12-2021

How To Train Your Algo: Investigating the Enablers of Bias in Algorithmic Development

Marta Stelmaszak Rosa
Portland State University, stmarta@pdx.edu

Citation Details

Stelmaszak, M. (2021) How To Train Your Algo: Investigating the Enablers of Bias in Algorithmic Development. International Conference on Information Systems, 12-15 December 2021.

How To Train Your Algo: Investigating the Enablers of Bias in Algorithmic Development

Completed Research Paper

Marta Stelmaszak
The School of Business
Portland State University
stmarta@pdx.edu

Abstract

Literature on algorithmic bias identifies its source in either biased data or statistical methods, more rarely in the development of algorithmic solutions as a potential factor. Because of the prior unknowability of algorithms, data scientists developing such solutions have to take various design decisions. Drawing from the flow-oriented approach, we study algorithmic unknowability and how data scientists respond to it in 35 public data science Jupyter notebooks containing algorithmic solutions to predict customer churn in a credit card dataset on a data science platform Kaggle.com. We offer a more thorough understanding of the unknowability in algorithmic development that can enable algorithmic bias: resource, problem, dataset, analytical, model, and performance unknowability. We find that in response, data scientists engage in bias-enabling interpretation, bias-enabling optionalizing, and bias-enabling experimentation. These findings contribute to literature on algorithmic bias and can help avert bias earlier in practice.

Keywords: Algorithms, algorithmic bias, machine learning, artificial intelligence, flow of action, information systems development, data science, algorithmic development

Introduction

"A kind of 'magic' happens between inputting data to train, validate, and test algorithms and receiving the final output data, usually in the form of discrete classifications or continuous probabilities", stated Baum and Haveman recently (2020, p. 270) in their editors' comments on the future of organizational theory. Researchers increasingly point out, however, that this algorithmic 'magic' may end up being a 'curse' in the form of algorithmic bias that occurs when "the outputs of an algorithm benefit or disadvantage certain individuals or groups more than others without a justified reason for such unequal impacts" (Kordzadeh and Ghasemaghaei 2021, p. 1). Algorithmic bias may result in adverse consequences for individuals, organizations, and society, and as such its effects are studied increasingly more often to avert negative impacts, but its sources are still not fully understood (Favaretto et al. 2019; Gal et al. 2018; Ransbotham et al. 2016; Tarafdar et al. 2020).

Information systems (IS) with its focus on socio-technical aspects is well positioned to contribute to knowledge on algorithmic bias by providing a better understanding of the mechanisms that generate it (Ransbotham et al. 2016). So far, the sources of algorithmic bias have been commonly identified in algorithm's inputs, that is biased data, statistical methods, and more recently - algorithm design (Barocas and Selbst 2016; Favaretto et al. 2019; Kordzadeh and Ghasemaghaei 2021). However, our current understanding of how bias may be introduced during algorithm design is partial and limited, pointing towards more social and organizational factors that can impact the development process (van den Broek et al. 2021; Ghasemaghaei et al. 2018; Ghasemaghaei and Turel 2021; Joshi 2020). Instead, we propose to turn to the technical side of algorithms to investigate how it contributes to enabling bias in the process of developing algorithmic solutions.

Algorithmic solutions, a term we use to refer to a wide range of information systems that deploy algorithms to process input data and produce computed outputs, are developed by data scientists, professionals skilled in computer science, programming, and development (Vaast and Pinsonneault 2021). The development of algorithmic solutions (henceforth algorithmic development, AD) opens up various possibilities to introduce bias because of their prior unknowability. We argue and demonstrate that such solutions are characterized by algorithmic unknowability of the relationships between inputs and outputs not only in run-time (Zhang et al. 2021), but also in design-time, or development. Data scientists charged with AD have to deal with this unknowability during development through exploratory programming (Kery et al. 2017; Kery and Myers 2017), and their subjective, individual responses may introduce bias to algorithm design. However, so far we have limited knowledge of design-time algorithmic unknowability and how data scientists specifically respond to it. To address this gap, we studied 35 public data science Jupyter notebooks containing algorithmic solutions developed to predict customer churn in a credit card dataset on a popular data science and machine learning platform Kaggle.com (Dissanayake et al. 2015; Mangal and Kumar 2016). Referring to a common problem faced by many companies and often tackled by algorithmic solutions, the credit card dataset attracted over 200 notebooks with code and comments describing attempts to best predict customer churn. Studying the flow of development action, including code, code execution and computational narratives, in Jupyter notebooks (Baygi Mousavi et al. 2021), we identify the precise sources of algorithmic unknowability and action that emerges in response. Focusing then on the role of data scientists in this action, we isolate three modes of response that they deploy: bias-enabling interpretation, bias-enabling optionalizing, and bias-enabling experimentation.

We build on current literature by identifying the technical roots of algorithmic unknowability, outlining a comprehensive set of discrete actions that take place during developing algorithmic solutions, and showing how the modes of data scientists' responses to unknowability may open up opportunities to introduce bias. In practice, our findings can help data scientists and managers identify the parts of the development process that may lead to the introduction of bias, and thus increase the likelihood of averting algorithmic bias. A better understanding why, how and where bias can occur in algorithmic design can help address the problem (Silva and Kenney 2019).

Background Literature

Algorithmic solutions are commonly perceived as a neutral source of objective knowledge derived nearly autonomously from datasets, as they help "to overcome many of the irrationalities and biases that plague domain experts, thereby providing more objective knowledge" (van den Broek et al. 2021, p. 7). In this perspective, human data scientists play only an instrumental role and should even be eliminated for the sake of efficiency and increasing objectivity (Davenport and Harris 2017). Similarly, AD is seen as a rational endeavor that follows a fixed, established process of deriving neutral insights from objective, neutral data, akin to the rational design paradigm in software engineering (Ralph 2018), or the perception of systems development as instrumental reasoning based on "naïve realism" where the developer is the expert who takes managers' objectives "and turns them into a constructed product, the system. Management dictates the ends; the developers use specific means to achieve the ends" (Hirschheim and Klein 1989, p. 1203).

Despite these firm convictions regarding algorithmic objectivity, a growing body of literature recognizes the issues of algorithmic bias, that is the fact that the outputs of an algorithmic solution may "benefit or disadvantage certain individuals or groups more than others without a justified reason for such unequal impacts" (Kordzadeh and Ghasemaghaei 2021, p. 1). This is widely perceived as detrimental and there is ongoing research that aims at identifying the consequences of algorithmic bias on individuals, organizations, and society (Kordzadeh and Ghasemaghaei 2021). Noting that this phenomenon is socio-technical in nature, researchers point out that especially IS can contribute to uncovering the antecedents of bias (Kordzadeh and Ghasemaghaei 2021; Ransbotham et al. 2016), as "for engineers and policymakers alike, understanding how and where bias can occur in algorithmic processes can help address it" (Silva and Kenney 2019, p. 37). For this reason, the understanding what mechanisms can generate algorithmic bias has been identified as a central challenge in IS research (Ransbotham et al. 2016). Below we provide an overview of literature on the sources of algorithmic bias, focusing on the role of the data scientist in AD, and we draw from exploratory programming as a response to algorithmic unknowability in AD.

Introducing Bias into Algorithmic Solutions

Three main potential enablers of bias, that is places where bias can be introduced into algorithmic solutions, have been discussed. First, a sizable amount of work focuses on identifying algorithm input, that is training data, as a potential source of bias (Barocas and Selbst 2016; Favaretto et al. 2019; Kordzadeh and Ghasemaghaei 2021): if biased data are used to train algorithms, they are likely to offer equally biased outputs (Tarafdar et al. 2020). Second, statistical discrimination has been identified as a contributor to bias, that is to say that algorithms, like all statistical methods, only represent objects via mathematical proxies, and so "adverse outcomes against protected classes might occur involuntarily due to the classification system" (Favaretto et al. 2019, p. 12).

