Fall 1-27-2017

Building a Multivariable Linear Regression Model of On-road Traffic for Creation of High Resolution Emission Inventories

James Eckhardt Powell
Portland State University

Building a Multivariable Linear Regression Model of On-road Traffic for Creation of High Resolution Emission Inventories

by

James Eckhardt Powell

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

Master Of Science
in
Physics

Thesis Committee:
Christopher Butenhoff, Chair
Linda George
M. Aslam K. Khalil
Andrew Rice

Portland State University
2016

Abstract

Emissions inventories are an important tool, often built by governments, and used to manage emissions. To build an inventory of urban CO_2 emissions and other fossil fuel combustion products in the urban atmosphere, an inventory of on-road traffic is required. In particular, a high resolution inventory is necessary to capture the local characteristics of transport emissions. These emissions vary widely due to the local nature of the fleet, fuel, and roads.

Here we show a new model of Average Daily Traffic (ADT) for the Portland, OR metropolitan region. The backbone is traffic counter recordings made by the Portland Bureau of Transportation at 7,767 sites over 21 years (1986-2006), augmented with PORTAL (The Portland Regional Transportation Archive Listing) freeway traffic count data. We constructed a regression model to fill in traffic network gaps using GIS data such as road class and population density. An EPA-supplied emissions factor was used to estimate transportation CO_2 emissions, which is compared to several other estimates for the city's CO_2 footprint.

DEDICATION

To my family: Carol L. Linné and Paul Y. Irvin, John G. Powell and Debby E. Powell, Ryan A. Powell and Kristin E. Bowen, Emmett J. Powell and Miles V. Powell. You are my inspiration.

ACKNOWLEDGMENTS

For our wonderful software libre stack, we thank the developers of GNU emacs, GNU/Linux, gcc, Debian and CentOS, Sun Grid Engine (SGE), GRASS GIS, PostgreSQL, R [1], Lattice [2], and cs2cs. A big thank you to Chad Sarni of PSU Physics for downloading the PORTAL data for us, and to William Garrick of PSU Research Computing for translating the TDAT into SQL. Kristin Tufte has been patient answering many queries we've had about PORTAL. Jamie Throckmorton and Tom Jensen of PBOT have unlocked the Bureau of Transportation's methods for us. Dr. Andrew Rice has helped us on many occasions by providing interesting avenues for new research and by his keen insight into the research problem. Dr. Aslam Khalil has been a steady support through some rocky periods. Dr. Khalil has maintained a big picture view of the field for me throughout, and he has my gratitude for introducing me to my advisor, Dr. Chris Butenhoff. I thank Dr. Butenhoff for his almost superhuman patience with my work, his insanely great science skills, and his ability to reign in my sometimes wild speculation thereby directing this research to the conclusion you will read about here.

Table Of Contents

List of Tables

List of Figures

Chapter 1 Introduction

§1.1 Background

> "The evidence is incontrovertible: global warming is occurring"(American Physical Society, Policy Statement 7.1, 2007-11-18)
>
> "Warming of the climate system is unequivocal" [3]

Global warming has already given us a warmer ocean/atmosphere system, less ice, and a sea level that has and will continue to rise [3, p. 4]. CO_2 in the atmosphere has been rising due to human activities, and the rise since 1750 has created a situation where anthropogenic CO_2 has enhanced the radiative forcing (RF) of the greenhouse [1] effect by a factor of 1.68 [1.33 to 2.03 W m^{-2}] compared to 1750 levels. CO_2 is thus the "elephant in the room," being the anthropogenic gas most responsible for global warming [3, p. 13].

Consequently, great energy has been and will continue to be focused on cutting, drastically, humanity's CO_2 footprint. Only two avenues are available to us: mitigation and sequestration. Sequestration, while a favorite of governments, is so far limited by the lack of incentive for power companies to pay for it, and difficulties with storage of the sequestered CO_2 [4]. We believe that mitigation is a key part of the anthropogenic CO_2 emissions problem and that emissions inventory (EI)s are a primary tool in our effort to find an effective means of mitigation.

The global change research community has been constructing high resolution

[1] A misnomer: greenhouses are primarily heated by convection, not radiation.

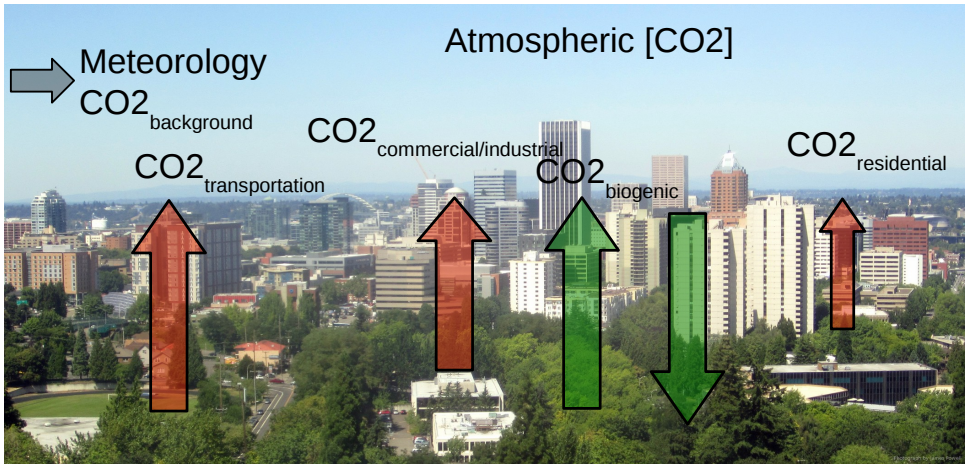


Figure 1.1: Factors determining urban CO_2 concentrations, after [8, Figure 2]. Photograph by the author.

(< 10 km x 10 km resolution) EI of CO_2 for several years [5]. The call for this type of EI is to improve policy making for climate change mitigation and to help scientists to close the carbon budget at small scales. Our field was recently described as "the nascent field of modeling and observing urban carbon pools and flows" [6]. Observers see the possibility that "an urban research campaign ... [that] combines surface observations [with] ... high-resolution flux estimation ... could transform carbon cycle science" (ibid). Meta-analyses have abundantly identified "the importance of using localized data for accuracy in terms of emissions estimates" [7], however such data and the expertise needed to analyze them are prohibitively expensive in many cases.

The fluxes of CO_2 in a city are complex, including biogenic, residential, commercial/industrial, and CO_2 from transportation and that blown in by winds in a regional mixing. Figure 1.1 shows these sources and sinks.

> The biggest significant change, one that really does alter what it is like to be human, may well be modern communication and *transportation*" [9, p. 249, emphasis mine]

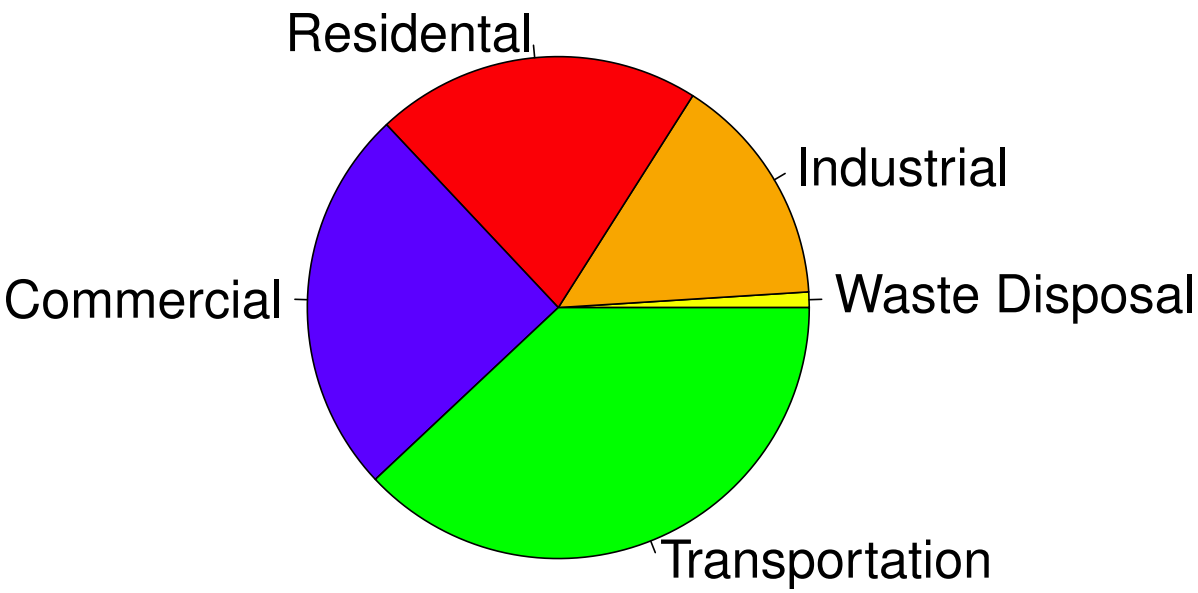


Figure 1.2: Portland, OR GHG emissions by sector [10].

Figure 1.2 demonstrates the importance of the transportation sector in CO_2 emissions for Portland. We believe, for reasons discussed throughout, that urban on-road transport is where the major focus for CO_2 emissions mitigation should be.

In 2014 [6] conducted a broad survey identifying the way forward from the current, weak knowledge regarding the pathway to a sustainable, low-carbon future for our cities. The situation is complex in the extreme; carbon use trajectories vary widely between developed and developing countries, stakeholders are many and powerful, and equity questions are strongly in play.

It will be possible to plot a sustainable course by improving our understanding through "integrated, coproduced approaches to urbanization" (ibid.), combining expertise of engineers, social scientists, and natural scientists. A repeated theme in this work is the importance of "researchers and practitioners who create a thorough quantification of city-level carbon emissions" (ibid).

Here we show a novel technique of making a high resolution EI by combining

two extensive and local data sources for transportation with several government-cultivated yet still local data sources for land-use (where the term land-use is taken loosely to include factors like population density and road density). We report here the methods and results for developing a high-resolution, high-accuracy CO_2 emissions inventory for the North American city of Portland, OR. This is the first work to our knowledge that examines raw traffic count data, a primary source; preceding efforts have relied on state-reported numbers for VMT, which were shown in [11] to be quite inaccurate for some states.

Hundreds of governments of all scales, world-wide are working to reduce CO_2 emissions, including for example the state of California [12], the city of Portland, OR [10], and a coalition of hundreds of cities in Local Governments for Sustainability (ICLEI) [6], and many nations under the Kyoto Protocol which was put forward in 1997 by the United Nations Framework Convention on Climate Change (UNFCCC). The Paris talk of 2016 resulted in a commitment by the US to reduce CO_2 emissions to "26-28 percent below 2005 levels by 2025" [13].

Beyond voluntary efforts such as these, binding agreements between states are emerging, and these are likely to have far-reaching economic impact. Such agreements will require a process known as monitoring reporting and verification (MRV) [14, p. 8423]. Emissions inventories are expected to be a primary means of demonstrating compliance, but the accuracy of EI generally are as large or greater than the expected emissions reductions.

Evidence is building that deep cuts in urban emissions will be required to mitigate the worst (and potentially catastrophic) effects of global warming [15, p. 486] [10] For example, in 2008 Wheeler wrote that "a small but growing number of U.S. states and cities have adopted long-term goals that would reduce CO_2 equivalents nearly 80% below the 2000 level by 2050." [15].

Policy makers are interested in CO_2 emissions mitigation as a source of options

(primarily cap and trade) in the face of potentially catastrophic climate change. Emissions are of interest because sequestration options appear highly limited [16]. An accurate EI is a required vehicle for proving the effectiveness of emissions mitigation policy and also a useful tool for exploring mitigation options.

Carbon cycle research includes two distinct important elements: a terrestrial sink that is hard to quantify (the biosphere) and a fossil fuel source that is well known. In inversion studies, the latter is often used to constrain the former, and even slight inaccuracies in spatial or temporal inventories of CO_2 emissions from fossil fuels result in uncertainty in estimations of Net Ecosystem Exchange (NEE) [17] [18].

The global change research program's attention has rightly fallen to the urban scale [6], because recently the balance of humanity became urbanized and the majority of global CO_2 emissions are urban. Past studies have produced EI on a global or national scale, however such work is not useful for understanding urban-scale sources and sinks because an EI commensurate with the urban scale is needed.

In the United States (US), where there is no national policy [2], the scale at which policy is being made is often urban.

Cities have been shown to be the source of most of the growth in on-road CO_2 emissions in recent decades [11], and cities have another reason to mitigate: cost savings. One study concluded that city managers are motivated by cost savings more than public opinion, and that cities can save money in some cases through mitigation [20].

Cities are also important for their activism (ICLEI) and because there is a governance advantage at work. We generally find that policymakers at the urban level have the ready ability to translate decisions into action especially in the

[2] In 2009 the EPA established a policy of maintaining an emissions inventory of Greenhouse Gas (GHG) from major emitters in the US [19] , indicating some further regulation may be forthcoming.

transport realm (Leonor Tarrasón, in a presentation at GEIA 2015).

As humanity works to curb CO_2 emissions, Integrated Assessment Model (IAM)s are a useful tool for categorizing the emitters. However, global IAMs ignore urban-level work, where much interesting effort is going into urban planning and urban mode shift work which will impact mitigation. Our high-resolution urban model will capture this effect and enable these local efforts to be a part of the conversation about CO_2 mitigation [21].

Globally, the transport sector was found to be the source of 23 % of CO_2 emissions in the year 2010 [22, p. 4]. On-road transport is 81% of that quantity [23]. We find that on-road transport (that is, not air, rail, or water) is the largest category of Greenhouse Gas (GHG) emissions nationwide in the USA [19], and 38% of GHG emissions in Portland, OR [10]. The transport sector overall was responsible for 33.4 % of GHG \ (1,704.6 MMT CO_2 eq.) emitted in 2012 in the US, the last year reported.

This work differs from the usual approach in that we draw from primary sources: the Portland Bureau of Transportation (PBOT) Traffic DATa (TDAT) archive and the Portland Regional Transportation Archive Listing (PORTAL) archive. there is another study which used road-segment level Vehicle Miles Travelled (VMT) to avoid any requirement of down scaling by proxy [11, p. 5000], our product offers a direct, local and bottom-up result.

We also see an opportunity to test some problems related to urban planning. Portland is an unusually planned city, unique in some ways: the only city in the United States with an elected metropolitan authority whose reach extends well beyond the city limits, and in 2011 the city rated 5th in the nation (after Denver, New York, Los Angeles, and Boston) for "the best combination of public transportation investment, ridership, and safety" [3].

[3] http://www.usnews.com/news/articles/2011/02/08/10-best-cities-for-public-transportation

So we see that matters of scientific interest (close the carbon cycle), civic interest (build a sustainable city), and international law (treat with others to avoid a tragedy of the commons) all demand much higher accuracy CO_2 emissions inventories, and this is the novel contribution of our work.

Our effort has produced an emissions inventory of high resolution and accuracy for the City of Portland, Oregon, in the United States of America, for CO_2 emissions from on-road transport. Our model, circoscope, is bottom-up and based on traffic counters which are believed to be a very reliable source of information, measured at 98% accurate [24].

§1.1.1 The Big Picture

We initiated the project in 2011 in the full understanding that a unique resource, the TDAT archive, existed and could, no doubt, be used to constrain the city's emissions inventory for CO_2. The conceptual model in Figure 1.3 was developed early and still guides our work.

We were also aware that the PORTAL group had been collecting counts on all of the region's freeways (restricted roads) for nine years [25]. Since TDAT has no data for restricted roads, this PORTAL archive is vital complement allowing us to complete our picture of the region's ADT.

We designed this research then to produce a high resolution EI for the City of Portland's CO_2 from on-road transport. Few other efforts have produced such high resolution EI for CO_2, two examples being the Hestia inventory for Indianapolis [26] and Kevin Gately's inventory for Boston [18], and McDonald et. al.'s 2014 fuel and traffic-count based inventories for a number of US cities [27].

The discrete counts of TDAT and PORTAL were not enough for this purpose. We needed to be able to model the whole road network with a meter by meter traffic volume indicator such as ADT or VMT. Our hypothesis was that a multiple linear

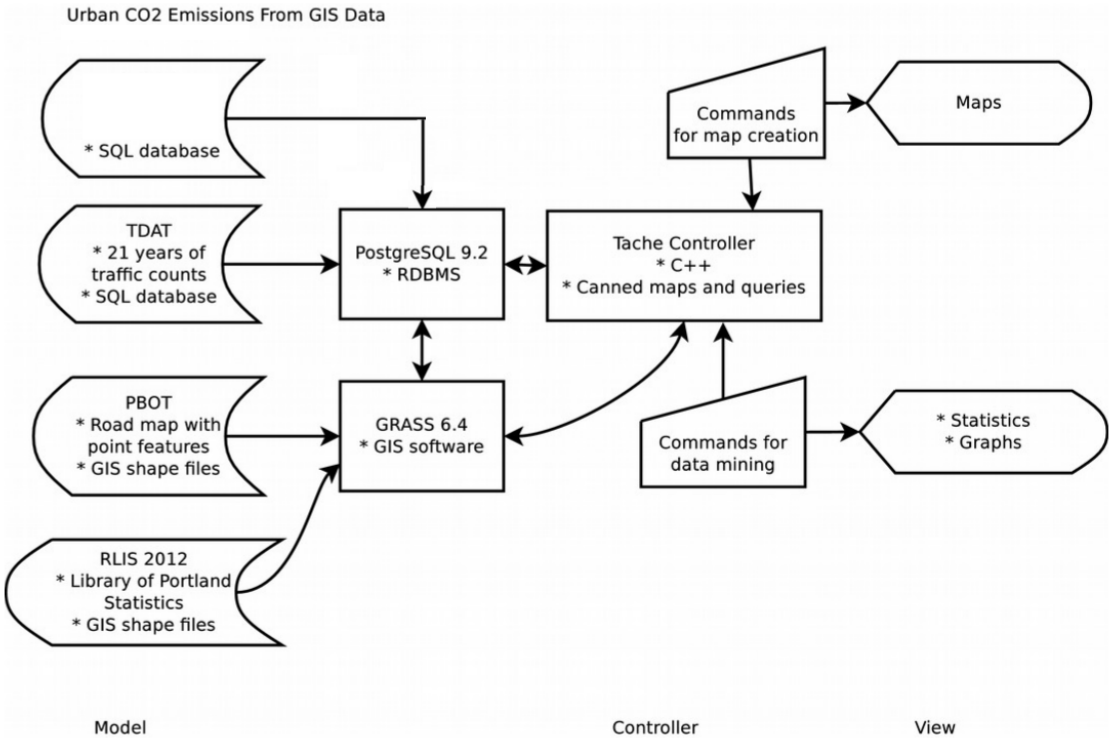


Figure 1.3: Conceptual model of the Circoscope research.

regression [28, Ch. 4] based on the road class (one of 32 classes such as primary arterial, secondary arterial, minor streets) and several Geographical Information System (GIS) statistics for the region surrounding the short road segment of interest could well predict traffic counts. We wanted a model where we could ask "if I went to this road segment on this day, at this time, how many vehicles would I count going by in an hour?" and get a good answer.

When viewed at from a daily traffic count perspective, the "how many vehicles" value, called ADT, is a common and well understood measure of traffic volume. Our model depends on not only how many vehicles are traveling, but on how far they go, so we combine ADT for a road segment with the length of the road segment to obtain another common measure VMT. VMT is suitable input to emissions modeling.

To obtain an accurate model of VMT, we considered many possible drivers for ADT in a process similar to those followed by [29] and [30]. We decided to use road class, population density in circles of various radii, and freeway on-ramp mileage in circles of various radii, to build a linear model of the form

$$\begin{aligned} ADT = {} & \beta_0 \text{ (a constant intercept)} \\ & + \beta_1 \times RC(\text{primary arterial}) + \beta_2 \times RC(\text{secondary arterial}) + \ldots \\ & + \beta_n \times POPDEN(\text{100m circle}) + \beta_{n+1} \times POPDEN(\text{200m circle}) + \ldots \\ & + \beta_m \times ONRAMP(\text{100m circle}) + \beta_{m+1} \times ONRAMP(\text{200m circle}) + \ldots \end{aligned} \tag{1.1}$$

where the RC variables are either 1 or 0 and the β are all constant values determined by ordinary least squares regression. The variables we selected are shown to be significant drivers of ADT.

The ADT values given by Equation 1.1 for each road segment (N=105,178) were multiplied by the length of the segment to give VMT. The VMT were multiplied by a nationwide average emissions factor from an EPA publication (`http://www.epa.`

`gov/climatechange/ghgemissions/ind-calculator.html` accessed 2014-12-08) and summed up to provide a citywide CO_2 emissions rate from on-road traffic. We compared the resulting estimate for the region's CO_2 footprint to Vulcan's estimate for the same region. We summed the estimate to get a single annual CO_2 emissions estimate for the city and compared it to a City of Portland estimate.

We plan to go forward by using Environmental Protection Agency (EPA) MOtor Vehicle Emission Simulator (MOVES) to provide more accurate local emissions factor (EF) based on local fleet composition. This will produce a high resolution (1km x 1km) hourly map of area CO_2 emissions - a source which will be combined with a map of Vegetation Photosynthesis and Respiration Model (VPRM) [31] to give area CO_2 emissions. This hourly EI will be input to the Weather Research and Forecasting (WRF) model, and samples from the model output compared to instrument recordings of CO_2 concentration at three sites in an archive maintained by Dr. Andrew Rice's group at Portland State University (PSU). We expect based on the fine results for a similar project [32] to be able to reproduce the Rice record with some accuracy, thereby ground-truthing our model.

Our model then will build confidence and give tighter estimates for the city's CO_2 emissions. This result will be useful in the fields of inverse modeling of CO_2 sources and sinks, and separately in the discovery and monitoring of the effectiveness of mitigation measures.

We have coined the name "Circoscope" for the traffic density model. We will use the term "TD0N" somewhat interchangeably with Circoscope, although TD0N is generally used to refer to the data cleaning pipeline (the computer code) while Circoscope refers to the linear model which is a product of analyzing the clean data.

Chapter 2 Methods

§2.1 Overview

In the course of building our model, two complicated and in some ways poorly documented databases have been processed to produce a synthetic, summary product which is both concise and accurate. Several best practice methodologies were employed to make the problem tractable. The include the methods of the archivist [33] [34][35], the data miner, the software engineer [36] [37], and the statistician [38] [39].

§2.2 The Archive

The research described here was carried out by analyzing a variety of primary materials. Two collections of records in particular have never been used for CO_2 emissions inventories before, to our knowledge. We adopt archival techniques to help the public in understanding and reproducing our work. We are comfortable that archival approaches are justified because these records, of traffic counts along roads and highways, 'provide "evidence of a person or organization's activities" '[34, p. 29].

Our work primarily involved obtaining, understanding, merging, and cleaning two large raw data sets: TDAT and PORTAL. These are described in the next section, Data Understanding.

TDAT	PORTAL
Hierarchical	Flat
Counting done deliberately, project-specific.	Counters installed permanently
Many outages and special events labeled	Few (any?) special events (holidays, accidents, outages) labeled

Table 2.1: A side-by-side comparison of the very different attributes of the TDAT and PORTAL archives.

§2.3 Data Understanding

To open the methods, we give an overview of the primary sources, TDAT and PORTAL. These have much in common. Both are recordings of traffic counts: the number of times a vehicle passed by on a road in a 15 minute period. The count sites in both are known by the lat/long of a counting site, and the counts can be and were summed to include vehicles passing in either direction and in any lane.

The organization and motivation of the two data sets is as different as can be. These differences are summarized in Table 2.1.

§2.3.1 TDAT

TDAT is a database of project-specific traffic counts maintained by the City of Portland. TDAT is continuously being added to at publication time.

The database is large (885 MB of Structured Query Language (SQL)) and spans a lot of time (1986-2006). TDAT is largely undocumented outside of suggestive table and field names, and in a white binder in the corner of an office at the PBOT. We connected software for graphically representing the data dictionary, in order to create Figures A.8 and A.9.

We obtained a GIS layer from the regional authority which assigns one of 32 road classes to each segment of road in the metropolitan region. A map of part of this area is given in Figure 2.1.

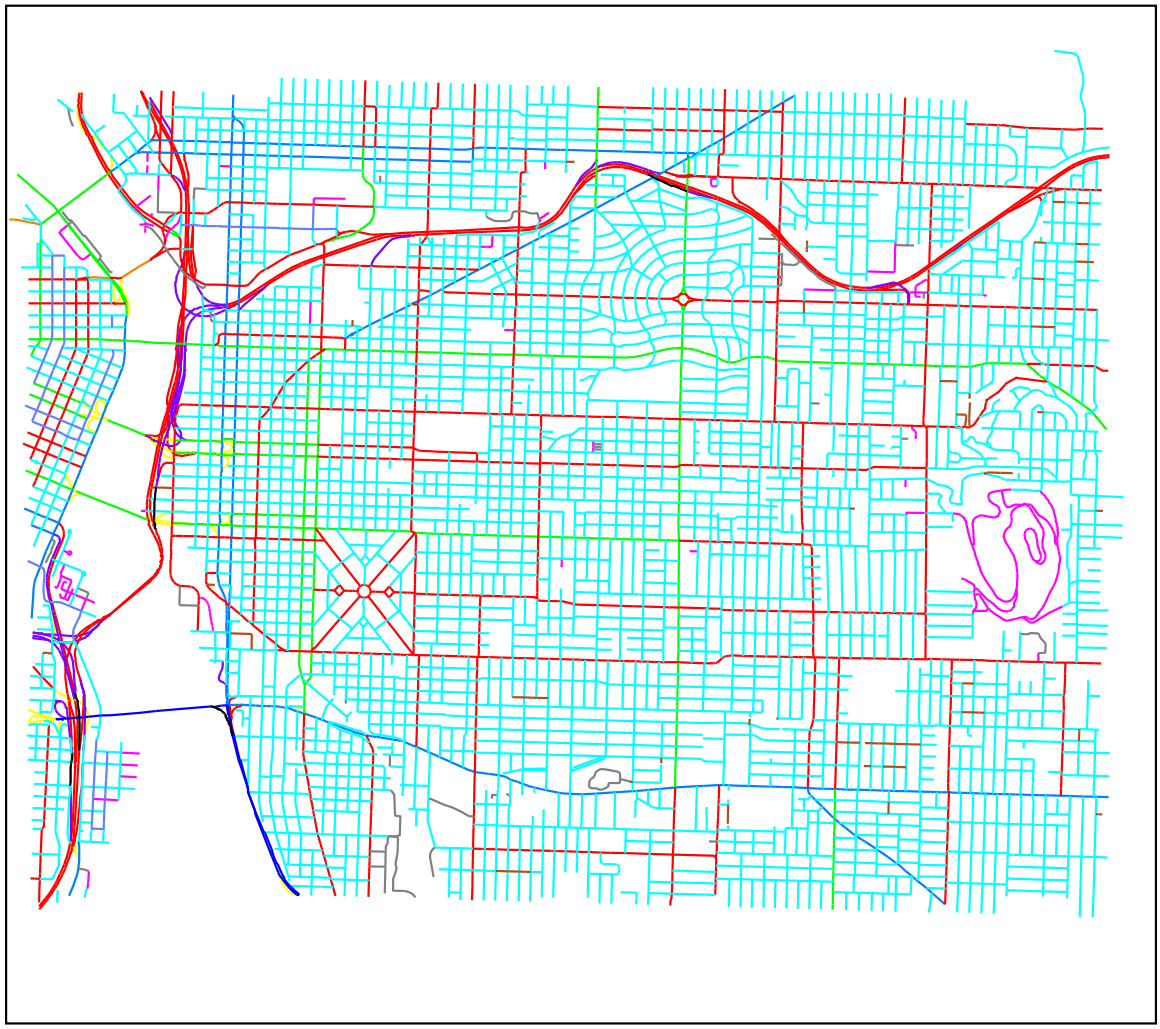


Figure 2.1: A map of a part of the study area showing colors indicating road class. The colors are as given in Table A.1.

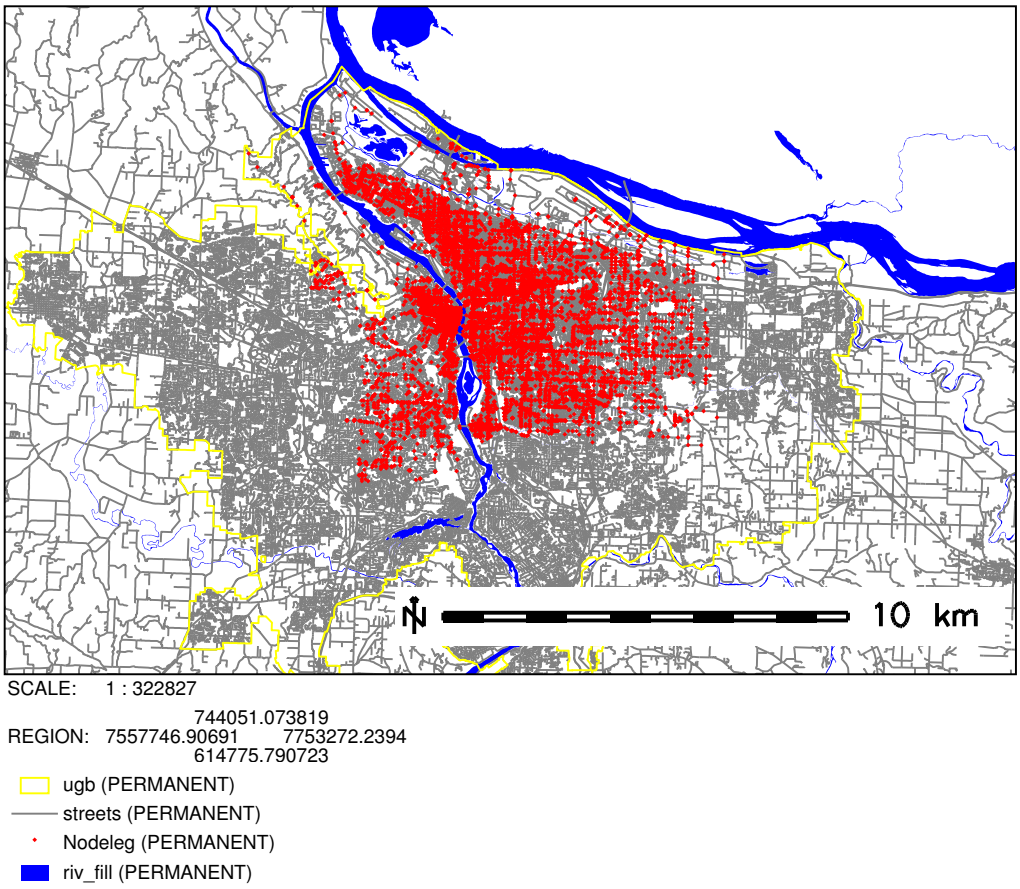


Figure 2.2: TDAT count sites in the Portland Metro Area. Red dots are sites. The yellow line is the Urban Growth Boundary. The blue is the Willamette and Columbia rivers, and the gray lines are streets. Data sources: Metro's RLIS GIS data and the PBOT TDAT2 archive.

To gain confidence in the usefulness of the TDAT data, we mapped the count sites. The result is in Figure 2.2. We observe that coverage is intense and fairly uniform in an area centered on downtown, but no coverage exists outside of the city limits leaving much of the metropolitan area unsampled.

§2.3.2 PORTAL

PORTAL [40] is an archive of regional freeway and highway traffic counts kept by PSU. PORTAL is continuously being added to at publication time. The project combines the resources of PSU, Oregon Department of Transportation (ODOT), Portland Metro, and the US Department of Transportation. Vehicle counts gathered from a network of 732 loop detectors planted in restricted roads in the Portland area are recorded for research purposes. Recordings are made at 351 different stations.

To gain confidence in the usefulness of the PORTAL data, we mapped the count sites. The result is in Figure 2.3. We observe dense and uniform sites throughout the metropolitan region.

§2.4 Data Cleaning and Reduction

> "ItâĂŹs like riding a psychotic horse toward a burning stable" (Robin Williams as Armand in "The Birdcage")

Our problem is similar to probing a crystal with a laser from many different directions to find paths with no defects. Defects that are found are categorized and the suspect data is cleaned out of the final product.

We resolved this problem by building an eight-stage pipeline, where each stage except the last cleans the data, identifying and isolating outliers. The last joins the clean count data to separately prepared "land use" data such as functional

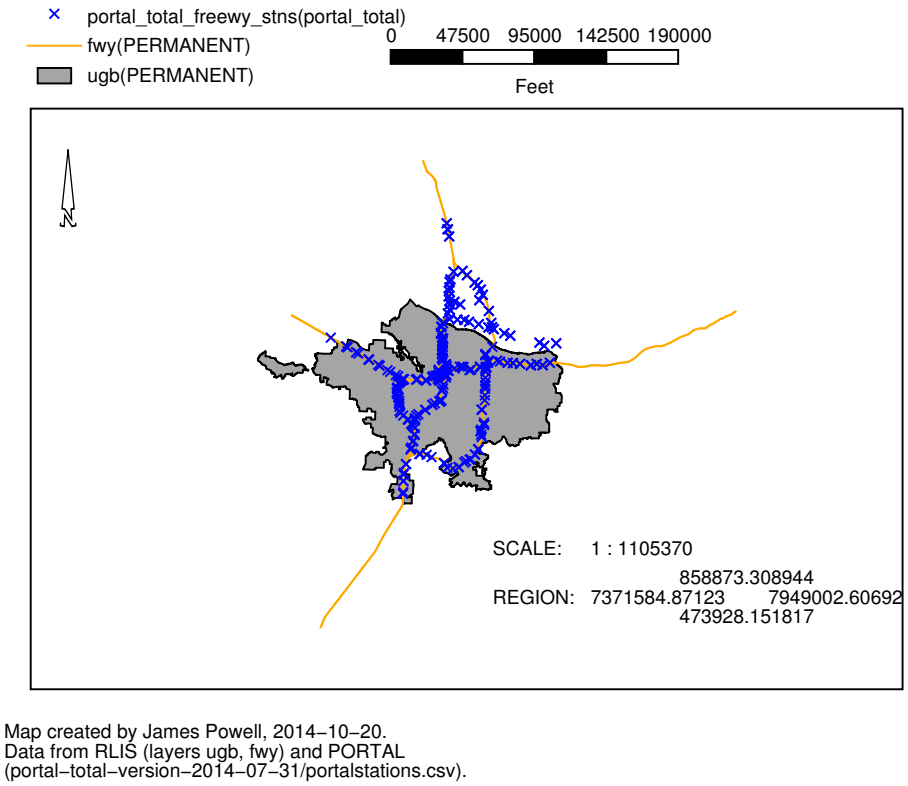


Figure 2.3: PORTAL sites in the Portland Metro Area. Blue crosses are stations, yellow lines are freeways, and the gray area is the urban growth boundary.

road class, population density, and on-ramp mileage in the area. The pipeline is presented in Appendix A.2.

§2.4.1 Data Quality Problems Common to Both Archives

The majority of our data cleaning filter applies equally to TDAT and PORTAL. We used a computing mechanism called materialized views to cast PORTAL in the guise of TDAT to enable a single code base to operate on both databases. We will discuss the common problems now.

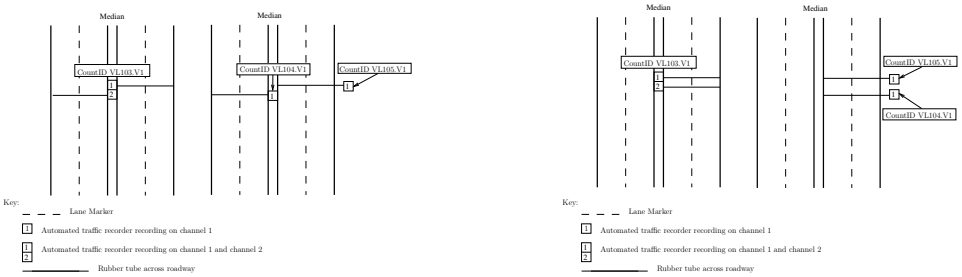
Negative Counts Both archives have a handful of instances where a count of less than zero was recorded. Discovery of this defect necessitated marking the whole counting day as an outlier.

Two (or more) channels of data exist that share only a single channel number. These channels (a TDAT term, these are known as detectors in PORTAL) are expected to be uniquely identified. Some counting days in both archives have multiple channels assigned to a single channel identifier.

Missing Data for a Channel We organize our counting into one or two channels. One channel counting indicates traffic flow in a one-way street (PORTAL highways and freeways are always two-channel). Two channels require traffic flow in two directions.

Both archives have a number of instances of data being recorded for one channel or lane of traffic, but the record for an expected second channel is missing entirely from the archive.

Incomplete Day Of Recordings The vast majority of outliers in both archives is the case where the recorder was only able to obtain part of the full day's recordings.



(a) Normal arrangements. (b) Pathological arrangements.

Figure 2.4: Known TDAT counter arrangements. Numbers inside of square boxes are channels.

A typical example will have recordings starting at 11 AM, and ending at 2 PM.

§2.4.2 TDAT-specific Data Quality Problems

The JAMAR-type traffic counters (pneumatic tubes and recorders manufactured by JAMAR corp.) used by PBOT have multiple channels (multiple tubes connected). The counting arrangements in Figure 2.4 capture all of the possibilities and are expected to be typical of project-level automated counting in any city. We can confidently report that TDAT's long history spanning 21 years and perhaps 8000 forty-eight hour projects has enabled ample exploration of the ways the data can be collected and recorded.

The Number Of Channels Is Not In (1,2) For This Count A slight handful of TDAT counts were marked as outliers because the number of channels was outside our canonical range of 1-2.

Unacceptable (or NULL) Excepttype TDAT has a field `excepttype` which usually indicates normal weekday, or normal weekend. Any alternative, for example the code #BIKE which indicates "count includes bicycles only, no motor vehicles" and the code #CONST which indicates "obstruction count data affected by

construction activity which does not close the entire roadway" [1] are excluded.

No Kounts Were Recorded Under This Count Id We introduce here the TDAT jargon of "kounts". A kount is an integer expressing the number of vehicles recorded as passing the device in a given time period, while a count is the introduction of a project-specific counting event. There are many cases in TDAT where a counting event was introduced (so a count ID was made) but no "kounts" were ever recorded.

§2.4.3 PORTAL-specific Data Quality Problems

PORTAL displayed a number of the same DQ problems that we found in TDAT. Two unique problems with PORTAL are detailed here.

PORTAL Station With No Other In the case of roads with bidirectional traffic, we require simultaneous recording of traffic in both directions. Many PORTAL stations do not supply this comprehensive recording; counts at these stations are filtered out.

§2.5 The Use of the Computer for Reproducibility and Automation

The analysis is enabled by automation. To reproduce our work, one begins with a "tarball" we have on our research computer - a single file which any programmer can easily unpack and build. The build concludes with a single executable file containing machine code and embedded SQL code, R code, and GIS map specifications. This program is executed from a Portable Operating System Interface - Unix (POSIX) command line, and when it finishes (in about 5 days) the Data Preparation and Modeling phases are complete.

[1] Private communication from PBOT's J. Throckmorton, 2014-07-22

As the research proceeds, we are configuring our computer to automatically repeat the above on a rolling basis, as a safeguard against the unfortunate case of a bug being introduced or fixed without notice and documentation, which changes our model in a meaningful way.

It is best practice to reuse software components when possible and to thus diminish the impact of a syndrome or anti-pattern known as "not invented here." We have maximized our use of other people's work, reducing our own work to a mere 10,000 lines of code. The relatively small footprint, which is only 4x bigger than the MAEROS coagulation model, for example, and just 1% of the size of the WRF research-grade meteorological model, is a great aid to us allowing transparency and parsimony to be the rule of the day. The software stack is shown in Figure 2.5, and most of these components are probably recognizable to the reader because of their famously useful rôle in a great number of computer-based experiments.

We were able to achieve this high level of automation and reproducibility by virtue of an initial, risky effort to build a program-that-programs, which we call the Zombie Caterpillar. The driver, called *tache* [2], at the top of the stack is able to initiate processes amongst any other component in the stack, locally or over a network, and interact with that component in complex ways. One example of the utility of the Zombie Caterpillar is that among the 10K lines of driver code is an Application Programming Interface (API) of great generality and depth for carrying out GIS analyses. Another example is the ability of the driver to call out to R for calculating statistics and other data reduction and error analysis steps. We also use Embedded SQL [3], a type-safe and null-aware means to use the archive from C++.

[2] *Tache*: a French word meaning task, but also stain, as in the stain of CO_2 in our atmosphere.
[3] Embedded: the interpreted SQL is embedded into a compiled C++ code.

