DSWorkFlow: A Framework for Capturing Data Scientists' Workflows

MOSHE MASH, Carnegie Mellon University
STEPHANIE ROSENTHAL, Carnegie Mellon University
REID SIMMONS, Carnegie Mellon University

While machine learning algorithms continue to improve, their success often relies upon the data scientists' ability to detect patterns, determine useful features and visualizations, select good models, and evaluate and iterate upon results. Data scientists often spend a long time making very little progress as they struggle to determine how to proceed. In this respect, the understanding of data scientists' workflows and challenges has recently attracted a great deal of scholarly interest. However, the literature is mostly based on interviews and qualitative research methodologies. With this in mind, we developed *DSWorkFlow*, a data collection framework that provides researchers with the ability to observe and analyze data scientists' cognitive workflows as they develop predictive models. Using DSWorkFlow, researchers can collect data from a Jupyter Notebook, to reconstruct the code execution order and extract relevant information about data scientist workflow alongside the concomitant collection of qualitative data. We tested the framework experimentally with seven data scientists as they each created three machine learning models to inform our extraction algorithms.

CCS Concepts: • **Human-centered computing** → **HCI design and evaluation methods**.

Additional Key Words and Phrases: data science process, workflow analysis, workflow extraction

1 INTRODUCTION

The increasing proliferation of vast data repositories and powerful computing resources and platforms has given rise to the concomitant development of data science and machine learning (ML) as disciplines concerned with the establishment and development of generalizable processes for extracting knowledge and insights from data [8]. Data science in particular is a rapidly developing field that has attracted a great deal of interest from organizations that seek to analyze their massive troves of data in order to gain new insights and drive their decision making [7]. Companies are hiring data scientists to scale their capabilities [36] in such domains as healthcare [12, 24], business intelligence [25, 38], finance [5, 34] and in scientific computing for physics [32], bioinformatics [4, 10] and meteorology [22, 33], among others.

Despite extensive research and advances in the design and development of ML algorithms, their success often relies on the proficiency of data scientists in performing such actions as determining useful features, selecting good models, and evaluating and iterating upon results [2, 15, 27]. A considerable body of recent qualitative studies (mostly interview-based) have sought to understand data scientist workflows and challenges [1, 6, 16, 20, 21, 28, 40].

As a whole, these studies have found that because every dataset and every data science task is different, the data science workflow is not monolithic, uniform, or sequential. Rather, it is contingent on the content and nature of the dataset and task and involves an iterative and exploratory process of trial and error, which many data scientists find challenging. Due to the open-endedness of the task of building accurate ML models, even expert data scientists often struggle during the development process in determining which actions to perform and what code to write [16]. As a result of their uncertainty, data scientists may collect additional data in spite of a solidified distribution [17], stop the process too early if they cannot make progress [16], or repeat the same actions, such as feature selection, over and over again as they focus on one aspect of the task at the expense of others [29].

In contrast to prior studies that have mostly relied on qualitative methodologies in the form of interviews and think-aloud sessions for data scientists to self-reflect on their processes, we propose a novel framework, *DSWorkFlow*, for capturing a more complete picture of data scientist workflows as they create machine learning models, including code, code output, and their thought processes.[1] The primary goal of DSWorkFlow is to bridge the gap between qualitative and quantitative methodologies, allowing researchers the opportunity to study data scientists as they are working through realistic problems in a familiar development environment - Jupyter Notebooks. Beyond its familiarity, Jupyter Notebooks also contain valuable data embedded within them, including output and execution data. By logging Notebooks regularly, DSWorkFlow can then parse the collected data and extract relevant information to reconstruct code and output sequences.

We tested our framework with seven data scientists as they worked on three machine learning tasks. Our study procedure includes a tutorial, think aloud protocol, and the three datasets and tasks. We then developed tools that used the Jupyter Notebook logs and audio and screen recordings for extracting action sequences that participants performed throughout their workflows. Our ongoing work uses the extracted output from DSWorkFlow to predict when data scientists may be stuck.

This paper describes the three distinct components of DSWorkFlow: Workflow Collection, Reconstruction, and Extraction. As a whole, we believe that using the DSWorkFlow framework will enable researchers like us to capture a richer understanding of the data science workflow and challenges that data scientists face as they work, and to develop better tools to support them.