While these two potential enablers of algorithmic bias are fairly well studied with established solutions proposed (Kordzadeh and Ghasemaghaei 2021), it is the third source that is most relevant to this paper: algorithm design. During AD, choices and decisions need to be made regarding the features to be used, weights attached to them, and objective functions that all impact how algorithms process inputs and the outputs they produce (Barocas and Selbst 2016; Kordzadeh and Ghasemaghaei 2021). Others note that the entire process of AD is dependent on data scientists' design choices: "insofar as the data scientist needs to translate a problem into formal computer coding, deciding on the target variable and the class labels is a subjective process" (Favaretto et al. 2019, p. 12). Going even further, "human subjectivity is at the very core of the design of data mining algorithms since the decisions regarding which attributes will be taken into account and which will be ignored are subject to human interpretation [12], and will inevitably reflect the implicit or explicit values of their designers [1]" (Favaretto et al. 2019, p. 17). Thus, literature recognizes that as long as data scientists are involved in AD, they will make subjective decisions and choices that impact algorithmic solutions and may open up possibilities for introducing bias.

Recent IS studies investigating the work of data scientists poignantly show their engagement in and impact on AD. For example, Ghasemaghaei and colleagues (Ghasemaghaei et al. 2018; Ghasemaghaei and Turel 2021) discuss the practices of knowledge hiding that data scientists engage in, thus impacting how algorithmic outputs are used in organizations by evasive hiding, playing dumb, and rationalized hiding, that all have varying effects on decision-making (Ghasemaghaei and Turel 2021). Joshi (2020) investigated how the practices of data scientists in the banking industry rely on both subjectivity and objectivity in the production of information, emphasizing the role of data scientists' choices at various stages of the analysis process, for example choices regarding algorithms, the use of variables to be included, methods of evaluation, and similar. In an ethnography of how a machine learning-based algorithmic solution was developed, van den Broek and colleagues (2021) show how knowledge produced within such systems relies on a hybrid practice combining machine learning and domain expertise: data scientists and subject matter experts need to reflect on developing algorithmic solutions for producing knowledge in the course of a mutual learning process that takes place during AD. Thus, even this nascent research shows that data scientists introduce their own subjectivities into the process. However, these findings are focused more on the social and organizational aspects that require data scientists to take decisions and make choices influencing the design, while little is still known about the technical side of algorithmic solutions that requires these choices and decisions.

Algorithmic Unknowability in Development

As with other autonomous tools, algorithmic solutions are characterized by algorithmic unknowability (Zhang et al. 2021), that is the *ex ante* and *ex post* unknowability of the input-output relationships. As we argue, this unknowability applies not only to run-time, but also to design-time, which we refer to as AD. Algorithmic unknowability in development means that during the process, the input-output relationships are unknowable *ex ante* to the developer: the data scientist does not know what outputs will be produced by each input component of the solution under development until such outputs are presented to her. As a basic example, the data scientist does not know whether there are any missing values in the dataset that need to be dealt with before running code to check for missing values and evaluating the results – the outcome of each command in code is only known after the code is executed, and only then the next design decision can be made. In AD then, "human actors and technology artifacts interact to produce specific cognitive outcomes" (Zhang et al. 2021, p. 7): decisions and choices that shape development. This unknowability is furthered by the fact that algorithms are non-deterministic which makes it difficult to

know the outcomes in advance (Seidel et al. 2018; Zhang et al. 2021), and the exact resources, such as computing power needed to run algorithms, are also unknowable in advance (Zhang et al. 2021). As such, AD is characterized by design-time unknowability, where the effects of development action are emergent, the relationships between inputs and outputs in the development process are unknown, and multiple potential trajectories are possible at every decision point.

It is for the reasons of algorithmic unknowability that AD differs from more traditional forms of IS development, as it does not have clear ends or goals that are knowable *a priori*. AD falls under exploratory programming, where there are no clear requirements for the code at the onset, and there is a broad space of possible solutions (Kery et al. 2017; Kery and Myers 2017), it is "a programming task in which a specific goal or a means to that goal must be discovered by iteratively writing code for multiple ideas" (Kery et al. 2017, p. 1). As such then, AD has open-ended goals and relies on exploring different approaches while the goal evolves based on insights, differently from traditional programming (Subramanian et al. 2019).

The Impact of Unknowability on Development

As a result of algorithmic unknowability, data scientists engage in exploratory programming in AD (Hill et al. 2016; Kery et al. 2017). Exploratory programming is key under circumstances of flexibility, discovery, and innovation, and its defining characteristic is "the practice of designing the goal *at the same time* as experimenting in code" (Kery and Myers 2017, p. 25). Data scientists write code to prototype or experiment with different ideas to obtain insight from data, but they do not engineer working code to match a pre-defined specification – instead, the goal is open-ended and evolves together with the evolution of code (Kery and Myers 2017). This kind of work is highly iterative, exploratory and non-linear (Patel et al. 2008), and relies heavily on the professional judgment and decisions made by data scientists (Rule et al. 2018).

Current research into the work of data scientists who engage in exploratory programming reveals the extent to which they have to draw on their own, subjective, human judgment in making various decisions in AD: "insights are sensitive to the methods used to produce them; small changes in how data are collected, cleaned or processed can lead to vastly different results" (Rule et al. 2018, p. 1). As data scientists engage in obtaining, cleaning, profiling, analyzing and interpreting data (Rule et al. 2018), they may be tasked with data merging and cleaning, sampling, feature selection, defining metrics, building predictive models, defining ground truth, hypothesis testing, and even applying insights or models to business (Kim et al. 2016). At every step, they "try different versions of the same analysis, slowly improve analytical methods, and hit numerous 'dead ends' before finding an explanation that 'fits' the data" which "can make it difficult to perform an 'objective' analysis" (Rule et al. 2018, p. 2), while data scientists must be clear and transparent in their reasoning "if others are to understand, and ultimately trust their work" (Rule et al. 2018, p. 1).

Current literature points towards algorithm design as a potential enabler of bias in algorithmic solutions, suggesting that human subjectivity, judgment and decisions in this process may introduce various biases. While most IS literature in this area focuses on the social and organizational factors that shape data scientists' engagement in AD, little is known about the technical aspect of algorithmic solutions that requires the developers to draw on their own judgment in making development decisions and choices. We argue that algorithmic unknowability inherent in such solutions requires data scientists to engage in exploratory programming in response, but we do not yet know enough about *the sources of unknowability in developing algorithmic solutions and how data scientists respond to it*.

Theoretical Framework

We adopt the flow-oriented approach and vocabulary, as suggested in a recent MISQ paper proposing the flow of action as a new lens to study continuous socio-technical transformation in a fluid and dynamic digital world (Baygi Mousavi et al. 2021): we want to 'think movement'. The flow-oriented approach focuses on becoming over time, tracing ongoing action and its results in time. The phenomena are understood as ongoingly (trans)forming accomplishments. This perspective encourages the focus on evolution, becoming, diversity, flow, movement, creativity and conditionality, among other aspects, and puts stability, spatiality, planning and actors into the background (Baygi Mousavi et al. 2021). The flow-oriented approach draws attention to the impact of contingencies, unpredictability, seeming insignificance of various flows of action in the explanations of how and why transformations happen. In IS, this for example entails a shift from

users and systems towards the temporal flows of using and computing (Baygi Mousavi et al. 2021; Yoo 2010), or a move from understanding software as static assemblies of modules and features to "continuous meandering flows of computational services or experiences—experiences made possible through confluences along unfolding trajectories of languages, frameworks, devices, data streams, web services, APIs, but also those of careers, habits of use, trends and fads, regulation, best practices, and so forth" (Baygi Mousavi et al. 2021, p. 15). The flow-oriented approach invites us to investigate the trails along which action flows (Baygi Mousavi et al. 2021). The flow of action cannot be broken up without affecting the phenomenon, as phenomena emerge from "the creative absorption of previous trajectories of action and their ongoing weaving into ever-new paths" (Baygi Mousavi et al. 2021, p. 16). Trajectories of flowing action unfolding along trails create "conditions of possibility" for further action in a certain direction (Baygi Mousavi et al. 2021, p. 16).

