# Predicting Data Scientist Stuckness During the Development of Machine Learning Classifiers

Moshe Mash
*Carnegie Mellon University*
mmash@andrew.cmu.edu

Shoshana Oryol
*Carnegie Mellon University*
soryol@andrew.cmu.edu

Reid Simmons
*Carnegie Mellon University*
rsimmons@andrew.cmu.edu

Stephanie Rosenthal
*Carnegie Mellon University*
srosenth@andrew.cmu.edu

***Abstract*—The success of data scientists in developing machine learning models is contingent on an iterative development process for detecting patterns in data, finding and extracting useful features, and maximizing their model's performance. However, it is often the case that they struggle during model development and become stuck and unable to make significant progress. We collected qualitative and quantitative data from the workflow of data scientists that allow us to learn from and examine such moments of stuckness. We used this data to develop a model for predicting stuckness based on real-time indicators, such as code artifacts, and then used the model to develop an innovative algorithm that determines precisely when a potential stuckness intervention should occur: as close as possible to the beginning of actual stuckness. Our algorithm's performance indicates the potential efficacy of predicting data scientist stuckness algorithmically under real-world circumstances and for real-world needs.**

## I. Introduction

Data science and machine learning are disciplines concerned with the extraction of knowledge and insights from increasingly large and more complex datasets [1]. As the field of data science has expanded, companies are hiring data scientists to engage in large-scale data analyses of massive troves of corporate data.

The emergence of data science as a field has led many researchers to study the challenges that data scientists face as they explore data and create machine learning models [2]. 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 [3]. Instead, it is contingent on the content and nature of the dataset and task and involves an iterative and exploratory process of trial and error that many data scientists (even experts) find challenging [3]–[6]. One particularly difficult aspect of this process is that the iterative nature of development leads data scientists to repeat the same actions over and over again in an attempt to determine the right combination of data features and machine learning models to maximize their performance metrics [7], [8]. At each step of their model development process, they struggle to decide whether to keep working on a particular task, such as feature selection, or to switch to a new task, such as data visualization or feature engineering, in an attempt to find a better model [9], [10].

At such points, data scientists can get stuck in a state where they either do not know how to proceed in order to continue making progress at improving their models or they continue performing the same actions repeatedly without any progress, potentially wasting valuable time that could be more gainfully employed elsewhere. As such, a tool that could detect stuckness and potentially provide feedback or alternative actions to perform could enable data scientists to be more efficient at developing machine learning models.

The present work has three main goals: (a) to study the nature of data scientists' behavior when they get stuck; (b) to analyze data scientists' code and output artifacts in order to extract relevant features for predicting stuckness; and (c) to develop an algorithm for determining optimal times for stuckness resolution interventions (e.g., offering alternate strategies) that would be feasible for implementation in existing real-world data science workflows.

While prior work that studied the data scientist workflow relied on qualitative methodologies in the form of interviews and observations of the data science workflow [4]–[7], [9], [11], [12], our in-situ data collection methodology allows us to analyze data scientist artifacts produced as data scientists are working, including code and code output, video recordings of the participants' screens and bodily and facial gestures, and audio recordings of their vocalized thought processes. This allows us to label data scientists' stuckness while they develop machine learning classifiers based on the qualitative data gathered and to develop a predictive model based on quantitative data extracted from the participants' code and output artifacts.

Finally, we developed an algorithm that smooths this predictive model in order to decide whether to potentially intervene and provide feedback to the data scientist that they are stuck. We optimized this confidence model such that the intervention would be as close as possible to the moment the data scientist got stuck, while minimizing false positives. Our results show that our algorithm predicts stuckness shortly after the data scientist actually gets stuck – $95\%$ of the true positive predictions were within $10$ minutes of stuckness onset, while the mean stuckness duration was $25.7$ minutes.

