting db purge to F")
} else if (length(args)==1) {
  purge_arg <- args[1]
  print(paste0("Applying db purge argument: ", args[1]))
}

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr, dplyr, ggplot2, stringr, magrittr, scales, lubridate, tidyr, sp, sf, rgdal, pbapply
Appendix A. Source Code

```r
,parallel
,RPostgres
)

# source("./.../aod_pm_modeling/src/init.R")
source("./src/functions.R")
source("./src/init.R")
# define global variables

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.timezone()
Sys.setenv(TZ="GMT")
Sys.timezone()

## connecting to local db
host <- 'pgsql120.rc.pdx.edu'
db <- 'canopycontinuum'
user <- 'porlando'
port <- 5433
pw <- scan("./batteries.pgpss", what = "") # in parent dir

# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres(,dbname = db
 ,host = 'pgsql102.rc.pdx.edu' # not sure why object host isn't working...
 ,port = port
 ,password = pw
 ,user = user)

# create unanimous time resolution for all data
time_resolution <- "1 sec"

# time zone applied to all data
time_zone <- "GMT"

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

# Albuquerque, NM, UTM 13N
```
utm_13N <- "+proj=utm +zone=13 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"

# Sacramento, CA, UTM 10S, meters
epsg_26911 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Oregon North NAD83 HARN meters
epsg_2838 <- "+proj=lcc +lat_1=46 +lat_2=44.33333333333334 +lat_0=43.66666666666666 +lon_0=−120.5 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs"

epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Function definition

# creating a custom not-in function
'!%in%' <- function(x,y) '\'!%in%'(x,y)

set.seed(1)

# remove geometry from sf object, convert to dataframe with only the elements/attributes
st_drop_geometry <- function(x) {
  if(inherits(x,"sf")) {
    x <- st_set_geometry(x, NULL)
    class(x) <- 'data.frame'
  }
  return(x)
}

# data processing

# this should work once I update my pdx table
# pa <- st_read(dsn = con
# ,geom_column = "geom"
# ,EWKB = TRUE
# ,query = "SELECT DISTINCT * FROM pdx_local_slope;"
# ,query = "SELECT DISTINCT * FROM pdx;"
# compare PurpleAir to DEQ during regional PM event (wildfire)
122 # query = paste0("SELECT DISTINCT * FROM pdx WHERE 
123 created_at >= ", start_date, ", AND created_at <= ", end_date, ", ")
124 # )
125 # lookup table for sensor geom
126 pa_points <- st_read(dsn = con
127 # ,geom_column = "geom"
128 # ,EWKB = TRUE
129 # ,query = "SELECT DISTINCT * FROM pdx_local_
130 slope;"
131 # ,query = "SELECT DISTINCT * FROM pdx;"
132 # compare PurpleAir to DEQ during regional PM
133 event (wildfire)
134 # ,query = "SELECT DISTINCT id , label , geom FROM
135 pdx GROUP BY id , label , geom;"
136 )
137 dbDisconnect(con)
138 # define projection
139 # st_crs(pa) <- epsg_26910
140 st_crs(pa_points) <- epsg_26910
141 # convert from UTC to UTC-8 or PST (same as DEQ)
142 # df$created_at <- lubridate::ymd_hms(df$created_at) - hours(8)
143 # make sure the datetime columns match
144 # names(pa)[names(pa) == "created_at"] <- "datetime"
145 # str(pa)
146 # check the time zone of pa data
147 # attr(df$datetime , "tzone")
148 # Flag data from faulty sensors
149 # pa_flag <- pa %>% dplyr::select(datetime , id , label , pm2_5_atm, geom)
150 # pa_flag$date <- as.Date(pa_flag$datetime)
151 # node_labels <- unique(gsub(" B$", "", pa_flag$label))
152 # head(pa_flag)
153 points <- unique(pa_points$geom)
154 # pa_df <- pa_flag
gem_id <- "POINT (512339.5 5042418)" # STAR LAB BETHANY (no comma)
156 start_date <- "2017-07-01" # in UTC timezone...
157 end_date <- as.character(Sys.Date() - 1)
# geoms <- unique(as.character(pa_flag$geom))
geoms <- unique(st_as_text(pa_points$geom))
# geoms <- rev(geoms)
# geom_id <- geoms[5]

flag_purpleair <- function(geom_id, day
  ,host = host
  ,db = db
  ,user = user
  ,port = port
  ,pw = pw) {
  # connect to db
  con <- dbConnect(drv = RPostgres::Postgres()
    ,dbname = db
    ,host = 'pgsql102.rc.pdx.edu'
    ,port = port
    ,password = pw
    ,user = user)

  # check to see if flag exists for node/day
  flag_check <- st_read(dsn = con
    ,query = paste0("SELECT DISTINCT * FROM pdx_
        ,WHERE date = "
          ,lubridate::ymd(day)
        ,", geom_id," '::geometry, ST_SRID(geom));"
  )

  # skip node/day flag re-write to db
  if(nrow(flag_check) == 0) {

    # pull day's data for specific node
    node <- st_read(dsn = con
      ,query = paste0("SELECT DISTINCT * FROM pdx WHERE
        ,created_at >= "
          ,lubridate::ymd(day) + hours(8),
        ," AND created_at < " 
        ,lubridate::ymd(day) + hours(8) +
        ,days(1)
      )

      ,query = paste0("SELECT DISTINCT * FROM pdx WHERE
        ,"2018-09-01"
      )
  )

  }
```r
# make sure there's enough data for this day and sensor...
if(nrow(node) > 0) {

    # Remove B from label name
    node$node <- gsub(" B", ",", node$label)

    node$created_at <- lubridate::ymd_hms(node$created_at) -
    lubridate::hours(8)
    node$date <- as.Date(node$date)

    st_crs(node) <- epsg_26910
    # Extract each day's observations from network
    # df <- dplyr::filter(pa_df, date == as.Date(day)) %>% as.data.frame()
    # df$geom <- as.character(df$geom)

    # Grab corresponding node_id
    # node_geom <- dplyr::filter(df, st_as_text(geom) == geom_id)
    # df$geom <- as.character(df$geom)

    # Pull day's total network data
    df <- st_read(dsn = con,EWKB = T, query = paste0("SELECT DISTINCT * FROM pdx WHERE
    created_at >= ", lubridate::ymd(day) + hours(8),
    " AND created_at < " # put dates in terms of local tz, instead of UTC
    ,lubridate::ymd(day) + hours(8) +
    days(1)
    "");
    #"", AND geom = ST_SETSRID("", geom_id," ::geometry, ST_SRID(geom));"
    )

    df$created_at <- lubridate::ymd_hms(df$created_at) - lubridate::hours(8)
    st_crs(df) <- epsg_26910

    # exclude specific node data from the remaining network
    network <- df %>%
    filter(pm2_5_atm < 500) %%
    filter(id %!in% unique(node$id))
    # network <- df %>% dplyr::filter(as.character(geom) != geom_id
    & pm2_5_atm < 500)

    )
```

# Compare A & B sensor of single node

node_wide <- node %>%
  # as.data.frame() %>%
  st_drop_geometry() %>%
  dplyr::select(created_at, label, pm2_5_atm) %>%
  group_by_at-vars(pm2_5_atm) %>%
  dplyr::mutate(row_id=1:n()) %>%
  ungroup() %>%
  spread(key=label, value=pm2_5_atm) %>%
  dplyr::select(-row_id)

mod <- lm(node_wide[[2]] ~ node_wide[[3]])

status_a <- data.frame(date = as.Date(day), label = names(node_wide[2]))
status_b <- data.frame(date = as.Date(day), label = names(node_wide[3]))

node_id <- unique(node$node)

status_a$id <- node %>% filter(node == node_id & sensor == "A") %>%
  dplyr::select(id) %>%
  unique() %>%
  st_drop_geometry() %>%
  as.character()

status_b$id <- node %>% filter(node == node_id & sensor == "B") %>%
  dplyr::select(id) %>%
  unique() %>%
  st_drop_geometry() %>%
  as.character()

status_a$r2 <- summary(mod)$adj.r.squared
status_b$r2 <- summary(mod)$adj.r.squared
status_asm <- mod$coefficients[[2]]
status_bsm <- mod$coefficients[[2]]
status_asb <- mod$coefficients[[1]]
status_bsb <- mod$coefficients[[1]]

status_a$median <- median(node_wide[[2]], na.rm = T)
status_b$median <- median(node_wide[[3]], na.rm = T)
status_a$median_log <- median(log(node_wide[[2]]+1), na.rm=T)
status_b$median_log <- median(log(node_wide[[3]]+1), na.rm=T)

median_network <- median(network$pm2_5_atm, na.rm = T)
median_log_network <- median(log(network$pm2_5_atm+1), na.rm = T)
sd_network <- sd(network$pm2_5_atm)
sd_log_network <- sd(log(network$pm2_5_atm+1))

# compare sd with local neighbors instead of entire network?
status_a$diff <- abs((status_a$median - median_network)/sd_network)
Appendix A. Source Code

```r
status_b$diff <- abs((status_b$median - median_network) / sd_network)
status_a$diff_log <- abs((status_a$median_log - median_log_network) / sd_log_network)
status_b$diff_log <- abs((status_b$median_log - median_log_network) / sd_log_network)

z_crit <- 1.96
z_log_crit <- log(z_crit)
# z_log_crit <- log(z_crit+1) # which is correct?
status_a$diff_log > z_log_crit
status_b$diff_log > z_log_crit

# TODO add conditional for days with low n (negative R2??!! with roof data)

# compare sensors to surrounding network if correlation is weak
if(summary(mod)$adj.r.squared < 0.85) {

  # FLAGGING
  if(status_a$diff_log > z_log_crit) {
    status_a$flag <- 1
  } else {
    status_a$flag <- 0
  }
  if(status_b$diff_log > z_log_crit) {
    status_b$flag <- 1
  } else {
    status_b$flag <- 0
  }

} else {
  # do not flag any sensor (strong correlation between A & B)
  status_a$flag <- 0
  status_b$flag <- 0
}

# flag local extrema despite decent R2 between A & B
# FLAGGING

# if(status_a$diff_log > z_log_crit) {
  # status_a$flag <- 1
# } else {
  # status_a$flag <- 0
# }
# if(status_b$diff_log > z_log_crit) {
  # status_b$flag <- 1
# } else {
  # status_b$flag <- 0
# }
```

# add in remaining global parameters each row
status <- rbind(status_a, status_b)
status$median_network <- median_network
status$median_log_network <- median_log_network
status$sd_network <- sd_network
status$sd_log_network <- sd_log_network
status$z_crit <- z_crit
status$z_log_crit <- z_log_crit
pdx_flag <- status

# adding the actual sfc geom back
# asdf <- pa_df %% filter(id == status_a$id) %>%
#  dplyr::select(id, label, geom)

# attaching geom to pdx_flag
pdx_flag$geom <- node$geom[1]
pdx_flag %>% st_as_sf()

# this is seriously ugly but it works and I don’t want to redo
# how I’m applying over characters...
# Re-attaching the sfc class geom to add to status$geom for st_write()
# node_char <- dplyr::filter(df, as.character(geom) == geom_id)
#  %>% dplyr::select(-geom)
#  head(1)
#
# node_geometry <- left_join(node_char, asdf
#  # by = c("label", "id")) %>% st_as_sf()
#
# node_geom <- unique(node_geometry$geom)
# something still broken with plot(pdx_flag), geom may be
# affected...
# pdx_flag$geom <- st_as_sfc(node_geom)
# pdx_flag %>% st_as_sf()
# st_crs(pdx_flag) <- st_crs(pa_df)
# return(status)
# upload to db instead of storing in memory?
# open connection to our db
# con <- dbConnect(drv = RPostgres::Postgres()
#  ,dbname = db
#  ,host = 'pgsql102.rc.pdx.edu'
#  ,port = port
#  ,password = pw
#  ,user = user)

st_write(dsn = con
Appendix A. Source Code

```
, obj = pdx_flag
, geom_name = "geom"
, append = T)
  # rm(pdx_flag, df, mod, network, node)
  # print(head(pdx_flag, 2))
  # this prints to console like a million times, but only the two
distinct rows are actually uploaded!
  print(paste0("Upload for ", node$node, " on ", day, " complete"))
  gc()
} else {
  print(paste0("No data for ", geom_id, " on ", day, ""))
}
} else {
  print(paste0("Flag already exists in db for ", geom_id, " on ",
day, ""))
  dbDisconnect(con)
}

# I need to loop through each sensor pair when flagging
flag_daily_purpleair <- function(geom_id, days
  , host = host, db = db, user = user,
  port = port, pw = pw)

pblapply(days, function(x) {
  # wrap in try block to avoid errors?
  try(
    flag_purpleair(day = x, geom_id = geom_id, host = host, db = db
    , user = user, port = port, pw = pw)
  )
  }
  
  # , cl = detectCores() - 1 #
  # , cl = 3 # still crashes session...
  gc()
  }

flag_network <- function(geoms, start_date, end_date
  , host = 'pgsql120.rc.pdx.edu'
  , db = 'canopycontinuum'
  , user = 'porlando'
  , port = 5433
  , pw = scan("./batteries.pgpss", what = "")
```
purge_db = F  # set T when using new/different outlier detection algo
)

if (purge_db) {
  print("Begin Database Purge...")
  con <- dbConnect(drv = RPostgres::Postgres()
      , dbname = db
      , host = 'pgsql102.rc.pdx.edu'
      , port = port
      , password = pw
      , user = user)
      
      # clear table before writing new data
      res <- dbSendQuery(con, statement = "delete from pdx_flag;")
      dbClearResult(res)
      res <- dbSendQuery(con, statement = "vacuum full pdx_flag;")
      dbClearResult(res)
      dbDisconnect(con)
      print("Database purge complete!")

  }

  # loop through a given node each day
  days <- seq.Date(as.Date(start_date)
      , as.Date(end_date)
      , by = "day")
      
  pblapply(unique(geoms), function(x) {
    flag_daily_purpleair(geom_id = x
      , days = days
      , host = host
      , db = db
      , user = user
      , port = port
      , pw = pw
    )
    # cat(""
    # cat(paste0("Uploading geom ", x, " to pdx_flag complete!"))) gc()
  })
  
  #,cl = 3 # 8GB per core
  ,cl = detectCores() - 1
  }
}
Appendix A. Source Code

```r
flag_network(geoms = geoms,
  start_date = start_date,
  end_date = end_date,
  purge_db = purge_arg)
```

```r
# # investigate individual cases
# asdf <- pa_flag %>%
# filter(date == "2018-07-27" & label == "STAR LAB BETHANY" | date == "2018-07-27" & label == "STAR LAB BETHANY B") %>%
# st_drop_geometry() %>%
# as.data.frame() %>%
# dplyr::select(pm2_5_atm, label) %>%
# group_by_at(vars(-pm2_5_atm)) %>%
# dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
# spread(key = label, value = pm2_5_atm) %>% dplyr::select(-row_id)
#
# # sub <- head(pa_flag, 1000)
# sub %>% st_drop_geometry()
# sub %>%
# group_by_at(vars(-pm2_5_atm)) %>%
# dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
# spread(key = label, value = pm2_5_atm) %>% dplyr::select(-row_id)
#
```

Source/purpleAirCF.R

```r
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018-09-17
# determine correction factors for Portland PurpleAir network
#
# set up environment
#
# load the necessary packages
if (!require(pacman)) {
```
install.packages("pacman")
library(pacman)
}

p_load(readr,
  readxl
  #, xlsx
  , ggplot2
  , plyr
  , dplyr
  #, multidplyr
  , broom
  , reshape2
  , tidyR
  , stringr
  , magrittr
  , rlist
  , grid
  , gridExtra
  , tidyquant
  , scales
  , qdapRegex
  , sp
  , sf
  #, mapview # ***** with specific leaflet install
  , devtools
  , RColorBrewer
  , classInt
  , htmltools
  , scales
  , htmlwidgets
  , httr
  , jsonlite
  , rgdal
  , pbapply
  , snow
  , parallel
  , data.table
  , RPostres
  , tsoutliers
  , forecast
  , MASS
  , viridis
  , corrplot
  , nngeo
)

# define global variables

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.timezone()
Sys.setenv(TZ="GMT")
Sys.timezone()

## connecting to local db
host <- 'pgsql120.rc.pdx.edu'
db <- 'canopycontinuum'
user <- 'porlando'
port <- 5433
pw <- scan("./batteries.pgpss", what = "") # in parent dir

# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres()
  ,dbname = db
  ,host = 'pgsql102.rc.pdx.edu' # not sure why object host isn't working...
  ,port = port
  ,password = pw
  ,user = user)

# create unanimous time resolution for all data
time_resolution <- "1 sec"

# time zone applied to all data
time_zone <- "GMT"

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"

# Oregon North NAD83 HARN meters
epgs_2838 <- "+proj=llc +lat_1=46 +lat_2=44.33333333333334 +lat_0=43.66666666666666 +lon_0=-120.5 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs"

# Oregon North NAD83 Meters UTM Zone 10
epgs_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
# Function definition

# creating a custom not-in function
'%! in%' <- function(x, y) ("%!'in%(x, y))

# plot density in ggplot
get_density <- function(x, y, n = 100) {
  dens <- MASS::kde2d(x = x, y = y, n = n)
  ix <- findInterval(x, dens$x)
  iy <- findInterval(y, dens$y)
  ii <- cbind(ix, iy)
  return(dens$z[ii])
}

set.seed(1)

# file_path <- "./data/frm/portland_5stn_20180101_to_20180719.csv"
# read function for the 5-min data that Meena sent over!
read_frm <- function(file_path) {
  # read in the raw data
  x <- read.csv(file_path, header = TRUE, skip = 2)
  # delete sub headers
  # x <- x[-c(1:2),]

  # convert date from character to POSIXct
  x$Date...Time <- mdy_hm(x$Date...Time, tz = time_zone)

  x <- x %>% na.omit()

  # convert to tidy format, gather on site id
  long <- x %>%
    gather(id, pm2_5_atm, -c(Date...Time))

  # convert characters in pm2_5 data to NA
  long$pm2_5_atm <- as.numeric(long$pm2_5_atm)

  # exclude NAs (No Data, Zero, DL/Computer, etc. gets converted to NA)
  long <- long %>% na.omit()

  # alter column names
  colnames(long) <- c("datetime", "site_id", "pm2_5_ref")
long $site_id <- gsub("^X[0-9]+\.\", "\", long $site_id)
long $site_id <- gsub("\".\", "\", long $site_id)

long $site_id <- ifelse (long $site_id == "Portland Near Rd", "Tualatin Bradbury Court", long $site_id)

long $lat [long $site_id == "Beaverton Highland Park"] <- 45.470183
long $lat [long $site_id == "Hillsboro Hare Field"] <- 45.528549
long $lat [long $site_id == "Portland SE Lafayette"] <- 45.496635
long $lat [long $site_id == "Portland Cully Helensview"] <- 45.562206
long $lat [long $site_id == "Sauvie Island"] <- 45.768528
long $lat [long $site_id == "Tualatin Bradbury Court"] <- 45.399189

long $lon [long $site_id == "Beaverton Highland Park"] <- -122.816418
long $lon [long $site_id == "Hillsboro Hare Field"] <- -122.972423
long $lon [long $site_id == "Portland SE Lafayette"] <- -122.602883
long $lon [long $site_id == "Portland Cully Helensview"] <- -122.575627
long $lon [long $site_id == "Sauvie Island"] <- -122.772305
long $lon [long $site_id == "Tualatin Bradbury Court"] <- -122.745538

long %<>% dplyr::select (datetime, site_id, pm2_5_ref, lat, lon)
long %<>% na.omit ()

long $site_id <- gsub("\"\", ",", long $site_id)
long $site_id <- tolower (long $site_id)
#long $site_id <- stringr::str_replace (long $site_id, ".\", "\")

return (long)

# establish avg interval for xts() function
time_step <- 5

# for clean 5-min breaks
align.time.down = function (x, n) {
  index (x) = index (x) - n
  align.time (x, n)
}

# remove geometry from sf object, convert to dataframe with only the elements/attributes
st_drop_geometry <- function (x) {
}
if (inherits(x,"sf")) {
    x <- st_set_geometry(x, NULL)
    class(x) <- 'data.frame'
}
return(x)

# for clean 5-min breaks
align.time.down = function(x,n) {
    index(x) = index(x) - n
    align.time(x,n)
}

# data processing

# start_date <- "2017-07-01"
# end_date <- "2018-09-30"
# inherit start/end date from init script in modeling project
source("./src/init.R")

# this should work once I update my pdx table
pa <- st_read(dsn = con
    #.geom_column = "geom" # deprecated!
    #.EWKB = TRUE
    #.query = "SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = "SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, " AND
    #.query = paste0("SELECT DISTINCT * FROM pdx WHERE pm2_5_atm < " max_pm_range, ")
    #.geom_column = "geom"
    #.EWKB = TRUE
    #.query = "SELECT DISTINCT * FROM pdx_local_slope;
    #.query = "SELECT DISTINCT * FROM pdx;"
    # compare PurpleAir to DEQ during regional PM event (wildfire)
    #.query = "SELECT DISTINCT id, label, geom
FROM pdx
    
    # lookup table for sensor geom
    pa_points <- st_read(dsn = con
    #.geom_column = "geom"
    #.EWKB = TRUE
    #.query = "SELECT DISTINCT * FROM pdx_local_slope;"
    #.query = "SELECT DISTINCT * FROM pdx;"
    # compare PurpleAir to DEQ during regional PM event (wildfire)
    #.query = "SELECT DISTINCT id, label, geom
FROM pdx
    

GROUP BY id, label, geom;"

# define projection
st_crs(pa) <- epsg_26910
st_crs(pa_points) <- epsg_26910

# convert from UTC to UTC-8 or PST (same as DEQ)
pa$created_at <- lubridate::ymd_hms(pa$created_at) - hours(8)

# read in the frm data (not actually frm, but neph...) provided by Meena
# frm_files <- c("./data/frm/sel_pm25_5min_20170701_to_20180918.csv")
frm_files <- ".\data/frm/bhp_hhf_tbc_sel_5min_pm25_20170701_to_20180918.csv"
frm <- ldply(frm_files, read_frm)
frm %>% st_as_sf(coords = c("lon","lat"), crs = wgs_84)

# matching the sel file with the 5stn id
# frm$site_id <- ifelse(frm$site_id == "PM2 5L_Est", "Portland SE Lafayette", frm$site_id)

# make sure the datetime columns match
names(pa)[names(pa) == "created_at"] <- "datetime"

str(frm)
str(pa)

# check the time zone of pa data
attr(pa$datetime, "tzone")
attr(frm$datetime, "tzone")

# TODO INSERT FLAG DATA SCRIPT HERE

# added to processPurpleAir in aod_modeling instead...
# pdx_flag <- st_read(dsn = con
# ,geom_column = "geom"
# ,query = "SELECT DISTINCT * FROM pdx_flag;")

# run initial cor plot before creating individual regressions
# nearby_sensors <- c("byuki project"
# pull all of the Portland PurpleAir data
nearby_sensors <- unique(pa$label)

# I'm only comparing SEL to each PurpleAir!?
sel <- frm %>% dplyr::filter(site_id == "Portland SE Lafayette" &
  datetime >= start_date & datetime <= end_date) %>%
  dplyr::select(datetime, id = site_id, pm2_5 = pm2_5_ref) %>%
  dplyr::group_by_at(vars(pm2_5)) %>%
  dplyr::mutate(row_id = 1:n()) %>%
  ungroup() %>%
  spread(key = id, value = pm2_5) %>%
  dplyr::select(-row_id) %>%
  # remove outliers
  filter('Portland SE Lafayette' < max_pm_range)

saveRDS(sel, ".data/frm/sel.RDS")

# Develop better filtering technique to exclude days when Plantower
# reads above threshold
system.time{
Purple <- pa %>% dplyr::filter(label %in% nearby_sensors & datetime >= start_date & datetime <= end_date) %>%
  # remove sticky geom
  as.data.frame() %>%
  dplyr::select(datetime, label = label, id, pm2_5 = pm2_5_atm) %>%
  # remove outliers
  # filter(pm2_5 < max_pm_range) %>%
  dplyr::group_by_at(vars(pm2_5)) %>%
  dplyr::mutate(row_id = 1:n()) %>%
  dplyr::select(-row_id)
  # convert to wide df for corr matrix
# Exclude days when sensor exceeds threshold?

```r
system.time(
  outliers <- pa %>% dplyr::filter(label %in% nearby_sensors &
                                  datetime >= start_date & datetime <= end_date) %>%
  # remove stick geom
  as.data.frame() %>%
  dplyr::select(datetime, label = label, id, pm2_5 = pm2_5_atm) %>%
  # remove outliers
  # filter(pm2_5 < max_pm_range) %>%
  filter(pm2_5 < 1000) %>%
  dplyr::group_by_at(vars(-pm2_5)) %>%
  dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
  dplyr::select(-row_id)
  # convert to wide df for corr matrix
  # spread(key = site_id, value = pm2_5) %>%
)
```

# long df after changing purple pipes and site_id
# asdf <- inner_join(purple, sel)
# asdf <- inner_join(purple, frm)
# asdf$year <- lubridate::year(asdf$datetime)
# asdf$month <- lubridate::month(asdf$datetime, label = TRUE)

# TODO loop through each ref/sensor pair!

```r
lapply(unique(frm$site_id), function(x) {

})
```

```r
r2_month <- asdf %>%
  group_by(year, month, id, label) %>%
  do(fitMonth = lm('Portland SE Lafayette ~ pm2_5, data = .)) %>%
  glance(fitMonth) %>%
  arrange(desc(r.squared))
```

# need a better way to get the slope and intercept without losing the
# std.error from each variable
```r
slope_month <- asdf %>%
  nest(-c(year, month, id, label)) %>%
```
mutate(
  fit = map(data, ~lm('Portland SE Lafayette ~ pm2_5, data = .)),
  tidied = map(fit, tidy)
) %>%
unnest(tidied) %>%
  # dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
  # dplyr::select(-c(std.error, statistic, p.value)) %>%
  # spread(key = term, value = estimate) %>%
  # dplyr::select(-c(row_id))
  filter(term != "(Intercept)") %>%
  filter(estimate < 5) %>%
  filter(estimate > -5)

intercept_month <- asdf %>%
  nest(-c(year, month, id, label)) %>%
  mutate(
    fit = map(data, ~lm('Portland SE Lafayette ~ pm2_5, data = .)),
    tidied = map(fit, tidy)
  ) %>%
  unnest(tidied) %>%
  # dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
  # dplyr::select(-c(std.error, statistic, p.value)) %>%
  # spread(key = term, value = estimate) %>%
  # dplyr::select(-c(row_id))
  filter(term != "Portland SE Lafayette") %>%
  filter(estimate < 25) %>%
  filter(estimate > -25)

# combine R2, slope, and intercept into single tidy dataframe
estimates_month <- inner_join(slope_month, intercept_month, by = c("year", "month", "id", "label"), suffix = c(".m", ".b"))
names(r2_month)[names(r2_month) == 'p.value'] <- "p.value.r2"
names(r2_month)[names(r2_month) == 'statistic'] <- "statistic.r2"

lm_month <- inner_join(r2_month, estimates_month, by = c("year", "month", "id", "label")) %>%
  arrange(year, month, id)

# Add August 2018 CF to DataBase

pdx_cf <- lm_month %>%
  # dplyr::select(-c(year.m, year.b)) %>%
  # dplyr::filter(month == "Aug", year == "2018") %>%
  arrange(desc(r.squared), id) %>%
dplyr::select(c(year, month, id, label, everything())) %>% unique()

pdx_cf$year <- as.character(pdx_cf$year)

# Apply best method from EDA to pdx_cf
# Do this after storing data in DB (store even bad sensors CF for now ...)
# pdx_cf %>%
#  group_by(id, label) %>%
#  slice(which.max(r.squared)) %>%
#  filter(r.squared >= 0.80) %>% arrange(label)

# Append geometry and convert to simplefeatures
pdx_cf <- inner_join(pdx_cf, pa_points) %>% st_as_sf()

# Prepare colnames for postgres (remove periods)
names(pdx_cf) <- gsub(x = names(pdx_cf), pattern = "\\." , "_") %>% tolower()

# HIMS Roof was returning Inf and NaN
pdx_cf %>% na.omit()

# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres()
  ,dbname = db
  ,host = 'pgsql102.rc.pdx.edu' # not sure why object host isn’t working...
  ,port = port
  ,password = pw
  ,user = user)

# Find a better way to do this... append = TRUE is not ideal, but append = FALSE and overwrite = TRUE converts numeric to real datatype!?
# Write to db as lookup table

# clear table before writing new data
res <- dbSendQuery(con, statement = "delete from pdx_cf;")
dbClearResult(res)
res <- dbSendQuery(con, statement = "vacuum full pdx_cf;")
dbClearResult(res)

# Write the new CF data to the pdx_cf table in canopycontinuum
st_write(obj = pdx_cf
  ,dsn = con
layer_options = "OVERWRITE=true"
,drop_table = FALSE
,try_drop = FALSE
,debug = TRUE
,append = TRUE
,overwrite = FALSE
,dataset_options = "GEOMETRY=AS_WKT"
)

