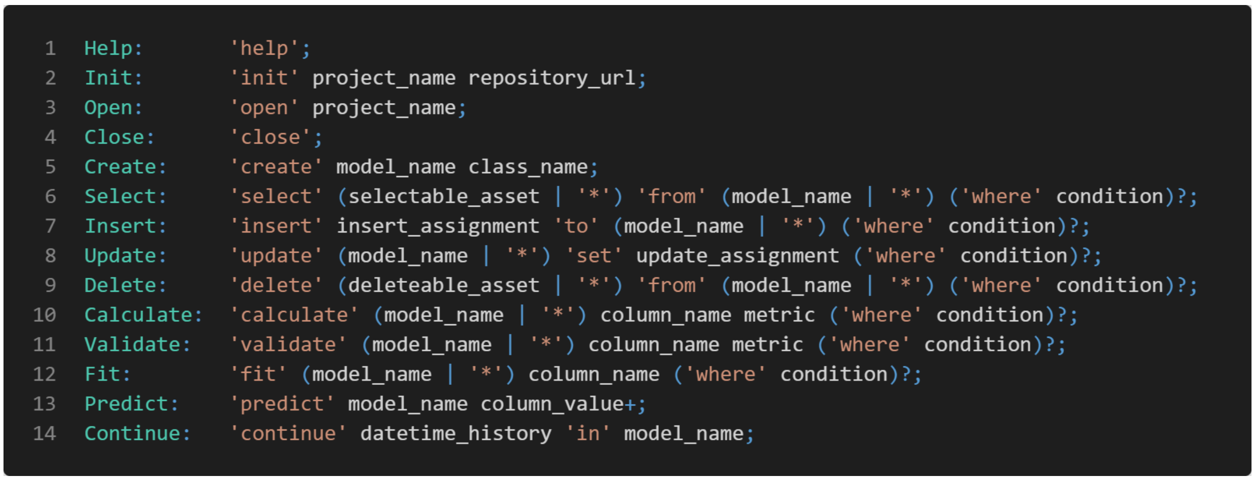
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



```
Help:       'help';
Init:       'init' project_name repository_url;
Open:       'open' project_name;
Close:      'close';
Create:     'create' model_name class_name;
Select:     'select' (selectable_asset | '*') 'from' (model_name | '*') ('where' condition)?;
Insert:     'insert' insert_assignment 'to' (model_name | '*') ('where' condition)?;
Update:     'update' (model_name | '*') 'set' update_assignment ('where' condition)?;
Delete:     'delete' (deleteable_asset | '*') 'from' (model_name | '*') ('where' condition)?;
Calculate:  'calculate' (model_name | '*') column_name metric ('where' condition)?;
Validate:   'validate' (model_name | '*') column_name metric ('where' condition)?;
Fit:        'fit' (model_name | '*') column_name ('where' condition)?;
Predict:    'predict' model_name column_value+;
Continue:   'continue' datetime_history 'in' model_name;
```

Development of a Query Language for Improved Versioning Support for Machine-Learning-Based Systems

Master's thesis in Computer science and engineering

Erik Tran

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Development of a Query Language for Improved Versioning Support for Machine-Learning-Based Systems

Erik Tran

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Development of a Query Language for Improved Versioning Support for Machine-Learning-Based Systems

Erik Tran

Supervisor: Daniel Strüber, Department of Computer Science and Engineering
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A part of the grammar in the meta-language textX of the developed query language

Typeset in LaTeX
Gothenburg, Sweden 2022

Development of a Query Language for Improved Versioning Support for Machine-Learning-Based Systems

Erik Tran

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis is about development of a query language for improved versioning support for machine-learning-based systems, focusing on the perspective of software engineers. The motivation is to combine the worlds of software engineers and data scientists as they have to work together effectively. Different existing tools that support management of machine learning assets are described and the reason why they are not fit for software engineers are explained. A design science approach method is applied for this thesis. Methods such as requirement elicitation, artifact feature elicitation and artifact feature prioritizations have been applied. Requirements were formed through independent research and are evaluated in four interviews. Features were implemented based on the requirements, and are evaluated as well in another four interviews. The artifact feature prioritization method includes construction of a traceability matrix. The final evaluation results indicated that the population who would hypothetically use this query language in its current state are users who are less experienced in management of machine learning assets. Discussions regarding future work such are related to scalability and data analysability are discussed in the report. The query language could expand to more advanced users if more advanced features are implemented e.g. features that supports data analysability or features that supports other models so that it is not restricted to only Scikit-learn classes that currently are the only classes that are able to be created by using the query language.

Keywords: software engineering, query language, machine learning, asset, versioning.

Acknowledgements

I would like to thank my supervisor Daniel Strüber at Chalmers for providing me with his guidance on the project and valuable feedback on writing my thesis.

I would also like to thank my examiner Jennifer Horkoff at Chalmers for providing me with her valuable feedback on the thesis report.

Finally, I would like to thank all the study participants who volunteered their time for interviews. Their valuable insights were very useful and helped with evaluating the thesis project.

Erik Tran, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1 Introduction

Data is the foundation of machine-learning-based systems. There are different types of data in such systems such as modeling code, implementation code, metadata, training data and model data. Tools that manage data with high traceability and versioning support are of great importance for machine-learning-based systems that evolve over time, since they can help reduce the work that needs to be done by developers, and also because developers might require a better comprehension of the evolution of such systems. Creating a version management tool for complex machine-learning-based systems while trying to achieve high interpretability, performance efficiency and interoperability is a challenging task.

This thesis is about development of a query language that is tailored to machine-learning-based systems, with a focus on improvement of versioning support for such systems by revising existing knowledge in the area. Such a system can help software engineers with retrieval of historical data that they might need from a version history.

The software engineering research areas in this thesis topic will partially be about knowledge seeking and mainly be about solution seeking. The research strategy that is going to be employed for this thesis work is a design science study type of research strategy as it is about development of an artifact which will be evaluated with input from users with relevant expertise.

1.1 Background and Motivation

There already exist several tools with versioning support for machine learning assets but are more towards data scientists' perspective and less effective for software engineers. The operations in such tools such as storing, tracking and querying data in a version history are ineffective. The ineffectiveness of these tools in addressing the needs of software engineers is explained more in Chapter 2.

This thesis is about the development of such a tool with improved version management support for machine learning assets that is focused on software engineers' perspective. This is accomplished by developing a tool-supported query language that offers domain-specific query functionalities for machine learning assets, being tailored to software engineers by being integrated with the available versioning system Git [1].

One of the major differences between software engineers' perspective and data scientists' perspective is that software engineers' role is to analyze user requirements and build a system that fulfills these requirements while data scientists' role is to analyze and manipulate big data. The motivation is to combine the world of software engineers and data scientists as both have to work together.

This thesis aims to develop a query language with effective versioning support for machine-learning-based systems that is more focused towards software engineers' perspective by partially examining existing tools with versioning support. This thesis explores different approaches in development of a querying tool with a high usability and comprehensibility that is more towards software engineers' perspective. This thesis research involves exploration of design space of possible solutions in a query language. In the end, the goal is a query language that supports interpretability, performance efficiency and interoperability in machine-learning-based software with a focus on the software engineers' perspective.

Query Language
A query language is a computer programming language that retrieves data from a data storage by sending queries [2]. A query is a request for data from a data storage [3]. A data storage can either be a database or any text file that contains data for extraction. The validity of a query command is valid if it is syntactically correct. The correctness of a query command syntax is determined by the defined grammar of the query language. A query for data can involve execution of code that e.g. extracts data and performs calculations. A query language typically can create, access and modify data. A query language can also be extended with other additional functionalities with the capability to execute code.

Machine Learning Experiment Management Tools
A machine learning experiment management tool is a tool able to plan, track and retrieve machine learning experiments and assets. Such tools can support developers and data scientists when building machine learning software systems. Such tools allows tracing back different versions of experiment runs [4].

1.2 Problem Description

When developing artificial intelligence enabled software systems (i.e., software with integrated artificial intelligence components), it is desirable to benefit from version control support. This is for the two main reasons commonly used for adopting version management systems: 1. To support collaborative development, 2. To be able to trace the version history of a particular project, allowing to retrieve and compare previous versions.

Commonly used version management systems such as Git are efficient to store and manage code, but not as efficient to store and manage machine learning assets. The reason for this is that Git does not have advanced, domain-specific functionalities. For example, it does not allow querying the version history to retrieve those model versions that had an accuracy score of at least 0.8. Same reason goes for existing querying approaches in classical version control systems such as PostgreSQL [5] as they are on the wrong abstraction level, and do not have advanced querying functionalities that are specific for machine learning assets.

Although there exist a few dedicated tools with versioning support for machine learning assets such as Neptune [6] [7] that can store, track and query datasets from a version history, these tools are more tailored towards the data scientists' perspective and less towards software engineers' perspective. Since they are not connected to established version control systems, they would require software engineers to adapt a completely different kind of tooling that they are used to for the management of machine learning assets. As both data scientists and software engineers have to collaborate with each other, there is a need for effective storage, tracking and querying datasets of machine learning assets based on available versioning systems. This would allow software engineers to use the systems they are familiar with, while eventually also providing support for data scientists via available interfaces to their dedicated tools, such as Neptune.

Thus, the problem addressed is a lack of querying tools with versioning support that are dedicated towards software engineers, allowing them to develop artificial intelligence enabled software systems with the kind of tools that they are familiar with. This problem results in redundancy in data storage with version history, and collaboration becomes less feasible [8]. It is currently still challenging to create tools with versioning support for machine-learning-based systems, while aiming to achieve a high model interpretability, performance efficiency and interoperability for both software engineers and data scientists. To confirm the quality of a tool, it is essential to conduct some type of research method such as interviews, and evaluate the tool with it. Although by creating a tool that favors software engineers over data scientists, there already exists several dedicated tools that are more tailored towards data scientists. The goal is to make a tool that allows effective collaboration between both software engineers and data scientists.

1.3 Research Questions

In this thesis research, there are several research questions (RQs) that will be addressed.