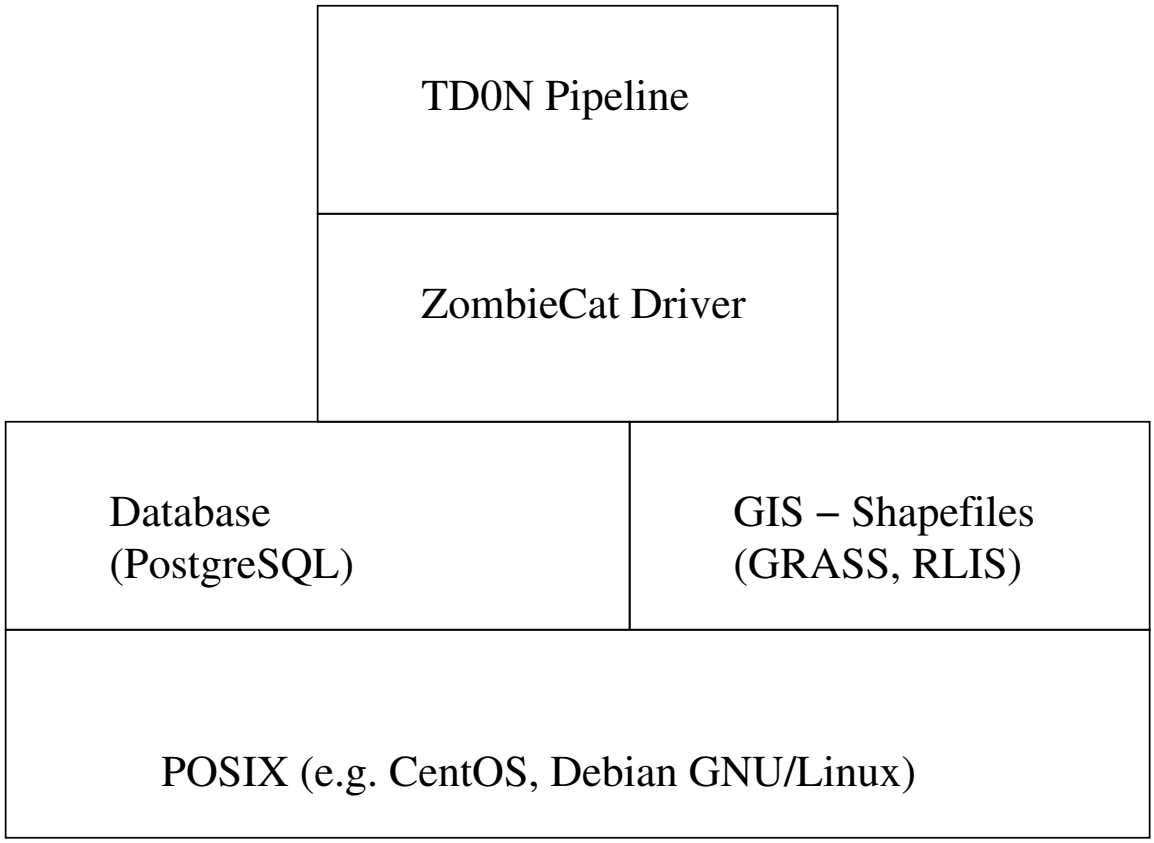


Figure 2.5: A schematic of the software stack used in the research.

§2.6 Statistics

We carried out an extensive statistical analysis of the cleaned data after our cleaning process completed. These steps involve data reduction and error analysis [41], and stepwise multiple linear regression [38] [28]. Quantile-quantile plots [42] [43] were used as diagnostics to demonstrate the normality of the distribution of the ϵ of the traffic counts in our linear regression. The linear regression was analyzed by holding back one third of the data for validation, and also by graphing the data at various levels of detail and using different slices or subsets to produce results and to test theories including a putative large weekend effect and a smaller seasonal signal.

This concludes the overview of our methods. We will now describe our methods in greater detail.

§2.7 Data Preparation

To select our data set, we started with all available counts. We then began the process of accumulating a set of very clean 24-hour days of steady counting. We then assigned a road class to each count site (Appendix A.5).

These two types of data, typified by the tuple (number of counts, timestamp of the beginning of a fifteen minute interval, lat, long, road class), are the desired result of preparing our data sets.

§2.7.1 TDAT

TDAT data were prepared by obtaining SQL for the entire database and associated GIS layers supplied by PBOT.

§2.7.2 PORTAL

As described in Section 2.3.2, we prepared the PORTAL data by making a GIS layer of the count stations and obtaining a complete record of the raw counts using the PORTAL web site. We decided that the best approach would be to use SQL views to cast one of the data sets into the form of the other. Success here meant that a single data cleaning code could manage both archives with little special-casing.

This approach proved beneficial. We found it easy to put PORTAL into the form of TDAT using Cartesian joins (a process where by data with packed coding, meaning little redundancy, is exploded to produce a simple, flat table with a lot of redundancy), and since the data are so similar in essence although very different in organization this step represented a classical unification of apparently disparate systems. It is very simply expressed, and it makes our system robust to accepting data in either form, while both forms are apparently completely logical in the context of the Bureaus that record the data.

The SQL code for this adapter is given in Appendix A.6.2.

§2.8 Deriving a Canonical Model of ADT

We require a continuous source of CO_2 for our program of eventually cross checking our model against measured CO_2 concentrations, but we're given discrete (in both time and space) examples in which the passage of a vehicle is noted at a particular location and time. We need to fill in spatial and temporal gaps in the discrete data to give reasonable values for traffic counts where no count data exist.

The map in Figure 2.6 demonstrates both the problem and our solution for the spatial part of the problem. The points marked X are the discrete historical cases where vehicles definitely were observed passing by during the time period of interest. The color of the lines is given by our model. We desire a close agreement

between these values.

§2.8.1 Selection of Land Use Regression Variables

The continuous values along the road segments are the product of a linear regression. We asked: what factors drive traffic volumes? As we see in [44], a number of factors are possible: population density, greenness indices, distance from the center of the city, even the vertical slope of a road segment. We generated the list of Table 2.2 and selected just population density and meters of restricted road onramp-type road in circles of 100m, 200m etc. out to about 800m.

We will now derive an equation of the form $ADT = a_1 RC_1 + a_2 RC_2 + \cdots a_n POPDEN + a_{n+1} ONRAMP$, and describe a process for filling the gaps in the emissions inventory, gaps in between count sites. The analysis proceeds by augmenting a matrix whose initial contents are only the lat/long of the count sites, 6000 of them, so it's a 2x6000 matrix which for reasons which shall become clear, we call node-leg_866_vector_repo_001. We will augment this with road class, population density, and on-ramp measurements.

§2.8.2 Establishing the Log-normality of the Counts

> "In general, we must keep in mind that there is no guarantee that use of these transformations will necessarily be better than analyzing the proportions directly; much depends on the data. The effectiveness of a transformation is best assessed by trying it on the data and then checking the fit of the model and the pattern of residuals that results." [28, p. 239] on using log- or arc-sine transformations on proportional data.

> "In the standard classifications, skewness in the distribution of errors

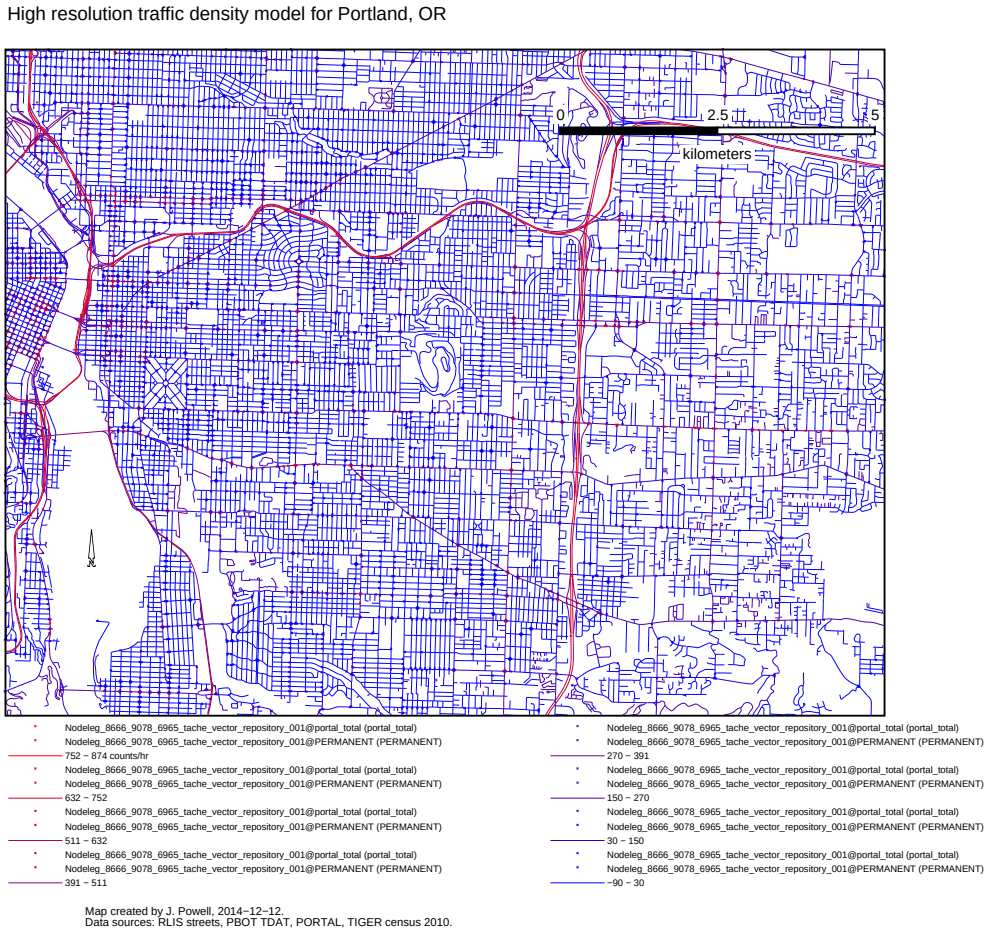


Figure 2.6: Actual counts and model counts on a map of Portland, OR. Vertices are raw data, while the color of the line segments is modeled based on road class, population density, and on-ramp mileage.

Variable	Subjective Importance (out of 10)
Road class	10
population density	10
meters of hwy on-ramps	10
percent of land cover that is impermeable (or tree covered, or lawn)	5
slope	0
aspect	0
proximity to (school, hospital, groc. store, etc.)	0
zoning mix	0
could train neural net on all poss. variables - soln is then general	0
temperature, precip, humidity	0
day of week	
weekend or not	
holiday or not	
month	
quarter	
season	
year	
decade	
major sporting event or concert that day	
Obama visits	
crime rate	1
direction of travel (EW vs. NS)	
proximity to river	
CO2 intensity recorded by Dr. Rice at 3 area locations.	0
Other Air Quality measurements by L. George	
Traffic as counted by other counting systems (L. Hill's thesis)	
New metric: is it on a line of steep travel gradient	
Distance to Max, Bus (limit the years in which red/yellow Max line operated	
L. George's student who has time series of NOx in PDX	
Impervious surface area (ISA)	
Volume-weighted Road Density	

Table 2.2: The result of our brainstorming effort, informed by research, about factors that could sensibly affect traffic counts in the city.

> tends to produce too many significant results in F- and t-tests. In addition, there is a loss of efficiency in the analysis." [45, p. 325]

We began our research expecting to find that our data would follow a log-normal distribution, because of the proportional nature of the data. Log-normality is expected if the treatments have an effect that is proportional instead of additive: an increase of 5 % per unit of treatment instead of an increase of 5 units [45, p. 329]. We consider it reasonable that for a given site, the evidence of the population density or mileage of on-ramps in the area around the site does suggest a proportional increase or decrease of the ADT compared to the mean ADT for that road class. This makes a log-normal distribution of the traffic counts likely.

Our exploration supports this assumption. Graphing the data is the best way to observe log-normality, and we see evidence of log-normality in the histograms of Figure 2.7, in which we plot the raw data. The result is obviously not Gaussian, or the histogram would look bell-shaped and the Q-Q plot would approximate a straight line. The following Figure 2.8 displays the data after a log transform. The Gaussian bell curve is apparent.

Homoscedastity, which is the property of constancy of the variance of the error term over the full range of the model fitted to the data, is an assumption of Ordinary Least Squares (OLS) fitting (we include a complete list of OLS assumptions in Section 3.0.11). It is only by homoscedastity that OLS is the "minimum variance unbiased estimator" [4]. In the absence of homoscedastity, the estimator has more variance, but it does not pick up a bias (ibid).

When the data are log-normal (that is, the logarithm of the raw data are normally distributed and the raw data themselves have the greatest number of counts at the low end of the distribution and exponentially decreasing frequency of counts

[4] https://stats.stackexchange.com/questions/52089/ what-does-having-constant-variance-in-a-linear-regression-model-mean/52107#52107 accessed 2016-10-23.

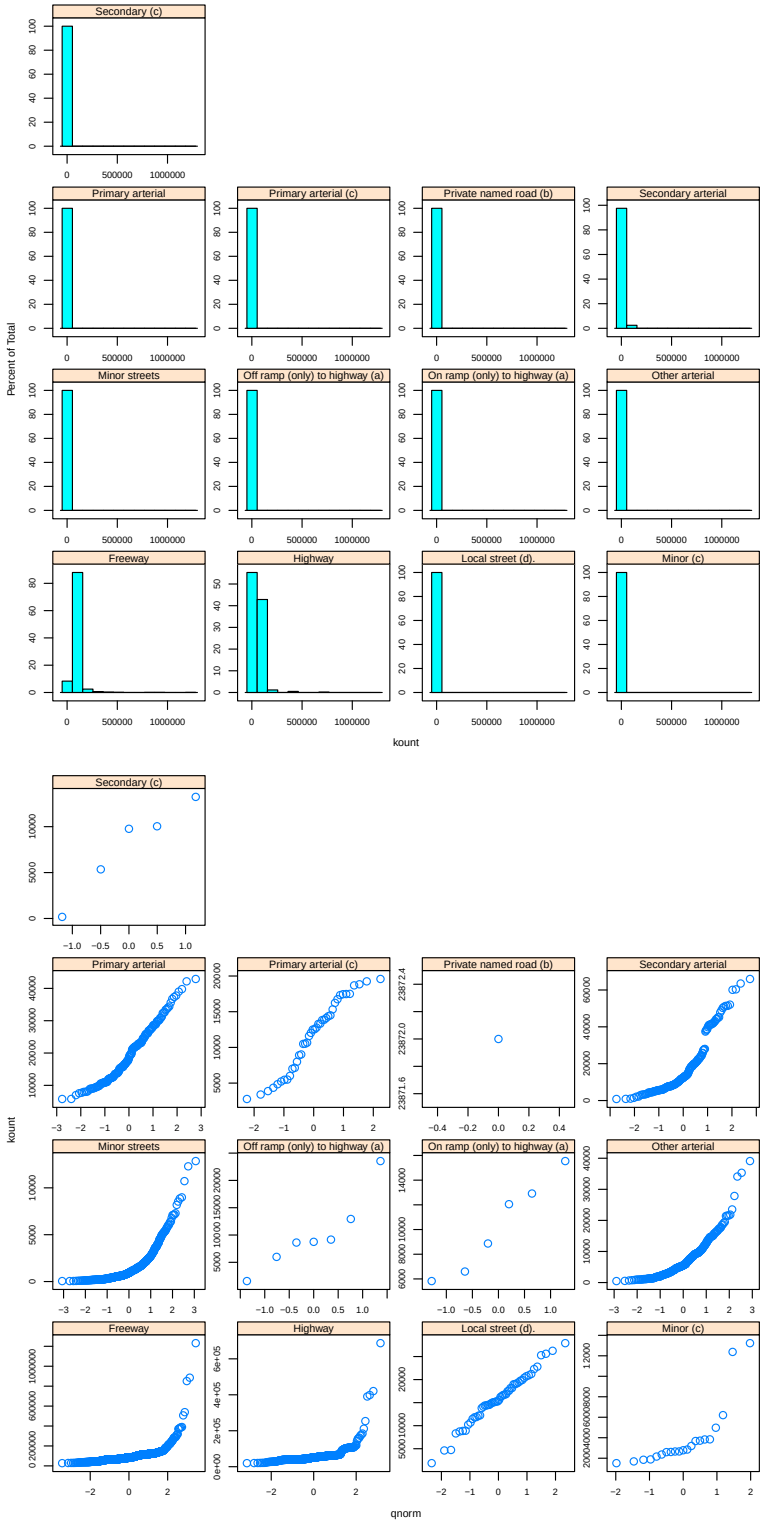


Figure 2.7: Histograms (top) and QQ plots (bottom) for a random sample of weekday ADT, untransformed. No indication of normality is seen in these graphs. Each panel is one of the road classes. Codes (a), (b) etc. are as in Figure 3.5.

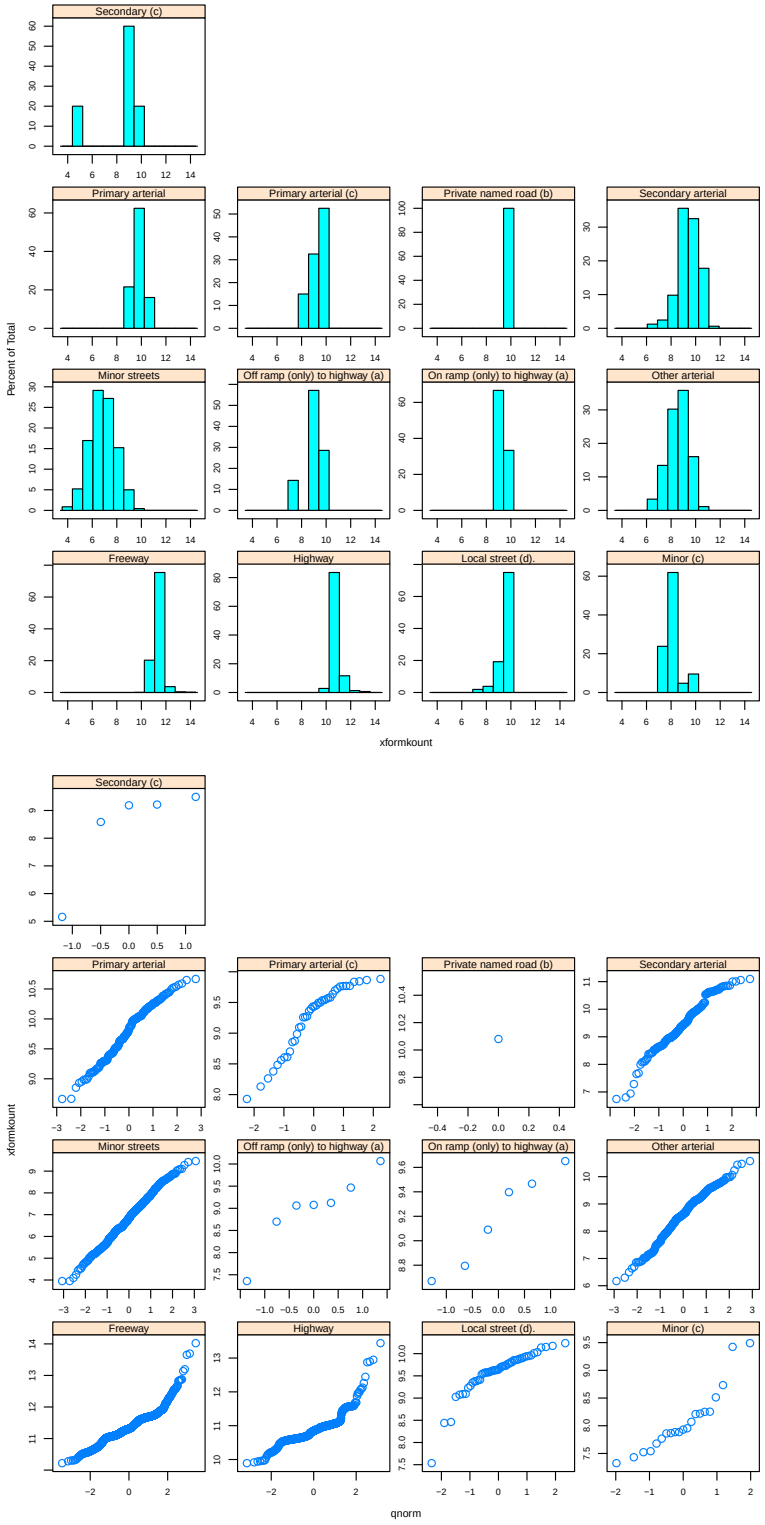
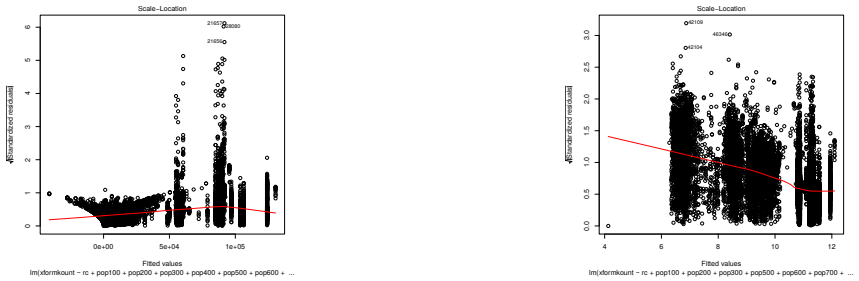


Figure 2.8: Histograms (top) and QQ plots (bottom) for a random sample of weekday ADT, log-transformed. Normality is seen in these graphs. Each panel is one of the road classes. Codes (a), (b) etc. are as in Figure 3.5.



(a) Raw data. (b) Log-xformed.

Figure 2.9: In (a), the square root of standardized residuals plotted against the fitted values for all of the raw data, without a log-transform. Observe the definite proportionality of the variance of the residuals against the fitted values. In (b), after the log-transform the residuals appear more normally distributed around zero.

in the higher ranks), the variance of the residuals will become a linear function of the count [45, p. 329]. Figure 2.9 displays a proportionality of the variance of the residuals that is clearly linear over some parts of the data.

Additionally, the graphical evidence is clearly indicative of a problem with the constancy of the variance until the log-transform, after which the variance is highly constant.

We suspected log-normality in the count data because count data (missing symmetry around the origin) do typically follow a log-normal distribution. We plotted the raw data as a histogram and we examined the residuals using a quantile-quantile plot to look for signs of the lack of normality. Normally distributed residuals are one requirement for a valid linear model.

We plotted as a histogram the untransformed and log-transformed ADT for all road classes, and tested the arterial and freeway counts for normality using Shapiro-Wilk (`shapiro.test` in R, [46, p. 382]). We tested for normality using the same test again after a log transform. The code for these plots and tests is in Appendix A.6.4.

§2.8.3 Using Multiple Linear Regression to make a Linear Model

We obtained an ADT record of 1.1 million count/days. We used our ADT data as input to a multiple linear regression following well established practice. The raw data were sectioned randomly using a high quality random number generator (R's `sample`) with $\frac{2}{3}$ of the rows marked "use for model" and the remaining $\frac{1}{3}$ of the rows marked "use for validation" using the code in Appendix A.6.9.

The raw data were log-transformed, a procedure justified by the log-normal structure of the data.

A stepwise algorithm, in particular `step()` in R, which uses the AIC metric (a metric which balances significant of each term in the regression against simplicity gained by having fewer terms) to decide on the importance of keeping a term, which resulted in a regression with 26 terms and an r^2 of 0.89. Without stepwise, the regression has 28 terms. The terms for pop400 and pop800 were dropped during the regression as not contributing sufficiently to the AIC metric. The r^2 is not affected by these two terms being dropped.

The results are presented in Section 3.0.13.

§2.8.4 Obtaining VMT

We followed the technique of [18], in which VMT is calculated by multiplying ADT by the length of a stretch of road.

§2.8.5 The Multiple Linear Regression

The software used in the linear regression is described in [46, pp. 403-414] and in [47, pp. 100-114]. This is research-grade linear regression applied in the most standard idiomatic manner.

The traffic counts emitted by the TD0N pipeline of Figure A.1 were subjected

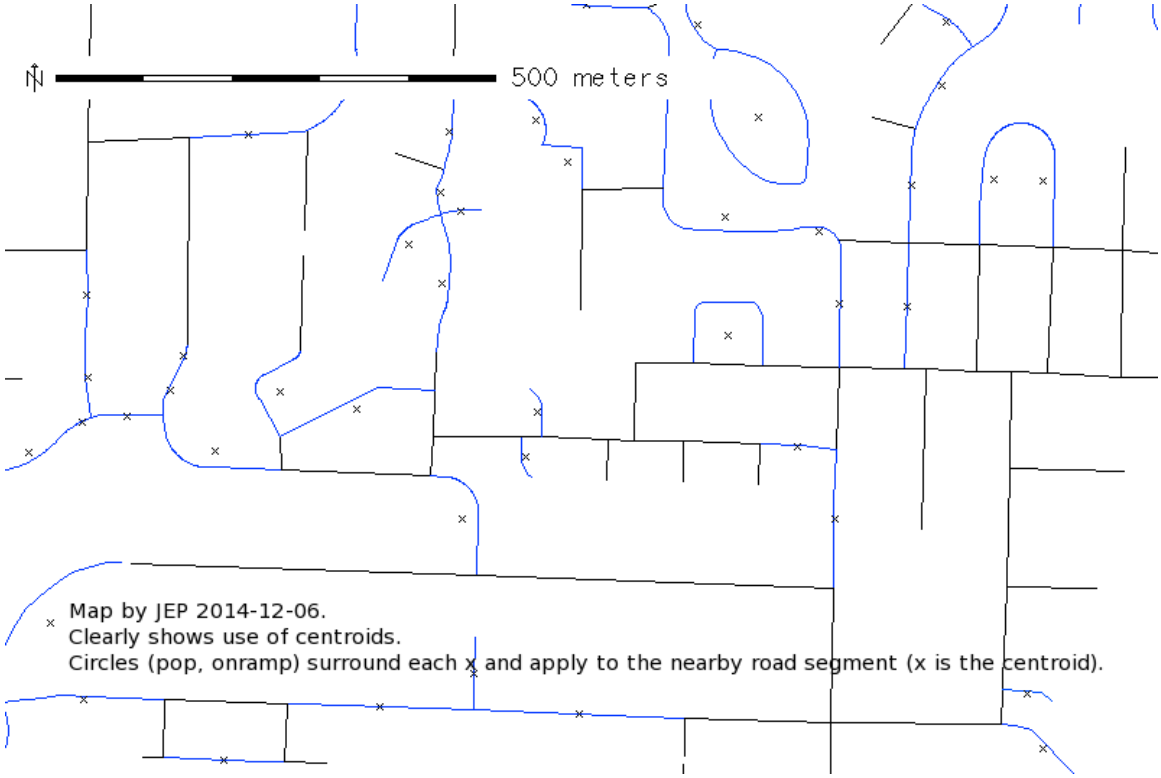


Figure 2.10: Centroids assigned to each road segment in RLIS streets.

to a battery of tests in an attempt to detect irregularities. We conducted this work as a series of experiments.

§2.9 Preparation for Massively Parallel LUR

To obtain the pseudo-land use variables of population density and on-ramp density for each road segment in the city, we first needed a representative point for each road segment. Ideally this would be a point halfway along the road segment, but the technique we landed on gave us centroids for the geometry of the line segment. These centroids are indicated in Figure 2.10.

§2.10 Massively Parallel LUR

We developed a layer of N=10^5 centroids for road segments in order to use the linear model TD0N. For each centroid we evaluated several Land Use Regression (LUR), specifically population density from the 2010 US Official Census and on-ramp density (km of onramp per km^2). We evaluated these inside circles centered at the centroids with several radii: 100 to 900m for population density, and 100 to 500m for onramp miles, with 100m increments. The calculation was estimated to require 28 days for completion. To shorten it, we developed a technique for using 250 central processing units (CPUs) on our Gaia computing cluster. This work required first learning how to deploy the GIS on those 250 CPUs. Once the GIS was operational however, this embarrassingly parallel computation completed in two days.

§2.10.1 Population Density

Population density was acquired from the US Census, 2010. The map of Figure 2.11 was created from the raw census data.

§2.11 Exploration

We constructed and used a few tools for data exploration. Our first look at the data graphically is the box plot of Figure 2.12 on page 35.

§2.12 Hypothesis Testing

We formed a series of logical hypothesis that we knew could be tested using the traffic count database.

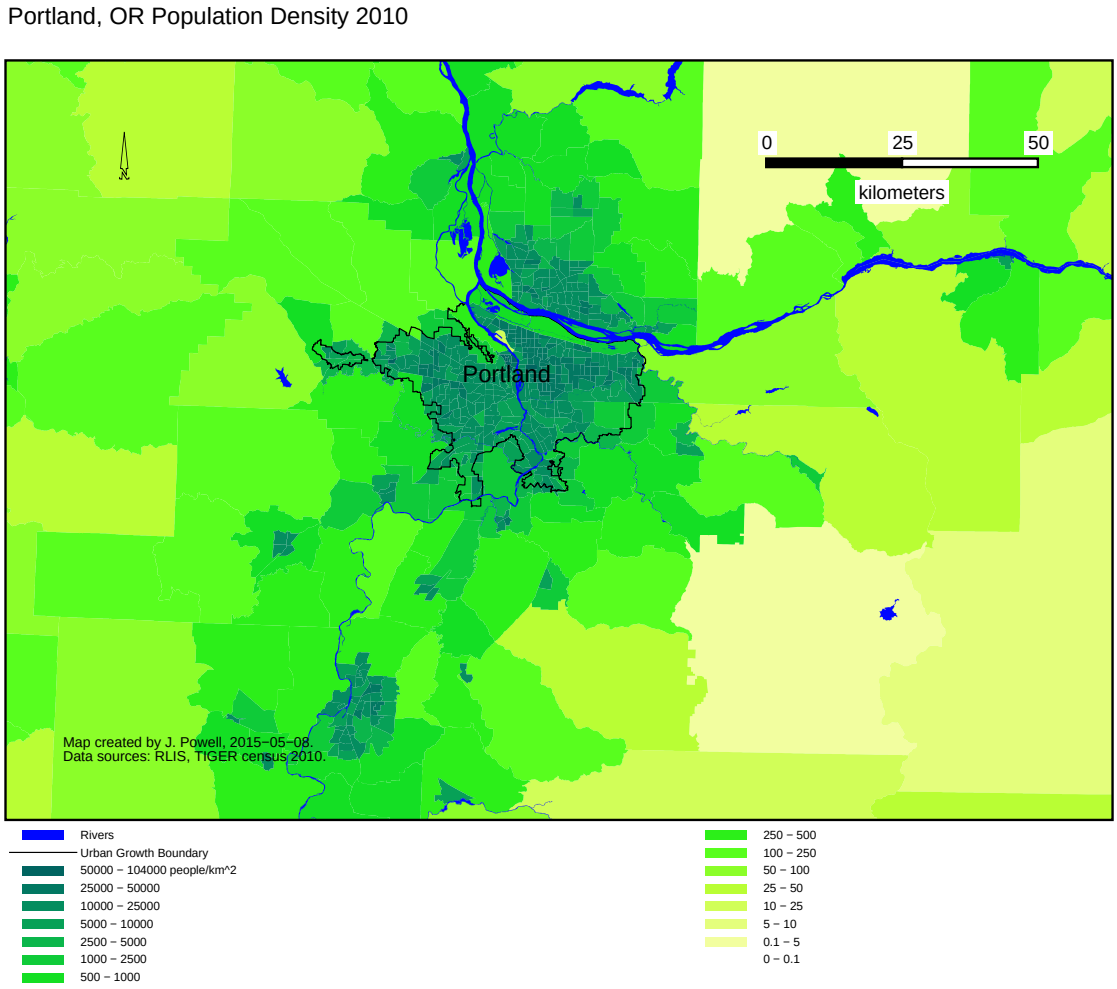


Figure 2.11: A population density map of the Portland, OR region. The rivers are a composite of RLIS's fine Metro-area base layer and TIGER's coarser state-wide coverage of waterways. The data are compiled at the level of the census tract, which is approximately the scale of a neighborhood.

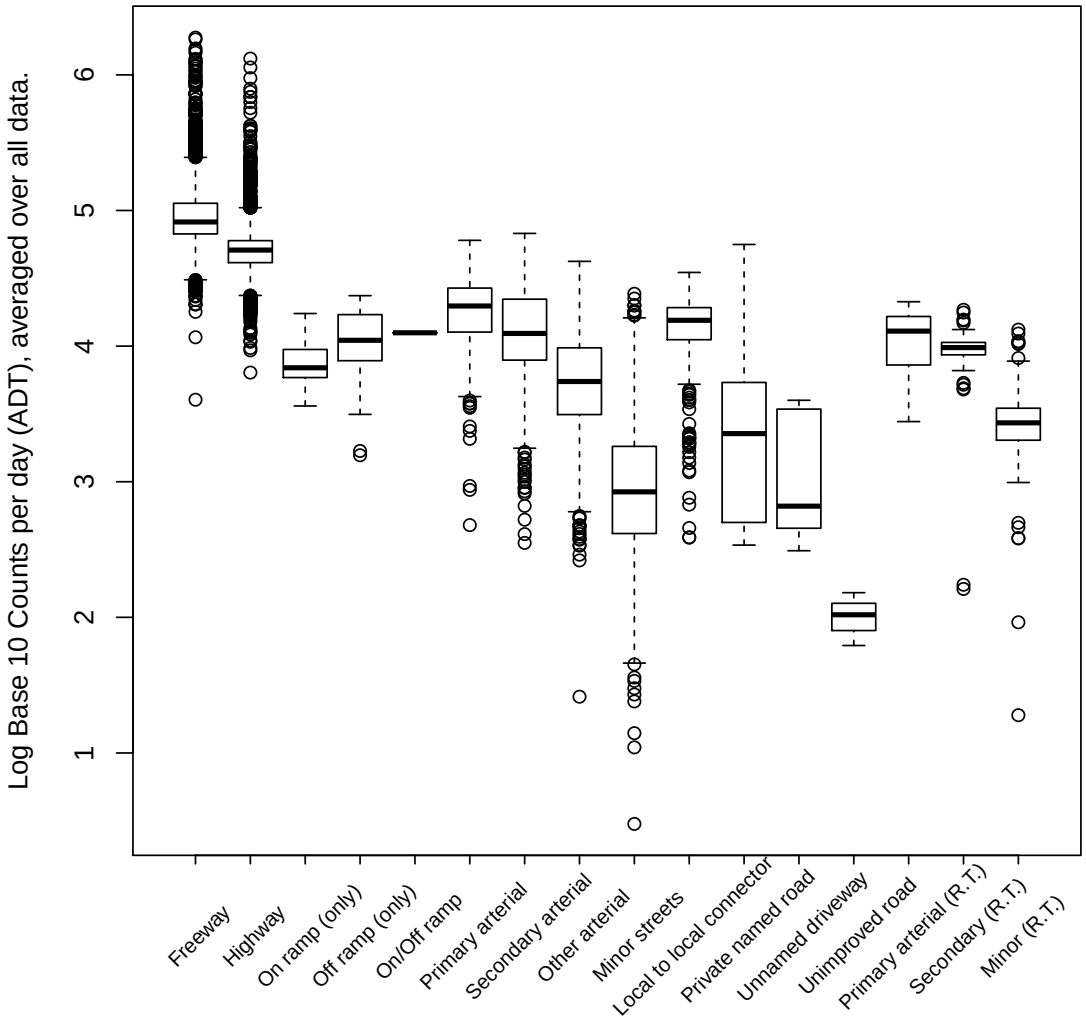


Figure 2.12: A box plot of the combined PORTAL and TDAT traffic count data after cleaning. N=34,308 ADT values were used, each representing one day of counting at one site. The boxes represent the interquartile range: 25 % of the data lie below the bottom line and 25 % lie above the top line, while the interior line shows the median. Adjacent values are represented by "whiskers" (dashed lines) extending off of the top and the bottom of the box. Calculation of the adjacent values is described in [46, p. 242]; they show the approximate range of the data. Extreme values outside this range are drawn as circles.

§2.12.1 Significance of Terms in the Linear Regression

> "The one-way analysis of variance model is one of the most useful models in the field of statistics. Many experimental situations are simply special cases of this model. Other models that appear to be much more complicated can often be considered as one-way models." [48, p. 2]

We believe that some of the terms we shall attempt to regress against will have a significant non-zero slope.

§2.12.2 Normally Distributed Residuals

"Another way to view a distribution is the quantile-quantile (Q-Q) plot." [46, p.241]

It is routine nowadays to examine the quantile-quantile plot of a linear fit's residuals [43] [47, p. 106]. We constructed a toy linear regression using values from a published table of Gaussian deviates [49] to practice linear regression and diagnostics. This code is present in Appendix A.6.1.

We used the same technique to produce diagnostic quantile-quantile plots for the real, entire data set.

§2.12.3 Weekend Effect

Common sense suggests that traffic will be strongly attenuated on the weekend. This effect was clearly seen in a study of traffic in a 2012 study of Lebanon [50]. Closer to home, work by Kendrick and George in 2015 [51], show this weekend effect.

To test for the presence of a weekend effect, we use the methods described in [39]: a t-test for the hypothesis that there is no difference between weekend and weekday mean values.

§2.12.4 Bimodal Distribution (Rush Hours)

Kendrick and George in 2015 [51], show the expected bimodal distribution on one of Portland's arterials. We show the same bimodal distribution with a morning rush hour and an evening rush hour. Another study in Lebanon showed the rush hour bimodality [50] which we reproduce for Portland.

§2.12.5 Seasonal Effect

We investigated the possibility of a seasonal effect on the amount of on-road traffic. Waked et. al. saw more traffic in winter and less in late summer/early fall in Lebanon [50].

§2.12.6 Moss Nitrogen Density Experiment

In collaboration with Dr. Andrew Rice, a data set of elemental analyses on moss was obtained. Errors in the latitude and longitude of the count sites were corrected and the sites were the moss samples were taken were mapped in Figure 2.13. We attempted to find a correlation between high (low) traffic counts and high (low) nitrogen concentrations in the moss.

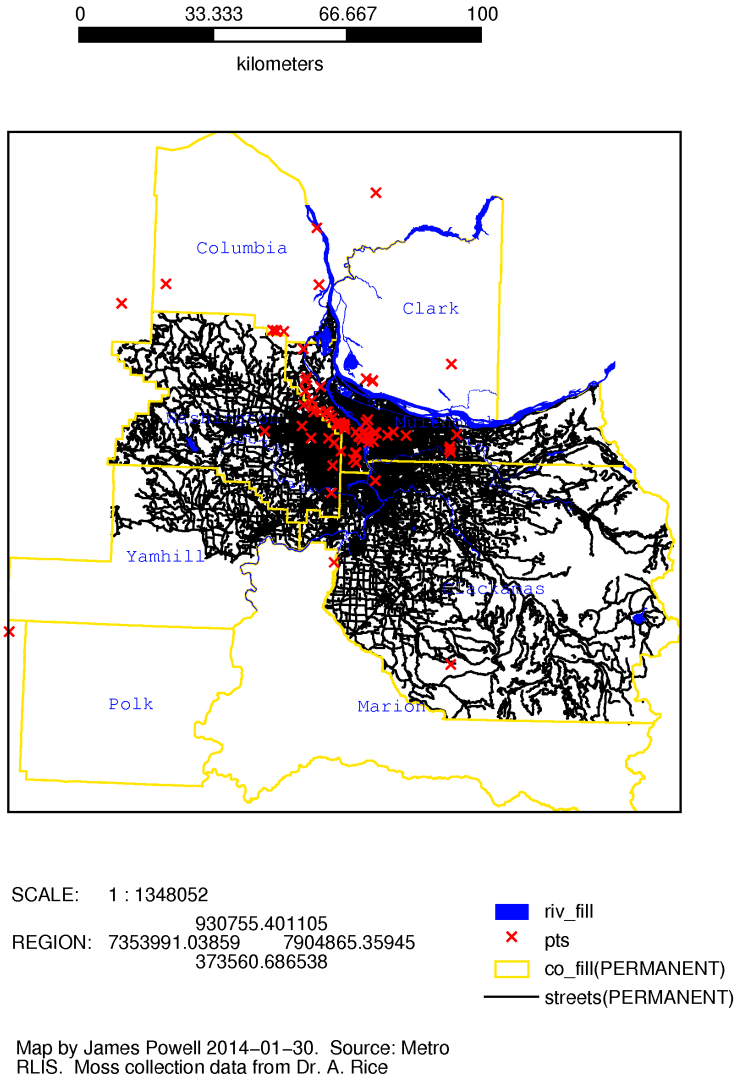


Figure 2.13: A map showing all 81 of the moss experiment sample sites. Red crosses are sample sites. Counties are named.