2 WORKFLOW COLLECTION

The first component of DSWorkFlow is Workflow Collection. DSWorkFlow supports the complete collection of realistic workflow data in a lab setting with a focus on two major goals: 1) the ability to extract the entire cognitive workflow, including both digital artifacts and thought processes; and 2) the capacity to collect a broad set of workflows in a variety of domains and tasks.

2.1 Digital Artifact Collection

In order to reduce the variability in programming languages and coding environments, we chose to limit our collection to Python code as it is currently the most popular language for data science [3, 19] and Jupyter Notebooks on account of their popularity for data science applications [13].

Jupyter Notebooks: A Jupyter Notebook is an interactive Python development environment that is used by the overwhelming majority of the data science community and which allows the development of Python code that can be sensibly replicated by others [31]. The notebook consists of a sequence of cells of various types. A code cell allows a developer to write and edit code, which is executed using the Python interpreter and output in real time in the space under that code cell. Markdown cells allow the developer to use rich text in order to document the computational process. Developers can add, remove, combine, and execute cells in any order at any time during the development process. Jupyter Notebooks are particularly well-suited to the initial development of machine learning models in which data scientists work iteratively to produce a proof of concept since the code in the notebook is executed immediately and the output is presented in-line.

Notebook Snapshots In addition to their popularity among data scientists, Jupyter Notebooks can be parsed to extract the data science process. The underlying structure of a Jupyter Notebook is a JSON document containing structured descriptions of all the cells in the notebook. The JSON document encodes the content of each cell, the output of those cells when run, and the execution order of the cells within its code. We make use of this format and structure to reconstruct the data scientists' workflow from a sequence of Jupyter Notebooks.

The content of the Jupyter Notebook files changes as data scientists write and execute cells. Cells that occur sequentially within the interface may never actually be run sequentially, but

[1]The complete framework can be obtained from our open-source data repository.

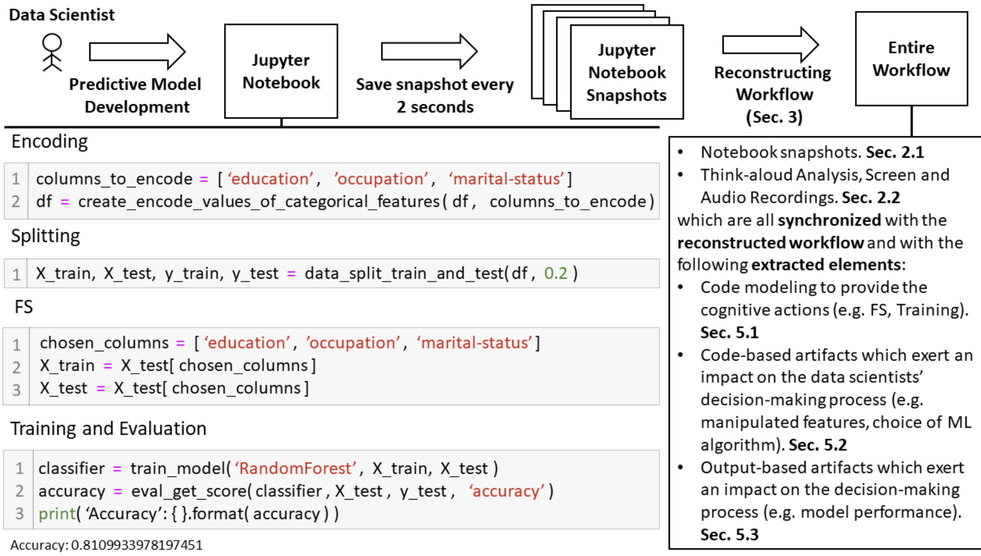

Fig. 1. DSWorkFlow process for reconstructing the entire cognitive workflow

a record of their execution order is embedded in the Notebook. Since we wish to preserve a complete chronological record of the data scientists' workflow including their code writing and code execution, DSWorkFlow automatically saves a copy of the Notebook, including the embedded data, in IPYNB files at regular intervals (2 seconds in our case). The procedure used to parse and sequence these Notebooks in order to recreate the development process is described in Section 3.