We appreciate that this is a novel approach to study IS phenomena, but we believe it is particularly fitting to the study of developing algorithmic solutions, not only because it foregrounds action, but primarily because it allows to zoom in on unknowability and its impact on subsequent action. The nascent literature on AD and exploratory programming confirms that there is a significant degree of contingencies, unpredictability and seemingly insignificant action that can have big impacts on the development of algorithms. This is in line with seeing algorithmic solutions as ongoingly (trans)forming accomplishments that become over time, based on the conditions of possibility revealed in action. In other words, we see algorithmic solutions as emerging in the flow of development action. We describe how this theoretical framework informed our research design next.

Research Design

Kaggle.com is a popular platform for data scientists and machine learning engineers where they can develop and improve their skills, as well as participate in corporate-sponsored competitions by addressing a variety of problems related to datasets published. Kaggle.com, part of Alphabet Inc, allows to upload datasets, set specific tasks and create interactive Jupyter notebooks where users can develop algorithmic solutions. Kaggle.com was selected as a setting because of its public availability and openness in sharing notebooks that allows an unprecedented access to the design of algorithmic solutions. Others have used Kaggle.com for research purposes as well (Dissanayake et al. 2015; Mangal and Kumar 2016).

The dataset we selected for this study is a well-regarded and popular one, containing the details of around 10,000 credit card customers of a bank, whereby a portion of customers churned. The goal is to identify, based on 18 variables such as age, salary, credit card limit and similar, what makes a customer churn (give up a credit card) to be able to predict customers at risk of churning in the future, as well as to identify the variables that are most predictive of the risk of churn ("Kaggle.Com" 2021). When we investigated the dataset, there were around 210 notebooks submitted that contained algorithmic solutions to this dataset, with constant daily activity in existing notebooks and new notebooks being added. Kaggle.com users can contribute with their proposed algorithmic solutions by creating and working on Jupyter notebooks. Jupyter notebooks are a popular format used among data scientists and machine learning engineers to develop, share and display their algorithmic solutions. The notebooks allow combining code and user-generated computational narratives of the steps taken and the findings, and can be easily shared with elements of code executed, that is for example with embedded diagrams or results. Segments of code can be re-executed at later stages in the Jupyter environment. A typical Kaggle.com notebook is hosted in Google cloud and gives users access to Google-sponsored processing resources.

We selected an open and public dataset rather than a competition because the majority of notebooks submitted for competitions are private and thus visible only to sponsor companies, and competitions are usually very specific and limit the variety of potential algorithmic solutions developed. In contrast, public notebooks allow good access to notebooks containing fairly unrestricted solutions with much more experimentation. From the datasets available on Kaggle.com, we selected the credit card customers dataset because it is related to a common problem that many organizations face, and it is a problem that is often tackled by developing algorithmic solutions, thus it is a good representative sample of what researchers in IS and management would consider of interest.

Data Collection

In January and February 2021, we collected 57 Jupyter notebooks that were created using the credit card customer dataset in Python as the programming language. The notebooks were arranged from the 'hottest' (a measure used on Kaggle.com to define notebooks with most activity, edits and highest votes by the community, *Kaggle.Com*, 2021), and thus those collected were considered among the 'hottest' at the time. We decided to select the 'hottest' notebooks as these were assessed as high quality by the community, thus were likely to contain well-developed algorithmic solutions. We discarded notebooks in R to eliminate differences in programming languages, and notebooks that contained only partial solutions, for example only analyzed data without building actual models. Using a feature available on Kaggle.com, we downloaded all of the selected notebooks and converted them to PDF documents to analyze them in nVivo. The PDF documents contained all code, code execution, and computational narratives. In total, we collected 841 pages of code, code execution and narrative in the PDF format (on average 24 pages per notebook) and 1,692 segments of code (on average 35 per notebook) consisting of several lines of code each.

Notebooks are appropriate a source of data to study the flow of action because they document the way action takes place, allowing for the study of the flow of action, accounting for code, its execution, and computational narratives, illuminating how both the technological artifact and the human actor (Zhang et al. 2021) interact to produce decisions and choices. Further, the sequential nature in which notebooks detail AD allows to study how actions flow from one another and how they create "conditions of possibility" for further action. This foregrounds the unknowability of algorithmic solutions and actions in response. The presence of computational narratives allows to focus on the role of data scientists, as in them developers explain, justify, and keep track of design decisions (Rule et al. 2018; Subramanian et al. 2019). Notebooks are used to provide explanatory annotations, but also give a narrative structure that explains and justifies choices (Kery et al. 2018).

Data Analysis

Bagyi Mousavi and colleagues offer some methodological suggestions regarding setting up a study that adopts the flow-oriented approach (2021). They encourage to focus on tracing the flows of action in answering how and along which lines phenomena are brought into being and carry on trans(forming), how and along which lines is the phenomenon flowing, what is the story of the happening of X (p. 46, 48, 52). Drawing from these suggestions, we set out to study action in AD by uncovering how and along which lines this developing is flowing, and what actions bring algorithmic solutions into being.

Methodologically, we adopted an action as a unit of analysis, where an action is the unfolding of steps directed at accomplishing a specific task, identified in code, code execution, and accompanying computational narrative. Once action towards accomplishing a specific task is completed, it enables the unfolding of ensuing action, and so on, with the main flow of action directed towards developing an algorithmic solution. We bracketed action we identified into stages commonly present in data science development (Kurgan and Musilek 2006), that is 1) Preparing the environment, 2) Reading in data, 3) Cleaning data, 4) Exploratory data analysis, 5) Pre-processing the dataset, 6) Building and training the model, 7) Testing and validating the model. In other words, we encountered these stages in the data and used them to group action we identified.

Stage	**Stage start**	**Action unfolding in the stage**			**Stage end**
Reading in data	Headline: "*Meeting the data*"	Action aimed at loading the data used into the notebook environment and understanding its shape to clean it appropriately			Following headline
Action	**Action start**	**Code**	**Code execution**	**Narrative**	**Action end**
Loading data	Computational narrative	*data = pd.read_csv('../input/ credit-card-customers/Bank Churners.csv')*	File attached to the notebook	*Lets open the data and see what we have*	Following code segment concerns a different task
Inspecting data	Computational narrative	*data.shape*	(10127, 23)	*Lets see the shapes of the data so we know*	Following code segment

				what we are dealing with	concerns a different task

Table 1. Example of identifying Loading data and Inspecting data action from Notebook_009

We proceeded by inductively coding the notebooks to identify action as defined above and exemplified in Table 1. Because of the inductive nature of our study, we oscillated between data analysis and further data collection. After coding the first 35 notebooks, we began to identify only action we had already coded for, and the subsequent 10 notebooks did not add any new action. At this point we decided to stop coding and analyzing the notebooks as we reached the point of saturation, arriving at 32 discernible actions. Subsequently, we developed sequences of development action in each notebook, as shown in Table 2.