# # define global graphics variables

# plot_width <- 14
# plot_height <- 10
# plot_dpi <- 600
#
# # axis_text_size <- 1.2
# # axis_title_size <- 16
# #
# # lower_limit <- 0
# # upper_limit <- 1
# #
# # lm_month$yearmon <- as.yearmon(paste(lm_month$month, lm_month$year)
# , format = "%b %Y")
# #
# # r2m <- ggplot(lm_month, aes(x = as.factor(yearmon), y = r.squared,
# , fill = month)) +
# # geom_boxplot() +
# # theme_bw() +
# # guides(fill = FALSE) +
# # xlab("") +
# # ylab(expression(R^2)) +
# # labs(title = "PurpleAir Correlation by Month") +
# # # facet_wrap(~id)
# # theme(plot.title = element_text(hjust = 0.5, size = 15, face = "
# bold"),
# #     plot.subtitle = element_text(hjust = 0.5, size =12, face = "
# bold"),
# #     axis.text = element_text(size=rel(axis_text_size), face = "
# bold", colour = "black"),
# #     axis.title = element_text(size=axis_title_size, face = "
# bold"),
# #     legend.title = element_text(size = 15, face = "bold"),
# #     legend.text = element_text(size = 12, face = "bold"))
# #
# # r2m
Appendix A. Source Code

```r
# sm <- ggplot(lm_month, aes(x = as.factor(yearmon), y = estimate.m, 
  fill = month)) + 
# geom_boxplot() + 
# theme_bw() + 
# guides(fill = FALSE) + 
# xlab("") + 
# ylab("Slope") + 
# labs(title = "PurpleAir Calibration Slopes by Month") + 
# # facet_wrap(~id) 
# theme(plot.title = element_text(hjust = 0.5, size = 15, face = "bold"), 
#       plot.subtitle = element_text(hjust = 0.5, size = 12, face = "bold"), 
#       axis.text = element_text(size = rel(axis_text_size), face = "bold", colour = "black"), 
#       axis.title = element_text(size = axis_title_size, face = "bold"), 
#       legend.title = element_text(size = 15, face = "bold"), 
#       legend.text = element_text(size = 12, face = "bold")) 

# im <- ggplot(lm_month, aes(x = as.factor(yearmon), y = estimate.b, 
  fill = month)) + 
# geom_boxplot() + 
# theme_bw() + 
# guides(fill = FALSE) + 
# xlab("") + 
# ylab("Intercept") + 
# labs(title = "PurpleAir Calibration Intercepts by Month") + 
# # facet_wrap(~id) 
# theme(plot.title = element_text(hjust = 0.5, size = 15, face = "bold"), 
#       plot.subtitle = element_text(hjust = 0.5, size = 12, face = "bold"), 
#       axis.text = element_text(size = rel(axis_text_size), face = "bold", colour = "black"), 
#       axis.title = element_text(size = axis_title_size, face = "bold"), 
#       legend.title = element_text(size = 15, face = "bold"), 
#       legend.text = element_text(size = 12, face = "bold")) 
```

Appendix A. Source Code

# blank <- grid.rect(gp=gpar(col="white"))
#
# boxplots <- arrangeGrob(r2m, sm, im, nrow = 3, ncol = 1)
# boxplots
# grid.arrange(r2m, sm, im, nrow = 3, ncol = 1)
#
# ggsave(file = paste0("./figures/summary/boxplots.png")
#     ,plot = boxplots
#     ,scale = 1
#     ,width = plot.width
#     ,height = plot.height
#     ,units = "in"
#     ,dpi = plot.dpi)
#
# # Matching Meena’s sensor abbreviations

# long_sensor <- c("byuki_project"
#     ",byuki_project B"
#     ",Haig St."
#     ",Haig St. B"
#     ",Marion Court Apartments"
#     ",Marion Court Apartments B"
#     ",STAR Lab Creston Kenilworth"
#     ",STAR Lab Creston Kenilworth B"
#     ",STAR Lab Powell–Hurst Gilbert"
#     ",STAR Lab Powell–Hurst Gilbert B"
#     ",STAR Lab Aloha"
#     ",STAR Lab Aloha B"
#     ",Lower Boones Ferry Road"
#     ",Lower Boones Ferry Road B"
#     ",STAR Lab — Hillsdale"
#     ",STAR Lab — Hillsdale B"
#     ",Red Fox Hills"
#     ",Red Fox Hills B"
#     ",PSU STAR Lab Hayden Island"
#     ",PSU STAR Lab Hayden Island B"
#     ",PSU STAR Lab Roof North"
#     ",PSU STAR Lab Roof North B"
#     ",PSU STAR Lab Roof South"
#     ",PSU STAR Lab Roof South B"
#     ",PSU STAR Lab Rose City Park"
#     ",PSU STAR Lab Rose City Park B")
#
# # matching Meena’s abbrev. for the sensors she focused on
# short_sensor <- c("byu1"
# byu2
# hag1
# hag2
# mca1
# mca2
# crk1
# crk2
# phg1
# phg2
# alo1
# alo2
# lbf1
# lbf2
# hil1
# hil2
# rfh1
# rfh2
# hay1
# hay2
# sln1
# sln2
# sls1
# sls2
# rcp1
# rcp2

# asdf$id <- mapvalues(asdf$id, from = long_sensor, to = short_sensor)

# asdf %>% filter(id %in% short_sensor)

# # compare total slope, r2, intercept with Meena’s results

# r2_total <- asdf %>%
# group_by(id) %>%
# do(fitTotal = lm(pm2_5~’Portland SE Lafayette’, data = .)) %>%
# glance(fitTotal) %>%
# arrange(desc(r.squared))

# # need a better way to get the slope and intercept without losing
# the std.error from each variable

# slope_total <- asdf %>%
# nest(-c(id)) %>%
# mutate(
# fit = map(data, ~lm(pm2_5~’Portland SE Lafayette’, data = .)),
# tidied = map(fit, tidy)
Appendix A. Source Code

```r
620 # ) %>%
621 # unnest(tidied) %>%
622 # dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
623 # dplyr::select(-c(std.error, statistic, p.value)) %>%
624 # spread(key = term, value = estimate) %>%
625 # dplyr::select(-c(row_id))
626 # filter(term != "(Intercept)") %>%
627 # filter(estimate < 5) %>%
628 # filter(estimate > -5)
629 #
630#
631 # intercept_total <- asdf %>%
632 # nest(-c(id)) %>%
633 # mutate(
634 # fit = map(data, ~lm(pm2_5~'Portland SE Lafayette', data = .)),
635 # tidied = map(fit, tidy)
636 # ) %>%
637 # unnest(tidied) %>%
638 # dplyr::mutate(row_id=1:n()) %>% ungroup() %>%
639 # dplyr::select(-c(std.error, statistic, p.value)) %>%
640 # spread(key = term, value = estimate) %>%
641 # dplyr::select(-c(row_id))
642 # filter(term != "'Portland SE Lafayette'") %>%
643 # filter(estimate < 25) %>%
644 # filter(estimate > -25)
645 #
646 # # combine R2, slope, and intercept into single tidy dataframe
647 # estimates_total <- inner_join(slope_total, intercept_total, by = c("id"),
648 # suffix = c(".m", ".b"))
649 #
650 # names(r2_total)[names(r2_total) == 'p.value'] <- "p.value.r2"
651 # names(r2_total)[names(r2_total) == 'statistic'] <- "statistic.r2"
652 # lm_total <- inner_join(r2_total, estimates_total, by = c("id")) %>%
653 # arrange(id)
654 #
655 # lm_summary <- lm_total %>%
656 # dplyr::select(c(id, r.squared, estimate.m, estimate.b))
657 # write.csv(lm_summary, row.names = FALSE, ".//reports/lm_total_results.csv"
658 #
659 #
660 # # running lm on Vivek’s sensors during August by hand!
661 # top10 <- lm_month %>% arrange(desc(r.squared)) %>% head(10)
```
# a <- asdf %>% filter(id == "STAR Lab Creston Kenilworth" & month == "Aug")
# moda <- lm(a$pm2_5 ~ a$’Portland SE Lafayette’)
# summary(moda)
#
# b <- asdf %>% filter(id == "STAR Lab Creston Kenilworth B" & month == "Aug")
# modb <- lm(b$pm2_5 ~ b$’Portland SE Lafayette’)
# summary(modb)
#
# BEGIN CF EDA

# # the slopes vary for a given sensor, despite having strong correlations...
# gg <- pdx_cf %>%
#   filter(r.squared >= 0.85) %>%
#   dplyr::select(year, month, id, label, r.squared, estimate.m, estimate.b) %>%
#   arrange(label)
# unique(gg$label)
#
# # temporary fix is to just use the CF with the strongest correlation, or take the average?
# max_r2 <- pdx_cf %>%
#   group_by(id, label) %>%
#   slice(which.max(r.squared)) %>%
#   dplyr::select(id, label, r.squared, estimate.m, estimate.b) %>%
#   filter(r.squared >= 0.85) %>% arrange(label)
#
# bad_r2 <- pdx_cf %>%
#   group_by(id, label) %>%
#   slice(which.max(r.squared)) %>% arrange(r.squared) %>%
#   dplyr::select(id, label, r.squared, estimate.m, estimate.b) %>%
#   filter(r.squared < 0.85) %>% arrange(label)
#
# mean_r2 <- pdx_cf %>%
#   group_by(id, label) %>%
#   filter(r.squared >= 0.85) %>%
#   dplyr::select(year = year, month, id, label, r.squared, estimate.m, estimate.b) %>%
#   summarise_if(is.numeric, .funs=mean) %>%
#   arrange(label)
#
# diff <- inner_join(mean_r2, max_r2, by = c("id", "label"), suffix = c(".mean", ".max")) %>%
Appendix A. Source Code

# dplyr::select(id, label, r.squared.mean, r.squared.max, estimate.m.mean, estimate.m.max, estimate.b.mean, estimate.b.max)
#
# diff$r2_diff <- abs(diff$r.squared.max-diff$r.squared.mean)
# diff$slope_diff <- abs(diff$estimate.m.max-diff$estimate.m.mean)
# diff$int_diff <- abs(diff$estimate.b.max-diff$estimate.b.mean)
#
# diff %<>% arrange(desc(slope_diff))
#
# END CF EDA

Source/unionStreets.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018–11–19
# combine RLIS and WA streets data into single shapefile

# needed for nvim-R
if (basename(getwd()) == "src") {
  setwd("../")
  getwd()
}

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr,
  dplyr,
  magrittr,
  tools,
  rgdal,
  raster,
  sp,
  sf,
  rgeos,
  stringr,
  RColorBrewer,
  pbapply,
  parallel)
# Oregon North NAD83 Meters UTM Zone 10
epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Portland Study Area
study_area <- readRDS("./study_area/study_area.RDS")

# Portland Streets (not from the rlis dir, we write there instead)
pdx <- readOGR("./data/pdx_streets/streets.shp") %>% spTransform(CRSobj = epsg_26910)

# Vancouver Streets
vnc <- readOGR("./data/wsdot/WSDOT___Local_Agency_Public_Road_Lines.shp")
vnc %>% spTransform(CRSobj = epsg_26910)
vnc %>% gIntersection(study_area) # does this only return lines that intersect the border of our study area, or also lines that are contained by it?

# Combine them together
x <- raster::union(pdx, vnc) # faster but it might be different...
writeOGR(x, dsn = ". /data/rlis/", layer = "streets", driver = "ESRI Shapefile", overwrite = T)

# using rgeos gUnion
# x <- gUnion(pdx, vnc) # very slow!

# sample data: line lengths
# df <- data.frame(len = sapply(1:length(x), function(i) gLength(x[i, ])))
# rownames(df) <- sapply(1:length(x), function(i) x@lines[[i]]@ID)

# SpatialLines to SpatialLinesDataFrame
# y <- SpatialLinesDataFrame(x, data = df)
# writeOGR(y, dsn = ". /data/rlis/", layer = "streets", driver = "ESRI Shapefile", overwrite = T)

# ran into a bunch of issues with the gdb with rgdal and sf...
# gdb file
# fgdb <- ". /data/wsdot/LAPR.gdb"

# List all feature classes in a file geodatabase
# subset(ogrDrivers(), grepl("GDB", name))
# fc_list <- ogrListLayers(fgdb)
# print(fc_list)

# Read the feature class
van <- readOGR(dsn=fgdb, layer="LAPR_Lines")
Appendix A. Source Code

```r
# van <- readOGR("./data/wsdot/LAPR.gdb") %>% spTransform(CRSobj = epsg_26910)
# van <- readOGR("./data/wsdot/LAPR.gdb", layer = "LAPR_Lines")
# sf <- st_read(fgdb, layer="LAPR_Lines") %>% st_transform(crs=epsg_26910)
# van %>% filter(FederalFunctionalClassDesc %in% c("Rural Minor Collector", "Urban Minor Collector", ))
# study_sf <- st_as_sf(study_area)
# sf <- st_intersection(sf, study_sf)
#

*#

Source/processNDVI.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018–10–18
# process raw hdf MODIS NDVI data, convert to GeoTiff, perform weekly
# , monthly averages

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal ,dplyr ,magrittr ,gdalUtils ,raster ,sp ,sf ,rgeos ,MODIS ,parallel ,ggplot2 ,pbapply
  ,# ,greenbrown # sourceforge repo
}
source("./src/init.R")
source("./src/functions.R")

study_area <- readRDS(study_area)
```

# determine bounding box for portland

# ndvi_path <- "/MODIS/MOD13A2.006"  # 1 km
ndvi_path <- "/MODIS/MOD13Q1.006"  # 250 m

ndvi_files <- list.files(path = ndvi_path, pattern = "\.hdf$",
                          all.files = FALSE, full.names = TRUE, recursive = TRUE)  # grab files from all
subdirectories

percentNA <- function(r) {
  sum(values(is.na(r)))/ncell(r)
}

# Reproject, crop, and resample to match study area
processNDVI <- function(file_path, study_area = study_area, proj = epsg_26910) {

cat("\n")

# extract date from file path
date <- stringr::str_extract(file_path, 
                           '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
date_clean <- stringr::str_replace_all(date, "\.", "-"")

dates <- seq.Date(from = as.Date(date_clean)
                 ,to = as.Date(date_clean) + lubridate::days(15)
                 ,by = "1 day")

sds <- get_subdatasets(file_path)

# ndvi <- sds[grepl("16DAY_1km_VI:1 km 16 days NDVI", sds)]  #
# readGDAL %>% raster
# evi <- sds[grepl("16DAY_1km_VI:1 km 16 days EVI", sds)]  #
# readGDAL %>% raster

ndvi <- sds[grepl("16DAY_250m_500m_VI:250m 16 days NDVI", sds)]  #
# readGDAL %>% raster
evi <- sds[grepl("16DAY_250m_500m_VI:250m 16 days EVI", sds)]  #
# readGDAL %>% raster
# Not sure how to make use of this layer yet... Only applies to Pixel Reliability == 1

# vi_quality <- sds[grepl("16DAY_250m_500m_VI:250m 16 days VI Quality", sds)] %>% readGDAL %>% raster

# 10 other variables (QA, cloud mask, water, climatology, aerosols ?)

# Exclude Low Quality Pixels from NDVI & EVI layers
pix_rel <- sds[grepl("16DAY_250m_500m_VI:250m 16 days pixel reliability", sds)] %>% readGDAL %>% raster

values(pix_rel)[values(pix_rel) == -1] <- NA  # Fill/No Data: Not Processed

values(pix_rel)[values(pix_rel) == 1] <- NA  # Marginal data: Useful, but look at other QA information

values(pix_rel)[values(pix_rel) == 2] <- NA  # Snow/Ice: Target covered with snow/ice

values(pix_rel)[values(pix_rel) == 3] <- NA  # Cloudy: Target not visible, covered with cloud

values(pix_rel)[values(pix_rel) == 0] <- 1  # Good Data: Use with Confidence

ndvi <- ndvi * pix_rel
ev <- evi * pix_rel

print(percentNA(ndvi))
print(percentNA(evi))

# Not useful for annual median values, but might be useful for daily interpolation...
na_threshold <- 1.00  # decimal percent

if (percentNA(ndvi) < na_threshold & percentNA(evi) < na_threshold) {

# combine NDVI and EVI into a single stack
r <- stack(ndvi, evi)
names(r) <- c(paste0("ndvi.", date), paste0("evi.", date))

r <- raster::projectRaster(from = r, method = 'bilinear', crs = proj)

study_area <- spTransform(study_area, CRSobj = proj)

if(!identical(crs(study_area), crs(r))) {

m <- mask(r, study_area)
cr <- crop(m, study_area)

pblapply(dates, function(x) {
pblapply(1:nlayers(cr), function(y) {
  layer_name <- names(cr[[y]])
  var <- gsub("\\..+$", "", layer_name)
  date <- gsub("-", "\\\", x)
  writeRaster(cr[[y]],
    filename = paste0("./output/", var, "/", var, ".", date, ".tif"),
    format = "GTiff",
    overwrite = TRUE)
  }# , cl = detectCores() - 1
})
# , cl = detectCores() - 1
}
}
# Call our processing function
pblapply(ndvi_files, function(x) {
  processNDVI(file_path = x,
    study_area = study_area
  )
  }# , cl = detectCores() - 1
) # , cl = 8

# test our output
ndvi <- stack(list.files(path = "./output/ndvi/", pattern = ".tif$", full.names = T))
plot(ndvi)
evi <- stack(list.files(path = "./output/evi/", pattern = ".tif$", full.names = T))
plot(evi)

# Convert 16-day composites to annual median layers
# Florian's temporal composite example (doesn't work for what I need)
# import 16 day NDVI
# tfs <- readRDS("./data/ndvi_tfs.RDS")
# ndvi <- sapply(tfs[[1]], "[[", 1)
# cdoy <- sapply(tfs[[1]], "[[", 2)
#
# ndvi_day <- temporalComposite(ndvi, cdoy, interval = "day", fun = mean)
# saveRDS(ndvi_day, "./data/ndvi_day_composite.RDS")
# ndvi_day <- readRDS("./data/ndvi_day_composite.RDS")

Source/processGridMET.R

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal,
       rgeos,
       sp,
       raster,
       dplyr,
       magrittr,
       pbapply,
       parallel,
       ncdf4,
       snow)

# needed for nvim-R
if (basename(getwd()) == "src") {
  setwd("./")
  getwd()
}

source("./src/init.R")

# converts gridMet # of days from 1900-01-01 to date (%Y-%m-%d)
formatDateLayer <- function(layer, var) {
  layer %>% stringr::str_replace_all("X", ".") %>%
    as.numeric() %>%
    as.Date(origin = "1900-01-01") %>%
    as.character()
  layer <- paste0(var, ".", layer)
  return(layer)
}

# processes gridMet stack for one variable for one year, saves to RDS (preserves layer names)
processGridMet <- function(file_name,
                           study_area # RDS path
                           , crs # output crs
                           , out_dir # output (rds)
Appendix A. Source Code

```r
# extract variable name and year
var_year <- basename(file_name) %>%
gsub(pattern = "\n", replacement = "")
var <- gsub("_[0-9][0-9][0-9][0-9][0-9]", "", var_year)
cat("\n")
print(paste0("Processing ", var_year))
r <- raster::stack(file_name)
# projectRaster(crs = crs(epsg_26910))
names(r) <- lapply(names(r), function(x) {
  formatDateLayer(layer = x,
    var = var)
})
# universal study area boundary box
x <- readRDS(study_area) %>% spTransform(CRSTobj = crs)
# create larger buffer before masking/cropping
y <- spTransform(CRSTobj = crs(r))
m <- mask(r, y, getCover = T) # masking on x excludes a few cells!
cr <- crop(m, extent(y))

# project using ngb for wind direction, bilinear interpolation for everything else
if(var == "th") {
  # project to epsg_26910 @ 1 km resolution
  cr %>% projectRaster(crs = crs, method = "ngb", res = sp_res)
    mask(x, getCover = T) %>%
    crop(extent(x))
} else {
  # project to epsg_26910 @ 1 km resolution
  cr %>% projectRaster(crs = crs, method = "bilinear", res = sp_res)
    mask(x, getCover = T) %>%
    crop(extent(x))
}

# Create sub-directory if necessary
sub_dir <- paste0(out_dir, "/", var)
if(!dir.exists(sub_dir)) {
  print(paste0("Creating sub-directory: ", sub_dir))
dir.create(sub_dir)
}

writeRaster(cr, file.path(paste0(out_dir, "/", var, "/", names(cr), ".tif"))
  , format = "GTiff"
  , overwrite = TRUE
  , bylayer = T
```
Appendix A. Source Code

87  \)
88  \textbf{rm(r, m, cr, x, y)}
89  \textbf{gc()}
90
91
92
93  \# \textbf{testing our function on a single file}
94  \# \textbf{processGridMet(file\_name = \texttt{\"gridMET/nc/pr\_2017.nc\"}}
95  \#  \textbf{, study\_area = \texttt{\"./study\_area/study\_area.RDS\"}}
96  \#  \textbf{, crs = crs}
97  \#  \textbf{, out\_dir = \texttt{\"./gridMET/\"}}
98  \#)
99
100
101  \texttt{grid\_Met\_Files \textless{}= list.files(path = \texttt{\"./gridMET/nc\"}, pattern = \texttt{\".nc\"},}
102  \texttt{full.names = T)}
103
104  \textbf{pblapply(grid\_Met\_Files, function(x) \{}
105  \textbf{processGridMet(file\_name = x}
106  \textbf{, study\_area = \texttt{\"./study\_area/study\_area.RDS\"}}
107  \textbf{, crs = crs}
108  \textbf{, out\_dir = \texttt{\"./data\"})}
109  \textbf{, cl = detect\_Cores()\textemdash{}1 \# Memory allocation error in parallel!}
110  \})

Source/processNLCD.R

1  \# Created by Philip Orlando @ Sustainable Atmospheres Research Lab
2  \# 2019–01–30
3  \# Transform NLCD into binary raster for each UHI campaign
4
5  \# Load Dependencies

6  \# Conditionally install \texttt{pacman} library if we can’t already load it
7  \textbf{if(!suppressPackageStartupMessages(\texttt{require(pacman)) \{}}
8  \textbf{install\_packages(\texttt{\textasciitilde{}pacman\texttt{)}}}
9  \textbf{library(pacman)}
10
11
12
13  \# Use the \texttt{pacman::p\_load()} to handle package installation/loading
14  \textbf{p\_load(raster # For working with raster data}
15  \textbf{, sp # Spatial Library}
16  \textbf{, sf # Simple Features Library}
17  \textbf{, raster\_Vis # Plot nice looking rasters}
18  \textbf{, rgdal # Geodata abstraction library for reading/writing}
19  \textbf{spatial data}
Appendix A. Source Code

```
, rgeos # Geometry engine open source for performing spatial transformation
, dplyr # Grammar of data manipulation
, magrittr # Provides the pipe operator (%>%)
, parallel # used for detectCores()
, pbapply # Multithreading with progress bar
 #, gdalUtils # Extra GDAL utilities
, snow # Another library for parallel computing
 #, geojsonio # Working with geojson
)

# Global Variable Definition

source("./src/init.R")
source("./src/functions.R")

# Set tmpdir for raster data
setRasterTempDir()

study_area <- readRDS("./study_area/study_area.RDS")

# Reading in our NLCD Layer

nlcd <- raster(list.files("./data/nlcd_2011_landcover_2011_edition_2014_10_10",
                           pattern = "\.img$",
                           full.names = T))

# Function Definition

processNLCD <- function(nlcd, study_area, outdir) {

  # Grab the CRS of the ambient temperature raster
  study_crs <- proj4string(study_area)

  # Let’s convert our raster into a bounding box vector of its extent
  # This will speed up our NLCD cropping
  bb <- as(extent(study_area), "SpatialPolygons")
  proj4string(bb) <- proj4string(study_area)

  # Buffer study region to accommodate distortion/edge effects from reprojecting later
  bb_buf <- gBuffer(bb, width = 10^5)
```
# Reproject ambient temperature to match NLCD
bb_nlcd <- spTransform(bb_buf, CRSobj = proj4string(nlcd))

# Make sure the projections are the same before cropping
if (identical(crs(bb_nlcd), crs(nlcd))) {
  cr <- crop(nlcd, bb_nlcd)
  # plot(cr)
  m <- mask(cr, bb_nlcd)
  # plot(m)
} else {
  stop("NLCD and BB CRS are not identical!")
}

beginCluster()

r <- projectRaster(m, crs = study_crs, res = sp_res, method = 'ngb')

r %>% crop(bb) %>% mask(bb)

endCluster()

# Add the attribute table back (it is lost during projection)
r@data@attributes <- m@data@attributes

# Retrieve color legend from NLCD data (it is also lost during projection)
r@legend@colortable <- m@legend@colortable

# Convert to stack of binary landcover layers
s <- binaryLandCover(r)

# Visually inspect the transformation
# levelplot(s, colorkey = F)

# Create an output directory if it doesn’t already exist
if (!dir.exists(outdir)) {
  dir.create(outdir)
}

# Save our transformed NLCD stack as a GeoTiff
writeRaster(s, file.path('./data', 'nlcd', paste0(names(s), '.tif')), overwrite=T, bylayer=T, format="GTiff")

# Save our transformed NLCD stack as an RDS to preserve the layer names!
# saveRDS(s, paste0(outdir, 'nlcd', study_region, '_', trav, '.RDS'))
Appendix A. Source Code

```r
# clean up
gc()
}

# Retrieves attribute names from NLCD
getAttributes <- function(nlcd) {
  # Let's create a lookup table for our landcover attributes
  metadata <- data.frame(id = nlcd@data@attributes[[1]]$ID,
                         attribute = as.character(nlcd@data@attributes[[1]]$NLCD.2011.Land.Cover.Class))

  # There are a lot of missing values that we should clean up
  metadata %<>% droplevels(exclude = "") %>% na.omit()

  # Let's clean up the attribute names while we're at it
  # First let's convert everything to lower case
  metadata$attribute <- tolower(metadata$attribute)

  # Next, let's replace any spaces and forward slashes with underscores
  metadata$attribute <- gsub(" /\", "_", metadata$attribute)

  # Finally, let's delete any commas!
  metadata$attribute <- gsub("", "", metadata$attribute)

  return(metadata)
}

# Define one-hot encoding function for NLCD raster
binaryLandCover <- function(nlcd) {
  # Retrieve number of unique landcover values
  id <- unique(getValues(nlcd)) %>% sort()

  # Retrieve landcover attributes metadata
  metadata <- getAttributes(nlcd)

  # Join the landcover attribute names to the observed landcover id values
  attributes <- merge(as.data.frame(id), metadata)
```

Appendix A. Source Code

# Loop through each landcover and create presence/absence layer then name it by LC code
s <- pblapply(id, function(x) {
    # Create a new raster layer for each landcover class
    # r <- assign(paste0("r", x), nlcd)
    r <- nlcd
    # Assign 1 if cell value matches landcover code, 0 otherwise
    r[] <- ifelse(r[] == x, 1, 0)
    # attach landcover code to layer name
    names(r) <- paste0(attributes[id == x][2])
    return(r)
})
# run our loop on multiple cores
,cl = detectCores() - 1
) %>%
    # combine each presence/absence raster into a stack
    stack()
}

# Create NLCD for Jackson's Lab5
# processTraverse(nlcd = nlcd, study_region = "lab5", trav = "am",
# outdir = ".\data\example\"

# Main processing function
# processNLCD <- function(nlcd, study_regions, travs, outdir) {
#    # # loop through each study region
#    # pblapply(study_regions, function(x) {
# # # loop through each traverse for each study region
# # pblapply(travs, function(y) {
# # # run our main processing function!
# # try(processTraverse(nlcd = nlcd, study_region = x, trav = y,
# outdir = outdir))
# # })
# # })
# # }

# Calling our functions
Appendix A. Source Code

processNLCD(nlcd = nlcd,
    ,study_area = study_area
    ,outdir = ". /data/nlcd/
)

# # Visually inspect the transformed data
#
# lapply(study_regions, function(x) {
# lapply(travs, function(y) {
# try({
# s <- readRDS(list.files("./data/nlcd", pattern = paste0(x, "_",
# ), full.names = T))
# levelplot(s, colorkey = F)
# })
# })
# })

Source/processRoads.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018-11-16
# convert road network vector to raster road density
#
# needed for nvim-R
if (basename(getwd()) == "src") {
  setwd("../")
}
getwd()

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr,
  ,dplyr
  ,magrittr
  ,tools
  ,rgdal
  ,raster
  ,sp
  ,sf
  ,rgeos
source("./src/init.R")

# Oregon North NAD83 Meters UTM Zone 10
epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Rasterize RLIS road data

# list of shape files in rlis
layers <- list.files("./data/rlis", pattern = "\.shp\$, full.names = T)

# rs <- readRDS("./study_area/study_area.RDS") %>% raster(res = 50, crs = crs)
# values(rs) <- 1
# layer <- layers[1]

processRoads <- function(layer = NULL # shapefile name w/ extension
,outdir = ".\data/" # output directory
,crs = epsg_26910 # spatial reference system
            in meters
, res = sp_res # raster resolution in meters
,study_area = "./study_area/study_area.RDS"
) {

  var <- file_path_sans_ext(basename(layer))
  outfile <- paste0(outdir, var, "/\", var, ".tif")
  subdir <- dirname(outfile)

  if(!dir.exists(subdir)) {
    print(paste0("Creating ", subdir, " subdirectory"))
    dir.create(subdir)
  }

  if(!file.exists(outfile)) {
    print(paste0(outfile, " is being processed."))
  }
}
# read in shapefile and reproject
vec <- readOGR(layer) %>% spTransform(CRSobj = crs)
rs <- readRDS(study_area) %>% raster(res = sp_res, crs =
projection(vec))
rs[] <- 1:ncell(rs)
print("rasterToPolygons")
rsf <- rasterToPolygons(rs)
# print("sp::polygonize")
# rsf <- sp::polygonize()
# rp <- intersect(vec, rsf) # I think this is causing memory error?
print("st_intersection")
rp <- st_intersection(st_as_sf(vec), st_as_sf(rsp))
print("as Spatial")
rp <- as(rp, "Spatial")

# Memory expensive!
print("rgeos density")
if ("SpatialLinesDataFrame" %in% class(vec)) {
  rp$density <- gLength(rp, byid=T)
} else if ("SpatialPolygonsDataFrame" %in% class(vec)) {
  rp$density <- gArea(rp, byid=T)
} else if ("SpatialPointsDataFrame" %in% class(vec)) {
  print("fix this to work with point layers later!")
  # girp$density <- gIntersection(rsp, vec, byid=T) %>% length()
  %>% as.integer()
  rp$density <- sp::over(vec, rsp)
  # rp$density <- gContains(rp, byid=T)
}

print("tapply")
z <- tapply(rp$density, rp$layer, sum)
r <- raster(rs)
r[as.integer(names(z))] <- z

a <- r
values(a)[values(a)<=0] <- NA
# values(a)[values(a)>0] <- 100
# values(a)[values(a)<=0] <- 100

pal <- colorNumeric(c("#0C2C84", "#41B6C4", "#FFFFCC"), values(a)
  ,
  na.color = "transparent")

leaflet() %>% addProviderTiles(providers$Stamen.TonerLite) %>%
addRasterImage(a, colors=pal) %>%
addLegend(pal = pal, values = values(a),
  title = "Road Density")
# write out
writeRaster(r
    , filename = outfile
    , format = "GTiff"
    , overwrite = T
)
#