Chapter 3 Results/Discussion

> 'A basic problem in analyzing a time series is to study the pattern of the correlation between values at different time instants, and to try to construct statistical models which "explain" the correlation structure of the series.' ([52, p. 2])

Our analysis proceeds by a series of hypothesis, tabulated in Table 3.1.

Our analysis begins by analyzing the combined PORTAL and TDAT product. We will note and analyze the sets separately when any qualitative difference emerges.

§3.0.7 Tables of Outliers

In data cleaning, tables of locations and in some cases location+days were made documenting problematic data. Any data that did not seem to be part of a a completely normal 24 hour count was discarded. Approximately 48 % of the data was not used to build the linear regression due to these irregularities, which are cataloged in Table 3.2 and Table 3.3. The code producing these numbers is given in Appendix A.6.11.

For PORTAL, the results are given in Table 3.3.

ID	Description
H01	Our ADT values compare well with other published estimates.
H02	High (low) moss nitrogen concentration correlates with high (low) traffic counts.
H03	The Shaprio-Wilk test will confirm that the count data are log-normal.
H04	Linear regression is appropriate for the traffic count data.
H05	The residuals of the linear regression are homoscedastic.
H06	The linear regression is statistically significant.
H07	That a linear regression will predict ADT in 1/3 of the data when the remaining 2/3 is used to make the model. The regression is based on road class and "land use regression variables" (population density and on-ramp mileage inside circles of various radii around the count site).
H08	There are few outliers in the cleaned data.
H09	Rush hours will produce a bimodal trend in the diel, and weekends will be highly attenuated in counts.
H10	Small seasonal differences are expected.

Table 3.1: A table summarizing the hypotheses tested in this research.

id	description	location/ days affected (approximate)
306	the number of channels is not in (1,2) for this count. A count represents a case where a device was recording. These are identified by unique count IDs.	6
311	Negative kounts (less than zero) were recorded. A kount is an integer expressing the number of vehicles recorded as passing the device in a given time period.	6
310	missing data for a channel	528
304	two (or more) channels of data exist that share only a single channel number.	1188
301	Unacceptable (or NULL) excepttype. We expect to find e.g. "Normal Weekday". ExceptType codes interesting facts about the count day, such as that it was on a weekend, or a holiday, or any ad hoc note such as "Lane Count" or "Bike Lane".	4268
298	no kounts were recorded under this count id	9170
303	incomplete day of recordings	56168

Table 3.2: The types of outliers located in TDAT with an approximate number of location/days affected. The id column is a unique identifier assigned to the outlier type.

id	description	location/ days affected (approximate)
296	PORTAL station with no other - it sits on one side of the freeway with no station on the other side.	unknown
309	Negative kounts (less than zero) recorded A kount is an integer expressing the number of vehicles recorded as passing the device in a given time period.	7
305	two (or more) channels of data exist that share only a single channel number.	588
307	missing data for a channel	2328
308	PORTAL count w/midday zero kounts (hr between 5 and 20)	3110
302	incomplete day of recordings	12987

Table 3.3: The types of outliers found in PORTAL with an approximate number of location/days affected. The id column is a unique identifier assigned to the outlier type.

§3.0.8 H01: Our ADT values compare well with other published estimates.

As one example, [51] reports "SE Powell Boulevard [in Portland, OR USA] is a major arterial roadway that runs east/west with peak hourly traffic volumes of 2800 vehicles and 28,000 Average Annual Daily Traffic (AADT)".

The Regional Land Information System (RLIS) streets shapefile identifies SE Powell Boulevard as having road class 1300 (primary arterial). Our data shows that for this road class generally the weekday AADT as being 20335.

We found that of all of the sites recorded in PORTAL, the ADT over the nine year period sampled is 83808 vehicles/day. This is within the range of values reported for these same roads in the the "Atlas of Oregon" [53], which gives a number of 73,700 vehicles per day passing on the I-5 freeway in the year 1998.

Our work to build an EI to compare with other published estimates indirectly validates our ADT model (see Section 3.0.19).

H02: High (low) moss nitrogen concentration correlates with high (low) traffic counts.

§3.0.9

We evaluated TD0N at sites where plant moss samples had been collected and fed into an elemental analyzer, and searched for a correlation between TD0N and [N] . demonstrating a wider and possibly more important application of the model in investigating transport-related air quality issues. We used the same method as [54], which [11] calls "volume weighted road density index" to predict N for moss.

We were not able to find evidence for this hypothesis. This experiment used an early version of our regression model which did not include freeway information. We were also unable to complete the experiment at 21 of 86 moss collection sites because they're outside of the Metro region and we have been unable to obtain street maps with the requisite road class information for this region. The map in Figure 3.1 shows these 21 sites. Our numerical analysis produced the graph in Figure 3.2. The best fit line has a surprising negative slope, which is statistically significant ($p < 0.001$). We believe that either or both of the limitations of the experiment (lack of freeway, lack of sites in remote regions), both of which reduce the available experimental sites to a highly homogeneous collection of inner-city non-freeway variety, make detecting this signal - which is obvious in other maps of the nitrogen concentration - impossible.

H03: The Shaprio-Wilk test will confirm that the count data are log-normal.

§3.0.10

We used the Shapiro-Wilk test to confirm our suspicion that the count data are log-normal, but the results indicate that the distribution of ADT for the freeway and arterial data are still unlikely to be normally distributed ($p < 2.2 \times 10^{-16}$).

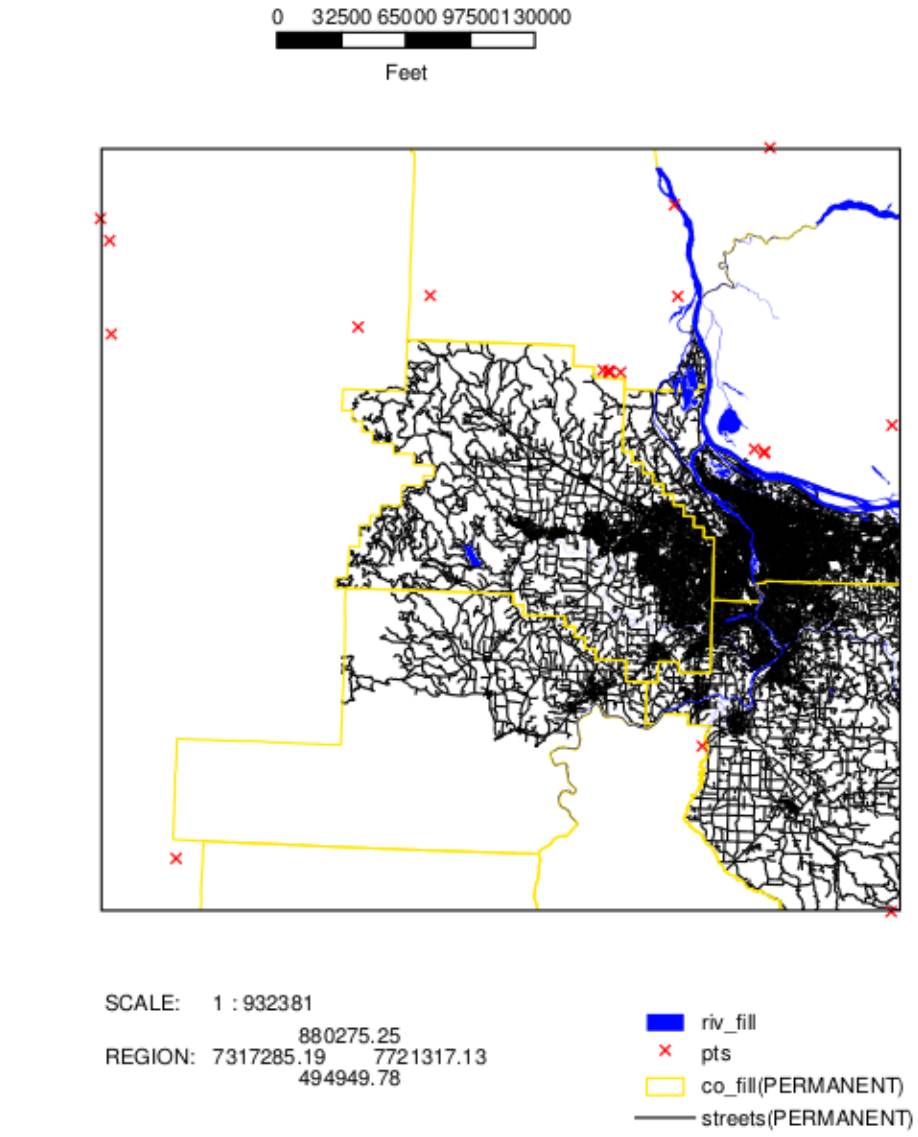


Figure 3.1: Moss collection sites outside of our ADT study area.

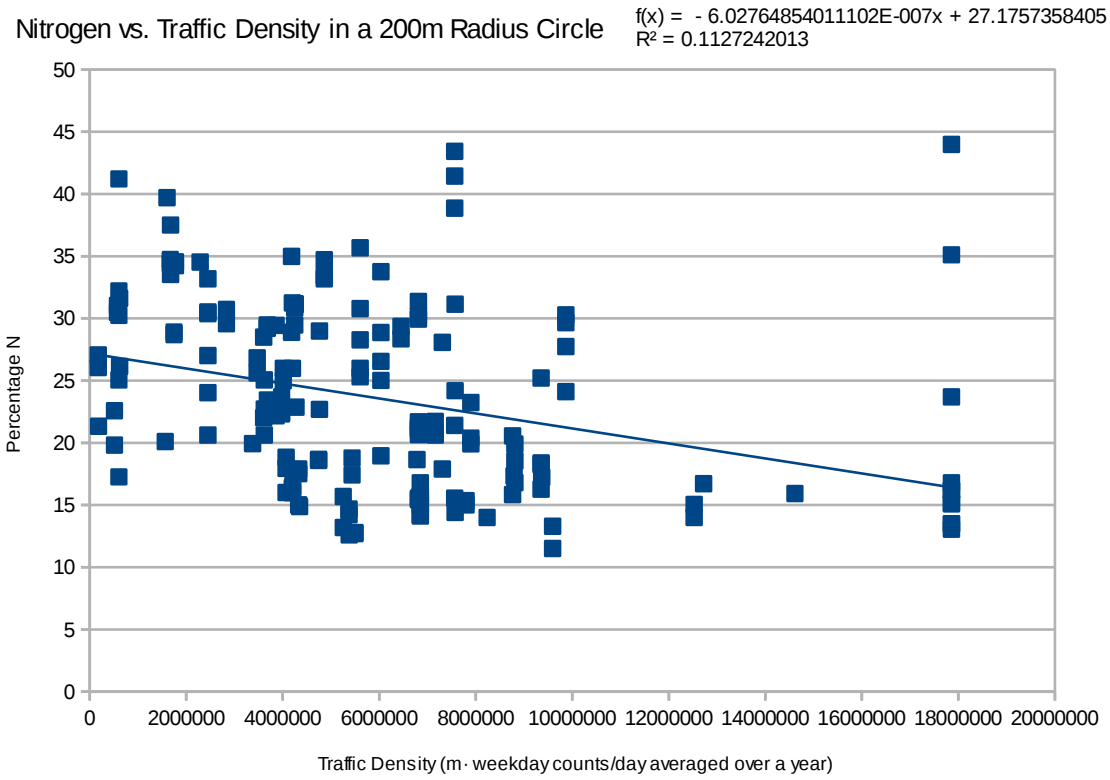


Figure 3.2: Plot of nitrogen content in moss vs. traffic counts.

H04: Linear regression is appropriate for the traffic count data + our LUR statistics

§3.0.11

The ability to derive a meaningful linear regression from a data set depends on a number of rather restrictive characteristics of the data [43]. These are neatly summarized in [46, pp. 412-413] as

1. Linearity.
2. Full rank: nonsingular predictor matrix.
3. Independent variables display exogenicity: ϵ's mean is 0 for the full range of the independent variables.
4. Homoscedasticity (constant variance of ϵ).
5. Nonautocorrelation between sample values.
6. Exogencity in the sampling: ϵ is not a product of any of the independent variables.
7. $\epsilon \sim N(0, \sigma^2)$

However, you can also produce useful linear models when there is some degree of violation of these requirements. We attempt to evaluate our data: does the data satisfy the seven requirements?

H05: Observing that the residuals of the linear regression are homoscedastic

§3.0.12

The QQ plots in the bottom panel of Figure 2.8 show a pretty linear feature over much of the data. We take this as good evidence of the homoscedasticity of the residuals after the log-transform. The graph of residuals in Figure 2.9 (b) lacks the definite linear funnel shapes of Figure 2.9 (a). We take this as further evidence that the residuals are homoscedastic after the log-transform.

§3.0.13 The linear regression results.

As discussed in section 2.8.3, we made a linear model using the well-understood least-squares regression method. Using the `lm` feature of R, we fit the data to a model of $xformkount\ rc + pop100 + pop200 + pop300 + pop500 + pop600 + pop700 + pop900 + or100 + or200 + or300 + or400 + or500$, where xformkount is log-transformed traffic counts in units of natural log of the ADT.

The residuals mapped out fairly normal, slightly skewed to the negative, with min = -5.7738, 1Q = -0.2286, Median -0.0117, 3Q = 0.2224, and Max = 3.6996. The coefficients are given in Table 3.4.

§3.0.14 H06: the linear regression is statistically significant

The regression produces an equation purporting to reproduce a linear trend in the raw data. A test exists against the null hypothesis that the slope of the regression line is actually zero. The significance factors in the coefficients Table 3.4 indicate that we have enough independent data to make a useful linear model involving fourteen road classes, seven population density radii, and five onramp mileage radii with $\alpha = 0.05$.

§3.0.15 H07: a linear regression can reproduce trends in the original traffic data

The count data held back during the creation of the linear model were plotted against the prediction of the counts made by the model. This is considered to be the ultimate test of a linear regression. The result shows a good fit, with a slope of 1.008 and an r^2 of 0.90. The Y-intercept is -0.09, indicating a slightly skewed fit. The graph in Figure 3.3 gives an overview of the result.

The fit is perhaps more convincing when seen in a map. Figure 2.6 on page 25 presents the graphical view.

Parameter	Estimate	Std. Error	t value	Pr ($> abs(t)$)	Significance Code
(Intercept)	1.108 (+01)	1.042 (-02)	1063.878	$<$ 2e-16	***
rc1200	-3.602 (-01)	1.194 (-02)	-30.172	$<$ 2e-16	***
rc1221	-2.956 (+00)	9.070 (-02)	-32.593	$<$ 2e-16	***
rc1222	-2.572 (+00)	7.996 (-02)	-32.172	$<$ 2e-16	***
rc1300	-1.672 (+00)	1.882 (-02)	-88.882	$<$ 2e-16	***
rc1400	-2.106 (+00)	1.959 (-02)	-107.472	$<$ 2e-16	***
rc1450	-2.917 (+00)	1.722 (-02)	-169.359	$<$ 2e-16	***
rc1500	-4.746 (+00)	1.463 (-02)	-324.383	$<$ 2e-16	***
rc1521	-2.152 (+00)	3.573 (-02)	-60.231	$<$ 2e-16	***
rc1700	-3.340 (+00)	1.467 (-01)	-22.768	$<$ 2e-16	***
rc1800	-4.407 (+00)	2.144 (-01)	-20.555	$<$ 2e-16	***
rc2000	-7.318 (+00)	5.667 (-01)	-12.912	$<$ 2e-16	***
rc5301	-2.216 (+00)	4.027 (-02)	-55.028	$<$ 2e-16	***
rc5401	-3.004 (+00)	9.267 (-02)	-32.420	$<$ 2e-16	***
rc5501	-3.728 (+00)	5.544 (-02)	-67.239	$<$ 2e-16	***
pop100	-1.268 (-02)	1.404 (-03)	-9.033	$<$ 2e-16	***
pop200	4.125 (-03)	8.723 (-04)	4.729	2.27 (-06)	***
pop300	-1.308 (-03)	4.104 (-04)	-3.188	0.001436	**
pop500	1.862 (-03)	2.774 (-04)	6.711	1.98 (-11)	***
pop600	-1.008 (-03)	3.398 (-04)	-2.966	0.003018	**
pop700	-3.697 (-04)	1.923 (-04)	-1.923	0.054517	.
pop900	2.519 (-04)	3.194 (-05)	7.888	3.20 (-15)	***
or100	-2.705 (-04)	6.146 (-05)	-4.402	1.08 (-05)	***
or200	-1.834 (-04)	4.744 (-05)	-3.867	0.000111	***
or300	6.071 (-04)	6.007 (-05)	10.107	$<$ 2e-16	***
or400	-4.600 (-04)	5.344 (-05)	-8.607	$<$ 2e-16	***
or500	2.567 (-04)	2.349 (-05)	10.925	$<$ 2e-16	***

Table 3.4: Coefficients from the linear regression. The units are in the natural log of the ADT. The significance codes are 0 '***' 0.001 '**' 0.01 '*' 0.05 '.'. The multiple R^2 is 0.8904.

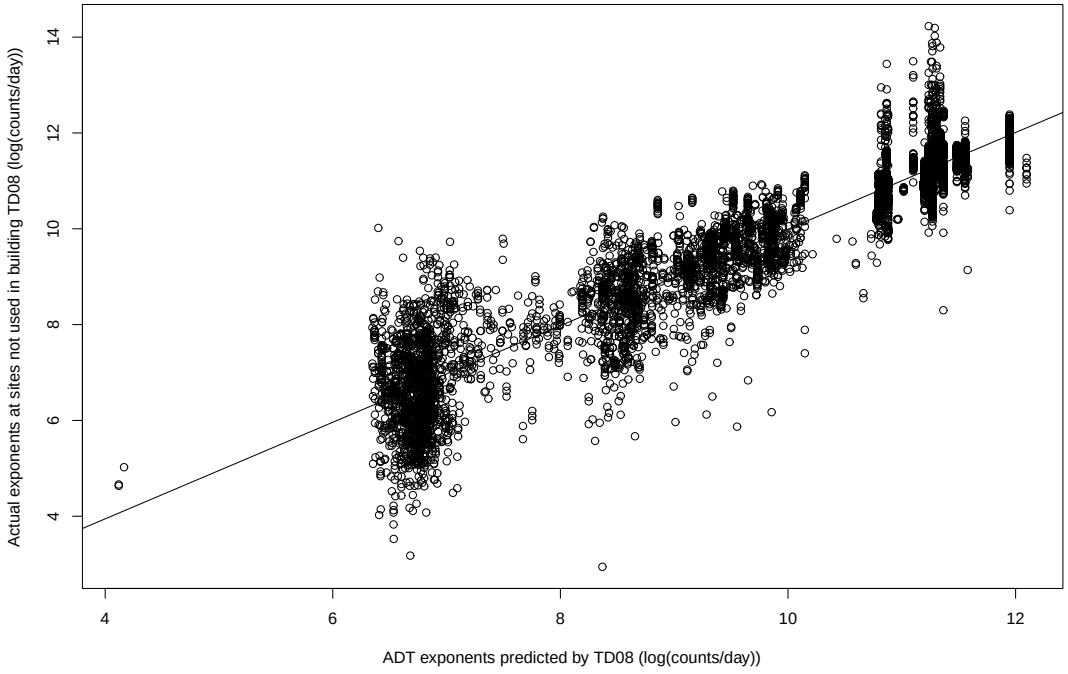


Figure 3.3: The actual count data in the set held back during model creation is plotted against the values for these counts predicted by the linear model. The equation of the line is y = -0.08613 + 1.00799x. The r^2 is 0.90. 11,463 data points are plotted.

§3.0.16 H08: there are few outliers in the cleaned data.

Our goal was to accept only counting that was as clean as possible. Any aberration was taken as grounds to throw out the count. We believe that this succeeded and the product is quite clean. To test this hypothesis, we used a statistical test to search for outliers.

We used Cook's Distance as described in [55] to determine data that exert extreme leverage in the regression. The result is given in Figure 3.4. Metrics like Cook's Distance are best interpreted in the context of the distribution of the data, and best used in a graphical setting as we have done here instead of as a strict cutoff point. We are pleased to see in the figure that very few of our large number of records exert any particularly strong leverage on the regression coefficients.

§3.0.17 H09: Rush hours will produce a bimodal trend in the diel, and weekends will be highly attenuated in counts.

We graphed the counts to search for this expected result. The results are shown in Figure 3.5. We note that the graph of Figure 3.5 would be misleading if the data included any days that did not include a full 24 hours of counting. In data cleaning we eliminated any counts that were not part of a full, clean 24 hours of counting so we can correctly interpret the graphical evidence of both the weekend effect and the bimodal diel cycle in Figure 3.5 without any misleading bias. The code for producing Figure 3.5 is in Appendix A.6.8.

We used a Welch Two Sample t-test to determine the probability that the weekend effect is present. We found a significant ($p < 0.0001$, t = -12.4, df = 1360) and large (23 %) drop off in traffic on the weekends for primary arterials. A significant ($p < 0.001$, t = -27, df = 37428) and large (22 %) drop in traffic on the weekends is seen for a pooled mean of all of the road classes. The evidence

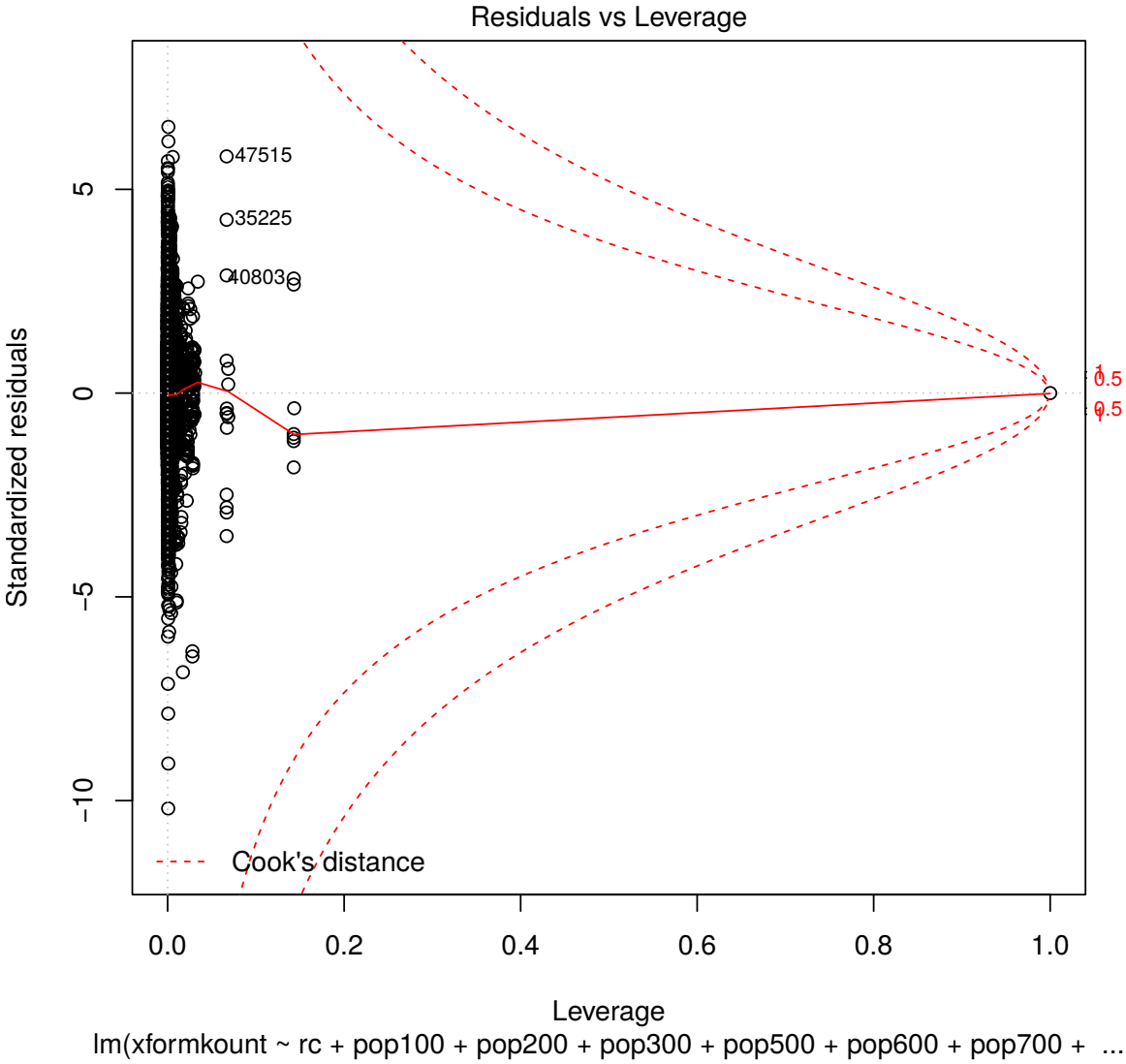


Figure 3.4: Leverage of the points used in the regression, with Cook's Distance marked.

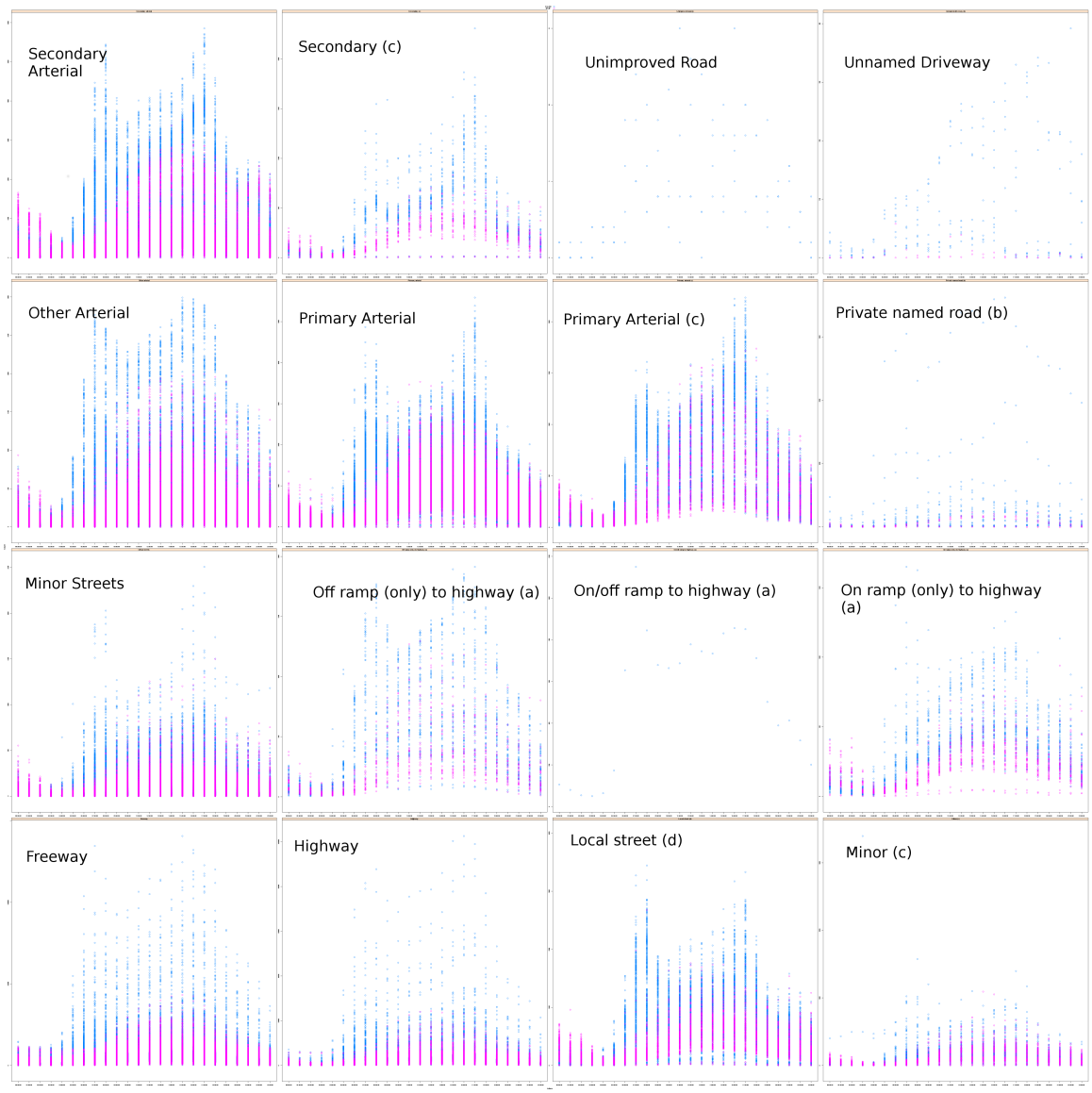


Figure 3.5: A plot showing the diurnal (diel) trends in the data. Each panel graphs all of the available data, as counts per day vs. the time of day (00:00-23:00). The blue colored dots are a single datum for each hourly binned count made on a week day, red are the same but for counts made on weekends only. (a) = "In the area maintained by the City of Portland only". (b) = "Private right-of-way exists". (c) = "With rapid transit". (d) = "To local street connector (Portland only, subarea = 'P')". (e) = "Named, but without valid addressing (Clackamas County only, subarea = 'C')". (f) = "Unnamed and without valid addressing (Clackamas County only, subarea = 'C')". (g) = "With valid address range and street name (in the area maintained by the City of Portland only)". (h) = "With NO Valid Address Range or Street Name (in the area maintained by the City of Portland only)". (i) = "No private right-of-way exists". (j) = "Passable by emergency vehicles (e-911) only (in the area maintained by the City of Portland only)".

suggests that we can safely reject the null hypothesis that there is no difference in ADT mean between weekends and weekdays. The code for the weekend test is in Appendix A.6.3.

§3.0.18 H10: Small seasonal differences are expected.

To test for seasonal differences, we partitioned the data by season, defining winter as December, January, and February. Spring is March, April, May. Summer is June, July, and August. Fall is September, October, and November.

We tested the mean of the average of all of the data for primary arterials against the mean of the subset of the data in each season using the Welch Two Sample t-test. We found a small (7.8 %) and significant ($p < 0.001$, $t = -3.64$, $df = 776$) enhancement in traffic volume during the winter. We found a small (5.9 %) and significant ($p < 0.01$, $t = 2.69$, $df = 877$) decrease in traffic volume during the spring. We found a small (7.2 %) and significant ($p < 0.001$, $t = 3.34$, $df = 7944$) decrease in traffic volume during the summer. We found no significant difference in the mean fall traffic ($p = 0.064$, $t = -1.85$, $df = 1057$) vs. the overall mean.

§3.0.19 H11: Our EI compares well with other, independent estimates

Our estimate was compared to other published estimates. The number we used for comparison are:

- 9.1E7 metric ton (MT) CO_2 emitted (average quantity for 1986-2006 (unrestricted) and 2005-2014 (restricted) from our own Circoscope inventory calculated using local VMT and a national average EF.
- 9.8E7 MT CO_2 e of GHGs emitted in 2001 (our own work extracting the Portland region's pixels from Vulcan [5])

- 2.5E7 MT CO_2 e of GHGs emitted in 2010 [56].
- 6.1E6 MT of GHGs emitted in 2005 - Metro (Portland's regional authority) estimate. [1].

Our estimate agrees closely with Vulcan's, and remains about ten times higher than the City of Portland and Metro's estimates, even though the latter included non-CO_2 gasses. This is typical of EI work: finding the best number is difficult and this motivates the WRF ground truthing of our Future Work (Section 5).

§3.1 Limitations

Our work is limited to just one mode (on-road transport), as is seen in other works of the genre [57] [18] [54] [58]. We cannot complete an emissions inventory for CO_2 without including other important sources such as off-road transport and residential and commercial fossil fuel use.

We are limited also to tank-to-wheel emissions, so our inventory does not include well-to-tank emissions, defined as "emissions from the extraction of primary energy carriers, and their conversion, refinement and delivery" [57, p. 7609] which are important for the purposes of mitigation.

We present results related to CO_2 only, but as Nickless et. al. write [57], other species with shorter lifetimes cause global warming (NOx, ozone, contrails and cirrus clouds) and are relevant to the problem of climate change which motivates this study and must be included in the comprehensive picture to enable the creation of useful short-term plans which are themselves a necessary complement to long-term plans.

[1] We multiplied "The average Portland-area resident emitted 4.05 tons of carbon dioxide and other greenhouse gases in 2005" from (`http://www.oregonmetro.gov/news/tailpipe-emissions-portland-region-climate-smart-communities-qa` accessed 2016-10-06) by the quantity of "1,500,000 people Metro serves" from (`http://www.oregonmetro.gov/regional-leadership` accessed 2016-10-16).

We calculate AADT for comparison with [51] simply by taking the arithmetic mean of the year's traffic. This is potentially biased because the counters are preferentially taken down for maintenance on certain days of the week. A better approach is described in [59, p. 1-5], creating an average of averages.

§3.1.1 Uncertainties

Here we identify several parts of the research that introduce uncertainty.

TDAT - conditions

TDAT identifies a handful of conditions marking a count as unusual. These are recorded as "conditions" in a table called public.volcount, and include values such as "Bike Path Only" and "Tax Day at the Post Office."

These condition-marked records are not presently excluded as outliers in the analysis. These condition-marked records appear to be outliers by the descriptive text just above, and the associated count records should be excluded as outliers, except possibly the synaptic ones.

Bikes being Counted

In Portland, there is a significant biking population. It is not known at this time whether bikes are counted along with cars by the automated traffic recorder (ATR)s that the city uses and thus are included in our count.

One Way Streets

We do not notice a street that is one way. As a result our ability to predict ADT is diminished; we underpredict two-way streets and overpredict one-way.

Miscoding of Weekends

If TDAT classifies a weekend as a weekday (or vice-versa), this would indicate some confusion in the record and we could detect that and use it as a reason to treat the day's count as an outlier. This would be done in Pass 1, where other exceptType (excepttype $\neq$ "normal weekday" and excepttype $\neq$ "normal weekend") are already marked as outliers.

Multiple, Independent Counts on one Site/Day

If there are instances of multiple, independent counts of the same place on the same day, we do not catch those currently and so over count.

Autocorrelation

To avoid the problem of autocorrelation, which we discussed in Section 3.0.11, in light of the results of Sections 3.0.17 and 3.0.18 it would be more accurate to use eight different equations of the form of Equation (1.1), one for each possible combination of weekend/weekday and season.

Chapter 4 Conclusion

A new model of ADT for the North American city of Portland, OR, USA has been created using multiple linear regression against a collection of 1.1 million full days of counting at a diverse array of 7,000 regional sites over the period spanning the years 1986-2006 (unrestricted roads) and 2005-2014 (restricted roads), supplemented by "land use" statistics for population density and extent of restricted road on-ramp in the area around each count site, and by identification of the functional road class (arterial, local, etc.) of the count sites.

The model was tested by holding back one-third of the data and using the model results to estimate ADT at each of the held-back count sites. The result appears to validate the model, producing a fit with a slope of 1.008 and an r^2 of 0.90.

We analyzed each road segment in a detailed map of the city by finding a centroid for each uninterrupted stretch of road (105,178 road segments). At each centroid, our "land use" variables (population density and on-ramp mileage) were evaluated using 250 CPUs in parallel on the PSU Gaia computing cluster.

These data were used with the model to estimate ADT for each road segment. The model ADT was multiplied by the road segment length to obtain VMT. This result is expected to be generally useful in building emissions inventories related to transport, meeting a current research need for high resolution emissions inventories to complement high resolution atmospheric models and neighborhood- or street segment- level policy-making decisions.

As a preliminary foray into the use of the data, the VMT values were mul-

tiplied by a nationwide-average CO_2 EF to estimate daily CO_2 emissions from on-road traffic. A city-wide EI for CO_2 was made by summing and scaling the road segment emission estimates from the model, and the resulting sum compared to other estimates. Our result closely agrees with another, the Vulcan inventory, which is a factor of ten greater than estimates made by the City of Portland and by Metro, the regional authority.

We demonstrated the construction of a 1km x 1km gridded inventory of CO_2 emissions using the nationwide-average EF and the model. The result appear to be useful, correctly highlighting freeways as CO_2 hotspots.

We used our model to search for a correlation between high ADT and high concentrations of nitrogen sampled in moss plants, but we have not been unable to establish the correlation.

Chapter 5 Future Work

Cities have a pressing need for aid in mitigating CO_2 emissions, and the TD0N model offers a window onto the city's emissions landscape at a high spatial and temporal resolution.

In his talk at AGU in 2014, K. Gurney reported a discovery of a small area of Salt Lake City which is responsible for a large portion of the city's CO_2 emissions. We plan to search for this effect in Portland.

The high resolution of our model enables our inventory to be used with WRF, a research-grade meteorological model, to predict CO_2 concentrations anywhere in the region. We will use WRF-GHG to generate model time series at high temporal resolution and compare these to tower measurements, ground truthing TD0N. We will acquire a map of biogenic emissions (perhaps Model of Emissions of Gases and Aerosols from Nature (MEGAN)) to supply this component of the carbon flux.

Top-down methods validate bottom-up methods [60] and our work will carry out this validation step in the future, using an existing data set of CO_2 concentration recordings made by Dr. Andrew Rice's group at three locations around town, locations chosen to sample a wide range of population densities.

We will extend our comparison to include the Emission Database for Global Atmospheric Research (EDGAR) [61] EI, the Highway Performance Monitoring System (HPMS) [11, Supp. p. 16] EI, the Vulcan [5] EI, and the National Emissions Inventory (NEI) [18]. We have developed a method to create gridded EI at various resolutions, for example the 1km EI of Figure 5.2. This type of summary result

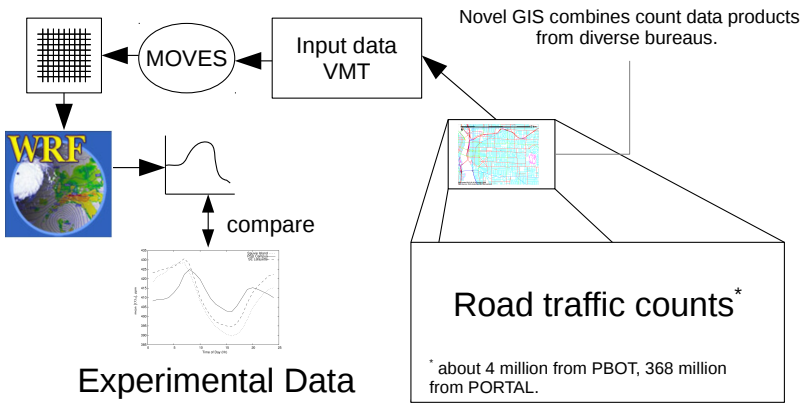


Figure 5.1: Conceptual model of the WRF components to the research.