Our findings, therefore, indicate the feasibility of predicting when data scientists get stuck. More broadly, we believe that

978-1-6654-4214-5/22/$31.00 ©2022 IEEE

this logging framework and our models open the door to new areas of research that would assess data scientist progress as they work and provide effective support to limit their frustrations and improve efficiency.

## II. Preliminary Data Collection

With a view to predicting data scientist stuckness, we initially used the DSWorkflow framework [13] to collect data from 7 students studying for an M.Sc. with a data science specialization in fields such as statistics and computer science and who also possessed a range of practical experience (from 6 months to several years). The participants were asked to develop 3 predictive models for 3 different datasets and target variables. This required them to explore and visualize data, engineer and select features, examine different ML algorithms and their hyperparameters, and train and test different models within a Jupyter Notebook. The participants were also asked to verbalize their thought processes as they developed their models and we recorded their screen and audio. Furthermore, we saved the participants' Jupyter Notebooks every two seconds to capture their code and code output over time.

We chose three publicly available datasets for our tasks from the Kaggle [14] and UCI [15] machine learning repositories: *Census Income (CI)* [16], *Customer Retention (CR)* [17], and *Australian Rain (AR)* [18]. The participants were given 90 minutes to complete each task, the order of which was randomized. In total, we collected 20 sessions (7 participants x 3 data sets, minus one that was not recorded properly). The study was approved by our IRB.

In order to label stuckness, two researchers, one with social science and humanities expertise and one with data science expertise, jointly reviewed the recording of one task from one participant to get a general idea of phenomena that would indicate stuckness behavior in the course of completing the task. Each researcher then labelled each of the 19 remaining recordings separately. Their labels were then compared for similarity. The labelers achieved a high inter-rater reliability, measured using Cohen's Kappa [19] as $\kappa = 0.93$. All disagreements between the labelers were resolved on a case-by-case basis.

The researchers' consolidated data eventually led to the crystallization and formulation of a list of indicative symptoms of stuckness. Specifically, the screen and audio-based analyses produced the following main symptoms as indicating a state of stuckness: (a) Scanning the Data and/or Jupyter Notebook; (b) Reduction in the Pace of Work; (c) Re-execution of Models and Cells; (d) Producing Output without Analyzing it; (e) Participants' Explicit Statements. In total, 6 experimental sessions out of 19 were found to present symptomatic evidence indicating that the data scientist was stuck, representing 4 out of 7 participants (2 participants got stuck twice). The lengths of the aforementioned stuckness periods (in minutes) were as follows: 32, 31, 29, 27, 25, and 24.

## III. Feature Extraction from Code Traces

While the symptoms were found to be very relevant and clear-cut in the audio and screen recordings, an automated algorithm cannot easily identify and track them. In addition, we would not expect a data scientist to vocalize their thought process and share their screen while they are working. Thus, we determined to create a machine learning model that can predict stuckness from code-related features alone. Towards that end, this section presents our approach to extracting features from the code and its output.

Because code written in Jupyter Notebooks is not necessarily run sequentially (the data scientist can add, run, delete, or modify code in any order), we used the DSWorkFlow framework to save snapshots of the participants' Notebooks every 2 seconds, to reconstruct their code and output artifacts in chronological order [13], and to automatically extract relevant information from the notebooks to predict stuckness. Finally, we synchronized the timing of these actions with the labelled stuckness in our video recordings.

Using both the reconstructed code and output artifacts from the Jupyter Notebooks, we developed techniques for extracting a large number of features that we hypothesized could be useful in predicting stuckness from the data scientists' code. Each of the features was refined through the iterative data science process of model training, evaluation, and error analysis. The following feature types represent the features that we determined to be important in the classification of stuckness:

*Action Counts.* Important types of features are those that capture data scientists' tendencies to repeat actions over and over. For example, we observed that participants often repeatedly performed feature selection by changing the line(s) of code indicating the relevant features and then retraining and evaluating their models for many different combinations of features. Action count features were thus computed as the number of times the data scientist executed each of the aforementioned actions within a given window size. We counted actions whose underlying code was both edited and executed and did not return an error in order to avoid cases where participants executed their entire Notebook but only made a change to one cell and also in order to avoid counting actions repeatedly when the data scientist was debugging.