Sequence of action	1.1 1.2 1.3 1.4	2.1 2.2	3.3 2.2 3.1 3.5	4.3 4.1 4.4 4.2 4.4 4.2 4.4 4.2 4.4 4.3
Stage	1 Preparing the environment	2 Reading in data	3 Cleaning data	4 Exploratory data analysis
Sequence of action	5.3 5.1 5.4 5.2	6.2 6.6 6.3 6.4	7.1 7.2 7.2	6.5 6.5 5.1 5.4 5.2 6.2 6.3 6.4
Stage	5 Pre-processing the dataset	6 Model building	7 Model validation	6 Model building
Sequence of action	7.1 7.2 7.4 7.2	6.2 6.6 6.4 6.3 6.4	7.1 7.2 7.4 7.2 6.5	6.2 6.6 6.4 6.3 6.4
Stage	7 Model validation	6 Model building	7 Model validation	6 Model building
Sequence of action	7.1 7.2 7.4 7.2 6.5 7.5			
Stage	7 Model validation			

Table 2 Example of a flow of action in Notebook_008 with a detailed sequence of action in stages

This provided us with a schematic representation of the flow of development action in every notebook. We report on our findings regarding the unknowability in development and how data scientists responded to it below.

Algorithmic Development Under Unknowability

In this section, we present the stages in developing an algorithmic solution with identified sources of unknowability, as well as responses to these types of unknowability evidenced in action. Algorithmic unknowability was evident in the notebooks we studied, exemplified by the fact that only the business problem was stated, that is the need to be able to predict customer attrition based on the available data, and when input data were obtained, the specifics of the final solution were unknown, they emerged in the flow of action.

Stage 1: Preparing the Environment Under Resource and Problem Unknowability

In the first step in AD, the programming environment is set up, which entails creating a new Jupyter notebook on Kaggle.com with a few clicks and initiating a new processing session. This is required to set up the notebook so that AD can start. Since all processing happens in the cloud, each notebook has some processing capacity allocated, with the maximum session lime of 9 hours, maximum of 19.6 GB disk space, maximum of 16 GB RAM and an indication of CPU usage in percentage, and an accelerator can be added at any point to increase processing capacity, with a choice of GPU and TPU v3-8. The kind of resources and their level is unknown, and thus this stage is characterized by resource unknowability. Next, a choice of programming language needs to be made, with Python as a default and the option of selecting R. Access to the internet, for example for loading data held online, can be enabled. At the point where these decisions are made, problem unknowability is also evident – the exact specifics of the problem and how it will be defined, e.g. the dependent variable, are unknown.

To deal with these sources of unknowability, data scientists set up the processing environment as defined by Kaggle.com, and have some choice over the parameters for the notebook to satisfy estimated processing needs, for example by enabling an accelerator, and they also select the programming language of their choice. Because of both resource and problem unknowability, in many notebooks data scientists import libraries, that is software packages up front, drawing on their experience and knowledge of packages that proved to be useful in the past. By default, Jupyter notebooks on Kaggle.com come equipped with some starter code that recommends importing some packages: *"# This Python 3 environment comes with many helpful analytics libraries # It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python # For example, here's several helpful packages to load*" (Notebook_002).

Finally, to deal with problem unknowability, we found that data scientists often identify the problem and the prediction task by re-stating it in the computational narrative: "*In this notebook we will try to find the most important reasons that a customer would churn and also devise multiple models that would predict churning customers*" (Notebook_001) that sometimes extends into a comprehensive summary of the contents of the notebook (e.g. Notebook_012). In this action, data scientists begin to interpret the problem in programming terms to identify the dependent variable.

Stage 2: Reading in Data Under Dataset Unknowability

In this stage, action is aimed at loading the data into the notebook and understanding the contents of the dataset. At this point, the data scientist may know the dataset somewhat from the description on Kaggle.com, but its exact specifics, such as data types and basic statistics, are unknown. In response to this prior dataset unknowability, we found that first data is loaded, which involves attaching data to the notebook environment, executed in a single line of code: *data=pd.read_csv("/kaggle/input/credit-card-customers/BankChurners.csv")* (Notebook_001), and then inspected: "*Meeting the data. Lets open the data and see what we have*" (Notebook_009), as data scientists need to familiarize themselves with the variables and values in the dataset. Checking data types is aimed at identifying what data types variables are stored in is essential to inform further data cleaning (e.g. "*There are 6 categorical and 14 numeric features.*", Notebook_022). Descriptive statistics are sometimes obtained to get a basic statistical summary of the dataset, also for the purposes of confirming the dataset is of the expected shape and informing the need of further data cleaning, e.g. "*There is a big gap between max and min values. This situation strengthens the presence of outliers*." (Notebook_022). In some cases, preliminary observations and conclusions are drawn from this summary that inform later exploratory data analysis, e.g. "*Some observations from the table: • Majority of clients are married. • Almost all clients have Blue Card (~%93)*" (Notebook_022).

Stage 3: Cleaning Data Under Dataset Unknowability

At this stage, action is aimed at removing unnecessary or unneeded elements from the dataset to enable further processing. Similar to the previous stage, here dataset unknowability is present and requires appropriate action: some elements need to be removed or changed, but it is not possible to know what interventions are needed in advance. In response, we found that in some notebooks action includes renaming variables, that is changing the names of variables to ease referring to them: "*Columns title is TO LONG, let's rename it*." (Notebook_012), removing unnecessary variables: *data=data.iloc[:,:-2]#deleting last two rows as mentioned in database* (Notebook_001), or removing outliers, that is rows in the dataset that contain values too far from the mean that could skew the model: "*credit limit, avg oepn to buy, total_trans_amt seems to have outliers*" (Notebook_002). These actions take place in some notebooks (renaming variables – 4, removing variables – 33, removing outliers – 2 notebooks), but sometimes are omitted or take place at different stages.

Dealing with missing values, that is checking for and removing missing values from the dataset is required as prediction models do not work with missing values. With large datasets, it cannot be known in advance if there are any missing values, so it is only after executing code to check for missing values that it can be known: "*We are off to a great start as there appears to be no missing values!*" (Notebook_016), "*there are no null/missing values SIGH!*" (Notebook_002), "*LUCKY! NO need to handle with NaN or missing data*" (Notebook_012). Otherwise, missing values have to be dealt with, for example by means of imputation, that

is artificial generation of data points to fill in the gaps, as suggested e.g. in. Notebook_030. The second action that allows the flow of AD to move forward is transforming data types, that is changing data types (from e.g. categorical) into types suitable to be used in model building: *data[data.select_dtypes(['object']).columns] = data.select_dtypes(['object']). → apply(lambda x: x.astype('category'))* (Notebook_001). Categorical data include for example marital status or education level, which are often coded to 0 (married) or 1 (single), and thus they cannot be used in model building in this format: "*Machine learning algorithms work best with numerical data. However, in our dataset, we have some categorical columns. These columns contain data in textual format; we need to convert them to numeric columns*." (Notebook_012). There are several different ways of dealing with categorical variables: "*These are the categorical variables, we will now either do encoding or make dummy so as to make them all numerical so that we can plot out heatmap and proceed further*" (Notebook_003).

Stage 4: Exploratory Data Analysis Under Analytical Unknowability

The exploratory data analysis stage concerns action aimed at learning insights from data to be used to guide model building. The whole of action at this stage is underpinned by analytical unknowability, that is the lack of prior knowability of the relationships between variables and their usefulness in developing the algorithmic solution. In response, AD usually starts with analyzing the dependent variable, where familiarizing with the predicted variable takes place so that data scientists can understand which independent variables may help predict it: *data.Attrition_Flag.value_counts()* (Notebook_001). The understanding of the dependent variable enabled by code execution informs other steps needed in pre-processing the data later: "*We can see that there are very few rows for 'attrited customer' we might need to oversample the data*" (Notebook_002). Unknowability is also countered by visualizing data, that is producing visual representations of independent variables. These visualizations inform which independent variables may be predictive and thus should be included in the model in later episodes. Visualizing data was the single most common action in the notebooks, appearing in as many as 203 code segments in the notebooks analyzed. Usually, multiple plots, graphs and charts are generated based on imported visualization libraries: *"[Data] is analyzed through visual exploration to gather insights about the model that can be applied to the data, understand the diversity in the data and the range of every field. We use a bar chart, box plot, distribution graph, etc. to explore each feature varies and its relation with other features including the target feature*" (Notebook_012). Some notebooks contain narratives describing main learnings from data visualization: "*it is analyzed through visual exploration to gather insights about the model that can be applied to the data, understand the diversity in the data and the range of every field. We use a bar chart, box plot, distribution graph, etc. to explore each feature varies and its relation with other features including the target feature*" (Notebook_012).