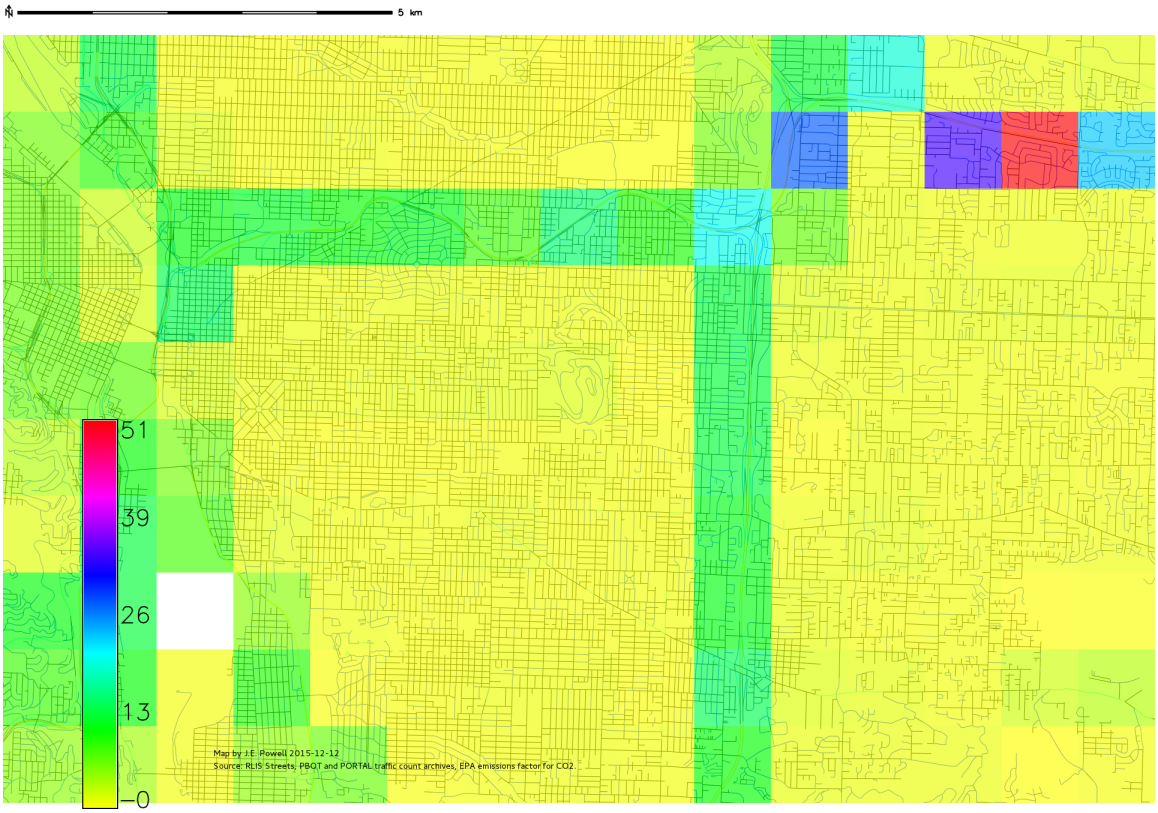


Figure 5.2: A 1 km resolution gridded CO_2 emissions inventory for Portland, OR developed using our model. We presented this result at the GEIA conference in Beijing, December 2015. The EI shows the expected high intensity of emissions along the east-west I-84 corridor (straight line, top center) and the north-south I-205 corridor (straight line, center right), zero emissions on an all-river pixel (the white pixel at bottom left), and fairly high emissions over downtown (top-left). The units are MT of CO_2/year.

makes for easy comparison with other gridded EIs and is typical of this type of research; for example [27] compare their result with EDGAR and Vulcan, and [18] compare their result with EDGAR, Vulcan, and HPMS.

We will repeat the moss N experiment, with more information than before, to find a signal.

We will continue to receive updates from PBOT and PORTAL about new traffic counts. We have completed one cycle of updates from PBOT already with no

problems, and we anticipate given our flexible data structures (CSV or SQL into an RDBMS, and then materialized views to add structure to flat data) that we can accommodate even alternative forms of the data as the archives change over time.

A novel aspect of our work is its exploration of the Portland, OR metropolitan region, which has been ranked as the "most compact" city in the USA [62]. More realistically, Portland is found to rank 80th out of 221 US cities in the "Measuring Sprawl 2014" index [63]. Portland ranks fifth the nation for light rail ridership. From 1998-2008, the average miles driven by a citizen of Portland remained flat at 20 miles a day, even while national average per-capita miles driven daily climbed steeply [1]. We expect to explore some hypotheses relating the city's lack of sprawl to the CO_2 emissions inventory, such as the possibility that CO_2 from on-road traffic should not be seen to grow linearly with the growth of the population but rather should lag somewhat behind it.

We will explore a hypothesis posed by Daniel Mendoza, who reviewed our work at AGU 2014, that Portland's Urban Growth Boundary should cause a visible effect by which population grows over time while freeway traffic holds constant. We believe other theories relating to Portland's progressive urban planning can be fruitfully explored through the high resolution TD0N model. We have concerns that super-emitters, real outliers such as diesel construction vehicles (bulldozers, cranes, dump trucks) or supercharged classic cars [2], will simultaneously not be captured by our model and yet represent a majority of emissions, a known problem for some pollutants today. We will learn whether this is true, because it will make calibration of the model with tower measurements difficult.

[1] `http://www.oregonlive.com/environment/index.ssf/2008/05/portland_reduces_its_carbon_fo.html` accessed 2016-10-06

[2] This is a hobby which is gaining in popularity in Portland. Walking home one Sunday evening in October 2016, I encountered a train of some thirty vehicles, all extremely loud, emerging from a weekly gathering. A passerby told me about it, that it's a growing movement coming up to Portland from areas such as Auburn in northern California.

Another concern is that the roughness of the urban terrain defeats the ability of WRF to model air flow over the city. This defect is being addressed, with some success, by a group at Berkeley, CA who have modified WRF to include a new module feeding the boundary outputs of very turbulent low-altitude activity into the higher (but still low-altitude) boxes of the WRF model.

Our group has already encountered a problem of this sort in doing basic WRF validation against recorded air-speed measurements: WRF simply would not run faced with the sheer altitude gains of the nearby Coast and Cascade mountain ranges.

Another concern is the very different diel patterns expected of light duty (LD) vs. heavy duty (HD) traffic, and rural vs. urban traffic as found in [27].

Our approach has also been accurately criticized as being "traditional" [3]. In an age when sensors are pervasive and capable, using the ATR devices is limiting. It is possible, for example, for a camera to film license plates, and for those plates to be looked up in the registration records to find the year, model, and fuel type of the vehicle. Additionally, it is possible today to mine cell phone location data for velocity and traffic density. More work needs to be done in this area.

c

[3]Leonor Tarrason speaking at GEIA 2015 in Beijing, China. Our EI of Figure 5.2 was projected on the big screen as an example of current work.

Bibliography

[1] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: `https://www.R-project.org`.

[2] Deepayan Sarkar. *Lattice : multivariate data visualization with R*. [London]: London : Springer, 2008,

[3] TF Stocker et al. "IPCC, 2013: climate change 2013: the physical science basis. Contribution of working group I to the fifth assessment report of the intergovernmental panel on climate change". In: (2013). URL: `http://www.ipcc.ch/pdf/assessment-report/ar5/wg1/WG1AR5_ALL_FINAL.pdf`.

[4] Juerg M Matter et al. "Rapid carbon mineralization for permanent disposal of anthropogenic carbon dioxide emissions". In: *Science (New York, N.Y.)* 352.6291 (2016), pp. 1312–.

[5] Kevin R. Gurney et al. "High Resolution Fossil Fuel Combustion CO_2 Emission Fluxes for the United States". In: *Environmental Science & Technology* 43.14 (2009), pp. 5535–5541. DOI: `10.1021/es900806c`. eprint: `http://pubs.acs.org/doi/pdf/10.1021/es900806c`.

[6] Patricia Romero-Lankao et al. "A critical knowledge pathway to low-carbon, sustainable futures: Integrated understanding of urbanization, urban areas, and carbon". In: *Earth's Future* 2.10 (2014), pp. 515–532. ISSN: 2328-4277.

[7] D. Mendoza et al. "Implications of uncertainty on regional CO_2 mitigation policies for the U.S. onroad sector based on a high-resolution emissions estimate". In: *Energy Policy, 55:386-395* (2013),

[8] James Powell, Chris L. Butenhoff, and Andrew L. Rice. *Network Level Carbon Dioxide Emissions From On-road Sources in the Portland OR, (USA) Metropolitan Area*. Poster at AGU, A53L-3370. 2014.

[9] Ian Stewart, Jack S. (Jack Sidney) Cohen, and Terry Pratchett. *The science of Discworld. 2, The globe*. Ed. by Ian Stewart and Jack S. (Jack Sidney) Cohen. London: London : Ebury, 2003.

[10] Sam Adams and Jeff Cogen, eds. *Climate Action Plan 2009*. City of Portland, 2009.

[11] Conor K Gately, Lucy R Hutyra, and Ian Sue Wing. "Cities, traffic, and CO_2: A multidecadal assessment of trends, drivers, and scaling relationships". In: *Proceedings of the National Academy of Sciences of the United States of America* 112.16 (2015), pp. 4999–. URL: http://www.pnas.org/content/112/16/4999.

[12] California Legislature. "California Global Warming Solutions Act (AB 32), Health & SC § 38500-38598". 2006. URL: http://www.arb.ca.gov/cc/docs/ab32text.pdf.

[13] Barack Obama. *White House press release. Accessed 2016-10-08*. Tech. rep. 2014. URL: https://www.whitehouse.gov/the-press-office/2014/11/11/fact-sheet-us-china-joint-announcement-climate-change-and-clean-energy-c.

[14] Kathryn McKain et al. "Assessment of ground-based atmospheric observations for verification of greenhouse gas emissions from an urban region". In: *Proceedings of the National Academy of Sciences of the United States of America* 109.22 (2012), pp. 8423–.

[15] Stephen M Wheeler. "State and municipal climate change plans: the first generation". In: *Journal of the American Planning Association* 74.4 (2008), pp. 481–496.

[16] Elizabeth C. Brodeen. "Sequestration, science, and the law: an analysis of the sequestration component of the California and northeastern states' plans to curb global warming". In: *Environmental Law* 37.4 (2007), p. 1217. ISSN: 0046-2276.

[17] Philippe Ciais et al. "The European carbon balance. Part 1: fossil fuel emissions". In: *Global Change Biology* 16.5 (2009). ISSN: 1354-1013.

[18] Conor K Gately et al. "A Bottom up Approach to On-Road CO_2 Emissions Estimates: Improved Spatial Accuracy and Applications for Regional Planning". In: *Environmental Science & Technology* 47.5 (2013), pp. 2423–2430.

[19] EPA. *U.S. Greenhouse Gas Inventory Report: 1990-2013*. Tech. rep. U.S. Government, 2015. URL: `http://www3.epa.gov/climatechange/ghgemissions/usinventoryreport.html`.

[20] Carolyn Kousky and Stephen H Schneider. "Global climate policy: will cities lead the way?" In: *Climate Policy* 3.4 (2003), pp. 359–372.

[21] Felix Creutzig et al. "Energy and environment. Transport: A roadblock to climate change mitigation?" In: *Science (New York, N.Y.)* 350.6263 (2015), pp. 911–.

[22] Ralph Sims et. al. *Intergovernmental Panel on Climate Change. Working Group III- Mitigation of Climate Change. Chapter 8 Transport*. 2014. DOI: `10.13140/2.1.4480.3521`.

[23] Lee Chapman. "Transport and climate change: a review". In: *Journal of transport geography* 15.5 (2007), pp. 354–367.

[24] Battelle. *Traffic Data Quality Measurement - Final Report*. Tech. rep. p. A-8, Table A.1. Completeness Statistics for Original Source Data. Office of Highway Policy

Information Federal Highway Administration U.S. Department of Transportation Washington, D.C., 2004.

[25] Kristin Tufte and et. al. *PORTAL*. Tech. rep. PSU, 2013.

[26] Kevin R Gurney et al. "Quantification of fossil fuel CO_2 emissions on the building/street scale for a large US City". In: *Environmental Science & Technology* 46.21 (2012), pp. 12194–12202.

[27] Brian C. Mcdonald et al. "High?resolution mapping of motor vehicle carbon dioxide emissions". In: *Journal of Geophysical Research: Atmospheres* 119.9 (2014), pp. 5283–5298. ISSN: 2169-897X.

[28] Harry Smith and Norman Richard Draper. *Applied regression analysis*. New York: New York : Wiley, 1981.

[29] Patrick H. Ryan and Grace K. LeMasters. "A Review of Land-use Regression Models for Characterizing Intraurban Air Pollution Exposure." In: *Inhalation Toxicology* 19 (2007), pp. 127–133. ISSN: 08958378. URL: `http://stats.lib.pdx.edu/proxy.php?url=http://search.ebscohost.com/login.aspx?direct=true&db=hch&AN=26641305&site=ehost-live`.

[30] Manju Mohan, Lalit Dagar, and BR Gurjar. "Preparation and validation of gridded emission inventory of criteria air pollutants and identification of emission hotspots for megacity Delhi". In: *Environmental monitoring and assessment* 130.1-3 (2007), pp. 323–339.

[31] Pathmathevan Mahadevan et al. "A satellite-based biosphere parameterization for net ecosystem CO_2 exchange: Vegetation Photosynthesis and Respiration Model (VPRM)". In: *Global Biogeochemical Cycles* 22.2 (2008). GB2005, n/a–n/a. ISSN: 1944-9224. DOI: `10.1029/2006GB002735`.

[32] Thomas Lauvaux et al. "Urban emissions of CO_2 from Davos, Switzerland: the first real-time monitoring system using an atmospheric inversion technique". In:

Journal of Applied Meteorology and Climatology (Aug. 21, 2013). ISSN: 1558-8424. DOI: 10.1175/JAMC-D-13-038.1.

[33] Society of American Archivists. *Describing archives : a content standard*. Chicago : Society of American Archivists, 2013.

[34] Kathleen D. Roe. *Arranging & Describing Archives & Manuscripts*. The Society of American Archivists, Chicago IL, 2005.

[35] Jerome S. Bruner. "The act of discovery". In: *Harvard Educational Review* 31 (1961), pp. 21–32. URL: http://hal.archives-ouvertes.fr/hal-00692072.

[36] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1998.

[37] F. Codd E. "A relational model of data for large shared data banks". In: *Communications of the ACM* 13.6 (1970). Ed. by M. Lynn, pp. 377–387. ISSN: 0001-0782.

[38] Annette J. Dobson. *An introduction to generalized linear models*. London; New York: Chapman and Hall, 1990.

[39] Herbert Friedman. *Introduction to Statistics*. Random House, Inc., New York, 1972.

[40] Robert L. Bertini Kristin A. Tufte. *PORTAL Data Quality Analysis*. Tech. rep. PSU, 2008. URL: http://www.its.pdx.edu/upload_docs/1248894239sryN2wqBix.pdf.

[41] John R. (John Robert) Taylor. *An introduction to error analysis : the study of uncertainties in physical measurements*. Mill Valley, Calif.: Mill Valley, Calif. : University Science Books, 1982,

[42] Martin B Wilk and Ram Gnanadesikan. "Probability plotting methods for the analysis for the analysis of data". In: *Biometrika* 55.1 (1968), pp. 1–17.

[43] R. Dennis Cook and Sanford Weisberg. *Residuals and influence in regression*. Ed. by Sanford Weisberg. New York: New York : Chapman and Hall, 1982.

[44] Md Shohel Reza Amin. "Application of Spatial Auto-Regressive Model for Determining Urban Land Market". In: *Journal of Bangladesh Institute of Planners* 2 (2009), pp. 107–115.

[45] William G. (William Gemmell) Cochran and George W. (George Waddel) Snedecor. *Statistical methods*. Ed. by William G. (William Gemmell) Cochran. Ames, Iowa: Ames, Iowa : Iowa State University Press, 1967.

[46] Joseph Adler. *R in a Nutshell*. O'Reilly Media, 2009.

[47] John M. Chambers and Trevor Hastie. *Statistical models in S*. Ed. by John M. Chambers and Trevor Hastie. Pacific Grove, Calif.: Pacific Grove, Calif. : Wadsworth & Brooks/Cole Advanced Books & Software, 1992.

[48] Dallas E. Johnson and George A. Milliken. *Analysis of messy data*. Ed. by Dallas E. Johnson. Belmont, Calif.: Belmont, Calif. : Lifetime Learning Publications, 1984.

[49] John I. (John Ignatius) Griffin. *Statistics : methods and applications*. New York: New York : Holt, Rinehart and Winston, 1962.

[50] Antoine Waked, Charbel Afif, and Christian Seigneur. "An atmospheric emission inventory of anthropogenic and biogenic sources for Lebanon". In: *Atmospheric Environment* 50 (2012), pp. 88–96. ISSN: 1352-2310.

[51] Christine M. Kendrick, Peter Koonce, and Linda A. George. "Diurnal and seasonal variations of NO, NO_2 and $PM_{2.5}$ mass as a function of traffic volumes alongside an urban arterial". In: *Atmospheric Environment* 122 (2015), pp. 133–141. ISSN: 1352-2310.

[52] M. B. (Maurice Bertram) Priestley. *Non-linear and non-stationary time series analysis*. San Diego: San Diego : Academic Press, 1988.

[53] Stuart Allan et al. *Atlas of Oregon*. Ed. by Stuart Allan, Aileen R Buckley, and James E Meacham. Eugene, Or., 2001.

[54] Max N. Brondfield et al. "Modeling and validation of on-road CO_2 emissions inventories at the urban regional scale". In: *Environmental Pollution* 170 (2012), pp. 113–123. ISSN: 0269-7491. DOI: 10.1016/j.envpol.2012.06.003. URL: http://www.sciencedirect.com/science/article/pii/S0269749112002783.

[55] John Fox. *Regression Diagnostics*. Newbury Park, Calif.: Newbury Park, Calif. : Sage Publications, 1991.

[56] C.O.P. *City of Portland Carbon Dioxide Reduction Strategy Update*. Tech. rep. City of Portland, Nov. 1997.

[57] Alecia Nickless, Robert J. Scholes, and Ed Filby. "Spatial and temporal disaggregation of anthropogenic CO_2 emissions from the City of Cape Town". In: (Nov. 2015). DOI: 10.17159/sajs.2015/20140387. URL: http://www.sajs.co.za/spatial-and-temporal-disaggregation-anthropogenic-co2-emissions-city-cape-town/alecia-nickless-robert-j-scholes-ed-filby.

[58] Yuqin Shu and Nina S.N. Lam. "Spatial disaggregation of carbon dioxide emissions from road traffic based on multiple linear regression model". In: *Atmospheric Environment* 45.3 (Jan. 2011), pp. 634–640. ISSN: 1352-2310. URL: http://www.sciencedirect.com/science/article/pii/S1352231010009234.

[59] *Highway Performance Monitoring System Field Manual*. Federal Highway Administration. Mar. 2014. URL: http://www.fhwa.dot.gov/policyinformation/hpms/fieldmanual/.

[60] Gregory J. Frost et al. "New Directions: GEIA's 2020 vision for better air emissions information". In: *Atmospheric Environment* 81 (Dec. 2013), pp. 710–712. ISSN: 1352-2310. URL: http://www.sciencedirect.com/science/article/pii/S1352231013006833.

[61] Johannes Gerardus Jozef Olivier et al. *Description of EDGAR Version 2.0: A set of global emission inventories of greenhouse gases and ozone-depleting substances*

for all anthropogenic and most natural sources on a per country basis and on 1 degree x 1 degree grid. 1996.

[62] R Ewing, R Pendall, and D Chen. "Measuring sprawl and its transportation impacts". In: *Travel Demand And Land Use 2003* 1831 (2003), pp. 175–183. ISSN: 0361-1981.

[63] Reid Ewing and Shima Hamidi. *Measuring Sprawl 2014*. Tech. rep. Accessed 2015-11-25. Smart Growth America, 2014. URL: `http://www.smartgrowthamerica.org/measuring-sprawl`.

[64] J. Frank Cook, Society of American Archivists. College, and University Archives Committee. *Forms manual*. Ed. by J. Frank Cook. Madison, Wisc.: Madison, Wisc., 1973.

[65] Michel Duchein. "Theoretical Principles and Practical Problems of *Respect des fonds* in Archival Science". In: (1983).

[66] Erich Gamma et al. *Design Patterns*. Addison-Wesley, Reading MA, 1995.

[67] EPA. *Using MOVES for Estimating State and Local Inventories of On-Road Greenhouse Gas Emissions and Energy Consumption*. E, 2012.

Chapter A Appendices

§A.1 Data Preprocessing Steps

Cleaning these two separate, different, and massive traffic count databases has been a great challenge. We reduced 14 GB of data - collected over many years with errors, missing data, redundant counts, and every creative possible misuse of the model system present. This has stressed the very limits of our data preparation abilities. It required years of dedicated labor. The end result is a clean, small, logical table of counts and the ability to aggregate reliable counts at the hourly or daily level, and a table of outliers (Tables 3.2 and 3.3) with reasons given for each of the problem cases.

§A.2 Data Cleaning Pipeline Flowchart

A pipeline was conceived and written out in ANSI standard flowchart code, reproduced here in Figures A.1, A.2, A.3, and A.4.

We learned through experiment that this type of engineering, taking the problem down to small easily digestible and explained steps is far superior to any attempt to combine the steps into one monolithic pass across the data.

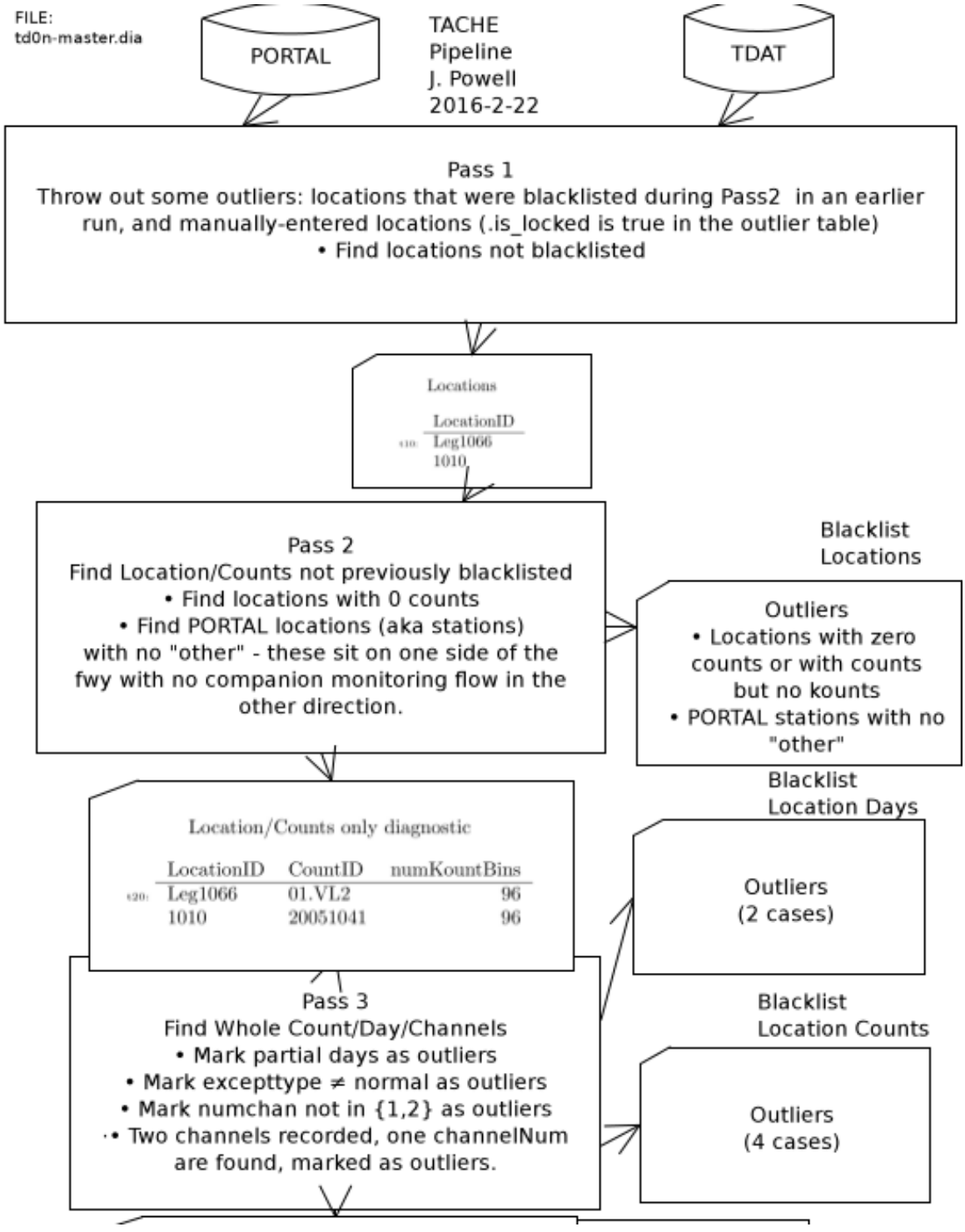


Figure A.1: TD0N pipeline, 1/4.

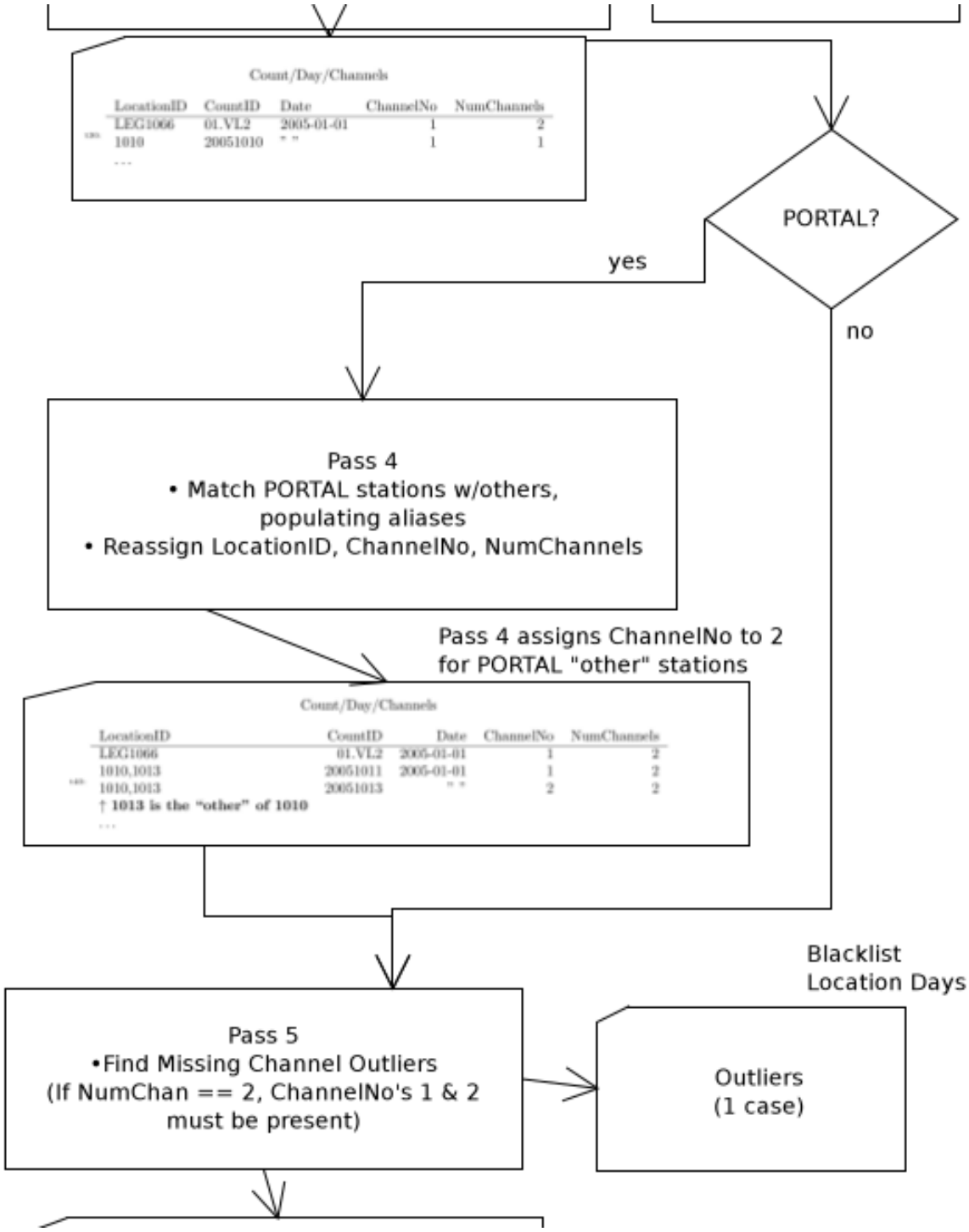


Figure A.2: TD0N pipeline, 2/4.

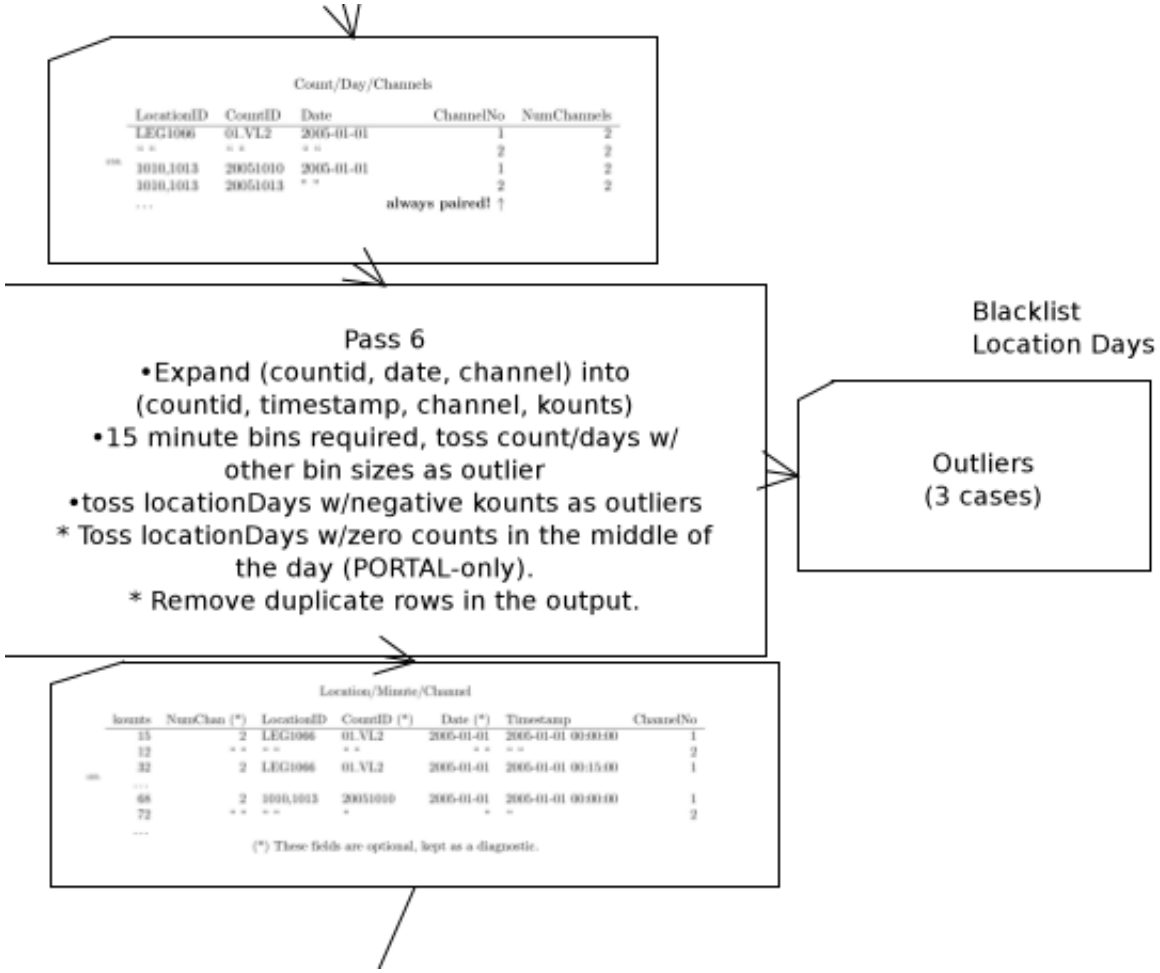


Figure A.3: TD0N pipeline, 3/4.

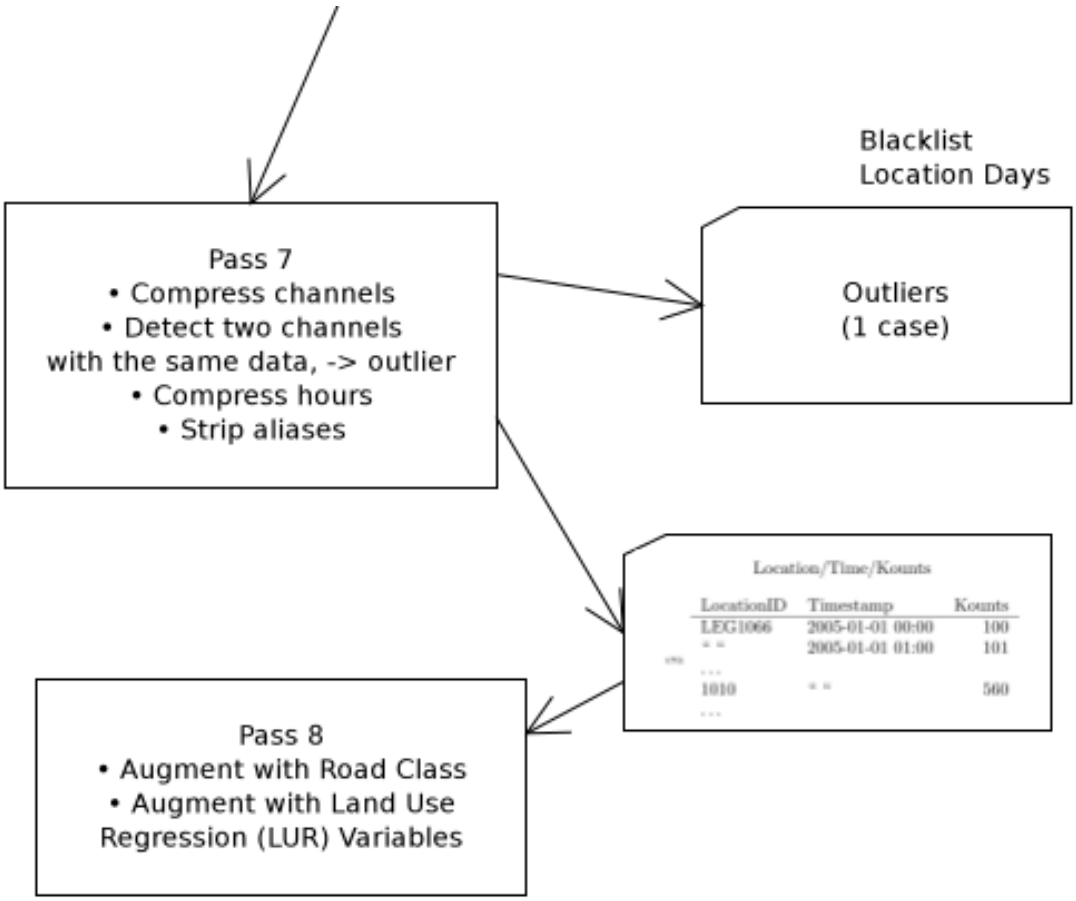


Figure A.4: TD0N pipeline, 4/4.

§A.3 Archival Process

> "Archives [. . .] include a range of format types beyond traditional paper, including [. . .] geographic information systems" [34, p. 3].

> "You're making history." (Jeffer Daykin, Capstone teacher at PSU in 2010, on the work of an archivist)

For both collections, we created accession description forms wherein our form is based primarily on the example given in [34, p. 53], with the idea of the Restrictions field, and the idea of giving our archive a distinctive name, was borrowed from the University of Wisconsin-Parkside University Archives Accession Data form in [64, p. 56].

To better document our activity and to allow some automatic interaction with the archive, an accession log [34, p. 52, Figure 4-4] is maintained in the relational database management system (RDBMS). The log is called "data_origins". It persists in the PostgreSQL database called _8666_9078_6965_namelist.

Our work commenced with the accession of a database called TDAT. This is a collection of traffic counting records made by the city of Portland (COP), almost all made with tube counters outfitted with a pneumatic hose. For the last 8-10 years, the city has been using the JAMAR brand [1] ATRs (Tom Jensen of PBOT, private communication, 2015-09-18).

TDAT was assigned for this research the catalog number 001 and is thus often referred to in this paper as Archive 001 or just Archive 1. The principle of *respect des fonds* [65] requires that we carefully isolate this archive from accidental or intentional modification, yet we will be continuously augmenting and annotating the archive by, to take one example of many, marking outliers. Our approach has been

[1] `http://jamartech.com/` accessed 2015-11-25.

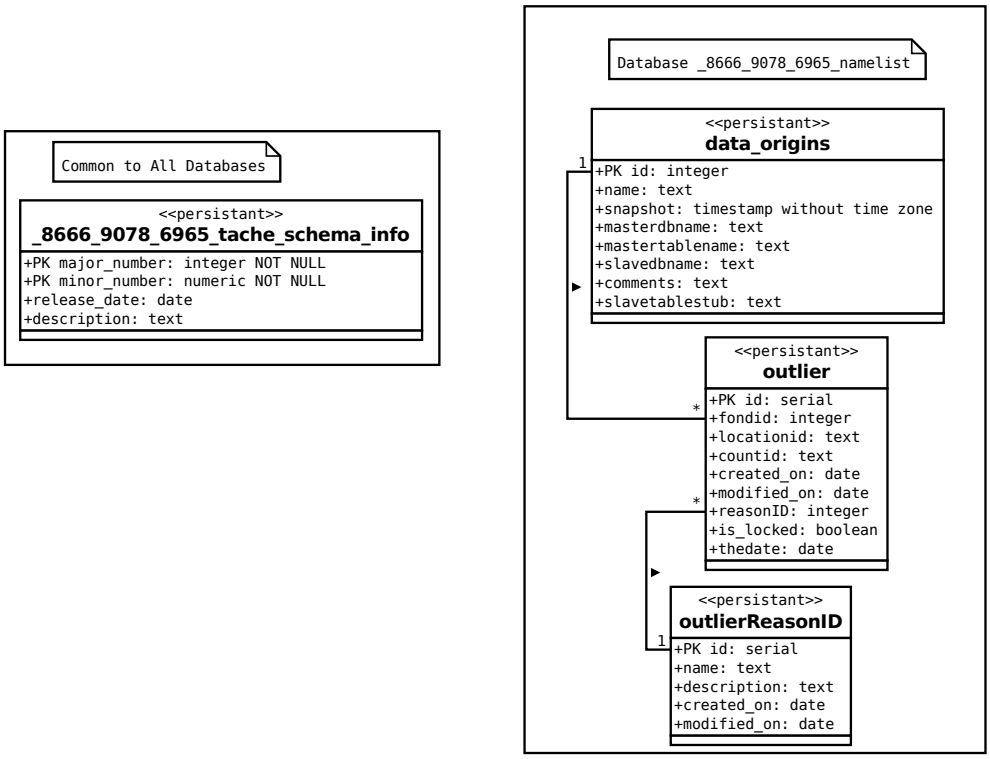


Figure A.5: A UML diagram of the namelist database. We created this finding aid to guide our manual and automated research into the archives.

to construct a number of auxiliary records that are co-resident with the archive, but clearly labeled as being foreign to the archive. That label is a sequence of three numbers emitted in sequence by a pseudo-random number generator. These numbers are 8666, 9078, 6965. These always appear together, usually as 8666_9078_6965 [2].

With our new identifier in hand, we constructed a finding aid, a catalog of the archive contents, called Database _8666_9078_6965_namelist (Figure A.5).

The accessioning process requires that a new row be added to the table called data_origins, which is unique to the entire archive (a singleton as described in [66]).

[2]This identifier is unique to the research described here and does not occur with import anywhere else in the entire cultural record of mankind as far as we are aware due to our random generation of these digits.

If an archive is listed in the data_origins table, it is part of our archive. The ID in data_origins is the archive catalog number, e.g. id=1 for TDAT as we described above. The name is something short and descriptive. Snapshot is a time-stamp that is attested and meaningfully descriptive of the temporal extent of the archive. Comments are free text but must include a citation for the value used in snapshot.

The remaining fields in data_origins guide our automatic analysis system to the named records in the archive where key indices and values are to be found.

The resulting archive we have dubbed the PSU Tarchive (traffic archive).

§A.3.1 TDAT

We solicited the TDAT count database from the city Bureau of Transportation (BOT). There was dialog within the Bureau, preserved in emails, expressing concern that transferring count data at a detailed level would be an unreasonable burden on the Bureau. It was suggested, and agreed to, that the Department would give us a copy of the entire database, which we would mine for the data at high resolution.

The database was downloaded by File Transfer Protocol (FTP) in an encoded form used by a commercial firm. Our first task was to crack the code to free the data. This task was given to the Research Computing Department at PSU, who worked on a solution for some months until the decision was made to purchasing a commercial decoder. The result is a database in industry standard SQL format.

Thus did data collection and description proceed. Exploring the data progressed through a series of email exchanges with a senior technologist at PBOT, who taught us how to find and extract counts, and a method for geocoding counts.