*Training/Evaluation Iteration.* The natural workflow of data scientists involves an iterative cycle where they write code for analyzing and manipulating data, train and evaluate ML models, and inspect how their changes affect the output in an effort to maximize the predictive model's performance [3]. With this in mind, we defined the training/evaluation iteration ($TEI$) as one iteration of this cycle (i.e. carrying out one or more manipulations of the data and training and evaluating various ML algorithms and/or tuning hyperparameters), and created a feature for each action that computes the number of times the action was performed divided by the number of $TEI$s within a window of time.

*Performance Measures.* Another aspect of stuckness is when

data scientists are not making progress in improving their model's performance. We extracted model prediction performance data, such as accuracy, F1, AUC, etc., by parsing the output of evaluation actions. We used this data to develop such features as last performance score and best performance score attained thus far, as well as indicators for whether performance improved recently.

Since data scientists can use different metrics based on the task's properties and their preferences, we defined a performance indicator, $PI$, that has a value of 1 if any of the extracted metric scores are higher by at least 0.01 points than the highest previous score. We then developed a feature, $\#PI$, that counts the number of $PI$s during a window of time (i.e., how many times the performance improved in the last $x$ minutes). The reasoning behind this feature is that the number of improvements in performance is expected to be low when a data scientist is struggling.

*Inactivity.* When data scientists struggle to determine what action to perform next, they tend to take frequent breaks from coding to re-examine their previous models and the data. We defined inactivity as not writing any code for more than a given number of seconds and accordingly defined the feature $\#inactivity$ as the frequency of this occurrence in a given time window. We also created a feature for the duration of longest period of inactivity within a given window of time.

## IV. Predictive Model

To predict stuckness using the features described in Section III, we implemented several machine learning models using the SciKit-Learn library including Decision Trees, Logistic Regression, Neural Networks, K-Nearest Neighbors, Support Vector Machine, Random Forests, and Gradient Boosting Decision Trees. Given 19 complete sessions[1], we used a leave-one-out methodology to train a single model using all but one session and evaluated the model on the remaining workflow session. We also conducted hyperparameter tuning on each model's important parameters.

We repeated this process with different sets of features and different sliding window sizes for the features that required it. In this context, window size refers to the amount of time in which we compute the features (e.g., counting the actions in a particular timeframe). In practice, this window corresponds to the amount of time that a support system would need to observe data scientists' coding actions and outputs before making a prediction. We tested window sizes of between 5 and 60 minutes with different window sizes for different features. We also examined models with a subset of these features, examined different combinations of metrics for the $\#PI$ feature, and used 30 and 60 seconds as the thresholds for the $\#inactivity$ feature.

We evaluated precision, recall, and the F1 score for each of our trained models using the features and window sizes that attained the highest $F1$ score. We do not use accuracy as a metric because our data is imbalanced – always predicting "not stuck" yields 89% accuracy. Moreover, our domain requires relatively high recall and precision in order to offer real-world usefulness. More specifically, a tool with low recall will not predict enough stuckness, and a tool with low precision will be more cumbersome than helpful as it will incorrectly predict stuckness too frequently. Based on an analysis of all the models, the following six features consistently produced the best performance: (a) Number of times that Feature Selection was performed in the last $x$ minutes ($\#FS$); (b) Number of times that Feature Engineering was performed in the last $x$ minutes ($\#FE$); (c) Number of times that Feature Selection was performed in the last $x$ minutes divided by the training-evaluation iterations in the last $x$ minutes ($\#FS/\#TEI$); (d) Number of times that Feature Engineering was performed in the last $x$ minutes divided by the training-evaluation iterations in the last $x$ minutes ($\#FE/\#TEI$); (e) Number of times that accuracy or recall performance improved in the last $y$ minutes ($\#PI$); (f) Number of times that the data scientist was inactive for at least 30 seconds in the last $y$ minutes ($\#inactivity$).