2.2 Cognitive Workflow Collection

In addition to capturing the code, output, and execution history, we also record the data scientist's screen and audio in order to track their visible and audible (i.e. think-aloud) interactions with the Jupyter Notebook interface (or with other applications such as a web browser).

Think-Aloud Protocol: The goal of our think-aloud protocol is to encourage the data scientists to express their thought processes as they are writing and executing code, interpreting the results, and testing their assumptions [11] (see Van Someren et al. [35] for an in-depth review of one such protocol). There are many possible reasons that this introspection could be important and complementary to other qualitative data collection techniques. For example, one could use the think aloud to determine what triggers data scientists to perform the actions they choose or alternatively the actions she or he perform while they struggle.

Screen and Audio Recording: We recorded the screen and the think-aloud audio using Screen-Rec [23], in order to allow the broadest possible range of analyses. The records includes complementary information that is not saved in the data logs. A record of the screen enables researchers to observe what the data scientist was looking at and for how long (e.g., if they are viewing output, scrolling through their code, or even if they leave the notebook to search the web for a specific API function call of a specific library). These audio and screen recordings could be synchronized with data logs to provide an additional dimension for analysis. While we did not use such devices in our test, additional devices such as eye trackers according to research needs as well.

Simplified Machine Learning API: The Python libraries most commonly used for non-specialized machine learning applications contain a variety of functionality, only some of which may be familiar to any particular data scientist [13]. To prevent any impact a lack of API expertise may exert on our own workflow collection (e.g., slowing down development, frequent online searches for API calls, etc.), we developed a set of functions on top of these libraries for participants in our study to use. These functions still allow for variation in different data scientists' development style in each part of the data science process (e.g., visualization, model training, evaluations) without requiring data scientists to navigate extensive APIs. More specifically, our machine learning API incorporates elements from the following Python libraries: SciKit-Learn [30] machine learning library; Pandas [26, 39] data analysis library, and Matplotlib [18] and Seaborn [37] visualization libraries. The current version of our API was designed for general classification tasks. Nonetheless, it is broad enough to support a variety of functions necessary for generating ML models and could be extended to specific tasks (e.g., Natural Language Processing or Computer Vision) and to regression problems by expanding it accordingly. The full API can be obtained from our open-source data repository at https://github.com/MoshikMash/DSWorkFlow.

The main advantage of our API is that it aggregates several methods from different objects into a single function. For example, when a data scientist would like to train and evaluate a ML algorithm model using SciKit-learn, they would need to create an algorithm-specific object, fit the training data to it, transform it to the test set and finally use a suitable function for the metric she or he would like to use for evaluation. On the other hand, our API allows the data scientist to attain the same objectives with only two functions – *train_model* with a parameter specifying the name of the ML algorithm and *eval_get_score* with a parameter specifying the evaluation metric to be used). An actual example of how the API was used by one of our participants is presented in Figure 1.

The use of our API is an optional part of DSWorkFlow depending on what aspects of the data science process that researchers could desire to focus on. Not using the API does not affect the functionality of DSWorkFlow.

3 WORKFLOW RECONSTRUCTION

The second component of DSWorkFlow is Workflow Reconstruction, which sequences the executed code cells from the Notebook Snapshots logged by the Workflow Collection component. An important piece of information embedded in the Jupyter Notebook is an execution counter that increments each time a cell is run. For example, if two cells A and B are executed in the following order A, B, A, B, DSWorkFlow can observe the execution counts increasing over the 4 executions as well as any code changes that are made to the cells between executions: $(A:1, B:-)$, $(A:1, B:2)$, $(A:3, B:2)$, $(A:3, B:4)$.

Concatenating cell code in increasing execution order in a new IPYNB file reconstructs the entire code execution from start to finish with execution timestamps that can be matched against the think-aloud recordings. No execution counts are lost since the notebook is saved at regular intervals even if the data scientist deletes or modifies the cells. The execution timestamps could also be useful for other analyses (e.g., determining the rate at which a data scientist is executing code or the best classifier they have trained in the last 15 minutes). This new reconstructed Jupyter Notebook associates each line of executed code with the corresponding snapshot that it was extracted from. This allows the researcher to refer back to the original Notebook file alongside the screen and audio recordings in order to examine the data scientist's decision making process qualitatively.