We built confidence in the quality of the TDAT data progressively over several year of exploration and analysis. The first potential problem was corruption in the form of the full proper names of city employees and contractors in a database

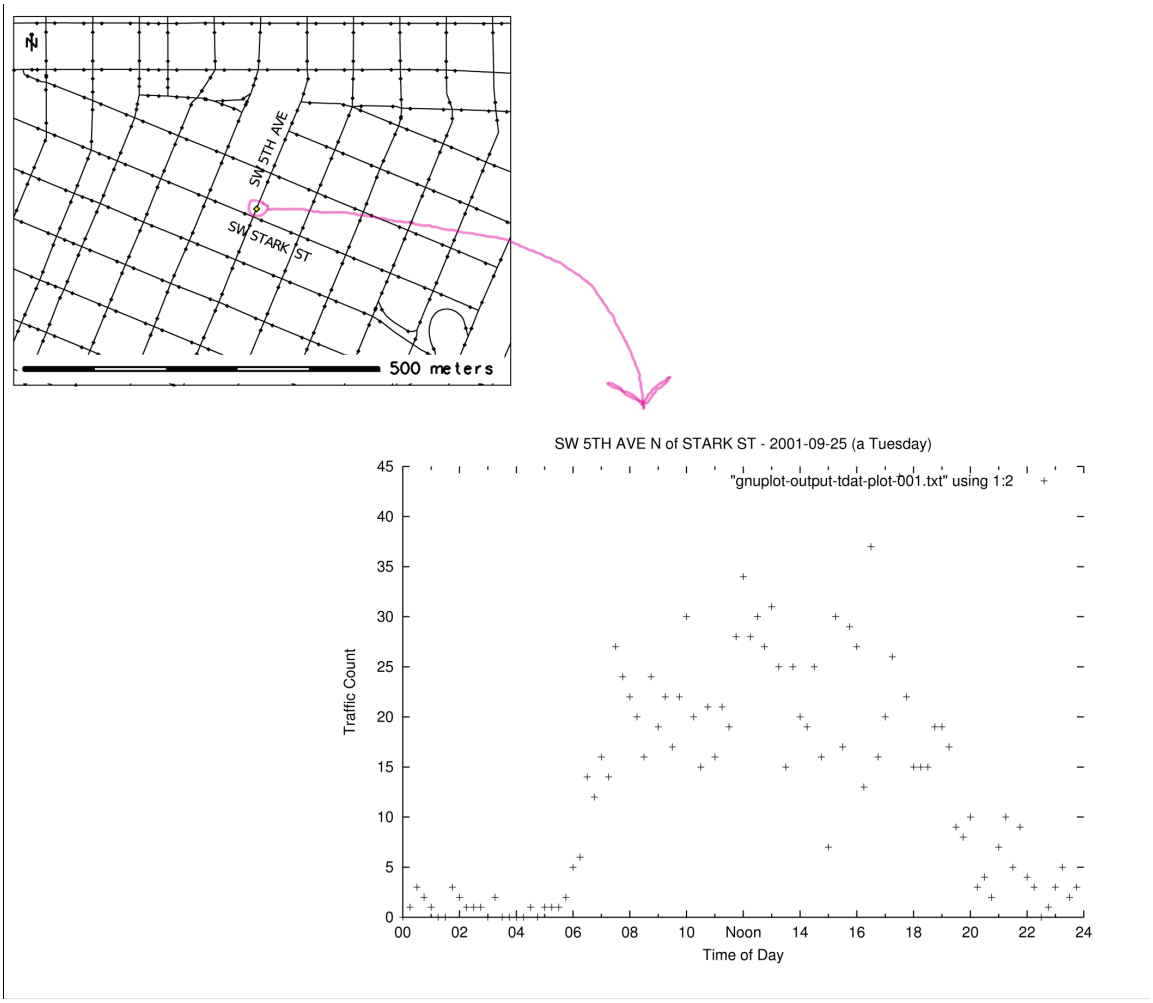


Figure A.6: A day of counting at one site in the center of the city.

slot (field) where street names are to be written. PBOT clarified that this was intentional because a location is required for each bill sent to an entity, and so when a person is the recipient, we get this.

We extracted counts for one site and graphed these in Figure A.6, and evaluated the graph for its common-sense scale and diurnal pattern.

We spent some time evaluating the structure of the counts. We produced the map of Figure A.7, a useful basic overview of the data we had obtained.

We obtained a GIS shapefile from RLIS which classifies the roads in a 32-factor

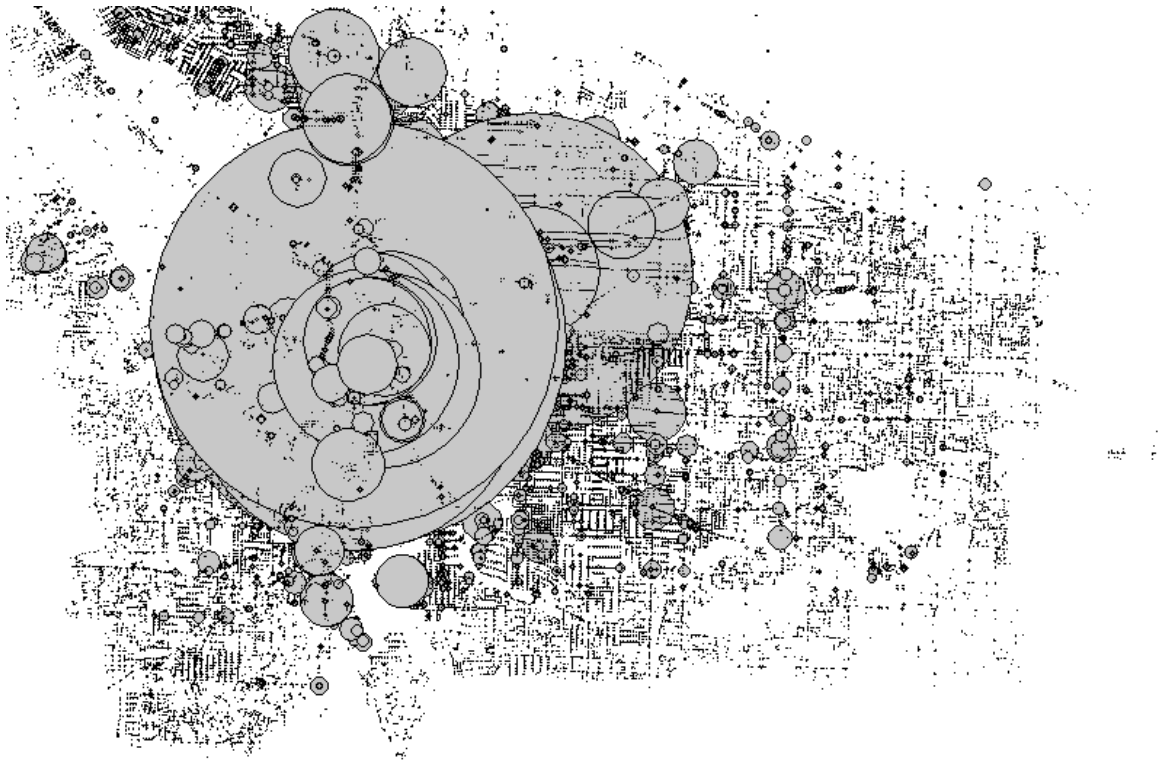


Figure A.7: Map of circles proportional to the number of times PBOT decided to open a count at the site. Note the big circles all focused on the downtown area and linear patterns of counts opened along major N/S and E/W corridors. North is up. The Willamette river can be seen as a lacuna opening in the top-left of the map. A large forested park is immediately to the west of the river at the top-left, represented as another lacuna. The remaining empty space represents the Columbia river to the north, and other jurisdictional boundaries to the east, south, and west.

type	color	
1110	red	Freeway
1121	red	On-ramp (only)
1122	black	Off-ramp (only)
1123	violet	On- and off-ramp (combination)
1200	blue	Highway
1221	purple	On ramp (only) to highway (in the area maintained by the City of Portland only)
1222	red	Off ramp (only) to highway (in the area maintained by the City of Portland only)
1300	indigo	Primary arterial
1400	green	Secondary arterial
1450	red	Other arterial
1500	cyan	Minor streets
1521	yellow	Local street to local street connector (Portland only, subarea = 'P').
1700	magenta	Private named road; private right-of-way exists
1800	gray	Unnamed driveway; private right-of-way exists
2000	brown	Unimproved road, passable by emergency vehicles (e-911) only (in the area maintained by the City of Portland only)
5301	red	Primary arterial with rapid transit
5401	orange	Secondary with rapid transit
5501	aqua	Minor with rapid transit

Table A.1: Represented road classes in the toy study area, with color codes assigned.

scheme (freeway, primary arterial, minor streets, &t). We defined a toy study area to practice on, and constructed Table A.1 to produce the map of

We built competence in mining the TDAT data through constant probing, reshuffling, comparing the results of a number of *ad-hoc* analyses that we conducted, punctuated with the formulation of questions for our domain experts at PBOT, who have always replied courteously and promptly [3].

Much of the complexity of mining the TDAT database is the result of the almost completely inverse structure of the TDAT RDBMS when compared to PORTAL. Where the latter is flat and simple making the addition of a new record just a single row in a table, TDAT involves adding several interlinked records in several

[3]Portland, OR has a tradition of strong civic involvement.

tables. Where PORTAL's counters are automatic, often neglected, and permanent installations, TDAT's counters are carefully hand-placed, watched over, and highly temporary. This leads to a great variety of corner cases which we shall document here.

§A.3.2 PORTAL

Accessioning the PORTAL process involved molding the PORTAL data dictionary to closely resemble the TDAT data dictionary. Consequently, we appropriate some terminology from TDAT. A count happens when someone decides, for any reason, to count vehicular traffic on a road. When this happens a unique (within the archive) count ID is generated and assigned to the count. For PORTAL, each station/day is assigned a unique count ID by combining the station ID and the day. A kount is a recording of vehicles that actually passed by in a given time period. A kount is always an exact integer.

We explored the published data and metadata thoroughly, and interacted via email with the PORTAL staff to obtain the information needed for our research. We learned that PORTAL has recorded traffic counts at 15 minute intervals on all of the area's freeways and highways for more than nine years.

The curated PORTAL data at RDE

We first obtained a curated and well-documented extract of the PORTAL data from the Research Data Exchange (RDE), (`https://www.its-rde.net` accessed 2014-03-20). The RDE is a US Federal Government-held database which hosts curated and documented data sets. The PSU PORTAL group put considerable time and effort into delivering a fine extract from their raw data, cleaned and documented. While the set is too small (just two months of coverage) to be useful for our project, we found this to be a useful starting point for learning how to clean

and reduce the highway and freeway data.

We sought for and obtained confirmation (K. Tufte, private email) that the RDE data set, which is identified by the name "Test Data Sets from Portland OR" with a publication date of 2012-04-16 2016-10-27) is identical to the data set known as "PORTAL FHWA" which is available from (`https://portal.its.pdx.edu/fhwa` accessed 2016-10-27).

We were able to use this valuable resource to develop our analysis while we continued to dialog with the PORTAL group about an extract of the raw data. Kristin Tufte was the principal on the PORTAL RDE (aka FHWA) data set preparation; we are not aware of the names of any other contributing parties.

In the process we built confidence in our tools and in the appropriateness of using the PORTAL data together with TDAT.

Mastering the PORTAL raw data

With the help of the PSU Research Experience for Undergrads (REU) program, who lent us Chad Sarni for two days to operate the PORTAL web site, we obtained a snapshot of the entirety of the raw count data (starting on 2005-01-01 and ending on 2014-07-27). With some searching, we obtained a complete map of the sites were counting was done.

We learned that PORTAL has both freeways and highways in it (and TDAT has highways but no freeways). The total list of freeways and highways in PORTAL is given in Table A.2. We concluded that the following roads are highways: OR 217, SR 500, US 26, SR 14 and we classified them as such for the research, leaving the rest of the roads in Table A.2 classified as freeways.

All of our GIS work was conducted using data projected into the Lambert Conformal Conic (LCC) projection, with ellipsoid grs80, selected because this is the default of a large curated base layer archive (RLIS) maintained by the regional

freeway name
I-84
I-5
I-405
I-205
WA I-5
WA I-205

Table A.2: Portal freeway coverage.

authority (Portland Metro). The freeway and highway recording station positions published by PORTAL are in lat/long coordinates. To obtain a GIS layer of the PORTAL station locations, we began with the lat/long station positions in the station list posted as part of the RDE curated set (Section 2.3.2). We required an LCC projection of these lat/long station positions for compatibility with the RLIS street map.

We used cs2cs version 4.8.0, the "cartographic coordinate system filter" (cs2cs documentation) by Frank Warmerdam and Gerald Evenden [4] to translate station positions from lat/long into LCC for compatibility with our RLIS street map.

§A.3.3 The TDAT Schema - a Finding Aid

The first step in understanding the complex TDAT database was to construct a finding aid in the form of a data dictionary. With the database in PostgreSQL after the translation work of Section 2.3.1, we were able to use the LibreOffice tool to quickly create the overview present in Figures A.8 and A.9.

§A.4 Data Cleaning Details

§A.4.1 TDAT

Q & A sessions with our PBOT experts proceeded through three distinct phases:

[4] `https://github.com/OSGeo/proj.4/wiki` accessed 2016-10-27.

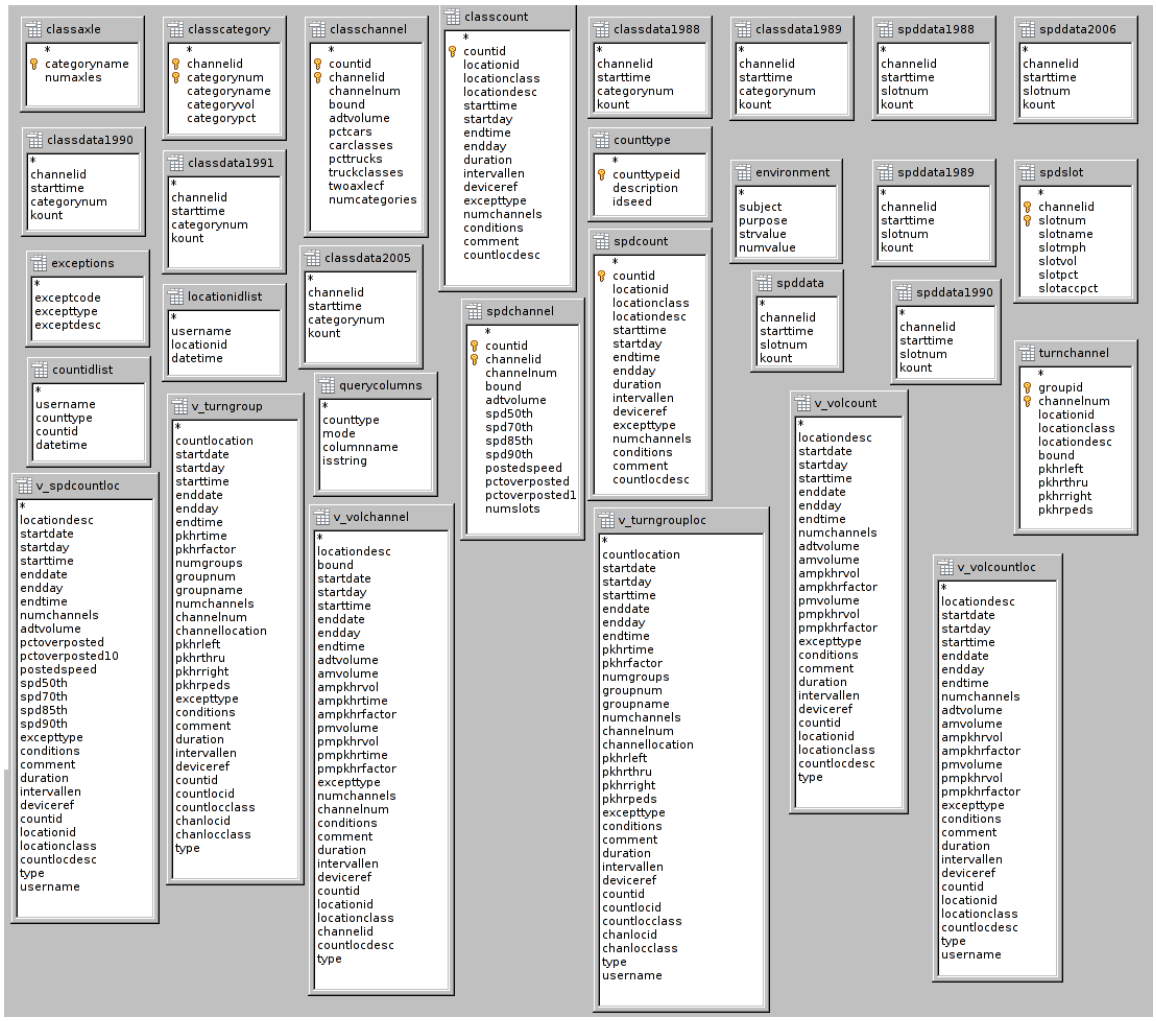


Figure A.8: The TDAT schema diagram, part 1.

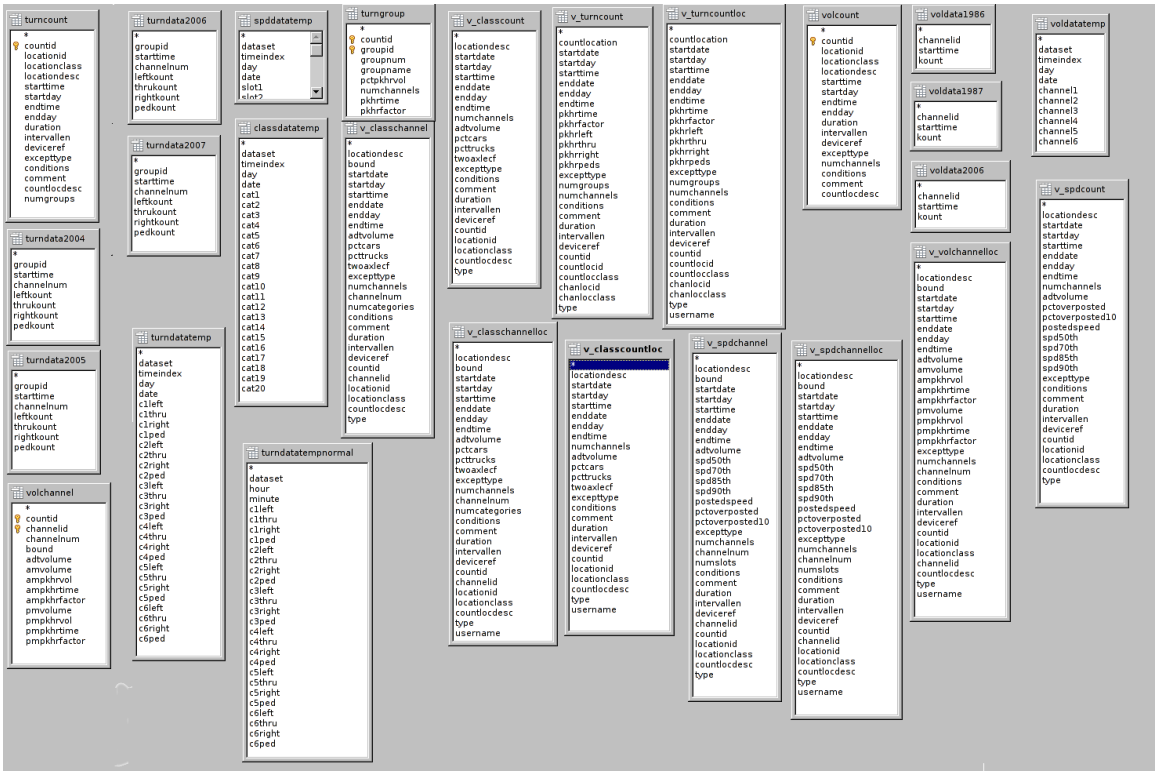


Figure A.9: The TDAT schema diagram, part 2.

1. Obtain believable, accurate counts for a given time and place.

2. Find the road functional class belonging to a count.

3. Surety that all counts that belong to a site are discovered and counted in data reduction.

Each of these phases brought new confidence in the quality of the data, although the process was not without bumps. Two in particular had us concerned. The first was the inclusion of people's names as detailed in Section 2.3.1. The second was a failure of the road class assignment of Phase 2 to properly assign a smattering of sites.

The first was explained easily as we said. The second spurred us on to a careful analysis, in early 2014, of road class assignment which we shall now discuss in some detail.

§A.5 TDAT Road Class Assignment - Technical Details

Road class in TDAT is complex. The Bureau of Transportation does not consider road functional class to have any place in the database of traffic counts: "we presently have no plans to put the federal classifications in our location model" (Jamie Throckmorton of PBOT, private communication). We identified a source of road class information provided by the regional planning authority (Metro). Identifying count sites in the BOT archive, which are point data in a GIS layer called segments, with roads in the Metro road map (RLIS streets) was a major GIS effort, but we are satisfied today that we have succeeded with this work.

The count sites are geocoded and assigned to one segment of the city's road network. The count sites are filed under the name nodeleg, which is a shapefile containing vertex geometry. The vertex attributes connect each vertex to a line

segment in the street map, which is called segments. Segments is another shapefile, containing line geometry. The attributes of segments identify the street name.

We consider compatibility of our system with EPA MOVES a priority, so we read [67] to find that MOVES requires road types "based on the HPMS classification" (ibid, p. 28). We obtained and compared a variety of candidates (ODOT, PBOT SDE, Topologically Integrated Geographic Encoding and Referencing (TIGER), HPMS, and RLIS streets).

We concluded by selecting RLIS streets, whose road functional class "codes are specific to our region" (Kelly Hauger, GIS specialist at Metro, private email), based on advice from an expert at PBOT. RLIS streets is a GIS shapefile curated by Portland's regional authority, part of the RLIS archive. It contains line geometry. The attributes identify the street name and the road class. We had to find a way to impute the road class from streets to segments.

§A.5.1 Our effort to use a spatial join fails due to great disparities between road maps

Our PBOT expert suggested a spatial join. If both maps are accurate, we reasoned, it would be quite easy to accomplish a spatial join, effectively copying the road class into segments. We had to proceed carefully, because the accuracy and usefulness of the model is a function of the accuracy and usefulness of the road functional class assignment. Thus, we spent time planning this step, exploring systems of road functional classification and alternative data sources for road class in our study area.

Our data sources selected, we attempted the spatial join. The join failed because of pervasive differences between the two vector layers (RLIS streets and TDAT segments). The two layers are evidently based on a common source put together perhaps in the 1970s, which have diverged enormously since that time. Another

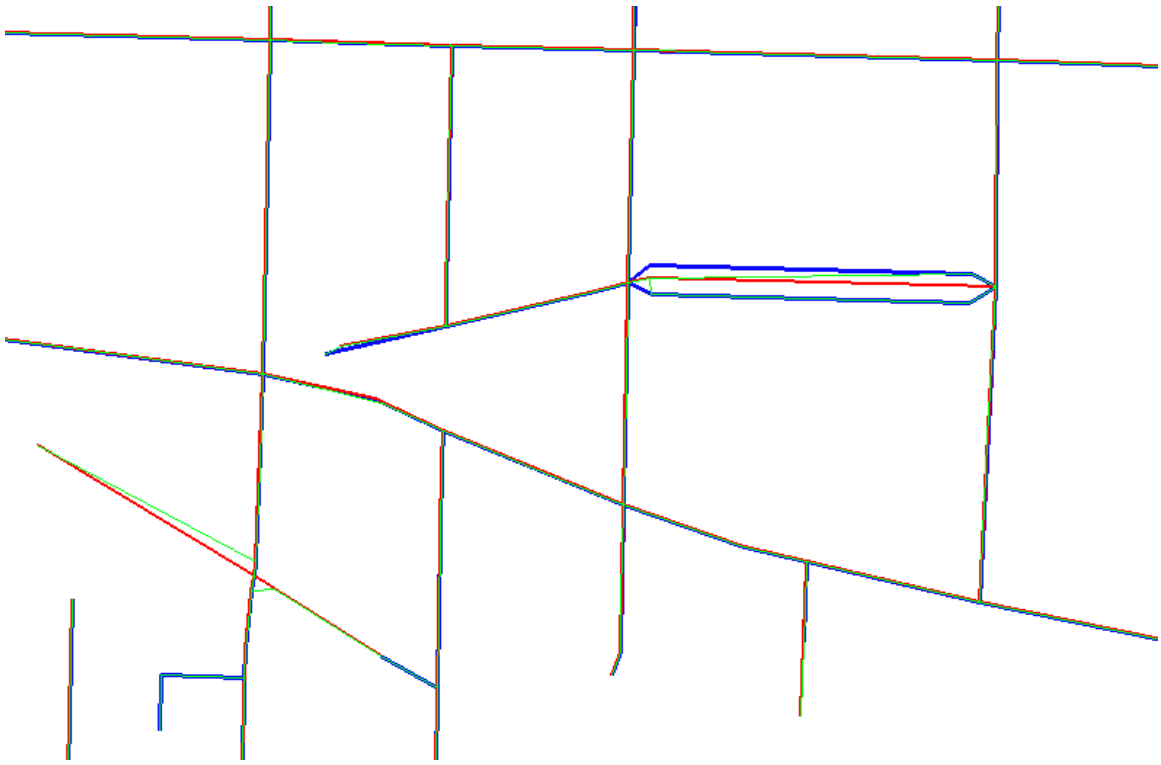


Figure A.10: An overlay map of the area around SE 29th and SE Waverleigh. North is up. Red lines are road segments from the PBOT segments. Blue lines are road segments from RLIS streets. Green lines represent the software's best attempt to spatially join RLIS streets with PBOT segments. Note the remarkable degree of divergence between the two maps, far too great for an automated system to reconcile them.

approach was needed.

One example of several we examined of the complete failure of an automated spatial join based just on the linear street segment geometry is given in Figure A.10. This type of divergence between the maps is pervasive, reflecting in the author's opinion decades of customization by two very different governmental units.

§A.5.2 A key is found that enables road class assignment

A single key identifies - often, but not always - a road segment in both layers. In TDAT the key is the last six digits of the parent ID returned by

```
psql -x -d rlis-gis -c "select * from locheir
```

```
where LocationID = 'LEG66360' and
systemid = 'STREET';"
```

. That select returns

```
: -[ RECORD 1 ]-------------
: locationid | LEG66360
: parentid   | 0040SEG138385
: systemid   | STREET
: myid       | 292816
```

The value 138385 matches the local ID in RLIS streets, as in

```
psql -x -d rlis-gis -c "select * from segstr_clean
 where localid = 138385;"
```

, which returns

```
-[ RECORD 1 ]--------------
cat        | 79928
comments   |
direction  |
fromint    |
ftype      | AVE
lcity      | PORT
lcounty    | MULT
leftadd1   | 300
leftadd2   | 398
leftzip    | 97204
length     | 262.5003299528
linkpath   |
localid    | 138385
location   |
modifiedby |
modifiedon |
modifiedus |
prefix     | SW
rcity      | PORT
rcounty    | MULT
```

```
rgtadd1    | 301
rgtadd2    | 399
rightzip   | 97204
shape_leng |
streetname | 5TH
toint      |
type       | 5501
zero       | 0
```

§A.5.3 An anomaly is found that casts doubt

The spatial join indicated how tenuous this link is (the two maps diverged significantly over time), and in a flyover we noticed an anomaly: highway-typed streets deep in a residential neighborhood. These mistyped streets were a serious concern, so we examined this case in particular and found that our analysis is unaffected because no traffic counting ever happened at these particular sites.

The flag was raised however, and to regain confidence in the road class assignment we tested and proved the theory that every counter site in TDAT *where counts actually happened* is sited within a short distance (4m) of a road segment in RLIS streets which agrees with the road class we previously found for the site by the tenuous link of Appendix A.5.2.

We found that 1.7 % of the sites do not have a road on the authoritative map (RLIS streets) with the appropriate road functional class within four meters of the count site. We are unable to say anything definite about these sites without ground truthing them, except that the remainder of the analysis came so close to perfectly matching road class via the tenuous link and road class by the type of spatial matching assigned here that they are most probably correctly assigned. Divergence in the location and dimensions in the road segments given in the two maps is great, as we have demonstrated, so it is expected that our careful analysis here, with its tight 4m radius requirement for a match, will not always find a match

even when the assignment is good.

At 98.7 % of the sites where a count actually happened, only one type of road is found. In the other 1.3 % of sites, there are more than one road classes represented in the 4m circle, but the "correct" road class is always among them. We conclude that it is highly likely that 100% of the sites where counts actually happened have been assigned the correct road class. This is not a given, because our method relies on a tenuous putative link between TDAT segments and RLISstreets, two street maps that seem to have shared a common ancestor a long time ago, but which have diverged quite thoroughly since that time (for example, many or most road segments have drifted, and many road segments have been added or deleted, to better fit the needs of the individual bureaus, which are COP PBOT and Oregon Metro - a regional planning organization).

One of the vague cases is given in Figure A.11. Due to the divergence in the segments and streets maps, either of the two roads shown could apply their road class to the site indicated by the yellow cross. The geocentric coordinate system (GCS) coordinates in RLIS streets projection are given.

This RC validation work is epitomized by the occurrence of the text `typesin` as in

#if (DIAGNOSTIC_TYPESIN)

ln << ",rc,typesin_4_m_01, ...

#endif

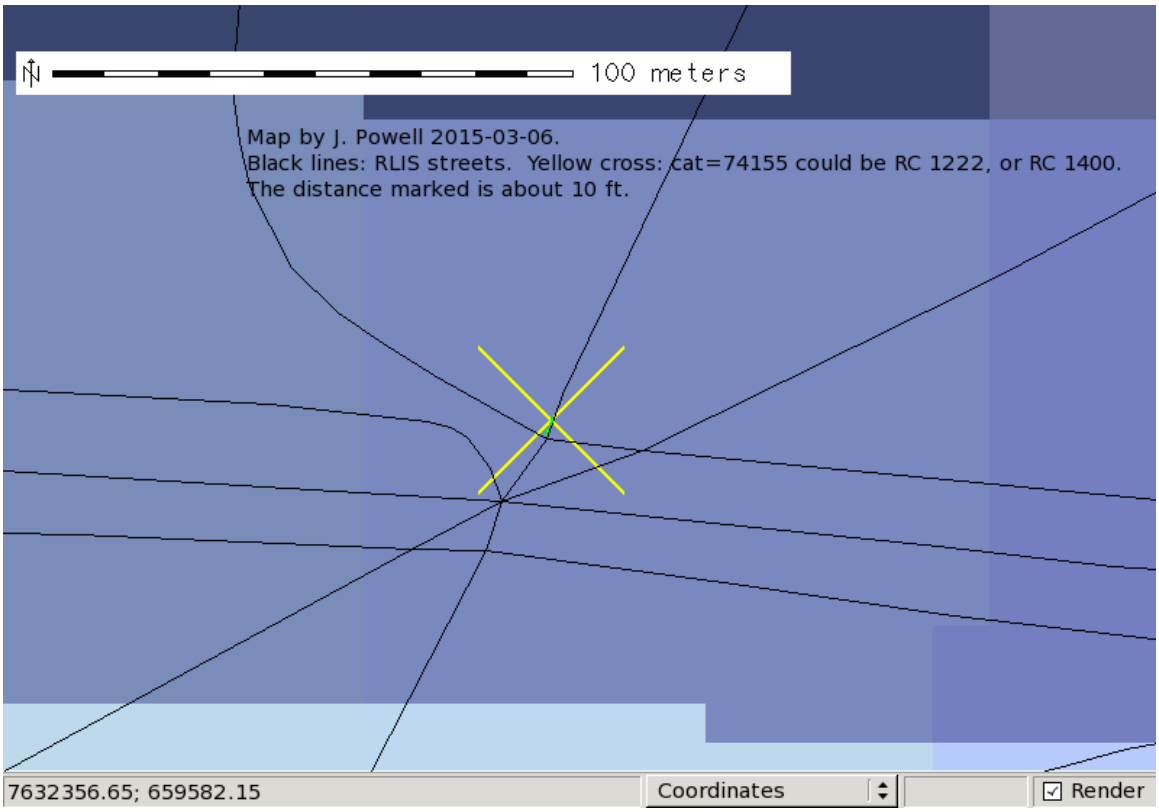


Figure A.11: Map showing one of the TDAT count sites where it is not possible without ground truthing to identify the road class of the site. RLIS streets includes two roads with different road classes in the immediate vicinity.

§A.6 Source code

§A.6.1 Tiny Toy Data For Sample Calculations

```
// Fi: gen.cc
// Au: James E. Powell
// Da: 2016-05-14
//
// Generate data similar to the TDON raw data, but a much smaller set.

#include <iostream>
#include <fstream>
#include <assert.h>
#include <iomanip>
#include <cmath>

using namespace std;

// First ten entries from (Griffin62) Table F-10
// Gaussian Deviates, p. 490.
double gd0[10]={-1.276, -1.218, -0.453,
-0.350,  0.723,  0.676,
-1.099, -0.314, -0.394,
-0.633};
double gd1[10]={-0.318, -0.799, -1.664,
+1.391,  0.382,  0.733,
+0.653,  0.219, -0.681,
+1.129};
double gd2[10]={-1.377, -1.257,  0.495,
-0.139, -0.854,  0.428,
-1.322, -0.315, -0.732,
-1.348};
double gd3[10]={+2.334, -0.337, -1.955,
-0.636, -1.318, -0.433,
+0.545,  0.428, -0.297,
+0.276};
double gd4[10]={-1.136,  0.642,  3.436,
-1.667,  0.847, -1.173,
-0.355,  0.035,  0.359,
+0.930};
double gd5[10]={ +0.414, -0.011,  0.666,
 -1.132, -0.410, -1.077,
 +0.734,  1.484, -0.340,
 +0.789};

void write_header(ofstream &fl)
{
```

```
fl << "rc1,rc2,dacount,dacount_delta,numkounts,pd1,pd2,or1,or2\n";
}

int main()
{
ofstream fl("data-gen.org");
assert(fl.good());
fl << setprecision(8);
int n = 9;
write_header(fl);
double rcmean[] = {25000., 800.};
for (int rc = 1; rc <= 2; rc++) {
for (int i = 0; i < n; i++) {
double m = rcmean[rc-1];
double pdmean = 5.*((m / 25000.) + 10.);
double ormean = 20.*((m/25000.0) + 10.);
double adt_raw_var = (rc == 1 ? gd0[i] : gd1[i]);
// mean plus natural variation
double adt = m + adt_raw_var * 50.0 * (m / rcmean[1]);
doublt adt_delta_raw_var = (rc == 1 ? gd0[i] : gd1[i]);
double adt_delta = 100 * fabs(adt_delta_raw_var);
// delta is bigger as var is bigger
adt_delta = pow(adt_delta*(m / (rcmean[1] / 100.0)),1.);
// pd1 is closely correlated with adt through gd0, with a
// little of its own natural variation through gd1.
double pdvar = gd0[i] * 10.0;
double pd1 = pdmean + pdvar + gd1[i];
// pd2 also is closely correlated with adt through gd0
      // (pdvar), with its own natural variation.  It's also
      // twice as large as pd1.
double pd2 = pdmean * 2.0 + pdvar * 2.0 + gd2[i];
// or1 is closely correlated with adt through gd0 (orvar),
// with its own natural variation through gd3.  It's three
// times bigger than the pds.
double orvar = gd0[i] * 30.0;
double or1 = ormean + orvar + gd3[i] * 3.0;
// or2 is twice as big as or1, but also closely correlated
      // with adt through gd0.
double or2 = ormean * 2.0 + orvar * 2.0 + gd4[i] * 6.0;
// rc1,rc2,adt,pd1,pd2,or1,or2
fl << (rc==1 ? 1 : 0) << "," << (rc==2 ? 1 : 0) << "," <<
         adt << ",";
fl << adt_delta << ",";
fl << 1 << ",";
fl << pd1 << "," << pd2 << "," << or1 << "," << or2 << endl;
}
}
}
```

§A.6.2 Adapt PORTAL to the TDAT data structure

We had a need for PORTAL data to appear, to the SQL interface, much like the much more hierarchically organized TDAT data. We accomplished this by creating the materialized views, using the SQL given here.

```
-- file: adapt-foreign-database-version-portal-total-MATERIALIZED.sql
-- au: James E. Powell
-- date: 2016-08-04
-- c.f. ~/projects/tache/milestones/milestone-04-03-2014 (SQL to
-- make materialized views for PORTAL-TOTAL)/
-- based on adapt-foreign-database-version-portal-fhwa-
--   MATERIALIZED.sql

\timing

-- Dropping the views should delete the indices also, "DROP TABLE
-- always removes any indexes"
-- (file:///usr/share/doc/postgresql-doc-9.4/html/sql-droptable.html)

drop materialized view nodeleg;
drop materialized view volchannel;
drop materialized view volcount;
drop materialized view voldata9999;
-- cascade is req'd for sf_ld15m cos 'view volcount_v depends on
-- materialized view sumfreeway_loopdata15min'
drop materialized view sumfreeway_loopdata15min cascade;

-- Sometimes I like to change the type of columns, and CREATE
--   OR REPLACE
-- doesn't work (see 0E3: view build fails changing data type,
-- https://stackoverflow.com/questions/36790981/
--   cannot-change-data-type-of-view-column-sql)
-- when that happens unless you utterly delete the views first.
drop view sumfreeway_loopdata15min_v;
drop view nodeleg_v;
drop view volchannel_v;
drop view volcount_v;
drop view voldata9999_v;

-- View sumfreeway_loopdata15min
-- based on "View: volchannel_v" (below)
-- See tache issue "0E1: groups of three..." for discussion.
CREATE OR REPLACE VIEW sumfreeway_loopdata15min_v AS
select starttime,sum(volume) as sumvol,highwayid,stationid from
  freeway_loopdata15min group by starttime,highwayid,stationid;
```

```
create materialized view sumfreeway_loopdata15min as
  select * from sumfreeway_loopdata15min_v;

-- View: nodeleg_v

-- DROP VIEW nodeleg_v;

CREATE OR REPLACE VIEW nodeleg_v AS
 SELECT DISTINCT portal_total_freewy_stns.stationid AS location,
    portal_total_freewy_stns.cat
   FROM portal_total_freewy_stns;

-- Materialized View: nodeleg

-- DROP MATERIALIZED VIEW nodeleg;

-- this works too (for all of the tables):
-- create materialized view nodeleg as select * from nodeleg_v;
CREATE MATERIALIZED VIEW nodeleg AS
 SELECT nodeleg_v.location,
    nodeleg_v.cat
   FROM nodeleg_v
WITH DATA;

-- View: volchannel_v

-- DROP VIEW volchannel_v;

CREATE OR REPLACE VIEW volchannel_v AS
 WITH honker AS (
         SELECT fd.stationid,
            ld.starttime::timestamp without time zone AS starttime,
            date_trunc('day'::text,
                       ld.starttime::timestamp without time zone)
                       AS truncstamp,
            date_part('dow'::text,
                      date_trunc('day'::text,
                                 ld.starttime::timestamp
                                    without time zone))
                      AS dow,
            ld.sumvol AS kount,
            row_number() OVER (ORDER BY fd.stationid, starttime)
              AS channelid
           FROM portal_total_freewy_stns fd
             JOIN sumfreeway_loopdata15min ld
               ON fd.stationid = ld.stationid
        )
 SELECT DISTINCT concat(honker.truncstamp, honker.stationid)
    AS countid,
```

```
    honker.channelid,
    1 AS channelnum
   FROM honker;

-- Materialized View: volchannel

-- DROP MATERIALIZED VIEW volchannel;

CREATE MATERIALIZED VIEW volchannel AS
 SELECT volchannel_v.countid,
    volchannel_v.channelid,
    volchannel_v.channelnum
   FROM volchannel_v
WITH DATA;

-- View: volcount_v

-- DROP VIEW volcount_v;

CREATE OR REPLACE VIEW volcount_v AS
 WITH honker AS (
         SELECT fd.stationid,
            ld.starttime::timestamp without time zone AS starttime,
            date_trunc('day'::text,
                       ld.starttime::timestamp without time zone)
                       AS truncstamp,
            date_part('dow'::text,
                      date_trunc('day'::text,
                                 ld.starttime::timestamp
                                   without time zone))
                       AS dow
           FROM portal_total_freewy_stns fd
             JOIN sumfreeway_loopdata15min ld
               ON fd.stationid = ld.stationid
        )
 SELECT DISTINCT honker.stationid AS locationid,
    honker.truncstamp AS starttime,
    concat(honker.truncstamp, honker.stationid) AS countid,
    15 AS intervallen,
        CASE
            WHEN honker.dow = 0::double precision
              THEN 'Normal Weekend'::text
            WHEN honker.dow = 1::double precision
              THEN 'Normal Weekday'::text
            WHEN honker.dow = 2::double precision
              THEN 'Normal Weekday'::text
            WHEN honker.dow = 3::double precision
              THEN 'Normal Weekday'::text
            WHEN honker.dow = 4::double precision
              THEN 'Normal Weekday'::text
```

```
            WHEN honker.dow = 5::double precision
              THEN 'Normal Weekday'::text
            WHEN honker.dow = 6::double precision
              THEN 'Normal Weekend'::text
            ELSE NULL::text
        END AS excepttype
   FROM honker;

-- Materialized View: volcount

-- DROP MATERIALIZED VIEW volcount;

CREATE MATERIALIZED VIEW volcount AS
 SELECT volcount_v.locationid,
    volcount_v.starttime,
    volcount_v.countid,
    volcount_v.intervallen,
    volcount_v.excepttype
   FROM volcount_v
WITH DATA;

-- View: voldata9999_v

-- DROP VIEW voldata9999_v;

CREATE OR REPLACE VIEW voldata9999_v AS
 WITH honker AS (
         SELECT fd.stationid,
            ld.starttime::timestamp without time zone AS starttime,
            date_trunc('day'::text,
                       ld.starttime::timestamp without time zone)
                       AS truncstamp,
            date_part('dow'::text, date_trunc('day'::text,
                      ld.starttime::timestamp without time zone))
                       AS dow,
            ld.sumvol AS kount,
            row_number() OVER (ORDER BY fd.stationid, starttime)
              AS channelid
           FROM portal_total_freewy_stns fd
             JOIN sumfreeway_loopdata15min ld
             ON fd.stationid = ld.stationid
        )
 SELECT DISTINCT honker.channelid,
    honker.starttime,
    honker.kount
   FROM honker;

-- Materialized View: voldata9999

-- DROP MATERIALIZED VIEW voldata9999;
```

```
CREATE MATERIALIZED VIEW voldata9999 AS
 SELECT voldata9999_v.channelid,
    voldata9999_v.starttime,
    voldata9999_v.kount
   FROM voldata9999_v
WITH DATA;

-- make sure they are up to date
REFRESH MATERIALIZED VIEW nodeleg;
REFRESH MATERIALIZED VIEW volchannel;
REFRESH MATERIALIZED VIEW volcount;
REFRESH MATERIALIZED VIEW voldata9999;

create index volc_chanid_index on volchannel (channelid);
-- => \timing
-- portal-total=> create index vold_chanid_index on
--   voldata9999 (channelid);
-- CREATE INDEX
-- Temps : 54856.375 ms
-- This particular index (voldata9999, channelid) speeds STAC calls
-- up by something like 3000 times.
-- They go from 5s each call (too slow for my purposes) to
--   flying by faster than I can see it.
-- That, folks, is why we create indices.
create index vold_chanid_index on voldata9999 (channelid);
create index volc_countid on volchannel (countid);
```

§A.6.3 Weekend Test

```
## File: weekend-ttest.r

## Au: J. E. Powell

## Da: 2016-10-10

##source("xform.r")

x<-read.table(

  "td08-after-subset-selection-and-rename-2016-10-04.csv",

  header=TRUE,sep=",")

lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")

x$weekendp <- as.factor(weekdays(

   as.Date(as.character(x$tsdate),"%Y-%m-%d")) %in% c('Sunday',

   'Saturday'))

z <- subset(x, x$rc==1300)
```

```
## z <- x

z1 <- subset(z, weekendp==TRUE)

z2 <- subset(z, weekendp==FALSE)

t.test(x=z1$kount, y=z2$kount)
```

Diagnostic Plots for the Log Transformation.

