Can We Do Better with What We Have Done? Unveiling the Potential of ML Pipeline in Notebooks

Yuangan Zou
University of Toronto
bill.zou@mail.utoronto.ca

Xinpeng Shan
University of Toronto
x.shan@mail.utoronto.ca

Shiqi Tan
University of Toronto
alexshiqi.tan@mail.utoronto.ca

Shurui Zhou
University of Toronto
shurui.zhou@utoronto.ca

***Abstract*—Computational notebooks are widely adopted by data scientists for experimenting with machine learning (ML) models. Despite the support for exploratory programming enabled by notebooks, they fall short in the ability to manage alternatives across different stages of the ML pipeline. In this study, we conduct a qualitative analysis to examine how data scientists explore various alternatives through a series of versions of notebooks on Kaggle. The findings indicate that data scientists investigate multiple alternatives at each stage across multiple versions, yet only a limited number of combinations from different stages are explored. Next, by combining alternatives from all stages to form previously unexplored paths, we discover that certain untested combinations of alternatives can outperform the best models as identified in the original notebooks. Moreover, by substituting the hyperparameter optimization and model configuration stages with AutoML methods, we observe that only a select number of ML pipelines experience improvement via AutoML, which implies the limitation of the current AutoML techniques. In summary, our study provides insights into the systematic and effective exploration of overlooked ML pipeline configuration combinations that yield superior results. The findings shed light on future research directions such as the development of tooling support of alternative management while striking a balance between manual exploration and automated optimization.**

***Index Terms*—Alternatives, computational notebooks, exploratory programming, code history, machine learning pipeline, version control, AutoML**

I. Introduction

Data scientists conduct exploratory programming in computational notebooks to derive insights from a large amount of data by building high-performance machine learning (ML) models [1], [2], [3], [4]. One critical aspect of exploratory programming is the need to compare and try various alternatives at different ML stages (e.g., data preparation, feature engineering, model configuration, and hyperparameter optimization) to finalize the best-performed model for their task [5], [6]. While the design of notebooks makes it easy for exploratory programming, they do not provide sufficient support for the comparison and management of alternatives in different stages [7], [8]. Data scientists frequently employ a "quick and dirty" method, which involves duplicating existing code into different cells within the same notebook and adjusting it as necessary, marking decision points with headings [9], [2]. This allows them to execute the alternative versions simultaneously and compare the results later [10], [7]. However, this practice often leads to messy notebooks that are hard to reproduce [11], [10], [2].

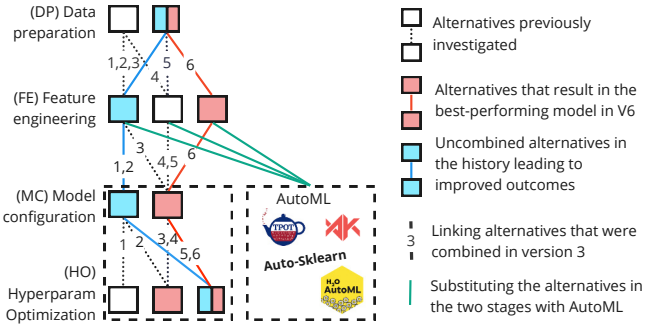


Fig. 1. Research overview. We extracted alternatives for each ML stage and manually combined the unexplored alternatives to form new ML pipelines.

To address these issues, researchers have developed various tools, including integrating a forking/branching mechanism into the notebook, which supports searching in edit histories to compare current and past results [3] and a side-by-side layout of alternatives for parallelism [7]. Similarly, prior studies designed methods to track the provenance of cells, enabling the comparison of successive cell versions [12], visualize and summarize the difference between versions of cells to help data scientists maintain an overview of the evolution of the explored ML pipeline [8], [13], [14]. While previous research focused on comparing alternatives for each cell, there is still a need to manually merge alternatives from various ML stages into a new pipeline, and then execute, document, and compare the outcomes. It remains unclear how analysts explore the combination of alternatives across different ML stages, if these alternatives are comprehensively analyzed during the ML lifecycle, and how to further support the exploration after extracting the diffs between versions of notebooks.

In this study, we conduct an exploratory study to gain insights into the existing practices of exploring alternatives and identify opportunities for enhancing exploratory programming using notebooks. We accomplish this by examining 52 notebooks with a total of 930 versions of ML pipelines and 23385 lines of code from the Kaggle platform [15], focusing

2576-3148/24/$31.00 ©2024 IEEE
DOI 10.1109/ICSME58944.2024.00052

on two main sets of questions: (1) understanding the current practices of exploring alternatives in notebooks (RQ1&2); (2) assessing the potential of unexplored alternative combinations (RQ3-5).

RQ1: What alternatives are data scientists exploring in each part of the ML pipeline?

Motivation: To initially understand the alternatives that were manually explored through different versions, we start by qualitatively examining the code changes between consecutive versions and identify the alternatives of each ML stage.

Results: The range of alternatives explored in different iterations can be classified into four main categories: data preparation, feature engineering, model configuration, and hyperparameter optimization. There is significant variation in the number of alternatives investigated within each notebook.

RQ2: How do data scientists explore alternatives in computational notebooks?

Motivation: To understand how alternatives are explored in notebooks through multiple versions and the way alternatives from each ML stage are combined to develop diverse ML pipelines, we document and identify the patterns of exploration.

Results: Data scientists explore alternatives in an iterative fashion including these steps: (1) constructing an ML pipeline and obtaining prediction results; (2) exploring model-related alternatives, namely hyperparameter optimization and model configuration; (3) exploring data-associated alternatives, including feature engineering and data preparation stage. Data scientists occasionally merge different alternatives to create a new ML pipeline, yet the median of our selected notebooks only represents 1.8% of all possible combinations. This situation drives our interest in delving deeper into the potential of unexplored combinations.

RQ3: Will an unexplored combination of alternatives outperform the original notebook? if yes, to what extent?

Motivation: Considering the significant number of alternative combinations that remain untested, we aim to investigate the possibilities of these unexplored ML pipelines. Therefore, we manually constructed new ML pipelines to compare their outcome with that of the originally best-performing model.

Results: Our study shows that for most notebooks (16/20), there exists an unexplored pipeline that can lead to better performance. In situations where the author has already explored every possible pipeline or there are too many possible pipelines to exhaustively search, we are not able to observe any performance improvement.

RQ4: Will incorporating AutoML toolkits into the ML pipeline surpass the results of a manually assembled ML pipeline? If yes, to what extent?

Motivation: Since AutoML toolkits are invented to facilitate effective exploration of alternatives, mainly in the model configuration and hyperparameter optimization stages, we substitute the alternatives manually explored with AutoML toolkits to evaluate the performance, as depicted in Figure 1.

Results: Incorporating AutoML tools does not lead to obvious improvement in terms of performance, with only 3 out of 20 notebooks showing enhanced performance, while 13 out of 20 notebooks experience a performance decrease and 4 notebooks with performance unchanged. Overall, we found that incorporating AutoML leads to a slight decrease (5.66%) in performance.

RQ5: Will a combination of alternatives from different data scientists outperform the original notebook result? If yes, to what extent?

Motivation: We take the initiative one step further by investigating how leveraging crowdsourced alternatives explored by various data scientists could result in better outcomes.

Results: Combining alternatives from different scientists does not seem to have a positive impact on performance overall, with 3 out of 10 notebooks showing performance improvement and 7 out of 10 notebooks showing either performance decrease or operational errors.

Our study shows the limitation of both integrating AutoML and foreign alternatives from other pipelines, specifically, the lack of domain knowledge of data and features, as well as the lack of compatibility and coherence across implementations of different data scientists. Our research indicates the potential and the need to support effectively and automatically exploring the combinations of alternatives during the ML lifecycle. The result augments past studies by offering a more in-depth look into the exploratory programming process.