# Calling our function
print('begin tapply loop')
pblapply(layers, function(x) processRoads(layer = x)
    , cl = detectCores()−1 # only run in parallel on Hecate
)
#

# # refactor
# vectorToRaster <- function(layer = NULL # shapefile name w/
# extension
#    , indir = "/data/rlis/" # directory
#    , outdir = "/data/" # output directory
#    , crs = epsg_26910 # spatial reference
#    , res = 50 # raster resolution in meters
#    , study_area = "/study_area/study_area.RDS"
# ) {
# # var <- file_path_sans_ext(layer)
# # outfile <- paste0(outdir, var, "/", var, ".tif")
# # if(!file.exists(outfile)) {
# # print(paste0(outfile, " is being processed."))
# # # read in shapefile and reproject
# # vector <- readOGR(dsn = indir, layer = var)
# # study_area <- readRDS(study_area) %>% spTransform(CRS(crs))
# # # check that we have a polylines feature class
# # if("SpatialLinesDataFrame" %in% class(vector)) {
# # # reproject vector

```r
# lines_utm <- spTransform(vector, CRS(crs))
# plot(study_area)
# plot(lines_utm, add = T)
# create empty raster
lines_utm_rst <- raster(extent(study_area), crs = crs(study_area), res = res)

# determine the length of lines in each cell
lengths <- pbsapply(1:ncell(lines_utm_rst), function(x) {
  # create placeholder raster
tmp_rst <- lines_utm_rst
  # fill in placeholder value
tmp_rst[x] <- 1
  # convert to vector
tmp.vec <- rasterToPolygons(tmp_rst)
  # print(x)
  # calculate the length of lines within each cell
  if (gIntersects(lines_utm, tmp.vec)) {
    lines_utm_crp <- crop(lines_utm, tmp.vec)
    lines_utm_crp_len <- gLength(lines_utm_crp)
    return(lines_utm_crp_len)
  } else {
    return(0)
  }
}, cl = detectCores() - 1)

# set the cell value to the road density
lines_utm_rst[] <- lengths / 1000 # convert to km

# write out
writeRaster(lines_utm_rst
  , filename = outfile
  , format = "GTiff"
  , overwrite = T
)

# spplot(lines_utm_rst, scales = list(draw = TRUE), xlab = "x",
#        ylab = "y",
#        col.regions = colorRampPalette(brewer.pal(9, "YlOrRd”)),
#        sp.layout = list("sp.lines", lines_utm))

if("SpatialPolygonsDataFrame" %in% class(vector)) {
  poly_utm <- spTransform(vector, CRS(crs))
  # create empty raster
```
poly_utm_rst <- raster(extent(study_area), crs = projection(study_area), res = res)

# determine the length of poly in each cell
areas <- pbsapply(1:ncell(poly_utm_rst), function(x) {
    # create placeholder raster
    tmp_rst <- poly_utm_rst
    # fill in placeholder value
    tmp_rst[x] <- 1
    # convert to vector
    tmp_vec <- rasterToPolygons(tmp_rst)
    # print(x)
    # calculate the area of poly within each cell
    if (gIntersects(poly_utm, tmp_vec)) {
        poly_utm_crp <- crop(poly_utm, tmp_vec)
        poly_utm_crp_area <- gArea(poly_utm_crp)
        return(poly_utm_crp_area)
    } else {
        return(0)
    }
}, cl = detectCores() - 1)

# set the cell value to the road density
poly_utm_rst[] <- areas

# write out
data = list(draw = TRUE), scales = list(draw = TRUE), xlab = "x",
ylab = "y", col.regions = colorRampPalette(brewer.pal(9, "YlOrRd")),
sp.layout = list("sp.lines", poly_utm))

# except when outFile already exists!


# }  

# # Very coarse, not going to work...  
# pop_r <- rasterize(x = acs_sp, y = template_r, field = "B0001_001E ", fun=mean, mask=F, na.rm=T)  

# leaflet() %>% addProviderTiles(providers$Stamen.TonerLite) %>%  
# addRasterImage(r, colors=pal) %>%  
# addLegend(pal = pal, values = values(r)),  
# title = "Road Density")  

# # SpatialLinesLengths method  
# rrst.poly <- rasterToPolygons(rs)  
# rp <- gIntersection(vec, rrst.poly, byid = T)  
# rp <- SpatialLinesDataFrame(rp, data.frame(row.names=sapply(slot(rp 
# , "lines"),  
# function(y) slot(y, "ID")), ID=1:length(rp),  
# length=  
# SpatialLinesLengths(rp)/1000))  
#  
# rd.rst <- rasterize(rp, rs, field="length", fun="sum")  

# The slow way  

# layers <- list.files("./data/rlis").shp$  
#  
# call our processing function  
# pbapply(layers, function(x) {  
# vectorToRaster(x)  
# })  

# # visually inspecting our newly cropped rasters  
# streets <- raster("./data/streets/streets.tif")  
# spplot(streets)  
# extent(streets)  
#  
# fwy <- raster("./data/fwy/fwy.tif")  
# spplot(fwy)  
# extent(fwy)  
#  
# arterial <- raster("./data/arterial/arterial.tif")  
# spplot(arterial)  
# extent(arterial)  
#  
# maj_art <- raster("./data/maj_art/maj_art.tif")
# spplot(maj_art)
# extent(maj_art)
#
# railroad <- raster("./data/railroad/railroad.tif")
# spplot(railroad)
# extent(railroad)
#
# railyards <- raster("./data/railyards/railyards.tif")
# spplot(railyards)
# extent(railyards)
#
# s <- stack(fwy, arterial, maj_art, streets, railroad, railyards)
# plot(s)
#
# Deprecated!
# # list rasters that need to be cropped
# r_names <- list.files("./data/rlis", pattern = "\.tif$", full.names=T)
# r_names
#
# file_path <- r_names[1]
# study_area <- "./study_area/study_area.RDS"
# crs <- epsg_26910
#
# # function that crops/masks raster layer by study area
# cropRaster <- function(file_path, study_area, crs) {
#   r <- raster(file_path)
#   x <- readRDS(study_area) %>% spTransform(CRSobj = crs(crs))
#   extent(r)
#   m <- mask(r, x, getCover = T)
#   extent(m)
#   # m <- r
#   extent(x)
#   cr <- crop(m, extent(x), snap="near")
#   extent(cr)
#   writeRaster(cr
#                 ,filename = file_path
#                 ,format="GTiff"
#                 ,overwrite=T
#                 )
# }
#
# calling our function
# pblapply(r_names, function(x) {
#   cropRaster(file_path = x
#             ,study_area = "./study_area/study_area.RDS"
#             )
# })
Appendix A. Source Code

Source/processPopDensity.R

# created by Philip Orlando
# Sustainable Atmospheres Research Lab

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(readr,
  rlang,
  ggplot2,
  plyr,
  dplyr,
  tidyverse,
  magrittr,
  rgeos,
  rgdal,
  sp,
  leaflet,
  sf,
  raster,
  mapview,
  tidycensus
)

source("./src/init.R")
source("./src/functions.R")

study_area <- readRDS("./study_area/study_area.RDS")
# GPW v4 data

gpw <- raster("./data/pop/gpw_v4_population_density_adjusted_to_2015_unwpp_country_totals_rev11_2015_30_sec.tif")
f <- paste0(tempdir(), "/", names(gpw), ".tif")
writeRaster(gpw, f, format = "GTiff", overwrite=T)
e <- bbox(extent(study_area))
tr <- rep(sp_res, 2)
# tr <- rep(1000, 2)
gdalUtils::gdalwarp(f, s_srs=proj4string(gpw), t_srs=proj4string(study_area), sub('\.tif', '_clipped.tif', f), tr=tr,
r='near', te=c(e), multi = TRUE, overwrite = TRUE)

r <- raster(list.files(tempdir(), "_near.*\.tif$", full.names=T))
print("Renaming clipped/projected layers")
names(r) <- sub('_clipped\$', '', names(r))
levelplot(r, margin=F)

lapply(c("near", "cubicspline", "bilinear", "cubic", "lanczos", "average"), function(i) {
  if(i == "near") {
    tr <- rep(1000, 2)
  } else {
    tr <- rep(50, 2)
  }
  cr <- gdalUtils::gdalwarp(f, s_srs=proj4string(gpw), t_srs=proj4string(study_area), sub('\.tif', paste0('_{i}_clipped\.tif'), f), tr=tr,
r=i, te=c(e), multi = TRUE, overwrite = TRUE)
  names(cr) <- paste0("gpw_{i}_" , unique(tr) , "m")
writeRaster(cr, filename = paste0("./data/pop/", names(cr), ".tif"), format = "GTiff", overwrite=T)
  if(i == "bilinear") {
    writeRaster(cr, filename = "/data/pop/pop_dens.tif", format = "GTiff", overwrite=T)
  }
  rasterVis::levelplot(cr, main = names(cr), margin=F)
  print(paste0("Processing ", names(cr)))
})
r <- raster("./data/pop/gpw_near_1000m.tif")
s <- stack(list.files("./data/pop", "50m\..tif\$", full.names = T))
plot(gpw, margin=F)
library(lattice)
levelplot(r, margin=F)
levelplot(s)

# get acs data

## grab our census key in the root directory
KEY <- scan("census_key.pgpass", what = "")
census_api_key(KEY, install = TRUE, overwrite = TRUE)
readRenviron("~/.Renviron")

## reading in census data
acs_or <- get_decennial(year = 2010 # unquote on linux
  ,geography = "block"
  ,variables = "B00001_001E" # unweighted total population
  ,state = "OR"
  ,county = c("Multnomah"
    ,"Washington"
    ,"Clackamas"
    ,"Yamhill"
    ,"Columbia"
    ,"Marion"
    ,"Linn"
    ,"Hood River"
    ,"Wasco"
    ,"Benton"
    ,"Polk"
    ,"Tillamook"
    ,"Clatsop"
    ,"Lincoln"
  ) # counties taken from wiki sacra metro area page
  ,key = KEY
  ,output = "wide"
  ,geometry = TRUE
  ,cb = F
  ,tigris_use_cache = TRUE
) na.omit()

acs_wa <- get_acs(year = 2017 # unquote on linux
  ,geography = "block group"
  ,variables = "B00001_001E" # unweighted total population
  ,state = "WA"
  ,county = c("Clark"
    ,"Cowlitz"
    ,"Skamania"
  )
```r
# Acquire the Waikum, Pacific, Lewis, Klickitat, and
# key = KEY
# output = "wide"
# geometry = TRUE
# cb = F
# tigris_use_cache = TRUE
# ) %>% na.omit()

# acs <- rbind(acs_or, acs_wa)
#
# # # Checkpoint get_acs for hecate
# # saveRDS(acs, "./output/acs.RDS")
# #
# acs <- readRDS("./output/acs.RDS")
# #
# # reproject into meters
# acs <- st_transform(acs, epsg_26910)
# #
# # Now we're creating a logical vector of geometries whose areas are
# # aren't greater than zero
# isAboveZero <- as.vector(sf::st_area(acs)) > 0
# #
# # Let's inspect the number of zero area geometries in Travis County,
# # table(isAboveZero)
# #
# # Now let's convert our sf object into a SpatialPolygonsDataframe,
# # excluding any zero area polygons!
# # acs_sp <- as(acs[isAboveZero,], "Spatial")
# #
# # # Trying out the rasterize function

# template_r <- raster("./data/ndvi/ndvi.2017.01.01.tif") # placeholder raster
# pop_r <- rasterize(x = acs_sp, y = template_r, field = "B00001_001E",
# fun=mean, mask=F, na.rm=T)
# spplot(acs_sp["B00001_001E"])
# #
# rasterVis::levelplot(pop_r, margin=F)
```
# # Read in fwy data for reference
# fwy <- readOGR("./data/rlis/fwy.shp") %>% spTransform(CRSobj = crs(pop_r)) %>% crop(pop_r) %>% fortify()
# df_r <- as.data.frame(pop_r, xy = T)
# df_r$pop_dens <- as.numeric(df_r$layer)
# ggplot() +
# geom_raster(data = df_r,
# aes(x = x, y = y, fill = pop_dens)) +
# scale_fill_viridis_c(option = "A") +
# geom_path(data = fwy, aes(x = long, y = lat, group = group),
# color = "grey60") +
# theme_minimal()

# Using custom density raster function

acsToRaster <- function(vector = NULL, study_area = study_area, buf_res = sp_res, res = sp_res, crs = epsg_26910) {

# # create an empty raster
# template_rst <- raster(extent(gBuffer(study_area, width = buf_res))
# ,crs = projection(study_area), res = sp_res)

# # determine the density of a feature in each cell
# density <- pbsapply(1:nCell(template_rst), function(x) {
# # create placeholder raster
# tmp_rst <- template_rst
# # add placeholder value
# tmp_rst[x] <- 1
# # convert single cell extent to vector
# tmp_vec <- rasterToPolygons(tmp_rst)
# # pbapply(X = vector, MARGIN = 1, FUN = function(y) {
# # Check to see if the feature is within the raster cell
# if (gIntersects(vector[y], tmp_vec)) {
# # Crop feature by the extent of a single raster cell
# vector_crp <- crop(vector, tmp_vec)
# # Perform different calculation for Points, Lines, and Polygons

}
if ("SpatialLinesDataFrame" %in% class(vector)) {
  cell_value <- gLength(vector_crp)
} else if ("SpatialPolygonsDataFrame" %in% class(vector)) {
  cell_value <- gArea(vector_crp)
} else if ("SpatialPointsDataFrame" %in% class(vector)) {
  cell_value <- gIntersection(vector_crp, tmp_vec) %>%
    length() %>% as.integer()
} else {
  print("Object not of class Spatial*DataFrame!")
  next
}
return(cell_value)

# If the feature does not intersect with the cell, assign 0
return(0)

,cl = detectCores() - 1

# set the cell value to the kernel density
template_rst[] <- density

# write to file
writeRaster(template_rst
, filename = outfile
, format = "GTiff"
, overwrite = T)

# list of shape files in rlis
layers <- list.files("./data/rlis", pattern = "\..shp$")

# call our processing function
pblapply(layers, function(x) {
  vectorToRaster(x)
})

#
Appendix A. Source Code

```r
# # determine the bounding box of the city boundary
# bb <- bbox(study_area)
#
# # specify the desired cell size
# cs <- c(sp_res, sp_res) # 2km x 2km
#
# # cell offset and starting point (bottom right corner?)
# cc <- bb[, 1] + (cs/2) # cell offset, centering bottom right corner
# at the halfway point
#
# # number of cells per direction
# cd <- ceiling(diff(t(bb))/cs)
#
# # convert to GridTopology
# grd <- GridTopology(cellcentre.offset = cc, cellsize = cs, cells.
dim = cd)
# # grd
#
# # convert to spatial grid dataframe
# sp_grd <- SpatialGridDataFrame(grd, data = data.frame(id=1:prod(cd)
# proj4string = CRS(proj4string(study_area))))
#
# # convert from SpatialGridDataFrame to polygons for gIntersection()
# with city boundary
# print("convert to grid sp")
# grid_poly <- as(sp_grd, "SpatialPolygonsDataFrame")
#
# # clipping grid by the city boundary
# # grid <- grid_poly[ sacra , ]
# # grid <- grid_poly
#
# # convert to sf object
# print("convert to grid sf")
# grid <- st_as_sf(grid_poly)
#
# # iterate through each grid cell to get the area-weighted
# population from the block group data
# print("begin pblapply!")
# df <- Reduce(rbind, pblapply(1:nrow(grid), function(i) {
# #
# # capture each row of our grid dataframe
# # cell <- grid[i, ]
# #
# # store cell id variable
```
# cell_id <- cell$id

# determine the grid cell area (4 km^2)
# cell_area <- as.numeric(st_area(cell$geometry))

# row_cell <- Reduce(rbind, lapply(1:nrow(acs), function(j) {
  # capture each row of our block group data
  bg <- acs[j, ]
  # determine if a block group intersects with a grid cell
  overlap <- st_intersects(cell$geometry, bg$geometry)[[1]]
  # calculate relative
  if(!is_empty(overlap)) {
    # bg_area <- as.numeric(st_area(bg$geometry))
    relative_area <- as.numeric(st_area(st_intersection(cell$geometry, bg$geometry)))
    # bg_pop <- bg$B00001_001E
    weighted_pop <- bg_pop * (relative_area/bg_area)
    bg_name <- bg$NAME
    # print each row to console
    print(paste(cell_id, bg_name, weighted_pop, bg_pop))
    # row_bg <- data.frame(grid_cell_id = cell_id
    # ,bg_name = bg_name
    # ,bg_pop = bg_pop
    # ,relative_area = relative_area
    # ,bg_area = bg_area
    # ,cell_area = cell_area
    # )
    return(row_bg)
  }
  # store each row in our output dataframe
  # output_df = rbind(output_df
  # ,data.frame(grid_cell_id = cell_id
  # ,bg_name = bg_name
  # ,weighted_pop = weighted_pop
  # ,bg_pop = bg_pop
  # ,relative_area = relative_area
  # ,bg_area = bg_area
  # ,cell_area = cell_area
  # ,stringsAsFactors=FALSE)
  # return(output_df)
# } } # return (row_cell) # } #, cl = detectCores() - 1 # ) # saveRDS(df, ".output/pop_dens.RDS") # stop("Hecate Processing complete")

# # create any empty dataframe
# output_names <- c("grid_cell_id"
# , "bg_name"
# , "weighted_pop"
# , "bg_pop"
# , "relative_area"
# , "bg_area"
# , "cell_area"
# )

# output_df <- data.frame(matrix(ncol = length(output_names)
# , nrow = 0))
# colnames(output_df) <- output_names
# print("Begin for loop :/")
# for(i in 1:nrow(grid)) {
# # capture each row of our grid dataframe
# cell <- grid[i, ]
# # store cell id variable
# cell_id <- cell$id
# # determine the grid cell area (4 km^2)
# cell_area <- as.numeric(st_area(cell$geometry))
# # iterate through each block group
# for(j in 1:nrow(acs)) {
# # capture each row of our block group data
# bg <- acs[j, ]
# # determine if a block group intersects with a grid cell
# overlap <- st_intersects(cell$geometry, bg$geometry)[[1]]
# # calculate relative
# if(!is_empty(overlap)) {
# #


# bg_area <- as.numeric(st_area(bg$geometry))
# relative_area <- as.numeric(st_area(st_intersection(cell$geometry, bg$geometry)))
# bg_pop <- bg$B00001_001E
# weighted_pop <- bg_pop*(relative_area/bg_area)
# bg_name <- bg$NAME

# print each row to console
# print(paste(cell_id, bg_name, weighted_pop, bg_pop))

# store each row in our output dataframe
output_df = rbind(output_df

# output_df = data.frame(grid_cell_id = cell_id
# bg_name = bg_name
# weighted_pop = weighted_pop
# bg_pop = bg_pop
# relative_area = relative_area
# bg_area = bg_area
# cell_area = cell_area
# ,stringsAsFactors=FALSE)

# sum each grid cell’s weighted pop for each intersecting bg
grid_pop <- output_df %>%
  group_by(grid_cell_id) %>%
  summarise(population = sum(weighted_pop))

# saveRDS(grid_pop, "./output/pop_dens/pop_dens.RDS")

# append pop data back to the original grid data
grid <- left_join(grid, grid_pop, by = c("id" = "grid_cell_id"))

# convert to sp class for leaflet?
grid_sp <- as(grid, "Spatial")
projection(grid_sp)
grid_sp <- spTransform(grid_sp, CRSobj = CRS(wgs_84))
projection(grid_sp)

# Convert to raster?
saveRDS(grid_sp, "./output/pop_dens/grid_sp.RDS")
# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal, rgeos, sp, ncd4, raster, rasterVis, dplyr, magrittr, pbapply, parallel, gdalUtils, tools, pryr, snow)

# Read in global variables
source("./src/init.R")

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

formatLayerName <- function(layer) {
  # layer %>% stringr::str_replace_all("X", "") %>
  # as.POSIXct(format = "%Y.%m.%d.%H.%M.%S")
  layer %>% stringr::str_replace_all("X", "")
  layer <- paste0("pbl.", layer)
  return(layer)
}

# GDAL WARP
warpStack <- function(stack, study_region) {
  print(paste0("Writing GeoTiffs for gdalwarp to ", tempdir()))
  ff <- paste0(tempdir(), "/", names(stack), ".tif")
  pbapply(ff, function(f) {
    layer <- stack[[basename(file_path_sans_ext(f))]]
    writeRaster(layer, f, format = "GTiff", overwrite=T)
  }, cl = detectCores() - 1)
Appendix A. Source Code

```r
eg <- bbox(extent(study_region))
tr <- c(sp_res, sp_res)
print("Begin gdalwarp")
pblapply(ff, function(f) {
  gdalUtils::gdalwarp(f, s_srs=proj4string(stack), t_srs=
                   proj4string(study_region), sub('\\.tif', '_clipped.tif', f), tr=tr,
            r='bilinear', te=c(e), multi = TRUE,
            overwrite = TRUE)
}
,cl = detectCores() - 1
)
rm(stack, e, tr, ff, study_region)
gc()
print(paste0("Reading in clipped/projected stack from ", tempdir()))
cr <- stack(list.files(tempdir(), "_clipped\.tif$", full.names=T))
print("Renaming clipped/projected layers")
names(cr) <- sub('_clipped\', '', names(cr))
return(cr)
}

# assigns mean value of entire layer to each cell
meanLayer <- function(stack) {
  # Average our buffer to get the best approximation of regional PBL
  print("Calculate meanLayer")
pblapply(1:nlayers(stack), function(x) {
    r <- stack[[x]]
    r[] <- raster::cellStats(r, stat='mean', na.rm=T) %>% as.numeric()
    writeRaster(r, paste0(tempdir(), "/", names(r), "_mean.tif"),
                overwrite=T, format="GTiff")
    rm(r)
    gc()
  }, cl = detectCores() - 1 # mem limit!
}
print(paste0("Reading in meanLayer stack from ", tempdir()))
s <- stack(list.files(tempdir(), "_mean\.tif$", full.names=T))
print("Renaming meanLayer stack")
names(s) <- sub('_mean\', '', names(s))
return(s)
}
```
# Extract min/max PBL each date
pblMinMax <- function(pbl_stack) {

  print("Calculate pblMinMax")

  # Extract timestamps from layer names
  timestamps <- names(pbl_stack) %>% gsub(pattern = "^pbl\.", replacement = "") %>% lubridate::ymd_hms()

  # Convert to vector of unique dates
  dates <- as.Date(timestamps) %>% gsub(pattern = "\-", replacement = ".") %>% unique()

  # Loop through each date to find the min/max PBL layers
  pblapply(dates, function(x) {

    # Find the row indices for a given date
    date_index <- grepl(paste0("pbl\.", x, "."), names(pbl_stack))

    # Extract these layers from the stack
    date_stack <- pbl_stack[[which(date_index)]]

    # Find the min and max layers for this date
    max_pbl <- max(date_stack)
    min_pbl <- min(date_stack)

    # Stack the results
    min_max <- stack(min_pbl, max_pbl)
    rm(min_pbl, max_pbl)
    gc()

    # Assign new names
    names(min_max) <- c(paste0("pbl.min.", x), paste0("pbl.max.", x))

    # Write to file
    print(paste0("Writing ", names(min_max), " to tmp file in ", tempdir()))
    writeRaster(min_max, paste0(tempdir(), "/", names(min_max), "_minmax.tif"), overwrite=T,
                format="GTiff", bylayer=T)

    # Garbage collection
    rm(date_stack, date_index, min_max)
    gc()
  }, cl = detectCores() -1)
Appendix A. Source Code

```r
print(paste0("Reading in min/max PBL stack from ", tempdir()))
s <- stack(list.files(tempdir(), "_minmax\.tif\$", full.names=T))
print("Renaming min/max PBL stack")
names(s) <- sub("_minmax\$", '', names(s))

# Garbage collection
rm(timestamps, dates)
gc()

return(s)
```

```r
processPBL <- function(file_name, study_area, crs, out_dir) {
  # Read in data
  r <- raster::stack(file_name)

  # Format layer names
  names(r) <- lapply(names(r), function(i) {
    formatLayerName(layer = i)
  })

  # Read in study area
  x <- readRDS(study_area) %>% spTransform(CRSobj = crs(crs))

  # GDAL WARP -------------------------------
  cr <- warpStack(stack = r, study_region = x)

  # Compute mean PBL for each 3-hour observation
  s <- meanLayer(cr)

  # Extract min/max PBL layers for each day
  s <- pblMinMax(pbl_stack = s)
  rm(cr, x)
  gc()

  # Split the min/max layers apart to save in separate directories
  print("Subset min/max from stack")
  s_min <- subset(s, grep("^pbl\.min\.\$", names(s), value = T))
  s_max <- subset(s, grep("^pbl\.max\.\$", names(s), value = T))

  # Write each stack type to file
  print("Writing min stack to file")
  pblapply(1:nlayers(s_min), function(i) {
```
writeRaster(s_min[[i]], file.path(paste0(out_dir, "min/", names(s_min[[i]]), ".tif")),
  format = "GTiff",
  overwrite = TRUE,
  bylayer = T)
}
}, cl = detectCores() - 1
)

print("Writing max stack to file")
pblapply(1:nlayers(s_max), function(i) {
  writeRaster(s_max[[i]], file.path(paste0(out_dir, "max/"), paste0
    (names(s_max[[i]]), ".tif")))
  ,format = "GTiff"
  ,overwrite = TRUE
  ,bylayer = T
})
}, cl = detectCores() - 1
)

# Clean up
print("Garbage collection")
rm(s_min, s_max, s)
gc()
}

# Create a list of raw PBL nc data
PBLFiles <- list.files(path = ".data/PBL/nc", pattern = "\.nc$", full.names = T)

# For testing our function
# file_name <- PBLFiles[2]

# Apply our data transformation function to each file
pblapply(PBLFiles, function(x) {
  processPBL(file_name = x,
    ,study_area = study_area
    ,crs = epsg_26910
    ,out_dir = ".data/PBL/")
  # Clean up
  gc()
  # , cl = detectCores() - 1
})

# f <- list.files(path = ".data/PBL/tif/", full.names = T)
# r <- stack(f)
# a <- subset(r, 1:16)
Source/plotElevation.R

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal, rgeos, sp, raster, dplyr, magrittr, pbapply, parallel, snow)

source("./src/init.R")

processElevation <- function(file_name, study_area = "./study_area/study_area.RDS",
  crs = crs, outdir = "./data/elev") {
  r <- raster(file_name)
  x <- readRDS(study_area) %>% spTransform(CRSobj = crs(crs))
  # create larger buffer before masking/cropping
  y <- rgeos::gBuffer(x, width = 10000)
  y %>% spTransform(CRSobj = crs(r))
  # x %>% spTransform(CRSobj = crs(r))
  m <- mask(r, y, getCover = T) # masking on x excludes a few cells!
  cr <- crop(m, extent(y))
  # project to epsg_26910 @ 50 m resolution
  cr %>% projectRaster(crs = crs, method = "bilinear", res = sp_res)
  cr %>% mask(x) %>% crop(extent(x))
  # out_file <- gsub(basename(file_name), pattern = "\.nc", replacement = ".tif")
  out_file <- "elev.tif"
  raster::writeRaster(cr
    ,filename = paste0(outdir, "/", outfile)
    ,format = "GTiff"
    ,overwrite = TRUE
  )
}
# I'm using the single elevation layer for each day. It's not going to change that much each
processElev(file_name = "/data/elev/aster_dem_pdx.tif"
    , crs = crs
    , outdir = "/data/elev"
)

Source/processNeph.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018-11-13
# process DEQ automatic data, convert to tidy format, save as RDS

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(rgdal
  , plyr
  , plyr
  , magrittr
  , gdalUtils
  , raster
  , sp
  , sf
  , rgeos
  , leaflet
  , lubridate
  # , MODIS
  # , bowerbird
  # , gstat
  , parallel
  , snow
  , pbapply
  , ggplot2
  , rts
  , tidyR
  , RPostgres
  , ggmap
  , gridExtra
  , readxl)

# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.timezone()
Sys.setenv(TZ="GMT")
Sys.timezone()

# time zone applied to all data
time_zone <- "GMT"

# Read in PM data

# Read in 1-hour Automatic Data from Excel
read_frm <- function(file_path) {

  # read in the raw data
  x <- read_excel(file_path)

  sites <- x[3,]
  sites <- sites[which(!is.na(sites))]
  sites <- sites[1,-1] %>% as.character() %>% gsub(pattern = "\s", replacement = "_") %>% tolower()
  header <- x[4,]

  body <-  x[-c(1:4),]

  # Hardcode column names for now...
  colnames(body) <- header[1,]
  colnames(body)[2] <- sites[1]
  colnames(body)[5] <- sites[2]
  colnames(body)[8] <- sites[3]
  colnames(body)[14] <- sites[5]
  colnames(body)[17] <- sites[6]

  x <- body[,c(1:2,5,8,11,14,17)]

  # convert date from character to POSIXct
  x$Date <- mdy_hm(x$Date, tz = time_zone)
# This drops a lot of days!
# x <- x %%% na.omit()

# convert to tidy format, gather on site id
long <- long %>%
  gather(id, pm2_5_atm, -c(Date))

# convert characters in pm2_5 data to NA
long$pm2_5_atm <- as.numeric(long$pm2_5_atm)

# exclude NAs (No Data, Zero, DL/Computer, etc. gets converted to NA)
long <- long %>%
  na.omit()

# alter column names
colnames(long) <- c("datetime",
  "site_id",
  "pm2_5_ref")

long$lat[long$site_id == "beaverton_highland_park"] <- 45.470183
long$lat[long$site_id == "hillsboro_hare_field"] <- 45.528549
long$lat[long$site_id == "portland_se_lafayette"] <- 45.496635
long$lat[long$site_id == "portland_cully_helensview"] <- 45.562206
long$lat[long$site_id == "sauvie_island"] <- 45.768528
long$lat[long$site_id == "tualatin_bradbury_court"] <- 45.399189

long$lon[long$site_id == "beaverton_highland_park"] <- -122.816418
long$lon[long$site_id == "hillsboro_hare_field"] <- -122.972423
long$lon[long$site_id == "portland_se_lafayette"] <- -122.602883
long$lon[long$site_id == "portland_cully_helensview"] <- -122.575627
long$lon[long$site_id == "sauvie_island"] <- -122.772305
long$lon[long$site_id == "tualatin_bradbury_court"] <- -122.745538
long %>%
  dplyr::select(datetime, site_id, pm2_5_ref, lat, lon)
long %>%
  na.omit()
return(long)