Analyzing independent variables can be conducted via other means than visual, for example by comparing counts *df['Income_Category'].value_counts()* (Notebook_002) and drawing conclusions from numerical analysis: "*We can see that there are more number of female who uses the card than males*" (Notebook_002). Identifying correlations often takes place at this stage, where action is aimed at finding out which variables correlate with the dependent variable: "*lets make a heatmap so as to get, whats the correlation b/w every other columns. But as we have categorical columns as well, so we will use integer encoding so make them numerical and then make heatmap*" (Notebook_003).

Stage 5: Pre-processing the dataset under model unknowability

Action at this stage concerns preparing the dataset for model building by giving it the right shape, and this stage is characterized by model unknowability: since the exact algorithm that will be deployed in the model built in the solution and its features and requirements are not yet known, action here needs to prepare the dataset to work well with a wide range of potential algorithms. Scaling data can be conducted to change the scale of variables to make sure that they do not influence the model unduly just because their unit. Scaling can sometimes include normalization, that is enforcing standard distribution on each variable: "*Next, we normalize numerical so that each feature has mean 0 and variance 1 using Standar Scaler*" (Notebook_005). Scaling is performed using appropriate libraries and executed using few lines of code. Another possible action is data resampling, that is increasing or decreasing the number of observations in one of the classes (in this case, attired customers) that is in a disproportionate minority, thus creating a balanced dataset. This is also performed using a common library package: "*To balance the dataset we use*

SMOTE which stands for Synthetic Minority Oversampling Technique" (Notebook_005). However, these actions are not uniformly present across all notebooks: we observed scaling data in 15 notebooks, and resampling data in 10.

While previous actions are optional and appear variably across notebooks, it is required at this stage to remove the target variable that should be predicted: "*Our data is now ready, and we can train our machine learning model. But first, we need to isolate the variable that we're predicting from the dataset*" (Notebook_012). It is essential for this to take place as otherwise the model is not predictive, since the target variable is known to the model. Usually, the dataset containing independent variables is named X, and the dataset containing the target variable - y, or very similar. A common action concerns the formatting of the dataset, that is adding or removing variables (other than the target variable which needs to be removed) to prepare the final dataset for model building. It can include removing other unnecessary variables, and putting together the dataset if it had been separated for example to transform categorical variables.

This stage ends with the action of splitting the dataset, that is dividing it into the training and test datasets that will enable model building and testing, respectively. This is conducted using a standard library and executed in one line of code: *X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, // → random_state=42)* (Notebook_001), where the dataset of predictive, independent variables X and the dataset of the target, dependent variable y (in this case: whether customer attired or not) is split into four datasets, with 70% of the data used for training (X_train and y_train datasets) and 30% of the dataset saved for later testing (X_test, y_test), where these numbers vary between notebooks.

Stage 6: Building and Training the Model Under Model Unknowability

Action at this stage is aimed at building a model based on a selected algorithm and training it on the training dataset to obtain a model capable of predictions. At this stage, prior model unknowability is evident: AD starts without a specific model pre-defined, that is the algorithm, features, weights and parameters emerge in development. Since the dataset we studied contained a supervised classification problem, all models used in the notebooks deployed algorithms applicable to this type of a predictive problem. The types of algorithms we observed were some simpler and common ones, such as logistic regression, a k-nearest-neighbour classifier, Naïve Bayes or decision tree classifier, to more advanced classifiers such as AdaBoost, Bernoulli NB, Extra Trees Classifier, Gradient Boosting Classifier, LGBM Classifier, Principal Component Analysis, Random Forest Classifier, Support Vector Classifier or XGBoost classifier. Two notebooks experimented with using a neural network for this task. Algorithm selection was always indicated by importing the selected algorithm's library, sometimes preceded with a narrative headline introducing the selected algorithm, and on few occasions, notebooks contained longer explanations of the selection: "*We are going to predict those customers who will churn using random forests which are widely used for classifiction problem like this one*" (Notebook_039). Regardless of the type of the algorithm selected, all these were implemented using pre-packaged functions, thus selecting the algorithm required importing the relevant library.

The next action is model training, that is applying the selected algorithm to the training dataset to build a model that maps the relationships between independent variables and the dependent variable, where these relationships are unknowable a priori. This action is described in detail in the narrative in Notebook_012: "*Now, we'll use a machine learning algorithm that will identify patterns or trends in the training data. This step is known as algorithm training. We'll feed the features and correct output to the algorithm; based on that data, the algorithm will learn to find associations between the features and outputs. After training the algorithm, you'll be able to use it to make predictions on new data. (…) To train this algorithm, we call the fit method and pass in the feature set (X) and the corresponding label set (y)*". In all notebooks studied, algorithms are deployed by importing them from pre-existing libraries. For example, in Notebook 001 we observe from *sklearn.linear_model import LogisticRegression // logmodel = LogisticRegression()*, whereby the algorithm for logistic regression is imported from the *sklearn* library in one line of code, and then invoked as *logmodel* in another line of code. Thus, *logmodel* is an instance of the logistic regression algorithm that is created in the notebook. In the next line of code, this model is trained: *logmodel.fit(X_train_pd,y_train)* – the instance of the logistic regression algorithm is "fit" to the training dataset. After this operation, the model is trained, that is it learned how the predictive (independent) variables (X) are correlated with the target (dependent) variable (y).

There are several optional actions at this stage. For example, in some notebooks creating the pipeline takes place first, that is coding a sequence of steps that can be executed repeatedly for different models, for example: "*Pipeline Steps: One Hot Encoding Quantile Proccesing Fit the model*" (Notebook_009). This means that a function is created that will repeat these steps (in Notebook_009: transforming data, formatting the dataset, fitting the model) with various models passed into the function. Another optional action is setting parameters, that is selecting the desired algorithm hyperparameters for model training. This action takes place especially with algorithms that are more sophisticated and those that actually deploy machine learning to uncover the best parameters. A typical example of parameter setting is a simpler case of specifying the number of nearest neighbors in the K-nearest neighbors classifier: *knnmodel = KNeighborsClassifier(n_neighbors=3)* (Notebook_001), in this case set to 3. With more complex algorithms, setting parameters is the first step to deploying machine learning: "*Now that we know the best architecture for the neural network, we can set these params for our model and get to comparing different models*" (Notebook_037). Another optional action is selecting features, that is identifying the variables that the model(s) trained base their predictions on. This is done to leave in variables that are strongly predictive and remove less predictive variables that may be decreasing the performance of the model. When taking place at this stage, selecting features is not based on the exploratory data analysis or conducted manually. Rather, a model is trained (usually a Random Forest Classifier or similar) solely for the purposes of obtaining "*features importances*", that is an indication of which features the model relied on the most in its classification.