In summary, this paper makes three major contributions: (1) An empirical study on how data scientists explore alternatives manually; (2) A replication package and dataset of explored alternatives for future automation research; and (3) Suggestions for future research directions and tooling support. The replication package is available at [16].

However, caution should be raised when generalizing the study's conclusions to broader contents. The dataset utilized in this study consists of ML pipelines sourced exclusively from Kaggle, and only high-quality notebooks were selected for our analysis purpose. Potential biases could exist since Kaggle datasets might not represent the full diversity of ML practices and pipelines used in various industries and research settings. Additionally, our AutoML experimentation is confined to four widely-used AutoML tools, which may not be representative of other toolkits.

II. Background

A. Exploring alternatives in the ML pipeline.

The ML pipeline refers to a systematic sequence of steps or processes designed to automate and streamline the workflow of developing, deploying, and maintaining machine learning models. The process usually encompasses several key stages, such as data preparation, feature engineering, model configuration, and hyperparameter optimization [17]. There is no unified definition of how many stages are included in an ML pipeline, and it's widely recognized that these workflows are non-linear and include multiple feedback loops [18].

Data scientists frequently experiment with various alternatives for certain stages by testing numerous models and parameters to determine the best fit [7]. Liu et al. conducted

interview studies to understand when, what, and how data scientists explore alternatives across various scenarios, including visual designs, datasets, hypotheses, and ML aspects. They identified both triggers (e.g., hitting a dead-end or a cognitive leap) and barriers (e.g., data scarcity or lack of expertise) to exploring alternatives with current tools [19]. Our research enhances previous efforts by examining the concrete alternatives that data scientists manually investigate at the source code level, aiming to determine whether unexplored ML pipelines could lead to improved outcomes.

Prior works have also studied the limitation of the design of the computational notebooks regarding comparing alternatives in the linear view [3], and further designed solutions to mitigate the challenges, such as introducing the 'forking' mechanism into notebooks to support side-by-side comparison. After the comparison, data scientists can delete the forks and only keep one that performs the best [7]. We contend that alternatives that do not show immediate promising results should not be discarded, as these outcomes may stem from an inadequately explored search space. Rather, we speculate that systematically exploring a combination of alternatives may lead to more promising outcomes than the traditional linear and limited search typically conducted by data scientists.

B. Studies on Computational Notebooks

Researchers have studied the computational notebook environment extensively and proposed solutions to improve the support during the exploratory programming process. For instance, previous studies have suggested approaches for addressing the issue of arbitrary execution order [10], [3] and better organizing code cells [20], facilitating communication in notebooks [21], providing visualization for better understanding of the code dependencies [22], improving reusability of notebooks [6], [23], documentation generation [24], [25], leveraging AI techniques to support exploratory programming using notebooks [26], etc.

The studies that are closely related to our project focus on managing variants and version control for notebooks. For example, Samuel et al. designed ProvBook as a Jupyter Notebook extension to track the version history at the cell level, allowing users to compare code changes side-by-side [12], however, it does not highlight the code diff. Similarly, researchers designed tools to visualize and summarize the difference between versions of cells to help data scientists maintain an overview of the evolution of the explored ML pipeline [8], [13], [14]. Several platforms such as GitHub and Kaggle support the versioning for notebooks, however, researchers found that most of the notebooks were uploaded after the exploration was done and mainly for backup purposes [27]. Our study complements prior studies by focusing on the scenario of managing and effectively exploring combinations of alternatives across the ML pipeline.

C. Automated machine learning (AutoML)

AutoML toolkits have been developed to automate tasks within the ML pipeline, aiming to lessen the burdensome cost of the trial-and-error method typically performed by data scientists. These toolkits support efficiently navigating the vast configuration space [28], [29], [30], [31]. Most AutoML toolkits focus on model generation and model evaluation. Specifically, model generation can be divided into search space and optimization methods. Search space is for generating either traditional ML models (a.k.a shallow ML models) or neural architectures for deep learning (DL) models. Correspondingly, the optimization methods can be classified into hyperparameter optimization and architecture optimization [28]. The specific number of models and hyperparameter sets to search can be defined by the user. Moreover, to support the automated feature engineering stage, researchers designed methods to generate large sets of features [32], [33], [34]. However, as feature engineering is heavily dependent on domain knowledge, none of the prior work includes AutoML tools in the feature engineering stage. Moreover, significant progress is still required to reach the anticipated objective. For example, prior research has outlined the difficulties encountered by data scientists in utilizing AutoML toolkits, revealing that these tools possess restricted capabilities. Notably, there is a deficiency in support for the operationalization of ML models/systems, referred to as MLOps, in addition to issues concerning model and data management [29]. A recent study from Alamin et al. provided a summary of 37 AutoML toolkits mentioned on StackOverflow, categorizing them based on the type of ML pipelines they support, including shallow ML models, deep learning models, or a combination of both model types [18], [29].

In this study, we are interested in exploring alternatives in not only the model configuration and hyperparameter optimization stages but also data preparation and feature engineering. Additionally, we evaluate the feasibility of taking all the human-explored alternatives in the first two ML stages and combined with AutoML infrastructure could result in models that outperform those generated by simply applying AutoML to the original notebook crafted by the data scientist.

III. Current Practices of Exploring Alternatives (RQ1&RQ2)

In this section, we describe the process of gathering data on a collection of high-quality notebooks with their series of revision histories, to qualitatively understand the alternatives that have been explored by data scientists.

A. Notebook Selection

We gathered notebooks from Kaggle, which is the largest platform for ML competitions and all the notebook versions are accessible to the public. Whenever the users use '*Save & Run All*', they execute the notebook from top to bottom and generate a new notebook version. A notebook version is a snapshot of the work including compiled code, log files, output files, and data sources. Gathering historical data on the ML pipeline allows us to systematically review modifications at various stages and identify the alternatives that data scientists have manually explored.

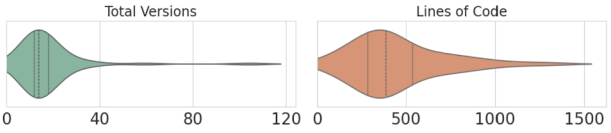


Fig. 2. Demographics of the 52 notebooks and 930 ML pipelines.

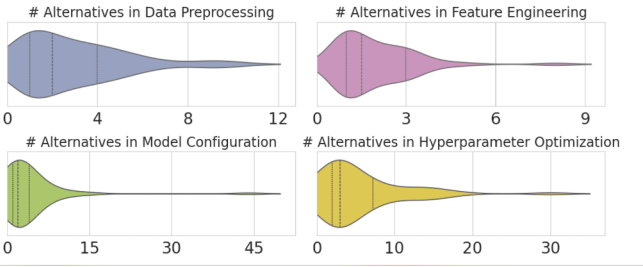


Fig. 3. Number of alternatives per category of all the 52 notebooks.

We chose the collection of "*Winning solutions of Kaggle competitions*" [35] as the starting point, which was collaboratively curated by Kaggle users as an indicator of high-quality notebooks. We established a series of criteria for inclusion: notebooks must (1) utilize Python as their programming language; (2) possess a minimum of five versions, providing a sufficient range of alternatives for analysis; (3) contain a comprehensive ML pipeline; and (4) be awarded medals or ranked in the top 20%, as a measure to assure and further ensure the notebooks' quality. Further, we exclude notebooks that have "beginner", "guide", "tutorial", or "introduction" in either the notebook or the competition title, to ensure the notebooks are focused on training ML models rather than educational purposes. Ultimately, 52 notebooks were gathered, covering 34 competitions and 5 domains including healthcare, social analysis, business, natural science, and education. We present the demographics of the notebooks in Figure 2.

B. Qualitative analysis of the alternatives

To identify the alternatives in each notebook, we analyzed the differences between consecutive versions to understand the motivations behind these changes, such as bug fixes, code refactoring, modifications to non-code elements, completion of the ML pipeline, and the exploration of alternative approaches. We concentrate exclusively on code modifications to investigate and extract the alternatives experimented with by data scientists at each ML stage, preserved across various versions.