§A.6.4 Shapiro-Wilks.

```
## Fi: lognormdiag.r
## Au: J.E. Powell
## Da: 2016-10-06
## De:
##
## We need to justify log-transforming kounts in the lm().
## We do that here by showing that in a sampling of RC, the
## data become normal on a log-transform.
library(lattice)
source("transrc.r")
Graphs <- function(x) {
    pdf("hist-lognormdiag-xform-2016-10-06.pdf")
    trellis.par.set(fontsize=list(text=7))
    histogram(~ xformkount|as.factor(rc),data=x,auto.key=TRUE,
              scales="free")
    dev.off()
    pdf("hist-lognormdiag-no-xform-2016-10-06.pdf")
    trellis.par.set(fontsize=list(text=7))
    histogram(~ kount|as.factor(rc),data=x,auto.key=TRUE,
      scales="free")
    dev.off()
    pdf("qq-lognormdiag-no-xform-2016-10-06.pdf")
    trellis.par.set(fontsize=list(text=7))
    qqmath(~ kount|as.factor(rc),data=x,auto.key=TRUE,
      scales="free")
    dev.off()
    pdf("qq-lognormdiag-xform-2016-10-06.pdf")
    trellis.par.set(fontsize=list(text=7))
    qqmath(~ xformkount|as.factor(rc),data=x,auto.key=TRUE,
      scales="free")
    dev.off()
}
xform <- function(x) {
  if (TRUE) {
      ## remove 10 outliers, see log entry for
      ## Wed Sep 21 20:11:19 2016
      x <- x[x$kount != 0,]
```

```
      ## log-transform.
      ## see Wed Sep 21 20:23:38 2016 :log:transform:
      x$xformkount <- log(x$kount)
  }
  else {
    x$xformkount <- x$kount
  }
  x
}
filename <-
  "td08-after-subset-selection-and-rename-2016-10-04.csv"
## use only 5000 records cos the full set is unwieldy to
## render the QQ
filename <- "td08-after-subset-selection-5000-2016-10-04.csv"
x <- read.table(filename, header=TRUE, sep=",")
x$rc <- apply(as.matrix(x$rc), 1, transrc)
lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
x$weekendp <- weekdays(as.Date(as.character(x$tsdate),
  "%Y-%m-%d")) %in% c('Sunday','Saturday')
x <- xform(x)
x <- x[x$weekendp == FALSE,]
if (FALSE) {
    Graphs(x)
}
## s/w
filename <- "td08-after-subset-selection-and-rename-2016-10-04.csv"
x <- read.table(filename, header=TRUE, sep=",")
x <- xform(x)
art <- x[x$rc == 1300,]
fwy <- x[x$rc == 1110,]
shapiro.test(art$kount)
shapiro.test(art$xformkount)
shapiro.test(sample(fwy$kount, 5000, replace=FALSE))
shapiro.test(sample(fwy$xformkount, 5000, replace=FALSE))
```

§A.6.5 Join the raw counts with our pseudo-LUR variables.

This task (pass 8 in the pipeline) is made easy by R's merge function. It's fast, too (1 minute). The input, `lur-min.csv`, is easily extracted from the GIS layer.

```
## Fi: join.r
## Au: J.E. Powell
## Da: 2016-08-15
##
## desc:
## based on -
```

```
## [[file:~/projects/tache/milestones/milestone-04-03-2015%20
## (TD08%20statistically%20defensible%20traffic%20density%20
## model%20with%20freeways)/join.r][file:~/projects/tache/
## milestones/milestone-04-03-2015
## (TD08 statistically defensible traffic density model with
## freeways)/join.r]] td008-input.csv comes out without any
## LUR info adorning it.  Use R merge to pull the lur variables.
## use "make td08-with-lur.csv" to run this.
lur <- read.table("lur-min.csv", header=TRUE, sep=",")
a <- read.table("rlis-gis-vec_repo_001_rconly.txt", header=TRUE,
       sep=",")
fname <-
  "data-from-mercredi 14 septembre 2016, 19:05:09 (UTC-0700).txt"
x <- read.table(fname, header=TRUE, sep=",")
z <- merge(lur,x)
z <- merge(z, a)
## sort before write
z <- z[with(z, order(rc, locid, tsdate, tstime)), ]
## write
write.csv(z,"td08-with-lur-2016-09-14.csv",row.names=FALSE)
```

§A.6.6 Reduce the data to one row for each day.

This computation takes a long time (about 8 hours). There is surely a better and fast way to do it in R, because using `rbind` is almost always a sign that you've broken the R paradigm.

```
## File: reduce.r
## Au: J.E. Powell
## Da: 2016-09-18
# based on file:~/projects/tache/milestones/milestone-2015-03-03-
#   (TD007 statistically defensible traffic density model with
#   freeways)/reduce.r]]
# http://stackoverflow.com/questions/2676554/in-r-how-to-find-
#   the-standard-error-of-the-mean
se <- function(x) sqrt(var(x)/length(x))
fname <- "td08-with-lur-2016-09-14.csv"
x <- read.table(fname, header=TRUE, sep=",")
templateRow = x[,1]
# step 0: sort it so we have a known qty
# http://stackoverflow.com/questions/1296646/how-to-sort-a-dataframe-
# by-columns
x <- x[with(x, order(locid, tsdate)), ]
# ... important early example of why step 1 is needed.
# 14   LEG10212  02111250.VL1 2002-11-13      0      0      ...
```

```
# 15   LEG10212  02111262.VL1 2002-11-13      0      0      ...
# ...
# http://stackoverflow.com/questions/1269624/how-to-get-row-from-
#   r-dataframe
# step 1: we have records where several rows describe one day.
## x <- x[1:4000,]
step1o <- x[1,]
templateRow <- x[1,]
#browser();
first <- 1
for (i in 2:length(x$kount)) {
  if (x[i,]$tsdate != templateRow$tsdate ||
      x[i,]$locid != templateRow$locid) {
    ## stash it.
    if (first == 1) {
      step1o <- templateRow
      first <- 0
    }
    else {
      step1o <- rbind(step1o, templateRow)
    }
##    cat("stash at ",i,"\n")
    templateRow <- x[i,]
  } else {
    ## add it
    ## I just add uncertainties here. TODO find the right rule,
    ## use it.
##    cat("add at ",i,"\n")
    templateRow$kount <- templateRow$kount + x[i,]$kount
##    cat("kount now ", templateRow$kount, "\n")
  }
}
# browser();
write.csv(step1o,"td08-with-lur-2016-09-14-reduced.csv",
          row.names=FALSE)
# stop("enough")
# step 2: we now have only one row for each location/day
```

§A.6.7 Translate road class (RC) identifiers to name

```
## File: transrc.r
## Au: J.E. Powell
## Da: 2016-09-27
##
## Based on src/roadclassdb.cc
## (a) = "In the area maintained by the City of Portland only"
## (b) = "Private right-of-way exists"
## (c) = "With rapid transit"
## (d) = "to local street connector (Portland only, subarea = 'P')
## (e) = " -named, but without valid addressing (Clackamas County only, subarea = 'C')"
## (f) = "-unnamed and without valid addressing (Clackamas County only, subarea = 'C')"
## (g) = "with valid address range and street name (in the area maintained by the City of Portland only)"
```

```
## (h) = "with NO Valid Address Range or Street Name (in the area maintained by the City of Portland only)"
## (i) = "no private right-of-way exists"
## (j) = "passable by emergency vehicles (e-911) only (in the area maintained by the City of Portland only)"
transrc <- function(rc) {
    rcnames <- c("Freeway","Ramps; interchanges & feeders","On-ramp (only)","Off-ramp (only)","On- and off-ramp (combination)",
            "Highway","On ramp (only) to highway (a)","Off ramp (only) to highway (a)","On/Off ramp to highway (a)",
            "Primary arterial","Secondary arterial","Other arterial","Minor streets","Local street (d).","Minor street (e)",
            "Minor street (f)","Private named road (b)","Private street (g)","Private named road (i)","Private Road (h).",
            "Unnamed driveway (b)","Unnamed driveway (i)","Unimproved road (j)","Path","Freeway (c)","Highway (c)",
            "Primary arterial (c)","Secondary (c)","Minor with railroad","Minor (c)","Unknown type (only in Yamhill County)",
            "Forest Service road")
rcC <- c(
1110, ## Freeway
1120, ## Ramps; interchanges & feeders
1121, ## On-ramp (only)
1122, ## Off-ramp (only)
1123, ## On- and off-ramp (combination)
1200, ## Highway
1221, ## On ramp (only) to highway (in the area maintained by the City of Portland only)
1222, ## Off ramp (only) to highway (in the area maintained by the City of Portland only)
1223, ## On/Off ramp to highway (in the area maintained by the City of Portland only)
1300, ## Primary arterial
1400, ## Secondary arterial
1450, ## Other arterial
1500, ## Minor streets
1521, ## Local street to local street connector (Portland only, subarea = 'P').
1550, ## Minor street -named, but without valid addressing (Clackamas County only, subarea = 'C')
1560, ## Minor street -unnamed and without valid addressing (Clackamas County only, subarea = 'C')
1700, ## Private named road; private right-of-way exists
1740, ## Private street with valid address range and street name (in the area maintained by the City of Portland only)
1750, ## Private named road; no private right-of-way exists
1760, ## Private Road with NO Valid Address Range or Street Name (in the area maintained by the City of Portland only).
1800, ## Unnamed driveway; private right-of-way exists
1850, ## Unnamed driveway; no private right-of-way exists
2000, ## Unimproved road, passable by emergency vehicles (e-911) only (in the area maintained by the City of Portland only)
3200, ## Path
5101, ## Freeway with rapid transit
5201, ## Highway with rapid transit
5301, ## Primary arterial with rapid transit
5401, ## Secondary with rapid transit
5500, ## Minor with railroad
5501, ## Minor with rapid transit
8224, ## Unknown type (only in Yamhill County)
9000  ## Forest Service road
)
##    browser()
   for (i in 1:length(rcnames)) {
        if (rcC[i] == rc)
            return (rcnames[i])
    }
    return ("unknown")
}
```

§A.6.8 Make a diel graph

```
## FILE: diel.r
## AU: J.E. Powell
## DA: 2016-10-04
## DE: plot traffic count summary diel, separated by weekday/weekend
library(lattice)
source("transrc.r")
## 5000 records randomly selected
fname <- "td08-with-lur-5000-2016-10-04-diel.csv"
## all records (takes long, result is huge like 3.2MB)
fname <- "td08-with-lur-2016-09-14.csv"
# fname <- "td08-after-subset-selection-5000-2016-10-04.csv"
x <- read.table(fname, header=TRUE, sep=",")
4## *help[R](as.Date)* advises us to set the locale.  Otherwise day
## names may be in e.g. French.
lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
## file:~/notes/R/R-has-knowledge-of-weekends.pdf
x$weekendp <- weekdays(as.Date(as.character(x$tsdate),"%Y-%m-%d")) %in% c('Sunday','Saturday')
## Adler p. 269
##xyplot(kount~tstime|weekendp,data=x,groups=
## adler p. 299
## pdf uses colors, and compresses.
## pdf("diel-2016-09-27.pdf")
## postscript("diel-2016-10-04.ps")
```

```
x$rc <- apply(as.matrix(x$rc), 1, transrc)
png(filename="diel-2016-10-04.png",width=4096,height=4096)
trellis.par.set(fontsize=list(text=7))
## xyplot(kount~tstime|as.factor(rc),data=x,groups=weekendp,auto.key=TRUE,scales="free")
xyplot(kount~tstime|as.factor(rc),data=x,groups=weekendp,auto.key=TRUE,scales="free")
dev.off()
```

Make a random subset of the raw data for validation purposes.

§A.6.9

Note that we seed the Random Number Generator (RNG) to make our subset selection reproducible. We use R because it is known to have a superior RNG compared to some other systems.

```
## File: make-random-subset.r
## Au: J.E. Powell
## Da: 2016-09-22
##
## Desc:
##   The ultimate test of any linear regression is whether
## it can reproduce values from the sampled set that were not part
## of making the model.
set.seed(86669078,kind="default", normal.kind="default")
filename <- "td08-with-lur-2016-09-14-reduced.csv"
x <- read.table(filename, header=TRUE, sep=",")
## Adler2009 p. 189
useForModelRowNums <- sample(1:nrow(x), 2*nrow(x)/3);
x$useForModel <- FALSE
for (i in 1:length(useForModelRowNums))
    x[useForModelRowNums[i],]$useForModel <- TRUE;
write.csv(x,"td08-after-subset-selection-2016-09-22.csv",row.names=FALSE)
```

The above code makes a reduced set: counts aggregated to the day level. To mark a random subset of the hourly set, used in deriving and testing the canonical diel cycle, the following code was used.

```
## File: make-random-subset-hourly.r
## Au: J.E. Powell
## Da: 2016-10-25
##
## Desc:
##   The ultimate test of any linear regression is whether
## it can reproduce values from the sampled set that were not part
## of making the model.
set.seed(86669078,kind="default", normal.kind="default")
filename <- "td08-with-lur-2016-09-14.csv"
x <- read.table(filename, header=TRUE, sep=",")
## Adler2009 p. 189
useForModelRowNums <- sample(1:nrow(x), 2*nrow(x)/3);
x$useForModel <- FALSE
for (i in 1:length(useForModelRowNums))
    x[useForModelRowNums[i],]$useForModel <- TRUE;
write.csv(x,"td08-hourly-after-subset-selection-2016-10-25.csv",row.names=FALSE)
```

Warning: this later code takes time, about 24 hours on a fast research host. That time is all spent in this loop:

```
> for (i in 1:length(useForModelRowNums))
+     x[useForModelRowNums[i],]\$useForModel <- TRUE;
```

suggesting that R is not optimized for simple iteration over a lot (1.1×10^6) of rows.

§A.6.10 Run a stepwise linear regression on the model subset.

```
#!/usr/bin/R
## Fi: lm.r
## Au: J.E. Powell
## Da: 2016-08-15
##
## Desc:
## Based on
## file:~/projects/tache/milestones/milestone-2015-04-03-
##  (TD08 statistically defensible traffic density model with
##   freeways)/lm.r
## which in turn was based on
## /home/powellj/projects/tache/milestones/milestone-04-10-2014
##   (TD004 statistically defensible traffic density model
##    with freeways)/lm.r

xform <- function(x) {
  if (TRUE) {
      ## remove 10 outliers, see log entry for Wed Sep 21 20:11:19 2016
      x <- x[x$kount != 0,]
      ## log-transform.
      ## see Wed Sep 21 20:23:38 2016                                    :log:transform:
      x$xformkount <- log(x$kount)
  }
  else {
    x$xformkount <- x$kount
  }
  x
}
filename <- "td08-after-subset-selection-and-rename-2016-10-04.csv"
x <- read.table(filename, header=TRUE, sep=",")
## *help[R](as.Date)* advises us to set the locale.  Otherwise day
## names may be in e.g. French.
lct <- Sys.getlocale("LC_TIME"); Sys.setlocale("LC_TIME", "C")
x$weekendp <- weekdays(as.Date(as.character(x$tsdate),"%Y-%m-%d")) %in% c('Sunday','Saturday')
x$rc <- as.factor(x$rc)
x <- xform(x)
## subset weekdays
x <- subset(x, weekendp==FALSE)
x <- subset(x, useForModel==TRUE)
y <- lm(formula=xformkount~rc+pop100+pop200+pop300+pop400+pop500+
        pop600+pop700+pop800+pop900+or100+or200+or300+or400+or500,
        data=x,singular.ok=FALSE)
cat("r^2 for full model:", summary(y)$r.squared, "\n");
# minimal y
# y <- lm(formula=dacount~rc1110+rc1120+rc1121+rc1122+rc1123+rc1200+
#   rc1222+rc1223+rc1300+rc1400+rc1450+rc1500+rc1521+rc1550+rc1560+
#   rc1700+rc1740+rc1750+rc1760+rc1800+rc1850+rc2000+rc3200+rc5101+
#   rc5201+rc5301+rc5401+rc5500+rc5501+rc8224+rc9000+pop100+pop200+
#   or100+or200,data=x,weight=w)
#anova(y) # as in \cite[p. 408]{adler2009r}
summary(y)
stop("enough")
if (TRUE) {
  #coe <- y$coefficients;
  #coe;
  # x$pred = x$X1200*coe["X1200"] + x$X1221*coe["X1221"] +
  # x$X1300*coe["X1300"] + x$X1400*coe["X1400"] + x$X1450*coe["X1450"]
  # plot(x$dailyavg~x$pred)
    z <- step(y);
  # summary(z)$r.squared; # -> 0.662145
  # anova(z)$'Pr(>F)';
    summary(z) # gives "t value", "Adjusted R-squared: 0.8323"
    postscript("diags-2016-09-22.ps")
    plot(z)
```

```
    dev.off()
}
```

§A.6.11 Count days for each outlier class

The values below (594, 264, etc) were multiplied by 2 to obtain the values in Table 3.2, because we found that the average duration of one count (recorded under one countid) is about 48 hours in TDAT.

```
: powellj@login:~$ echo "select reasonid,count(countid)
from outlier where fondsid = 1 group by reasonid" |
psql -h node01 -d _8666_9078_6965_namelist

|    reasonid | count |
|-------------+-------|
|         304 |   594 |
|         310 |   264 |
|         311 |     3 |
|         303 | 28084 |
|         297 |     0 |
|         301 |  2134 |
|         306 |     3 |
|         298 |  4585 |
|         299 |     0 |
|         312 |     1 |
| (10 lignes) |       |
|             |       |
```

§A.7 Source Code for the Pipeline

We call this pipeline ”TD0N” - traffic density, N being some integer. We can be more specific, the version presented in this appendix is our eighth, so we call this one ”TD08”.

Some interesting statistics.

- 80389 : number of potential count sites located in the study area (TDAT).
- 9352 : number of potential count sites where someone at PBOT allocated a countid (indicating an intention to count I presume) (TDAT).
- 7767 : number of actual counts sites, where a counter was actually laid on the road and read off into the database (TDAT).
- 48 : average number of hours that a counter was left on the road (TDAT).
- 15 : the usual bin size for the counters (in minutes) (TDAT).
- 60 : a bin size also used for some few counters (in minutes) (TDAT).
- 21 : approximate number of years of traffic count data contained in the PBOT database (starting in 1986, ending in 2006) (TDAT)
- 11072 : lines of C++ code written to produce the maps and statistics reported here.
- 8 : minutes to read all counts in PORTAL-FHWA
- 12 : gigabytes of raw ASCII data in PORTAL-TOTAL
- 885 : megabytes of SQL in TDAT

- 1128: CPUs Total in the Gaia cluster (localhost/Ganglia snapshot of 2014-11-26)
- 94: Hosts (total) in the Gaia cluster (localhost/Ganglia snapshot of 2014-11-26)
- 105178: road segments in the PDX street map
- 2005 to 2014: years accumulated in PORTAL-TOTAL