We varied the window sizes, $x$ and $y$, to understand the impact they exert on performance. Specifically, $x$ represents the window size which relates to actions (i.e. data manipulations and model training and evaluations) and $y$ represents the window size which relates to making progress. The best performance was attained by a Decision Tree (DT) model with a maximum depth of 6 ($F1 = 0.88$, $recall = 0.81$, $precision = 0.96$) with $x = 15$ and $y = 5$. The second highest score was attained by a Support Vector Machine with a $2^{nd}$-degree polynomial kernel ($F1 = 0.83$, $recall = 0.8$, $precision = 0.87$). The other models all had much lower performance.

## V. Stuckness Data Collection

The results of our predictive model indicate that it should work well in real-word applications. However, we wanted to verify this by comparing against participant-validated labels. To this end, we collected additional data from 12 M.Sc. or Ph.D. students who completed (randomly) one of the Telecom or Customer Retention tasks (as those have more types of variables). We repeated the same methodology described in Section II but used a video camera to record participants' faces and bodies and had the labelers (who were about 5 feet away from the participants) label stuckness moments in real time by watching the video and listening to the participants' think-aloud verbalization. We also conducted a post-hoc questionnaire to understand the participants' own feelings of stuckness after the task which covered the moments in which they believed they were stuck and their timestamps. This study was also approved by our IRB.

After the interview, we compared the stuckness timestamps mentioned by the participants and those of the labelers and resolved any outstanding disagreements. In this respect, it should be noted that the participants and the labelers largely agreed on the stuckness windows; the disagreements were largely about the exact second that stuckness began and ended.

[1]In addition to the one set of recordings not saved, we were also missing the code logs from a second session.

In addition, we also discovered two video-based symptoms that correspond with our notion of stuckness during this experiment: (a) Rigidity of Facial Expressions and a Lack of Motion; (b) Touching the Face and Hand Motions.

In total, data scientists indicated stuckness in 7 of the 12 experimental sessions. The mean length of those stuckness periods was 25.71 minutes, with a standard deviation of 7.15, a minimum of 18 minutes, and a maximum of 42 minutes. Analysis using Cohen's Kappa [19] showed that the labelers attained a high degree of inter-rater reliability ($\kappa_1 = 0.92$). Furthermore, the equivalent Kappa score between the participants and each of the labelers was $\kappa_2 = 0.93$.

Finally, we evaluated the DT model with the new batch of data and the aforementioned 6 features by employing 3-fold cross validation such that each fold contains 3 test sessions while the remaining 9 sessions are used for the model training process. We obtained an F1-score of 0.83, a recall of 0.82, and a precision of 0.81, which allows us to be fairly confident that our model would perform satisfactorily in real-world applications.

## VI. Stuckness Confidence Algorithm

Our end goal is to predict a data scientist's stuckness as close as possible to the actual beginning of the stuckness period. To this end, we used the collected data and the ground truth labels that were described in Section V and the DT model that was described in Section IV. Specifically, that model predicts whether the data scientist is stuck at any moment during the completion of the task. However, we wish to not only capture stuckness moments alone but to also capture them as early as possible. Furthermore, we would not want to intervene when the data scientist's behavior indicates stuckness for only a short period of time (e.g. thinking "gaps" between one set of actions and another). We thus developed an algorithm that indicates the level of confidence in stuckness and whether the data scientist's behavior indicates stuckness for long enough to justify an intervention.