During our analysis, we document alternatives at both the method level and the variable level. For example, for a function with a series of parameters, we denote it as `n(a,b,c,...)`, in which n represents the number of variables in the function and `(a,b,c,...)` represents the number of assigned values for each variable. For example, if there are three alternative variables explored for hyperparameter optimization, such as learning rate, epoch number, and batch size, and each has 4, 5, and 6 possible values respectively, we record it as `3(4,5,6)`.

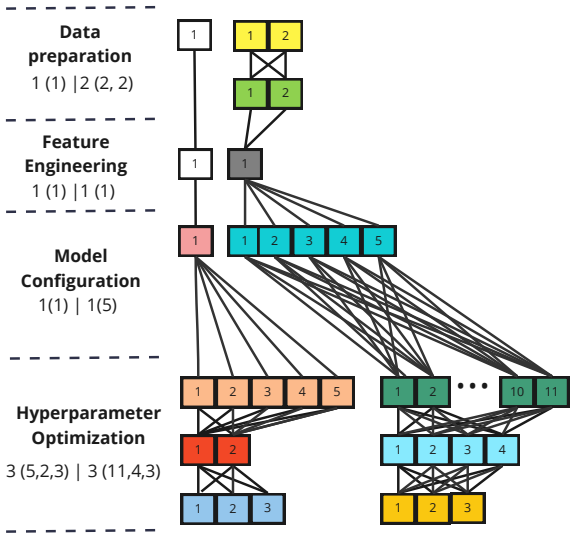


Fig. 4. Alternatives at the statement level of NB5.

The total number of alternative-values combinations would be `4*5*6=120`. In Figure 4, we visualize the alternatives that were created and explored in NB5. For the pipeline path that starts from the white box noted with `1(1)` in data preparation, the author introduced a set of functions in version 5 to remove HTMLs, emojis, URLs and duplicate columns in the dataset. We treat this set of functions as a single data-cleaning activity and note it as `1(1)`. For the other pipeline path that starts with yellow boxes, as in version 16, the author first formed a step of creating a function in the data preparation level in order to remove training samples with incorrect labels. Next, in version 46, the author introduced two boolean variables, *target_corrected* and *target_big_corrected* to remove training samples with incorrect labels and clean training data (i.e., removing emojis, punctuations, URLs and abbreviations). Therefore, we denote that there are two method-level alternatives, For each of these 2 alternatives (functions), we can choose whether or not to include it in the pipeline, therefore, we denote it as `2(2,2)`. For the first two notebooks, the first three authors independently went through the notebooks (NB5, NB15) and labeled the alternatives, the corresponding ML stage, and the version number. To address and resolve any discrepancies, they conducted two rounds of discussions. This method fostered a collaborative and comprehensive evaluation, minimizing the likelihood of unresolved issues. Following this, three authors independently labeled 16 to 17 notebooks each.

C. Results

RQ1: What alternatives are data scientists exploring in each part of the ML pipeline?

We observed that the alternatives explored by data scientists can be categorized into four main activities across the ML pipeline, including data preparation, feature engineering, model configuration, and hyperparameter optimization.

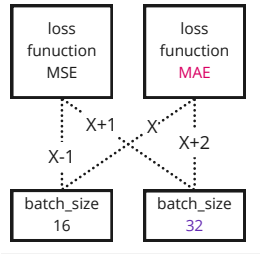
- **Data preparation** includes changes in cleaning, preprocessing, and wrangling, such as correcting issues (e.g., removing punctuation, imputing missing values, resizing images), and enhancing the dataset by adding new attributes, forming tri-grams, and aggregating data. For example, the author of NB5 added a method to remove training samples with incorrect labels in version 16.
- **Feature engineering** contains the changes in the process of transforming raw data into relevant features that improve model accuracy, involving variable selection, creation, and transformation based on domain knowledge. For example, the author of NB14 removed 2 features in version 3, leading to a model with better performance.
- **Model configuration** involves adjusting the hyperparameters related to the architecture or individual components of the model, including elements such as layers and loss functions. For instance, data scientists may try to modify the number and type of layers or activation functions inside the model. They sometimes also configure different models or stack multiple models together. For example, the author of NB5 added another dropout layer when defining the model structure in version 12.
- **Hyperparameter optimization** refers to the process of adjusting various parameters that control the learning process, such as the learning rate, the number of training epochs, and the batch size, among others. For example, the author of NB5 explored 5 different learning rates across versions 25 to 29.

Figure 3 summarizes the distribution of the alternatives in each category. For data preparation, the number of alternatives that have been explored in each notebook ranges from 1 to 10 (median = 1, SD = 2.048), and the number of alternative-values ranges from 1 to 2,304 (median = 3, SD = 333.72). For feature engineering, the number of alternatives that have been explored in each notebook ranges from 1 to 4 (median = 1, SD = 0.65), and the number of alternative-values ranges from 1 to 24 (median = 2, SD = 3.601). For model configuration, the number of alternatives have been explored in each notebook ranges from 1 to 44 (median = 2, SD = 6.429), and the number of alternative-values ranges from 1 to 45,120 (median = 3, SD = 6252.073). For hyperparameter optimization, the number of alternatives have been explored in each notebook ranges from 1 to 30 (median = 3, SD = 5.29), and the number of alternative-values ranges from 1 to 14,745,600 (median = 12, SD = 2044287.974). The detailed notebook information can be found in the replication package [16].

RQ1: Analysts investigated various alternatives across different ML stages, potentially resulting in a large number of ML pipelines.

RQ2: How do data scientists explore alternatives in computational notebooks?

Iteratively explore alternatives in the reversed order. We observed that data scientists tend to explore alternatives in an iterative fashion. Specifically, the data scientists began their process by constructing a full pipeline and developing the model. Following this, they initiated adjustments to the model's hyperparameters in pursuit of optimizing the model's performance. After several iterations of tuning, they might reconsider and explore different approaches to feature engineering (NB15, 16, 17, etc). In some cases, they alter data preparation methods (NB2, 5, 7, etc.) In other cases, they change the model configuration or adopt a new model into the pipeline (NB 6, 13, etc.). Finally, they start another round of hyperparameter tuning. Data scientists often experiment with alternatives in the ML stage in reverse order as shown in Figure 1. NB43 and NB31 are notable examples that conducted the exploration in this manner. In the first few versions of the notebook, the authors have processed the data and defined a model to run initial experiments with a set of hyperparameters. Rather than adhering to the initial configurations, the authors have revisited the earlier stages with each new version based on the experimental results and built alternative models and associated hyperparameters. Many iterations of the original complete pipeline would occur as the notebook version progressed. For example, compared to previous versions, version x implemented a new loss function (Alternative 1), version x+1 increased the batch size (Alternative 2), and version x+2 included both alterations and combined the new loss function with the increased batch size.



Combinatory Exploration of alternatives from different stages. It was noted that upon developing alternatives for specific stages, data scientists manually integrated these alternatives to construct various ML pipelines, evaluating their performance through successive versions. 51 Out of 52 notebooks have attempted synthesizing previous versions to create new experimental ML pipelines. As presented in Figure 1, it depicts how data scientists manually investigate various combinations of alternatives across different ML stages to develop a model with improved performance. This pattern not only highlights the iterative nature of ML model development but also underscores the data scientists' inclination to explore the combinatory search space of pipeline configurations. However, among all the possible combinations of alternatives, data scientists only explored 11% pipelines on average with a median of 1.80%. This motivates us to evaluate the unexplored combinations of alternatives and supports the underlying intuition that constitutes RQ3&4, which will be examined in the subsequent stages of our analysis.