- **RQ1**: How to design a query language with versioning support for machine learning assets that is focused towards software engineers?
- **RQ1.1**: What querying functionalities do software engineers need and should be included in such a query language?
- **RQ1.2**: How to implement such a query language on top of an existing version control system?
- **RQ2**: To which extent does the developed query language achieve its goal of enabling effective versioning of machine learning assets?

1.4 Thesis Outline

Chapter 1 introduces briefly on the background and motivation, problem description and research questions of this thesis project.

Chapter 2 consists of brief descriptions of prior work that are related to this thesis project as well as parts that make this thesis project differ from prior work.

Chapter 3 explains the design cycles applied in this thesis project, requirement elicitation based on prior related work, artifact feature elicitation based on elicited requirements, artifact feature prioritizations based on initially elicited and defined features. The decision of the technology stack utilized for the development of the artifact which in this case is the developed query language is explained. The evaluation method utilized for the developed query language is described. Finally, the requirement satisfaction methodology is described.

Chapter 4 provides the results generated by using every mentioned method that are applied in this thesis. Results such as the requirements, the features and the language structure of the developed query language are described and explained. Finally, evaluation results are provided.

Chapter 5 provides an example of usage scenarios of the query language.

Chapter 6 discusses potential improvements of the query language and the threats to validity of this thesis.

Chapter 7 concludes this thesis with a concise description of the results and discusses possible future work such as further development of the query language.

2 Related Work

This chapter describes prior related work that are about machine-learning-based data management systems with versioning support. This would help with understanding flaws in such systems, and what types of requirements software engineers would need. Each paragraph in this section is about a different topic. Each topic relates to requirements that might be needed in machine-learning-based systems. Additionally, each topic is identified with a number for referencing purposes further in multiple parts of the report.

Topic 1, Machine Learning Asset Management
According to a recently published survey paper about machine learning asset management tools [7], providing improved tool support for developers of artificial intelligence based systems is still one of the relevant challenges today in the field of artificial intelligence based systems. In the present, machine learning techniques are becoming critical components of many software systems, which has caused a large increase in adoption of machine learning techniques in many software systems in the information technology industry sector. The need for this adoption is becoming a critical component to many software systems because it is associated with large-scale development of systems that require utilization of machine learning techniques. Furthermore in the survey paper, it has been reported that there has been an increasing number of tools for tracking and managing machine learning experiments, which indirectly addresses the challenges of providing improved tools for machine-learning-based systems.

Topic 2, OrpheusDB
According to a few research papers published by Silu et al. [8] [9], data science teams spend significantly more time constructing, curating, and analyzing datasets because of the tremendous growth of datasets. Different versions of datasets are generated by data processing operations such as data transformation and cleaning, feature engineering and normalization. However, there is no tool that allows effective storage, tracking and querying versioned datasets, resulting in redundancy in versioned data storage, which makes collaboration less feasible. A tool that the authors and its team has created is called OrpheusDB which is a wrapper on top of a relational database for data management of structured data with versioning support. The tool OrpheusDB supports a range of querying functions by a mix of structured query language (SQL) and Git-style version commands. However, the tool is built specifically for relational databases only. An assumption that the authors have made is that SQL is best fit for querying data and versioning information. The goal that

their study differs from this thesis study is that while OrpheusDB focuses only on querying datasets, this thesis focuses more on querying machine learning assets, a term that includes datasets, model data and metadata.

Topic 3, Data Management in Large-Scale Machine Learning Pipelines
A research paper published by a team in Google Research [10] discussed the data management issues that arise in large-scale machine learning pipelines. The team focused on the issues related to understanding, validating, cleaning, and enriching training data. Fundamentally, the problem is that it is unclear on how to ensure that the data is valid. The result of using invalid data will lead to low quality machine learning models in return, which would affect the production in machine-learning-based systems. The authors of the research paper have acknowledged that production machine learning pipelines do not always provide the scaffolding required by existing solutions. This thesis aims to develop a query language with improved versioning support with a goal to solve these kinds of issues with data management with production machine learning pipelines.

Topic 4, DSVC and DataHub
According to a research paper published by Anant et al. [11], relational databases have limited support for data collaboration when working with large datasets. The authors tried to overcome this issue by proposing two integrated systems: a dataset version control system (DSVC) and a hosted platform built on top of DSVC called DataHub. The system DSVC has functionalities that allows data scientists to work around with divergent collections of datasets, and scales to larger and more structured datasets compared to Git. DataHub allows collaborative data analysis building on their DSVC and supports better interaction capabilities. DataHub also has some tool support such as data cleaning, data search and integration, and data visualization. During the building of the two integrated systems that the authors proposed, they have addressed several challenges regarding management and querying large multi-version datasets that do not apply to regular source-code version control. A challenge the authors had during development of DSVC was regarding how to define a conflict, detect conflicts between branches, and merge non-conflicting divergent branches. Traditional source-code version control systems such as Git define conflicts by concurrent modifications to the same line. This is semantically equivalent to detecting row-level conflicts for structured datasets. There would be no lost update between two branches if there is a change of altering disjoint attributes for overlapping rows in one of the branches, but this still leads to conflicts. This is one of the problems in traditional version control systems when working with large-scale datasets. Their study is somewhat relevant to this thesis. DataHub is a platform that enables querying capabilities of large-scale datasets, while the goal of this thesis is to develop a query language for machine learning assets that includes datasets, model data and metadata.

Topic 5, DVC
A research paper published by Amine et al. [12] discussed about usage of tools such as Data Versioning Control (DVC) that enables versioning support for machine

learning data, models, pipelines and model evaluation metrics. Machine learning artifacts need to be synchronized with each other and as well with the source and test code of the software applications that utilize the machine learning models. Therefore, coupling between software artifacts needs to be managed and updated. According to the studies of the research paper, the results of 391 GitHub projects using DVC showed that more than half of the DVC files in a project are changed at least once every one-tenth of the project's lifetime. Additionally, the DVC files had a tight coupling between other artifacts. Around a quarter of pull requests of change in source code and around half of pull requests of change in tests required changes in the DVC files of the projects. An average of 78% of the studied projects that utilized tools such as DVC showed a non-constant pipeline complexity. As reported in the paper, it requires frequent and continual changes of the DVC files of a project in order to manage connection between other artifacts. In order to tackle such complexity, this thesis aims to develop a query language that does not require such a file that needs continual changes.

Topic 6, Large-Scale Data Analytics and Statistical Machine Learning
A research paper published by Arun et al. [13] discussed large-scale data analytics using statistical machine learning in many modern data-driven applications and that developers have been tackling data management related challenges that arise in machine learning workloads. Many systems that include advanced analytics have been built to tackle this challenge regarding data management of machine learning assets. The research paper reviews and analyzes key data management challenges and techniques such as query optimization, partitioning, and compression in machine-learning-based systems. That is relevant to this thesis as the goal of this thesis is to develop a query language with as low complexity as possible, and this is achieved by utilizing the data management techniques that the research paper has discussed.

Topic 7, Google Colaboratory
According to a research paper published by Mary et al. [14], some prior work findings revealed that data scientists are under-utilizing versioning tools like Git, which made collaboration in data science experimental tasks more difficult. An integrated development environment that data scientists commonly utilize for collaboration in data science experimental projects is Google's Colaboratory [15]. Furthermore, some other findings that the research paper has found is that software engineering's way of versioning by utilizing Git has usually been avoided. This is because Colaboratory already provides versioning support similar to Google Docs [16] way of versioning. Colaboratory allows a real-time collaboration in a computational notebook where users can simultaneously edit. Although data scientists utilize Colaboratory for real-time collaborative data science, they usually have a difficult time with understanding past experiments by reading the version history provided by Colaboratory. A functional requirement that is highly favored for the query language is model versioning support.

Topic 8, Computational Notebooks
Another research paper published by Souti et al. [17] has pinpointed nine pain

points, three of them being "Share and Collaborate", "Reproduce and Reuse" and "Archival" which are related to versioning, by utilizing computational notebooks. The pain point "Reproduce and Reuse" also describes the importance of replicating or reusing code in order to achieve high variability with model hyperparameters and environment dependencies. Again, this is a pain point of utilizing computational notebooks such as Colaboratory Notebook or Jupyter Notebook [18]. A functional requirement that is highly favored is model reproducibility as it enables high variability which allows model optimizations.

Topic 9, Reproducibility in Machine Learning Pipelines
A research paper published by Peter et al. [19], discusses the importance of reproducibility in machine learning pipelines, and the lack of reproducibility of modeling is an existing problem for any machine learning practitioners. The lack of reproducibility in machine learning pipelines causes significant financial costs, lost time and sometimes personal reputation as well. Data provenance refers to how data was collected historically, and is the most difficult challenge to ensure reproducibility. In other words, to achieve high reproducibility, data provenance needs to be achieved a priori.

3
Method

This chapter describes how the query language is developed in an iterative approach by following several guidelines. These guidelines are described in the design science method that is applied in this thesis, in Section 3.1. Section 3.2 explains how the requirements are elicited and formed. Section 3.3 explains each elicited feature that is based on the elicited requirements. Additionally, the elicited features are prioritized in Section 3.4 utilizing a prioritization method with traceability matrix. Section 3.5 explains the overall decision of the technology stack utilized for the development of the query language. The evaluation method utilized for evaluating requirements and features are described in Section 3.6. Finally, the requirement satisfaction method that is utilized is described in Section 3.7.

3.1 Design Science Cycles

To develop a query language that has a specific purpose, it is necessary to apply a design science method. The work flow of this thesis follows an existing design science method [20]. The design science method consists of in total seven guidelines.

Guideline 1 is about defining the artifact early to confirm whether it involves some type of knowledge contribution or not. In the case of this thesis, the artifact is the developed query language.

Guideline 2 is about working in iterations to improve the artifact and the knowledge in each iteration as well as contributing to each research question in each iteration. The length of each iteration is one week.

Guideline 3 is about defining the research questions where each research question is related to a specific point e.g. the problem, the solution and the evaluation. The research questions in this thesis are presented in Section 1.3.

Guideline 4 is about having regular meetings, where the student and the supervisor discusses current progress at each iteration, and what to do for next iterations, and how to improve progress.