# establish avg interval for xts() function
time_step <- 5

# for clean 5-min breaks
align.time.down = function(x, n) {
  index(x) = index(x) - n
}
align.time(x, n)
}

# remove geometry from sf object, convert to dataframe with only the
elements/attributes
st_drop_geometry <- function(x) {
  if (inherits(x, "sf")) {
    x <- st_set_geometry(x, NULL)
    class(x) <- "data.frame"
  }
  return(x)
}

# read functions for neph data that Meena provided
read_2017_neph <- function(file_path) {
  x <- read.csv(file_path)
  names(x)[names(x) == "Datetime.Local"] <- "datetime"
  x$datetime <- as.POSIXct(x$datetime, "%Y-%m-%d %H:%M:%S")
  names(x)[names(x) == "stn_id"] <- "site_id"
  names(x)[names(x) == "Sample.Measurement"] <- "pm2_5_ref"
  x$sit_id[x$sit_id == 410670111] <- "Beaverton Highland Park"
  x$sit_id[x$sit_id == 410670004] <- "Hillsboro Hare Field"
  x$sit_id[x$sit_id == 410510080] <- "Portland SE Lafayette"
  x$sit_id[x$sit_id == 410090004] <- "Sauvie Island"
  x$sit_id[x$sit_id == 410670005] <- "Tualatin Bradbury Court"
  x$lat[x$sit_id == "Beaverton Highland Park"] <- 45.470183
  x$lat[x$sit_id == "Hillsboro Hare Field"] <- 45.528549
  x$lat[x$sit_id == "Portland SE Lafayette"] <- 45.496635
  x$lat[x$sit_id == "Portland Cully Helensview"] <- 45.562066
  x$lat[x$sit_id == "Sauvie Island"] <- 45.768528
  x$lat[x$sit_id == "Tualatin Bradbury Court"] <- 45.399189
  x$lon[x$sit_id == "Beaverton Highland Park"] <- -122.816418
  x$lon[x$sit_id == "Hillsboro Hare Field"] <- -122.972423
  x$lon[x$sit_id == "Portland SE Lafayette"] <- -122.602883
  x$lon[x$sit_id == "Portland Cully Helensview"] <- -122.575627
  x$lon[x$sit_id == "Sauvie Island"] <- -122.772305
  x$lon[x$sit_id == "Tualatin Bradbury Court"] <- -122.745538
  x %<>% dplyr::select(datetime, sit_id, pm2_5_ref, lat, lon)
  x %<>% na.omit()
Appendix A. Source Code

```r
return(x)
}

read_2017_5min <- function(file_path) {
  # read in the raw data
  x <- read.csv(file_path, header = TRUE, skip = 2)
  # delete sub headers
  # x <- x[-c(1:2),]
  # convert date from character to POSIXct
  x$Date...Time <- mdy_hm(x$Date...Time, tz = time_zone)
  x <- x %>% na.omit()
  # convert to tidy format, gather on site id
  long <- x %>%
    gather(id, pm2_5_atm, -c(Date...Time))
  # convert characters in pm2_5 data to NA
  long$pm2_5_atm <- as.numeric(long$pm2_5_atm)
  # exclude NAs (No Data, Zero, DL/Computer, etc. gets converted to NA)
  long <- long %>% na.omit()
  # alter column names
  colnames(long) <- c("datetime",
          "site_id",
          "pm2_5_ref")
  long$site_id <- gsub("^X[0-9]+\S\.", "", long$site_id)
  long$site_id <- gsub("\S\.", "", long$site_id)
  long$site_id <- ifelse(long$site_id == "Portland Near Rd", "Tualatin Bradbury Court", long$site_id)
  long$lat[long$site_id == "Beaverton Highland Park"] <- 45.470183
  long$lat[long$site_id == "Hillsboro Hare Field"] <- 45.528549
  long$lat[long$site_id == "Portland SE Lafayette"] <- 45.496635
  long$lat[long$site_id == "Portland Cully Helensview"] <- 45.562206
  long$lat[long$site_id == "Sauvie Island"] <- 45.768528
  long$lat[long$site_id == "Tualatin Bradbury Court"] <- 45.399189
  long$lon[long$site_id == "Beaverton Highland Park"] <- -122.816418
```
Appendix A. Source Code

```r
long$lon[long$site_id == "Hillsboro Hare Field"] <- -122.972423
long$lon[long$site_id == "Portland SE Lafayette"] <- -122.602883
long$lon[long$site_id == "Portland Cully Helensview"] <- -122.575627
long$lon[long$site_id == "Sauvie Island"] <- -122.772305
long$lon[long$site_id == "Tualatin Bradbury Court"] <- -122.745538

long %<>% dplyr::select(datetime, site_id, pm2_5_ref, lat, lon)
long %<>% na.omit()

long$site_id <- gsub("\", ",", long$site_id)
long$site_id <- tolower(long$site_id)
#long$site_id <- stringr::str_replace(long$site_id, ".", ",")
return(long)
}

read_2018_5min <- function(file_path) {
  # read in the raw data
  x <- read_excel(file_path)
  # delete sub headers
  x <- x[,c(1:2),]
  x <- x[,2:ncol(x)]
  names(x)[names(x) == "Date & Time - start of obs period"] <- "Date"
  # convert date from character to POSIXct
  x$Date <- ymd_hms(x$Date, tz = time_zone)
  x <- x %>% na.omit()
  # convert to tidy format, gather on site id
  long <- x %>%
    gather(id, pm2_5_atm, -c(Date))
  # convert characters in pm2_5 data to NA
  long$pm2_5_atm <- as.numeric(long$pm2_5_atm)
  # exclude NAs (No Data, Zero, DL/Computer, etc. gets converted to NA)
  long <- long %>% na.omit()
  # alter column names
  colnames(long) <- c("datetime", "site_id")
```
Appendix A. Source Code

, "pm2_5_ref")

# long $site_id <- gsub("^X[0-9]+\", "", long$site_id)
#long$site_id <- gsub("\", " ", long$site_id)

long$site_id <- ifelse(long$site_id == "Portland Near Rd", "Tualatin Bradbury Court", long$site_id)

long$lat[long$site_id == "Beaverton Highland Park"] <- 45.470183
long$lat[long$site_id == "Hillsboro Hare Field"] <- 45.528549
long$lat[long$site_id == "Portland SE Lafayette"] <- 45.496635
long$lat[long$site_id == "Portland Cully Helensview"] <- 45.562206
long$lat[long$site_id == "Sauvie Island"] <- 45.768528
long$lat[long$site_id == "Tualatin Bradbury Court"] <- 45.399189

long$lon[long$site_id == "Beaverton Highland Park"] <- −122.816418
long$lon[long$site_id == "Hillsboro Hare Field"] <- −122.972423
long$lon[long$site_id == "Portland SE Lafayette"] <- −122.602883
long$lon[long$site_id == "Portland Cully Helensview"] <- −122.575627
long$lon[long$site_id == "Sauvie Island"] <- −122.772305
long$lon[long$site_id == "Tualatin Bradbury Court"] <- −122.745538

long %<>% dplyr::select(datetime, site_id, pm2_5_ref, lat, lon)
long %<>% na.omit()

long$site_id <- gsub("\", "_", long$site_id)
long$site_id <- tolower(long$site_id)
#long$site_id <- stringr::str_replace(long$site_id, ".", "_")

return(long)

read_2018_neph <- function(file_path) {

  # extract header
  header <- read.csv(file_path, nrow = 1, skip = 2, header = FALSE, stringsAsFactors = FALSE) %>%
    select_if(~ !any(is.na(.)))

  # extract body
  x <- read.csv(file_path, header = TRUE, skip = 4, stringsAsFactors = FALSE)
  colnames(header) <- unlist(header[1,])
  colnames(x) <- colnames(header)
names(x)[names(x) == "Date & Time"] <- "datetime"
x$datetime <- as.POSIXct(x$datetime, format = "%m/%d/%Y %H:%M")
x <- x %>%
gather(key = site_id, value = pm2_5_ref, -datetime)
x$pm2_5_ref <- as.numeric(x$pm2_5_ref)
x <- x %>% na.omit()
x$lat[x$site_id == "Beaverton Highland Park"] <- 45.470183
x$lat[x$site_id == "Hillsboro Hare Field"] <- 45.528549
x$lat[x$site_id == "Portland SE Lafayette"] <- 45.496635
x$lat[x$site_id == "Portland Cully Helensview"] <- 45.562206
x$lat[x$site_id == "Sauvie Island"] <- 45.768528
x$lat[x$site_id == "Tualatin Bradbury Court"] <- 45.399189
x$lon[x$site_id == "Beaverton Highland Park"] <- -122.816418
x$lon[x$site_id == "Hillsboro Hare Field"] <- -122.972423
x$lon[x$site_id == "Portland SE Lafayette"] <- -122.602883
x$lon[x$site_id == "Portland Cully Helensview"] <- -122.575627
x$lon[x$site_id == "Sauvie Island"] <- -122.772305
x$lon[x$site_id == "Tualatin Bradbury Court"] <- -122.745538

return(x)

neph2017 <- read_2017_5min("./data/frm/bhp_hhf_tbc_sel_5min_pm25_20170701_to_20180918.csv")
neph2018 <- read_2018_5min("./data/frm/pm25_5min_bhp_hhf_pch_sel_sis_tbc_sep18_to_may19.xlsx")

# TODO Sauvie Island is missing from 2017 data!

# converting from hourly to daily, computing min, max, mean, sd
neph <- rbind(neph2017, neph2018)

# remove any duplicates (e.g. September 2018)
neph %>% unique()

neph$date <- lubridate::date(neph$datetime)
neph_daily <- neph %>% dplyr::select(-datetime) %>% group_by(date, site_id, lat, lon) %>%
summarize_if(is.numeric,
  funs(pm2.5.mean = mean
       ,pm2.5.min = min
       ,pm2.5.max = max
       ,pm2.5.sd = sd
       ,pm2.5.median = median)
neph %>% st_as_sf(coords = c("lon","lat"), crs = wgs_84)
neph_daily %>% st_as_sf(coords = c("lon","lat"), crs = wgs_84)
# Inspect the data and clean up outliers
summary(neph)
summary(neph_daily)

# Yikes there are definitely some issues with the mean/max values
plot(neph_daily$date, neph_daily$pm2.5.mean)

# Let's filter by our max pm range init variable (250)
neph_daily %>% filter(pm2.5.mean <= max_pm_range)
summary(neph_daily)
neph %>% filter(pm2_5_ref <= max_pm_range)
summary(neph)

# Looks a lot better now
plot(neph_daily$date, neph_daily$pm2.5.mean)

# TODO Create neph 15-minute averages?

# Write our clean neph data to file
saveRDS(neph_daily, ".\data\deq\neph1day.RDS")
saveRDS(neph, "./data/eq/neph5min.RDS")
neph <- read_frm("./data/frm/AutomaticData5_29_2019.xlsx")
neph %>% st_as_sf(coords = c("lon","lat"), crs = wgs_84)
neph %>% filter(pm2_5_ref <= max_pm_range)
saveRDS(neph, "./data/deq/neph1hour.RDS")
Appendix A. Source Code

15 , dplyr
16 , magrittr
17 , gdalUtils
18 , raster
19 , sp
20 , sf
21 , rgeos
22 , parallel
23 , lubridate
24 , ggplot2
25 , rts
26 , RPostgres
27 , ggmap
28 , gridExtra)
29
30 source("./src/functions.R")
31 source("./src/init.R")
32
33 # set timezone
34 # set timezone to UTC/GMT to match postgres
35 # otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
36 # DEQ only uses PST!
37 Sys.timezone()
38 Sys.setenv(TZ="GMT")
39 Sys.timezone()
40
41 # Connect to Database
42
43 ## connecting to local db
44 host <- 'pgsql120.rc.pdx.edu'
45 db <- 'canopycontinuum'
46 user <- 'porlando'
47 port <- 5433
48 pw <- scan("./batteries.pgpss", what = "") # in parent dir
49
50 # open connection to our db
51 con <- dbConnect(drv = RPostgres::Postgres()
52 , dbname = db
53 , host = 'pgsql102.rc.pdx.edu' # not sure why object host isn’t working...
54 , port = port
55
# Reading in hourly PurpleAir data

# Add select distinct subquery into this later!
pa_h <- st_read(dsn = con

# geom_column = "geom"
# EWKB = TRUE
# query = paste0("SELECT date_trunc('hour', created_at) hourly, label, avg(pm2_5_atm), geom
FROM pdx
" GROUP BY label, date_trunc('hour', created_at), geom;")

# Reading in PurpleAir data

start_date <- "2017-07-01"
end_date <- "2018-10-20"

pa <- st_read(dsn = con

# geom_column = "geom" # deprecated!
# EWKB = TRUE
# query = paste0("SELECT date(a.created_at), date_part('hour', a.created_at) as hour, a.id, a.sensor, a.label, avg(a.pm2_5_atm) pm2_5_atm, a.geom
FROM (SELECT DISTINCT * FROM pdx WHERE created_at >= "'" lubridate::ymd(start_date) + hours(8)
"' AND created_at <= "'" lubridate::ymd(end_date) + hours(8),
"') a
GROUP BY a.id, a.sensor, a.label, a.geom, date(a.created_at), date_part('hour', a.created_at);")
#)
Appendix A. Source Code

```r
, lubridate::ymd(end_date) + hours(8), "
',
)

# convert from UTC to UTC-8 or PST (same as DEQ)
pacs_created_at <- lubridate::ymd_hms(pacs_created_at) - lubridate::hours(8)

# rename the datetime column
names(pac)[names(pac) == "created_at"] <- "datetime"

# check the time zone of pac data
attr(pac$datetime, "tzone")

# set SRID
st_crs(pac) <- epsg_26910
pac$date <- lubridate::date(pac$datetime)
head(pac)

# Apply Individual Correction Factors each sensor (Aug 2018)

pac_cf <- st_read(dsn = con
  
  #, geom_column = "geom"
  
  #, EWKB = TRUE
  
  , query = "SELECT DISTINCT * FROM pdx_cf;" )

# remove geom for now...

pac %<>% st_drop_geometry()
pac_cf %<>% st_drop_geometry()

# pac_cf$year %<>% as.integer()
pac$id %<>% as.integer()

# # Find the CF with the maximum R2 for each sensor, excluding anything below 0.8
# pac_cf %<>%
# group_by(id, label) %>%
# # use the month with highest correlation (includes August 2017 for some sensors)
# # filter(year == 2018 & month == "Aug") %>%
# # slice(which.max(r_squared)) %>%
# # filter(r_squared > 0.85) %>%
# # na.omit()

# join CF and PA data

pac %<>% inner_join(pac_cf %>% dplyr::select(c(id, label, r_squared, slope, intercept))) %>%
  dplyr::select(c(datetime, id, label, everything()))
```
# apply correction factor
pa$pm2_5_cf_x <- (pa$pm2_5_atm*pa$slope)+pa$intercept

# inspect the new data (compare with lm figures)
pa %>% dplyr::select(c(datetime, label, id, slope, pm2_5_atm, pm2_5_cf_x, r_squared)) %>%
  filter(datetime == as.Date("2018-08-01")) %>%
  arrange(label)

# inspect raw data and cleaned data
pa %>% dplyr::filter(r_squared > 0.85) %>% ggplot(aes(x = label, y = pm2_5_cf_x)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

pa %>% filter(pm2_5_cf_x < max_pm_range & r_squared > 0.85 & slope >=
0.4 & slope <= 1.6) %>%
na.omit() %>%
dplyr::filter(r_squared > 0.8) %>% ggplot(aes(x = label, y = pm2_5_cf_x)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

pa %>% filter(pm2_5_cf_x < max_pm_range & r_squared > 0.85 & slope >=
0.4 & slope <= 1.6) %>%
a filter(r_squared > 0.8) %>% ggplot(aes(x = as.factor(id), y = slope)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

pa_cf %>% ggplot(aes(x = 1, y = slope, fill = as.factor(1))) + geom_violin() + xlab("") +
theme_minimal() + ggtitle("PurpleAir Calibration Factors") +
theme(legend.title = element_blank(), legend.position = "none",
axis.text.x = element_blank())

pa_cf %>% ggplot(aes(x = 1, y = r_squared, fill = as.factor(1))) +
  geom_violin() + xlab("") +
theme_minimal() + ggtitle("PurpleAir Calibration Factors") +
theme(legend.title = element_blank(), legend.position = "none",
axis.text.x = element_blank())

# Remove faulty sensors
pa_flag <- st_read(dsn = con
  #, geom_column = "geom"
  #, EWKB = T
  , query = "SELECT DISTINCT * FROM pdx_flag;")

# examine quality of flagging on suspect sensors
top_flag <- pa_flag %>% filter(flag == 1) %>%
  dplyr::select(label, id) %>%
  group_by(label, id) %>%
  mutate(n = n()) %>%
  summarise_if(is.numeric, mean) %>%
  arrange(desc(n))

# ensure id column type matches pa
pa$id <- as.character(pa$id)

# With pdx_flag == 0

# TODO confirm outlier detection algo is not worthwhile with new CF methods...
# cleaning data by slope between 0.5 and 1.5 helps reduce additional outliers
# pa_clean <- pa %>% inner_join(pa_flag, by = c("date", "id", "label "))
# # exclude faulty sensor data, and trim extremes
# filter(flag == 0 & pm2_5_cf_x <= max_pm_range & r_squared >= 0.9)
# Not using the pdx_flag algo

pa_clean <- pa %>% inner_join(pa_flag, by = c("date", "id", "label ")
  filter(pm2_5_cf_x <= max_pm_range & r_squared >= 0.9)

# exclude faulty sensor data, and trim extremes
# filter(pm2_5_cf_x <= max_pm_range & r_squared >= 0.85 & slope >=
#        0.5 & slope <= 1.5)

nrow(pa_clean)/nrow(pa)

pa_clean %>% ggplot(aes(x = label, y = pm2_5_cf_x)) +
  geom_violin() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
# pa_clean <- pa %>%
# filter(pm2_5_cf_x < max_pm_range & r_squared > 0.85 & slope >= 0.4 & slope <= 1.6) %>%
# na.omit()

pa_5m <- pa_clean %>%
dplyr::select(-c(date, geom))

pa_summary <- pa_clean %>%
dplyr::select(-datetime) %>%
group_by(date, id, sensor, label) %>%
summarise_at("pm2_5_cf_x", funs(mean, median, min, max))

pa_summary$diff <- abs(pa_summary$mean - pa_summary$median)

pa_summary %>%
group_by(id, sensor, label) %>%
summarise_at("diff", funs(mean, median, max, min)) %>%
arrange(desc(max))

pa_24h <- pa_clean %>%
dplyr::select(-c(datetime, geom)) %>%
group_by(date, id, sensor, label) %>%
summarise_if(is.numeric, funs(mean), na.rm = T)

pa_24h_all <- pa_dirty %>%
group_by(date, id, sensor, label) %>%
summarise_if(is.numeric, funs(mean), na.rm = T)

pa_24h_filter <- pa_filter %>%
group_by(date, id, sensor, label) %>%
summarise_if(is.numeric, funs(mean), na.rm = T)

pa_24h %>% ggplot(aes(x = label, y = pm2_5_cf_x)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

pa_24h %>% ggplot( aes(x = date, y = pm2_5_cf_x, col = flag)) +
  geom_point() +
  theme_bw()

pa_24h_all %>% filter(pm2_5_cf_x <= max_pm_range & label %in% top_flag$labl) %>%
na.omit() %>%
ggplot( aes(x = date, y = pm2_5_cf_x, col = as.factor(flag))) +
  geom_point(alpha = 0.6) +
theme_bw() +
scale_color_manual(values = c("#000000", "#ba2525")) +
labs(y = "PM2.5 (ug/m3)", x = "", color = "Outlier Flag", title = 
  "Modified Z-Score Outlier Detection") +
theme(legend.position="bottom")

ggplot( aes(x = date, y = pm2_5_cf_x, col = as.factor(flag))) +
  geom_point(alpha = 0.6) +
  theme_bw() +
scale_color_manual(values = c("#000000", "#ba2525")) +
labs(y = "PM2.5 (ug/m3)", x = "", color = "Outlier Flag", title = " 
  Modified Z-Score Outlier Detection") +
  theme(legend.position="bottom")
ggsave(last_plot()
  ,filename = paste0("./figures/outlier_detection/", print_clean _time(), "-Jan-Mar-z-score-outlier-ts.png")
)

pa_24h_all %>% filter(date >= "2018-01-01" & date <= "2018-03-01")
  %>%
ggplot( aes(x = date, y = pm2_5_cf_x, col = as.factor(flag))) +
  geom_point(alpha = 0.6) +
  theme_bw() +
scale_color_manual(values = c("#000000", "#ba2525")) +
labs(y = "PM2.5 (ug/m3)", x = "", color = "Outlier Flag", title = " 
  Modified Z-Score Outlier Detection") +
  theme(legend.position="bottom")
ggsave(last_plot()
  ,filename = paste0("./figures/outlier_detection/", print_clean _time(), "-Jul-Sep-z-score-outlier-ts.png")
)

pa_24h_filter %>% na.omit() %>% ggplot( aes(x = date, y = pm2_5_cf_x,
  col = as.factor(flag))) +
  geom_point(alpha = 0.25) +
  theme_bw() +
scale_color_manual(values = c("#000000", "#ba2525")) +
labs(y = "PM2.5 (ug/m3)", x = "", color = "Outlier Flag", title = " 
  Modified Z-Score Outlier Detection") +
```r
theme(legend.position="bottom")

ggsave(last_plot(),
filename = paste0("./figures/outlier_detection/", print_clean_time(), "-z-score-outlier-ts.png")
)

# Reattach geometry

# PurpleAir geometry
# lookup table for sensor geom
pa_points <- st_read(dsn = con
  #,geom_column = "geom"
  #,EWKB = TRUE
  ,query = "SELECT DISTINCT id, label, geom FROM pdx;"
)

# set SRID
st_crs(pa_points) <- epsg_26910

# # Average pm per node
# pa_5m$node <- gsub(" B$", ",", pa_5m$label)

# This takes forever and isn’t being used...
# # pa_5m$id <- as.integer(pa_5m$id)
# # pa_5m %<% inner_join(pa_points) %>% st_as_sf()
# st_crs(pa_5m) <- epsg_26910
#
# # convert geom to text for grouping
# pa_5m$geom_text <- st_as_text(pa_5m$geom)
#
# # average node value by node, date, geom_text
# pa_5m_node <- pa_5m %>% ungroup() %>%
# # dplyr::select(-c(id, sensor, label)) %>%
# # group_by(date, node, geom_text) %>%
# # summarise_if(is.numeric, funs(mean)) %>% ungroup() %>%
# # dplyr::select(date, node, pm2_5_cf_x, geom_text)
#
# # convert back to sf
# pa_5m_node$geom <- st_as_sfc(pa_5m_node$geom_text)
# pa_5m_node %<% st_as_sf()
# st_crs(pa_5m_node) <- epsg_26910
# str(pa_5m_node)
#
# pa_5m_node %<% as("Spatial")
```
# Average pm per node
pa_24h$node <- gsub(" B\$", "", pa_24h$label)

# attach geometry
pa_points$id <- as.character(pa_points$id)

# convert geom to text for grouping
pa_24h$geom_text <- st_as_text(pa_24h$geom)

# average node value by node, date, geom_text
pa_24h_node <- pa_24h %>% ungroup() %>%
dplyr::select(-c(id, sensor, label)) %>%
group_by(date, node, geom_text) %>%
summarise_if(is.numeric, funs(mean)) %>%
dplyr::select(c(date, node, pm2_5_cf_x, geom_text))

head(pa_24h_node)

# convert back to sf
# pa_24h_node$geom <- st_as_sfc(pa_24h_node$geom_text)
# pa_24h_node %>% st_as_sf()
# st_crs(pa_24h_node) <- epsg_26910

# make sure it preserved geometries when node names are the same...
miller <- pa_24h_node %>% filter(node == "Miller")

# exclude bad Miller sensor in NE Portland
pa_24h_node %>%
filter(geom_text != miller[1,]$geom_text)

# drop geom_text column
pa_24h_node %>% dplyr::select(-geom_text)

# finally, convert back to sp
pa_24h_node %>% as("Spatial")

# Write to file

# saveRDS(pa_5m_node, ".\data\purpleair\pm_df_5m.RDS")
saveRDS(pa_24h_node, ".\data\purpleair\pm_df_24h.RDS")
# writeOGR(pa, ".\data\purpleair/", layer = "pm_df_5m", driver = "ESRI Shapefile", overwrite = T)
# writeOGR(pa_24h, "./data/purpleair/", layer = "pm_df_24h", driver = "ESRI Shapefile", overwrite = T)

Source/buffer.R

define by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018−11−19
# create daily stacks of GIS predictors, buffer and extract them

# needed for nvim−R
if (basename(getwd()) == "src") {
  setwd(".
  getwd()
}

# load the necessary packages

if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr
, dplyr
, magrittr
, tools
, rlang
, rgdal
, raster
, rasterVis
, corrplot
, sp
, rgeos
, ggplot2
, stringr
, RColorBrewer
, pbapply
, parallel
, randomForest
, ranger
, tibble
, broom
, lubridate
, #,mlr # one of these conflicts with raster somewhere...
, #,tuneRanger
, breakDown # visualize ranger model
# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

# create a list of daily predictor variables (abbreviations)
vars <- c(
  "pbl.min",
  "pbl.max",
  "pr",
  "vs",
  "th",
  "tmmn",
  "tmmx",
  "rmin",
  "rmax",
  ","spatial"
  ","ndvi" # annual median (not daily resolution anymore...)
  ","evi" # annual median (not daily resolution anymore...)
  ","pm2.5.mean"
  ","pm2.5.max"
  ","pm2.5.min"
  ","pm2.5.sd"
  ","pm2.5.median"
)

# list of static predictors
static <- c(
  "fwy",
  "streets",
  "maj_art",
  "arterial",
  "railroad",
  "railyards",
  "elev",
  "pop_dens" # GFW v4 Interpolated Population Density Data
  "barren_land",
  "cultivated_crops",
  "deciduous_forest",
  "developed_high_intensity",
  "developed_low_intensity",
  "developed_medium_intensity",
  "developed_open_space",
  "emergent_herbaceous_wetlands",
  "evergreen_forest"
)
Appendix A. Source Code

# Define global model start–time to compare RDS objects and figures later

clean_time <- print_clean_time()
saveRDS(clean_time, "./models/clean_time/clean_time.RDS")
saveRDS(clean_time, paste0("./models/clean_time/", clean_time, "_clean-time.RDS"))

# Define read functions for static and dynamic variables

# read functions for daily and static predictors
readVar <- function(var, date = NULL, study_area) {
  if (is.null(date)) {
    # print(var)
    f <- list.files("./data", pattern = paste0("^", var, "\.tif$"), full.names = T, recursive = T)
  } else {
    f <- list.files("./data", pattern = paste0("^", var, "\." , date, "\.tif$"), full.names = T, recursive = T)
  }
  # Exclude buffered rasters from list.files!
  f <- f[!grep("buf_vars|buf_static", f)]
  if (length(f) == 1) {
    r <- raster(f) %>% resample(study_area, method = 'ngb')
  } else print(paste0("More than one file selected for raster layer : ", var, "!'"))
  return(r)
}

# Focal Buffer functions

buffers <- c(0, 50, 100, 300, 500, 1000) # ESCAPE Project Buffers
# buffers <- c(0, 25, 100) # for testing
# buffers <- c(sp_res, seq(0, 1200, 50)) # Meena's Buffers
# buffers <- c(0) # No Buffers (still need to run the script though)