RQ2: Data scientists often manually combine alternatives from various stages to create a new ML pipeline. Nonetheless, due to the constraints of current notebook designs and the vast search space of alternatives, many potential ML pipelines remain unexplored.

IV. Investigating Unexplored Combination of Alternatives (RQ3-5)

In this section, we outline the experiments conducted on previously uninvestigated combinations of alternatives and evaluate the performance. Furthermore, we evaluate the effectiveness of advanced AutoML solutions, pinpointing the obstacles they encounter as well as the possibilities they offer.

A. AutoML Tools Selection.

As described in Section II, there are two types of AutoML packages. The first type optimizes for hyperparameters based on user-defined bounds and optimization strategy, while the second type optimizes for both the model and the hyperparameters without the need to define anything in advance. For the selection of AutoML tools, we first limit our selection range to the second type since we assume that the user has no prior knowledge about which hyperparameters to optimize and what bounds to apply. Given the extensive time required for experiments and qualitative data analysis, we included only a subset of popular tools as a starting point. Eventually, we picked four of the most popular toolkits, as indicated by their popularity in multiple sources [36], including:

- AutoKeras [37]: *keras-team/autokeras*, with 9.1K stars;
- TPOT [38]: *EpistasisLab/tpot*, with 9.6K stars;
- Auto-sklearn [39]: *automl/auto-sklearn*, with 7.5K stars;
- H2O AutoML [40]: *h2oai/h2o-3*, with 6.8K stars.

These toolkits are mostly focusing on the model configuration and hyperparameter tuning stages. Each toolkit has specific application scenarios. For example, Auto-sklearn is for regular scikit-learn classifiers and regressors; TPOT optimizes ML pipelines using genetic programming to help with data preparation and modeling algorithms and model hyperparameters [41]; H2O AutoML provides automated model selection and ensembling for the H2O ML and data analytics platform; and AutoKeras is built on top of Keras, a popular deep learning framework that acts as an interface for the TensorFlow library. Among them, AutoKeras and Auto-sklearn are based on deep learning models and TPOT and H2O are for traditional ML models. To make a fair comparison, for deep learning models, we run either AutoKeras or Auto-sklearn based on the original model structure developed by the author. For notebooks that utilize traditional ML models, we choose TPOT or H2O.

As a result, out of 52 notebooks, we chose those that are compatible with any of the four AutoML toolkits. Each selected notebook must offer at least two alternatives in the data preparation and feature engineering phases, ensuring that post-AutoML application, alternatives remain for further exploration. This criteria led to a total of 20 notebooks, as summarized in Table I.

B. Methodology

Our approach consists of three stages, which are aimed at assessing the performance of untested ML pipelines (RQ3), evaluating AutoML toolkits (RQ4), and examining the mix of alternatives trialled by various data scientists (RQ5).

Step 1 (RQ3): Merging Alternatives into one notebook as configuration options. We gather alternatives for each stage of ML by examining successive versions. When changes are related to bug fixes or restructuring, we integrate these modifications into the alternative, rather than considering them as separate alternatives. For example, if version N is exploring an alternative and version $N + 1$ fixes a bug introduced in version N, we integrate the bug fix into the alternative, treating them as a consolidated entity. We pick the latest working version of the notebook as a baseline since it is typically the most complete version and free of bugs that would prevent ML pipeline execution. Table I, columns 4-7 show the number of alternatives identified at each ML stage. Usually, it takes about 6-8 hours to analyze a notebook that has approximately 20 versions to get it fully operational. Next, we assemble a "merged" notebook that enables us to iterate through all possible combinations of alternatives, thereby creating a variety of ML pipelines. Table I, column "Total #Pipelines" display the count of constructed ML pipelines built for each notebook, with totals ranging from 12 to 44,236,800.

To estimate the execution time, we refer to the Kaggle record for the execution time of the latest version and multiply it by the number of all possible ML pipelines. This gives us an estimated total running time needed to execute all merged pipelines. The total execution time for the sample notebooks varies from 12 minutes to 70707610 hours. Due to restricted computing capabilities, we opt for random sampling from the entire set of possible combined alternatives with a 95% confidence level and 10% margin of error whenever the total runtime surpasses 30 hours. To validate that the sampled subset of experimental ML pipelines could represent the whole pipelines, we conduct experiment on NB13, where we execute all 1,152 potential pipelines, followed by running a randomly selected sample of 89. Subsequently, we carry out a Student's t-test, yielding a p-value of 0.43. This outcome suggests that there is no statistically significant difference between the results obtained from running all alternatives and those from executing a randomly selected subset.

Step 2 (RQ4): Integrating AutoML into the ML pipeline. To evaluate the performance of AutoML toolkits, we replace the hyperparameter optimization and model configuration-related code with the selected AutoML toolkits. We then fed the AutoML tool the various iterated alternatives outputted from the exploration of the data preparation and feature engineering stages. Before feeding data into the AutoML tool, we split the data into training and evaluation sets. We then manually specify the total running time to be 30 hours to keep consistent with Step 1. We have observed that running AutoML on large datasets can be time-consuming. In some cases(NB6), even a runtime of 30 hours may not suffice for completing a single iteration of the training set. In this case, we manually decrease the population size, which controls the number of possible candidate model pipelines in one iteration, in order to make sure that the AutoML tool is able to return at least one model pipeline for further evaluation purposes. The selected AutoML tool will integrate through several

TABLE I
CHARACTERISTICS OF THE SELECTED 20 NOTEBOOKS FOR RQ3-5 AND CORRESPONDING RESULTS.

ID	RQ5 Used	Domain	#Versions	Data Prep.	Feature Eng.	Model Config.	Hyperparam. Tuning	Total # Pipelines	Author explored Pipelines	Merged Pipelines	Model Ensemble
1	y	Natural Science	23	1(4)	1(3)	1(4)	4(3,3,3,3)	1296	7	1296	
2	y	Natural Science	14	1(6)	1(2)	1(1)	1(1)	12	12	12	
3	y	Natural Science	11	1(2)	1(1)	1(2)	4(2,2,3,5)	240	9	240	
4		Healthcare	14	1(2)	1(3)	1(2)	1(2)	24	6	24	
5		Natural Science	104	1(1) 2(2,2)	1(1) 1(1)	1(1) 1(5)	3(5,2,3) 3(11,4,3)	2010	49	92	1(1)
6		Healthcare	35	1(1)	1(1)	2(1,1)	2(4,11) 1(1)	5324	23	5324	2(11,11)
7		Social Analysis	29	1(1)	1(1)	4(5,2,2,2)	1(5)	160	16	61	
8		Business	21	1(3)	1(1)	1(1) 1(1) 1(1)	5(3,2,2,2,2) 5(2,2,2,2,2) 3(2,2,2)	98304	8	2304	1(2)
9		Business	9	1(3)	1(2)	1(1)	2(4,3)	72	7	36	
10	y	Business	15	1(1)	1(3)	1(1)	18(4,5,5,3,2,2,2,2,2, 2,2,2,2,2,2,4,3,2)	44236800	8	97	
11	y	Business	13	1(2)	1(1)	1(2)	10(4,4,2,2,2,3,2,2,3,2)	36864	8	96	
12	y	Business	6	1(1)	2(2,2)	1(2)	2(2,4)	64	4	64	
13	y	Healthcare	12	3(3,2,2)	1(3)	1(1)	5(2,2,2,2,2)	1152	6	1152	
14	y	Healthcare	9	5(2,2,2,2,2)	2(2,2)	1(2)	2(2,2)	1024	7	88	
15	y	Business	20	1(1)	1(2)	1(2)	8(5,3,2,3,5,3,3,2)	32400	20	128	
16		Healthcare	8	5(2,2,2,2,2)	2(3,2)	3(2,2,2)	5(2,2,2,2,2)	49152	5	115	
17		Healthcare	12	1(1)	1(1)	1(3) 1(2)	3(2,2,2)	240	8	240	1(5)
18	y	Education	15	6(2,2,2,2,2,2)	1(1)	3(2,2,2)	1(1)	512	6	310	
19		Business	16	1(1)	2(1, 2)	1(3)	1(2)	12	6	12	
20	y	Education	6	1(1)	1(1)	3(1,1,1)	4(4,2,2,2)	96	6	96	

models and automatically optimize the model hyperparameters based on the training set and evaluate them through cross-validation [42]. In the end, the AutoML tool returns the best-performing model pipeline with the highest evaluation score as the final AutoML model. We then apply the final AutoML model to the evaluation set, using the same evaluation metric as the original submission in order to compare the performance.