Guideline 5 is about shift emphasis between cycles, where each research question should be focused on each cycle. A cycle consists of several iterations. The length of a cycle is approximately one month. In the case of this thesis, the length of each cycle is approximately six weeks. Each cycle in this thesis addresses each research question to a preliminary extent as listed below. All cycles are connected to each research question, but each cycle is more focused to a specific research question.

- **Cycle 1** focuses on **RQ1** and **RQ1.1**.
- **Cycle 2** focuses on **RQ1.2**.
- **Cycle 3** focuses on **RQ2**.

Guideline 6 is about describing the artifact precisely and concisely. The description of the artifact benefits the reader and allows easier understanding of other sections regarding discussions such as threats to validity (see Section 6.5) and future work (see Section 7.2), to focus on the learning and less on the developed artifact itself.

Guideline 7 is about writing the thesis report iteratively to fill in findings found in chronological order during development in each cycle.

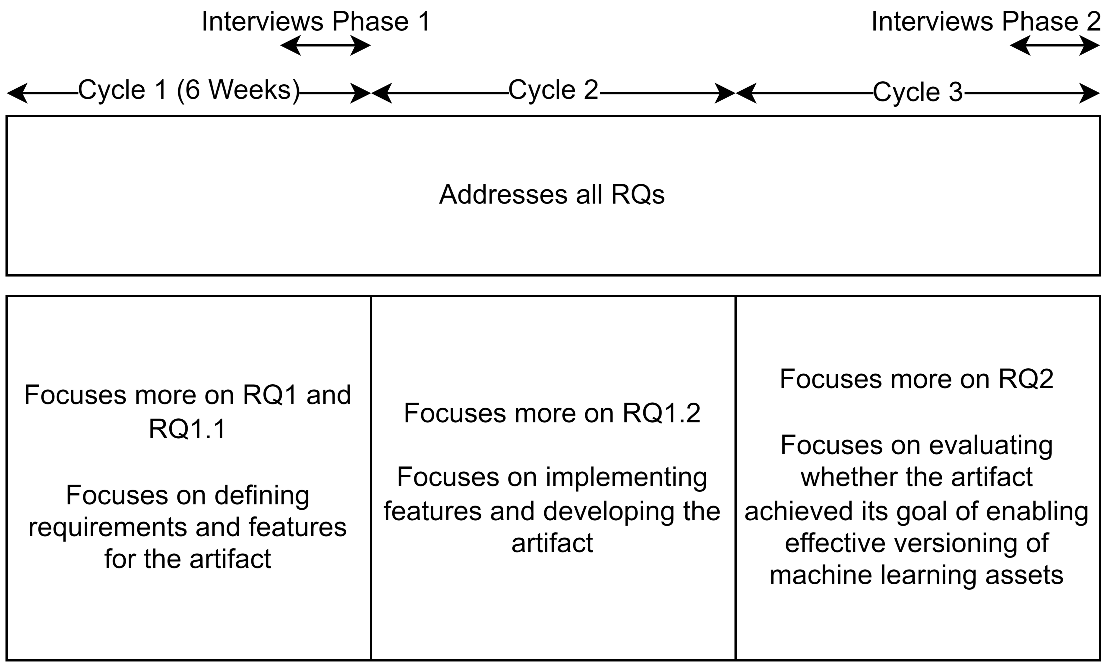

Interviews Phase 1
Interviews Phase 2
Cycle 1 (6 Weeks)
Cycle 2
Cycle 3
Addresses all RQs
Focuses more on RQ1 and RQ1.1
Focuses on defining requirements and features for the artifact
Focuses more on RQ1.2
Focuses on implementing features and developing the artifact
Focuses more on RQ2
Focuses on evaluating whether the artifact achieved its goal of enabling effective versioning of machine learning assets

Figure 3.1: Timeline of design science cycles.

3.2 Requirement Elicitation

To define requirements for the query language, it is necessary to conduct requirement elicitation methods. Conducting these requirement elicitation methods helped with generating different requirements. These requirements are categorized into two categories: functional requirements and non-functional requirements.

The elicited functional requirements were initially determined mainly by independent research about prior related work (see Chapter 2) and systematically analysing different related dedicated tools' functionalities. Chapter 2 contains several different topics where each topic relates to requirements that might be needed in machine-learning-based systems. One of the elicited functional requirements was initially also determined partially by the knowledge of the author of this thesis. This knowledge is about understanding how to build a query language on SQL-like commands which is a crucial prerequisite knowledge for understanding the topic of development of a query language. This knowledge is described in Section 1.1

The elicited non-functional requirements were initially determined by systematically analysing a standard reference list of non-functional requirements, as provided by ISO/IEC 25010:2011 [21]. The relevance of each quality aspect from ISO/IEC 25010:2011 for the system was manually assessed, and kept those requirements that were deemed as most relevant. Each non-functional requirement that is kept is further analysed by independent research in different related work.

Each requirement is explained in detail about how they are derived and motivated in Section 4.1. Conducting these elicitation methods also helped generate ideas of potential feasible features for the query language.

An interview-based evaluation method is conducted to gather insights from interviewees about the initial elicited requirements. This confirms the initially elicited requirements and potentially elicits new requirements that can be formed by the interviewees' insights. This method is more described and explained in Section 4.5.

3.3 Artifact Feature Elicitation

To develop the query language, several initial features for it need to be elicited and defined a priori. The initial features are defined based on initial elicited requirements for the query language. A complete list of elicited features are provided in Section 4.2. Descriptions and detailed explanations on how each elicited feature is formed, and motivations on how each elicited feature satisfies the elicited requirements are provided.

An interview-based evaluation method was conducted during the first phase (see Figure 3.1) and insights were gathered. These insights helped with confirming the initial elicited features. Ideas for new potential features are gathered by the insights from the interviewees. This is more explained in Section 4.5.

3.4 Artifact Feature Prioritization

To prioritize which features to implement for the artifact first based on the elicited requirements and elicited features, a traceability matrix is created. It eases the development progress as it displays the importance of each elicited feature and eases decision-makings on which features to prioritize first for implementation. Detailed description for each requirement is provided in Section 4.1 and detailed description for each feature is provided in Section 4.2.

3.5 Artifact Technology Stack

Before developing the query language, features are elicited and defined beforehand in Section 3.3 and prioritized beforehand in Section 3.4 to ease the development progress. To develop the query language, a technology stack used for developing the query language is chosen. The technology stack chosen is based on the knowledge and preferences of the author of this thesis project. The main goal is to satisfy as many requirements as possible.

To develop the query language, the integrated development environment Visual Studio Code is utilized. The programming language Python is chosen as it contains many useful machine learning libraries such as Scikit-learn, PyTorch, TensorFlow and Keras. A meta-language framework that is used for development of domain-specific languages that includes support for regular expressions written in Python called textX [22] is utilized for the development of the query language. The machine learning library Scikit-learn is utilized as it provides a wide range of different machine learning modules of classes such as classification, regression and clustering modules. The GitHub repository data storage is utilized and acts as the database for the query language. To access GitHub repository data storage, the version control system Git is integrated and utilized in the query language.

3.6 Evaluation Method

Two phases of interviews were conducted to evaluate the developed query language. The first phase of interviews was conducted at the end of the first cycle. The second phase of interviews was conducted at the end of the last cycle. These cycles are previously displayed in Figure 3.1. Each phase of interviews consisted of four interviewees. The interviewees from the first phase were allowed to join the second phase of interviews. However, the second phase of interviews was planned to include several new interviewees who did not participate during the first phase of interviews as well to get as much fresh insights as possible. The interviewees' insights are systematically analysed in a thematic analysis. Most of the interviewees are students of my supervisor's current course called "Software engineering for data-intensive AI applications" which is a software engineering course that is precisely in the same context as our proposed thesis of developing an artificial intelligence enabled software system. Furthermore, students are useful as a stand-in for practitioners in research studies as there has been several studies suggesting that students practitioners and industry practitioners do not have a statistically significant difference, when performing tasks with tooling that the practitioners are not familiar with [23] [24]. Regarding the interviewees motivation to participate in an interview, there are benefits they could gain from it. A benefit that they can gain from this is that I could potentially also be a suitable candidate for their evaluation part of their potential future studies as well. Another benefit they could gain from this is potentially gain some ideas about different and unique ways of managing machine-learning-based systems.

Interviews First Phase

The goal of the first phase of interviews is to gain insights about the initially elicited requirements and making sure that the requirements are clear, which focuses on RQ1, specifically RQ1.1. At the time of conduction of interviews in the first phase, the query language is partially implemented. The interviewees are introduced with a brief background of the thesis project. Subsequently, the elicited requirements are displayed and explained. A quick demo of the partially implemented query language is presented. Finally, questions regarding the elicited requirements are asked. Some example questions asked are "*Based on your own experiences of developing machine-learning-based software, would the requirement model reproducibility be useful?*", "*Are there any missing requirements that would be good to have, based on your own experiences?*" and "*What do you think about the usage of the query language, based on what has been shown? What do you like, what needs improvement?*". A list of questions asked is provided in Appendix A.1. A summary of the conducted interviews during the first phase is provided in Section 4.5.

The choice of participants should be experienced enough. An assumption that most of the chosen participants are experienced enough is the fact that they have been lectured or/and supervised by my supervisor who is experienced in AI engineering [25].

Insights are gathered in each interview. The insights provides ideas about potential new requirements and new features that might be missing in the query language. It also helps with deciding whether the features that are defined but not implemented yet, should be implemented or not, e.g. the interviewees might suggest features that are similar compared to initially elicited and defined features. The insights also provides ideas about what the query language is capable of and what it could achieve.

Interviews Second Phase

The goal of the second phase of interview is to gain insights about the features and making sure that the features are clear, which focuses on RQ2. Before introducing the thesis project, the interviewees' experience are assessed with a few questions related to their experience related to this thesis topic. After that, the interviewees are introduced with a brief background of the thesis project. Next, a quick demo of the query language is presented which was done by following the steps provided in Chapter 5. Finally, questions regarding the features are asked.