# Checkpoint with clean_time
saveRDS(buffers, paste0("../data/buffers/", clean_time, "_buffers.RDS" ))

# Run a single moving window
focalBufferRaster <- function(rast, bufferDist) {
    # skip wind direction and PBL during buffering
    if (!grepl(paste0('^th|pbl.*|pm2\.5\.*|barren_land\.*|cultivated_crops\.*|deciduous_forest\.*|developed_high_intensity\.*|developed_medium_intensity\.*|developed_low_intensity\.*|developed_open_space\.*|developed_open_forest\.*|emergent_herbageous_wetlands\.*|evergreen_wetlands\.*|hay_pasture\.*|herbageous\.*|mixed_forest\.*|open_water\.*|shrub_scrub\.*|woody_wetlands\.*', names(rast)))) {
        buf <- focal(rast, w = focalWeight(rast, bufferDist, type="circle"), na.rm = T)
        buf[is.na(buf)] <- 0
        names(buf) <- paste0(names(rast), ".", bufferDist, "m")
        return(buf)
    } else if(grepl(paste0('^barren_land\.*|cultivated_crops\.*|deciduous_forest\.*|developed_high_intensity\.*|developed_medium_intensity\.*|developed_low_intensity\.*|developed_open_space\.*|developed_open_forest\.*|emergent_herbageous_wetlands\.*|evergreen_wetlands\.*|hay_pasture\.*|herbageous\.*|mixed_forest\.*|open_water\.*|shrub_scrub\.*|woody_wetlands\.*', names(rast)))) {
        buf <- focal(x = rast, fun = sum, w = focalWeight(rast, bufferDist, type = "rectangle"), na.rm = T)
        buf[is.na(buf)] <- 0
        names(buf) <- paste0(names(rast), ".", bufferDist, "m")
    }
}
```
    return (buf)
}
}

# Wind Vector Averaging Function
focalBufferWind <- function(stack, bufferDist) {

  # grab the wind direction and speed layers from stack
  wd <- raster::subset(stack, grep('^th\.*', names(stack), value = T))
  ws <- raster::subset(stack, grep('^vs\.*', names(stack), value = T))

  # determine u & v components
  u <- ws * sin(2*pi*wd/360) * (-1)
  v <- ws * cos(2*pi*wd/360) * (-1)

  # focal on each component
  mean_u <- focal(u, focalWeight(u, bufferDist, type = "circle"), na.rm=T)
  mean_v <- focal(v, focalWeight(v, bufferDist, type = "circle"), na.rm=T)

  # convert back to degrees
  mean_wd <- (atan2(mean_u, mean_v) * 360/2/pi) + 180

  # assign wd buffer name to output
  names(mean_wd) <- paste0(names(wd), ".", bufferDist, "m")

  return (mean_wd)
}

focalBufferOneHot <- function(stack, buffers, cl) {

  lc <- raster::subset(stack, grep(paste0('^barren_land\.*|\^cultivated_crops\.*|\^deciduous_forest\.*|\^developed_high_intensity\.*|\^developed_medium_intensity\.*|\^developed_low_intensity\.*|\^developed_open_space\.*|\^emergent_herbageous\.*|\^evergreen_forest\.*|\^hay_pasture\.*|\^herbageous\.*|\^mixed_forest\.*|\^barren_land\.*', collapse = "\|"))

  return (lc)
}
```
Appendix A. Source Code

```r
202 , ^open_water\..*|^shrub_.
203 \s+|^woody_wetlands\.\s^')
204 , names(stack), value = T))
205 lc_buf <- pblapply(buffers, function(bufferDist) {
206 lc_s <- lapply(1:nlayers(lc), function(i) {
207 # Focal weight with sum normalizes between 0–1???
208 lc_prop <- focal(x = lc[[i]], fun = sum, w = focalWeight(lc[[i]]),
209 bufferDist, type = "rectangle"), na.rm = T)
210 names(lc_prop) <- paste0(names(lc[[i]]), ".", bufferDist, 'm')
211 return(lc_prop)
212 }) %>% stack()
213 return(lc_s)
214 }
215 , cl = cl
216 ) %>% stack()
217 return(lc_buf)
218 }
219
220 # apply focalBufferRaster to entire stack
221 focalBufferStack <- function(stack, bufferDist) {
222 s_list <- lapply(1:nlayers(stack), function(x) {
223 focalBufferRaster(rast = stack[[x]], bufferDist)
224 })
225
226 # Ignored layer names return NULL and must be filtered out
227 s <- s_list[!apply(s_list, is.null, logical(1))] %>% stack()
228
229 # wind direction buffering (requires wd and ws)
230 if (any(grepl('^th\.\s^', names(stack))) & any(grepl('^vs\.\s^',
231 names(stack)))) {
232 wd <- focalBufferWind(stack, bufferDist)
233 s %>% stack(wd)
234 rm(wd)
235 gc()
236 }
237 # Replace NAs with 0
238 # s[is.na(s)] <- 0 # This renames the stack to layer.1 etc.!!!!
239 return(s)
240 }
241
242 # Run all of the focal buffers at once!
243 createBuffers <- function(stack, buffers, cl = NULL) {
244 if(is.null(cl)) {
```
Appendix A. Source Code

244  s_list <- pblapply(buffers, function(x) focalBufferStack(stack, bufferDist = x), cl = cl)
245  } else {
246    s_list <- lapply(buffers, function(x) focalBufferStack(stack, bufferDist = x))
247  }
248  s <- s_list[!vapply(s_list, is.null, logical(1))] %>% stack()
249  return(s)
250 }

# function resamples and combines all GIS predictors into single raster stack
stackPredictors <- function(date=NULL, crs, study_area, vars) {
  if (rlang::is_null(date)) {
    s <- lapply(vars, function(x) readVar(var = x, study_area = study_area)) %>% stack()
  } else {
    s <- lapply(vars, function(x) readVar(var = x, date = date, study_area = study_area)) %>% stack()
  }
  return(s)
}

# stackPredictors <- function(date, crs, study_area, static, vars) {
#  # # Study Area Extent Raster used for resampling...
#  # # a <- readRDS(study_area) %>% spTransform(CRSobj = crs) %>% raster(res = sp_res, vals = 0)
#  # # a <- study_area
#  # # read daily predictors
#  # r <- lapply(vars, function(x) readVar(var = x, date = date, study_area = study_area)) %>% stack()
#  # # read static predictors
#  # y <- lapply(static, function(x) readStatic(var = x, study_area = study_area)) %>% raster::stack()
#  # # combine them
#  # s <- raster::stack(r, y)
#  # rm(r, y)
#  # gc()
#  # return(s)
#  #
Appendix A. Source Code

# Raster processing functions

# Combine all predictors for a given day
bufferPredictors <- function(date, crs, study_area, bb, vars) {
    # Study Area Extent Raster used for resampling...
    a <- readRDS(study_area) %>% spTransform(CRSobj = crs) %>% raster(res = sp_res, vals = 0)
    # Final bounding box
    b <- readRDS(bb) %>% spTransform(CRSobj = crs) %>% raster(res = sp_res, vals = 0)
    # Create Raster Stack
    s <- stackPredictors(date = date,
                          crs = crs,
                          study_area = a,
                          vars = vars)
    # Buffering
    s_buf <- createBuffers(s, buffers = buffers)
    # Add PBL and Neph data (not buffered)
    pbl <- raster::subset(s, grep('^pbl\..*', names(s), value = T))
    neph <- raster::subset(s, grep('^pm2\.5\..*', names(s), value = T))
    neph <- raster::subset(s, grep('^pm2\.5\..median\..*', names(s), value = T))
    s_buf %>% raster::stack(pbl, neph)
    # strip date from daily predictor layer names
    names(s_buf) <- gsub(x = names(s_buf), pattern = paste0("\..", date), replacement = "")
    # Rename 0m buffer to match non-buffered layers
    names(s_buf) <- gsub(x = names(s_buf), pattern = paste0("\..0m\$"), replacement = "")
    # Crop extent to original bounding box (10 times smaller than buffer boundary)
    s_buf %>% raster::crop(b, snap = 'near') %>% resample(b, method = 'ngb')
    s_buf %>% raster::mask(b)
# Write s_buf to file

# return a cropped raster (using the largest buffer distance to cut from)
writeRaster(s_buf
, filename = paste0("./data/buf_vars/", date, ".", names
(s_buf), ".tif")
   , bylayer=T
   , format="GTiff"
   , overwrite=T)

# lapply(1:nlayers(s_buf), function(n) {
#   writeRaster(s_buf[[n]]
#   , filename = paste0("./data/buf/", date, ".", names(s_buf[[n]]), ".tif")
#   , format="GTiff"
#   , overwrite=T
# })

rm(neph, pbl, s, a, b, s_buf)
gc()
}

# Apply focal buffers to static variables

bufferStatic <- function(crs, study_area, bb, static) {
  # Study Area Extent Raster used for resampling...
a <- readRDS(study_area) %>% spTransform(CRSoobj = crs) %>% raster(res = sp_res, vals = 0)

  # Final bounding box
b <- readRDS(bb) %>% spTransform(CRSoobj = crs) %>% raster(res = sp_res, vals = 0)

  # Create Raster Stack
s <- stackPredictors(crs = crs
                      , study_area = a
                      , vars = static)

  # Buffering

  s_buf <- createBuffers(s, buffers = buffers
                           , cl = detectCores()-1)
# Larger Buffers for population density? Henderson paper?

```
pop_buf <- createBuffers(s[['pop_dens']],
                      buffers = c(750, 1250, 1500, 2000, 2500),
                      cl = detectCores() - 1)

s_buf <- stack(s_buf, pop_buf)
```

# # Loops through buffers for NLCD data

```
lc_buf <- focalBufferOneHot(s, buffers = buffers
                      , cl = detectCores() - 1
                      )

s_buf <- stack(s_buf, lc_buf)
```

# Rename 0m buffer to match non-buffered layers

```
names(s_buf) <- gsub(x = names(s_buf), pattern = paste0("\ losses ", replacement = " ")
```

# Crop extent to original bounding box (10 times smaller than buffer boundary)

```
s_buf %<>% raster::crop(b, snap = 'near') %<>% resample(b, method = 'ngb')
s_buf %<>% raster::mask(b)
```

# Write s_buf to file

```
writeRaster(s_buf
                , filename = paste0("./data/buf_static/", names(s_buf), 
                ".tif")
                , bylayer=T
                , format="GTiff"
                , overwrite=T)
```

# wrap our stacking function into a date sequence generator

```
bufferDailyPredictors <- function(start_date
                      , end_date
                      , crs
```
Appendix A. Source Code

```r

# Purge any remnant buffer files before loop
f_vars <- list.files("./data/buf_vars", full.names = T)
if(!rlang::is_empty(f_vars)) {
  print("Purging ./data/buf_vars...")
  pblapply(f_vars, file.remove, cl = detectCores() - 1)
}

f_static <- list.files("./data/buf_static", full.names = T)
if(!rlang::is_empty(f_static)) {
  print("Purging ./data/buf_static...")
  pblapply(f_static, file.remove, cl = detectCores() - 1)
}

# Create static buffers only once --------------------------
print("Creating new static buffers")
bufferStatic(crs = crs, bb = bb, study_area = study_area, static = static)

dates <- sequenceDates(start_date, end_date)
print("Creating new daily buffers...")
# Reduce and rbind are necessary for parallel processing (avoid
date.layer.buf.tif generation)!
# Do not refactor this!!! Leave it alone!!! Still doing it on
hecate?
# df <- Reduce(rbind, pblapply(dates, function(x) {
#   bufferPredictors(date = x
#   #, study_area = study_area
#   #, bb = bb
#   #, crs = crs
#   #, vars = vars
#   #, static = static
#   # }
#   , cl = detectCores() - 1
#   # )
# )
# return(df)

pblapply(dates, function(x) {
  bufferPredictors(date = x
  #, study_area = study_area
  #, bb = bb
  #, crs = crs
  #, vars = vars
```
# Running our function on multiple days

```r
buffer_time <- system.time(bufferDailyPredictors(start_date = start_date, end_date = end_date, crs = epsg_26910, study_area = study_area, bb = bb, vars = vars, static = static))

saveRDS(buffer_time, paste0("./models/buffer_time/", clean_time, "_buffer_time.RDS"))
```

**Source/extract.R**

```
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018–11–19
# create daily stacks of GIS predictors, buffer and extract them
# needed for nvim–R
if (basename(getwd()) == "src") {
  setwd("../")
}
getwd()

# load the necessary packages
if (!require(pacman)) {
```
Appendix A. Source Code

```r
install.packages("pacman")
library(pacman)

p_load(plyr,
  dplyr,
  magrittr,
  tools,
  rgdal,
  raster,
  rasterVis,
  corrplot,
  sp,
  rgeos,
  ggplot2,
  stringr,
  RColorBrewer,
  pbapply,
  parallel,
  randomForest,
  ranger,
  tibble,
  broom,
  lubridate
#

# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

# Load in our most recent dataset by timestamp
# clean_time <- readRDS("./models/clean_time/clean_time.RDS")
# clean_time <- print_clean_time()

# clean_time <- "2019-06-19_04_31_39"
# clean_time <- "2019-04-12_09_43_08"
# clean_time <- "2019-04-15_22_56_59"
# clean_time <- "2019-04-22_20_43_46"
# clean_time <- "2019-04-26_15_20_16"

# Raster processing functions

# Combine all predictors for a given day
```

Appendix A. Source Code

```r
extractPredictors <- function(date, crs, study_area, bb, pm_df,
                          static, scale_neph = scale_neph) {

  # Study Area Extent Raster used for resampling...
  a <- readRDS(study_area) %>% spTransform(CRSobj = crs) %>% raster(
    res = sp_res, vals = 0)

  # read PurpleAir data
  pa <- readRDS(pm_df) %>% spTransform(CRSobj = crs)

  # Final bounding box
  b <- readRDS(bb) %>% spTransform(CRSobj = crs) %>% raster(res = sp_res,
    vals = 0)

  # Read in our variable buffers
  s_buf_vars <- raster::stack(list.files("./data/buf_vars", pattern =
                                 paste0(date, ".*\.tif$"), full.names = T))

  # strip date from daily predictor layer names
  names(s_buf_vars) <- gsub(x = names(s_buf_vars), pattern = paste0("^X",
                           date, "\."), replacement = "")

  # # Read in our static buffers
  # s_buf_static <- raster::stack(list.files("./data/buf_static",
                                 pattern = ".*\.tif$", full.names = T))

  # Combine static and variable buffer stack
  s_buf <- stack(s_buf_vars, static)

  # # Reclassify NA that buffer script missed
  # s_buf[is.na(s_buf)] <- 0

  # # Add x and y values as predictors to raster layer
  #  # lat <- lon <- subset(s_buf, grep("pm2.5.median", names(s_buf),
  # value = T))
  # xy <- coordinates(s_buf)
  # lon[] <- xy[,1]
  # lat[] <- xy[,2]
  # names(lon) <- "x"
  # names(lat) <- "y"
  # plot(stack(lon, lat))
  # s_buf %>% stack(lon, lat)

  # # Compare differences between s_buf layers and df attributes
  # colnames(df[,which(colnames(df) %in% names(s_buf))])
```

Appendix A. Source Code

# Extract PurpleAir data

```r
pa <- pa[which(pa$date == gsub("\\" , "\", date)) ,]
```

# Simple extract grabs only the intersecting cell value
# s_extract_simple <- raster::extract(s_buf, pa, method = 'simple',
# sp = T, na.rm = T)
```
```
s_extract <- raster::extract(s_buf, pa, method = 'bilinear', sp = T
# na.rm = T)
```

# Convert to data.frame
```
df <- raster::as.data.frame(s_extract) %>% na.omit()
```

# Rename lat and lon to re-convert to raster later...
```
names(df)[names(df) == "coords.x1"] <- "lon"
```
```
names(df)[names(df) == "coords.x2"] <- "lat"
```

# Attach temporal variables

# These are categorical variables that get converted into integers
during modeling

# Add DayOfWeek variable
```
df$weekday <- lubridate::wday(df$date) %>% factor(levels = c(1:7))
```

# Add Season variable
```
df$season <- getSeason(df$date) %>% factor(levels = c(1:4))
```

# # Add Month Variable? Captures phenological differences in PM
# from vegetation?
```
df$month <- lubridate::month(df$date) %>% factor(levels = c(1:12))
```

# Weekend/Weekday Captures differences in on-road mobile emissions?
```
df$weekend <- ifelse(lubridate::wday(df$date) %in% c(6,7), 1, 0)
```
```
%>% factor(levels = c(0:1))
```

# strip date from daily predictor layer names
# names(df) <- gsub(x = names(df), pattern = paste0("\\" , date),
# replacement = "")

# Convert back to sp (used later)
```
xy <- df[,c("x", "y")]
```
```
sp <- SpatialPointsDataFrame(coords = xy, data = df)
```
```
# proj4string(sp) <- proj4string(s_extract)
```
```
Appendix A. Source Code

# Convert Absolute PM to Neph-scaled PM

# Explicitly define response variable as "Y"
# df <- cbind(Y=pa$data$pm2.5_cf_x, df)
df$Y <- df$pm2.5_cf_x

# if (scale_neph) {
#   # Determine relative PM concentration
#   df$Y <- df$Y - df$pm2.5.median
# #}

df$Y_scaled <- (df$Y - df$pm2.5.median)

# # add these extra variables as raster layers?
# df_sp <- df
# coordinates(df_sp) <- ~x+y
# gridded(df_sp) <- T
# dfr <- rasterFromXYZ(df_sp)
# These will be added within predictRasters instead?

return(df)
gc()

# wrap our stacking function into a date sequence generator

extractDailyPredictors <- function(start_date,
                                    end_date
                                    ,crs
                                    ,study_area
                                    ,bb
                                    ,pm_df
                                    ,scale_neph
                                    ) {

dates <- sequenceDates(start_date, end_date)

# Read in the static data once
static <- raster::stack(list.files("./data/buf_static", pattern = ".*\.*tif$", full.names = T))

df <- Reduce(rbind, pblapply(dates, function(x) {
  extractPredictors(date = x
        ,study_area = study_area
        ,pm_df = pm_df
        ,bb = bb
        ,scale_neph = scale_neph
        ,crs = crs
        ,study_area = study_area
        ,static = static
        ,df = df
        )
       )

return(df)
gc()
Appendix A. Source Code

183 ,bb = bb
184 ,crs = crs
185 ,static = static
186 ,pm_df = pm_df
187 ,scale_neph = scale_neph)
188 # parallelization takes place within each loop (faster)
189 ,cl = detectCores()-1
190 )
191
192 return(df)
193 gc()
194 }
195
196 # Running our function on multiple days
197 extract_time <- system.time(df <- extractDailyPredictors(start_date =
198 = "2018-09-01" )
199 # start_date
200 ,end_date =
201 "2018-02-16" # for quick profiling
202 ,crs = crs
203 ,study_area =
204 df
205 ,scale_neph =
206 saveRDS(extract_time, paste0("./models/extract_time/", clean_time, "_extract_time.RDS"))
207
def <- df[complete.cases(df),]
208 saveRDS(df, paste0("./models/df/", clean_time, "_df.RDS"))

Source/modeling.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2019-02-21
# Train RF model on predictor stack
# needed for nvim-R
Appendix A. Source Code

```
if (basename(getwd()) == "src") {
  setwd("../")
  getwd()
}

# load the necessary packages

if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(plyr, dplyr, magrittr, tools, rgdal, raster, rasterVis, corrplot, sp, rgeos, ggplot2, stringr, RColorBrewer, pbapply, parallel, randomForest, ranger, tibble, broom, lubridate
  #,mlr # one of these conflicts with raster somewhere...
  #,tuneRanger
  ,breakDown # visualize ranger model
  ,car
  ,lm.beta
  ,PerformanceAnalytics
  ,Matrix
  ,xgboost
  ,data.table
  ,mltools
  ,caret
  #,sjmisc
  #,sjPlot
)
```

# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

# CV plotting function
source("./src/ggRegression.R")

# Modeling

# Load in our most recent dataset by timestamp
# clean_time <- print_clean_time()
clean_time <- readRDS("./models/clean_time/clean_time.RDS")
# clean_time <- "2019-04-12_09_43_08"
# clean_time <- "2019-04-15_22_56_59"
# clean_time <- "2019-04-22_20_43_46"
# clean_time <- "2019-04-26_15_20_16"
# clean_time <- "2019-04-28_15_15_11"

# TODO Change this back later!
df <- readRDS(paste0("./models/df/", clean_time, "_df.RDS"))
# df <- readRDS(paste0("./models/df/df.RDS"))

# Exclude negative values in response variable
df <- df[df$Y>0,]

# Add constant to all elevation to avoid negative values
elev <- raster("./data/buf_static/elev.100m.tif")
min_elev <- min(values(elev))

# Setting up log transformed predictors for later
df$log_pm2.5.median <- log(df$pm2.5.median+1e-6)
df$log_fwy.100m <- log(df$fwy.100m+1e-6)
df$log_fwy.300m <- log(df$fwy.300m+1e-6)
df$log_fwy.500m <- log(df$fwy.500m+1e-6)
df$log_fwy.1000m <- log(df$fwy.1000m+1e-6)
df$log_elev.100m <- log(df$elev.100m+1e-6+abs(min_elev))
df$log_elev.300m <- log(df$elev.300m+1e-6+abs(min_elev))
df$log_elev.500m <- log(df$elev.500m+1e-6+abs(min_elev))
df$log_elev.1000m <- log(df$elev.1000m+1e-6+abs(min_elev))
df$log_pbl.min <- log(df$pbl.min+1e-6)
df$log_railroad.100m <- log(df$railroad.100m+1e-6)
df$log_railroad.300m <- log(df$railroad.300m+1e-6)
df$log_railroad.500m <- log(df$railroad.500m+1e-6)
df$log_railroad.1000m <- log(df$railroad.1000m+1e-6)
df$log_rmin <- log(df$rmin+1e-6)
saveRDS(df, paste0("./models/df/df.RDS"))

# setting seed for reproducibility
set.seed(6489)

# 70/30 Holdout
sub <- sample(1:nrow(df), floor(0.7*nrow(df)))
train <- df[sub,]
test <- df[-sub,]
saveRDS(train, paste0("./models/train/", clean_time, "_train.RDS"))
saveRDS(test, paste0("./models/test/", clean_time, "_test.RDS"))

# train <- readRDS(paste0("./models/train/", clean_time, "_train.RDS"))
# test <- readRDS(paste0("./models/test/", clean_time, "_test.RDS"))

# Correlation Plot EDA

# Loop through each base var and develop invididual corrplot

vars <- df %>% select_if(is.numeric) %>% colnames() %>%
str_replace_all("\\d\[1,4]\m\$", ") %>% unique()
lapply(vars, function(i) {
try({
cmatrix <- data.frame(df[,grepl(paste0("^Y|^", i), names(df))])
names(cmatrix)[names(cmatrix) == 'pm2.5.median'] <- 'neph'
# Exclude columns that all contain zero (produces NaN error later)
cmatrix <- cmatrix[, colSums(cmatrix != 0) > 0]
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77ADD", "#4477AA"))
res1 <- cor.mtest(cmatrix, conf.level = 0.95, na.rm=T)
p.mat <- cor.mtest(cmatrix)$p
corrplot(cor(cmatrix), col = rev(col(200)), order = "hclust", number.cex = 0.7,
,addCoef.col = "black"
,tl.col = "black", tl.srt = 90
,p.mat = p.mat
,sign.level = 0.05, insig = "blank", diag = F)
png(filename = paste0('./figures/corrplot/', clean_time, 'corrplot.png'))
Appendix A. Source Code

141 , height = 8
142 , width = 8
143 , units = "in"
144 , res = 300)
145 dev.off()
146 }
147 }
148 }

149 # buffers <- c("25m", "50m", "100m", "300m", "500m", "1000m")
150 #
151 # lapply(buffers, function(i) {
152 # try {
153 # cmatrix <- data.frame(df [, grepl(\"^Y$|pm2.5.median|pbl.max|pbl.min\", i), names(df)])
154 # names(cmatrix)[names(cmatrix) == 'pm2.5.median'] <- 'neph'
155 # cmatrix <- cmatrix[, colSums(cmatrix != 0) > 0]
156 # col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
157 # res1 <- cor.mtest(cmatrix, conf.level = 0.95, na.rm=T)
158 # p.mat <- cor.mtest(cmatrix)$p
159 # corrplot(corr(cmatrix), col = rev(col(200)), order = "hclust", number.cex = 0.7
160 #   ,addCoeff.col = "black"
161 # , tl.col = "black"
162 # , tl.srt = 90
163 # , p.mat = p.mat
164 # ,sig.level = 0.05
165 # , insig = "blank"
166 # , diag = F)
167 # png(filename = paste0(\'/figures/corrplot/\', clean_time, '_corrplot.png')
168 # , height = 8
169 # , width = 8
170 # , units = "in"
171 # , res = 300)
172 # dev.off()
173 # })
174 # }
175 #
176 # Summer/Winter sensitivity analysis
177 #
178 # eda w/ Meena
179 # spring <- df %>% filter(month %in% c(4, 5, 6))
180 # forestfire <- df %>% filter(month %in% c(8,9))
181 # winter <- df %>% filter(month %in% c(10,11,12))
182 #
183 # spring$class <- "spring"
Appendix A. Source Code

# forestfire$class <- "ff"
# winter$class <- "winter"
#
# season_df <- rbind(spring, forestfire, winter)
#
# season_df %>% ggplot(aes(y = Y, x = class)) +
#   geom_boxplot()
#
# df %>% ggplot(aes(y = Y, x = season)) +
#   geom_boxplot()
#
# quantile(spring$Y)
# quantile(forestfire$Y)
# quantile(winter$Y)
#
# quantile(df$arterial)
# quantile(df$Y[df$arterial > 1.37 & df$arterial < 2.27])
#
# asdf <- df %>% mutate(qart = ntile(arterial, 4), qmaj_art = ntile(maj_art, 4))
#   group_by(qart, qmaj_art, month) %>%
#   summarise(Y_q = mean(Y), n = n(), art_mean = mean(arterial), maj_art_mean = mean(maj_art))
#
# asdf <- df %>% mutate(qart = ntile(arterial, 4), qmaj_art = ntile(maj_art, 4))
#   group_by(qmaj_art, month) %>%
#   summarise(pm_mean = mean(Y), n = n(), art_mean = mean(arterial),
#             maj_art_mean = mean(maj_art))
#
# asdf$month <- as.integer(asdf$month)
#
# mod <- lm.beta(Y ~ arterial*evi, data = df)
# summary.lm.beta(mod)
# summary(mod)
# lm.beta(mod)
#
# Hyperparameter Tuning with Cross Validation

# Begin caret hyperparam tuning
#
# fit <- caret::train(
#   x = train[, predictors]
#   , y = response
#   , method = 'ranger'
#   , tuneGrid = grid
#   , trControl = fitControl
#   , importance = "permutation"
#   , num.trees = 500
# , respect.unordered.factors = T
# , keep.inbag = T
# )
#
## Print tuning duration in minutes
## (proc.time() - ptm) / 60
#
## Let’s plot RMSE ~ mtry
## plot(fit)
#
## Store the best performing model in a new object
## rfr_caret <- fit$finalModel
## rfr_caret
#
# Can’t get the formula notation to work with caret
## rfr_caret_formula <- caret::train(
##   data = train
##   , form = Y_scaled ~ Y-weekday-season
##   , method = ‘ranger’
##   , tuneGrid = grid
##   , trControl = fitControl
##   , num.trees = 500
##   , importance = ‘permutation’
##   , mtry = ceiling((length(which(predictors==T))/3)*1.0)
##   , respect.unordered.factors = T
##   , keep.inbag = T
##   , splitrule = ‘variance’
##   , min.node.size = 5 # regression
## )
#
## breakDown::broken(rfr_caret, train)
#
## Checkpoint our model objects
## saveRDS(fit, paste0("./models/fit/", clean_time, "_fit.RDS"))
## saveRDS(rfr, paste0("./models/rfr/", clean_time, "_rfr.RDS"))
#
# Ranger

# Following the DataCamp course principles for repeatedcv
fitControl <- caret::trainControl(method = "repeatedcv"
                                  , number = 5 # K-folds
                                  , repeats = 3
                                  , verboseIter= T)
#
# store start time
ptm <- proc.time()
# Train model differently depending on `scale_neph`

# Manually select predictors to train model on (`scale_neph` handles `pm2.5.median`)

# Select single buffer for each predictor based on the strongest correlation and instinctive sign

```r
initial_predictor_names <- c("fwy.100m" # largest buffer to smooth data
   ,"elev.300m" # maybe smooth this out to larger buffer prevent overfitting?
   ,"streets.100m" # include or not?
   ,"arterial" # negative relationship all buffers
   ,"pr" # pr is one of the better correlated predictors, but may be introducing noise...
   ,"th" # ignore wd because of resolution!
   ,"maj_art" # negative relationship/non-relationship for buffers
   ,"pb1.max"
   ,"pb1.min"
   ,"railroad.1000m" # strongest predictor
   ,"railyards.1000m" # negative relationship
   ,"rmin.1000m" # rh and temp are introducing noise to the model
   ,"rmax.1000m"
   ,"tmmn.1000m"
   ,"tmrx.1000m" # Strong positive relationship, dominated by forest fires...
   ,"vs.1000m" # Strong negative relationship, but data quality/resolution issues...
   ,"th.1000m" # omit entirely due to poor resolution?
   ,"weekday"
   ,"season"
   ,"month"
   ,"weekend"
   ,"barren_land.1000m"
   ,"cultivated_crops.100m" # Positive correlation
   ,"pop_dens.2000m" # Larger buffers don’t make a difference!
   ,"developed_high_intensity.500m" # no correlation with PM
   ,"developed_low_intensity.500m" # positive correlation at 500m, negative at 1000m
   ,"developed_medium_intensity.500m" # positive correlation for all buffers
   ,"developed_open_space.1000m" # take out later?```

# Adjusting model parameters based on the selected predictors
correlation "ndvi.100m" # largest buffer with negative correlation
  correlation "evi.100m" # largest buffer with negative correlation
  "deciduous_forest.1000m"
  "evergreen_forest.1000m"
  "mixed_forest.1000m"
  "shrub_scrub.100m" # Exclude?
  "hay_pasture.1000m" # positive correlation?
  "herbaceuous.500m" # negative at 500m, positive at 1000m
  "emergent_herbaceuous_wetlands.1000m"
  "open_water.100m" # LUR overfits this! (correlated with railroads...)
  "woody_wetlands" # positive correlation for all buffers
  "lon"
  "lat"
)

# Use hclust to determine which predictor to select for each collinear group

  cmatrix <- data.frame(df[,grepl(paste0("^Y$|^",paste0(initial_predictor_names, collapse = ' | '), "$"), names(df))])
  cmatrix <- df[, names(df) %in% c("Y", "pm2.5.median", initial_predictor_names)]
  names(cmatrix)[names(cmatrix) == 'pm2.5.median'] <- 'neph'
  # Exclude columns that all contain zero (produces NaN error later)
  cmatrix <- cmatrix[, colSums(cmatrix != 0) > 0]
  cmatrix <- dplyr::select_if(cmatrix, is.numeric)
  col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
  res1 <- cor.mtest(cmatrix, conf.level = 0.95, na.rm=T)
  sp.mat <- cor.mtest(cmatrix)$p
  saveRDS(cor(cmatrix), paste0("./models/cmatrix/", clean_time, "_cmatrix.RDS"))

  # Checkpoint our correlation matrix
  cdf <- as.data.frame(as.table(cor(cmatrix)))
  cdf %>% dplyr::filter(Var1 != Var2 & Var1 == "Y") %>% arrange(desc(Freq))

  # Figure out something better later
  ggplot(cdf, aes(reorder(Var2, Freq, sum), Freq)) +
Appendix A. Source Code

```r
geom_col() +
coord_flip() +
theme_bw()

corplot(cor(cmatrix), col = rev(col(200)), order = "hclust", number.cex = 0.5,
  addCoef.col = "black"
  type = "lower"
  method = "color"
  tl.col = "black"
  tl.srt = 90
  p.mat = p.mat
  sig.level = 0.10
  insig = "blank"
  diag = F)

abbr_names <- paste0({
  names(cmatrix) %>% sub(pattern = "\.\d+ m$", replacement = "") %>%
  sub(pattern = "[:punct:]", replacement = "") %>%
  abbreviate(minlength = 3)
}, ".", str_extract(names(cmatrix), "\d-m$")
)

abbr_names <- str_replace_all(abbr_names, "\\NA$", "")
names(cmatrix) <- abbr_names