To this end, we define $t$ as the current time during the task, $X_t$ as the feature vector at time $t$, and $T$ as a *Temporal Scope* that represents the time window (in minutes) that we examine in determining the confidence level. In particular, $X_{t-T}...X_t$ represents all the feature vectors between $t-T$ and $T$. We also define $P(c_k|X_t)$ as the probability generated by our predictive model (see Section V) for the current action and $P(c_k)$ as the static prior. We then use those definitions for the calculation of $P(c_k|X_i...X_t)$, which is the probability that the data scientist is experiencing a stuckness period at time $t$ given the performed actions between the $i$-minute and the $t$-minute, where $k \in \{Stuck, Not\ Stuck\}$:

$$P(c_k|X_i...X_t) = \begin{cases} \frac{p(c_k|X_t)\cdot P(c_k)\cdot P(c_k|X_i...X_{t-1})}{\sum_k p(c_k|X_t)\cdot P(c_k)\cdot P(c_k|X_i...X_{t-1})}, & \text{if } i \leq t \\ p(c_k), & \text{otherwise} \end{cases} \quad (1)$$

Our algorithm then estimates the stuckness probability using Equation 1 for each action and alerts the user about stuckness periods when the result is greater than a predefined threshold value, $\lambda$ $(0 \leq \lambda \leq 1)$, given a predefined $T$

$$pred(X_t|T,\lambda) = \begin{cases} Stuck, & \text{if } P(Stuck|X_{t-T}...X_t) > \lambda \\ Not\ Stuck, & \text{otherwise} \end{cases} \quad (2)$$

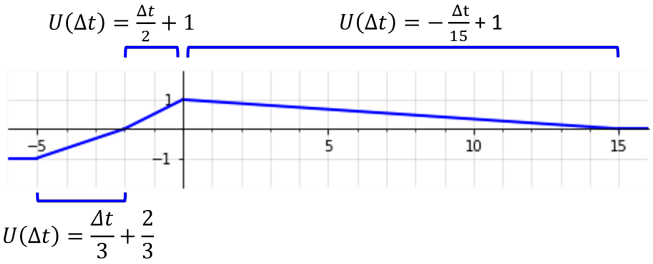


Fig. 1. Utility Function. The graph describes the obtained utility, $U$, as a function of the time difference, $\Delta t$, between the predicted start of stuckness and the Ground Truth's stuckness period start time.

There is an obvious trade off between how close to the beginning of the actual stuckness period an algorithm predicts that the data scientist is stuck and the rate of false positives – typically, the longer the algorithm waits to make a prediction, the more accurate it will be. To optimize that trade off, we developed a function (Figure 1) to compute the utility of making a prediction, with utility falling off further from the start of the stuckness period and negative utility if stuckness is predicted significantly before the data scientist is actually stuck.

Specifically, in Figure 1 $\Delta t$ represents the difference between the predicted start of stuckness and the Ground Truth (GT) start time. When $\Delta t > 0$, a lower $\Delta t$ represents an earlier prediction (closer to the GT stuckness start time) and will therefore will be assigned a higher utility. On the other hand, $\Delta t < 0$ represents cases where the predictions occurred before stuckness actually began. If the prediction is relatively close to the actual stuckness start, the utility is positive, but low; if the prediction is far from the actual start, the utility is negative.

The parameters of the utility function (the constants in Figure 1) were determined based on statistical measures of stuckness duration in our data (e.g. 18 minutes is the minimal duration of a stuckness period in our data). In addition, any prediction that is not associated with an actual stuckness period is assigned a utility of $-1$, and if the algorithm does not predict stuckness during the data scientist's entire task session, it is assigned a utility of zero (neither helps nor hinders).

To evaluate this algorithm, we used the data from the second study (Section V) and iterated over $0 \leq T \leq 30$ (at 5 minute intervals) and $0 \leq \lambda \leq 1$ to find the optimal hyperparameters that maximize our utility function. Specifically, for each setting, we used 3-fold cross validation (220 tests, $\binom{12}{3}$), with each fold containing 3 experimental sessions while the remaining 9 sessions are used for the model training process (the cumulative number of experimental test sessions is thus $220 \cdot 3 = 660$). At each fold we train a DT classifier with a