This phase helps with gathering insights from interviewees to help with confirming that the implemented features for the query language are required to satisfy the elicited requirements. Some example questions asked are "*How would you rate your own understanding of the query language on a scale from 1 (very low) to 5 (very high)?*" and "*Based on your own experience in developing ML projects: How would* *you rate the usefulness of the query language on a scale from 1 (not useful at all) to 5 (very useful)?*". A list of questions asked is provided in Appendix A.2. The list contains a mix of open-ended questions, closed-ended questions, rating questions and multiple choice questions. Questions Q2.1 - Q2.4 are related to participants' experience in machine-learning-based systems and are asked before a demo of the project. This helped with assessing participants' experience and evaluating their insights on the query language. Questions Q2.5 - Q2.10 are related to features of the developed query language and are asked after a demo of the project and the developed query language. A summary of the conducted interviews during the second phase is provided in Section 4.6. A Goal-Question-Metric (GQM) approach [26] is applied for the second phase of interviews. A list of goals and questions with metrics are provided below.

- **Goal G1**: Evaluate the effectiveness of the querying tool.
  - **Question Q1**: Who would hypothetically use the querying tool? (RQ1)
    - **Metric M1**: Developers' rating on their own experience in machine-learning-based systems
    - **Metric M2**: Developers' rating on their own experience in query languages
    - **Metric M3**: Developers' rating on the usefulness of the query language of this thesis project
  - **Question Q2**: Which features are the most useful in the querying tool? (RQ1, RQ2)
    - **Metric M4**: Developers' subjective rating on the most useful feature
  - **Question Q3**: How easy is it to use the querying tool? (RQ2)
    - **Metric M5**: Developers' rating on their own understanding of the developed query language of this thesis project

There are interview questions that relates to each metric and are asked during the interviews. The final score of each metric (except metric M4) is the calculated mean score of the participants. Metric M4 is decided by a subjective rating by the participants. The feature that is most chosen as the most useful is the metric value. This metric would give some insights about the usefulness of the query language.

There are interview questions that lead to more detailed information. For GQM Q1, interview questions such as "*What are you studying?*" and "*Which study degree level?*" provides more detailed information on participants' background. For GQM Q2, interview questions such as "*Which features are the least useful?*", "*Which of the features need improvement? For each of them: what kind of improvement?*" and "*Do you have ideas of new features that can further improve the usefulness of the query language?*" provides more detailed feedback on the prototype.

3.7 Requirement Satisfaction

The interviewees during the first phase provided valuable insights. The insights helped with confirming the initial elicited requirements as well as providing ideas for new requirements. Initially defined requirements that the interviewees agree on helps with confirming the requirements. Interviewees might disagree with a requirement or a feature with an explanation. If the explanation makes sense, then the disagreed requirements or features are satisfied by either modifying the requirement descriptions, or removing the requirements, or modifying the features, or adding in new features. New requirements that were formed by the help of interviewees' insights are analysed by independently researching about it in relevant research papers. After confirming the need for the new requirements, new features are implemented for the query language to satisfy the new and confirmed requirements. The new requirements suggested by the interviewees are confirmed based on prior related work as well as partially based on the knowledge of the author of this thesis project.

The interviewees during the second phase also provided valuable insights which helped with evaluating the query language. The second phase interviews were also the final evaluation of the query language. Valuable feedback on features of the query language were provided by the interviewees. Features that needed improvement can be updated and improved in future. New feature ideas suggested by interviewees are taken into consideration and discussed in Section 7.2.

4
Results

This chapter presents the results by utilizing the methods mentioned in the previous Chapter 3. The initially elicited and defined requirements are described, explained and motivated. The new requirements formed by the insights gathered from the first phase of interviews are also provided. The features that were used to structure the development of the query language are provided. The traceability matrix that displays relationships between the requirements and features is provided. The implementation design of the artifact such as the grammar, the code, the structure and the versioning is described and explained. A summary of interviewees' insights about the query language is provided for each phase of interviews.

4.1 Requirements

Requirements are categorized in two categories: functional and non-functional. Each requirement is described in detail how they were derived and what they entail. Motivation for each requirement is explained. Additionally, each initially elicited requirement is evaluated during the first phase of interviews, and insights were gathered, which is more described in Section 4.5.

4.1.1 Functional Requirements

The query language is developed initially based on a few functional requirements. A list of initially elicited functional requirements is provided below.

- **FR1**: Asset Manipulation
- **FR2**: Model Versioning
- **FR3**: Model Reproducibility
- **FR4**: Data Provenance
- **FR5**: Model Interpretability

Asset Manipulation
This functional requirement is important as it is crucial to have asset manipulation capabilities as part of the query language in order to manipulate and manage machine learning assets. Asset manipulation is the core functional requirement for the query language. This requirement enables users of the query language to run different commands that manipulate machine learning assets. The query language is also

capable of running commands that execute code. By being able to execute code, the query language could provide more valuable data such as different performance metric values or prediction values of machine learning models. The query language should be able to manipulate machine learning assets such as different available machine learning model class-specific hyperparameters with domain-specific functionalities such as selecting, inserting, updating and deleting. The query language should also be able to create and train machine learning models. Additionally, the query language should also be able to calculate and validate different available performance metric values. The query language should also be able to predict values of trained machine learning models. All of these querying functionalities would enable tuning of machine learning model hyperparameters in many different paths for model optimizations. The query language should also be able to train the model as well. In conclusion, this functional requirement allows other mentioned functional requirements to be achievable.

The goal is to offer an integrated solution for machine learning experimentation and versioning. Therefore, the envisioned query language should offer support for domain-specific machine learning tasks as well as more traditional asset manipulation tasks.

Model Versioning
This functional requirement is derived from and motivated in Chapter 2, Topic 7. Model versioning is one of the main functional requirements for the query language as it would enable users to save different versions of machine learning assets such as binary file of the model and metadata file of the model. Each time a model asset is manipulated, a new version will be saved in the metadata with updated values including date time of when it happened, identification information of the person such as the GitHub email that conducted the asset manipulation. This improves traceability and data provenance when using the query language. The query language should also be able to execute regular Git commands which enables saving model assets in a GitHub repository. This would be useful in case files such as the binary file of the model or the metadata of the model are lost locally which could be saved by fetching the lost files from its GitHub repository. The ability to checkout to different branches would also help with improving versioning support. This would ease the evaluation and reviewing process of others' work in different branches and versions. It also allows creating backup versions. The goal is to achieve high collaboration between software engineers and data scientists.

Model Reproducibility
This functional requirement is derived from and motivated in Chapter 2, Topic 8 and 9. Model reproducibility is an important functional requirement. The ability to repeat an experiment and reproduce using the same model but with different model hyperparameters is one of the functional requirements. The goal of this requirement is to ease the ability of tuning hyperparameters in order to achieve faster and continuous improvements of machine learning models. Models with updated hyperparameters are able to be evaluated by calculating and checking the class-specific

performance metric values of the model. The query language also enables machine learning models to go back to previous versions that are tracked in the metadata, which is the main purpose of this functional requirement.

Data Provenance
This functional requirement is derived from and motivated in Chapter 2, Topic 9. Data provenance is an important functional requirement as well as it enables tracking abilities, such as being able to track when and how an asset is modified and by whom. This applies to all processing steps in a machine learning pipeline that includes data collection, data merging, data cleaning and model hyperparameter modifications. The goal of this requirement is to avoid "visibility debt" in the context of technical debt, which means to avoid missing data dependencies and histories [27].

Model Interpretability
This functional requirement is derived from a research paper published by Zachary et al. [28] that discussed the importance of interpretability of machine learning models. A high interpretability benefits machine learning engineers with increased trust, transferability, informativeness and more fair and ethical decision-making. A high interpretability also benefits machine learning models with transparency. Interpretability is an important functional requirement in machine learning modeling and can benefit the query language of this thesis. When using a highly interpretable machine learning model created by the query language for predictions, it would ease the comprehension of how the predictions were processed. This would lead to an improved evaluation of the model.

During the first phase (see Figure 3.1), the query language is evaluated by an interview-based evaluation method (see Section 3.6) and insights were gathered. New functional requirements are formed and derived from interviewees' insights (see Section 4.5). A list of derived functional requirements elicited from insights from the first phase of interviews is provided below.

- **FR6**: Data Analysability

Data Analysability
This is a functional requirement derived from insights gathered during the first phase of interviews. Achieving this functional requirement enables the ability to analyse data by e.g. utilizing some kind of visualization tool. Utilization of visualization tools helps with understanding and interpreting data which helps with discovering importances in data such as outliers that might be crucial information. This improves evaluation of a model created using the query language by analysing the data. Ways to achieve data analysability is more discussed in Section 7.2.

4.1.2 Non-functional Requirements

The query language is also developed based on a few non-functional requirements. A list of initially elicited non-functional requirements is provided below.

- **NFR1**: Performance Efficiency
- **NFR2**: Interoperability

Performance Efficiency
This non-functional requirement is fetched from a standard reference list of non-functional requirements, as provided by ISO/IEC 25010:2011 [21]. Performance efficiency is a common and an important non-functional requirement for the query language. The query language should be able to load, dump and train models in a way where it does not slow down the query language overall. Machine learning models that utilize large datasets would require more time and data capacity to operate which would slow down the speed for model optimization improvements.

Interoperability
This non-functional requirement is fetched from a standard reference list of non-functional requirements, as provided by ISO/IEC 25010:2011 [21]. Interoperability is a non-functional requirement for the query language that would help with implementing features that could improve versioning support for machine learning assets. The query language should be able to integrate with a versioning control system such as Git and be able to fetch information that might be important such as data in different versions. The importance of versioning is also previously explained and motivated for the functional requirement FR2 in Section 4.1.1.

As mentioned, during the first phase (see Figure 3.1), the query language is evaluated by an interview-based evaluation method (see Section 3.6) and insights were gathered. New non-functional requirements are also formed and derived from interviewees' insights (see Section 4.5). A list of derived non-functional requirements elicited from insights from first the phase of interviews is provided below.

- **NFR3**: Scalability

Scalability
Scalability is a non-functional requirement derived from insights gathered during the first phase of interviews. A high scalability improves performance efficiency when working with large datasets. To achieve a high scalability, the query language would need to be able to manage large datasets in an effective way. Extracting data from a large dataset usually takes a long time. A high scalability reduces the time required to extract that data. Ways to achieve scalability is more discussed in Section 7.2.