To illustrate the workflow reconstruction process, imagine that a data scientist ran the code described in Figure 1 (according to the figure's cell order). Then, they wanted to examine whether Random Forest would attain better performance by dropping the feature "age" by modifying the first line of code in the third (FS) cell to `columns = [['education', 'occupation']]`, and running this cell followed by the next code cell (Training and Evaluation). DSWorkFlow will capture and save the Jupyter Notebook including the cells' code, their output (e.g., accuracy for the Training and

Evaluation code cell), and their execution number after each time cells are run. The reconstructed workflow will be made up of the code rows described in the figure, followed by the modified FS cell and the Training and Evaluation code cell.

4 TESTING THE FRAMEWORK

Before proceeding to a discussion of DSWorkFlow's Workflow Extraction component, we present an application of DSWorkFlow in which we employed the aforementioned testing procedure and which will appear in a future publication. More specifically, we tested our data collection and reconstruction components on seven M.Sc. data science students (5 females and 2 males aged 23-28), and used their data to help inform our third DSWorkFlow component - Workflow Extraction.

The application relates to a common scenario that occurs when data scientists develop predictive models – they repeat the same actions, such as feature selection, over and over again as they focus on one aspect of the task at the expense of others, even if it does not help them improve the ML model [29]. This state of affairs wastes a lot of expensive time that could be more gainfully employed to other ends.

With this in mind, we labeled these moments using the records and think-aloud analysis and, in turn, used the data logs to develop a predictive model that could predict this "stuckness". Preliminary results show that this prediction task can be accomplished with a relatively high degree of recall (i.e., detecting when they are stuck) and low precision (i.e., predicting stuckness when they are not).

4.1 Experimental Procedure

We chose a within-subject in-lab study where each participant was asked to develop a machine learning model for each of three different datasets. Studying a data scientist performing on multiple datasets will eventually enable us to analyze the effects that a given dataset can exert on a participant's workflow and understand the similarities and differences across data scientists.

Based on pilot data, we determined that 90 minutes would be enough time for a data scientist to both explore a new dataset and iterate on ML modeling several times. Because 4.5 hours is comparatively long for a single session, we split the study into two parts; participants got instructions and performed one modeling task on the first day and then came back on a different day to complete two more tasks. In the first meeting, participants signed a consent form and completed a tutorial to build a sample machine learning model using our API and Jupyter Notebooks.

After the participants completed each task, they filled out a questionnaire that was meant to validate their understanding of the task, to capture their final thoughts about their models and their performance, and to assess the impact of DSWorkFlow on their workflow. The participants were also paid $45 (USD) per meeting for their time at the end of each of the two experimental sessions. The study was also approved by Carnegie Mellon University's Institutional Review Board (IRB).

4.2 Tasks

We selected three publicly available datasets which relate to general knowledge domains (i.e., which require no specfic expertise to understand) and which contain mixed data types (categorical, numerical, binary, dates, etc.) and the participants were asked to develop models to predict the target variables given the features provided.

Census Income Task (Census): The dataset [9] contains demographic information about a sample of the US population (e.g., age, marital status, degrees conferred, occupation). The target is to predict whether a person earns more than $50,000 (USD) per year.

Telecom Customer Churn Task (Telecom) The dataset [14] contains information about a telecommunication company's customers (e.g., demographic information, tenure, monthly payment, enrolment in services). The target is to predict whether the customer was retained or if they closed their account.

Australian Rain Task (Rain) The dataset [41] contains daily weather observations from Australian weather stations (e.g., wind speed, wind direction, temperature). The target is to predict whether it would rain the following day.

5 WORKFLOW EXTRACTION

The third and final component of DSWorkFlow is Workflow Extraction. This component processes the reconstructed workflow discussed in Section 3 and produces the framework's final output which can be downloaded as both a CSV file and a Python Pickle (library for saving an extracted Python Object) used as a Pandas dataframe for further analysis. The sub-components that we describe below help support our future work on identifying when a data scientist is stuck, but are general enough to be useful and adaptable to many other analyses or such as to accommodate a wide variety of research needs. The sub-components' code can also be obtained from our open-source data repository.