Setting parameters is often present together with another optional action, parameter tuning, where algorithm parameters are changed to train alternative model(s) based on the same algorithm but with different parameters. For example, for a Random Forest Classifier, such parameters may include the number of features, the number of levels in the tree, and the minimum values for splitting the trees. This usually takes the form of a function that iterates over code with changing the values to train a number of models with changed parameter values: *#decision tree pruning max_depth=[] acc_gini=[] acc_entropy=[] for i in range(1,50,5): // dtree=DecisionTreeClassifier(criterion='gini',max_depth=i) dtree.fit(X_train,y_train)//pred=dtree.predict(X_test)//acc_gini.append(accuracy_score(y_test,pred))//dtree=DecisionTreeClassifier(criterion='entropy',max_depth=i)dtree.fit(X_train,y_train)//pred=dtr ee.predict(X_test) // acc_entropy.append(accuracy_score(y_test,pred)) max_depth.append(i)* (Notebook_002). In this case, depth, Gini coefficient and entropy coefficient are modified in the range from 1 to 50 by 5 points, and respective models are trained, tested, and stored for model performance comparison. Parameter tuning is a type of experimentation, that is selecting and applying a set of different parameters to train a different model on the training dataset in search of better performance. Running experiments in this way was less common in the notebooks than training different algorithms, and included for example selecting features with the highest importance: "*Now we look for the features with higher importance, to run a new Random Forest using only some of the most important ones*" (Notebook_005).

Stage 7: Testing and Validating the Model Under Performance Unknowability

Action aimed at testing the built model on the test dataset and verifying its performance to confirm its predictive capacity takes place in stage seven, and is a result of performance unknowability, that is the fact that the actual results and performance of the models built and trained cannot be known in advance. In response, various action takes place. Predicting on unseen data relies on applying the trained model to the test dataset that was retained earlier to obtain predictions as to the target, dependent variable. This is customarily done in one line of code, e.g. *ypred = model_1.predict(X_test)* (Notebook_017), whereby the trained model *model_1* is applied to *X_test* dataset to yield predictions saved in *ypred*. The results of this prediction are then used to assess the performance of the model by comparing the predicted values of the target, dependent variable y stored in *ypred* with actual values of y that were retained and not used in training or testing.

Evaluating model performance involves calculating performance measures according to selected criteria to obtain a measure of the model performance. All notebooks contain a way to evaluate model performance. This is usually done both for the training and test dataset, albeit it is commonly accepted to use the performance results on the test dataset as final. The most common measure of model performance in the notebooks is accuracy, that is how many times the model predicted the correct classification of the dependent variable in comparison to the known but withheld labels in the test dataset. The second common

measure in this dataset is recall, that is how often the model correctly identified churning customers – this measure is particularly relevant to the problem set in the dataset. Sometimes models are evaluated on the time it took to train them. Evaluating model performance is done using pre-packaged functions, most often imported from *sklearn*. In some notebooks, this scene has some narrative: "*Then using accuracy_score and the confusion matrix to evaluate the models performance*" (Notebook_004). Comparing model performance, that is comparing the calculated performance measure between alternative models trained to identify the best performing model, is an action taking place where various models were trained. In some notebooks, this comparison is conducted across a number of metrics. Also optionally, this action may contain visualizing results, where visual representations of the model(s) performance are displayed, often to select the best performing model: "*Before that making any decisions let's make one more graph, but for another metric: recall (which is important for us)*" (Notebook_029) for the comparison of multiple notebooks. In some notebooks, we identified one final action: indicating the best performing model, where the narrative included a clear statement of the model identified as best performing, e.g. "*We can see that combining the features generated with PCA with the others is what gives the better results, this can be due to higher degrees of freedom for the model*" (Notebook_028). A large number of notebooks do not make any comments on the best performing model, either because there is only one model trained or because the numerical results are considered as self-explanatory.

The Modes of Responses to Unknowability

As discussed above, we identified various sources of algorithmic unknowability in developing algorithmic solutions: resource, problem, dataset, analytical, model, and performance unknowability. Various actions take place in response to this unknowability, as summarized in Table 3 below. In response to unknowability, action is characterized by modes of data scientists reacting to it: unknowability is inherently present in algorithmic design, yet design action needs to proceed to develop an algorithmic solution, and thus a way of reasoning through, accepting or minimizing unknowability needs to be deployed. We call these data scientists' modes of responses, and we identified three modes: interpreting, optionalizing, and experimenting. The deployment of these modes of response is required for AD to continue, as only this way current action can take place, and thus enable the conditions of possibility for the following action.

Interpreting is a mode of dealing with unknowability by reasoning through it, and it is deployed to tackle problem, dataset, analytical, and performance unknowability. When engaging in interpreting, data scientists rely on and draw from their own judgment and understanding to interpret outputs, which informs the next action. This is evident in computational narratives that allow data scientists to explain their thinking: "*In my personal opinion, this variable shows the maximum number of consecutive months of inactivity and a customer is classified as churned after 6 or 7 months of inactivity*" (Notebook_032), or "*A key assumption made at this point is that any column relative to a time (e.g. months inactive), for leavers, is reflective of their tenure rather than a fixed point. If the latter is true, differences in these columns may be due to timing (for instance, if months inactive counts the last twelve months, and a customer left 8 months ago,their inactivity would be high by default)*" (Notebook_028). In these two cases, in response to dataset unknowability, data scientists analyzed code execution, reasoned through what was unknowable, and interpreted the variables based on their own judgment when inspecting data and analyzing independent variables, respectively.

Optionalizing as a mode of dealing with output unknowability rests on accepting algorithmic unknowability and taking decisions regarding AD from optional choices – since resource, dataset, and model unknowability cannot be ever fully eliminated, decisions regarding AD are made from a variety of choices. In this mode, various decisions and choices are often explicated in computational narratives, and their subjectivity is either left without a comment: "*These are the categorical variables, we will now either do encoding or make dummy so as to make them all numerical so that we can plot out heatmap and proceed further*" (Notebook_003) when transforming data types, or acknowledged as such: "*The main reason I enjoy using this library is because it given me a starting point as to where I should start my data analysis*" (Notebook_007), when importing libraries. Optionalizing is also present in how the notebooks differed with respect to actions being deployed, namely we found scaling data only in 15 notebooks, resampling data in 10 notebooks, and removing outliers in 2 notebooks.

We identified experimenting as a mode of responding to model unknowability. The goal of developing an algorithmic solution in the case we studied was to design the most accurate classification model. Since the exact model to be built, that is the algorithm, weights, parameters and features used, that would yield the best performance was unknown in advance, data scientists responded to this unknowability by minimizing it through trying out various approaches to modeling. In all but one notebook, after the first model was trained, attempts have been made to achieve a better outcome through running experiments that involve selecting other algorithms, selecting features, or parameter tuning and training other models on the same dataset: "*After fitting the train data, we can see the results of using the different combinations of parameters*" (Notebook_004). In notebooks that we investigated, it was not uncommon for up to 7 experiments to be conducted.