4.2 Features

The query language is developed initially based on initial elicited requirements. Each feature is initially defined and formed based on the knowledge of the author of this thesis during iterative development of the query language during Cycle 1 and 2 (see Figure 3.1). Each feature group relates to one or several elicited requirements. Each feature group contains one or several features. Each feature contains an identifier and description. Feature identifiers are used for referencing purposes previously in Section 3.4, and does not indicate any kind of ordering when they are formed. Each feature is described about how they work. Each feature group has a purpose and motivates what its features entail e.g. satisfying specific elicited requirements. In addition to every mentioned requirement that each feature satisfies, each feature also tries to satisfy the requirement NFR1 by implementing these features in a way where it can achieve sufficient performance efficiency of the query language overall e.g. by improving code quality to avoid slowing down the query language in any possible ways.

Tables of initially derived features are provided below. Each table contains feature identifiers and feature description. These features are going to be evaluated by the help of several interviewees' insights during the first phase of interviews (see Figure 3.1). The whole evaluation process is previously described in Section 3.6.

F1.1	The query language shall be able to initialize a project.
F1.2	The query language shall be able to open a project.
F1.3	The query language shall be able to close a project.
F1.4	The query language shall be able to create a machine learning model of a specified class.

Table 4.1: Features that enable the ability to switch between projects as well as creating new machine learning models.
Motivation: Enables requirement FR1 and NFR2 to be satisfiable by enabling other features implemented to be utilized.

F2.1	The query language shall be able to select asset(s) of machine learning model(s).
F2.2	The query language shall be able to filter selected asset(s) by specifying a condition.

Table 4.2: Features that enable the ability to select asset(s) of machine learning model(s).
Motivation: To satisfy requirement FR1 and FR5.

F3.1	The query language shall be able to insert asset(s) to machine learning model(s).
F3.2	The query language shall be able to insert asset(s) to filtered machine learning model(s) by specifying a condition.

Table 4.3: Features that enable the ability to insert asset(s) to machine learning model(s).
Motivation: To satisfy requirement FR1.

F4.1	The query language shall be able to update asset(s) of machine learning model(s).
F4.2	The query language shall be able to update asset(s) of filtered machine learning model(s) by specifying a condition.

Table 4.4: Features that enable the ability to update asset(s) of machine learning model(s).
Motivation: To satisfy requirement FR1.

F5.1	The query language shall be able to delete asset(s) of machine learning model(s).
F5.2	The query language shall be able to delete asset(s) of filtered machine learning model(s) by specifying a condition.

Table 4.5: Features that enable the ability to delete asset(s) of machine learning model(s).
Motivation: To satisfy requirement FR1.

F6.1	The query language shall be able to calculate performance metric values of machine learning model(s).
F6.2	The query language shall be able to calculate performance metric values of filtered machine learning model(s) by specifying a condition.

Table 4.6: Features that enable the ability to calculate performance metric values of machine learning model(s).
Motivation: To increase usefulness of requirement FR1.

F7.1	The query language shall be able to validate performance metric values of machine learning model(s).
F7.2	The query language shall be able to validate performance metric values of filtered machine learning model(s) by specifying a condition.

Table 4.7: Features that enable the ability to validate performance metric values of machine learning model(s).
Motivation: To increase usefulness of requirement FR1.

F8.1	The query language shall be able to train machine learning model(s).
F8.2	The query language shall be able to train filtered machine learning model(s) by specifying a condition.

Table 4.8: Features that enable the ability to train machine learning model(s).
Motivation: To increase usefulness of requirement FR1.

F9	The query language shall be able to predict values using a machine learning model.

Table 4.9: Feature that enable the ability to predict values using a machine learning model.
Motivation: To increase usefulness of requirement FR1.

F10.1	The query language shall be able to save new versions of machine learning model(s).
F10.2	The query language shall be able to retrieve older versions of a machine learning model.

Table 4.10: Features that enable the ability to save new versions and retrieve older versions of a machine learning model.
Motivation: To satisfy requirements FR2, FR3 and FR4.

F11	The query language shall be able to execute Git commands.

Table 4.11: Feature that enable the ability to execute Git commands.
Motivation: To satisfy requirements FR3 and NFR2.

4.3 Traceability Matrix

The traceability matrix maps elicited requirements and elicited features. It contains rows of features and columns of requirements, which displays whether there are any dependencies between them with True (T) or False (F) values. Features are the user-visible functionality increments that were used to structure the development of the query language.

	FR1	FR2	FR3	FR4	FR5	NFR1	NFR2
F1.1	T	F	F	F	F	T	T
F1.2	T	F	F	F	F	T	T
F1.3	T	F	F	F	F	T	T
F1.4	T	T	T	T	T	T	T
F2.1	T	F	F	F	T	T	F
F2.2	T	F	F	F	T	T	F
F3.1	T	T	F	T	T	T	F
F3.2	T	T	F	T	T	T	F
F4.1	T	T	F	T	T	T	F
F4.2	T	T	F	T	T	T	F
F5.1	T	T	F	T	T	T	F
F5.2	T	T	F	T	T	T	F
F6.1	T	T	F	T	T	T	F
F6.2	T	T	F	T	T	T	F
F7.1	F	F	F	F	T	T	F
F7.2	F	F	F	F	T	T	F
F8.1	T	T	F	T	T	T	F
F8.2	T	T	F	T	T	T	F
F9	F	F	F	F	T	T	F
F10.1	T	T	T	T	T	T	T
F10.2	T	T	T	T	T	T	F
F11	T	T	T	T	F	T	T

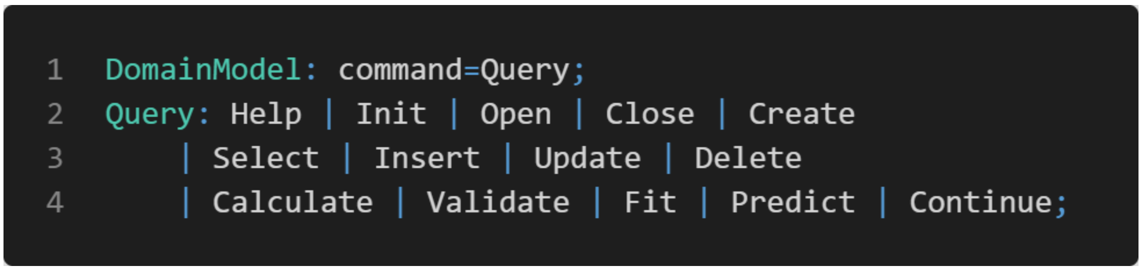
Table 4.12: Maps rows of elicited requirements and columns of elicited features. Displays whether there are any dependencies between them with True (T) or False (F) values.

4.4 Artifact Implementation Design

This section describes and explains the overall implementation design and structure of the developed query language. The implementation design is based on the technology stack utilized for development of the query language which is previously described earlier in Section 3.5. An example of usage scenarios of the query language and more detailed outputs are provided in Section 5.

4.4.1 Grammar

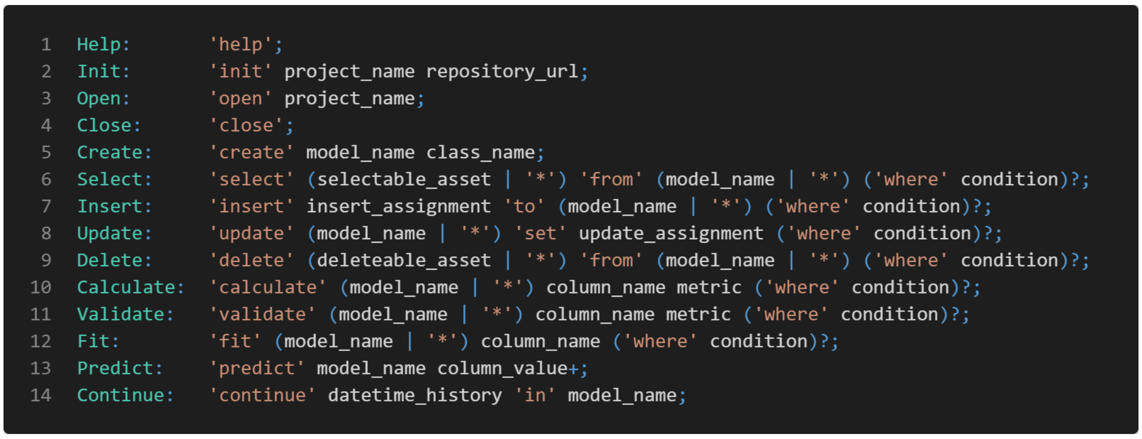
The grammar for the query language is written in textX. A list of main command initials of the grammar is provided in Figure 4.1 displayed below.



```
DomainModel: command=Query;
Query: Help | Init | Open | Close | Create
     | Select | Insert | Update | Delete
     | Calculate | Validate | Fit | Predict | Continue;
```

Figure 4.1: A list of main command initials of the grammar.

Definitions for each main command initials of the grammar is provided in Figure 4.2 displayed below. Brief explanations of how the grammar works are described.



```
 1  Help:       'help';
 2  Init:       'init' project_name repository_url;
 3  Open:       'open' project_name;
 4  Close:      'close';
 5  Create:     'create' model_name class_name;
 6  Select:     'select' (selectable_asset | '*') 'from' (model_name | '*') ('where' condition)?;
 7  Insert:     'insert' insert_assignment 'to' (model_name | '*') ('where' condition)?;
 8  Update:     'update' (model_name | '*') 'set' update_assignment ('where' condition)?;
 9  Delete:     'delete' (deleteable_asset | '*') 'from' (model_name | '*') ('where' condition)?;
10  Calculate:  'calculate' (model_name | '*') column_name metric ('where' condition)?;
11  Validate:   'validate' (model_name | '*') column_name metric ('where' condition)?;
12  Fit:        'fit' (model_name | '*') column_name ('where' condition)?;
13  Predict:    'predict' model_name column_value+;
14  Continue:   'continue' datetime_history 'in' model_name;
```

Figure 4.2: Definitions for each main command initials of the grammar.

Line 1:
Contains sub-grammar for the help-command. Outputs a help message.