5.1 Action Modeling

There are several possible levels of abstraction that DSWorkFlow could employ to model the workflows from a code function level up to such tasks as wrangling, profiling and modeling as defined in [20]. We chose an intermediate *action* level representing functionality categories in SciKit-Learn as [1] and is reflected in the statements used by participants in our study. For example, participants performed feature selection (FS) several different ways since the code does not use any API code or any library function but rather uses Pandas syntax to filter columns from a dataframe. Sometimes, they used double brackets:

```
X_train[['education', 'occupation', 'marital-status', 'age']]
```

while other times they declared a list and then used single brackets for the feature selection:

```
columns = ['education', 'occupation', 'marital-status', 'age']
X_train[columns]
```

These different methods represent the same functionality. The advantage of using a list variable is that the data scientist does not need to write feature names twice when choosing from the train set and the test set though both are possible and appeared in our small dataset.

Representing code at a higher level, which we call *actions* could allow analysis to focus on the meaning of the lines and not the actual code writing itself. In particular, we chose the following actions: 1) Visualization; 2) Feature Engineering (FE); 3) Encoding i.e., transforming the values of a feature from categorical to numerical; 4) Splitting data into train and test sets; 5) Scaling features or the normalization of the feature values; 6) Feature Selection (FS); 7) Training of models; 8) Hyperparameter tuning (HPT); and 9) Evaluation by computing or visualizing performance metrics. Our action extraction method is a semi-automated parsing approach to assign each line or several lines of code into one of these nine action categories.

Many of the action assignments are straightforward, particularly when the code contains the names of one of our API functions or is included among a data science library's functionality categories. Some cases, however, require more sophisticated reasoning for correct labelling, such as the FS code shown in Figure 1. In order to label these lines of code properly, our parsing algorithm detects when a list is first created and then searches for the list's variable name in subsequent lines in order to label those lines according to the latter action. In a small number of cases, such as visualizations that are not supported by the API or selecting features to be printed rather than used for training models, the parser cannot assign labels and we must label them manually. After automated and manual labeling, 8571 actions were collected across our 7 data scientist participants in each of their three tasks (99% were labeled automatically).

5.2 Extracting Code-Based Artifacts

In addition to actions, DSWorkFlow extracts code-based artifacts that our study showed to be relevant to the data scientists' decision making process. For instance, an important piece of information that is extracted by the framework is the features that the data scientist selected or engineered when performing FS or FE. For example, as described in the FS part in Figure 1, the framework searches the lines of code labeled action FS, and extracts the set of features that the participant chose for the model (e.g., education, occupation, marital-status) using the same code that searches for lists described above. Because DSWorkFlow extracts these lists over the whole workflow, it also indicates which features were added or removed from the set since the last time the data scientist performed Feature Selection.

Similarly, the framework extracts the chosen model (e.g., Random Forest (RF) in the training part in Figure 1), the parameters chosen when training models with non-default parameters (e.g., maximum tree length for Random Forest) and the specific metric used when evaluating the model (e.g., accuracy). All participants in our study used our API, and thus this extraction is performed by analyzing the *train_model* and *eval_get_score* function calls. Performing this extraction on general SciKit-Learn functions is feasible, but would require modifying this component.

In addition, DSWorkFlow extracts an indicator of whether code has changed since the last time the cell was executed. This indicator is important because data scientists often execute cells without modifying them just because they formed part of their model development pipeline's "flow." This is accomplished in practice by carrying out comparisons between the current code snapshot and previous code snapshots.

5.3 Extracting Output Artifacts

During our studies, our participants commented on several of the output artifacts that were relevant to their decision making. The most common observation was the evaluation metric score. Participants used the score values to determine what models to use next (e.g., *"...LR's accuracy is greater than KNN's so I prefer using LR [logistic regression]..."*), and which features to use (e.g., *"...The performance is better for some of the classifiers (e.g. LR and Random Forest) once I remove the 'education' feature..."* and *"...I am trying to see whether there are features with strong correlations with the target that are not correlated with other features that I did not include in the model. So, I will add Temp3pm (strong correlation and not correlated with any other feature)..."*). We also observed them comparing several different metrics to compare classifiers (e.g., *"...The AUC is better for LR. However, the recall for KNN is higher..."*).

We implemented an algorithm to extract the results for all the metrics that the data scientists printed, because the metrics that different data scientists care about varied given the task and the person. Moreover, we found that data scientists often use several metrics to evaluate their model.