Stage and type of unknowability	Action in response to unknowability	Mode of response
Stage 1: Preparing the environment **Resource unknowability:** computational resources needed to develop the algorithmic solution are unknown before the work begins **Problem unknowability:** the formulation of the problem the algorithmic solution is supposed to tackle in programming terms is unknown up front	- Setting up the environment: deploying the default computational resources in the cloud environment - Selecting the programming language: choosing a flexible programming language to accommodate potential future needs - Identifying the problem: restating and clarifying the problem to be tackled to ensure the appropriate dependent variable is identified - Importing libraries: importing a variety of software packages that may be useful	Optionalizing: taking decisions regarding the steps in the algorithmic development process from optional choices Interpreting: relying on own judgment and understanding to interpret algorithmic inputs informing further work
Stage 2: Reading in data **Dataset unknowability:** the exact contents and specific characteristics of the dataset are unknown in advance	- Loading in data: attaching data to the notebook - Inspecting data: familiarizing with the variables and values in the first few rows of the dataset to confirm the shape of the dataset - Checking data types: identifying what data types variables are stored in to inform the need for data cleaning - Obtaining descriptive statistics: familiarizing with the basic statistics summarizing the dataset to inform the need of data cleaning	Interpreting: relying on own judgment and understanding to interpret algorithmic inputs informing further work
Stage 3: Cleaning data **Dataset unknowability:** the exact contents and specific characteristics of the dataset are unknown in advance	- Dealing with missing values: checking for and potentially removing missing values in the dataset to enable modelling - Renaming variables: changing the default names of variables to different names to ease referring to variables - Removing outliers: checking for and potentially removing rows from the dataset that are too far from the mean to eliminate the risk of skewing the model - Transforming data types: changing data types to those that can be used in model building	Optionalizing: taking decisions regarding the algorithmic development process from optional choices
Stage 4: Exploratory data analysis **Analytical unknowability:** the lack of prior knowability of the relationships between variables and their usefulness in developing the algorithmic solution	- Analyzing the dependent variable: numerically familiarizing with the target variable that the model should predict to uncover independent variables may be predictive - Analyzing independent variables: numerically familiarizing with variables that can possibly help predict the target variable to know which should be included in the model - Identifying correlations: identifying which variables correlate with the dependent variable and thus may be predictive to include them in the mode - Visualizing data: producing visual representations of variables to learn which may be relevant and need to be included in the model	Interpreting: relying on own judgment and understanding to interpret algorithmic inputs informing further work
Stage 5: Pre-processing the dataset **Model unknowability:** the exact model to be built, including the algorithm, weights, parameters and features, is unknown in advance	- Scaling data: changing the scale of variables to ensure that none of them has an undue bearing on the model - Resampling data: increasing or decreasing the number of observations to ensure the dataset is balanced in terms of observations - Formatting the dataset: adding or removing variables to prepare the final dataset for model building - Splitting the dataset: dividing the dataset into training and test dataset to enable model building	Optionalizing: taking decisions regarding the steps in the algorithmic development process from optional choices

Stage 6: Building and training the model **Model unknowability:** the exact model to be built, including the algorithm, weights, parameters and features, is unknown in advance	- Creating the pipeline: coding a sequence of steps that are executed repeatedly for different models - Selecting the algorithm: choosing the algorithm to be used in the model to be trained - Setting parameters: selecting the desired algorithm parameters for the model(s) to be trained - Model training: applying the selected algorithm to the training dataset to build a model of relationships between independent variables and the dependent variable - Running experiments: selecting and applying different parameters to the algorithms to build alternative models in search of better performance - Selecting features: identifying the features that the model(s) base their predictions on most strongly and excluding the less predictive ones to improve predictive powers - Parameter tuning: changing the algorithm parameters for alternative model(s) to be trained to improve prediction results	Experimenting: trying out various approaches to modeling to experiment with obtaining the best result
Stage 7: Testing and validating the model **Performance unknowability:** actual results and performance of the models built and trained cannot be known in advance	- Predicting on unseen data: applying the trained model to the test dataset to obtain predictions - Evaluating model performance: calculating performance according to the selected criteria to obtain a statement of model performance - Comparing model performance: comparing the calculated performance between alternative models trained to identify the best performing model - Visualizing results: producing visual representations of alternative models' performance to select the best performing model - Indicating the best performing model: identifying the model that is characterized by the best performance for use in later prediction on new datasets	Interpreting: relying on own judgment and understanding to interpret algorithmic inputs informing further work

Table 3. Types of unknowability, actions deployed in response and their modes

Discussion

This paper was motivated by the need to uncover the sources of unknowability in developing algorithmic solutions and data scientists' responses to this unknowability. We identified six types of unknowability present in AD: resource, problem, dataset, analytical, model, and performance unknowability. We were able to focus on algorithmic unknowability drawing from the flow-oriented approach that emphasizes contingencies, unpredictability and the role of seemingly insignificant actions in how various phenomena emerge. Consequently, we identified actions that take place during development and investigated how these actions present a response to unknowability. Since human actors and technology artifacts interact to produce cognitive outcomes (Zhang et al. 2021) in algorithmic development, we isolated data scientists' modes of response to unknowability from the actions: reasoning through unknowability by interpreting, accepting its presence by optionalizing, and minimizing it by experimenting.

The modes of response allow us to explain how bias can make its way into AD. While none of the actions themselves are inherently biased, the modes of response to algorithmic unknowability can be *bias-enabling*, that is they open up the possibility to introduce data scientist's bias into the process. *Bias-enabling interpreting*, that is relying on own judgment and understanding to interpret algorithmic outputs informing further work, makes it possible to introduce own bias into, for example, analyzing independent variables to select which ones may be predictive of the outcome, or evaluating model performance, where bias towards certain criteria over others (e.g. speed versus accuracy) may influence the final indication of the best performing model. *Bias-enabling optionalizing*, taking decisions regarding AD from a range of optional choices, enables the possibility to introduce bias for example by opting in or out of removing outliers, deciding for or against transforming certain data types so that they can be taken into account in modelling, or in resampling data to artificially increase or decrease the number of certain observations. *Bias-enabling experimenting*, whereby data scientists try out various approaches to modeling aiming at

obtaining the bet result, opens up scope for introducing bias in what algorithms are selected, what parameters are set, and what features are selected.

Our findings offer three theoretical contributions. First, we build on literature that analyses the enablers of algorithmic bias (Barocas and Selbst 2016; Favaretto et al. 2019; Kordzadeh and Ghasemaghaei 2021). Focusing on algorithmic design as a potential source of bias (Barocas and Selbst 2016; Favaretto et al. 2019; Kordzadeh and Ghasemaghaei 2021), we extend this literature by providing a comprehensive overview of AD and identifying 32 discrete actions where bias may be enabled through data scientists' modes of responding to unknowability. We show that bias may enter AD not only through previously acknowledged bias-enabling choices of features or weights, but in seemingly trivial actions, such as removing outliers, resampling data, or evaluating model performance. Similarly, we extend current IS literature shedding light on the role of data scientists in shaping the outputs of algorithmic solutions by supplementing it with a more technical perspective. Not only social or organizational factors, such as struggles over access to knowledge, deploying various cognitive frames, or the mechanisms of producing organizational knowledge (van den Broek et al. 2021; Ghasemaghaei et al. 2018; Ghasemaghaei and Turel 2021; Joshi 2020) influence how data scientists develop algorithmic solutions, but it is also their prior unknowability as a technical aspect that shapes data scientists' engagement in AD, thus influencing outputs. Further research is needed to understand how the social, organizational, and technical factors interact in shaping data scientists' decisions in the course of AD, and in consequence what kind of bias may be enabled. Since our work was conducted outside of organizational boundaries, our findings should be tested and refined within an organizational setting.

We argued that algorithmic unknowability, the *ex ante* and *ex post* unknowability of the input-output relationship in algorithmic solutions (Zhang et al. 2021), is not only evident in run-time, where users engaging such solutions have to cope with their variability and unpredictability, but also in design-time. Data scientists developing algorithmic solutions need to deal with unknowability during AD. We contribute to this line of reasoning by identifying six types of algorithmic unknowability present in design-time: resource, problem, dataset, analytical, model, and performance unknowability. We hypothesize that these types of unknowability may also apply to run-time, but further research is needed to confirm this and thus contribute further to the understanding of the sources and consequences of algorithmic unknowability.

Finally, we drew on the concept of exploratory programming which captures how, in general, data scientists engage in AD where specific programming goals and tasks are not defined *a priori* but rather discovered while writing code (Kery et al. 2017; Kery and Myers 2017). We extend this literature by showing that as part of thus conceived exploratory programming, data scientists deploy bias-enabling interpreting, optionalizing, and experimenting as modes of response to algorithmic unknowability. By doing so, we provide a better understanding of the characteristics of exploratory programming, and we help delineate it as a different type of IS development where the unknowability of the system to be developed defies some commonly held assumptions about development (Hirschheim and Klein 1989; Ralph 2018). More work is needed to understand AD as a distinct form of information systems development.