Line 2:
Contains sub-grammar for the init-command. Initializes a project by specifying a GitHub repository.

Line 3:
Contains sub-grammar for the open-command. Opens an existing project.

Line 4:
Contains sub-grammar for the close-command. Closes the current project.

Line 5:
Contains sub-grammar for the create-command. Creates a machine learning model by specifying an existing Scikit-learn class.

Line 6:
Contains sub-grammar for the select-command. Selects machine learning assets of machine learning models. Filter models by specifying a condition.

Selecting all (*) retrieves the metadata containing all versions of the specified model.

Line 7:
Contains sub-grammar for the insert-command. Inserts machine learning assets to machine learning models. Filter models by specifying a condition.

Line 8:
Contains sub-grammar for the update-command. Updates machine learning assets of machine learning models. Filter models by specifying a condition.

Line 9:
Contains sub-grammar for the delete-command. Deletes machine learning assets of machine learning models. Filter models by specifying a condition.

Line 10:
Contains sub-grammar for the calculate-command. Calculates a specified performance metric value of machine learning models. Filter models by specifying a condition.

Line 11:
Contains sub-grammar for the validate-command. Validates a specified performance metric value of machine learning models. Filter models by specifying a condition.

This validation feature uses a method called "sklearn.model_selection.cross_validate", which is a method provided by Scikit-learn [29]. This method returns an array of scores such as test_score, train_score, fit_time and score_time.

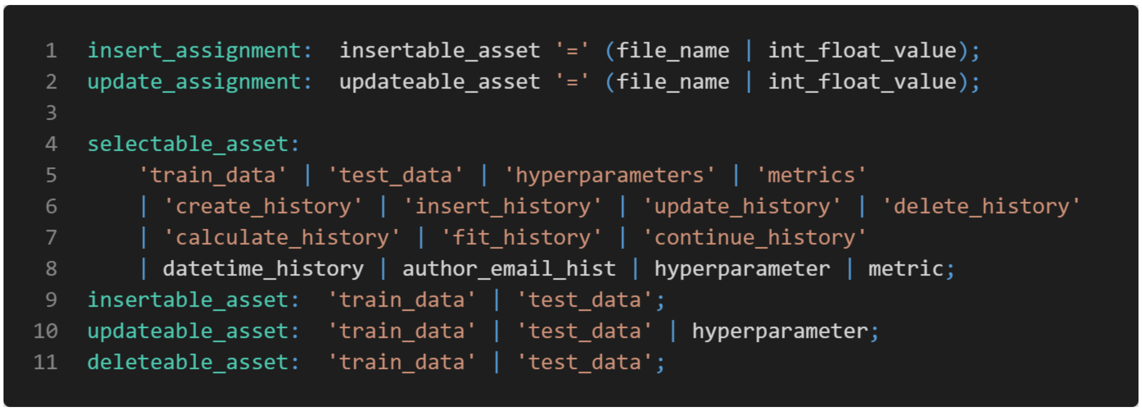
Line 12:
Contains sub-grammar for the fit-command. Trains machine learning models. Filter models by specifying a condition.

Line 13:
Contains sub-grammar for the predict-command. The machine learning model makes a prediction.

DecisionTreeClassifier is the only class that was tested to make predictions. Other classes have not yet been tested and it is unknown to the author whether they can make predictions or not.

Line 14:
Contains sub-grammar for the continue-command. Continues a machine learning model in a specified version.

Assets are defined and its grammar is shown below in Figure 4.3.



```
 1  insert_assignment:  insertable_asset '=' (file_name | int_float_value);
 2  update_assignment:  updateable_asset '=' (file_name | int_float_value);
 3
 4  selectable_asset:
 5      'train_data' | 'test_data' | 'hyperparameters' | 'metrics'
 6      | 'create_history' | 'insert_history' | 'update_history' | 'delete_history'
 7      | 'calculate_history' | 'fit_history' | 'continue_history'
 8      | datetime_history | author_email_hist | hyperparameter | metric;
 9  insertable_asset:  'train_data' | 'test_data';
10  updateable_asset:  'train_data' | 'test_data' | hyperparameter;
11  deleteable_asset:  'train_data' | 'test_data';
```

Figure 4.3: Asset definitions in the grammar for the commands select, insert, update and delete.

Line 1:
Contains sub-grammar for assignment of data to assets that are insert-able.

Line 2:
Contains sub-grammar for assignment of data to assets that are update-able.

Line 4-8:
Contains sub-grammar for the assets that are select-able.

Selecting a date time by specifying the exact date time retrieves the version containing its metadata with its version-specific values.

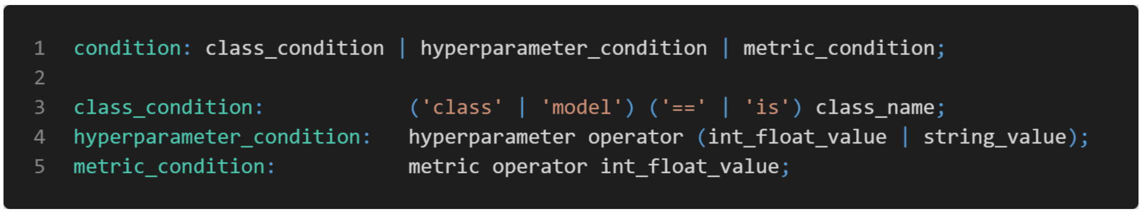
Selecting the GitHub email of an author of a project of the query language retrieves all versions of specified models that have been modified by the specified author.

Line 9:
Contains sub-grammar for the assets that are insert-able.

Line 10:
Contains sub-grammar for the assets that are update-able.

Line 11:
Contains sub-grammar for the assets that are delete-able.

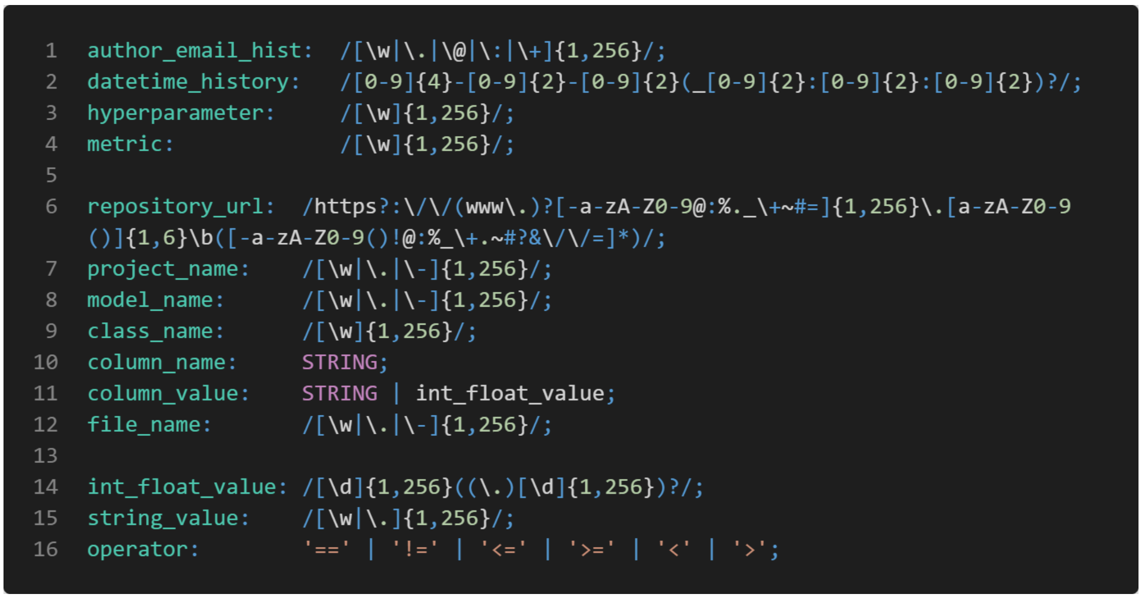
Conditions are defined and its grammar is shown below in Figure 4.4. A condition is either a class, hyperparameter or metric condition type. Specifying a class condition filters away machine learning models that are not the specified machine learning model (Scikit-learn) class. Specifying a hyperparameter condition filters away machine learning models that do not satisfy the specified hyperparameter condition. A hyperparameter condition should contain a machine learning class-specific hyperparameter, comparison operator and a value of either integer, float or string. Specifying a metric condition filters away machine learning models that do not satisfy the specified class-specific performance metric value condition. All conditions are validated in run-time of program execution.



```
condition: class_condition | hyperparameter_condition | metric_condition;

class_condition:            ('class' | 'model') ('==' | 'is') class_name;
hyperparameter_condition:   hyperparameter operator (int_float_value | string_value);
metric_condition:           metric operator int_float_value;
```

Figure 4.4: Grammar definitions for conditions.

A few specific assets are defined in regular expressions and the grammar for each of those is shown in Figure 4.5 below. A quick reference for regular expression syntax is available at a website (https://regex101.com/) [30].



```
author_email_hist:  /[\w|\.|\@|\:|\+]{1,256}/;
datetime_history:   /[0-9]{4}-[0-9]{2}-[0-9]{2}(_[0-9]{2}:[0-9]{2}:[0-9]{2})?/;
hyperparameter:     /[\w]{1,256}/;
metric:             /[\w]{1,256}/;

repository_url:  /https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-9
()]{1,6}\b([-a-zA-Z0-9()!@:%_\+.~#?&\/\/=]*)/;
project_name:    /[\w|\.|\-]{1,256}/;
model_name:      /[\w|\.|\-]{1,256}/;
class_name:      /[\w]{1,256}/;
column_name:     STRING;
column_value:    STRING | int_float_value;
file_name:       /[\w|\.|\-]{1,256}/;

int_float_value: /[\d]{1,256}((\.)[\d]{1,256})?/;
string_value:    /[\w|\.]{1,256}/;
operator:        '==' | '!=' | '<=' | '>=' | '<' | '>';
```

Figure 4.5: Regular expression definitions for a few assets.

The definition of a regular expression variable called int_float_value accepts a value to be either an integer value or a float value.

The STRING-keyword is a predefined keyword in textX that accepts a string value with double quotation marks (") or single quotation marks (') at the beginning and at the end of the string.