Source code for the TD0N pipeline forms the remainder of this document.

```
// FILE: pass10.cc
// AU: James E. Powell
// DATE: 2016-09-15

#include "tache.h"
#include "tdOn-uml.h"
#include <string.h>

using namespace std;

#define PORTAL_ONEOFF 0
string gPickDate1;
string gPickDate2;
bool gDoingPortalToy = false;

Pass10::Pass10(bool doToy, const OneFonds &fonds) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()
                    +string(NC_TBLNAM_POSTFIX) // table
                    )
{
  tout << "Pass10 has been instantiated." << endl;
  if (gDoingPortalToy || PORTAL_ONEOFF) {
    cerr << "WARNING: doing PORTAL Toy mode.\n";
    FILE_LOG() << "WARNING: doing PORTAL Toy mode.\n";
  }
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-10");
}

vector<Location> *Pass10::QueryForAccumulation(const std::string &tableName)
{
  FILE_LOG() << "Pass10::QueryForAccumulation(\"" << tableName << "\")\n";
  LOG_TAB_IN();
  vector<Location> *result = new vector<Location>;
  stringstream query;
  query << "select " << GetFonds().GetLocationKey();
  // tableName is the fonds's mastertable plus magic number (see
  // parent constructor).
  query << " from " << tableName;
  if (GetPrefs().m_HasLocation) {
    query << " where ";
    query << GetFonds().GetLocationKey() << "='";
    query << GetPrefs().m_Location << "'";
  }
  query << " order by " << GetFonds().GetLocationKey();
  tout << "Master query: " << query.str() << "\n";
  strs_type *lids;
  lids = get_strlist(query.str());
  // int maxI = GetFonds().GetFondsID() == 1 ? 250 : 20; // <- PORTAL
  // takes 1 hr when maxI = 20
  int maxI = GetFonds().GetFondsID() == 1 ? 250 : 10;
  // <- PORTAL takes 27m when maxI = 10
  //
  // but I think only location ID 1010 records come through meaning
  // since no records come from the other station, pass50 writes them
  // all off as missing the second channel ("3 records were found to
  // be 'missing data for a channel'-type outliers" it says).
#if (PORTAL_ONEOFF)
```

```
  // boost portal up to 100 records - it's an hour long very good test.
  // (10 records is a 30 minute not very good test).
  maxI = GetFonds().GetFondsID() == 1 ? maxI : 100;
  // make sure some records make it to pass60 for this one-off PORTAL run.
#endif
  int numI = 0;
  char hname[2048];
  assert(gethostname(hname, 2048) == 0);
  LOG("hname: %s", hname);
  bool limitRun = (string("pliny").compare(hname) == 0);
  // limitRun = true;
#if (PORTAL_ONEOFF)
  limitRun = true;
#endif
  gDoingPortalToy = (limitRun && GetFonds().GetFondsID() == 2);
  if (limitRun)
    tout << "We're running on pliny (JEP's desktop) or "
         << "PORTAL_ONEOFF is set.  Limited to first "
         << maxI << " lids.\n";
  else
    tout << "We're running on \"" << hname << "\", not pliny.  Unlimited.\n";
  if (gDoingPortalToy) {
    cout << "Portal toy mode.\n";
    Location loc;
    loc.m_LocationID = "1989";
    result->push_back(loc);
    loc.m_LocationID = "1990";
    result->push_back(loc);
  }
  else {
    cout << "Not Portal toy mode.\n";
    for (strs_type::const_iterator i = lids->begin(); i != lids->end()
           && (limitRun ? numI < maxI : true)    // -- use to severely
           // -- limit operations
           // -- for gprof etc.
           ; i++, numI++) {
      Location loc;
      loc.m_LocationID = *i;
      result->push_back(loc);
    }
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " Location items.\n";
  return result;
}

/* Important to have companion stations listed as boring/not boring:

   portal-total=> select other_8666 from
   Nodeleg_8666_9078_6965_tache_vector_repository_001 where stationid =
   '1503';

   other_8666
   ------------
   1504
   (1 ligne)
*/
bool Pass10::IsBoringLoc(const std::string &li)
{
  bool boring = true;
  if (GetPrefs().DoToy()) {
    if (li.compare("LEG5567") == 0) {
      boring = false;
      gPickDate1 = string("2006-02-08");
      gPickDate2 = string("2006-02-08");
    }
    else if (li.compare("1010") == 0) {
      boring = false;
      gPickDate1 = string("2006-04-16");
      gPickDate2 = string("2006-04-23");
    }
#if (TACHE_DO_DIURNAL)
    gPickDate2 = gPickDate1;
#endif
    if (!boring)
      tout << "not skipping " << (li) << " ... it's not boring.\n";
  }
  else
    boring = false;
  return boring;
}

vector<Location> *Pass10::RemoveBoringLoc(vector<Location> &lids)
{
  vector<Location> *result = new vector<Location>;
```

```
  int numBoring = 0;
  int numOutlier = 0;
  for (vector<Location>::const_iterator li = lids.begin();
       li != lids.end(); li++) {
    if (IsBoringLoc(li->m_LocationID)) {
      numBoring++;
      continue;
    }
    if (GetFonds().IsOutlierLoc(li->m_LocationID)) {
      numOutlier++;
      continue;
    }
    result->push_back(*li);
  }

  // n.b. the text of the next two FILE_LOGs shall contain the text in
  // macro TAG_OUTL to allow this type of grep to find the range of
  // removals:
  //
  // grep outl7739 /tmp/myappaE0BKc  | sort -u
  //
  // Similar text exists in all of these methods:
  // Pass10::RemoveBoringLoc
  // Pass20::RemoveOutlierCountIDs
  // Pass30::RemoveOutlierCountDays
  FILE_LOG() << TAG_OUTL << "Removed " << numBoring
             << " outlier records (boring location type).\n";
  FILE_LOG() << TAG_OUTL << "Removed " << numOutlier
             << " outlier records (location type).\n";
  return result;
}

void Pass10::populate()
{
  FILE_LOG() << "Pass10::populate()\n";
  LOG_TAB_IN();
  ConnectToDB();
  m_Locations = QueryForAccumulation(GetTable());
  // this select gives us something like
  //   cat  | location
  // -------+----------
  //   1061 | LEG213
  //   1062 | LEG36653
  //   1065 | LEG210
  //   1066 | LEG247
  // [...]
  vector<Location> *newLoc = RemoveBoringLoc(*m_Locations);
  delete m_Locations;
  m_Locations = newLoc;
  if (m_Locations->size()) {
    tout << "Found " << m_Locations->size() << " locations.\n";
  } else {
    if (GetPrefs().m_HasLocation) {
      tout << "No non-outlier records found at the selected location '"
           << GetPrefs().m_Location << "'.\n";
    }
    throw("No locations found.  Aborting.");
  }
  FILE_LOG() << "processing " << m_Locations->size()
             << " rows from " << GetTable() << "\n";
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//  false: unable to validate the values in the database.  They are NOT GOOD.
bool Pass10::validate()
{
  bool result = false;
  return result;
  ConnectToDB();
  stringstream alllocs;
  for (vector<Location>::const_iterator i = m_Locations->begin();
       i != m_Locations->end(); i++)
    alllocs << (*i).m_LocationID << "\n";
  FILE_LOG() << "Validating locs: \"\n" << alllocs << "\"\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
```

```
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(alllocs.str(), knownGood);
  result = true;
  return result;
}



// FILE: pass20.cc
// AU: James E. Powell
// DATE: 2016-09-15
#include "tache.h"
#include "td0n-uml.h"

std::string PORT_NO_OTH("PORTAL station with no other_8666 - it sits "
                        "on one side of the freeway with no station on "
                        "the other side.");

std::string NKRC_DESC("no kounts were recorded under this countid");
std::string NKRL_DESC("countids were assigned to this location, "
                      "but no kounts were recorded at this location");
std::string NCR_DESC("no counts were recorded at this location");

LocationCount::LocationCount(const std::string& lid,
                             const std::string& cid) : m_LocationID(lid),
                                                       m_CountID(cid),
                                                       m_NumKountBins("unset")
{}
LocationCount::LocationCount() {}

inline std::ostream& operator<<(std::ostream& a, const LocationCount &b)
{
  return a << "[a LocationCount with lid = " << b.m_LocationID
           << " and cid " << b.m_CountID << "]";
}

using namespace std;

Pass20::Pass20(bool doToy, const OneFonds &fonds,
               std::vector<Location> *theLocs) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_Locations(theLocs)
{
  tout << "Pass20 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-20");
}

Pass20::~Pass20()
{
  delete m_Locations;
}

countid_type *Pass20::QueryForAccumulation(const std::string &locationid)
{
  LOG("Pass20::QueryForAccumulation(\"%s\")", locationid.c_str());
  LOG_TAB_IN();
  stringstream query;
  query << "select countid from public.volcount where locationid='"
        << locationid << "'";
  if (GetPrefs().m_HasCountID) {
    query << " and CountID='";
    query << GetPrefs().m_CountID << "'";
  }
  query << " order by CountID";
  tout << "Master query: " << query.str() << "\n";
  //  cout << "qu: " << query.str() << endl;
  //  exit(1);
  countid_type *result;
  result = get_strlist(query.str().c_str());
  LOG_TAB_OUT();
  LOG("-> (found %d rows)", result->size());
  return result;
}
```

```
// Called from populate().
// Calls QueryForAccumulation to get countids.
vector<LocationCount> *Pass20::AddLocationCounts(vector<Location> &lids)
{
  FILE_LOG() << "Pass20::AddCounts(...)\n";
  LOG_TAB_IN();
  int numOutlierLocs = 0;
  int numOutlierLocCnts = 0;
  int numLocCounts = 0;
  vector<LocationCount> *result = new vector<LocationCount>;
  for (vector<Location>::iterator li = lids.begin();
       li != lids.end(); li++) {
    const Location &l = (*li);
    const string &lid = l.m_LocationID;
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\".\n";
    change_db(GetFonds().GetSlaveDBName());
    // tout << "Pushing countids for lid: " << lid << endl;
    countid_type *countids_b4 = QueryForAccumulation(lid);
    if (countids_b4->size() == 0) {
      GetFonds().MarkOutlier(&lid, NULL, NULL, NCR_DESC);
      numOutlierLocs++;
      delete countids_b4;
      continue;
    }
    FILE_LOG() << "Counts were recorded at this location.\n";
    FILE_LOG() << "QueryForAccumulation returned "
               << countids_b4->size() << " countids.\n";
    cids_type *countids = RemoveOutlierCountIDs(lid, *countids_b4);
    delete countids_b4;
    if (countids->size() == 0) {
      FILE_LOG() << "No counts remain after removing outliers. "
        "Moving on to the next location.\n";
      delete countids;
      continue;
    }
    FILE_LOG() << countids->size()
               << " counts remain after removing outliers.\n";
    char hname[2048];
    assert(gethostname(hname, 2048) == 0);
    bool limitRun =
      (string("pliny").compare(hname) == 0) && GetFonds().GetFondsID() == 2;
    int maxI = 5;
    int numI = 0;
    if (limitRun) {
      tout << "In pass20, we're (still) running on pliny (JEP's desktop) "
           << "and this is (still) PORTAL.\n"
           << "Limited to first " << maxI << " location+counts.\n";
    }
    bool gotKounts = false;
    for (cids_type::iterator ci = countids->begin();
         ci != countids->end() &&
           (limitRun ? numI < maxI : true); // -- use to severely
                                            // -- limit operations for
                                            // -- gprof etc.
         ci++, numI++) {
      counts_type kounts;
      times_type times;
      ch_type channels;
      GetFonds().SelectTimesAndCounts(*ci, kounts, times, channels);
      if (kounts.size()) {
        gotKounts = true;
        numLocCounts += kounts.size();
        LocationCount newLoc;
        newLoc.m_LocationID = lid;
        newLoc.m_CountID = *ci;
        result->push_back(LocationCount(lid, *ci));
        FILE_LOG() << "STAC returned " << kounts.size()
                   << " kounts, so we're keeping this loc/count.\n";
        FILE_LOG() << "So, as a consequence we now have "
                   << result->size() << " items in the result list.\n";
      }
      else {
        FILE_LOG() <<
          "STAC returned zero kounts, "
          "so we're throwing out this loc/count.\n";
        GetFonds().MarkOutlier(&lid, &(*ci), NULL, NKRC_DESC);
        numOutlierLocCnts++;
      }
    }
    if (!gotKounts) {
      FILE_LOG() << "No kounts were recorded at this location. "
        "Marking this location as outlier.\n";
      GetFonds().MarkOutlier(&lid, NULL, NULL, NKRL_DESC);
```

```
      numOutlierLocs++;
    }
    delete countids;
  } // for (vector<Location>::iterator li = lids.begin();
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " LocationCounts.\n";
  cout << TAG_OUTL << "In " << GetFonds() << ",\n" << TAG_OUTL
       << "Pass20 saw " << lids.size()
       << " locations, and of these\n"
       << TAG_OUTL << numOutlierLocs << " location records were found to be"
       << " empty (no counts or counts but no kounts).\n"
       << TAG_OUTL << "Tache "
       << "today is designed to work with existing kounts records only so\n"
       << TAG_OUTL
       << "these were all marked as outliers of one of these two classes:\n"
       << TAG_OUTL << "- " NCR_DESC "\n"
       << TAG_OUTL << "- " NKRL_DESC "\n";
  if (numOutlierLocCnts) {
    cout << TAG_OUTL << "Additionally, we investigated " << numLocCounts
         << " loc+counts to discover that\n"
         << numOutlierLocCnts << " of these are empty, and are now filed\n"
         << TAG_OUTL << "as outliers under "
         << "'no kounts were recorded under this countid'.\n";
  }
  else
    cout << TAG_OUTL << "All loc+counts in this fonds were "
         << "found to have kounts under them.\n";
  return result;
}

// The caller is responsible for freeing the result.
cids_type *Pass20::RemoveOutlierCountIDs(const string &lid,
                                         const cids_type &cids_b4)
{
  cids_type *result = new cids_type;
  int numRemoved = 0;
  FILE_LOG() << "Pass20::RemoveBoringLocCounts(...)\n";
  LOG_TAB_IN();
  assert(result);
  for (cids_type::const_iterator i = cids_b4.begin();
       i != cids_b4.end(); i++) {
    bool itsOK = true;
    std::string cid = *i;
    itsOK = !GetFonds().IsOutlierLocCount(lid, cid);
    if (itsOK) {
      if (gDoingPortalToy) {
        string dateOfInterest_1("2013-01-16");
        string dateOfInterest_2("2013-01-17");
        string dateNow = cid.substr(0, 10);
        //         cout << "dateNow: \"" << dateNow << "\"\n";
        //         cout << "doi1: \"" << dateOfInterest_1 << "\"\n";
        itsOK = (dateNow.compare(dateOfInterest_1) == 0 ||
                 dateNow.compare(dateOfInterest_2) == 0);
      } // if (gDoingPortalToy)
      if (itsOK)
        result->push_back(cid);
    } // if (itsOK) {
    else
                        numRemoved++;
  }
  // n.b. the text of the next FILE_LOG shall contain the text in
  // macro TAG_OUTL to allow this type of grep to find the range of
  // removals:
  //
  // grep outl7739 /tmp/myappaE0BKc  | sort -u
  // Similar text exists in all of these methods:
  // Pass10::RemoveBoringLoc
  // Pass20::RemoveOutlierCountIDs
  // Pass30::RemoveOutlierCountDays
  FILE_LOG() << TAG_OUTL << "Removed " << numRemoved
             << " outlier records (loc+countid type).\n";
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " cids.\n";
  return result;
}

std::vector<Location> *Pass20::SieveNoOthers(std::vector<Location> *lids)
{
  FILE_LOG() << "Pass20::SieveNoOthers(...)\n";
  LOG_TAB_IN();
  int numOutliers = 0;
  vector<Location> *result = new vector<Location>;
  for (vector<Location>::iterator li = lids->begin();
       li != lids->end(); li++) {
    const Location &l = (*li);
```

```
    const string &lid = l.m_LocationID;
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\".\n";
    change_db(GetFonds().GetSlaveDBName());
    stringstream query;
    query
      << "select "
      "other_8666 from Nodeleg_8666_9078_6965_tache_vector_repository_001"
      << " where stationid = '" << lid << "';";
    // cout << query.str() << endl;
    bool isNull;
    string otherID = do_string_lookup(query.str(), &isNull);
    if (isNull) {
      FILE_LOG() << "No other_8666 found.  Marking it as an outlier.\n";
      GetFonds().MarkOutlier(&lid, NULL, NULL, PORT_NO_OTH);
      numOutliers ++;
    }
    else {
      FILE_LOG() << "Found other (similar to PORTAL's 'opposite' "
                 << "in total_freewy_stns) with id: " << otherID << endl;
      result->push_back(l);
    }
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " Locations.\n";
  cout << TAG_OUTL << "In " << GetFonds() << ",\n"
       << TAG_OUTL << "Pass20 saw " << lids->size()
       << " locations, and of these\n"
       << TAG_OUTL << numOutliers
       << " locations were found to be PORTAL locations"
       << " without 'others' (matching stations for traffic going the other\n"
       << TAG_OUTL << "direction.  Tache today is designed to work "
       << " with PORTAL stations\n"
       << TAG_OUTL << "that record traffic flowing both ways "
       << "simultaneously (and indeed\n"
       << TAG_OUTL << "TDAT recordings on two channels in "
       << "opposite directions) so\n"
       << TAG_OUTL << "these were all marked as outliers of class:\n"
       << TAG_OUTL << "\"" << PORT_NO_OTH << "\"\n";
  delete lids;
  return result;
}

void Pass20::populate()
{
  FILE_LOG() << "Pass20::populate()\n";
  LOG_TAB_IN();
  ConnectToDB();
  // Any fonds02 lacking others get tossed.
  bool isPortal = (GetFonds().GetName().compare("PORTAL-TOTAL")==0);
  if (isPortal)
    m_Locations = SieveNoOthers(m_Locations);
  // this select gives us something like
  //   cat  | location
  // -------+----------
  //   1061 | LEG213
  //   1062 | LEG36653
  //   1065 | LEG210
  //   1066 | LEG247
  // [...]
  if (m_Locations->size()) {
    tout << "Found " << m_Locations->size() << " locations.\n";
  } else {
    if (GetPrefs().m_HasLocation) {
      tout << "No records found at the selected location '"
           << GetPrefs().m_Location << "'.\n";
      if (isPortal)
        tout << "Perhaps the station has no 'other_8666'.\n";
    }
    throw("No locations found.  Aborting.");
  }
  m_LocationCounts = AddLocationCounts(*m_Locations);
  tout << "Found " << m_LocationCounts->size() << " location/counts.\n";
  // Output a little extra 'I hear and obey' when then user has gone
  // out of the way to specify that tache shall operate on one single
  // location (the -L flag).
  if (GetPrefs().m_HasLocation && m_LocationCounts->size())
    tout << "Specifically, this location specified on the command line " <<
      "using -L, which is: \"" << (*m_LocationCounts)[0] << "\".\n";
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
```

```
//  false: unable to validate the values in the database.  They are
//         NOT GOOD.
bool Pass20::validate()
{
  bool result = false;
  return result;
}

void Pass20::DumpCache()
{
  LOG("Pass20::DumpCache(...)");
  LOG_TAB_IN();
  change_db(GetFonds().GetMasterDBName());
  //  drop_table(TACHE_MAGIC_NUMBER "_pass20_cache");
  string tnam(TACHE_MAGIC_NUMBER "_pass20_cache");
  // n.b. we could write this to a CSV file.  Using postgresql has
  // some major advantages:
  //
  // 1. parsing CSV is a PITA - type inference must happen at least.
  // 2. PK enforcement will catch me if I write duplicate entries.
  if (!table_exists(tnam))
    create_table(tnam, "lid text, cid text, PRIMARY KEY (lid, cid)");
  for (std::vector<LocationCount>::const_iterator i =
         m_LocationCounts->begin();
       i != m_LocationCounts->end(); i++) {
    const LocationCount &lc = (*i);
    stringstream qs;
    qs << "INSERT INTO " << tnam << " (lid, cid) VALUES (";
    qs << "'" << lc.m_LocationID << "',";
    qs << "'" << lc.m_CountID << "')";
    assert(exec_query(qs.str()) == 0);
  }
  LOG_TAB_OUT();
  LOG("-> null");
}



// FILE: pass30.cc
// AU: James E. Powell
// DATE: 2016-09-15

#include "tache.h"
#include "td0n-uml.h"

typedef enum pops {
  s_begin = 0,
  s_accum = 1,
  s_dump_bin = 2,
  s_end_of_list = 3,
  s_accept = 4,
  s_mark_outlier = 5,
  s_skip_till_tomorrow = 6,
  s_skip_till_tomorrow_after_day_start = 7
} pops;

using namespace std;

std::string
NUMCHANOUTL_DESC("the number of channels is not in (1,2) for this count.");
std::string
TC_DESC("two (or more) channels of data exist that "
        "share only a single channel number.");
std::string IDOR_DESC("incomplete day of recordings");
std::string
EXTYP_DESC("Unacceptable (or NULL) excepttype.  "
           "We expect to find e.g. \"Normal Weekday\".");

CountDayChannel::CountDayChannel(const std::string& lid,
                   const std::string& cid,
                   std::string & theDate,
                   int theChannelID,
                   int numChan)
: m_LocationID(lid),
    m_CountID(cid),
    m_Date(theDate),
    m_ChannelID(theChannelID),
    m_NumChannels(numChan)
{}

CountDayChannel::CountDayChannel() {}

Pass30::Pass30(bool doToy,
               const OneFonds &fonds,
               std::vector<LocationCount> *theLocCounts) :
```

```
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_LocationCounts(theLocCounts),
  m_DayStartText("00:00:00"), m_DayEndText("23:45:00")

{
  tout << "Pass30 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-30");
}

Pass30::~Pass30()
{
  delete m_LocationCounts;
}

// returns:
//   true: zero times/counts found.  Skip this location/count.
//   false: we found times/counts.  Accept this location/count.
bool Pass30::QueryForAccumulation(const string &cid,
                                  counts_type &kounts,
                                  times_type &times,
                                  ch_type &channels)
{
  bool result;
  FILE_LOG() << "Pass30::QueryForAccumulation(\"" << cid << "\")\n";
  LOG_TAB_IN();
  GetFonds().SelectTimesAndCounts(cid, kounts, times, channels);
  result = (kounts.size() == 0);
  if (result) {
    string rstr("warning: no counts were recorded under "
                "this countid, should not happen in pass30.");
    tout << rstr << "\n";
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result << " (found " << kounts.size()
             << " rows in Pass30::QueryForAccumulation)\n";
  return result;
}

int FindNumChan(const ch_type &channels)
{
  int numChan = 0;
  assert(channels.size());
  for (unsigned int cc = 0; cc < channels.size(); cc++) {
    if (channels[cc] > numChan)
      numChan = channels[cc];
  }
  return numChan;
}

// Called from populate().
// Calls QueryForAccumulation to get countids.
vector<CountDayChannel> *Pass30::
AddCountDayChannels(vector<LocationCount> &lcs)
{
  FILE_LOG() << "Pass30::AddCountDayChannels(...)\n";
  LOG_TAB_IN();
  vector<CountDayChannel> *result = new vector<CountDayChannel>;
  int numCounts = 0;
  int numDays = 0;
  int numOutliers = 0;
  int numBadNumChanOutl = 0;
  int numTwoChanOneNumOutl = 0;
  int numSeen = 0;
  for (vector<LocationCount>::iterator li = lcs.begin();
       li != lcs.end(); li++) {
    const LocationCount &l = (*li);
    const string &lid = l.m_LocationID;
    const string &cid = l.m_CountID;
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\", countID \"" << cid << "\".\n";
    numCounts ++;
    change_db(GetFonds().GetSlaveDBName());
    // tout << "Pushing countids for lid: " << lid << endl;
    counts_type counts;
    times_type times;
    ch_type channels;
    if (QueryForAccumulation(cid, counts, times, channels))
      continue;
    RemoveOutlierCountDays(lid, cid, counts, times, channels);
```

```
    if (times.size() == 0) {
      FILE_LOG() << "All results were outlier.  Moving to next location.\n";
      continue;
    }
    FILE_LOG() << "Reading over " << times.size() <<
      " records in the counts array resulting from the last SELECT.\n";
    pops st = s_begin;
    string fmt = string("%H:%M:%S");
    string dfmt = string("%Y-%m-%d");
    string dayOfKount;
    int numChan = FindNumChan(channels);
    // tache is designed for two cases: either a single channel with
    // channel id 1, or two channels with channel ids 1, 2.
    if (numChan != 1 && numChan != 2) {
      //      cerr << "Warning: numChan = " << numChan << " in lid " << lid
      //           << " and cid " << cid << ".\n";
      FILE_LOG() << "Warning: numChan = " << numChan << " in lid " << lid
           << " and cid " << cid << ".\n";
      GetFonds().MarkOutlier(&lid, &cid, NULL, NUMCHANOUTL_DESC);
      numBadNumChanOutl ++;
      continue;
    }
    if (numChan < 0) {
      cerr << "Skipping negative channel count case.\n";
      // this never happens and if it did we would want to make an
      // outlier record for it instead of blindly continuing.
      assert(0);
    }
    string lastGoodTimestamp("");
    for (unsigned int chan = 1; chan <= static_cast<unsigned int>(numChan);
         chan++) {
      for (unsigned int cc = 0; cc < times.size(); cc++) {
        assert(channels[cc] <= numChan);
        if (channels[cc] != static_cast<int>(chan)) {
#define SPAMOK 0
#if (SPAMOK)
          // Spammy, but useful for debugging when -L flag is supplied.
          FILE_LOG() << "channels[" << cc
                     << "] == " << channels[cc]
                     << " which is != the current channel which is chan="
                     << chan << " so I'm skipping it.\n";
#endif
          continue;
        }
        else {
#if (SPAMOK)
          FILE_LOG() << "channels[" << cc
                     << "] == " << channels[cc]
                     << " which is == the current channel which is chan="
                     << chan << " so I'm NOT skipping it.\n";
#endif
        }
        string timesText = MakeTimesText2(&times[cc], fmt.c_str());
        string dayOfKount = MakeTimesText2(&times[cc], dfmt.c_str());
        FILE_LOG() << "at cursor=" << cc << " st=" << st
                   << " timestamp: " << dayOfKount << " "
                   << timesText << " lastGoodTimestamp="
                   << lastGoodTimestamp << "\n";
        if (lastGoodTimestamp.compare(timesText) == 0
            && st != s_skip_till_tomorrow_after_day_start
            && st != s_skip_till_tomorrow) {
          FILE_LOG()
            << "We found two timestamps in a row, both with the same "
            << "channel number.  This is a case of "
            << "issue OFA: new outlier - two channels of records "
            << "that share a single channel number.\n";
          string dayOfKount = MakeTimesText2(&times[cc], dfmt.c_str());
          GetFonds().MarkOutlier(&lid, &cid, &dayOfKount, TC_DESC, true);
          numTwoChanOneNumOutl ++;
          st = s_skip_till_tomorrow_after_day_start;
        }
        else
          lastGoodTimestamp = timesText;
        bool doAgain = true;
        while (doAgain) {
          doAgain = false;
          switch (st) {
          case s_begin:
            // find a day start.
            numDays ++;
            numSeen = 0;
            if (m_DayStartText.compare(timesText) == 0) {
              st = s_accum;
              doAgain = true;
            }
```

```
  else {
    st = s_skip_till_tomorrow;
    FILE_LOG()
      << "The current kount is the first one in a day, "
      << "but it's not a day start.\n";
    GetFonds().MarkOutlier(&lid, &cid, &dayOfKount, IDOR_DESC, true);
    numOutliers ++;
  }
  break;
case s_skip_till_tomorrow_after_day_start:
  if (m_DayStartText.compare(timesText) == 0)
    FILE_LOG()
      << "Outlier marked but we need to "
      << "dodge over this beginning-of-day timestamp.\n";
  else
    st = s_skip_till_tomorrow;
  break;
case s_skip_till_tomorrow:
  // find a day start after marking this day as outlier.
  if (m_DayStartText.compare(timesText) == 0) {
    // We found a timestamp at time 00:00:00.  It's a brand new day.
    st = s_accum;
    doAgain = true;
  }
  break;
case s_accum:
  numSeen ++;
  //       cout << "s_accum, cc: " << cc << " and times.size(): " <<
  //         times.size() << "\n";
  if (m_DayEndText.compare(timesText) == 0) {
    // We found a timestamp at time 23:45:00.  It's the
    // end of a day.
    FILE_LOG() << "... the current kount is day-end\n";
    st = s_accept;
    doAgain = true;
  } // handling the current kount being at day-end
  else if ((cc + 1) == times.size() ||
           ((cc + 2) == times.size() && numChan == 2)) {
    FILE_LOG()
      << "The current kount is the last one in the whole list.\n";
    if (m_DayEndText.compare(timesText)) {
      FILE_LOG() << "... but it's not a day-end\n";
      st = s_mark_outlier;
      doAgain = true;
    }
  } // handling end of the whole list
  else {
    // n.b. that this depends on earlier if clauses,
    // particularly "if ((cc + 1) == times.size()) {" (the
    // first one).
    string nextDayOfKount =
      MakeTimesText2(&times[cc + 1], dfmt.c_str());
    if (dayOfKount.compare(nextDayOfKount)) {
      FILE_LOG()
        << "... we are about to enter a new day but "
        << "never saw dayendtext.\n";
      st = s_mark_outlier;
      doAgain = true;
    }
  } // handling entering a new day
  break;
case s_accept:
  st = s_begin;
  cout << "numSeen: " << numSeen << "\n";
  if (numSeen != 96) {
    // see issue "0FC: midday data dropout isn't caught"
    FILE_LOG()
      << "This count has data drop out in the middle of the day.\n";
    GetFonds().MarkOutlier(&lid, &cid, &dayOfKount, IDOR_DESC, true);
    numOutliers ++;
  }
  else {
    FILE_LOG()
      << "This count has 96 entries (24 hrs x 4 entries/per). "
      << "Saving it.\n";
    result->push_back(CountDayChannel(lid, cid, dayOfKount,
                                      chan, numChan));
  }
  break;
case s_mark_outlier:
  // tout << cc << " should be end of day and isn't, tossed
  // day as outlier.\n";
  GetFonds().MarkOutlier(&lid, &cid, &dayOfKount, IDOR_DESC, true);
  numOutliers ++;
  st = s_begin;
```

```
          break;
        default:
          tout << "unknown state " << st << endl;
          assert(0);
          break;
        } // switch (st)
        if (doAgain)
          FILE_LOG() << "doAgain was set.\n";
      } // if (doAgain)
    } // for (unsigned int cc = 0; cc < times.size(); cc++)
  } // for (unsigned int chan = 1; chan <= numChan; chan++) {
} // for (vector<LocationCount>::iterator li = lcs.begin();
LOG_TAB_OUT();
cout << TAG_OUTL << "In " << GetFonds() << ",\nPass30 saw " << numCounts
     << " countIDs together covering "
     << numDays << " days, and of these\n"
     << TAG_OUTL << numOutliers
     << " countid+day time series recordings were found to be"
     << " only partial-day records (e.g. a record from\n"
     << TAG_OUTL
     << "11:30 to 14:30).  Tache today is designed to work with full-day\n"
     << TAG_OUTL
     << "records (from 00:00 through to 23:45 in 15 minute increments) "
     << "only,\n" << TAG_OUTL
     << "so these were all marked as outliers with the description\n"
     << TAG_OUTL << IDOR_DESC << ".\n";
if (numBadNumChanOutl) {
  cout << TAG_OUTL << "Additionally, "
       << numBadNumChanOutl << " of " << numCounts
       << " loc+counts were found to\n"
       << TAG_OUTL
       << "have an illegal number of channels.  Tache is designed today\n"
       << TAG_OUTL
       << "to work with exactly 1 or 2 channels, so these were marked\n"
       << TAG_OUTL << "as outliers with the description.\n"
       << TAG_OUTL << NUMCHANOUTL_DESC;
}
if (numTwoChanOneNumOutl) {
  cout << TAG_OUTL
       << "Additionally, " << numTwoChanOneNumOutl << " of " << numCounts
       << " loc+counts+days were found to\n"
       << TAG_OUTL
       << "have two or more rows with the same "
       << "timestamp and channel number.\n"
       << TAG_OUTL << "(issues 0FA, 0FB) so these were marked\n"
       << TAG_OUTL << "as outliers with the description.\n"
       << TAG_OUTL << TC_DESC << "\n";
}
FILE_LOG() << "-> " << result->size() << " CountDayChannels.\n";
return result;
}

// Pass30::RemoveOutlierCountDays is called once per lid+cid.
//
// The arrays in kounts, times, channels (which should all be the same
// length) are the time series recorded under the given cid, at the
// given lid.  This routine effectively deletes the outlier cases
// which have previously been identified in the time series and which
// were recorded in the outlier table as loc+count+day type outliers.
void Pass30::RemoveOutlierCountDays(const string &lid,
                                    const string &cid,
                                    counts_type &kounts,
                                    times_type &times,
                                    ch_type &channels)
{
  int numRemoved = 0;
  FILE_LOG() << "Pass30::RemoveOutlierCountDays(...)\n";
  LOG_TAB_IN();
  counts_type rKounts;
  times_type rTimes;
  ch_type rChan;
  string dfmt = string("%Y-%m-%d");
  string dayBeingSkipped;
  for (unsigned int cc = 0; cc < times.size(); cc++) {
    string dayOfKount = MakeTimesText2(&times[cc], dfmt.c_str());
    bool itsOK = true;
    itsOK = (dayBeingSkipped.compare(dayOfKount) == 0) ? false :
      !GetFonds().IsOutlierLocCountDate(lid, cid, dayOfKount);
    if (itsOK) {
      rKounts.push_back(kounts[cc]);
      rTimes.push_back(times[cc]);
      rChan.push_back(channels[cc]);
    }
    else {
      numRemoved++;
```

```
      dayBeingSkipped = dayOfKount;
    }
  }
  // n.b. the text of the next FILE_LOG shall contain the text
  // in the macro TAG_OUTL to allow this type of grep to find the range of
  // removals:
  //
  // grep outl7739 /tmp/myappaE0BKc  | sort -u
  //
  // Similar text exists in all of these methods:
  // Pass10::RemoveBoringLoc
  // Pass20::RemoveOutlierCountIDs
  // Pass30::RemoveOutlierCountDays
  FILE_LOG() << TAG_OUTL << "Removed " << numRemoved
             << " outlier records (loc+count+day type).\n";
  LOG_TAB_OUT();
  kounts = rKounts;
  times = rTimes;
  channels = rChan;
  FILE_LOG() << "-> " << kounts.size() << " count/day records.\n";
}

std::vector<LocationCount> *
Pass30::SieveExceptType(std::vector<LocationCount> *lcs)
{
  FILE_LOG() << "Pass30::SieveExceptType(...)\n";
  LOG_TAB_IN();
  int numOutliers = 0;
  vector<LocationCount> *result = new vector<LocationCount>;
  for (vector<LocationCount>::iterator li = lcs->begin();
       li != lcs->end(); li++) {
    const LocationCount &l = (*li);
    const string &lid = l.m_LocationID;
    const string &cid = l.m_CountID;
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\", count ID \"" << cid << "\".\n";
    change_db(GetFonds().GetSlaveDBName());
    stringstream query;
    query
      << "select excepttype from public.volcount where locationid='"
      << lid << "'"
      << " and countid='" << cid << "'";
    // cout << query.str() << endl;
    bool isNull;
    string exceptType = do_string_lookup(query.str(), &isNull);
    if (isNull) {
      FILE_LOG() << "NULL excepttype found.  Marking it as an outlier.\n";
      GetFonds().MarkOutlier(&lid, &cid, NULL, EXTYP_DESC);
      numOutliers++;
    }
    else {
      FILE_LOG() << "Found excepttype: " << exceptType << endl;
      if (exceptType.compare("Normal Weekday") == 0 ||
          exceptType.compare("Weekend/Holiday") == 0 ||
          exceptType.compare("Normal Weekend") == 0)
        result->push_back(l);
      else {
        FILE_LOG() << "Unacceptable excepttype.";
        GetFonds().MarkOutlier(&lid, &cid, NULL, EXTYP_DESC);
        numOutliers++;
      }
    }
  }
  cout << TAG_OUTL << "In " << GetFonds() << ",\n" << TAG_OUTL << "Pass30 ";
  if (numOutliers) {
    cout << "saw " << lcs->size()
         << " countIDs, and of these "
         << numOutliers << " countid recordings were found to be"
         << " of an excepttype other than normal weekday/weekend.\n"
         << TAG_OUTL
         << "Tache today is designed to work with only records marked as "
         << "normal\n" << TAG_OUTL
         << "so these were all marked as outliers with the "
         << "description\n"
         << TAG_OUTL << "\"" << EXTYP_DESC << "\".\n";
    // This is only allowed in fonds 1 (TDAT).  The other fonds (PORTAL)
    // has excepttype faked up by the materialized views and they are always
    // "normal" as of today 2016-05-13.
    assert(GetFonds().GetFondsID() == 1);
  }
  else
    cout << "found no countids of excepttype other "
         << "than normal weekday/weekend.\n";
  delete lcs;
  LOG_TAB_OUT();
```

```
  FILE_LOG() << "-> " << result->size() << " Location/Counts.\n";
  return result;
}

void Pass30::populate()
{
  FILE_LOG() << "Pass30::populate()\n";
  LOG_TAB_IN();
  ConnectToDB();
  // this select gives us something like
  //   cat  | location
  // -------+----------
  //   1061 | LEG213
  //   1062 | LEG36653
  //   1065 | LEG210
  //   1066 | LEG247
  // [...]
  m_LocationCounts = SieveExceptType(m_LocationCounts);
  if (m_LocationCounts->size()) {
    tout << "Found " << m_LocationCounts->size()
         << " location/counts after SieveExceptType.\n";
  } else
    throw("No location/counts found.  Aborting.");
  m_CountDayChannels = AddCountDayChannels(*m_LocationCounts);
  tout << "Found " << m_CountDayChannels->size()
       << " location/count/day/channels after AddCountDayChannels.\n";
  //  printsome_helper("Pass30", m_CountDayChannels, 60);
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//  false: unable to validate the values in the database.  They are NOT GOOD.
bool Pass30::validate()
{
  bool result = false;
  return result;
#if (0)
  ConnectToDB();
  typedef vector<string> nc_type;
  nc_type dacount;
  // creating the dacount attribute creates several additional attributes,
  // which are counted together as a unit with the dacount attribute.
  stringstream dacountSelect;
  // GetTable() is the master table for the fonds, e.g. Nodeleg.
  dacountSelect << "select locid || ':' || countid || ':' "
                << "|| countdate || ':' || "
                << "dacount || ':' || dacount_delta || ':' || "
                << "dacount_n || ':' || rc || ':' || exceptType "
                << "|| ':' || numkounts from "
                << GetTable() << " order by "
                << "locid,countid,countdate";
  dacount = get_strlist(dacountSelect.str());
  stringstream allcnts;
  for (nc_type::const_iterator i = dacount.begin();
       i != dacount.end(); i++)
    allcnts << (*i) << "\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(allcnts.str(), knownGood);
  result = true;
  return result;
#endif
}



#include "tache.h"
#include "td0n-uml.h"
#include "portallocation.h"
#include "countdaychannel.h"
```

```
using namespace std;

Pass40::Pass40(bool doToy,
               const OneFonds &fonds,
               std::vector<CountDayChannel> *theCDCs) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_CountDayChannels_In(theCDCs),
  m_plocs(NULL)
{
  tout << "Pass40 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-40");
  m_CountDayChannels_Out = new std::vector<CountDayChannel>;
}

Pass40::~Pass40()
{
  delete m_CountDayChannels_In;
}

std::unordered_map<std::string, PortalLocation>
  *Pass40::CollectPortalLocs(vector<CountDayChannel> *cdcs)
{
  FILE_LOG() << "Pass40::CollectPortalLocs()\n";
  LOG_TAB_IN();
  std::unordered_map<std::string, PortalLocation> *result =
    new std::unordered_map<std::string, PortalLocation>;
  for (vector<CountDayChannel>::iterator li = cdcs->begin();
       li != cdcs->end(); li++) {
    const CountDayChannel &l = (*li);
    const string &lid = l.m_LocationID;
    const string &cid = l.m_CountID;
    // Confirm our assertion that all locations have a four-digit ID.
    // (see use of PORTAL_STATION_ID_LEN elsewhere for the reason
    // for this strong requirement).
    assert(lid.length() == PORTAL_STATION_ID_LEN);
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\", countID \"" << cid << "\".\n";
    change_db(GetFonds().GetSlaveDBName());
    // Do we have an entry for this station?
    const std::unordered_map<std::string, PortalLocation>::const_iterator
      oneLoc = result->find(lid); // see pass3.cc, onefonds.cc
    if (oneLoc == result->end()) {
      // No.  Do we have one for its 'other'?
      stringstream query;
      query <<
        "select other_8666 from "
        "Nodeleg_8666_9078_6965_tache_vector_repository_001"
            << " where stationid = '" << lid << "';";
      // cout << query.str() << endl;
      bool isNull;
      string otherID = do_string_lookup(query.str(), &isNull);
      // Stations without others should have been thrown out in
      // Pass20::SieveNoOthers.
      assert(!isNull);
      FILE_LOG() <<
        "Found other (similar to PORTAL's 'opposite' in total_freewy_stns) "
        "with id: " << otherID << endl;
      std::unordered_map<std::string, PortalLocation>::iterator varLoc
        = result->find(otherID);
      if (varLoc == result->end()) {
        // Totally new to us.  Create a record.
        PortalLocation newLoc(lid);
        newLoc.AddAlias(otherID);
        (*result)[lid] = newLoc;
        (*result)[otherID] = newLoc;
      }
      else {
        // Yes, it's been registered under its otherid.  Add lid as an alias.
        varLoc->second.AddAlias(lid);
      }
    }
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " unique PORTAL locs.\n";
  return result;
}

std::vector<CountDayChannel> *Pass40::ResetChannels()
{
```

```
  FILE_LOG() << "Pass40::ResetChannels()\n";
  LOG_TAB_IN();
  std::vector<CountDayChannel> *result = new std::vector<CountDayChannel>;
  for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In->begin();
       cdc != m_CountDayChannels_In->end(); cdc++) {
    const string &lid = cdc->m_LocationID;
    const string &cid = cdc->m_CountID;
    FILE_LOG(LogFile::Debug) << "Checking location ID \""
                             << lid << "\", countID \"" << cid << "\".\n";
    std::unordered_map<std::string, PortalLocation>::iterator varLoc
      = m_plocs->find(lid);
    assert(varLoc != m_plocs->end());
    CountDayChannel outCDC = *cdc;
    int useChan = varLoc->second.FindAliasID(lid);
    assert(useChan >= 0);
    // MangleName() concatenates the two portal stations (self, other)
    // with a comma.  We're finishing the merge of two stations on
    // opposite sides of the freeway.
    outCDC.m_LocationID = varLoc->second.MangleName();
    outCDC.m_ChannelID = useChan + 1;
    outCDC.m_NumChannels = 2;
    // CountIDs in PORTAL have their original station appended.  This
    // breaks sorting and grouping in pass50 (see issue 0BF: pass50
    // marks all PORTAL records as outliers).  Thus we continue
    // merging the data from the two stations by removing the original
    // 4 digit station ID and replacing it with the mangled name
    // version of the location ID (see just above).
    string oname = outCDC.m_CountID.substr(0,
                                           outCDC.m_CountID.length()
                                           - PORTAL_STATION_ID_LEN);
    stringstream nname;
    nname << oname << outCDC.m_LocationID;
    outCDC.m_CountID = nname.str();
    result->push_back(outCDC);
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result->size() << " CDCs with mangled names.\n";
  return result;
}


#define MYFNAME PORTAL_FNAME

void Pass40::populate()
{
  FILE_LOG() << "Pass40::populate()\n";
  LOG_TAB_IN();
  const std::string &fname = GetFonds().GetName();
  if (fname.compare(MYFNAME)) {
    FILE_LOG() << "Fonds \"" << fname << "\""
               << " <> the fonds for which Pass40 is necessary (\""
               << MYFNAME << "\") and so ::populate is a no-op.\n";
    // Copy the daychannels for the next pass to use.
    m_CountDayChannels_Out =
      new std::vector<CountDayChannel>(*m_CountDayChannels_In);
  }
  else {
    ConnectToDB();
    m_plocs = CollectPortalLocs(m_CountDayChannels_In);
    if (m_plocs->size()) {
      tout << "Found " << m_plocs->size()
           << " unique locations after CollectPortalLocs.\n";
    } else
      throw("No location/counts found.  Aborting.");
    m_CountDayChannels_Out = ResetChannels();
    tout << "Found " << m_CountDayChannels_Out->size()
         << " location/count/day/channels after ResetChannels.\n";
    cout << "Random ten with mangled names:\n";
    for (int i = 0; i < 10; i++) {
      int which = random() % m_CountDayChannels_Out->size();
      cout << which << ": " << (*m_CountDayChannels_Out)[which] << "\n";
    }
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//  false: unable to validate the values in the database.  They are NOT GOOD.
bool Pass40::validate()
{
  bool result = false;
  return result;
#if (0)
```

```
  ConnectToDB();
  typedef vector<string> nc_type;
  nc_type dacount;
  // creating the dacount attribute creates several additional attributes,
  // which are counted together as a unit with the dacount attribute.
  stringstream dacountSelect;
  // GetTable() is the master table for the fonds, e.g. Nodeleg.
  dacountSelect << "select locid || ':' || countid || ':' || "
                << "countdate || ':' || "
                << "dacount || ':' || dacount_delta || ':' || "
                << "dacount_n || ':' || "
                << "rc || ':' || exceptType || ':' || numkounts from "
                << GetTable() << " order by "
                << "locid,countid,countdate";
  dacount = get_strlist(dacountSelect.str());
  stringstream allcnts;
  for (nc_type::const_iterator i = dacount.begin();
       i != dacount.end(); i++)
    allcnts << (*i) << "\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(allcnts.str(), knownGood);
  result = true;
  return result;
#endif
}



#include "tache.h"
#include "tdOn-uml.h"
#include "countdaychannel.h"
#include <algorithm> // sort

std::string MCHAN_DESC("missing data for a channel");
std::string
WRCH_DESC("channel 2 data recorded but numchannels==1");

using namespace std;
Pass50::Pass50(bool doToy,
               const OneFonds &fonds,
               std::vector<CountDayChannel> *theCDCs) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_CountDayChannels_In(theCDCs),
  m_CountDayChannels_Out(NULL)
{
  tout << "Pass50 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-50");
}

Pass50::~Pass50()
{
  delete m_CountDayChannels_In;
}

typedef enum pops {
  s_begin = 0,
  s_reset_chseen = 1,
  s_continue = 2,
  s_validate_channels_seen = 3
} pops;

void Pass50::ncabort(vector<CountDayChannel>::iterator &cdc,
                     int expectNumChan)
{
  // Does this ever happen?
  stringstream errmsg;
```

```
  errmsg << "abort: cdc->m_NumChan=" << cdc->m_NumChannels
         << " but expectNumChan=" << expectNumChan
         << " at cdc index "
         << (cdc - m_CountDayChannels_In->begin())
         << ".\n";
  throw(errmsg.str());
}

// Based on Pass30::AddCountDayChannels.
// Called from populate().
vector<CountDayChannel> *Pass50::FindMissingChannelOutliers_WholeDay()
{
  FILE_LOG() << "Pass50::FindMissingChannelOutliers_WholeDay(...)\n";
  LOG_TAB_IN();
  vector<CountDayChannel> *result = new vector<CountDayChannel>;
  vector<CountDayChannel> *oneGroup = NULL;
  int numCDC = m_CountDayChannels_In->size();
  int numOutliersWRCH = 0;
  int numOutliersMCHAN = 0;
  pops st = s_begin;
  string lastKey("");
  bool chSeen[2] = {false, false};
  string fmt = string("%Y-%m-%d %H:%M:%S");
  int numSeen = 0;
  // N.B. we need not remove outlier count days here, cos they've been
  // removed in Pass30.
  FILE_LOG() << "Reading over " << m_CountDayChannels_In->size() <<
    " records in the counts array resulting from pass40.\n";
  // We are assuming a really orderly set of records.  Let's peek at
  // the first few.
  int numTotal = m_CountDayChannels_In->size();
  int numToPrint = std::min(60,
                            static_cast<int>(m_CountDayChannels_In->size()));
  cout << "First " << numToPrint << "/" << numTotal
       << " total records in pass50 (input, full day):\n";
  int printCursor = 0;
  for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In->begin();
       printCursor < numToPrint; printCursor++, cdc++)
    cout << (printCursor + 1) << ": " << (*cdc) << "\n";
  int expectNumChan = -1;
  bool skipGroup = false;
  for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In->begin();
       cdc != m_CountDayChannels_In->end();
       cdc++) {
    FILE_LOG() << "cursor=" << (cdc - m_CountDayChannels_In->begin())