Step 3 (RQ5): Integrating alternatives explored by different data scientists into one ML pipeline. To guarantee compatibility and streamline integration into a single ML pipeline with minimal code adjustments, we concentrate on combining notebooks from the same competition. Among the 20 sampled notebooks, we select 10 notebooks from four competitions: (1) Natural Language Processing with Disaster Tweets [43] (NB1, 2, 3); (2) Santander Customer Transaction Prediction [44] (NB10, 11, 12); (3) ICR - Identifying Age-Related Conditions [45] (NB13, 14); and (4) Feedback Prize - English Language Learning [46] (NB18, 20). Like in Step 1, we integrate alternatives from compatible notebooks into a single ML pipeline. The notebooks are considered compatible if they utilize the same dataset and can be integrated with minimal adjustments, ensuring a streamlined and error-free execution process. For the purpose of assessing the viability of integrating different approaches explored by various data scientists, we refrain from making any logical corrections to the merged notebook.

Evaluation. The performance of each formed ML pipeline is assessed using the validation set. In cases where the original competition did not use a validation set or the test datasets were not made available, we allocate 20% of the original training data to serve as the validation set. We ensure the training and validation datasets remain consistent throughout the exploration process to maintain uniformity in model training

TABLE II
SUMMARY OF EXPERIMENT SETTING.

		Kaggle	Colab	Clusters
GPU	Name	Tesla P100	Tesla T4	Tesla V100
	freq. (MHz)	1328.5	1590.0	1530.0
	memory (MB)	16276	15102	32501
	#multiproc.	56	40	80
	RAM (GB)	31	12	187
	disk space (GB)	32673	472	1737
CPU	name	Intel/Xeon)	Intel/Xeon	
	Freq. (MHz)	2199.998	2199.998	
	#cores	2	1	
	RAM (GB)	31	12	
	disk space (GB)	32308	174	

and distinctly identify the impact.

C. Experiment Setup

All experiments and measurements were conducted using one of the three settings. The specifications are summarized in Table II. The first method involves leveraging the Kaggle platform by forking the original notebook, integrating it on Kaggle, and executing it using Kaggle's computing resources. This technique eliminates the need to download substantial input datasets or configure the environment locally. It is our preferred method because Kaggle offers an environment that is compatible with the analyzed notebook. Nonetheless, due to Kaggle's GPU resource constraints, we also resort to alternative platforms for our analyses as needed, such as server clusters. The high-performance computing clusters can be efficient on large-scale computations and provide more computational power compared to Kaggle. However in order to use this approach, we need to download all input datasets and set up the environment on the cluster. If the previous two methods are not suitable for some notebooks, Google Colab and the local machine are also good options, which provide GPU and TPU support while on local machines we could gain full control over the environment. Depending on the size and complexity of the merged notebooks, we decided to utilize the most suitable approach each time. For instance, when dealing with smaller and less intricate merged notebooks, running them on platforms like Kaggle or Colab is a more convenient option due to the pre-configured environment and streamlined execution. For large and complex notebooks, utilizing the computational power of clusters would be a better option.

D. Results of RQ3: Original pipeline vs. Merged pipeline

Table III summarize the results RQ3-5. The metrics marked with an asterisk (*) in the table are indicators where lower values represent better performance, as these metrics quantify losses and errors.

Our experiment reveals that, excluding 4 out of 20 notebooks that remained constant, combinations of unexplored alternatives in the other 16 notebooks lead to an increase in performance of 13.57% on average, with a minimum and maximum observed increase of 1.09% (NB10) and 43.08% (NB1), respectively. In particular, during the data preparation phase of NB1, the author investigated the process of "pre-padding text sequences" as part of a natural language processing task, and in the stage of model configuration, the author opted for an alternative approach by "expanding the number of units in the recurrent layer." However, these two distinct alternatives were not previously integrated into a single ML pipeline in the version history. In our study, we discovered that this untried combination resulted in significant improvements, increasing the accuracy from 65% to 93%.

Regarding the notebooks where the experimental ML pipeline did not surpass the original version, there could be several reasons. For instance, in NB2, despite the existence of numerous alternatives across various ML stages, the author has explored every possible combination of these alternatives throughout the four stages across 14 versions of the notebooks. In NB11, there are in total 36864 possible combinations of alternatives and we only run 96 of them (to achieve 95% confidence level and 10% MoE), it is possible that we did not come across a combination that can lead to improvement.

Finding: While data scientists have manually merged various alternatives across different stages of the ML pipeline to create diverse ML pipelines in multiple versions of notebooks, strong evidence suggests the presence of an unexplored combination that could result in a more effective model. Exploring combinations of different alternatives is both time and resource-consuming, most data scientists are not able to exhaustively explore the alternatives due to either time or computing resource limitations.

E. Results of RQ4: Evaluating AutoML toolkits

[Original pipeline vs. Original pipeline + AutoML] Upon integrating AutoML into the original ML pipelines by replacing the hyperparameter optimization and model configuration stage, our experiments indicate that merely 3 out of 20 notebooks (NB1, 19, 20) exhibited enhanced performance by 6.15%, 9.41%, and 10.61%, respectively. Amongst the rest, 4 notebooks saw identical performance, while the remaining 13 notebooks have decreased by 16.59% on average, spanning from 1.89% to 59.77% decrement.

[Merged pipeline vs. Merged pipeline + AutoML] We additionally integrated AutoML into the manually merged alternative pipelines by replacing the hyperparameter optimization and model configuration stage. Out of the 20 notebooks, only NB19 yielded better performance (8.14%) compared to the manually merged ML pipelines as the AutoML tool was able to select a better model-hyperparameter combination than the original pipeline. Two notebooks maintained the same performance and the rest 17 notebooks had their performance decrease of 17.41% on average ranging from 1.08% to 56.25%.

We further draw comparisons between the original and manually merged pipelines, both with AutoML incorporated. Similar to their non-AutoML counterparts, we see that 9 out of 20 notebooks maintained the same performance, while the rest 11 notebooks improved upon the original pipeline with AutoML varying from 2.70% to 37.14%, with an average increase of 12.27%. The results showed that ML alternatives in data preparation and feature engineering have the potential