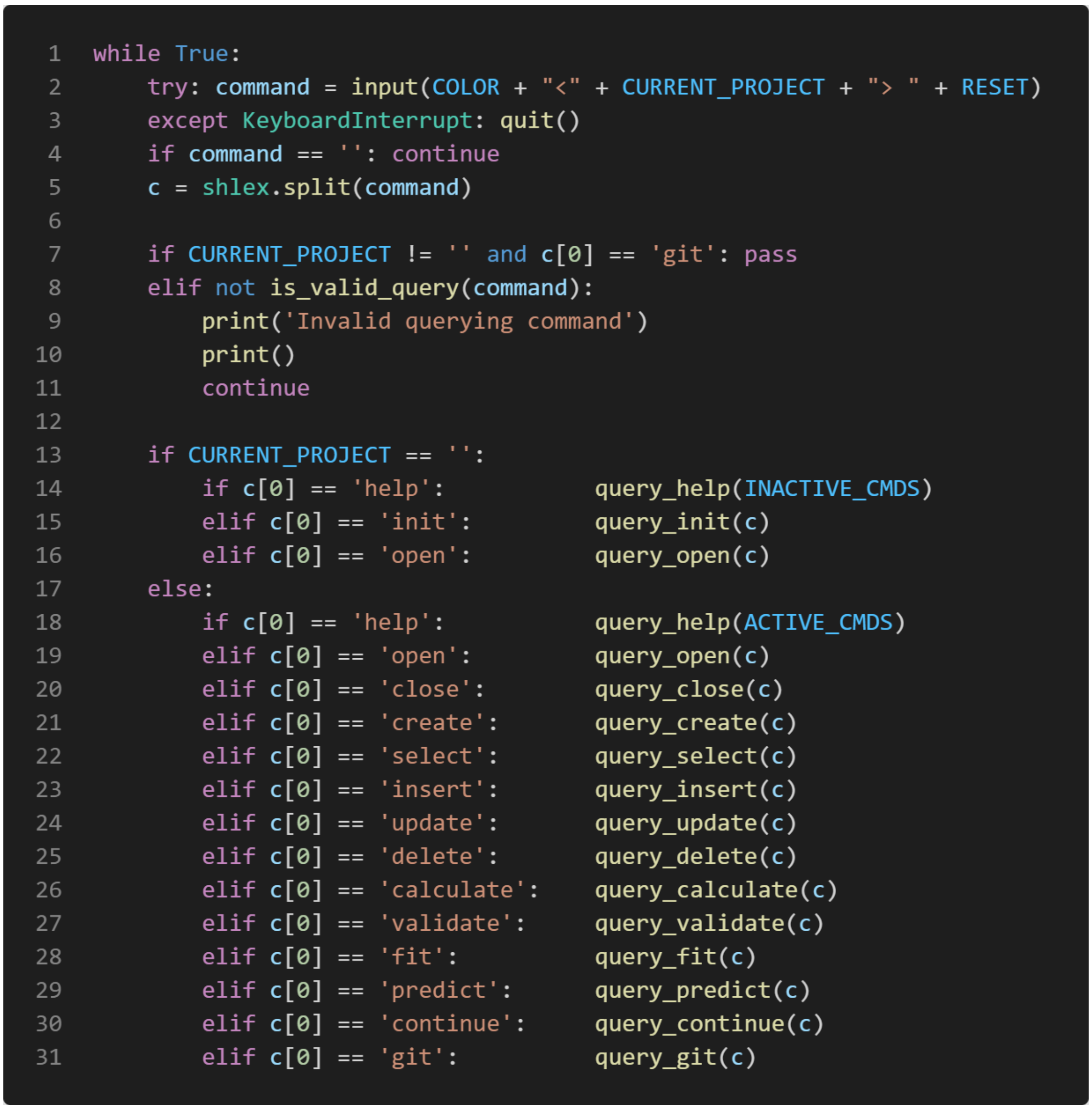
The definition of a regular expression variable called string_value accepts a value to be a string value without quotation marks.

Note that a variable called operator that is included in Figure 4.5 is not a regular expression definition, and is just a regular keyword definition.

4.4.2 Code

A program for the query language is implemented and the code is written in Python. The purpose of the program is to validate querying commands by checking the syntax of the querying command and as well as handling the querying command if the querying command is valid. A detailed inspection of the code is provided in a GitHub repository and is available upon request as the repository is currently private for now.

The main function of the program contains a while-loop. The while-loop continuously reads input of querying commands. The validation of the input is checked. If the input is invalid, the program continues by waiting for a new input. If the input is valid, the program handles the querying command correspondingly. Except if the querying command is detected as a Git-command, it does not require validation check with the defined grammar as it will instead be validated with the grammar of Git.



```
 1  while True:
 2      try: command = input(COLOR + "<" + CURRENT_PROJECT + "> " + RESET)
 3      except KeyboardInterrupt: quit()
 4      if command == '': continue
 5      c = shlex.split(command)
 6  
 7      if CURRENT_PROJECT != '' and c[0] == 'git': pass
 8      elif not is_valid_query(command):
 9          print('Invalid querying command')
10          print()
11          continue
12  
13      if CURRENT_PROJECT == '':
14          if c[0] == 'help':            query_help(INACTIVE_CMDS)
15          elif c[0] == 'init':          query_init(c)
16          elif c[0] == 'open':          query_open(c)
17      else:
18          if c[0] == 'help':            query_help(ACTIVE_CMDS)
19          elif c[0] == 'open':          query_open(c)
20          elif c[0] == 'close':         query_close(c)
21          elif c[0] == 'create':        query_create(c)
22          elif c[0] == 'select':        query_select(c)
23          elif c[0] == 'insert':        query_insert(c)
24          elif c[0] == 'update':        query_update(c)
25          elif c[0] == 'delete':        query_delete(c)
26          elif c[0] == 'calculate':     query_calculate(c)
27          elif c[0] == 'validate':      query_validate(c)
28          elif c[0] == 'fit':           query_fit(c)
29          elif c[0] == 'predict':       query_predict(c)
30          elif c[0] == 'continue':      query_continue(c)
31          elif c[0] == 'git':           query_git(c)
```

Figure 4.6: Partial implementation code of main loop function of the program.

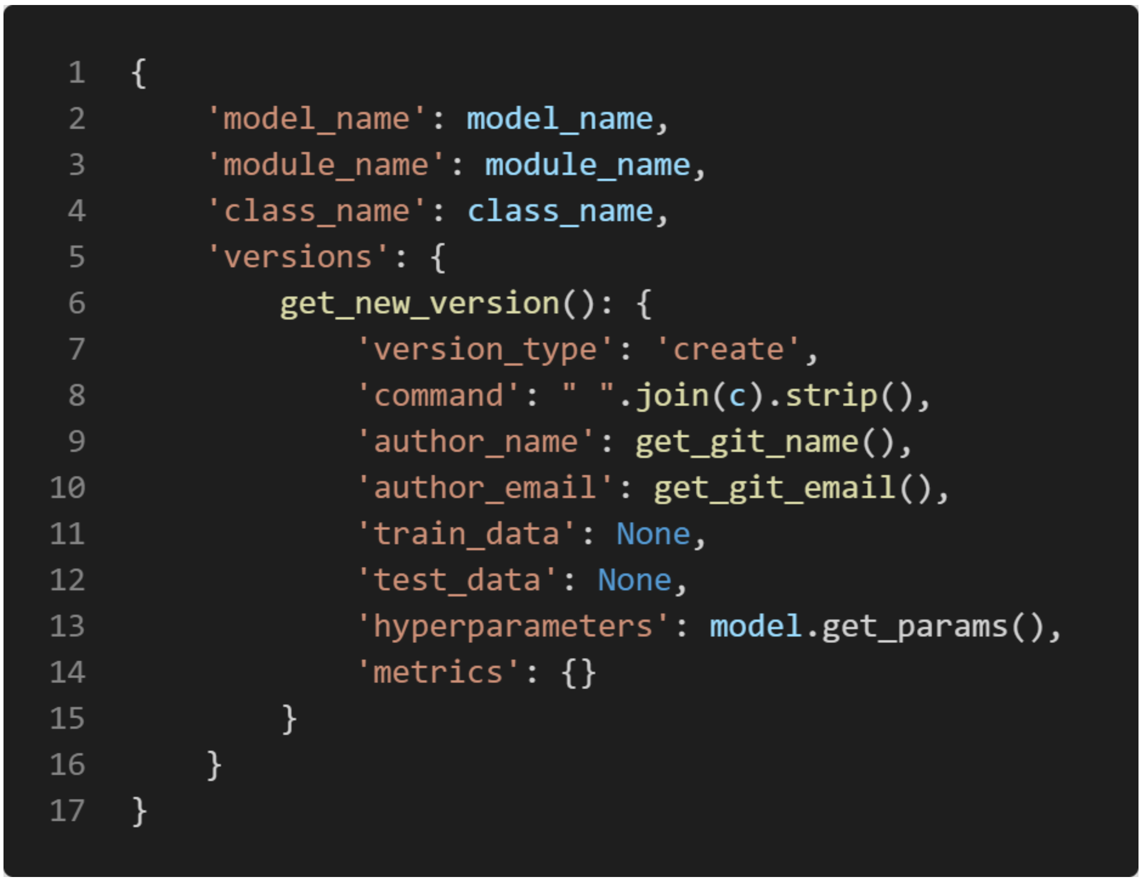
As seen in the code (at line 13-31), several commands are available depending whether a project is currently active or not. If no project is currently active then the following commands: "help", "init" and "open" are available. If a project is currently active then the commands such as "help", "open", "close", "create", "select", "insert" and more are available as shown in Figure 4.6 above. A project can be activated by using the open-command. A project can be closed using the close-command. A project can be initialized using the init-command. Using the open-command when a project is already active switches the current active project to the specified project in the command. The rest of the querying commands are previously briefly explained for Figure 4.2.

4.4.3 Structure

There are two core files when managing machine learning assets: 1. The binary file, 2. The metadata file. If either of the files is lost, the machine learning model would not be able to be managed properly. Integration with Git and utilization of Git solves the problem of loss of either core files. Utilizing Git saves files as a backup in case any of the files are lost.