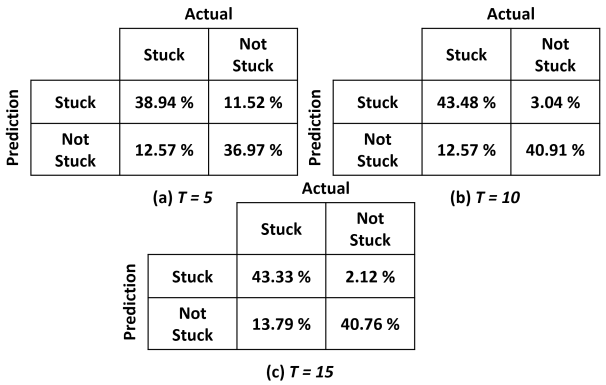


Fig. 2. Confusion Matrices for Different Temporal Scopes. Values are the percent of cases among all test set sessions.

maximum depth of six and use Equation 2 for each action in order to predict stuckness moments.

Finally, we summed the utilities from all the tests and normalized the values. Specifically, the lowest possible cumulative utility is $-660$ (a case where a utility of $-1$ is obtained for all the test sessions), and the highest possible utility is 385, as $7/12$ of the sessions in our data included stuckness periods and the number of tested sessions is 660 ($660 \cdot 7/12 = 385$)). We thus normalize the total utility, $U_{total}$, as $U_{norm} = (U_{total} + 660)/(385 + 660)$. In the rest of this paper, we use $U$ to indicate $U_{norm}$.

## VII. Stuckness Confidence Algorithm Results

We analyzed the impact that the Temporal Scope, $T$, exerts on the obtained utility. Specifically, we analyzed the highest obtained utility, $U$, (across all $\lambda$s from 0 to 1 with a step size of 0.1) as a function of $T$, and we found that there is a global maximum at $T = 10$, with a utility of 0.83 when $\lambda = 0.9$.

To better understand this observation, we examine the predictions associated with $T = 5, 10,$ and 15 by analyzing their confusion matrices (see Figure 2). The *true positive (TP)* predictions (the upper-left entries in each matrix) represent the cases where $\Delta t \geq 0$ and $\Delta t$ is less than the stuckness period's length. *False positive (FP)* predictions (upper-right entries) are where the algorithm predicts stuckness when none exists, which occurs when one of the following cases apply: (a) $\Delta t < 0$; (b) $\Delta t$ is greater than the stuckness period's length; or (c) the algorithm predicts stuckness during a session that does not include any stuckness moments whatsoever. *False negative (FN)* predictions (lower-right entries) relate to cases where the data scientist got stuck during the session but our algorithm did not predict any stuckness. Finally, *true negative (TN)* predictions (lower-right entries) are where the algorithm does not predict stuckness at all in cases where the data scientist did not get stuck throughout the entire session.

As can be seen in Figure 2, the success rates (i.e. *TP* + *TN*) of our algorithm when using $T = 5, 10,$ and 15 are 75.81%, 84.39%, and 84.09% respectively. An interesting observation that can be gathered from this Figure is that $T = 5$'s *FP* rate is relatively high (11.52%) compared to the equivalent rate for $T = 10$ (3.04%) and $T = 15$ (2.12%). This high *FP* rate can be explained by the fact that 5 minutes is a too short a time to determine true stuckness, so even quasi-stuckness behavior is interpreted as stuckness.

TABLE I
*TP* Prediction Analysis

| | % *TP* Predictions (% of total ) | | |
|---|---|---|---|
| Case / Temporal Scope ($T$) | 5 | 10 | 15 |
| $\Delta t = 0$ | 68.86(26.81) | 29.27(12.73) | 22.39(6.67) |
| $0 < \Delta t < 5$ | 27.24(10.61) | 36.58(15.9) | 22.39(9.7) |
| $5 \leq \Delta t < 10$ | 0(0) | 29.27(12.73) | 33.21(14.39) |
| $10 \leq \Delta t < 15$ | 0(0) | 0(0) | 19.57(8.48) |
| $\Delta t \geq 15$ | 3.9(1.52) | 4.88(2.12) | 9.44(4.09) |