TABLE III
COMPARING MANUALLY MERGED ALTERNATIVES AND AUTOML INTEGRATION WITH THE CURRENT BEST ML MODEL

ID	Metric	Range	O	M	O+A	M+A	M vs O (%)	O+A vs O (%)	M+A vs O+A (%)	M+A vs M (%)	AutoML Toolkit
1	Accuracy	(0, 1)	0.65	**0.93** ↑	0.69 ↑	0.71 ↑	43.08	6.15	2.90	-23.66	TPOT
2	Accuracy	(0, 1)	**0.89**	**0.89** →	0.66 ↓	0.68 ↓	0.00	-25.84	3.03	-23.60	TPOT
3	Accuracy	(0, 1)	0.84	**0.93** ↑	0.84 →	0.84 →	10.71	0.00	0.00	-9.68	TPOT
4	Accuracy	(0, 1)	0.72	**0.96** ↑	0.63 ↓	0.70 ↓	33.33	-12.5	11.11	-27.08	auto-sklearn
5	Accuracy	(0, 1)	0.82	**0.96** ↑	0.70 ↓	0.70 ↓	17.07	-14.63	0.00	-27.08	auto-sklearn, TPOT
6	Laplace LL	$(-\infty, 0)$	-6.91	**-6.77** ↑	-7.78 ↓	-7.57 ↓	2.03	-12.59	2.70	-11.82	TPOT
7	Accuracy	(0, 1)	0.71	**0.72** ↑	0.68 ↓	0.70 ↓	1.41	-4.23	2.94	-2.78	AutoKeras
8	RMSLE*	$(0, \infty)$	1.46	**1.42** ↑	1.65 ↓	1.44 ↑	2.74	-13.01	12.73	-1.41	auto-sklearn, TPOT
9	RMSE*	$(0, \infty)$	0.23	**0.22** ↑	0.23 →	0.23 →	4.35	0.00	0.00	-4.55	H2O
10	Accuracy	(0, 1)	0.92	**0.93** ↑	0.92 →	0.92 →	1.09	0.00	0.00	-1.08	auto-sklearn, TPOT
11	CV Score	(0, 1)	0.90	0.90 →	0.90 →	0.90 →	0.00	0.00	0.00	0.00	TPOT
12	AUC	(0, 1)	**0.53**	**0.53** →	0.52 ↓	0.52 ↓	0.00	-1.89	0.00	-1.89	TPOT
13	Balanced Log Loss*	(0, 1)	0.13	**0.10** ↑	0.15 ↓	**0.10** ↑	23.08	-15.38	33.33	0.00	auto-sklearn, TPOT
14	Log Loss*	(0, 1)	0.25	**0.16** ↑	0.32 ↓	0.25 →	36.00	-28.00	21.88	-56.25	TPOT
15	RMSE*	$(0, \infty)$	0.26	**0.20** ↑	0.29 ↓	0.28 ↓	23.08	-11.54	3.45	-40.00	TPOT
16	Accuracy	(0, 1)	0.84	**0.88** ↑	0.79 ↓	0.82 ↓	4.76	-5.95	3.80	-6.82	TPOT
17	Laplace LL	$(-\infty, 0)$	-6.90	**-6.82** ↑	-7.62 ↓	-7.62 ↓	1.16	-10.43	0.00	-11.73	TPOT
18	R^2 Coefficient	(0, 1)	**0.87**	**0.87** →	0.35 ↓	0.48 ↓	0.00	-59.77	37.14	-44.83	TPOT
19	AUC	(0, 1)	0.85	0.86 ↑	**0.93** ↑	**0.93** ↑	1.18	9.41	0.00	8.14	TPOT
20	Mean RMSE*	(0, 1)	0.66	**0.58** ↑	0.59 ↑	0.59 ↑	12.12	10.61	0.00	-1.72	TPOT

O – results from the **original** pipeline; **O+A** – Integrating **AutoML** into the original pipeline; **M** – manually **Merged** ML pipelines; **M+A** – Integrating AutoML into the merged pipelines. Asterisk * emphasizes metrics where lower values are preferable. **Laplace LL** – Laplace Log Likelihood;

to enhance the performance of ML models. This reinforces the findings of RQ2.1, which highlighted that unexplored combinations of alternatives could yield better outcomes.

When comparing the search space created by data scientists and AutoML regarding the model configuration and hyperparameter tuning stages, our results show that AutoML cannot replace data scientists. There are many alternatives that are not directly model-related. For example, the number of folds for cross-validation, whether to shuffle the training data, etc. Exploration of pipeline configuration is needed in these cases to obtain an optimal result, which requires augmentation in the AutoML training to perform these searches as well.

In addition to training configuration alternatives, while there is a clear overlap between the search spaces selected by the user and the search spaces of AutoML for model-related alternatives, AutoML cannot cover all the models that the users have selected or would have preferred, particularly when the selection of models relies on domain knowledge of the ML task and the dataset [29]. For example in NB1, the author explored recurrent neural networks, which are not a type of model architecture covered by AutoKeras. Similarly, for NB13, the TPOT classifier explored the XGBoost model. The data scientist also explored XGBoost as a potential model alternative, albeit with a different and more effective hyperparameter search space.

Finding: Our study found that the current AutoML toolkits cannot replace data scientists when exploring alternatives in the ML pipeline. Our proposed approach shows promise in helping data scientists further improve their ML pipelines by incorporating their exploratory history with AutoML, rather than relying entirely on algorithms as is the case with AutoML.

F. Results of RQ5: Crowdsourding alternatives

As we would like to further understand if crowdsourcing alternatives that have been explored in different ML pipelines can produce better results, we manually merged alternatives from different ML pipelines into one. The outcomes indicate that in only one competition, merging three compatible notebooks (NB10, 11, 12) achieved enhanced performance after merging the alternatives, with the top outcome surpassing the original notebooks by 0.024%, 0.034%, and 63.81% respectively. The reason why NB10 and NB11 show little improvement is that the authors fork each other's notebooks and make further modifications to them, in this case, these two notebooks share almost the same libraries, function calls, as well as alternative configurations, with an average similarity score of 0.9949 across all stages. The integration of alternatives from each other does not make a noticeable difference in the final result. While NB12 shows significant performance improvement after integrating alternatives from the feature engineering stage of NB10, the potential reason is that the alternative in feature engineering from NB10 is creating new features based on statistical properties of the original features, while the feature engineering of NB12 is adjusting original features to have positive correlations with the target variable and normalizing them. This might not have effectively captured the underlying patterns in the data or might have introduced noise. For the rest seven notebooks from three competitions, after integrating alternatives from other compatible notebooks, we observe either a decrease in performance (NB13, 14, 1, 2) or operation errors (NB18, 20, 3) due to data structure size and format mismatch, missing files, and resource exhaustion. Specifically, NB18 and NB20 are not able to integrate with