Our work also offers contributions to practice. The actions we identified as taking place during AD provide managers with a better understanding of what is happening when such solutions are developed, and make visible the level of subjectivity that enters such actions through the data scientists' modes of bias-enabling responses. As a result, those responsible for AD have a better understanding of discrete actions where bias may enter, and in consequence can take steps to prevent it. Both managers and developers themselves, when conscious of bias-enabling interpreting, optionalizing, and experimenting, can become more aware of how these modes, when deployed, may channel implicit or explicit bias into AD.

Conclusions

In this work, we focused on algorithmic unknowability in design-time as a catalyst for action during the development of algorithmic solutions. We identified specific types of such unknowability, and outlined the role of data scientists' modes of response to unknowability as they develop solutions. Action that we traced in each Jupyter notebook containing an algorithmic solution was heavily influenced by these modes of response affecting the individual, subjective decisions taken by data scientists. In fact, all notebooks and algorithmic solutions developed that we studied followed different flows and yielded different results. This reflects our main argument that bias-enabling interpreting, optionalizing, and experimenting opens up

possibilities to introduce subjective, individual perspectives which may reflect individual biases of data scientists. Each instance of dealing with algorithmic unknowability by interpreting outputs, selecting some actions but not others, and experimenting with various features and parameters potentially enables bias.

References

Barocas, S., and Selbst, A. 2016. "Big Data's Disparate Impact," *California Law Review* (104:1), pp. 671–729.

Baum, J. A. C., and Haveman, H. A. 2020. "Editors' Comments: The Future of Organizational Theory," *Academy of Management Review* (45:2), pp. 268–272.

Baygi Mousavi, R., Introna, L., and Hultin, L. 2021. "Everything Flows: Studying Continuous Socio-Technological Transformation in a Fluid and Dynamic Digital World," *MIS Quarterly* (45:1), pp. 1–62.

van den Broek, E., Sergeeva, A., and Huysman, M. 2021. "When the Machine Meets the Expert: An Ethnography of Developing AI for Hiring," *MIS Quarterly*, pp. 1–50.

Davenport, T. H., and Harris, J. 2017. *Competing on Analytics: Updated, with a New Introduction: The New Science of Winning*, Harvard Business Press.

Dissanayake, I., Zhang, J., and Gu, B. 2015. "Task Division for Team Success in Crowdsourcing Contests: Resource Allocation and Alignment Effects," *Journal of Management Information Systems* (32:2), pp. 8–39.

Favaretto, M., De Clercq, E., and Elger, B. S. 2019. "Big Data and Discrimination: Perils, Promises and Solutions. A Systematic Review," *Journal of Big Data* (6:12), Springer International Publishing, pp. 1–27.

Gal, U., Jensen, T. B., and Stein, M. K. 2018. "People Analytics in the Age of Big Data: An Agenda for IS Research," *ICIS 2017: Transforming Society with Digital Innovation*, pp. 0–11.

Ghasemaghaei, M., Ebrahimi, S., and Hassanein, K. 2018. "Data Analytics Competency for Improving Firm Decision Making Performance," *Journal of Strategic Information Systems* (27:1), Elsevier, pp. 101–113.

Ghasemaghaei, M., and Turel, O. 2021. "Possible Negative Effects of Big Data on Decision Quality in Firms: The Role of Knowledge Hiding Behaviours," *Information Systems Journal* (31:2), pp. 268–293.

Hill, C., Bellamy, R., Erickson, T., and Burnett, M. 2016. "Trials and Tribulations of Developers of Intelligent Systems: A Field Study," *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* (2016-Novem), pp. 162–170.

Hirschheim, R., and Klein, H. K. 1989. "Four Paradigms of Information Systems Development," *Communications of the ACM* (32:10), pp. 1199–1216.

Joshi, M. P. 2020. "Custodians of Rationality: Data Science Professionals and the Process of Information Production in Organizations," *AMCIS 2020 Proceedings*, pp. 0–1.

"Kaggle.Com." 2021. *Kaggle.Com. Available at: www.kaggle.com. Accessed on: 15 March 2021.*

Kery, M. B., Horvath, A., and Myers, B. 2017. "Variolite: Supporting Exploratory Programming by Data Scientists," *Conference on Human Factors in Computing Systems - Proceedings* (2017-May), pp. 1265–1276.

Kery, M. B., and Myers, B. A. 2017. "Exploring Exploratory Programming," *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* (2017-Octob), pp. 25–29.

Kery, M. B., Radensky, M., Arya, M., John, B. E., and Myers, B. A. 2018. "The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool," *Conference on Human Factors in Computing Systems - Proceedings* (2018-April).

Kim, M., Zimmermann, T., DeLine, R., and Begel, A. 2016. "The Emerging Role of Data Scientists on Software Development Teams," *Proceedings - International Conference on Software Engineering* (14-22-May-), pp. 96–107.

Kordzadeh, N., and Ghasemaghaei, M. 2021. "Algorithmic Bias: Review, Synthesis, and Future Research Directions," *European Journal of Information Systems* (00:00), Taylor & Francis, pp. 1–22.

Kurgan, L. A., and Musilek, P. 2006. "A Survey of Knowledge Discovery and Data Mining Process Models," *Knowledge Engineering Review* (21:1), pp. 1–24.

Mangal, A., and Kumar, N. 2016. "Using Big Data to Enhance the Bosch Production Line Performance: A Kaggle Challenge," *Proceedings of 2016 IEEE International Conference on Big Data*.

Patel, K., Fogarty, J., Landay, J. A., and Harrison, B. 2008. "Investigating Statistical Machine Learning as a Tool for Software Development," *Conference on Human Factors in Computing Systems -*

Proceedings, pp. 667–676.
Ralph, P. 2018. "The Two Paradigms of Software Development Research," *Science of Computer Programming* (156), Elsevier B.V., pp. 68–89. (https://doi.org/10.1016/j.scico.2018.01.002).
Ransbotham, S., Fichman, R. G., Gopal, R., and Gupta, A. 2016. "Special Section Introduction: Ubiquitous IT and Digital Vulnerabilities," *Information Systems Research* (27:4), pp. 834–847.
Rule, A., Tabard, A., and Hollan, J. D. 2018. "Exploration and Explanation in Computational Notebooks," *Conference on Human Factors in Computing Systems - Proceedings* (2018-April), pp. 1–12.
Seidel, S., Berente, N., Lindberg, A., Lyytinen, K., and Nickerson, J. V. 2018. "AutonomousTools and Design: A Triple-Loop Approach to Human-Machine Learning," in *Communications of the ACM*, pp. 50–57.
Silva, S., and Kenney, M. 2019. "Algorithms, Platforms, and Ethnic Bias," *Communications of the ACM* (62:11), pp. 37–39.
Subramanian, K., Völker, S., Zubarev, I., and Borchers, J. 2019. "Supporting Data Workers to Perform Exploratory Programming," *Conference on Human Factors in Computing Systems - Proceedings*, pp. 1–6.
Tarafdar, M., Teodorescu, M., Tanriverdi, H., Robert Jr., L. P., and Morse, L. 2020. "Seeking Ethical Use of AI Algorithms: Challenges and Mitigations," *Proceedings of the 41th International Conference on Information Systems, December 13-16, India.*, pp. 0–7.
Vaast, E., and Pinsonneault, A. 2021. "When Digital Technologies Enable and Threaten Occupational Identity: The Delicate Balancing Act of Data Scientists," *MIS Quarterly*, pp. 1–55.
Yoo, Y. 2010. "Computing in Everyday Life: A Call for Research on Experiential Computing," *MIS Quarterly* (34:2), pp. 213–231.
Zhang, Z., Lindberg, A., Lyytinen, K., and Yoo, Y. 2021. "The Unknowability of Autonomous Tools and the Liminal Experience of Their Use," *Information Systems Research*, pp. 1–58.