# chart.Correlation(cmatrix, histogram = T)
cmatrix <- cmatrix[, colSums(cmatrix != 0) > 0]
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
res1 <- cor.mtest(cmatrix, conf.level = 0.95, na.rm=T)
p.mat <- cor.mtest(cmatrix)$p

png(filename = paste0("./figures/corplot/", clean_time, "_initial_corplot.png")
, height = 6
, width = 8
, units = "in"
, res = 300
)
corplot(cor(cmatrix), col = rev(col(200)), order = "hclust"
  addCoef.col = "black"
  tl.col = "black"
  tl.srt = 90
  number.cex = 0.5
  p.mat = p.mat
  sig.level = 0.05
```

---

```
```
Appendix A. Source Code

# , insig = "blank"
#, addrect = 7 # objectively set this somehow?
#, diag = F)

dev.off()

# Checkpoint our correlation matrix
cdf <- as.data.frame(as.table(cor(cmatrix)))
cdf %<>% dplyr::filter(Var1 != Var2 & Var1 == "Y") %>% arrange(desc(Freq))

# Figure out something better later
ggplot(cdf, aes(reorder(Var2, Freq, sum), Freq)) +
  geom_col() +
  coord_flip() +
  theme_bw()

#

# Prune predictor list based on corrplot hclust results? Or conduct two-step RF for variable selection?
predictor_names <- c("elev.100m" # maybe smooth this out to larger buffer prevent overfitting?
  ,"log_fwy.100m" # largest buffer to smooth data
  ,"fwy.100m" # largest buffer to smooth data
  ,"streets.100m" # include or not?
  ,"arterial" # negative relationship all buffers
  ,"pr" # pr is one of the better correlated predictors, but may be introducing noise...
  ,"th" # ignore wd because of resolution!
  ,"maj_art" # negative relationship/non-relationship for buffers
  ,"pbl.max"
  ,"pbl.min"
  ,"ndvi.100m" # largest buffer with negative correlation
  ,"evi.100m" # largest buffer with negative correlation
  ,"railroad.1000m" # strongest predictor
  ,"log_railroad.1000m" # strongest predictor
  ,"railyards.1000m" # negative relationship
  ,"pop_dens.2000m" # Should I exclude this and just use NLCD?
  ,"rmin" # rh and temp are introducing noise to the model
  ,"rmax.1000m"
  ,"tmmn.1000m"
"tmx.1000m"

# vs.1000m

# th.1000m' # omit entirely due to poor resolution?

#, "weekday"

, "season"

#, "month"

, "weekend"

#, "barren_land.1000m"

#, "cultivated_crops.100m" # Positive correlation with PM

#, "deciduous_forest.1000m"

#, "developed_high_intensity.500m" # no correlation with PM

#, "developed_low_intensity.100m" # positive correlation at 500m, negative at 1000m

#, "developed_medium_intensity.300m" # positive correlation for all buffers

#, "developed_open_space.1000m" # take out later?

#, "evergreen_forest.1000m"

#, "hay_pasture.1000m" # positive correlation? exclude?

#, "herbaceous.500m" # negative at 500m, positive at 1000m

#, "emergent_herbaceous_wetlands.1000m"

#, "mixed_forest.1000m"

#, "open_water.100m" # overfits?

#, "shrub_scrub.100m"

#, "woody_wetlands" # positive correlation for all buffers

#, "lon"

#, "lat"

)

if (scale_neph) {
  predictors <- names(train) %in% predictor_names
  form <- as.formula(paste0("Y_scaled~", paste(predictor_names,
    collapse = "+"))
} else {
  predictors <- names(train) %in% c("pm2.5.median", predictor_names)
  form <- as.formula(paste0("Y~pm2.5.median+", paste(predictor_names,
    collapse = "+"))
}

# Temporary kitchen-sink formula!
# form <- as.formula("Y~lat-lon-Y_scaled-pm2.5_cf_x-date-node")
Appendix A. Source Code

```r
# predictor_names <- train[, names(train) %in% c("lat", "lon", "Y", "Y_scaled", "node", "date", "pm2.5_cf_x")]
# predictors <- names(train) %in% predictor_names

cmatrix <- df[, names(df) %in% c("Y", "pm2.5.median", predictor_names)] %>% dplyr::select_if(is.numeric)

col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
res1 <- cor.mtest(cmatrix, conf.level = 0.95, na.rm=T)
p.mat <- cor.mtest(cmatrix)$p

png(filename = paste0("./figures/corrplot/", clean_time, "_final_corrplot.png"),
     height = 6,
     width = 8,
     units = "in",
     res = 300)
corrplot(cor(cmatrix), col = rev(col(200)), order = "hclust",
     #,addCoef.col = "black"
     ,tl.col = "black"
     ,tl.srt = 90
     #,number.cex = 0.5
     #,p.mat = p.mat
     #,sig.level = 0.05
     #,insig = "blank"
     ,addrect = 7 # objectively set this somehow?
     ,diag = F)
dev.off()

# Checkpoint our model formula (ease of reproducibility later)
saveRDS(form, paste0("./models/form/", clean_time, "_form.RDS"))

# Define our conditional response variable (absolute or scaled PM)
if(scale_neph) {
    response <- train[, names(train) == 'Y_scaled'] # Scaled Response
} else {
    response <- train[, names(train) == 'Y'] # Scaled Response
}

# Ensure the correct predictors are selected
names(train[, predictors])
```
497  # LUR

498  # Exclude weird negative PA value
499  df <- df[df$Y > 0,]
500  sub2 <- sample(1:nrow(df), 5000) # limit of shapiro test
501  df2 <- df[sub2,]
502  df2 <- train

506  # Independent Variable Transformations

507  # Limit influence of outliers via log transform
508  lapply(c("pm2.5.median", predictor_names), function(n) {
509    if (is.numeric(train[[n]])) {
510      par(mfrow=c(1,2))
511      hist(train[[n]], main = n, xlab="")
512      boxplot(train[[n]], main = n)
513    }
514  })

516  # Predictors with outliers
517  # df2$log_pm2.5.median <- log(df2$pm2.5.median+1e-6)
518  # df2$log_fwy.100m <- log(df2$fwy.100m+1e-6)
519  # df2$log_elev.100m <- log(df2$elev.100m+1e-6)
520  # df2$log_pbl.min <- log(df2$pbl.min+1e-6)
521  # df2$log_railroad.100m <- log(df2$railroad.100m+1e-6)
522  # df2$log_rmin <- log(df2$rmin+1e-6)
523
524  log_predictors <- dplyr::select(train, starts_with("log")) %>%
525    colnames()

527  lapply(log_predictors, function(n) {
528    if (is.numeric(train[[n]])) {
529      par(mfrow=c(1,2))
530      hist(train[[n]], main = n, xlab="")
531      boxplot(train[[n]], main = n)
532    }
533  })

534  par(mfrow=c(1,1))

538  # Response Variable Transformation
# Response variable’s residuals normally distributed?
qqnorm(train$Y)
qqline(train$Y, col = "red", lwd = 2)

# TODO make sure this still makes sense...

# Formula for independent variables
iv <- "log_pm2.5.median+log_fwy.100m+log_elev.100m+log_pbl.min+ndvi.100m+log_railroad.100m+pop_dens.100m+log_rmin+month"
iv <- "pm2.5.median+log_fwy.100m+elev.100m+pbl.min+ndvi.100m+log_railroad.100m+pop_dens.100m+rmin+season"

# Create a landuse regression model too?
form_raw <- paste0("Y~", iv)
# lm <- lm(form, data = train)
summary(lm)
par(mfrow=c(2,2))
plot(lm)
par(mfrow=c(1,1))

# Box-Cox Transformation
bc <- MASS::boxcox(object=lm, data = train)
best_lambda <- bc$x[which(bc$y==max(bc$y))]}
train$Y_bc <- train$Y^best_lambda
qqnorm(train$Y_bc)
qqline(train$Y_bc, col = "red", lwd = 2)

# Log Transformation
train$Y_log <- log(train$Y+1e-6)
form_log <- "Y_log~log_pm2.5.median+log_fwy.100m+log_elev.100m+log_pbl.min+ndvi.100m+log_railroad.100m+pop_dens.100m+log_rmin+season+
month"
form_log <- as.formula(paste0(’Y_log~pm2.5.median+’, paste(predictor_names, collapse = ' + ')))
form_log <- paste0("Y_log~", iv)

# Box-Cox Transformation
bc <- MASS::boxcox(object=lm, data = train)
best_lambda <- bc$x[which(bc$y==max(bc$y))]}
train$Y_bc <- train$Y^best_lambda
qqnorm(train$Y_bc)
qqline(train$Y_bc, col = "red", lwd = 2)
Appendix A. Source Code

```r
lm_log <- lm(form_log, data = train)
summary(lm_log)
par(mfrow=c(2,2))
plot(lm_log)
par(mfrow=c(1,1))

# Normality of residuals
hist(residuals(lm))
hist(residuals(lm_bc))
hist(residuals(lm_log))
boxplot(residuals(lm))
boxplot(residuals(lm_bc))
boxplot(residuals(lm_log))

# We will always reject null when sample size is too great! Rely on
# graphical assessment instead!
try(shapiro.test(residuals(lm)))
try(shapiro.test(residuals(lm_bc)))
try(shapiro.test(residuals(lm_log)))

# Heteroscedasticity of residuals

# We will always reject null when sample size is too great!
	split_lm <- median(lm$fitted) #find a median of residuals
g1_lm <- lm$resid[lm$fitted < split_lm] #divide all residuals into two
g2_lm <- lm$resid[lm$fitted > split_lm]
var.test(g1_lm, g2_lm) #test the equal variance assumption
	split_bc <- median(lm_bc$fitted) #find a median of residuals
g1_bc <- lm$resid[lm_bc$fitted < split_bc] #divide all residuals into two
g2_bc <- lm$resid[lm.bc$fitted > split_bc]
var.test(g1_bc, g2_bc) #test the equal variance assumption
	split_log <- median(lm_log$fitted) #find a median of residuals
g1_log <- lm$resid[lm_log$fitted < split_log] #divide all residuals into two
g2_log <- lm$resid[lm_log$fitted > split_log]
var.test(g1_log, g2_log) #test the equal variance assumption

# Multicollinearity

try(car::vif(lm))
try(car::vif(lm_bc))
try(car::vif(lm_log))
```
# Reduce full model to parsimonious model

# TODO verify model

# Hybrid Approach

lm_step <- step(lm, direction="both")

lm_step <- MASS::stepAIC(lm, direction = "both")

lm_step <- caret::train(as.formula(form), data = train, method = "lmStepAIC")

lm_step$finalModel

summary(lm_step$finalModel)

summary(lm)

car::vif(lm_step$finalModel) # nice, lower VIFs now

# anova(lm_step$finalModel, lm_log) # caret messes with response variable name...

# Checkpoint our explanatory model

lm_final <- lm_step$finalModel

summary(lm_final)

# TODO manually exclude insignificant variables before stepAIC?

# TODO handle categorical variables after stepAIC?

# TODO handle categorical variables significance?

# Fixing this by hand for now...

# lm_final <- lm("Y~pm2.5.median+elev.100m+log_fwy.100m+pbl.min+ndvi.100m+log_railroad.100m+rmin+season+weekend+lat", train)

form <- "Y~pm2.5.median+elev.100m+pbl.min+log_railroad.1000m+rmax.1000m+tmmx.1000m+season+weekend+lat"

lm_final <- lm(form, train)

summary(lm_final)

# # TODO stepVif with initial_predictors instead of hclust?

# df2$Y_log <- log(df2$Y+1)

# form_log <- as.formula(paste0("Y_log~pm2.5.median+", paste(predictor_names, collapse = ’+’)))

# lm_log <- lm(form_log, data = df2)

# summary(lm_log)

# summary(lm)

# Old method (without IV inspection, assumptions, and explanatory model)

# lm_final <- lm(form, data = df)

# summary(lm) # how to get standardized effects?

summary.lm.beta(lm_final, standardized = F)
# Checkpoint our LUR
saveRDS(lm_final, paste0("./models/lm/", clean_time, "_lm.RDS"))

# Build grid of possible hyperparam combos

grid <- expand.grid(mtry = c(ceiling((length(which(predictors==T))/3) * 1.0)
            , ceiling((length(which(predictors==T))/3)*1.5)
            , ceiling((length(which(predictors==T))/3)*2.0)
            , ceiling((length(which(predictors==T))/3)*2.5)
            , ceiling((length(which(predictors==T))/3)*2.7))

rfr <- ranger(data = cbind(train[, predictors], Y = response)
            , dependent.variable.name = 'Y'
            , num.trees = 500
            , importance = 'permutation'
            , mtry = ceiling((length(which(predictors==T))/3)*1.0)
            , respect.unordered.factors = T
            , keep.inbag = T
            , splitrule = 'variance'
            , min.node.size = 5 # regression
            )

rfr

saveRDS(rfr, paste0("./models/rfr/", clean_time, "_rfr.RDS"))

# # Here’s how to use ranger with the index notation
# rfr2 <- ranger(}
# data = train
# Y~.−Y_scaled−weekday−season−pm2_5_cf_x
# data = cbind(train[, predictors], Y_scaled = train$Y_scaled)
# ,dependent.variable.name = 'Y_scaled'
data = cbind(train[, predictors], Y = response)
# ,dependent.variable.name = 'Y'
# ,num.trees = 500
# ,importance = 'permutation'
# ,mtry = ceiling((length(which(predictors==T))/3)*1.0)
# ,respect.unordered.factors = T
# ,keep.inbag = T
# ,splitrule = 'variance'
# ,min.node.size = 5 # regression

# rfr2 # Not good right now! Fix predictor indexing to match formula!
!!

# XGBoost

x_train <- train[,predictors] %>% as.data.table() %>% mltools::one_hot(dropUnusedLevels = F)
y_train <- train$Y
x_test <- test[,predictors] %>% as.data.table() %>% mltools::one_hot(
   dropUnusedLevels = F)
y_test <- test$Y

# Test that factor encoding is preserved with month subset
df_dt_test <- df_dt %>% filter(month == "1") %>% as.data.table()
df_oh_test <- mltools::one_hot(df_dt_test, dropUnusedLevels = F)

xgb_trcontrol = trainControl(
   method = "cv",
   number = 5,
   allowParallel = TRUE,
   verboseIter = FALSE,
   returnData = FALSE)

xgbGrid <- expand.grid(nrounds = c(100,200), # this is n_estimators in the python code above
   max_depth = c(10, 15, 20, 25),
colsample_bytree = seq(0.5, 0.9, length.out = 5),
   ## The values below are default values in the
   etas = c(0.1),
   gamma=0,
min_child_weight = 1,
subsample = 1)

xgb_model <- caret::train(x_train, y_train, method = "xgbTree"
                         ,trControl = xgb_trcontrol, tuneGrid =
                         xgbGrid, type="prob")

xgb_model$bestTune

predicted <- predict(xgb_model, x_test)
residuals <- y_test - predicted
RMSE <- sqrt(mean(residuals^2))
cat("The root mean square error of the test data is ", round(RMSE,3), 
    "\n")

y_test_mean = mean(y_test)
# Calculate total sum of squares
tss = sum((y_test - y_test_mean)^2)
# Calculate residual sum of squares
rss = sum(residuals^2)
# Calculate R-squared
rsq = 1 - (rss/tss)
cat("The R-square of the test data is ", round(rsq,3), "\n")
saveRDS(xgb_model, paste0("./models/xgb/", clean_time, "_xgb.RDS"))

# Random Forest

# Save a randomForest version too
rf <- randomForest(x = train[, predictors]
                    ,y = response
                    ,ntree = 500
                    ,importance = T
                    ,mtry = rf$mtry # just grab the best mtry from
                    the ranger hyperparam tuning for now...
                    ,nodesize = 5
                    ,nPerm = 1
                    ,do.trace = F
                    ,na.action = na.fail
                    ,keep.inbag = T
                    )

varImpPlot(rf)
plot(rf)
rf

# Checkpoint our model output
saveRDS(rf, paste0("./models/rf/", clean_time, ",rf.RDS"))

# Variable importance plots

# Top 20 predictors
rfr$variable$importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  dplyr::select(variable = "rowname", IncNodePurity = ".") %>%
  arrange(IncNodePurity) %>%
  tail(n = 20) %>%
  mutate(variable =forcats::fct_inorder(variable)) %>%
  ggplot() +
  geom_col(aes(x = variable, y = IncNodePurity)) +
  coord_flip() +
  theme_bw()

  ggsave(plot = last_plot(),
  filename = paste0("./figures/variable_importance/", clean_time, ",varImpPlot20.png")
  )

# Top 50 predictors
rfr$variable$importance %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  dplyr::select(variable = "rowname", IncNodePurity = ".") %>%
  arrange(IncNodePurity) %>%
  tail(n = 50) %>%
  mutate(variable =forcats::fct_inorder(variable)) %>%
  ggplot() +
  geom_col(aes(x = variable, y = IncNodePurity)) +
  coord_flip() +
  theme_bw()

  ggsave(plot = last_plot(),
  filename = paste0("./figures/variable_importance/", clean_time, ",varImpPlot50.png")
  )

# Convert relative PM to Absolute if scale_neph is T

if (scale_neph) {
  # Add Neph median to predicted spatial residual
}
predictions <- predict(rfr, test[,predictors])$predictions+test$pm2.5.median
predict_full <- predict(rfr, df[,predictors])$predictions+df$pm2.5.median

# Add Neph median to observed spatial residual
test$X <- test$Y_scaled + test$pm2.5.median
df$X <- df$Y_scaled + df$pm2.5.median

# Build test regression dataframe
cvr <- data.frame(x = test$X, y = predictions)
cvr_full <- data.frame(x = df$X, y = predict_full)

# Create test regression model
rfc_cv <- lm(y~x, data = cvr)
rfc_cv_full <- lm(y~x, data = cvr_full)

# Build test regression dataframe
cvr <- data.frame(x = test$Y, y = predict(rfr, test[,predictors])$predictions)
cvr_full <- data.frame(x = df$Y, y = predict(rfr, df[,predictors])$predictions)

# Create regression validation model
rfc_cv <- lm(y~x, data = cvr)
rfc_cv_full <- lm(y~x, data = cvr_full)

# Build another test regression with lm instead of rfr
cv_lm <- data.frame(x = test$Y, y = predict(lm_final, test[,predictors]))
cv_xgb <- data.frame(x = y_test, y = predicted)
# rank-deficit warning ignored for now...

# Store summary of model as new object
modr <- summary(rfc_cv)
modr

# modr_full <- summary(rfc_cv_full)
# modr_full

# Checkpoint our regression validation model object summary
saveRDS(modr, paste0("./models/mod_summary/", clean_time, "_mod_summary.RDS"))
# modr <- readRDS(paste0("./models/mod_summary/", clean_time, "_mod_summary.RDS"))

# TODO verify p-value thing... lm() should be suffice

# Compute RMSE on test data versus predictions
rmse <- sqrt(mean((cvr$y - cvr$x)^2))
rmse

# Plot Validation Regression

testRegression(df = cvr, x = "x", y = "y", outdir = "./figures/lm_obs_pred/", clean_time = paste0(clean_time, "_rf"), reduced = F)
testRegression(df = cv_xgb, x = "x", y = "y", outdir = "./figures/lm_obs_pred/", clean_time = paste0(clean_time, "_xgb"), reduced = F)
testRegression(df = cv_lm, x = "x", y = "y", outdir = "./figures/lm_obs_pred/", clean_time = paste0(clean_time, "_lm"), reduced = F)
# testRegression(df = cvr, x = "x", y = "y", outdir = "./figures/misc/", clean_time = clean_time, reduced = F)
# testRegression(df = cvr_full, x = "x", y = "y", outdir = "./figures/lm_obs_pred/full_", clean_time = clean_time, reduced = F)

# Plot regression with train and test data
# TODO Store model summary stats in table for safe keeping

# LOOCV

# Determine viable nodes
n_obs_node <- train %>% group_by(node) %>% summarise(n = n())
n_threshold <- 100

p <- ggplot(n_obs_node, aes(x = reorder(node, -n), y = n)) + geom_col() + geom_hline(yintercept = n_threshold, col = "firebrick", linetype = "dashed") + theme_minimal() + theme(axis.text.x = element_text(angle = 90, vjust = 0.5), axis.title.x = element_blank())
png("./figures/loocv/n_threshold/n_threshold.png", width = 8, height = 6, units = "in")
Appendix A. Source Code

914   ,res = 300)
915   P
916   dev.off()
917
918   # Exclude nodes with too few observations
919   n_obs_node %>% filter(n >= n_threshold)
920
921   # Extract names from remaining nodes
922   nodes <- unique(n_obs_node$node)
923
924   # Loop through each node during LOOCV
925
926   # TODO keep low n nodes during modeling or not?
927   loocv_results <- Reduce(rbind, pblapply(nodes, function(n) {
928     # Exclude node of interest
929     df_loocv <- filter(train, node %!in% n)
930
931     # Run models -------------------
932     # TODO log transform lm predictors???
933     # TODO include k-folds during modeling?
934     lm_loocv <- lm(form, df_loocv)
935     rf_loocv <- ranger(
936       data = df_loocv
937       ,form
938       #x = train[, predictors]
939       #, y = response
940       #, dependent.variable.name = 'YScaled'
941       ,num.trees = 500
942       ,importance = 'permutation'
943       ,mtry = ceiling((length(which(predictors==T))/3)*1.0)
944       ,respect.unordered.factors = T
945       ,keep.inbag = T
946       ,splitrule = 'variance'
947       ,min.node.size = 5 # regression
948     )
949
950   # Predict holdout node -------------------
951   holdout_loocv <- filter(train, node %in% n)
952
953   holdout_loocv$pred_lm <- predict(lm_loocv, holdout_loocv)
954   holdout_loocv$pred_rf <- predict(rf_loocv, holdout_loocv)$predictions
955
956   # Clean up the node name -------------------
957   n <- str_replace_all(n, "[:punct:]", " ")
958   n <- str_squish(n)
\begin{verbatim}
  n <- str_replace_all(n,"\s+"," ")
  # Checkpoint these models for ensemble predicting
  saveRDS(lm_loocv, paste0("./models/loocv_models/lm/", clean_time, "_", n, "_lm_loocv.RDS"))
  saveRDS(rf_loocv, paste0("./models/loocv_models/rf/", clean_time, "_", n, "_rf_loocv.RDS"))
  # Visualize CV
  testRegression(df = holdout_loocv, x = "Y", y = "pred_lm", outdir = 
                  ".figures/loocv/lm/", clean_time = paste0(clean_time, "_lm_lo_", n),
                  reduced = F)
  testRegression(df = holdout_loocv, x = "Y", y = "pred_rf", outdir = 
                 ".figures/loocv/rf/", clean_time = paste0(clean_time, "_rf_lo_", n),
                 reduced = F)
  # Store obs–predicted model results
  mod_lm <- lm(pred_lm~Y, data = holdout_loocv)
  mod_rf <- lm(pred_rf~Y, data = holdout_loocv)
  summary_lm <- broom::glance(mod_lm)
  summary_rf <- broom::glance(mod_rf)
  summary_lm$slope <- mod_lm$coef[[2]]
  summary_rf$slope <- mod_rf$coef[[2]]
  summary_lm$intercept <- mod_lm$coef[[1]]
  summary_rf$intercept <- mod_rf$coef[[1]]
  summary_lm$model_type <- "lm"
  summary_rf$model_type <- "rf"
  summary_df <- rbind(summary_lm, summary_rf)
  return(summary_df)
}

# EDA LOOCV

loocv_results_lm <- loocv_results %>% filter(model_type == "lm")
\end{verbatim}
Appendix A. Source Code

```r
loocv_results_rf <- loocv_results %>% filter(model_type == "rf")
loocv_results %>% group_by(model_type) %>% summarise_if(is.numeric, mean)
loocv_results %>% ggplot(aes(x = model_type, y = adj.r.squared, fill = model_type)) + geom_violin() + guides(fill = F) + theme_minimal() + theme(axis.text = element_text(face = "bold"), axis.title = element_text(face = "bold"))
loocv_results %>% ggplot(aes(x = model_type, y = sigma, fill = model_type)) + geom_violin() + guides(fill = F) + theme_minimal() + ylab("rmse") + theme(axis.text = element_text(face = "bold"), axis.title = element_text(face = "bold"))
loocv_results %>% ggplot(aes(x = model_type, y = slope, fill = model_type)) + geom_violin() + guides(fill = F) + theme_minimal() + theme(axis.text = element_text(face = "bold"), axis.title = element_text(face = "bold"))
loocv_results %>% ggplot(aes(x = model_type, y = intercept, fill = model_type)) + geom_violin() + guides(fill = F) + theme_minimal() + theme(axis.text = element_text(face = "bold"), axis.title = element_text(face = "bold"))

# LOOCV Ensemble predicting the holdout

lm_mods <- lapply(list.files("./models/loocv_models/lm/", paste0(clean_time, ".*\.RDS$"), full.names = T), readRDS)
rf_mods <- lapply(list.files("./models/loocv_models/rf/", paste0(clean_time, ".*\.RDS$"), full.names = T), readRDS)
lm_preds <- lapply(1:length(lm_mods), function(m) {

```

cv_lm <- data.frame(x = test$Y, y = predict(lm.mods[[m]], test[, predictors]), i = seq(1:length(test$Y)))
return(cv_lm)
}
rf_preds <- lapply(1:length(rf.mods), function(m) {
  cv_rf <- data.frame(x = test$Y, y = predict(rf.mods[[m]], test[, predictors])$predictions, i = seq(1:length(test$Y)))
  return(cv_rf)
})
# Compute mean of ensemble prediction
ens_lm_cv <- data.table::rbindlist(lm_preds)$lapply(.SD, mean, list(i))
ens_rf_cv <- data.table::rbindlist(rf_preds)$lapply(.SD, mean, list(i))
testRegression(df = ens_lm_cv, x = "x", y = "y", outdir = ".\figures\lm_obs_pred/", clean_time = paste0(clean_time, "_lm_ensemble"), reduced = F)
testRegression(df = ens_rf_cv, x = "x", y = "y", outdir = ".\figures\lm_obs_pred/", clean_time = paste0(clean_time, "_rf_ensemble"), reduced = F)
# testRegression(df = ens_xgb_cv, x = "x", y = "y", outdir = ".\figures/\lm_obs_pred/", clean_time = paste0(clean_time, "_xgb_ensemble"), reduced = F)

Source/predict.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2019–02–21
# Predict PM2.5 surfaces with RF model
# needed for nvim–R
if (basename(getwd()) == "src") {
  setwd("../")
  getwd()
}
# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
Appendix A. Source Code

```r
# Load in global variables
source("./src/init.R")

# Load in misc function
source("./src/functions.R")

clean_time <- readRDS("./models/clean_time/clean_time.RDS")
# clean_time <- "2019-04-12_09_43_08"
# clean_time <- "2019-04-15_22_56_59"
# clean_time <- "2019-04-22_20_43_46"
# clean_time <- "2019-04-26_15_20_16"
# clean_time <- "2019-06-10_15_31_05"