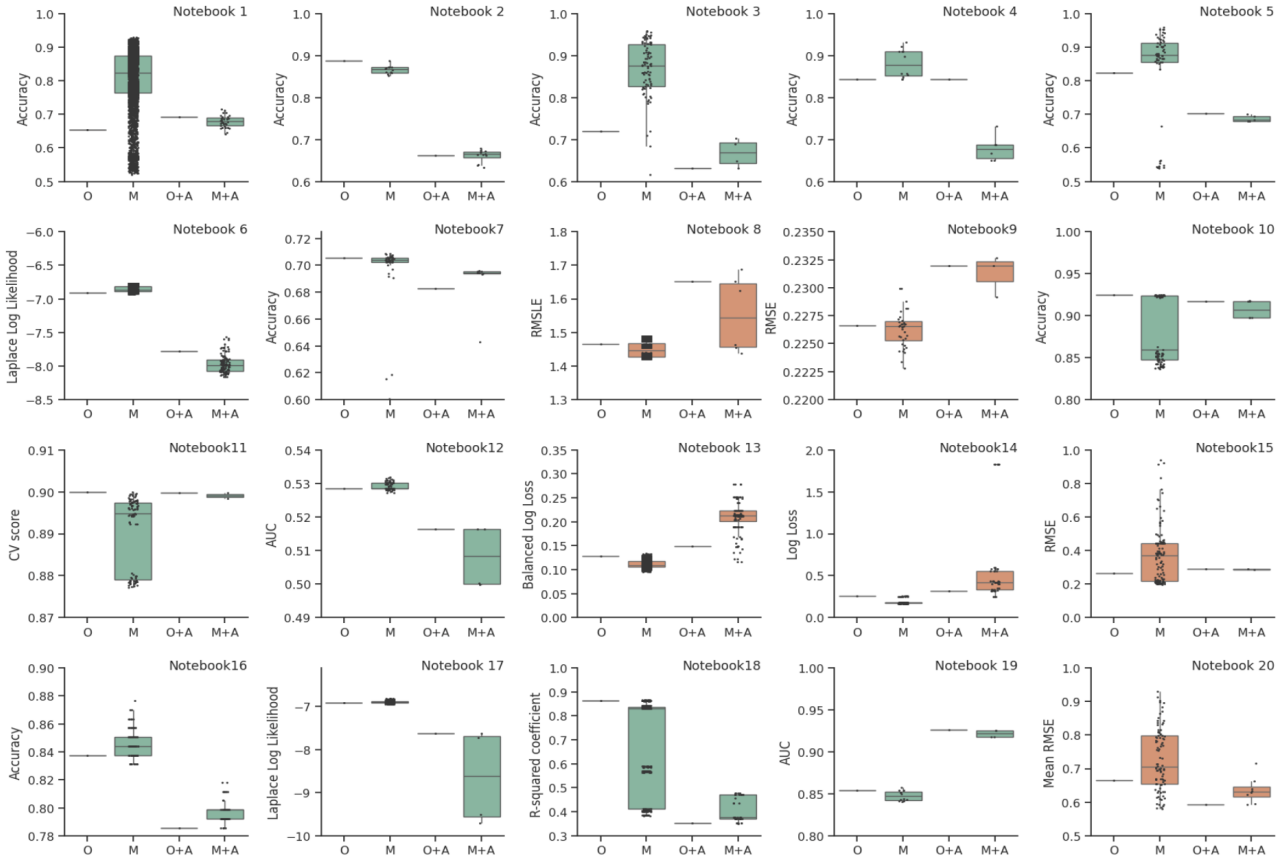


Fig. 5. Results of RQ3 and 4. The green color indicates **improved** performance of models with **higher** metric values on the y-axis, such as Accuracy and Likelihood. Conversely, the orange color signifies **better** model performance with **lower** metric values on the y-axis, like Loss and Error.)

each other due to low similarity in the data preparation, model configuration and hyperparameter optimization stages, as well as data format mismatch in the feature engineering stage. The average decrease in performance is 11.47%, with a maximum decrease of 22.5% (NB1) and a minimum decrease of 0.024% (NB13). For notebooks that share similar approaches and function calls (NB13, 14), after we integrate alternatives from each other, the performance is not obviously impacted, while for notebooks that vary much in implementation, integrating alternatives from each other will introduce a significant decrease in its performance (NB1, 2).

Finding: Our study shows the limitation of integrating alternatives from different ML pipelines, namely the lack of compatibility and coherence across implementations of different analysts. Despite the potential for improvement demonstrated in some notebooks, such integration often leads to disparate outcomes due to inconsistencies in data structures, processing methods, and underlying assumptions. A large amount of manual effort is usually required to remove these inconsistencies for a successful integration.

V. Discussions and Implications

A. *Exploration cost within the search space*

Computational notebooks offer a great platform for exploratory programming, yet they fall short in facilitating the exploration of alternatives at every ML stage smoothly, owing to their linear layout. In this study, we investigate the potential of the unexplored search space. While we successfully identified at least one alternative combination that surpassed the original submission in the majority of instances (16 out of 20), the exhaustive search through the entire space, which grows exponentially with the number of past alternatives, demands a significant investment of time and resources. Given the massive number of possible combinations of alternatives as discussed in RQ3, it can be seen that the cost of exhaustive search does not justify the increase in performance. Additionally, incompatibility errors are likely to occur when alternatives from different stages of the ML pipeline are combined together. For instance, out of all possible feature engineering alternatives, only a subset of them will be compatible with a specific model configuration since the model expects some particular features. As a result, our findings underscore that

manually exploring different combinations is not practical, highlighting the need for future assistance and direction to navigate the search space effectively.

B. Limitation of AutoML.

Although AutoML toolkits are invented to mitigate such problems and automate some of the explorations, especially during model configuration and hyperparameter optimization stages [31], doubts remain about the practical usability of these AutoML toolkits [47]. Potential reasons for AutoML yielding worse results can be attributed to several factors: (1) Limited search space of AutoML tools: AutoML tools come with a pre-defined search space for algorithms, hyperparameters as well as model architectures. It is possible that our problem requires a specific algorithm or model that falls outside of the default search space of AutoML tools; and (2) Improper feature engineering: AutoML tools like TPOT provide automated feature engineering as part of their pipeline optimization process including normalization, feature selection and dimension reduction. In our study, since we are feeding AutoML tools with data that is already processed and cleaned, it is possible that AutoML tools have performed redundant feature engineering which negatively impacts the model accuracy. Future studies could broadly explore the compatibility and ease of use of AutoML toolkits, aiming to improve their integration into the notebook environment.

C. Tooling support for organizing and exploring alternatives

With the surge in complexity and scale of ML pipelines, existing tools have substantially streamlined ML lifecycle management, primarily in experiment tracking, result visualization, and version control. Nevertheless, there still remains a gap in the exploration and understanding of pipeline alternative search spaces. In light of our experimental findings, we propose several future research and tooling directions.

Automatically extracting the alternative entities for each ML stage. Most existing tools support model and data versioning. Manual tools require the user to label and register stages and versions of the ML pipeline. While previous research has explored ways to automatically tag source code associated with different ML stages [48], [49], there are currently no ready-made tools that have been incorporated into the computational notebook environment. Future research could design automatic tools that require minimal labelling overhead, yet compromise on the granularity of control and usually require extra wrapper code or hooks to track experiments. Additionally, integrating code to streamline the construction of executable ML pipelines would be advantageous.

Search Interval: Cumulative vs Iterative. Considering the search space may expand exponentially, it would be beneficial to progressively introduce possible combinations of alternatives as analysts contribute new options. A cumulative search is when the search is only employed when the manual searches and experiments are completed. All registered versions will be combined to analyze the best alternative. Iterative search attempts to explore the search space each time a new checkpoint is registered. This approach is more computationally intensive, but the immediate feedback of alternatives could provide real-time insights that contribute to future experiments.

VI. Threats to Validity

In structuring our study to minimize bias and mitigate the impact of random noise, we acknowledge the possibility that our strategies might not have been fully effective. This section examines potential validity threats to our research.

Regarding internal validity, our empirical research required significant manual intervention, potentially introducing subjectivity and bias. To mitigate this risk, we adopted an open coding approach: two authors independently verified and cross-checked all findings. Any disagreements in labelling were discussed and resolved by a third author. Moreover, the potential exists for errors in the source code when we manually integrate multiple notebooks into one, possibly due to our lack of familiarity with both the notebook and the dataset domain. To counteract this risk, we have conducted thorough testing of our implementation. Additionally, we have designated two authors to each notebook to ensure they reach a consensus and obtain consistent results.

Concerns regarding external validity relate to the extent to which our findings can be generalized. The dataset used for the study contains ML pipelines sourced exclusively from Kaggle, specifically utilizing 52 notebooks for addressing RQ1&2, and 20 notebooks to answer RQ3-5. The selection was limited to high-quality competition notebooks ranking in the top 20%, ensuring the inclusion of comprehensive ML pipelines and sufficient alternative explorations. However, this selection criterion may restrict the generalizability of our conclusions to more typical and less ideal scenarios. Therefore, constructing a more extensive dataset is necessary to draw more broadly applicable conclusions. Furthermore, our experiments are restricted to four widely-used AutoML tools, which may not fully represent the breadth of available toolkits. As a result, caution is advised when generalizing the study's findings beyond this scope. Future research will aim to include other emerging tools to improve the generalizability of the results.

VII. Conclusion

In this project, we examine the methods data scientists use to explore alternatives with computational notebooks and evaluate the possibilities of unexplored ML pipelines that could deliver superior results. However, qualitative analysis and the manual construction of ML pipelines are labour-intensive. We suggest directions for future research and the development of tools to enhance support for exploratory programming.

Acknowledgment

Zhou and Zou's work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN2021-03538. We also thank Jimmy Yang, Tina Yang, and Chan Hyuk Yang for their assistance in exploring the direction, conducting preliminary analyses, and collecting data.