We detect the type of metric by searching for the metric's name as an API function parameter (for example, see the last cell in Figure 1) and extract the score from the output. This can also be done in a very similar manner with other APIs too. For example, if we were to search for accuracy in SciKit-Learn, then we would be looking for the "accuracy_score" function within the Jupyter Notebook (since SciKit-Learn contains a dedicated function for each metric). Alternately, the data scientist could evaluate a model and assign the result into a variable and print it later. In this case, the framework detects when the variable was saved in order to associate it with its metric type. Another possible scenario we need to take into account is one where a confusion matrix includes several metrics (e.g. TP, FP, TN, and FN for binary classifications). In this case, the framework extracts all metric scores.

An additional and interesting scenario relates to the iterative cycle of analysis associated with writing code for analyzing and manipulating data, training ML algorithms and evaluating and inspecting the output to maximize the predictive model's performance [2, 15, 27]. In this case, the framework saves a counter variable that counts the number of training and evaluation iterations to allow the researcher to distinguish between different iterations. For example, a researcher might check whether the performance improved between iterations. In addition, DSWorkFlow saves an indicator of whether a line of code generates an error from the Python interpreter along with the error output displayed, as this can be indicative of fixing code and rerunning cells.

The Workflow Extraction tools that we contribute are general purpose and can be useful for a variety of applications. The action modeling allows researchers to investigate patterns at a higher level than the lines of code, and our code and output extraction algorithms detect a variety of useful information, including the models, metrics, and features used. Our framework is also extensible and researchers can write Python code to extract new features for their own applications. For example, we developed features that describe the number of times the data scientist succeeded in improving their model's performance over time, which we use in predicting when a data scientist is stuck.

6 CONCLUSION

Data science and machine learning are new, rapidly growing, and highly influential fields. Thus, scholars have expressed a great deal of interest in attempting to understand data scientists' cognitive workflows. With this in mind, this paper introduces DSWorkFlow, a novel framework for the collection of data scientists' workflows, containing Workflow Collection, Reconstruction, and Extraction components.

A primary goal in developing DSWorkFlow was capturing as much of the data science workflow as possible. With this in mind, our framework allows the collection of data from several sources (both quantitative and qualitative) such as think-aloud analyses, screen and audio recordings, reconstructed code and output workflow from Jupyter notebooks, and the extraction of relevant information. This, in turn, enables researchers to conduct extensive analyses of data scientist workflow and "connect the dots" by synchronizing information from different sources.

We posit that DSWorkFlow will facilitate a broad range of quantitative research on data scientists' cognitive workflow, and help establish a bridge between theoretical and applied research, as well as between quantitative and qualitative methodologies in the field. Using our DSWorkFlow framework, researchers will be able to study data scientists as they are exploring data and developing models and open the door to the development of a number of applications to support data scientists, such as predicting when data scientists are struggling and providing them feedback about how they can make progress, comparing data scientists from varying levels of expertise or fields and facilitating data science training and instruction in both academic and corporate settings.

ACKNOWLEDGMENTS

This paper is based in part upon work funded and supported by the Department of Defense under contract FA8702-15-D-0002. This research was also funded in part by JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *Ai Magazine* 35, 4 (2014), 105–120.

[3] Anaconda. 2020. *State of Data Science Report*. https://www.anaconda.com/press/anaconda-releases-2020-state-of-data-science-survey-results

[4] Jörg Behler. 2016. Perspective: Machine learning potentials for atomistic simulations. *The Journal of chemical physics* 145, 17 (2016).

[5] Tsung-Sheng Chang. 2011. A comparative study of artificial neural networks, and decision trees for digital game content stocks price prediction. *Expert Systems with Applications* 38, 12 (2011), 14846–14851.

[6] Anamaria Crisan, Brittany Fiore-Gartland, and Melanie Tory. 2020. Passing the Data Baton: A Retrospective Analysis on Data Science Work and Workers. *IEEE Transactions on Visualization and Computer Graphics* (2020).

[7] Thomas H Davenport and DJ Patil. 2012. Data scientist. *Harvard business review* 90, 5 (2012), 70–76.

[8] Vasant Dhar. 2013. Data science and prediction. *Commun. ACM* 56, 12 (2013), 64–73.