# These are used to extract predictor names... must match the LOOCV models!
rfr <- readRDS(paste0("./models/rfr/", clean_time, ".rfr.RDS"))
```

Appendix A. Source Code

64 rf <- readRDS(paste0("./models/rf/", clean_time, "_rf.RDS"))
65 lm <- readRDS(paste0("./models/lm/", clean_time, "_lm.RDS"))
66 xgb <- readRDS(paste0("./models/xgb/", clean_time, "_xgb.RDS"))
67
68 # Ensemble mod lists
69 lm.mods <- lapply(list.files("./models/loocv_models/lm/", paste0(clean_time, ".*\.RDS$"), full.names = T), readRDS)
70 # rf.mods <- lapply(list.files("./models/loocv_models/rf/", paste0(clean_time, ".*\.RDS$"), full.names = T), readRDS)
71
72 # Scale extracted elevation data based on global minima
73 elev <- raster("./data/buf_static/elev.100m.tif")
74 min_elev <- min(values(elev))
75
76 # Predictions
77 predictRasters <- function(date = date
78   #, crs = crs
79   #, rf = rf
80   , scale_neph = scale_neph
81   #, predictors = predictors
82   , static = static
83   #, rfr = rfr
84 ) {
85
86   # TODO stack list of desired predictors instead of all of them?
87   # read in each date’s raster stack previously created in 
88   # extractPredictors
89   s <- stack(list.files("./data/buf_vars", pattern = date, full.names = T))
90
91   # strip date from daily predictor layer names
92   names(s) <- gsub(x = names(s), pattern = paste0("^X", date, "\."), replacement = "")
93
94   static <- stack(list.files("./data/buf_static", pattern = "\..tif$", full.names = T))
95   s <- stack(s, static)
96
97   # Reduce stack size to increase performance
98   lm_names <- names(lm$coefficients)
99   rf_names <- names(rfr$variable.importance)
100  # xgb_names <- names(xgb$pred) # should match RF model
101
102  mod.names <- unique(c(lm_names, rf_names))
103  mod.names <- sub("^log\_", "", mod.names)
104  suppressWarnings(s <- subset(s, mod.names))
# Predict
# convert to dataframe before prediction
df <- as.data.frame(s, xy = T, na.rm = T)
# This takes ~6-minutes per stack!
# system.time(df <- as.data.frame(s, xy = T, na.rm = T))

# Is this faster than converting the whole stack?
# system.time(
#   df <- Reduce(cbind, pblapply(1:nlayers(s), function(n) {
#     if (n < nlayers(s)) {
#      asdf <- as.data.frame(s[[n]], na.rm = T)
#     } else {
#       asdf <- as.data.frame(s[[n]], xy = T, na.rm = T)
#     }
#     # print(names(asdf))
#     return(asdf)
#   }), cl = detectCores(-1)
# )
#
# Change x, y columns to lon, lat
names(df)[names(df) == "x"] <- "lon"
names(df)[names(df) == "y"] <- "lat"

# Setting up log transformed predictors for later
df$log_pm2.5.median <- log(df$pm2.5.median+1e-6)
df$log_fwy.100m <- log(df$fwy.100m+1e-6)
df$log_fwy.300m <- log(df$fwy.300m+1e-6)
df$log_fwy.500m <- log(df$fwy.500m+1e-6)
df$log_fwy.1000m <- log(df$fwy.1000m+1e-6)
df$log_elev.100m <- log(df$elev.100m+1e-6+abs(min_elev))
df$log_elev.300m <- log(df$elev.300m+1e-6+abs(min_elev))
df$log_elev.500m <- log(df$elev.500m+1e-6+abs(min_elev))
df$log_elev.1000m <- log(df$elev.1000m+1e-6+abs(min_elev))
df$log_pbl.min <- log(df$pbl.min+1e-6)
# df$log_railroad.100m <- log(df$railroad.100m+1e-6)
# df$log_railroad.300m <- log(df$railroad.300m+1e-6)
# df$log_railroad.500m <- log(df$railroad.500m+1e-6)
df$log_railroad.1000m <- log(df$railroad.1000m+1e-6)
# df$log_rmin <- log(df$rmin+1e-6)

# Add lat and lon to stack?
# s $ l a t <- df $ l a t
# s $ l o n <- df $ l o n

# system.time({
# s _ e x t r a c t <- raster::extract(s, extent(bb))
# s _ extract _sp <- raster::extract(s, extent(bb), sp=T, na.rm = T, 
# method = 'bilinear')
# df <- s _ extract %>% raster::as.data.frame() %>% na.omit()
# df _sp <- s _ extract _sp %>% raster::as.data.frame() %>% na.omit()
# })

# So we need to add l a t , l o n , and temporal variables to our df  
# after converting from stack


clean_date <- hyphenDate(date)

# Add DayOfWeek variable
df$weekday <- lubridate::wday(clean_date) %>% factor(levels = c(1:7))

# Add Season variable
df$season <- getSeason(clean_date) %>% factor(levels = c(1:4))

# # Add Month Variable? Captures phenological differences in PM 
# from vegetation?
df$month <- lubridate::month(clean_date) %>% factor(levels = c(1:12))

# Weekend/Weekday Captures differences in on-road mobile emissions? 
df$weekend <- ifelse(lubridate::wday(clean_date) %in% c(6,7), 1, 0) 
%>% factor(levels = c(0:1))

# Store Neph data in separate vector (it is deleted in following 
# line!)

pm2.5.median <- df$pm2.5.median

# Exclude any extra predictors from input data not required by 
# ranger.predict (pm2.5.median)
df %>% dplyr::select(names(rfr$variable.importance))  # dplyr to 
# the rescue

# Determine the difference in predictors between input data and 
# ranger.predict

missing_model_predictors <- names(rfr$variable.importance)[which( 
    names(rfr$variable.importance) %in% colnames(df) )]
extra_input_predictors <- colnames(df)[which(colnames(df) %in% 
    names(rfr$variable.importance))]

# Confirm there are no missing expected predictors in our df
if (length(missing_model_predictors) > 0 || length(extra_input_predictors) > 0) {
    print(paste0("Number of missing expected predictors: ", missing_model_predictors))
    print(paste0("Number of extra input predictors: ", extra_input_predictors))
    stop()
} else {
    # Single model predictions

    # Index our input data on the predictors expected by the model!
    system.time(pred_rfr <- predict(rfr
        , data = df[, names(rfr$variable.importance)]
        , verbose = T
        , type = "response"
        , type = "se"
    ))

    # # Figure out how to predict raster with linear model...
    # system.time(pred_lm <- predict(lm, df, se.fit = T))

    # system.time(pred_rf <- predict(model = rf
        , object = s
        , type = "response"
        , progress = "text"
    ))

    # TODO figure out 1d predict time with xgb???
    # # XGBoost
    # df_xgb <- df %>% as.data.table() %>%
    # mltools::one_hot(dropUnusedLevels = F) %>%
    # # feature order is necessary for xgb!!!!
    # dplyr::select(xgb$finalModel$feature_names)
    # system.time(pred_xgb <- predict(xgb, df_xgb))

    # Ensemble predictions

    lm_preds <- lapply(1:length(lmMods), function(m) {
        pred_m <- predict(lmMods[[m]], df, se.fit = T)
        ens_df <- data.frame(i = seq(1:nrow(df)), fit = pred_m$fit, se = pred_m$se.fit)
        return(ens_df)
    })
# RF ensemble predictions requires too much memory!
# rf_preds <- lapply(1:length(rf_mods), function(m) {
#   pred_m <- predict(rf_mods[[m]], data = df[,names(rfr$variable.importance)], type = "se")
#   ens_df <- data.frame(i = seq(1:nrow(df)), fit = pred_m$predictions, se = pred_m$se)
#   return(ens_df)
# })

# # Compute mean of ensemble predictions
ens_lm <- data.table::rbindlist(lm_preds[,lapply(.SD, mean), list(i)]
# ens_rf <- data.table::rbindlist(rf_preds[,lapply(.SD, mean), list(i)]

# if neph data is an offset
if(scale_neph) {
# Single model method _________________________________
# if the neph data scales the prediction
s$"pm2.5.rf" <- pred_rfr$predictions + pm2.5.median
# s$"pm2.5.lm" <- pred_lm + pm2.5.median
# s$"pm2.5.xgb" <- pred_xgb + pm2.5.median

# Ensemble method _________________________________
# s$"pm2.5.rf" <- ens_rf$fit + pm2.5.median
s$"pm2.5.lm" <- ens_lm$fit + pm2.5.median

# All of these methods return the same plot
# # trying Jackson’s methods (returns the same looking raster layer)
# template <- s[[1]]
# template[complete.cases(df)] <- pred_rfr$predictions + pm2.5.median
# plot(template)
# values(template) <- pred_rfr$predictions + pm2.5.median
# plot(template)
# plot(s$pm2.5.predicted)
}

} else {
# Single model predictions _________________________________
# if neph data is a predictor
s$"pm2.5.rf" <- pred_rfr$predictions
# s$"pm2.5.lm" <- pred_lm$fit
Appendix A. Source Code

```r
# s$"pm2.5.xgb" <- pred_xgb

# ensemble predictions
# s$"pm2.5.rf" <- ens_rf$fit
s$"pm2.5.lm" <- ens_lm$fit
#

# Return prediction SE for each model
s$pm2.5.rf.se <- pred_rfr$se
s$pm2.5.lm.se <- pred_lm$se

# Ensemble error method
s$pm2.5.rf.se <- ens_rf$se
s$pm2.5.lm.se <- ens_lm$se
# s$pm2.5.xgb.se <- ""

# # extract just the prediction layer
# pred_rf <- s$pm2.5.rf
# names(pred_rf) <- paste0(date, ".pm2.5.rf."),
# pred_lm <- s$pm2.5.lm
# names(pred_lm) <- paste0(date, ".pm2.5.lm."),
# pred_xgb <- s$pm2.5.xgb
# names(pred_xgb) <- paste0(date, ".pm2.5.xgb."),

pm_stack <- s[[which(grepl("pm2.5." , names(s)))]]
# % dropLayer("pm2.5.median")

names(pm_stack) <- paste0(date, ".", names(pm_stack))
m(s)
rm(df)
gc()

writeRaster(pm_stack, filename = paste0("./data/predicted_raster/", names(pm_stack), ".tif"),
            bylayer = T,
            format = "GTiff",
            overwrite = T)

rm(pm_stack)
gc()
}

# writeRaster(crop(mod
# #
```

---

**Appendix A. Source Code**

```r
# s$"pm2.5.xgb" <- pred_xgb

# ensemble predictions
# s$"pm2.5.rf" <- ens_rf$fit
s$"pm2.5.lm" <- ens_lm$fit
#

# Return prediction SE for each model
s$pm2.5.rf.se <- pred_rfr$se
s$pm2.5.lm.se <- pred_lm$se

# Ensemble error method
s$pm2.5.rf.se <- ens_rf$se
s$pm2.5.lm.se <- ens_lm$se
# s$pm2.5.xgb.se <- ""

# # extract just the prediction layer
# pred_rf <- s$pm2.5.rf
# names(pred_rf) <- paste0(date, ".pm2.5.rf."),
# pred_lm <- s$pm2.5.lm
# names(pred_lm) <- paste0(date, ".pm2.5.lm."),
# pred_xgb <- s$pm2.5.xgb
# names(pred_xgb) <- paste0(date, ".pm2.5.xgb."),

pm_stack <- s[[which(grepl("pm2.5." , names(s)))]]
# % dropLayer("pm2.5.median")

names(pm_stack) <- paste0(date, ".", names(pm_stack))
m(s)
rm(df)
gc()

writeRaster(pm_stack, filename = paste0("./data/predicted_raster/", names(pm_stack), ".tif"),
            bylayer = T,
            format = "GTiff",
            overwrite = T)

rm(pm_stack)
gc()
}

# writeRaster(crop(mod
# #
```
# , filename = paste0("./data/predicted_raster/", names
(mod), ",", date, ".tif")
# , bylayer=T
# , format = "GTiff"
# , overwrite = T # will this throw an error?

}

# loop prediction function over each day
predictDailyRasters <- function(start_date = start_date
, end_date = end_date
# , crs = epsg_26910
# , scale_neph = scale_neph
# , rfr = rfr) {

dates <- sequenceDates(start_date = start_date, end_date = end_date)

# preds <- Reduce(rbind, pblapply(dates, function(x) {
# predictRasters(date = x
# # , crs = crs
# # , rfr = rfr
# # , cl = detectCores(-1))
#
# # Clear out existing files before writing new predictions
f <- list.files("./data/predicted_raster", ".tif", full.names = T)
lapply(f, file.remove)

pblapply(dates, function(x) {
    predictRasters(date = x
# , crs = crs
# , rfr = rfr
# , scale_neph = scale_neph
})
, cl = detectCores(-1)
# , cl = 6
)

# calling the function
predict_time <- system.time(predictDailyRasters(
    start_date = start_date
    # start_date = 
"2017-10-10"
    , end_date = end_date
)}
"2017-10-10" #, end_date =
#, crs = crs
, scale_neph = scale_neph
#, rfr = rfr
)

print("predictDailyRasters completed!")

# Checkpoint stack
s <- stack(list.files("./data/predicted_raster/", "rf.tif", full.names = T))
saveRDS(s, paste0("./data/predicted_stack/", clean_time, "_stack_pm2_5_rf.RDS"))

s <- stack(list.files("./data/predicted_raster/", "lm.tif", full.names = T))
saveRDS(s, paste0("./data/predicted_stack/", clean_time, "_stack_pm2_5_lm.RDS"))

# s <- stack(list.files("./data/predicted_raster/", "xgb.tif", full.names = T))
# saveRDS(s, paste0("./data/predicted_stack/", clean_time, "_stack_pm2_5_xgb.RDS"))

print("Saving rf, lm, xgb stacks!")

# r <- raster("./data/predicted_raster/X2017.10.10.pm2.5.predicted.tif")
# levelplot(r, margin = F)

saveRDS(predict_time, paste0("./models/predict_time/", clean_time, "_predict_time.RDS"))

index <- eval(parse(text = "1:16"))

png(paste0("./figures/levelplot/", clean_time, "_levelplot_{\text{min}}_index\_\text{max(index)}\_\text{res}_{\text{units}}.png"),
    width = 12
, height = 8
, res = 300
, units = "in"
)
levelplot(s[[index]])

dev.off()
Appendix A. Source Code

392  index <- eval(parse(text = "70:85"))
393  png(paste0("./figures/levelplot/", clean_time, "_levelplot_", min(index), "_.", max(index), ".png"))
394    ,width = 12
395    ,height = 8
396    ,res = 300
397    ,units = "in"
398  }
399  levelplot(s[[index]])
400  dev.off()
401
402  index <- eval(parse(text = "90:105"))
403  png(paste0("./figures/levelplot/", clean_time, "_levelplot_", min(index), "_.", max(index), ".png"))
404    ,width = 12
405    ,height = 8
406    ,res = 300
407    ,units = "in"
408  }
409  levelplot(s[[index]])
410  dev.off()
411
412  index <- eval(parse(text = "105:120"))
413  png(paste0("./figures/levelplot/", clean_time, "_levelplot_", min(index), "_.", max(index), ".png"))
414    ,width = 12
415    ,height = 8
416    ,res = 300
417    ,units = "in"
418  }
419  levelplot(s[[index]])
420  dev.off()
421
422  # This one usually looks really weird!
423  index <- eval(parse(text = "200:215"))
424  png(paste0("./figures/levelplot/", clean_time, "_levelplot_", min(index), "_.", max(index), ".png"))
425    ,width = 12
426    ,height = 8
427    ,res = 300
428    ,units = "in"
429  }
430  levelplot(s[[index]])
431  dev.off()
432
433  index <- eval(parse(text = "300:315"))
source_code

Appendix A. Source Code

```r
# Appendix A. Source Code

levelplot(s[[index]])
device.off()

index <- eval(parse(text = "370:395"))
levelplot(s[[index]])
device.off()

Source/vectorToRaster.R

vectorToRaster <- function (layer = NULL, # shapefile name w/ file extension
                          indir = ".//data/rlis/", # directory path
                          outdir = ".//data/rlis/", # output directory
                          buf_res = sp_res) # 50 m buffer
  {
    # Grab the layer name without file extension
    var <- basename(tools::file_path_sans_ext(layer))
    # Create outfile
    outfile <- paste0(outdir, var, ".tif")
    # Check to see if the outfile already exists
    if (!file.exists(outfile)) {
      print(paste0(outfile, " is being processed."))
      # read in shapefile and reproject
      vector <- readOGR(dsn = indir, layer = var) %>% spTransform(
        CRSobj = proj4string(df))
      # create an empty raster
```
```r
template_rst <- raster(extent(gBuffer(df, width = buf_res)), crs = projection(df), res = buf_res)

# determine the density of a feature in each cell
density <- pbsapply(1:ncell(template_rst), function(x) {

  # create placeholder raster
  tmp_rst <- template_rst

  # add placeholder value
  tmp_rst[x] <- 1

  # convert single cell extent to vector
  tmp_vec <- rasterToPolygons(tmp_rst)

  # Check to see if the feature is within the raster cell
  if (gIntersects(vector, tmp_vec)) {
    # Crop feature by the extent of a single raster cell
    vector_crp <- crop(vector, tmp_vec)

    # Perform different calculation for Points, Lines, and Polygons
    if ("SpatialLinesDataFrame" %in% class(vector)) {
      cell_value <- gLength(vector_crp)
    } else if ("SpatialPolygonsDataFrame" %in% class(vector)) {
      cell_value <- gArea(vector_crp)
    } else if ("SpatialPointsDataFrame" %in% class(vector)) {
      cell_value <- gIntersection(vector_crp, tmp_vec) %>% length()
    } else {
      print("Object not of class Spatial*DataFrame!")
      next
    }
  }

  return(cell_value)
}

cl = detectCores() - 1

# set the cell value to the kernel density
template_rst[] <- density

# write to file
writeRaster(template_rst)
```
Appendix A. Source Code

67     , filename = outfil
68     , format = "GTiff"
69     , overwrite = T)
70 } else {
71     print(paste0(outfil, " already exists!"))
72 }
73 }
74
75 # # list of shape files in rlis
76 # layers <- list.files("./data/rlis", pattern = "\.shp$")
77 #
78 # # call our processing function
79 # pblapply(layers, function(x) {
80 # vectorToRaster(x)
81 # })
82

Source/purpleair_id_key.py

1 # pull 3rd part purpleair data from json api
2 import json
3 import urllib
4 import requests
5 import time
6 from datetime import datetime
7 import calendar
8 import sys
9 import itertools
10 import os
11 import pandas as pd
12
13 file_name = "pa_id_key.txt"
14 dir_name = "./data/pa_id_key/"
15 full_path = os.getcwd() + "//" + dir_name + "//" + file_name
16 row = 0
17
18 d = datetime.utcnow()
19 unixtime = calendar.timegm(d.utctimetuple())
20
Appendix A. Source Code

315

,'Lon'
,'PM2_5Value'
,'LastSeen'
,'State'
,'Type'
,'Hidden'
,'Flag'
,'isOwner'
,'A_H'
,'temp_f'
,'humidity'
,'pressure'
,'AGE'
,'Stats'
]
)

print

## assigning PurpleAir API to url
url = "https://www.purpleair.com/json"

## GET request from PurpleAir API
try:
    r = requests.get(url)
    print '[*] Connecting to API...'
    print '[*] GET Status: ', r.status_code
except Exception as e:
    print '[*] Unable to connect to API...'
    print 'GET Status: ', r.status_code
print e

try:
    ## parse the JSON returned from the request
    j = r.json()
except Exception as e:
    print '[*] Unable to parse JSON'
    print e

try:
    ## iterate through entire dictionary
    for sensor in j['results']:
        df.loc[row] = pd.Series(dict(datetime = datetime.
            fromtimestamp(sensor['LastSeen']))
            ,ID = sensor['ID'])
78  ,ParentID = sensor['ParentID']
79  ,Label = sensor['Label']
80  ,DEVICE_LOCATIONTYPE = sensor['DEVICE_LOCATIONTYPE']
81  ,THINGSPEAK_PRIMARY_ID = sensor['THINGSPEAK_PRIMARY_ID']
82  ,THINGSPEAK_PRIMARY_ID_READ_KEY = sensor['THINGSPEAK_PRIMARY_ID_READ_KEY']
83  ,THINGSPEAK_SECONDARY_ID = sensor['THINGSPEAK_SECONDARY_ID']
84  ,THINGSPEAK_SECONDARY_ID_READ_KEY = sensor['THINGSPEAK_SECONDARY_ID_READ_KEY']
85  ,Lat = sensor['Lat']
86  ,Lon = sensor['Lon']
87  ,PM2_5Value = sensor['PM2_5Value']
88  ,LastSeen = sensor['LastSeen']
89  ,State = sensor['State']
90  ,Type = sensor['Type']
91  ,Hidden = sensor['Hidden']
92  ,Flag = sensor['Flag']
93  ,isOwner = sensor['isOwner']
94  ,A_H = sensor['A_H']
95  ,temp_f = sensor['temp_f']
96  ,humidity = sensor['humidity']
97  ,pressure = sensor['pressure']
98  ,AGE = sensor['AGE']
99  ,Stats = sensor['Stats']

100 )

101    print df.loc[[row]]
102    row += 1
103    df.to_csv(full_path, sep = "","", index = False, encoding = 'utf-8')
104  except Exception as e:
105      print '[*] Error, no data was written to file'
106      print e

Source/create_observation.sql

1  create table observation (  
2    created_at  varchar(20)  
3    id  varchar(10)  
4    sensor  char(1)  
5    label  varchar(100)  
6    temp_f  numeric  
7    humidity  numeric  
8    pm1_0_atm  numeric  
9    pm2_5_atm  numeric  
10    pm10_0_atm  numeric
Appendix A. Source Code

Source/create_pdx_cf.sql

```
cREATE TABLE pdx_cf ( 
  year varchar (4) ,
  month varchar (3) ,
  id integer ,
  label varchar (100) ,
  r_squared numeric ,
  adj_r_squared numeric ,
  sigma numeric ,
  statistic_r2 numeric ,
  p_value_r2 numeric ,
  df numeric ,
  logLik numeric ,
  AIC numeric ,
  BIC numeric ,
  deviance numeric ,
  df_residual numeric ,
  term_m varchar (100) ,
  estimate_m numeric ,
  std_error_m numeric ,
  statistic_m numeric ,
  p_value_m numeric ,
  term_b varchar (100) ,
  estimate_b numeric ,
  std_error_b numeric ,
  statistic_b numeric ,
  p_value_b numeric ,
  geom geometry ) ;
```

Source/thingspeakCollect.R

```
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018-08-31
```
# pull PurpleAir data from thingspeak API and upload to Postgres database

# thingspeak read function

# create function to collect purpleair data ~8000 rows at a time
thingspeak_collect <- function(row, start = "2017-01-01", end = as.character(Sys.Date())) {
  # for testing
  # start_date <- "2018-06-01"
  # end_date <- "2018-06-08"

  # 5-min resolution required for overlapping A & B sensors...
  time_resolution <- "5 min"
  time_step <- 5 # same as time res without string "min"

  # define align 5-minute time function

  # for clean 5-min breaks
  align.time.down = function(x, n) {
    index(x) = index(x) - n
    align.time(x, n)
  }

  # ignore rows without a ParentID
  if (is.na(row$ParentID)) {
    # next
    print("parent ID is empty")
  } else {
    # API ID & KEYS

    # grab parent row from pa_sf
    parent_row <- pa_sf %>% filter(ID == row$ParentID)

    # skip indoor air monitors!!!
    if (parent_row$DEVICE_LOCATIONTYPE == "indoor") {
      print(paste("Skipping indoor sensor", parent_row$Label))
      next
    }

    # primary api id and key pairs A sensor
    primary_id_a <- parent_row$THINGSPEAK_PRIMARY_ID
    primary_key_a <- parent_row$THINGSPEAK_PRIMARY_ID_READ_KEY
# secondary api id and key pairs A sensor
secondary_id_a <- parent_row$THINGSPEAK_SECONDARY_ID
secondary_key_a <- parent_row$THINGSPEAK_SECONDARY_ID_READ_KEY

# primary api id and key pairs B sensor
primary_id_b <- row$THINGSPEAK_PRIMARY_ID
primary_key_b <- row$THINGSPEAK_PRIMARY_ID_READ_KEY

# convert geometry to text
row$geometry <- st_as_text(row$geometry, EWKT = TRUE)

# row$geometry <- st_as_sfc(row$geometry) # converts back to geom

# create a weekly sequence

# need to break up our entire request into 8000 length chunks...
weeks <- seq(from = as.Date(start), to = as.Date(end), by = "week") %>% as.data.frame()

# assign vector name
colnames(weeks) <- "date"

# API requests across weekly sequence

# make weekly request to api (need to vectorize this...)
for (i in 1:nrow(weeks)) {

# extract start and end dates from our weekly sequence
start_date <- weeks$date[i]
end_date <- weeks$date[i + 1]

# if the end data is in the future, then use the current date as the final end point
if (is_empty(end_date) | is.na(end_date)) {
  end_date <- Sys.Date()
}

# A sensor primary url to pull from api
primary_url_a <- paste0("https://api.thingspeak.com/channels/",primary_id_a,"/feeds.json?api_key=")
Appendix A. Source Code

```
,primary_key_a
,"&start="
,start_date
,"%2000:00:00&end="
,end_date
,"%2023:59:59")

# A sensor secondary url to pull from api
secondary_url_a <- paste0("https://api.thingspeak.com/channels/

primary_id_a
,"/feeds.json?api_key="
,secondary_key_a
,"&start="
,start_date
,"%2000:00:00&end="
,end_date
,"%2023:59:59")

# B sensor primary url to pull from api
primary_url_b <- paste0("https://api.thingspeak.com/channels/

primary_id_b
,"/feeds.json?api_key="
,primary_key_b
,"&start="
,start_date
,"%2000:00:00&end="
,end_date
,"%2023:59:59")

# B sensor secondary url to pull from api
secondary_url_b <- paste0("https://api.thingspeak.com/channels/

secondary_id_b
,"/feeds.json?api_key="
,secondary_key_b
,"&start="
,start_date
,"%2000:00:00&end="
,end_date
,"%2023:59:59")

# try pulling from thingspeak API

# exception handling API request
```
primary_request_a <- NULL
secondary_request_a <- NULL
primary_request_b <- NULL
secondary_request_b <- NULL

primary_attempt_a <- 0
secondary_attempt_a <- 0
primary_attempt_b <- 0
secondary_attempt_b <- 0

# request A sensor data
while (is.null(primary_request_a) && primary_attempt_a <= 3) {
  try (primary_request_a <- jsonlite::fromJSON(primary_url_a))
  print(paste0("Primary Attempt A: ", primary_attempt_a))
  primary_attempt_a <- primary_attempt_a + 1
  Sys.sleep(2)
}
if (is.null(primary_request_a)) {
  print(paste0(primary_url_a, " failed!"))
}

while (is.null(secondary_request_a) && secondary_attempt_a <= 3) {
  try (secondary_request_a <- jsonlite::fromJSON(secondary_url_a))
  print(paste0("Secondary Attempt A: ", secondary_attempt_a))
  secondary_attempt_a <- secondary_attempt_a + 1
  Sys.sleep(2)
}
if (is.null(secondary_request_a)) {
  print(paste0(secondary_url_a, " failed!"))
}

# request B sensor data
while (is.null(primary_request_b) && primary_attempt_b <= 3) {
  try (primary_request_b <- jsonlite::fromJSON(primary_url_b))
  print(paste0("Primary Attempt B: ", primary_attempt_b))
  primary_attempt_b <- primary_attempt_b + 1
  Sys.sleep(2)
}
if (is.null(primary_request_b)) {
  print(paste0(primary_url_b, " failed!"))
}
while (is.null(secondary_request_b) && secondary_attempt_b <= 3) {
    try (secondary_request_b <- jsonlite::fromJSON(secondary_url_b))
    print(paste0("Secondary Attempt B: ", secondary_attempt_b))
    secondary_attempt_b <- secondary_attempt_b + 1
    Sys.sleep(2)
}

if (is.null(secondary_request_b)) {
    print(paste0(secondary_url_b, " failed!"))
}

# next if request is NULL

if (is_empty(primary_request_a$feeds) | is_empty(secondary_request_a$feeds) | is_empty(primary_request_b$feeds) | is_empty(secondary_request_b$feeds)) {
    print(paste0(start_date, "-", end_date, " ", parent_row$Label, " is empty..."))
    next
} else {
    print(paste0(start_date, "-", end_date, " ", parent_row$Label, " A & B are being processed..."))
}

# channel A field names

primary_fields_a <- c("created_at",
                     "entry_id",
                     "pm1_0_atm",
                     "pm2_5_atm",
                     "pm10_0_atm",
                     "uptime_min",
                     "rss_i_wifi_strength",
                     "temp_f",
                     "humidity",
                     "pm2_5_cf_1")

secondary_fields_a <- c("created_at",
                        "entry_id",
                        "p_0_3_um",
                        "p_0_5_um",
                        "p_1_0_um",
                        "p_2_5_um",
                        "p_5_0_um")
# channel B field names

```r
primary_fields_b <- c("created_at",
"entry_id",
"p_10_0_um",
"pm1_0_cf_1",
"pm10_0_cf_1")
```

```r
# channel B field names

```r
primary_fields_b <- c("created_at",
"entry_id",
"p_10_0_um",
"pm1_0_atm",
"pm10_0_atm",
"free_heap_memory",
"analog_input",
"sensor_firmware_pressure",
"not_used",
"pm2_5_cf_1")
```

```r
# channel B field names

```r
primary_fields_b <- c("created_at",
"entry_id",
"p_0_3_um",
"p_0_5_um",
"p_1_0_um",
"p_2_5_um",
"p_5_0_um",
"p_10_0_um",
"pm1_0_cf_1",
"pm10_0_cf_1")
```

# read in primary data for A sensor

```r
primary_df_a <- primary_request_a$feeds
```

```r
# assign A field names for primary data

# colnames(primary_df) <- primary_fields_a # threw error when missing fields ...
```

```r
# assign primary A field names dynamically

```r
names(primary_df_a)[names(primary_df_a) == "field1"] <-
primary_fields_a[3]
```

```r
names(primary_df_a)[names(primary_df_a) == "field2"] <-
primary_fields_a[4]
```

```r
names(primary_df_a)[names(primary_df_a) == "field3"] <-
primary_fields_a[5]
```

```r
names(primary_df_a)[names(primary_df_a) == "field4"] <-
primary_fields_a[6]
```
names( primary_df_a)[ names( primary_df_a) == 'field5' ] <- primary_fields_a[7]
names( primary_df_a)[ names( primary_df_a) == 'field6' ] <- primary_fields_a[8]
names( primary_df_a)[ names( primary_df_a) == 'field7' ] <- primary_fields_a[9]
names( primary_df_a)[ names( primary_df_a) == 'field8' ] <- primary_fields_a[10]

# remove non-numeric columns before grouping by date
primary_df_a <- primary_df_a %>% dplyr::select(-c(entry_id, uptime_min, rssi_wifi_strength))

# cast from character to numeric class
try(
  {
    primary_df_a$pm1_0_atm <- as.numeric(primary_df_a$pm1_0_atm)
    primary_df_a$pm2_5_atm <- as.numeric(primary_df_a$pm2_5_atm)
    primary_df_a$pm10_0_atm <- as.numeric(primary_df_a$pm10_0_atm)
    primary_df_a$temp_f <- as.numeric(primary_df_a$temp_f)
    primary_df_a$humidity <- as.numeric(primary_df_a$humidity)
    primary_df_a$pm2_5_cf_1 <- as.numeric(primary_df_a$pm2_5_cf_1)
  }, silent = TRUE)

# aggregate by clean 5-min breaks
primary_xts_a <- xts(primary_df_a[,-1], as.POSIXct(primary_df_a$created_at, format = "%Y-%m-%dT%H:%M:%SZ", tz = "GMT")

primary_means_a <- period.apply( primary_xts_a, endpoints( primary_xts_a, "mins", k = time_step), mean)
primary_rounded_a <- align.time.down(primary_means_a, time_step*60)
primary_df_a <- fortify(primary_rounded_a)
names( primary_df_a)[ names( primary_df_a) == "Index" ] <- "created_at"

primary_df_a$created_at <- as.character(primary_df_a$created_at)

# add sensor label
primary_df_a$sensor <- "A"
# read in secondary data for A sensor

```
secondary_df_a <- secondary_request_a$feeds

# assign A field names for secondary data
# colnames(secondary_df) <- secondary_fields_a # breaks when fields are missing

# assign secondary A field names dynamically
names(secondary_df_a)[names(secondary_df_a) == 'field1'] <- secondary_fields_a[3]
names(secondary_df_a)[names(secondary_df_a) == 'field2'] <- secondary_fields_a[4]
names(secondary_df_a)[names(secondary_df_a) == 'field3'] <- secondary_fields_a[5]
names(secondary_df_a)[names(secondary_df_a) == 'field4'] <- secondary_fields_a[6]
names(secondary_df_a)[names(secondary_df_a) == 'field5'] <- secondary_fields_a[7]
names(secondary_df_a)[names(secondary_df_a) == 'field6'] <- secondary_fields_a[8]
names(secondary_df_a)[names(secondary_df_a) == 'field7'] <- secondary_fields_a[9]
names(secondary_df_a)[names(secondary_df_a) == 'field8'] <- secondary_fields_a[10]
```