REFERENCES

[1] M. B. Kery, A. Horvath, and B. A. Myers, "Variolite: Supporting exploratory programming by data scientists." in *Proc. Conf. Human Factors in Computing Systems (CHI)*, vol. 10, 2017, pp. 3 025 453–3 025 626.

[2] 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 *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2020, pp. 1–12.

[3] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers, "The story in the notebook: Exploratory data science using a literate programming tool," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2018, pp. 1–11.

[4] S. Alspaugh, N. Zokaei, A. Liu, C. Jin, and M. A. Hearst, "Futzing and moseying: Interviews with professional data analysts on exploration practices," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 22–31, 2018.

[5] K. Patel, J. Fogarty, J. A. Landay, and B. Harrison, "Investigating statistical machine learning as a tool for software development," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2008, pp. 667–676.

[6] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2018, pp. 1–12.

[7] N. Weinman, S. M. Drucker, T. Barik, and R. DeLine, "Fork it: Supporting stateful alternatives in computational notebooks," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2021, pp. 1–12.

[8] K. Eckelt, K. Gadhave, A. Lex, and M. Streit, "Loops: Leveraging provenance and visualization to support exploratory data analysis in notebooks," in *COMPUTER GRAPHICS forum*, vol. 43, no. 3, 2024.

[9] M. B. Kery and B. A. Myers, "Exploring exploratory programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2017, pp. 25–29.

[10] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2019, pp. 1–12.

[11] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A large-scale study about quality and reproducibility of jupyter notebooks," in *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 2019, pp. 507–517.

[12] S. Samuel and B. König-Ries, "Provbook: Provenance-based semantic enrichment of interactive notebooks for reproducibility." in *International Semantic Web Conference(ISWC) (P&D/Industry/BlueSky)*, 2018.

[13] D. Kerzel, S. Samuel, and B. König-Ries, "Towards tracking provenance from machine learning notebooks." in *International Joint Conference on Knowledge Discovery and Information Retrieval(KDIR)*, 2021, pp. 274–281.

[14] K. Eckelt, A. Hinterreiter, P. Adelberger, C. Walchshofer, V. Dhanoa, C. Humer, M. Heckmann, C. Steinparz, and M. Streit, "Visual exploration of relationships and structure in low-dimensional embeddings," *IEEE Transactions on Visualization and Computer Graphics*, 2022.

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

[16] "Replication package." 2024. [Online]. Available: https://zenodo.org/records/12696888

[17] Y. Wang, Y. Wang, T. Zhang, Y. Yu, S.-C. Cheung, H. Yu, and Z. Zhu, "Can machine learning pipelines be better configured?" in *Proc. Int'l Conf. Software Engineering (ICSE)*, 2023, pp. 463–475.

[18] 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 *Proc. Int'l Conf. Software Engineering (ICSE)*. IEEE, 2019, pp. 291–300.

[19] J. Liu, N. Boukhelifa, and J. R. Eagan, "Understanding the role of alternatives in data analysis practices," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 1, pp. 66–76, 2019.

[20] S. Titov, Y. Golubev, and T. Bryksin, "Resplit: Improving the structure of jupyter notebooks by re-splitting their cells," in *Proc. Int'l Conf. Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2022, pp. 492–496.

[21] D. Kang, T. Ho, N. Marquardt, B. Mutlu, and A. Bianchi, "Toonnote: Improving communication in computational notebooks using interactive data comics," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2021, pp. 1–14.

[22] J. Wenskovitch, J. Zhao, S. Carter, M. Cooper, and C. North, "Albireo: An interactive tool for visually summarizing computational notebook structure," in *2019 IEEE visualization in data science (VDS)*. IEEE, 2019, pp. 1–10.

[23] A. Mathisen, T. Horak, C. N. Klokmose, K. Grønbæk, and N. Elmqvist, "Insideinsights: Integrating data-driven reporting in collaborative visual analytics," in *Computer Graphics Forum*, vol. 38, no. 3. Wiley Online Library, 2019, pp. 649–661.

[24] A. Y. Wang, D. Wang, J. Drozdal, M. Muller, S. Park, J. D. Weisz, X. Liu, L. Wu, and C. Dugan, "Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks," *ACM Transactions on Computer-Human Interaction*, vol. 29, no. 2, pp. 1–33, 2022.

[25] C. Yang, S. Zhou, J. L. Guo, and C. Kästner, "Subtle bugs everywhere: Generating documentation for data wrangling code," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 304–316.

[26] A. M. McNutt, C. Wang, R. A. Deline, and S. M. Drucker, "On the design of ai-powered code assistants for notebooks," in *Proc. Conf. Human Factors in Computing Systems (CHI)*, 2023, pp. 1–16.

[27] H. Dong, S. Zhou, J. L. Guo, and C. Kästner, "Splitting, renaming, removing: a study of common cleaning activities in jupyter notebooks," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, pp. 114–119.

[28] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[29] G. U. Md Abdullah Al Alamin, "How far are we with automated machine learning? characterization and challenges of automl toolkits," *Elsevier Empirical Software Engineering (EMSE)*, 2024.

[30] C. Wang, Z. Chen, and M. Zhou, "Automl from software engineering perspective: Landscapes and challenges," in *Proc. Working Conf. Mining Software Repositories (MSR)*. IEEE, 2023, pp. 39–51.

[31] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray, "Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–24, 2019.

[32] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 2015, pp. 1–10.

[33] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 979–984.

[34] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, "Cognito: Automated feature engineering for supervised learning," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2016, pp. 1304–1307.

[35] "Winning solutions of kaggle 270 competition." 2023. [Online]. Available: https://www.kaggle.com/code/sudalairajkumar/winning-solutions-of-kaggle-competitions

[36] "Top 10 automated machine learning(auto ml) tools used in 2020-2021," Online. [Online]. Available: https://www.linkedin.com/pulse/top-10-automated-machine-learningauto-ml-tools-used-2020-2021-sahu/

[37] "Autokeras." [Online]. Available: https://autokeras.com/

[38] "Tpot," https://epistasislab.github.io/tpot/.

[39] "Auto-sklearn." [Online]. Available: https://automl.github.io/auto-sklearn/master/

[40] "H2o automl." [Online]. Available: https://h2o.ai/platform/h2o-automl/

[41] J. Brownlee, "Tpot for automated machine learning in python," https://machinelearningmastery.com/tpot-for-automated-machine-learning-in-python/.

[42] R. S. Olson and J. H. Moore, "Tpot: A tree-based pipeline optimization tool for automating machine learning," in *Workshop on automatic machine learning*. Proceedings of Machine Learning Research(PMLR), 2016, pp. 66–74.

[43] "Kaggle competition-natural language processing with disaster tweets." [Online]. Available: https://www.kaggle.com/competitions/nlp-getting-started

[44] "Santander customer transaction prediction." [Online]. Available: https://www.kaggle.com/competitions/santander-customer-transaction-prediction

[45] "Icr - identifying age-related conditions." [Online]. Available: https://www.kaggle.com/competitions/icr-identify-age-related-conditions
[46] "Feedback prize - english language learning." [Online]. Available: https://www.kaggle.com/competitions/feedback-prize-english-language-learning
[47] M. A. A. Alamin and G. Uddin, "Challenges and barriers of using low code software for machine learning," *arXiv preprint arXiv:2211.04661*, 2022.
[48] Y. Jiang, C. Kästner, and S. Zhou, "Elevating jupyter notebook maintenance tooling by identifying and extracting notebook structures," in *Proc. Int'l Conf. Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 399–403.
[49] G. Zhang, M. A. Merrill, Y. Liu, J. Heer, and T. Althoff, "Coral: Code representation learning with weakly-supervised transformers for analyzing data analysis," *EPJ Data Science*, vol. 11, no. 1, p. 14, 2022.