[9] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/adult

[10] Vedran Dunjko and Hans J Briegel. 2018. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics* 81, 7 (2018).

[11] K Anders Ericsson and Herbert A Simon. 1984. *Protocol analysis: Verbal reports as data.* the MIT Press.

[12] Jarrod D Frizzell, Li Liang, Phillip J Schulte, Clyde W Yancy, Paul A Heidenreich, Adrian F Hernandez, Deepak L Bhatt, Gregg C Fonarow, and Warren K Laskey. 2017. Prediction of 30-day all-cause readmissions in patients hospitalized for heart failure: comparison of machine learning and other statistical approaches. *JAMA cardiology* 2, 2 (2017), 204–209.

[13] Joel Grus. 2019. *Data science from scratch: first principles with python.* O'Reilly Media.

[14] Pooja Gupta. 2017. Telecom Customer Churn Task. https://www.kaggle.com/puja19/telcom-customer-churn

[15] Mark A Hall and Lloyd A Smith. 1998. Practical feature subset selection for machine learning. In *Proceedings of the 21st International Conference on World Wide Web*. Springer, Australasian Computer Science Conference.

[16] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. 2016. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 162–170.

[17] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and Visualizing Data Iteration in Machine Learning. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[18] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[19] Kaggle. 2018. *State of Data Science Report*. https://www.kaggle.com/kaggle/kaggle-survey-2018

[20] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2917–2926.

[21] Mary Beth Kery. 2017. Tools to support exploratory programming with data. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 321–322.

[22] Konstantinos G Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. 2018. Machine learning in agriculture: A review. *Sensors* 18, 8 (2018), 2674.

[23] TeddySoft Ltd. 2020. ScreenRec. https://www.screenrec.com

[24] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

[25] Scott A Neslin, Sunil Gupta, Wagner Kamakura, Junxiang Lu, and Charlotte H Mason. 2006. Defection detection: Measuring and understanding the predictive accuracy of customer churn models. *Journal of marketing research* 43, 2 (2006), 204–211.

[26] The pandas development team. 2020. *pandas-dev/pandas: Pandas*. https://doi.org/10.5281/zenodo.3509134

[27] Kayur Patel, Naomi Bancroft, Steven M Drucker, James Fogarty, Andrew J Ko, and James Landay. 2010. Gestalt: integrated support for implementation and analysis in machine learning. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*. 37–46.

[28] Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 667–676.

[29] Kayur Patel, James Fogarty, James A Landay, and Beverly L Harrison. 2008. Examining Difficulties Software Developers Encounter in the Adoption of Statistical Machine Learning.. In *AAAI*. 1563–1566.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[31] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.

[32] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. 2018. Machine learning at the energy and intensity frontiers of particle physics. *Nature* 560, 7716 (2018), 41–48.

[33] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. 2015. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 802–810.

[34] Regina Esi Turkson, Edward Yeallakuor Baagyere, and Gideon Evans Wenya. 2016. A machine learning approach for predicting bank credit worthiness. In *2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR)*. IEEE, 1–7.

[35] MW Van Someren, YF Barnard, and JAC Sandberg. 1994. *The think aloud method: a practical approach to modelling cognitive*. Citeseer.

[36] Siobhan Wagner and Shelly Hagan. 2019. Finance Needs People Who Work Well With Robots. (2019). https://www.bloomberg.com/news/articles/2019-08-20/finance-needs-people-who-work-well-with-robots

[37] Michael Waskom and the seaborn development team. 2020. *mwaskom/seaborn*. https://doi.org/10.5281/zenodo.592845

[38] Chih-Ping Wei and I-Tang Chiu. 2002. Turning telecommunications call details to churn prediction: a data mining approach. *Expert systems with applications* 23, 2 (2002), 103–112.

[39] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 – 61. https://doi.org/10.25080/Majora-92bf1922-00a

[40] Kanit Wongsuphasawat, Yang Liu, and Jeffrey Heer. 2019. Goals, Process, and Challenges of Exploratory Data Analysis: An Interview Study. *arXiv preprint arXiv:1911.00568* (2019).

[41] Joe Young. 2019. Australian Rain Task. https://www.kaggle.com/jsphyg/weather-dataset-rattle-package