Table I thus details the *TP* predictions (upper-left cells in Figure 2). Each entry represents the percentage of cases among the *TP* predictions that are associated with a given Temporal Scope and the values in parentheses are the percentage of sessions that are associated with that Temporal Scope. Interestingly, we can see that 68.86% of the *TP* predictions when using $T = 5$ were predicted at the minute when the stuckness began, which is much higher than the equivalent values for $T = 10$ (29.27%) and $T = 15$ (22.39%). These 68.86% represent (26.81%) of all 660 test sessions and those predictions were assigned the highest possible utility ($U = 1$).

This observation, in turn, can be explained by the rapidity in which our algorithm reports stuckness periods as it detects a behavior that indicates stuckness when using a short Temporal Scope ($T$). This property, in turn, helps our model catch the stuckness very quickly but, on the other hand, causes a high rate of *FP* predictions as we saw in Figure 2 and as explained above. Put differently, a designer of an intervention tool, or users themselves, could adjust $T$ to a value that represents the tradeoff between predicting stuckness as early as possible and a high rate of *FP*s.

## VIII. Conclusion

Data scientists often struggle to improve their predictive models and get stuck without making any significant progress. We captured this by collecting qualitative and quantitative data and specifically recordings of visual (screens and video) and audio (think aloud) thought processes alongside code and output artifacts.

We analyzed the recordings to label stuckness and then developed a model that would predict stuckness using only features that could be extracted from code and output sequences (feasible for real-time capture). Moreover, we developed an algorithm that makes use of this predictive model to determine optimal moments for intervention, such that the intervention will occur as early as possible after the start of the stuckness period but will also produce a low rate of false positive predictions as we would like to refrain from disturbing users during productive work. Accordingly, our future work will involve creating intervention tools and testing them with data scientists as they work.

## ACKNOWLEDGMENTS

This research was funded 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] V. Dhar, "Data science and prediction," *Communications of the ACM*, vol. 56, no. 12, pp. 64–73, 2013.

[2] T. H. Davenport and D. Patil, "Data scientist," *Harvard business review*, vol. 90, no. 5, pp. 70–76, 2012.

[3] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 667–676.

[4] C. Hill, R. Bellamy, T. Erickson, and M. Burnett, "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, 2016, pp. 162–170.

[5] M. B. Kery, "Tools to support exploratory programming with data," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 321–322.

[6] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "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, 2019, pp. 291–300.

[7] K. Patel, J. Fogarty, J. A. Landay, and B. L. Harrison, "Examining difficulties software developers encounter in the adoption of statistical machine learning." in *AAAI*, 2008, pp. 1563–1566.

[8] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[9] K. Wongsuphasawat, Y. Liu, and J. Heer, "Goals, process, and challenges of exploratory data analysis: An interview study," *arXiv preprint arXiv:1911.00568*, 2019.

[10] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, "Gamut: A design probe to understand how data scientists understand machine learning models," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.

[11] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson, "How data science workers work with data: Discovery, capture, curation, design, creation," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–15.

[12] D. J.-L. Lee, S. Macke, D. Xin, A. Lee, S. Huang, and A. Parameswaran, "A human-in-the-loop perspective on automl: Milestones and the road ahead," *Data Engineering*, vol. 58, 2019.

[13] M. Mash, S. Rosenthal, and R. Simmons, "Dsworkflow: A framework for capturing data scientists' workflows," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–7.

[14] "Kaggle," 2021. [Online]. Available: https://www.kaggle.com

[15] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[16] ——, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/adult

[17] P. Gupta, "Telecom customer churn task," 2017. [Online]. Available: https://www.kaggle.com/puja19/telcom-customer-churn

[18] J. Young, "Australian rain task," 2019. [Online]. Available: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package

[19] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.