# remove non-numeric columns before grouping
secondary_df_a <- secondary_df_a %>% dplyr::select(-c(entry_id))

# cast from character to numeric class
try {
  secondary_df_a$p_0_3_um <- as.numeric(secondary_df_a$p_0_3_um)
  secondary_df_a$p_0_5_um <- as.numeric(secondary_df_a$p_0_5_um)
  secondary_df_a$p_1_0_um <- as.numeric(secondary_df_a$p_1_0_um)
  secondary_df_a$p_2_5_um <- as.numeric(secondary_df_a$p_2_5_um)
  secondary_df_a$p_5_0_um <- as.numeric(secondary_df_a$p_5_0_um)
  secondary_df_a$p_10_0_um <- as.numeric(secondary_df_a$p_10_0_um)
  secondary_df_a$pm1_0_cf_1 <- as.numeric(secondary_df_a$pm1_0_cf_1)
}
Appendix A. Source Code

```r
secondary_df_a$pm10_0_cf_1 <- as.numeric(secondary_df_a$pm10_0_cf_1)

# aggregate by clean 5-min breaks
secondary_xts_a <- xts(secondary_df_a[, -1], as.POSIXct(secondary_df_a$created_at,
  format = "%Y-%m-%dT%H:%M:%SZ",
  tz = "GMT"))

secondary_means_a <- period.apply(secondary_xts_a, endpoints(secondary_xts_a, "mins", k = time_step), mean)

secondary_rounded_a <- align.time.down(secondary_means_a, time_step * 60)

secondary_df_a <- fortify(secondary_rounded_a)

names(secondary_df_a)[names(secondary_df_a) == "Index"] <- "created_at"

secondary_df_a$created_at <- as.character(secondary_df_a$created_at)

# add sensor label
secondary_df_a$sensor <- "A"

# read in primary data for B sensor

primary_df_b <- primary_request_b$feeds

# assign primary B field names for primary data
# colnames(primary_df) <- primary_fields_b # breaks when fields are missing

# assign B field names dynamically
names(primary_df_b)[names(primary_df_b) == 'field1'] <- primary_fields_b[3]

names(primary_df_b)[names(primary_df_b) == 'field2'] <- primary_fields_b[4]

names(primary_df_b)[names(primary_df_b) == 'field3'] <- primary_fields_b[5]

names(primary_df_b)[names(primary_df_b) == 'field4'] <- primary_fields_b[6]

names(primary_df_b)[names(primary_df_b) == 'field5'] <- primary_fields_b[7]

names(primary_df_b)[names(primary_df_b) == 'field6'] <- primary_fields_b[8]
```
Appendix A. Source Code

```r
names(primary_df_b)[names(primary_df_b) == 'field7'] <-
primary_fields_b[9]
names(primary_df_b)[names(primary_df_b) == 'field8'] <-
primary_fields_b[10]

# remove non-numeric columns before grouping by date
primary_df_b <- primary_df_b %>% dplyr::select(-c(entry_id,
  free_heap_memory, analog_input, sensor_firmware_pressure, not_used ))

# cast from character to numeric
try({
  primary_df_b$pm1_0_atm <- as.numeric(primary_df_b$pm1_0_atm)
  primary_df_b$pm2_5_atm <- as.numeric(primary_df_b$pm2_5_atm)
  primary_df_b$pm10_0_atm <- as.numeric(primary_df_b$pm10_0_atm)
  primary_df_b$pm2_5_cf_1 <- as.numeric(primary_df_b$pm2_5_cf_1)
}, silent = TRUE)

# aggregate by clean 5-min breaks
primary_xts_b <- xts(primary_df_b[, -1],
  as.POSIXct(primary_df_b$created_at, format = "%Y-%m-%dT%H:%M:%SZ",
  tz = "GMT"))
primary_means_b <- period.apply(primary_xts_b, endpoints(
  primary_xts_b, "mins", k = time_step), mean)
primary_rounded_b <- align.time.down(primary_means_b, time_step*60)
primary_df_b <- fortify(primary_rounded_b)
names(primary_df_b)[names(primary_df_b) == "Index"] <- "created_at"
primary_df_b$created_at <- as.character(primary_df_b$created_at)

# add sensor label
primary_df_b$sensor <- "B"

# read in secondary data for B sensor
secondary_df_b <- secondary_request_b$feeds
```
# assign B field names for secondary sensor
# colnames(secondary_df) <- secondary_fields_b

# assign B field names dynamically
names(secondary_df_b)[names(secondary_df_b) == 'field1'] <- secondary_fields_b[3]
names(secondary_df_b)[names(secondary_df_b) == 'field2'] <- secondary_fields_b[4]
names(secondary_df_b)[names(secondary_df_b) == 'field3'] <- secondary_fields_b[5]
names(secondary_df_b)[names(secondary_df_b) == 'field4'] <- secondary_fields_b[6]
names(secondary_df_b)[names(secondary_df_b) == 'field5'] <- secondary_fields_b[7]
names(secondary_df_b)[names(secondary_df_b) == 'field6'] <- secondary_fields_b[8]
names(secondary_df_b)[names(secondary_df_b) == 'field7'] <- secondary_fields_b[9]
names(secondary_df_b)[names(secondary_df_b) == 'field8'] <- secondary_fields_b[10]

# remove non-numeric columns before grouping
secondary_df_b <- secondary_df_b %>% dplyr::select(-c(entry_id))

# cast character to numeric class
try({
secondary_df_b$p_0_3_um <- as.numeric(secondary_df_b$p_0_3_um)
secondary_df_b$p_0_5_um <- as.numeric(secondary_df_b$p_0_5_um)
secondary_df_b$p_1_0_um <- as.numeric(secondary_df_b$p_1_0_um)
secondary_df_b$p_2_5_um <- as.numeric(secondary_df_b$p_2_5_um)
secondary_df_b$p_5_0_um <- as.numeric(secondary_df_b$p_5_0_um)
secondary_df_b$p_10_0_um <- as.numeric(secondary_df_b$p_10_0_um)
secondary_df_b$pm1_0_cf_1 <- as.numeric(secondary_df_b$pm1_0_cf_1)
secondary_df_b$pm10_0_cf_1 <- as.numeric(secondary_df_b$pm10_0_cf_1)
}, silent = TRUE)
# aggregate by clean 5-min breaks
secondary_xts_b <- xts(secondary_df_b[-1],
  as.POSIXct(secondary_df_b$created_at,
    format = "%Y-%m-%dT%H:%M:%SZ",
    tz = "GMT"))

secondary_means_b <- period.apply(secondary_xts_b, endpoints(
  secondary_xts_b, "mins", k = time_step), mean)

secondary_rounded_b <- align.time.down(secondary_means_b,
  time_step*60)

names(secondary_df_b)[names(secondary_df_b) == "Index"] <- "created_at"

secondary_df_b$created_at <- as.character(secondary_df_b$created_at)

# add sensor label
secondary_df_b$sensor <- "B"

# add temp and humidity from primary A to primary B sensor

met_data <- primary_df_a %>%
  dplyr::select(created_at, temp_f, humidity)

print("joining met data")

primary_df_b <- primary_df_b %>%
  left_join(met_data) # read in primary data for B sensor

# attach PurpleAir API attributes to primary thingspeak data

primary_df_a$label <- parent_row$Label
primary_df_a$id <- parent_row$ID

secondary_df_a$label <- parent_row$Label
secondary_df_a$id <- parent_row$ID

primary_df_b$label <- row$Label
primary_df_b$id <- row$ID

secondary_df_b$label <- row$Label
secondary_df_b$id <- row$ID

# handle missing device location from B sensors!

# filter indoor air quality data

# grab location type only from parent ID
parent_location <- filter(pa_sf, ID == row$ParentID) %>%
dplyr::select(DEVICE_LOCATIONTYPE)

primary_df_a$DEVICE_LOCATIONTYPE <- parent_location$DEVICE_LOCATIONTYPE
secondary_df_a$DEVICE_LOCATIONTYPE <- parent_location$DEVICE_LOCATIONTYPE

primary_df_b$DEVICE_LOCATIONTYPE <- parent_location$DEVICE_LOCATIONTYPE
secondary_df_b$DEVICE_LOCATIONTYPE <- parent_location$DEVICE_LOCATIONTYPE

# filter out indoor purple air data
primary_df_a <- primary_df_a %>% filter(DEVICE_LOCATIONTYPE != "inside")
primary_df_a <- primary_df_a %>% dplyr::select(-DEVICE_LOCATIONTYPE) # threw error without dplyr::
primary_df_b <- primary_df_b %>% filter(DEVICE_LOCATIONTYPE != "inside")
primary_df_b <- primary_df_b %>% dplyr::select(-DEVICE_LOCATIONTYPE) # threw error without dplyr::

# print("test point 5")

secondary_df_a <- secondary_df_a %>% filter(DEVICE_LOCATIONTYPE != "inside")
secondary_df_a <- secondary_df_a %>% dplyr::select(-DEVICE_LOCATIONTYPE) # threw error without dplyr::
secondary_df_b <- secondary_df_b %>% filter(DEVICE_LOCATIONTYPE != "inside")
secondary_df_b <- secondary_df_b %>% dplyr::select(-DEVICE_LOCATIONTYPE) # threw error without dplyr::

# print("test point 6")

print("joining primary_a secondary_a")

# create wide dataframe to use less rows (tidy 100k rows per week per sensor) _________________________________
df_wide_a <- inner_join(primary_df_a, secondary_df_a, by = c("created_at","sensor","label","geom","id"))
Appendix A. Source Code

# reorder columns
df_wide_a <- df_wide_a %>% dplyr::select(
  created_at # put this first out of convention
  #,entry_id
  ,id
  ,label
  ,sensor
  #,uptime_min # Channel A
  #,rssi_wifi_strength # Channel A
  ,temp_f # Channel A
  ,humidity # Channel A
  ,pm1_0_atm
  ,pm2_5_atm
  ,pm10_0_atm
  ,pm1_0_cf_1
  ,pm2_5_cf_1
  ,pm10_0_cf_1
  ,p_0_3_um
  ,p_0_5_um
  ,p_1_0_um
  ,p_2_5_um
  ,p_5_0_um
  ,p_10_0_um
  #,geom # put this last out of convention
)

print("joining primary_b secondary_b")

# create wide dataframe to use less rows (tidy 100k rows per week per sensor)
df_wide_b <- inner_join(primary_df_b, secondary_df_b, by = c("created_at"
  ,"sensor"
  ,"label"
  #,"geom"
  ,"id"))

# reorder columns
df_wide_b <- df_wide_b %>% dplyr::select(
  created_at # put this first out of convention
  #,entry_id
  ,id
  ,label
  ,sensor
```
# , uptime_min # Channel A
# , rssi_wifi_strength # Channel A
, temp_f # Channel A
, humidity # Channel A
, pm1_0_atm
, pm2_5_atm
, pm10_0_atm
, pm1_0_cf_1
, pm2_5_cf_1
, pm10_0_cf_1
, p_0_3_um
, p_0_5_um
, p_1_0_um
, p_2_5_um
, p_5_0_um
, p_10_0_um
, geom # put this last out of convention
)

# Join Primary and Secondary data and omit NAs
df_wide <- bind_rows(df_wide_a, df_wide_b)
df_wide <- df_wide[complete.cases(df_wide),]
df_wide <- df_wide %>% na.omit()

# bring geom back

if (nrow(df_wide) == 0) {
  print("df_wide is empty, skipping db upload")
  next
}

print("adding geom")
df_wide$geom <- rows$geometry
print(class(df_wide))
print(class(df_wide$geom))
#print("converting from character to wkt")
#df_wide$geom <- st_as_sfc(df_wide$geom)
#print("converting to sf df")
df_wide <- st_as_sf(df_wide, wkt = "geom")
print(class(df_wide$geom))
print(class(df_wide))

observation <- df_wide %>% na.omit()
```
```r
# open connection to our db
con <- dbConnect(drv = RPostgres::Postgres(),
                 dbname = db,
                 host = 'pgsql102.rc.pdx.edu' # not sure why object host isn't working...
                 ,port = port,
                 password = pw,
                 user = user)

# TODO update database without producing dupes!

# writes only new independences to db
print("writing to db")
st_write(dsn = con,
         ,obj = observation
         ,query = "INSERT INTO pdx_local_slope ON CONFLICT DO NOTHING;" # this isn't working, writes twice...
         ,layer_options = "OVERWRITE=true"
         ,drop_table = FALSE
         ,try_drop = FALSE
         ,debug = TRUE
         ,append = TRUE
         ,dataset_options = "GEOMETRY=AS_WKT"
)

# cleaning up
invisible(gc())
```

Source/uploadDailyThingspeak.R

# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018–07–16
# pull PurpleAir data from thingspeak API and upload to Postgres database

# set up environment

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
```
library (pacman)

p_load (readr,
         readxl
         #, xlsx
         ggplot2
         ,plyr
         ,dplyr
         ,broom
         ,reshape2
         ,tidyr
         ,stringr
         ,magrittr
         ,rlist
         ,grid
         ,gridExtra
         ,tidyquant
         ,scales
         ,qdapRegex
         ,sp
         ,rgeos
         ,sf
         ,devtools
         ,RColorBrewer
         ,classInt
         ,htmltools
         ,scales
         ,htmlwidgets
         ,httr
         ,jsonlite
         ,rgdal
         ,pbapply
         ,snow
         ,parallel
         ,data.table
         ,RPostgres
         ,leaflet
)

# define global variables

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.setenv(TZ="GMT")

## connecting to local db
host <- 'pgsql120.rc.pdx.edu'
db <- 'canopycontinuum'
user <- 'porlando'
port <- 5433
pw <- scan("./batteries.pgss", what = "")

# initially connect to clear existing data
con <- dbConnect(drv = RPostgres::Postgres(),
                 dbname = db,
                 host = 'pgsql102.rc.pdx.edu' # not sure why object host isn’t working...
                 port = port
                 password = pw
                 user = user)

# create unanimous time resolution for all data
time_resolution <- "1 sec"

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs "

# Albuquerque, NM, UTM 13N
utm_13N <- "+proj=utm +zone=13 +ellps=WGS84 +datum=WGS84 +units=m +no_defs "

# Sacramento, CA, UTM 10S, meters
epsg_26911 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs "

# Oregon North NAD83 HARN meters
epsg_2838 <- "+proj=lcc +lat_1=46 +lat_2=44.33333333333334 +lat_0=43.66666666666666 +lon_0=-120.5 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs "

# Oregon North NAD83 Meters UTM Zone 10
epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs "

# Function definition

# creating a custom not-in function
'%! in%' <- function(x,y) ('%!in%'(x,y))

# set path to output of python script (up to date purpleair list)
pa_path <- "./data/pa_id_key/pa_id_key.txt"

# sourcing thingspeak collect function
source("./src/thingspeakCollect.R")

# upload new data each day to db

# convert to char because of thingspeak_collect legacy code...
yesterday <- as.character(Sys.Date() - 1)
today <- as.character(Sys.Date())
tomorrow <- as.character(Sys.Date() + 1)

# grab up to date list of global PurpleAir network
system("python ./purpleair_id_key.py") # this unexpectedly crashes when running python within RStudio

# reading our scraped purpleair id data
pa <- read.csv(pa_path
    ,stringsAsFactors = FALSE
    ,header = TRUE)

# converting to simple features class
pa_sf <- st_as_sf(pa
    ,coords = c("Lon", "Lat")
    ,crs = wgs_84
    ,na.fail = FALSE)

# reading in urban growth boundary shapefile
portl <- readOGR(dsn = "/data/shape/ugb.shp") # n =
portl <- spTransform(portl, CRSobj = CRS(epsg_26910))
portl <- st_as_sf(portl)
pa_sf <- st_transform(pa_sf, crs = st_crs(portl))

# Buffer Portland Study Area to capture Urban–Rural Gradient!
portl_buf <- st_buffer(portl, dist = 1e5)

# subset purpleair that intersect  ubg
pa_sf <- pa_sf[portl_buf,]

# same thing as above but with a progress bar
pbapply(pa_sf, 1, thingspeak_collect, start = yesterday, end =
yesterday)
# created by Philip Orlando @ Sustainable Atmospheres Research Lab
# Canopy Continuum Project, USFS
# PI Dr. Linda George
# 2018−07−16
# pull PurpleAir data from thingspeak API and upload to Postgres database

# set up environment

# load the necessary packages
if (!require(pacman)) {
  install.packages("pacman")
  library(pacman)
}

p_load(readr,
       readxl,
       ggplot2,
       plyr,
       dplyr,
       broom,
       reshape2,
       tidyr,
       stringr,
       magrittr,
       rlist,
       grid,
       gridExtra,
       tidyquant,
       scales,
       qdapRegex,
       sp,
       sf,
       devtools,
       RColorBrewer,
       classInt,
       htmltools,
       scales,
       htmlwidgets,
       httr,
       jsonlite,
       rgdal,
       pbapply,
       snow,
       parallel,
       data.table,
       RPostgres
```r

# define global variables

# set timezone to UTC/GMT to match postgres
# otherwise, st_read() will convert timestamp to US/Pacific with PST/PDT!
# DEQ only uses PST!
Sys.setenv(TZ="GMT")

## connecting to local db
host <- 'pgsql120.rc.pdx.edu'
db <- 'canopycontinuum'
user <- 'porlando'
port <- 5433
pw <- scan("./batteries.pgpss", what = "")

# initially connect to clear existing data
con <- dbConnect(drv = RPostgres::Postgres() 
                , dbname = db
                , host = 'pgsql120.rc.pdx.edu' # not sure why object host isn't working...
                , port = port
                , password = pw
                , user = user)

# create unanimous time resolution for all data
time_resolution <- "1 sec"

# CRS
wgs_84 <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs "

# Albuquerque, NM, UTM 13N
utm_13N <- "+proj=utm +zone=13 +ellps=WGS84 +datum=WGS84 +units=m +no_defs "

# Sacramento, CA, UTM 10S, meters
epsg_26911 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Oregon North NAD83 HARN meters
epsg_2838 <- "+proj=lcc +lat_1=46 +lat_2=44.33333333333334 +lat_0=43.66666666666666 +lon_0=-120.5 +x_0=2500000 +y_0=0 +ellps=GRS80 +units=m +no_defs "
```

# Oregon North NAD83 Meters UTM Zone 10

epsg_26910 <- "+proj=utm +zone=10 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"

# Function definition

function(x, y) {in(x, y)}

# set path to output of python script (up to date purpleair list)
pa_path <- "./data/pa_id_key/pa_id_key.txt"

# sourcing thingspeak collect function
source("./src/thingspeakCollect.R")

# loop begin
# upload new data each day to db

# grab up to date list of global PurpleAir network
system("python ./purpleair_id_key.py") # this unexpectedly crashes when running python within RStudio

# reading our scraped purpleair id data
pa <- read.csv(pa_path
  ,stringsAsFactors = FALSE
  ,header = TRUE)

# converting to simple features class
pa_sf <- st_as_sf(pa
  ,coords = c("Lon", "Lat")
  ,crs = wgs_84
  ,na.fail = FALSE)

# reading in urban growth boundary shapefile
portl <- readOGR(dsn = "/data/shape/ugb.shp") # n =
portl <- spTransform(portl, CRSobj = CRS(epsg_26910))
portl <- st_as_sf(portl)
pa_sf <- st_transform(pa_sf, crs = st_crs(portl))

# Buffer Portland Study Area to capture Urban-Rural Gradient!
portl_buf <- st_buffer(portl, dist = 1e5)

# subset purpleair that intersect ugb
pa_sf <- pa_sf[portl_buf,]
Appendix A. Source Code

# invisible(apply(pa_sf
#   ,MARGIN = 1 # applies over rows
#   ,FUN = thingspeak_collect
# ))

# same thing as above but with a progress bar
pbapply(pa_sf, 1, thingspeak_collect, start = "2017-07-01", end = as.character(Sys.Date())
   ,cl = detectCores()-1 # linux-specific (doesn’t work on windows!)
)

Source/purpleair_watchdog.py

### Created by: Philip Orlando
### Sustainable Atmospheres Research Lab
### Portland State University
### 2018-02-15

import json
import urllib
import requests
import time
from datetime import datetime
import calendar
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import getpass
import pyfiglet
import termcolor
import itertools
import sys

## program header
termcolor.cprint(pyfiglet.figlet_format('PurpleAir\nWatchdog', font='slant'),
Appendix A. Source Code

28 'magenta', attrs=['bold'])
29
## create send email Class
30 class Gmail(object):
31     def __init__(self, email, recipient, password):
32         self.email = email
33         self.password = password
34         self.recipient = recipient
35         self.server = 'smtp.gmail.com'
36         self.port = 587
37         session = smtplib.SMTP(self.server, self.port)
38         session.ehlo()
39         session.starttls()
40         session.ehlo
41         session.login(self.email, self.password)
42         self.session = session
43
44     def send_message(self, subject, body):
45         headers = [
46             "From: " + self.email,
47             "Subject: " + subject,
48             "To: " + self.recipient,
49             "MIME-Version: 1.0",
50             "Content-Type: text/html"
51         ]
52         headers = "\n\n".join(headers)
53         self.session.sendmail(
54             self.email,
55             self.recipient,
56             headers + "\n\n\n" + body)
57
58
## define email parameters:
59     sender = 'phlp.orlando@gmail.com'
60
## recipient = 'h6lg@pdx.edu'
61     recipient = 'psustarlab@pdx.edu'
62
## secured raw_input for email password
63     email_password = getpass.getpass('[*] Enter the email server password : ')
64
## creating a list of sensor IDs
65     sensorID = [
66         # PDX Sites
67         3786  # PSU Star Lab Cully
Appendix A. Source Code

```
,3787 # PSU Star Lab Cully B
#,3357 # Irvington
#,3358 # Irvington B
#,5826 # NE 12th & Tillamook
#,5827 # NE 12th & Tillamook B
#,7018 # Miller
#,7019 # Miller B
#,2317 # Portsmouth Portland
#,2318 # Portsmouth Portland B
,2037 # PSU STAR Lab Hayden Island
,2038 # PSU STAR Lab Hayden Island B
,1606 # PSU STAR Lab Roof North
,1607 # PSU STAR Lab Roof North B
,1569 # PSU STAR Lab Roof South
,1570 # PSU STAR Lab Roof South B
#,7386 # PSU STAR Lab Roof South SD
#,7387 # PSU STAR Lab Roof South SD B
,2045 # PSU STAR Lab Rose City Park
,2046 # PSU STAR Lab Rose City Park B
,2065 # STAR Lab Hillsdale
,2066 # STAR Lab Hillsdale
,2059 # STAR Lab Aloha
,2060 # STAR Lab Aloha B
,2055 # STAR LAB BETHANY
,2056 # STAR LAB BETHANY B
,2043 # STAR Lab Creston Kenilworth
,2044 # STAR Lab Creston Kenilworth B
,2057 # STAR Lab Homestead Neighborhood
,2058 # STAR Lab Homestead Neighborhood B
,2053 # STAR Lab Powell–Hurst Gilbert
,2054 # Star Lab Powell–Hurst Gilbert B
,3707 # STAR Jesuit HS
,3708 # STAR Jesuit HS B
,3729 # PSU STAR Lab Edgewater
,3730 # PSU STAR Lab Edgewater B
,3684 # Lower Boones Ferry
,3685 # Lower Boones Ferry B
,3775 # PSU STAR Lab Lost Park
,3776 # PSU STAR Lab Lost Park B
#,2033 # Star Lab Jesuit
#,2034 # Star Lab Jesuit B
#,2566 # VerdeVista
#,2567 # VerdeVista B
#,3404 # Woods Park
#,3405 # Woods Park B
#,3281 # Red Fox Hills
#,3282 # Red Fox Hills B
```
Appendix A. Source Code

# ,3233 ## Marion Court Apartments
# ,3234 ## Marion Court Apartments B
# ,2741 ## College Park
# ,2742 ## College Park B

# ABQ Sites
,7366 # PSU STAR Lab Calle Suenos
,7367 # PSU STAR Lab Calle Suenos B
,9758 # PSU STAR Lab Homestead Trail ABQ
,9759 # PSU STAR Lab Homestead Trail ABQ B
,10304 # PSU STAR Lab Carrington
,10305 # PSU STAR Lab Carrington B
,9746 # PSU STAR Lab ABQTRAM
,9747 # PSU STAR Lab ABQTRAM B
,7370 # PSU STAR Lab Parkway
,7371 # PSU STAR Lab Parkway B
,7404 # PSU STAR Lab Old Town
,7405 # PSU STAR Lab Old Town B
,7420 # PSU STAR Lab ABQ Mark Twain
,7421 # PSU STAR Lab ABQ Mark Twain B
,10278 # PSU STAR Lab Bandelier
,10279 # PSU STAR Lab Bandelier B
,7406 # PSU STAR Lab Parkland Hills Albuquerque
,7407 # PSU STAR Lab Parkland Hills Albuquerque B
,9730 # Elyse Pl. Albuquerque
,9731 # Elyse Pl. Albuquerque B
,9782 # PSU STAR Lab Sunstar Loop
,9783 # PSU STAR Lab Sunstar Loop B
,9754 # PSU STAR Lab – Loma del Rey
,9755 # PSU STAR Lab – Loma del Rey B

## establish downtime intervals

downHour = int(60**2)
downDay = int(60*60*24)
#downDay = int(60)
#sleep_time = int(30)
sleep_time = int(60*15) ## 15-minute scan

## wrap our algo into a while loop:
while True:

## retrieving the current timestamp and converting it to unixtime
t = datetime.utcnow()
startTime = calendar.timegm(t.utctimetuple())
nextDay = startTime + downDay


try:
  del(offline_sensors[:])

except Exception:
  pass

offline_sensors = []

while calendar.timegm(datetime.datetime.utcnow().utctimetuple()) < nextDay:
  d = datetime.datetime.utcnow()
  unixtime = calendar.timegm(d.utctimetuple())

  print

  ## assigning PurpleAir API to url
  url = "https://www.purpleair.com/json"

  ## GET request from PurpleAir API
  try:
    r = requests.get(url)
    print '[*] Connecting to API...'  
    print '[*] GET Status: ', r.status_code
  except Exception as e:
    print '[*] Unable to connect to API...'  
    print 'GET Status: ', r.status_code
    print e
    print

  try:
    ## parse the JSON returned from the request
    j = r.json()
  except Exception as e:
    print '[*] Unable to parse JSON'
    print e
    print
  try:
    ## iterate through entire dictionary
    for sensor in j['results']:

      ## retrieve only the sensors from our list
      if sensor['ID'] in sensorID:

        ## determine if a sensor has been down for a day
```python
if (unixtime - int(sensor['LastSeen'])) > downHour and sensor['ID'] not in [int(y) for x in offline_sensors for y in x.split()]:
    print '[*] Sensor', sensor['Label'], 'went offline at',
datetime.fromtimestamp(sensor['LastSeen'])
down_sensor = str(sensor['Label'])
down_ID = str(sensor['ID'])
offline_sensors.append(down_ID)  ## this could be better handled
## send email
    msg = 'has been down since ' + str(datetime.fromtimestamp(
        sensor['LastSeen']))
gm = Gmail(sender, recipient, email_password)
gm.send_message(down_sensor, msg)
    print '[*] Notification sent.'

elif (unixtime - int(sensor['LastSeen'])) > downHour and sensor['ID'] in [int(y) for x in offline_sensors for y in x.split()]:
    print '[*] Sensor', sensor['Label'], 'has been down since',
datetime.fromtimestamp(sensor['LastSeen'])
    print '[*] Notification has already been sent to', recipient

else:
    print '[*]', sensor['Label'] + ':', sensor['ID'], '
lastSeen', datetime.fromtimestamp(sensor['LastSeen'])

    try:
        offline_sensors.remove(sensor['ID'])  # clear a sensor once it
        goes back online!

    except:
        pass
    print
    print '[*] nextScan', datetime.fromtimestamp(unixtime + sleep_time)
    print '[*] startTime', datetime.fromtimestamp(startTime)
    print '[*] unixtime', datetime.fromtimestamp(unixtime)
    print '[*] nextDay', datetime.fromtimestamp(nextDay)
    time.sleep(sleep_time)

except Exception as e:
    print '[*] Delivery to', recipient, 'failed!' print e
    print '[*] Will try again in one hour....'
    time.sleep(sleep_time)

# empty our list of offline sensors each day
#offline_sensors[:] = []
```
Appendix B

ThingSpeak PurpleAir Field Descriptions

Channel A and B, primary and secondary ThingSpeak channels together provide 32 fields for each sensor. There are six $\mu g m^{-3}$ values and six particle counts for each channel (laser) as well as temperature, humidity, WiFi signal (RSSI), sensor uptime, free memory and analog input.

B.1 Channel A

B.1.1 PrimaryData

field1: PM1.0 (CF=ATM) ug/m3
field2: PM2.5 (CF=ATM) ug/m3
field3: PM10.0 (CF=ATM) ug/m3
field4: Uptime (Minutes)
field5: RSSI (WiFi Signal Strength)
field6: Temperature (F)
field7: Humidity (%)
field8: PM2.5 (CF=1) ug/m3 This is the field to use for PM2.5
B.1.2 SecondaryData

field1: 0.3um particles/deciliter
field2: 0.5um particles/deciliter
field3: 1.0um particles/deciliter
field4: 2.5um particles/deciliter
field5: 5.0um particles/deciliter
field6: 10.0um particles/deciliter
field7: PM1.0 (CF=1) ug/m3 This is the field to use for PM1.0
field8: PM10 (CF=1) ug/m3 This is the field to use for PM10

B.2 Channel B

B.2.1 PrimaryData

field1: PM1.0 (CF=ATM) ug/m3
field2: PM2.5 (CF=ATM) ug/m3
field3: PM10.0 (CF=ATM) ug/m3
field4: Free HEAP memory
field5: ADC0 (analog input) voltage
field6: SENSOR Firmware 2.5 and up: Atmospheric Pressure
field7: (NOT USED)
field8: PM2.5 (CF=1) ug/m3 This is the field to use for PM2.5
B.2.2 SecondaryData

field1: 0.3um particles/deciliter
field2: 0.5um particles/deciliter
field3: 1.0um particles/deciliter
field4: 2.5um particles/deciliter
field5: 5.0um particles/deciliter
field6: 10.0um particles/deciliter
field7: PM1.0 (CF=1) ug/m3 This is the field to use for PM1.0
field8: PM10 (CF=1) ug/m3 This is the field to use for PM10