               << ", cdc: " << *cdc << endl;
    bool doAgain = true;
    while (doAgain) {
      stringstream key;
      key << cdc->m_LocationID << ","
          << cdc->m_CountID << "," << cdc->m_Date;
      stringstream nextkey;
      vector<CountDayChannel>::iterator nextCDC = (cdc + 1);
      if (nextCDC != m_CountDayChannels_In->end()) {
        nextkey << nextCDC->m_LocationID << ","
                << nextCDC->m_CountID << "," << nextCDC->m_Date;
      }
      FILE_LOG() << "at cursor=" << (cdc - m_CountDayChannels_In->begin())
                 << " st=" << st << " key: " << key.str()
                 << " lastKey: \"" << lastKey << "\""
                 << " and skipGroup: " << skipGroup << "\n";
      doAgain = false;
      switch (st) {
      case s_begin:
        assert(cdc->m_NumChannels == 1 ||
               cdc->m_NumChannels == 2);
        // The group 'lives' by being copied into the result, or dies
        // (deleted) all together.
        if (oneGroup) delete oneGroup;
        oneGroup = new vector<CountDayChannel>;
        doAgain = true;
        lastKey = "";
        expectNumChan = cdc->m_NumChannels;
        st = s_reset_chseen;
        break;
      case s_reset_chseen:
        chSeen[0] = false;
        chSeen[1] = false;
        st = s_continue;
        doAgain = true;
        break;
      case s_continue:
        //FILE_LOG() << "s_continue: lastKey \"" << lastKey
        //           << "\" and key: \"" << key.str() << "\"\n";
        oneGroup->push_back(*cdc);
```

```
      // We're stepping through a group of entries that all belong
      // to one count + day.  Make sure these entries have the
      // designated number of channels, picked by the first entry
      // in the group.
      assert(cdc->m_ChannelID == 1 || cdc->m_ChannelID == 2);
      if (cdc->m_NumChannels != expectNumChan)
        ncabort(cdc, expectNumChan);
      // This next assert may become an outlier, indicating duplicate
      // records for one loc+cid+time+chan.
      assert(chSeen[cdc->m_ChannelID - 1] == false);
      FILE_LOG() << "Recording presence of data in channel "
                 << cdc->m_ChannelID << "\n";
      chSeen[cdc->m_ChannelID - 1] = true;
      if (cdc->m_NumChannels == 1 &&
          cdc->m_ChannelID != 1) {
        GetFonds().MarkOutlier(&cdc->m_LocationID, &cdc->m_CountID,
                               &cdc->m_Date, WRCH_DESC, true);
        numOutliersWRCH++;
        skipGroup = true;
        cerr << "WARNING: The skipGroup = true code path has never "
             << "been followed,\n"
             << "because this type of outlier has not been seen as of\n"
             << "2016-07-20.  Proceed with caution.\n";
      }
      if (nextCDC == m_CountDayChannels_In->end()) {
        // End of the input set.  Did we see all of the channels?
        st = s_validate_channels_seen;
        doAgain = true;
      }
      else if (nextkey.str().compare(key.str()) == 0
               || nextkey.str().length() == 0)
        FILE_LOG() << "still in group...\n";
      else {
        FILE_LOG() << "End of a group.  Did we see all of the channels?\n";
        st = s_validate_channels_seen;
        doAgain = true;
      }
      break;
    case s_validate_channels_seen:
      numSeen = 0;
      if (skipGroup) {
          FILE_LOG() << "Done skipping a group.\n";
          st = s_begin;
          skipGroup = false;
          continue;
      }
      for (int i = 0; i < cdc->m_NumChannels; i++)
        if (chSeen[i])
          numSeen++;
      FILE_LOG() << "st=s_validate_channels_seen, and numSeen is " << numSeen
                 << " and numchan=" << cdc->m_NumChannels << "\n";
      if (numSeen != cdc->m_NumChannels) {
        GetFonds().MarkOutlier(&cdc->m_LocationID,
                               &cdc->m_CountID,
                               &cdc->m_Date, MCHAN_DESC, true);
        numOutliersMCHAN++;
      }
      else {
        FILE_LOG() << "It's a good result.  Saving this group.\n";
        copy(oneGroup->begin(), oneGroup->end(), back_inserter(*result));
      }
      st = s_begin;
      break;
    } // switch (st)
    if (doAgain)
      FILE_LOG() << "doAgain was set.\n";
    else
      lastKey = key.str();
  } // while doAgain
 } // for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In->b...
 if (oneGroup) delete oneGroup;
 oneGroup = NULL;
 LOG_TAB_OUT();
 FILE_LOG() << "-> " << result->size() << " CountDayChannels.\n";
 cout << TAG_OUTL << "In " << GetFonds() << ",\nPass50 saw " << numCDC
      << " count/day/channels, and of these\n"
      << TAG_OUTL << numOutliersMCHAN << " records were found to be"
      << " '" << MCHAN_DESC << "'-type outliers, and " << numOutliersWRCH
      << "\n" << TAG_OUTL << "were found to be '" << WRCH_DESC
      << "'-type outliers.\n";
 return result;
}

void Pass50::populate()
{
```

```
  FILE_LOG() << "Pass50::populate()\n";
  LOG_TAB_IN();
#if (0)
  const std::string &fname = GetFonds().GetName();
  // this is true and useful in pass50 but also possibly an
  // unnecessarily complication - it's harmless to check all fonds
  // (unless the thing gets too slow in which case this is an obvious
  // optimization).  "premature optimization is the root of all evil"
  // (Donald Knuth).
  std::string MYFNAME("PORTAL-TOTAL");
  if (fname.compare(MYFNAME)) {
    FILE_LOG() << "Fonds \"" << fname << "\""
               << " <> the fonds for which Pass50 is necessary (\""
               << MYFNAME << "\") and so ::populate is a no-op.\n";
    // Copy the daychannels for the next pass to use.
    //
    // We only run pass50 for tdat, because portal is the only other
    // fonds at the moment at portal is guaranteed to have exactly two
    // channels by the station-matching logic of pass40.
    m_CountDayChannels_Out = new
      std::vector<CountDayChannel>(*m_CountDayChannels_In);
  }
  else
#endif
  {
    ConnectToDB();
    // Pass30 emits rows in a strange order (see issue OBA:
    // interesting sort in pass50).  This necessarily evolves because
    // of its function.  We need a different order here, so we sort
    // the list.
    sort(m_CountDayChannels_In->begin(), m_CountDayChannels_In->end(),
         sortbylidciddatechan);
    m_CountDayChannels_Out = FindMissingChannelOutliers_WholeDay();
    tout << "Found " << m_CountDayChannels_Out->size()
         << " location/count/day/channels after "
         << "FindMissingChannelOutliers_WholeDay.\n";
#if (0)
    cout << "Random ten with mangled names:\n";
    for (int i = 0; i < 10; i++) {
      int which = random() % m_CountDayChannels_Out->size();
      cout << which << ": " << (*m_CountDayChannels_Out)[which] << "\n";
    }
#endif
  }
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//  false: unable to validate the values in the database.  They are NOT GOOD.
bool Pass50::validate()
{
  bool result = false;
  return result;
#if (0)
  ConnectToDB();
  typedef vector<string> nc_type;
  nc_type dacount;
  // creating the dacount attribute creates several additional attributes,
  // which are counted together as a unit with the dacount attribute.
  stringstream dacountSelect;
  // GetTable() is the master table for the fonds, e.g. Nodeleg.
  dacountSelect << "select locid || ':' || countid || ':' || "
                << "countdate || ':' || "
                << "dacount || ':' || dacount_delta || ':' || dacount_n || "
                << "':' || rc || ':' || exceptType || ':' || numkounts from "
                << GetTable() << " order by "
                << "locid,countid,countdate";
  dacount = get_strlist(dacountSelect.str());
  stringstream allcnts;
  for (nc_type::const_iterator i = dacount.begin();
       i != dacount.end(); i++)
    allcnts << (*i) << "\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
```

```
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(allcnts.str(), knownGood);
  result = true;
  return result;
#endif
}



// FILE: pass60.cc
// AU: James E. Powell
// DATE: 2016-09-13
#include "tache.h"
#include "tdOn-uml.h"
#include "countdaychannel.h"
#include "locationminutechannel.h"
#include "kounttimechannel.h"
#include <algorithm> // sort

#define VERBOSE_FOR_DUPS 0
#define P60V 0 // verbose for this module
#define P60L() if (P60V) FILE_LOG()

// Quick status overview.  What comes in is a row of CDCs with one row
// per location/count/day.  Recall that a count in TDAT can have
// multiple days.  These are the first two records:
//
// 1: [a CountDayChannel with lid = LEG10248 and cid 04121308.VL1,
// date 2004-12-14, ChannelNo = 1 and numChan = 1]
//
// 2: [a CountDayChannel with lid = LEG10248 and cid 04121308.VL1,
// date 2004-12-13, ChannelNo = 1 and numChan = 1]
//
// Two days, one countid.  However, my workhorse
// SelectTimesAndCounts(...) gives back all of the days under a
// countid.  So, we have to filter out days that belong to other rows
// of the input as we step through STAC output.

using namespace std;
string BADBINSZ_DESC("bin size is other than 15 minutes");
string NEGCT_DESC("kounts negative (less than zero) recorded");
string MDDROP_DESC("PORTAL count w/midday zero kounts (hr >= 5 and <= 20)");

Pass60::Pass60(bool doToy,
               const OneFonds &fonds,
               vector<CountDayChannel> *theCDCs) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_CountDayChannels_In(theCDCs),
  m_LocationMinuteChannels_Out(NULL)
{
  tout << "Pass60 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-60");
  m_InPortal = (GetFonds().GetName().compare("PORTAL-TOTAL") == 0);
}

Pass60::~Pass60()
{
  delete m_CountDayChannels_In;
}

// Method: FindBadBins
// Returns:
//   true: this countid has a nonstandard (non-15-minute) bin size
//  false: this countid has a standard (15-minute) bin size.
bool Pass60::FindBadBins(vector<CountDayChannel>::iterator &cdc)
{
  bool result = false;
  change_db(GetFonds().GetSlaveDBName());
  int ilen;
  if (m_InPortal) {
    // hardcoded for portal, see ish 0C2: PORTAL pass60 finds too
    // many 'bin size other than 15 mins'
    ilen = 15;
  }
  else
    ilen = find_interval_length_for_countid(cdc->m_CountID);
  if (ilen != 15) {
```

```
    GetFonds().MarkOutlier(&cdc->m_LocationID, &cdc->m_CountID,
                           NULL, BADBINSZ_DESC, true);
    result = true;
  }
  return result;
}

// Method: FindZKounts
// Returns:
//   true: there are zero kounts in this day, midday.
//  false: there are no zero kounts in this day, midday
bool Pass60::FindZKounts(timestamp &theTime, int kount,
                         vector<CountDayChannel>::iterator &cdc)
{
  P60L() << "Pass60::FindZKounts(...)\n";
  bool result = false;
  // check for zero kounts in midday.
  string tmpDate =  MakeTimesText2(&theTime, "%H");
  // In \cite[p. 600-601]{stroustrup00:_c_progr_languag},
  // Stroustrup seems to recommend the "useful functions"
  // atof and friends, and these are part of C++ as well as
  // C, so I'm using atoi here.
  int dayTime = atoi(tmpDate.c_str());
  if (errno) {
    perror("atoi");
    assert(0);
  }
  if (kount == 0) {
    if (dayTime >= 5 && dayTime <= 20) {
      GetFonds().MarkOutlier(&cdc->m_LocationID, &cdc->m_CountID,
                             NULL, MDDROP_DESC, true);
      result = true;
    }
  }
  P60L() << "-> " << result << "\n";
  return result;
}

// Method: RemoveDups
//
// Description:
//
// A wonderfully naive approach to finding duplicate rows in the
// output table.  Potentially problematic due to performance (lookups
// are O(log(n)) per Stroustrup) and/or size - this duplicates the
// entire output, which is already potentially gigabytes large.
// Following Knuth "premature optimization &tc" I will not touch it
// until it fails.
vector<LocationMinuteChannel> *
RemoveDups(vector<LocationMinuteChannel> *outTable)
{
  FILE_LOG()
    << "Pass60::RemoveDups([" << outTable->size() << " LMC entries])\n";
  LOG_TAB_IN();
  // sort by LocID, timestamp [ascending], chan. [ascending]
  sort(outTable->begin(),
       outTable->end(),
       sortbylidcidtimechan);
  vector<LocationMinuteChannel> *result = new vector<LocationMinuteChannel>;
  // "A \empth{map} is a sequence of (key, value) pairs that provides
  // for fast revtrieval based on the key."
  // \cite[p. 480]{stroustrup00:_c_progr_languag}.
  map<string, LocationMinuteChannel> fastRet;
  for (vector<LocationMinuteChannel>::iterator lmc = outTable->begin();
       lmc != outTable->end();
       lmc++) {
    string hc = lmc->hashkey();
    // This is "not as cheap as subscripting an array with an integer"
    // (ibid p. 485).  If it turns out to be too slow, "a hashed
    // container is often the answer, (\section 17.6)" (ibid)
    map<string, LocationMinuteChannel>::iterator p = fastRet.find(hc);
    if (p != fastRet.end()) {
      FILE_LOG() << "duplicates are found.  New:\n"
                 << (*lmc) << " with hashkey \""
                 << (*lmc).hashkey() << "\n and existing:\n"
                 << (*p).second << " with hashkey \"" << (*p).second.hashkey()
                 << "\"\n";
    }
    else {
      FILE_LOG() << "Brand new is this: " << (*lmc) << " with hashkey \""
                 << (*lmc).hashkey() << "\n";
      fastRet[hc] = *lmc;
      result->push_back(*lmc);
    }
  }
```

```
  delete outTable;
  FILE_LOG() << "A sample of the output follows.\n";
  printsome_helper("Pass60-after-remove-dups", result, -1); // 60
  FILE_LOG() << "-> " << result->size()
             << " LocationMinuteChannels resulting from pass60::RemoveDups.\n";
  return result;
}

// Based on Pass50::FindMissingChannelOutliers_WholeDay()
// Called from populate().
vector<LocationMinuteChannel> *Pass60::ExpandIntoKounts()
{
  FILE_LOG() << "Pass60::ExpandIntoKounts()\n";
  LOG_TAB_IN();
  vector<LocationMinuteChannel> *result = new vector<LocationMinuteChannel>;
  int numCDC = m_CountDayChannels_In->size();
  int numOutliersBADBINSZ = 0;
  int numOutliersNEGCT = 0;
  int numOutliersMDDROP = 0;
  string lastKey(""), lastCountID(""), skipCountID("");
  FILE_LOG() << "Reading over " << numCDC <<
    " records in the counts array resulting from pass50.\n";
  printsome_helper("Pass60", m_CountDayChannels_In, 60);
  bool isOutDay = false;
  vector<LocationMinuteChannel> *oneGroup =
    new vector<LocationMinuteChannel>;
  for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In->begin();
       cdc != m_CountDayChannels_In->end();
       cdc++) {
    FILE_LOG() << "cursor=" << (cdc - m_CountDayChannels_In->begin())
               << ", cdc: " << *cdc << endl;
    if (skipCountID.compare(cdc->m_CountID) == 0) {
      FILE_LOG() << "skipping cos it's equal to skipCountID.\n";
      continue;
    }
    if (lastCountID.compare(cdc->m_CountID) != 0) {
      // It's a new count for us - validate bin size.
      if (FindBadBins(cdc)) {
        numOutliersBADBINSZ++;
        skipCountID = cdc->m_CountID;
        P60L() << "skipping cos it's a BADBINSZ outlier.\n";
        continue;
      }
    }
    counts_type kounts;
    times_type times;
    ch_type channels;
    if (m_InPortal) FILE_LOG() << "cdc->m_CountID: \""
                               << cdc->m_CountID << "\"\n";
    string useCountID = m_InPortal ?
      // the next line, string oname = ..., is almost verbatim from pass40.cc.
      // TODO: make this .substr a method on CDC, use it also in pass40.cc.
      cdc->m_CountID.substr(0,
                            cdc->m_CountID.length()
                            - (PORTAL_STATION_ID_LEN + 1))
      : cdc->m_CountID;
    GetFonds().SelectTimesAndCounts(useCountID, kounts, times,
                                    channels);
    if (m_InPortal) {
      // add the other station's kounts, times.
      string otherStation = cdc->m_CountID.substr(cdc->m_CountID.length()
                                                  - PORTAL_STATION_ID_LEN,
                                                  cdc->m_CountID.length());
      string useCountIDStem
        = cdc->m_CountID.substr(0, cdc->m_CountID.length() -
                                (PORTAL_STATION_ID_LEN
                                 + 1 + PORTAL_STATION_ID_LEN));
      FILE_LOG() << "otherStation: \"" << otherStation << "\"\n";
      FILE_LOG() << "useCIDStem: \"" << useCountIDStem << "\"\n";
      string useCountID2(useCountIDStem);
      useCountID2 += otherStation;
      FILE_LOG() << "useCountID2: \"" << useCountID2 << "\"\n";
      counts_type kounts2;
      times_type times2;
      ch_type channels2;
      GetFonds().SelectTimesAndCounts(useCountID2, kounts2, times2,
                                      channels2);
      // We're on the second station, we force these channelIDs to be
      // 2.  See issue 0E6: portal, pass60 the STAC calls put every
      // single channelid to 1
      for (ch_type::iterator i = channels2.begin();
           i != channels2.end(); i++)
        (*i) = 2;
      copy(kounts2.begin(), kounts2.end(), back_inserter(kounts));
      copy(times2.begin(), times2.end(), back_inserter(times));
```

```
  copy(channels2.begin(), channels2.end(), back_inserter(channels));
}
vector<ktc> allofem;
for (unsigned int i = 0; i < kounts.size(); i++) {
  ktc tp;
  tp.kount = kounts[i];
  tp.time = times[i];
  tp.channel = channels[i];
  allofem.push_back(tp);
}
sort(allofem.begin(), allofem.end(), sortbytimechan);
P60L() << "STAC returned " << allofem.size() << " items.\n";
int cc = 0;
for (vector<ktc>::iterator i = allofem.begin();
     i != allofem.end(); i++, cc++) {
  ktc &item = (*i);
  P60L() << "at cc=" << cc << " (isOutDay=" << isOutDay
         << ", ktc=" << item << "):\n";
  P60L() << "result.size() now " << result->size()
         << " and group.size() now " << oneGroup->size()
         << "\n";
  if (VERBOSE_FOR_DUPS)
    cout << cc << ": " <<
      MakeTimesText2(&item.time, "%Y-%m-%d %H:%M:%S")
         << " in count/day " << (*cdc) << " which has date "
         << cdc->m_Date << "\n";
  // only look at rows with the current channel number.
  if (cdc->m_ChannelID != item.channel) {
    P60L() << "skipping cos this record's in the wrong channel.\n";
    continue;
  }
  // Only look at rows in the current date.
  string dateOfKount = MakeTimesText2(&item.time, "%Y-%m-%d") ;
  if (dateOfKount.compare(cdc->m_Date)) {
    P60L() << "skipping cos this record's in the wrong date.\n";
    if (VERBOSE_FOR_DUPS) {
      cout << "at index " << cc << ", ";
      cout << "date of kount: " << dateOfKount
           << " is different from the date of the CDC which is "
           << cdc->m_Date << "\n";
    }
    continue;
  }
  string nextDateOfKount;
  if ((i + 1) != allofem.end())
    nextDateOfKount = MakeTimesText2(&(i + 1)->time, "%Y-%m-%d") ;
  if (item.kount < 0) {
    isOutDay = true;
    // On a single incidence of a negative kount, we toss the
    // whole count (which is typically 48 hrs of counting in TDAT,
    // 24 hrs in PORTAL).  We arguably lose some good days of
    // counting but the code is /way/ simpler.
    GetFonds().MarkOutlier(&cdc->m_LocationID, &cdc->m_CountID,
                           NULL, NEGCT_DESC, true);
    numOutliersNEGCT++;
  }
  // If we didn't just reject it due to negative kount, maybe
  // we'll reject it cos of midday zeros, but only in PORTAL
  // cos TDAT is expected (tiny little roads sometimes) to have
  // midday zeros.
  if (m_InPortal && !isOutDay) {
    isOutDay = FindZKounts(item.time, item.kount, cdc);
    if (isOutDay) numOutliersMDDROP++;
  }
  if (isOutDay) {
    P60L() << "NOT stashing it, isOutday = " << isOutDay << "\n";
  }
  else {
    P60L() << "stashing it in the group.\n";
    LocationMinuteChannel lmc;
    lmc.m_LocationID = cdc->m_LocationID;
    lmc.m_CountID = cdc->m_CountID;
    lmc.m_Date = cdc->m_Date;
    lmc.m_Timestamp = item.time;
    lmc.m_ChannelID = item.channel;
    lmc.m_Kount = item.kount;
    lmc.m_NumChannels = cdc->m_NumChannels;
    oneGroup->push_back(lmc);
    P60L() << "  group size now " << oneGroup->size() << "\n";
  }
  // At day end or end of set, we decide whether to
  // preserve records for the day.
  if ((i + 1) == allofem.end() ||
      nextDateOfKount.compare(dateOfKount)) {
    P60L() << "we are at end of day.\n";
```

```
        if (cdc->m_NumChannels == 1 ||
            (cdc->m_NumChannels == 2 && item.channel == 2)) {
          if (isOutDay) {
            isOutDay = false;
            P60L() << "isOutDay was set, so we ignore the day.\n";
          }
          else {
            P60L() << "isOutDay was NOT set, so we keep the day.\n";
            P60L() << "before: oneGroup has " << oneGroup->size()
                   << " records, and result has " << result->size()
                   << " records.\n";
            copy(oneGroup->begin(), oneGroup->end(), back_inserter(*result));
            P60L() << "after: oneGroup has " << oneGroup->size()
                   << " records, and result has " << result->size()
                   << " records.\n";
            P60L() << "result.size() now " << result->size() << "\n";
          }
          delete oneGroup;
          oneGroup = new vector<LocationMinuteChannel>;
        }
      }
    } // for (int cc = 0; cc < numKounts && !isOutDay; cc++) {
  } // for (vector<CountDayChannel>::iterator cdc = m_CountDayChannels_In...
  delete oneGroup;
  P60L() << "result.size() at completion of big loop is "
         << result->size() << "\n";
  result = RemoveDups(result);
  LOG_TAB_OUT();
  cout << TAG_OUTL << "In " << GetFonds() << ",\n"
       << TAG_OUTL << "Pass60 saw " << numCDC
       << " count/day/channels, and of these\n"
       << TAG_OUTL << numOutliersBADBINSZ << " records were found to be"
       << " '" << BADBINSZ_DESC << "'-type outliers, and " << numOutliersNEGCT
       << "\n" << TAG_OUTL << "were found to be '" << NEGCT_DESC
       << "'-type outliers.\n";
  if (m_InPortal) {
    cout << TAG_OUTL << numOutliersMDDROP << " records were found to be"
         << " '" << MDDROP_DESC << "'-type outliers, which are only\n"
         << TAG_OUTL
         << "detected in PORTAL (fonds 2), and not in TDAT (fonds 1).\n";
  }
  else
    cout << TAG_OUTL << "This is not PORTAL, so we did not check for "
         << "zero counts in the midday.\n";
  FILE_LOG() << "A sample of the output follows.\n";
  printsome_helper("Pass60", result, 60);
  FILE_LOG() << "-> " << result->size()
             << " LocationMinuteChannels resulting from "
             << "pass60::ExpandIntoKounts.\n";
  return result;
}

void Pass60::populate()
{
  FILE_LOG() << "Pass60::populate()\n";
  LOG_TAB_IN();
  ConnectToDB();
  m_LocationMinuteChannels_Out = ExpandIntoKounts();
  tout << "Found " << m_LocationMinuteChannels_Out->size()
       << " location/minute/channels after "
       << "ExpandIntoKounts.\n";
#if (0)
  cout << "Random ten with mangled names:\n";
  for (int i = 0; i < 10; i++) {
    int which = random() % m_CountDayChannels_Out->size();
    cout << which << ": " << (*m_CountDayChannels_Out)[which] << "\n";
  }
#endif
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//  false: unable to validate the values in the database.  They are NOT GOOD.
bool Pass60::validate()
{
  bool result = false;
  return result;
#if (0)
  ConnectToDB();
  typedef vector<string> nc_type;
  nc_type dacount;
  // creating the dacount attribute creates several additional attributes,
  // which are counted together as a unit with the dacount attribute.
```

```
  stringstream dacountSelect;
  // GetTable() is the master table for the fonds, e.g. Nodeleg.
  dacountSelect << "select locid || ':' || countid || ':' || "
                << "countdate || ':' || "
                << "dacount || ':' || dacount_delta || ':' || dacount_n || "
                << "':' || rc || ':' || exceptType || ':' || numkounts from "
                << GetTable() << " order by "
                << "locid,countid,countdate";
  dacount = get_strlist(dacountSelect.str());
  stringstream allcnts;
  for (nc_type::const_iterator i = dacount.begin();
       i != dacount.end(); i++)
    allcnts << (*i) << "\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(allcnts.str(), knownGood);
  result = true;
  return result;
#endif
}



// FILE: pass70.cc
// AU: James E. Powell
// DATE: 2016-07-26
#include "tache.h"
#include "td0n-uml.h"
#include "countdaychannel.h"
#include "locationminutechannel.h"
#include "locationtimekounts.h"
#include <algorithm> // sort

using namespace std;
string
CHANCLONE_DESC("cloned channels (same data in two or more channels)");

string
NORC_DESC("the roadclass is null in GetRC() for this location");

string
LATEGAME_PARTIALDAY_DESC("A partial day was found in pass 70 "
                         "for this loc+count.  This should never happen. "
                         "Diagnose it please.");

Pass70::Pass70(bool doToy,
               const OneFonds &fonds,
               std::vector<LocationMinuteChannel> *theLMCs) :
  ChecksumAttribute(doToy,   // bool isToy,
                    fonds,    // const OneFonds &fonds,
                    true,    // bool attrInMaster,
                    fonds.GetMasterTableName()+string(NC_TBLNAM_POSTFIX)),
  m_LocationMinuteChannels_In(theLMCs),
  m_LocationTimeKounts_Out(NULL)
{
  tout << "Pass70 has been instantiated." << endl;
  // The short name is used here
  // ./checksumattribute.cc:125:        << GetShortName() << ".txt";
  // strictly to output the raw data for the checksum.
  SetShortName("pass-70");
  m_InPortal = (GetFonds().GetName().compare("PORTAL-TOTAL") == 0);
}

Pass70::~Pass70()
{
  delete m_LocationMinuteChannels_In;
}

typedef enum p70s_RCC {
  s_begin = 0,
  s_rec_kount = 1,
  s_do_fencepost = 2
} p70s_RCC;
```

```
typedef enum p70s_DR {
  s_dr_begin = 0,
  s_dr_rec_kount = 1,
  s_dr_do_fencepost = 2
} p70s_DR;

// Based on Pass50::FindMissingChannelOutliers_WholeDay()
// Called from populate().
vector<LocationMinuteChannel> *Pass70::RemoveChanClones()
{
  FILE_LOG() << "Pass70::RemoveChanClones()\n";
  LOG_TAB_IN();
  vector<LocationMinuteChannel> *result = new vector<LocationMinuteChannel>;
  int numLMC = m_LocationMinuteChannels_In->size();
  int numOutliersCHANCLONE = 0;
  LocationMinuteChannel modelLMC;
  bool hetFound = false;
  p70s_RCC state = s_begin;
  vector<LocationMinuteChannel> group;
  unsigned int kountRec[MAX_NCHAN] = {0,0}; // plan for 2 channels -
                                            // hardcoded limit today
                                            // 2016-07-28.
  // sort by LocID, timestamp [ascending], chan. [ascending]
  sort(m_LocationMinuteChannels_In->begin(),
       m_LocationMinuteChannels_In->end(),
       sortbylidtimechan);
  FILE_LOG() << "Reading over " << numLMC <<
    " records in the counts array resulting from pass60.\n";
  const char *titleNow = "Pass70::RemoveChanClones input";
  FILE_LOG() << "A sample of the input follows (please find it in "
             << "stdout under title \"" << titleNow << "\").\n";
  printsome_helper(titleNow,
                   m_LocationMinuteChannels_In, -1);
  for (vector<LocationMinuteChannel>::iterator lmc =
         m_LocationMinuteChannels_In->begin();
       lmc != m_LocationMinuteChannels_In->end();
       lmc++) {
    FILE_LOG() << "cursor=" << (lmc - m_LocationMinuteChannels_In->begin())
               << ", lmc: " << *lmc << endl;
    bool doAgain = true;
    while (doAgain) {
      doAgain = false;
      switch (state) {
      case s_begin:
        hetFound = false;
        group.clear();
        if (lmc->m_NumChannels == 1)
          result->push_back(*lmc);
        else {
          modelLMC = *lmc;
          state = s_rec_kount;
          doAgain = true;
        }
        break;
      case s_rec_kount:
        group.push_back(*lmc);
        assert(lmc->m_ChannelID <= MAX_NCHAN);
        assert(lmc->m_ChannelID >= 1);
        kountRec[lmc->m_ChannelID - 1] = lmc->m_Kount;
        // The code below is set to handle more than two channels
        // yet it's a natural fact today 2016-08-04 that all data
        // in fonds 1 and 2 has been normalized to 1 or two channels,
        // and 1-channel recordings never get past state==s_begin.
        if (lmc->m_NumChannels != 2) {
          FILE_LOG() << "Strange case of mixed channels for cid "
                     << lmc->m_CountID << ".\n"
                     << "Advise caution.  See issue OF1: dup cids "
                     << "(multiple cids for one loc+date) exist = add "
                     << "new outlier class.\n";
        }
        // We expect these two-channel records to come ordered as chan
        // 1, chan 2, chan 1, chan 2 etc.  So, when the channel of this
        // row is 2 (= lmc_m_NumChannels) it's a sign that if we don't
        // have heterogenity in the signal (e.g. all channels have same
        // data) yet then we never will.
        if (!hetFound && lmc->m_ChannelID ==
            static_cast<int>(lmc->m_NumChannels)) {
          unsigned int expectKount = kountRec[0];
          FILE_LOG() << "Reached end of recordings for this timeslice.\n";
          FILE_LOG() << "expectKount has been set to " << expectKount << ".\n";
          for (int i = 1; i < static_cast<int>(lmc->m_NumChannels) &&
                 !hetFound; i++) {
            FILE_LOG() << "Index " << i << ": " << kountRec[i] << "\n";
            if (kountRec[i] != expectKount) {
```

```
            FILE_LOG() << "... heterogenity!\n";
            hetFound = true;
          }
        }
        FILE_LOG() << "Done checking for hetero in this time slice.  Result:"
                   << hetFound << ".\n";
      }
      if ((lmc + 1) == m_LocationMinuteChannels_In->end()) {
        FILE_LOG() << "The next recording is the very last one. "
                   << "Going to fencepost.\n";
        state = s_do_fencepost;
        doAgain = true;
      }
      else if (!(lmc + 1)->IsInDayGroup(modelLMC)) {
        // To avoid being labeled as an outlier, some record over
        // the entire day must be different between channels 1 and
        // 2.
        FILE_LOG() << "The next recording is not in the day group "
                   << "of the current modelLMC.  Going to fencepost.\n";
        state = s_do_fencepost;
        doAgain = true;
      }
      break;
    case s_do_fencepost:
      if (hetFound) {
        FILE_LOG() << "At fencepost, we found heterogenity.  Copying "
                   << group.size() << " items on over.\n";
        copy(group.begin(), group.end(), back_inserter(*result));
      }
      else {
        FILE_LOG()
          << "At fencepost, no heterogenity found.  "
          << "Reporting dup chan outlier.\n";
        GetFonds().MarkOutlier(&lmc->m_LocationID, &lmc->m_CountID,
                               NULL, CHANCLONE_DESC, true);
        numOutliersCHANCLONE += group.size();
      }
      state = s_begin;
      break;
    } // switch (st) {
  } // while (doAgain) {
} // for (vector<LocationMinuteChannel>::iterator lmc = m_LocationMinu...
LOG_TAB_OUT();
FILE_LOG() << "-> " << result->size() << " LocationMinuteChannels.\n";
cout << TAG_OUTL << "In " << GetFonds() << ",\n"
     << TAG_OUTL << "Pass70 saw " << numLMC
     << " location/minute/channels, and of these\n"
     << TAG_OUTL << numOutliersCHANCLONE << " records were found to be"
     << " '" << CHANCLONE_DESC << "'-type outliers.\n";
const char *titleNow2 = "Pass70::RemoveChanClones output";
FILE_LOG() << "A sample of the output follows (please find it in stdout "
           << "under title \"" << titleNow2 << "\").\n";
printsome_helper(titleNow2, result, 60);
return result;
}

vector<LocationTimeKounts> *
Pass70::DataReduction(vector<LocationMinuteChannel> *in)
{
  FILE_LOG() << "Pass70::DataReduction()\n";
  LOG_TAB_IN();
  // sort by LocID, timestamp;
  sort(in->begin(), in->end(), sortbylidtime);
  vector<LocationTimeKounts> *result = new vector<LocationTimeKounts>;
  int numLMCs = in->size();
  LocationMinuteChannel modelLMC;
  p70s_DR state = s_dr_begin;
  unsigned int accumKount = 0;
  FILE_LOG() << "Reading over " << numLMCs <<
    " records in the counts array resulting from pass60.\n";
  const char *titleNow = "Pass70::DataReduction input";
  FILE_LOG() << "A sample of the input follows (please find it "
             << "in stdout under title \"" << titleNow << "\").\n";
  printsome_helper(titleNow, in, 60);
  int numInHourGroup = 0;
  for (vector<LocationMinuteChannel>::iterator lmc = in->begin();
       lmc != in->end();
       lmc++) {
    FILE_LOG() << "cursor=" << (lmc - in->begin())
               << ", lmc: " << *lmc << endl;
    bool doAgain = true;
    while (doAgain) {
      doAgain = false;
      switch (state) {
      case s_dr_begin:
```

```
      modelLMC = *lmc;
      accumKount = 0;
      state = s_dr_rec_kount;
      doAgain = true;
      numInHourGroup = 1;
      break;
    case s_dr_rec_kount:
      accumKount += lmc->m_Kount;
      if ((lmc + 1) == in->end()) {
        FILE_LOG() << "This is last one, stashing the result.\n";
        state = s_dr_do_fencepost;
        doAgain = true;
      }
      else if ((lmc + 1)->IsInHourGroup(modelLMC)) {
        numInHourGroup ++;
      }
      else {
        cout << "nihg: " << numInHourGroup << "\n";
        FILE_LOG()
          << "Next one is not in the hour group.  Stashing the result.\n";
        state = s_dr_do_fencepost;
        doAgain = true;
      }
      break;
    case s_dr_do_fencepost:
      // this assertion will fire if dups are found as in issue
      // "0F3: hour group identity test puts timestamps in the
      // same hour group into different groups".
      if (numInHourGroup != 4 && numInHourGroup != 8) {
        cerr << "Warning: numInHourGroup == " << numInHourGroup
             << ".  You'll have partial days in the CSV output.  Diagnose, "
             << "or drop as outlier.  See tache issue 101: PORTAL hour "
             << "group starts at 15 minutes in.  I am marking it as an "
             << "outlier to warn future sailors.\n";
        GetFonds().MarkOutlier(&lmc->m_LocationID, &lmc->m_CountID,
                               NULL, LATEGAME_PARTIALDAY_DESC, true);
        assert(0);
      }
      else {
        result->push_back(LocationTimeKounts(modelLMC.m_LocationID,
                                             modelLMC.m_Timestamp,
                                             accumKount));
        FILE_LOG() << "Result stashed.\n";
      }
      state = s_dr_begin;
      break;
    } // switch (st) {
  } // while (doAgain) {
} // for (vector<LocationMinuteChannel>::iterator lmc = m_LocationMinu...
LOG_TAB_OUT();
const char *titleNow2 = "Pass70::DataReduction output";
FILE_LOG() << "A sample of the output follows (please find it "
           << "in stdout under title \"" << titleNow2 << "\").\n";
printsome_helper(titleNow2, result, 60);
return result;
}

// Pass70::PopulateRC could take too long, and it's undeniably more
// efficient to do this way back in pass 1 when life is simple and the
// whole output is just a list of locations.  However, then we would have
// to propagate the RC value down through all of the other passes which
// I can't be bothered to do right now.  Phht.
void Pass70::PopulateRC()
{
  FILE_LOG() << "Pass70::PopulateRC()\n";
  int numTossed = 0;
  vector<LocationTimeKounts> *newKounts = new vector<LocationTimeKounts>;
  for (vector<LocationTimeKounts>::iterator i =
         m_LocationTimeKounts_Out->begin();
       i != m_LocationTimeKounts_Out->end(); i++) {
    string useLoc = m_InPortal ?
      i->m_LocationID.substr(0,4) : i->m_LocationID;
    bool notFound = false;
    bool isNull = false;
    int useRC = GetFonds().GetRC(useLoc, &notFound, &isNull);
    assert(!notFound);
    if (isNull) {
      GetFonds().MarkOutlier(&i->m_LocationID, NULL,
                             NULL, NORC_DESC, true);
      numTossed++;
    }
    else {
      i->m_RC = useRC;
      newKounts->push_back(*i);
    }
```

```
  }
  delete m_LocationTimeKounts_Out;
  m_LocationTimeKounts_Out = newKounts;
  cout << TAG_OUTL << "Tossed " << numTossed << " of "
       << m_LocationTimeKounts_Out->size()
       << " entries due to roadclass being null for their location.\n";
  cout << TAG_OUTL << newKounts->size() << " entries remain.\n";
  FILE_LOG() << "-> void\n";
}

void Pass70::populate()
{
  FILE_LOG() << "Pass70::populate()\n";
  LOG_TAB_IN();
  ConnectToDB();
  m_LocationTimeKounts_Out = DataReduction(RemoveChanClones());
#if (ATTEMPT_EMIT_RC)
  PopulateRC();
#endif
  tout << "Found " << m_LocationTimeKounts_Out->size()
       << " location/minute/channels after DataReduction.\n";
  // print the set.  This is in theory a highly useful product:
  // squeaky clean due to all of the work in this and other passes.
  // ~/projects/tache/src $ (random)
  //  1241967739153956173
  // I use 56173 as the sole special code for the final, final output
  // of tache.  I got it from RNG, as run just above on 2016-08-03.
  printsome_helper("TD0N output", m_LocationTimeKounts_Out, -1, 56173);
#if (0)
  cout << "Random ten with mangled names:\n";
  for (int i = 0; i < 10; i++) {
    int which = random() % m_CountDayChannels_Out->size();
    cout << which << ": " << (*m_CountDayChannels_Out)[which] << "\n";
  }
#endif
  LOG_TAB_OUT();
  FILE_LOG() << "-> void\n";
}

// returns:
//   true: the values in the database have been validated GOOD.
//
//   false: unable to validate the values in the database.  They are
//          NOT GOOD.
bool Pass70::validate()
{
  bool result = false;
  return result;
#if (0)
  ConnectToDB();
  typedef vector<string> nc_type;
  nc_type dacount;
  // creating the dacount attribute creates several additional attributes,
  // which are counted together as a unit with the dacount attribute.
  stringstream dacountSelect;
  // GetTable() is the master table for the fonds, e.g. Nodeleg.
  dacountSelect << "select locid || ':' || countid || ':' || "
                << "countdate || ':' || "
                << "dacount || ':' || dacount_delta || ':' || dacount_n || "
                << "':' || rc || ':' || exceptType || ':' || numkounts from "
                << GetTable() << " order by "
                << "locid,countid,countdate";
  dacount = get_strlist(dacountSelect.str());
  stringstream allcnts;
  for (nc_type::const_iterator i = dacount.begin();
       i != dacount.end(); i++)
    allcnts << (*i) << "\n";
  string knownGood;
  if (GetIsToy())
    knownGood = "unknown";
  else {
    if (GetFonds().GetName().compare("TDAT") == 0)
      knownGood = "f4a9c356cb746942c45f183bb744b4f4e10059d2327494e257";
    else if (GetFonds().GetName().compare("PORTAL-FHWA") == 0)
      knownGood = "04f6cc64c960c37b3aec4a713460b8f9f931167dae00bc773";
    else if (GetFonds().GetName().compare("PORTAL-TOTAL") == 0)
      knownGood = "89471ee3b510da222fc2997ca4e4c47a089317e96e66a021a";
    else
      cerr << "unknown fonds name in dacountsattribute::validate"
        " (=no cert in tache/docs/certs for this fonds)" << endl;
  }
  result = generic_validate(allcnts.str(), knownGood);
  result = true;
  return result;
#endif
```

```
}



// FILE: countdaychannel.h
// AU: James E. Powell
// DATE: 2016-07-26
#pragma once
#include "tdOn-uml.h"
// similar to inline std::ostream& operator<<(std::ostream& a,
// const LocationMinuteChannel &b) in locationminutechannel.h
inline std::ostream& operator<<(std::ostream& a, const CountDayChannel &b)
{
  return a << "[a CountDayChannel with lid = " << b.m_LocationID
           << " and cid " << b.m_CountID << ", date " << b.m_Date
           << ", ChannelNo = " << b.m_ChannelID
           << " and numChan = " << b.m_NumChannels << "]";
}

// sortbylidciddatechan used in pass50
// "comparison function object (i.e. an object that satisfies the
// requirements of Compare) which returns âĂŃtrue if the first argument
// is less than (i.e. is ordered before) the second" (devhelp for c++,
// deb 8 cppreference-doc-en-html)
inline bool sortbylidciddatechan(const CountDayChannel &b1,
                                 const CountDayChannel &b2)
{
  // sort the input list by:
  // lid + cid + date + channelno
  FILE_LOG() << "Compare_cdc b1: " << b1 << "\n";
  FILE_LOG() << "Compare_cdc b2: " << b2 << "\n";
  int res = b1.m_LocationID.compare(b2.m_LocationID);
  if (res == 0) {
    res = b1.m_CountID.compare(b2.m_CountID);
    if (res == 0) {
      res = b1.m_Date.compare(b2.m_Date);
      if (res == 0) {
        // sort in increasing order
        res = b1.m_ChannelID < b2.m_ChannelID;
      }
    }
  }
  return res > 0;
}



// FILE: locationminutechannel.h
// AU: James E. Powell
// DATE: 2016-07-26
#pragma once
#include "tdOn-uml.h"
// similar to inline std::ostream& operator<<(std::ostream& a, const
// CountDayChannel &b) in countdaychannel.h.
inline std::ostream& operator<<(std::ostream& a,
                                const LocationMinuteChannel &b)
{
  timestamp useStamp = b.m_Timestamp;
  return a << "[a LocationMinuteChannel with lid = " << b.m_LocationID
           << " and cid " << b.m_CountID << ", date " << b.m_Date
           << ", ChannelNo = " << b.m_ChannelID
           << " and numChan = " << b.m_NumChannels << ", timestamp "
           << MakeTimesText2(&useStamp, "%Y-%m-%d %H:%M:%S")
           << ", kount: "
           << b.m_Kount << "]";
}

// sortbylidciddatechan used in pass70 based on
// countdaychannel.h::inline bool sortbylidciddatechan(const
// CountDayChannel &b1,...
//
// "comparison function object (i.e. an object that satisfies the
// requirements of Compare) which returns âĂŃtrue if the first argument
// is less than (i.e. is ordered before) the second" (devhelp for c++,
// deb 8 cppreference-doc-en-html)
inline bool sortbylidtime(const LocationMinuteChannel &b1,
                          const LocationMinuteChannel &b2)
{
  // sort the input list by:
  // lid + timestamp
#if (0)
  // (very) spammy -> 100 GB for a full run
  FILE_LOG() << "Compare_lmc b1: " << b1 << "\n";
  FILE_LOG() << "Compare_lmc b2: " << b2 << "\n";
#endif
```

```
  int res = b1.m_LocationID.compare(b2.m_LocationID);
  if (res == 0)
    res = b1.m_Timestamp < b2.m_Timestamp;
  return res > 0;
}

// sort by LocID, timestamp [ascending], chan. [ascending]
inline bool sortbylidtimechan(const LocationMinuteChannel &b1,
                              const LocationMinuteChannel &b2)
{
  // sort the input list by:
  // lid + timestamp
#if (0)
  FILE_LOG() << "Compare_lmc b1: " << b1 << "\n";
  FILE_LOG() << "Compare_lmc b2: " << b2 << "\n";
#endif
  int res = b1.m_LocationID.compare(b2.m_LocationID);
  if (res == 0) {
    if (b1.m_Timestamp == b2.m_Timestamp)
      res = b1.m_ChannelID < b2.m_ChannelID;
    else
      res = b1.m_Timestamp < b2.m_Timestamp;
  }
  return res > 0;
}


// sort by LocID, cid. [ascending], timestamp [ascending], channel
inline bool sortbylidcidtimechan(const LocationMinuteChannel &b1,
                                 const LocationMinuteChannel &b2)
{
  // sort the input list by:
  // lid + timestamp
#if (0)
  FILE_LOG() << "Compare_lmc b1: " << b1 << "\n";
  FILE_LOG() << "Compare_lmc b2: " << b2 << "\n";
#endif
  int res = b1.m_LocationID.compare(b2.m_LocationID);
  if (res == 0) {
    res = b1.m_CountID.compare(b2.m_CountID);
    if (res == 0) {
      if (b1.m_Timestamp == b2.m_Timestamp)
        res = b1.m_ChannelID < b2.m_ChannelID;
      else
        res = b1.m_Timestamp < b2.m_Timestamp;
    }
  }
  return res > 0;
}



// FILE: kounttimechannel.h
// AU: James E. Powell
// DATE: 2016-09-13
#pragma once

class ktc {
 public:
  int kount;
  timestamp time;
  int channel;
};

// similar to inline std::ostream& operator<<(std::ostream& a,
// const LocationMinuteChannel &b) in locationminutechannel.h
inline std::ostream& operator<<(std::ostream& a, const ktc &b)
{
  timestamp theTime = b.time;
  return a << "[a ktc with kount = " << b.kount
           << " and time " << MakeTimesText2(&theTime, "%Y-%m-%d %H:%M:%S")
           << ", channel " << b.channel << "]";
}

// sortbylidciddatechan used in pass60, based on
// countdaychannel.h::inline bool sortbylidciddatechan(const
// CountDayChannel &b1,...  "comparison function object (i.e. an
// object that satisfies the requirements of Compare) which returns
// âĂŃtrue if the first argument is less than (i.e. is ordered before)
// the second" (devhelp for c++, deb 8 cppreference-doc-en-html)
inline bool sortbytimechan(const ktc &b1,
                           const ktc &b2)
{
  // sort the input list by:
  // lid + timestamp
```

```
  //  FILE_LOG() << "Compare_ktc b1: " << b1 << "\n";
  //  FILE_LOG() << "Compare_ktc b2: " << b2 << "\n";
  int res;
  if (b1.time == b2.time) {
    if (b1.channel == b2.channel)
      res = 0;
    else
      res = b1.channel < b2.channel;
  }
  else
    res = b1.time < b2.time;
  return res > 0;
}



// FILE: locationminutechannel.cc
// AU: James E. Powell
// DATE: 2016-07-26
#include "tache.h"
#include "locationminutechannel.h"
#include <iostream>

using namespace std;

string LocationMinuteChannel::hashkey()
{
  std::string binTimeOfKount = MakeTimesText2(&m_Timestamp,
                                              "%Y-%m-%d %H:%M:%S");
  std::stringstream key;
  key << m_LocationID << ":"
    //      << m_CountID << ":"
      << binTimeOfKount << ":"
      << m_ChannelID;
  std::string theStr = key.str();
  return theStr;
}

// hashcode is used strictly to identify duplicate rows in the output
// of pass6.
int LocationMinuteChannel::hashcode()
{
  std::string binTimeOfKount = MakeTimesText2(&m_Timestamp,
                                              "%Y-%m-%d %H:%M:%S");
  std::stringstream key;
  key << m_LocationID << ":"
      << m_CountID << ":"
      << binTimeOfKount << ":"
      << m_ChannelID;
  std::string theStr = key.str();
  int strLen = theStr.length();
  int result = 0;
  for (int i = 0; i < strLen; i++)
    result += static_cast<int>(theStr[i]);
  cout << "hashed key \"" << theStr << "\' to code " << result << "\n";
  return result;
}

bool LocationMinuteChannel::IsInDayGroup (const LocationMinuteChannel & b)
{
  bool result = true;
  FILE_LOG() << "LocationMinuteChannel::IsInDayGroup(this="
             << (*this) << ", b=" << b << ")\n";
  LOG_TAB_IN();
  if (m_LocationID.compare(b.m_LocationID))
    result = false;
  //  else if (m_CountID.compare(b.m_CountID))
  //    result = false;
  else if (m_Date.compare(b.m_Date))
    result = false;
  LOG_TAB_OUT();
  FILE_LOG() << "-> " << result << "\n";
  return result;
}

bool LocationMinuteChannel::IsInHourGroup (const LocationMinuteChannel & b)
{
  bool result = true;
  FILE_LOG() << "LocationMinuteChannel::IsInHourGroup(this="
             << (*this) << ", b=" << b << ")\n";
  LOG_TAB_IN();
  if (!IsInDayGroup(b))
    result = false;
  else {
    timestamp tsA = m_Timestamp;
```

```
    timestamp tsB = b.m_Timestamp;
    string hrA = MakeTimesText2(&tsA, "%H");
    string hrB = MakeTimesText2(&tsB, "%H");
    if (hrA.compare(hrB))
      result = false;
  }
  LOG_TAB_OUT();
  FILE_LOG() <<"-> " << result<< "\n";
  return result;
}



// FILE: locationtimekounts.h
// AU: James E. Powell
// DATE: 2016-07-26
#pragma once
#include "td0n-uml.h"

inline LocationTimeKounts::LocationTimeKounts() {}

inline LocationTimeKounts::LocationTimeKounts (std::string locID,
                                               timestamp theTimeStamp,
                                               unsigned int theAccumKount)
  : m_LocationID(locID), m_Timestamp(theTimeStamp), m_Kounts(theAccumKount)
{
}

// similar to inline std::ostream& operator<<(std::ostream& a, const
// CountDayChannel &b) in countdaychannel.h.
inline std::ostream& operator<<(std::ostream& a,
                                const LocationTimeKounts &b)
{
  timestamp useStamp = b.m_Timestamp;
#if (0)
  return a
    << "[a LocationTimeKounts with lid = ")
    << b.m_LocationID
    << ", timestamp "
    << MakeTimesText2(&useStamp, "%Y-%m-%d %H:%M:%S")
    << ", kounts: "
    << b.m_Kounts << "]";
#else
//
return a
<< b.m_LocationID
#if (ATTEMPT_EMIT_RC)
<< ","
<< b.m_RC
#endif
<< ","
<< MakeTimesText2(&useStamp, "%Y-%m-%d %H:%M:%S")
<< ","
<< b.m_Kounts << "\n";
#endif
}
```