Binary File
The binary file is the heart of the model. It is created once during the creation of a machine learning model. It is updated each time a model is trained.

Metadata File
The metadata file is the brain of the model. It is created once during the creation of a machine learning model. It is updated each time a trackable querying command is executed. A trackable command is every command that is included as a select-able history asset as shown in Figure 4.3.



```
{
    'model_name': model_name,
    'module_name': module_name,
    'class_name': class_name,
    'versions': {
        get_new_version(): {
            'version_type': 'create',
            'command': " ".join(c).strip(),
            'author_name': get_git_name(),
            'author_email': get_git_email(),
            'train_data': None,
            'test_data': None,
            'hyperparameters': model.get_params(),
            'metrics': {}
        }
    }
}
```

Figure 4.7: Metadata structure containing different assets.

The metadata consists of the model name, the Scikit-learn module name, the Scikit-learn class name of the module and a list of versions. A version is defined by the date time during execution of a trackable command. Each version consists of the versioning type which is the executed command initial, the whole command, the author's Git full name, the author's Git email, the training data, the test data, class-specific hyperparameters with values for each and class-specific performance metric with values for each. Each version is saved for the purpose to achieve model versioning (FR2) and model reproducibility (FR3). Each asset contained in the metadata file is needed to achieve data provenance (FR4).

4.4.4 Versioning

There are two types of versioning utilized in the query language. The first type of versioning is related to the metadata. The metadata updates the history of executions of trackable querying commands. Each version is trackable. This type of versioning is done locally in offline mode i.e. does not require connection to any kind of server.

The second type of versioning is related to Git. Git can save model files locally in offline mode and as well can be pushed to the online repository. Git can be used as a backup in order to save and retrieve any core files that might be lost by any kind of reason, either manually or potential run-time errors (which might not have been fixed yet).

4.5 First Evaluation

The first phase of interviews consisted of four interviewees. A simple table of interviewees is provided below.

	SM	**SDL**
Participant 1	SE and Management	Bachelor
Participant 2	SE and Management	Bachelor
Participant 3	Data Science and AI	Master
Participant 4	Data Science and AI	Doctoral

Table 4.13: Displays participants' experience related to this thesis topic. Abbreviations: SM: Study Major, SDL: Study Degree Level, SE: Software Engineering, AI: Artificial Intelligence.

The questions asked during the interviews are provided in Appendix A.1. The requirements of the query language are evaluated by the interviewees. Insights from interviewees were gathered and summarized.

The insights were summarized by first identifying each opinion as a code. The codes are then labeled into labels. A full list of code-label pairs can be found in Appendix A.3. Themes are then generated and formed by the labels. The themes are then discussed and summarized in regards to the RQs.

A few examples of interview questions and codes:

Q1.4: How important is the requirement interpretability?

> **Code:** "Uh, it depends on what applications it's about. For example, I predict which medications a patient should take, for example, to determine whether that patient has cancer or not. When you make such decisions, you want to understand how the machine learning model processed, when it came to that decision. Because patients want to know... Like, the model says I have cancer, but I also want to know why it says so. For such applications, it is important. But then there are other things, such as predicting an email is spam or not, which is not important." - Participant 4
> **Note:** Transcript translated from Swedish to English

Label: Depends on the type of applications when it comes to interpretability

Q1.8: What do you think about the usage of the query language, based on what has been shown? What do you like, what needs improvement?

> **Code:** "Ehm, I think it's good that it sort of strives to look like regular sequel commands a little bit. I think that's a good thing, cuz you always feels right at home, you know what it's about, that's good. Although that comes with the backside if your command were to behave differently in some way from regular sequel command. If it were to perform a function that it is completely, like, it does not look like or feel like regular sequel functions at all, then that might be a bit of a surprise. So there might be small learning bumps. So that might be the backside of it. So there's the feeling, I know this, I've seen this. And there is also, wow this didn't behave the way I expected it to. So it might be a two-edged sword, so to speak. But I like the idea to go for a sequel kind of style." - Participant 3

Labels: Follows standard sequel commands, Possibly can behave differently compared to regular sequel commands, Possible learning bumps

> **Code:** "I think I understand roughly what it's about, but not exactly in detail. It is an SQL-like language. You have made a program that takes care of all the steps that you normally need to do manually, such as starting a GitHub repository, defining your model, training the model. Improvement then would be to implement support for more libraries, so not only for Scikit-learn, but your language could also support PyTorch, Keras or TensorFlow. When it comes to data analysis and stuff like that, because now you just load the data and train. You have to prepare the data and normalize it as well." - Participant 4
> **Note:** Transcript translated from Swedish to English

Labels: Easy to understand the idea, Possible improvement by adding support for more libraries, Possible improvement by adding support for data analysis

Labels to Themes:

Labels	**Themes**
Agreement on model versioning	Agreement on Elicited Requirements
Agreement on model reproducibility	Agreement on Elicited Requirements
Agreement on data provenance	Agreement on Elicited Requirements
Depends on the type of users when it comes to interpretability	Requirement Prioritization
Agreement on interpretability	Agreement on Elicited Requirements
Depends on the type of applications when it comes to interpretability	Requirement Prioritization
Agreement on performance efficiency	Agreement on Elicited Requirements
Depends on what kind of aspect when it comes to performance efficiency	Requirement Prioritization
Agreement on interoperability	Agreement on Elicited Requirements
Approval of no missing requirements	Agreement on No Missing Requirements
Lack of experience	Validity
Easy to use	Ease of Use
Follows standard sequel commands	Ease of Use
How to handle big data	Elicitation of New Requirements
Possibly can behave differently compared to regular sequel commands	Ease of Use
Possible learning bumps	Ease of Use
Easy to understand the idea	Ease of Use
Possible improvement by adding support for more libraries	Elicitation of New Requirements
Possible improvement by adding support for data analysis	Elicitation of New Requirements

Table 4.14: Displays conversion from code labels from first phase of interviews to themes.

List of Generated Themes (For First Phase of Interviews):

- Agreement on Elicited Requirements
- Requirement Prioritization
- Agreement on No Missing Requirements
- Elicitation of New Requirements
- Validity
- Ease of Use

Each following paragraph below is about each theme such that it relates to the RQs. Each theme description is according to the interviewees' opinions.

Agreement on Elicited Requirements (RQ1)
The majority agreed on the requirements model versioning, model reproducibility, data provenance, interpretability, performance efficiency and interoperability. It is useful to backtrace what has been done, it is important to keep track of different versions. Participant 3 thought that some parts of Git would actually be beneficial to enrich the versioning of the query language, which is an agreement to the requirement interoperability.

Requirement Prioritization (RQ1)
Participant 2 thought that interpretability might be unnecessary for some people that does not require an understanding on how exactly in detail the prediction is calculated by the trained machine learning model. Participant 4 thought that the importance of interpretability depends on what kind of applications the model is used for. It is important to know how the model came to the conclusion about what kind of medicines to take to know whether the patients have cancer or not. It is not important to know how the model came to the conclusion whether an email is spam or not. Participant 3 thought that Python is not a very performant programming language that might affect the performance efficiency in the query language which might contradict the elicitation of the requirement. Also, versioning does not require efficient performance, however it is important that it does not slow down the query language overall. Participant 4 provided some more input, that the performance efficiency is dependent on the training time of the model which depends on the library that is being used.

Agreement on No Missing Requirements (RQ1)
The majority agreed on that there are no missing requirements. Participant 3 thought that it is a good list of requirements. The listed requirements are broad. Captures most of the important things.

Elicitation of New Requirements (RQ1)
Participant 2 thought that an improvement would be to add a timer to check how long it would take to train models with large datasets, which indicated the need for support for larger datasets in the query language. A newly elicited requirement

called "Scalabilty" is formed (as mentioned previously in Section 4.1.2). Participant 4 thought that an improvement would be adding support for other libraries such as PyTorch, Keras and TensorFlow as the query language is currently utilizing only Scikit-learn classes, which might not be useful for advanced users who requires using other classes. Also, participant 4 thought that the query language would expand to more advanced users if the query language is updated new features that are able to analyse data of the datasets used by the models as it is currently only able to load the data and train the model. Analysis of datasets would include visualization of data, preparation of data and normalisation of data. Another newly elicited requirement called "Data Analysability" is also formed (as mentioned previously in Section 4.1.1). Different ways to achieve the newly formed requirements are discussed in Section 7.2.

Validity (RQ1)
Participant 1 mentioned its own lack of experience which might affect the validity of the specific interview.

Ease of Use (RQ2)
Participant 1, 2 and 3 thought that the overall usage of the query language is good. Participant 3 thought that it is good that the query language strives to look like regular sequel commands, and that it feels right at home, and can see the usefulness of the query language. Although that comes with the backside if the query command is to behave differently compared to regular sequel commands. There might be small learning bumps. It might behave differently than expected. It is like a two-edged sword.

4.6 Second Evaluation

The second phase of interviews consisted of four interviewees. The questions asked during the interviews are provided in Appendix A.2. The features of the query language are evaluated by the interviewees. Interviewees' experience are asked (Q2.1 - Q2.4) and are summarized in a table below.

	SM	**SDL**	**EML**	**EQL**	**PFP**
Participant 1	SE and Management	Bachelor	3	3	False
Participant 2	SE and Management	Bachelor	1	2	False
Participant 3	Data Science and AI	Master	1.5	3.5	True
Participant 4	Data Science and AI	Doctoral	4	3	True

Table 4.15: Displays participants' experience related to this thesis topic. Abbreviations: SM: Study Major, SDL: Study Degree Level, EML: Experience Level in Managing Machine Learning Assets (1 to 5) (1: very inexperienced, 5: very experienced), EQL: Experience in Query Languages (1 to 5) (1: very inexperienced, 5: very experienced), PFP: Participated During First Phase of Interviews, SE: Software Engineering, AI: Artificial Intelligence.

The questions related to the features asked during the interviews are provided in Appendix A.2. The features of the query language are evaluated by the interviewees. Insights from interviewees were gathered and summarized.

The summarization of insights from the second phase of interview is similar to summarization of insights from the first phase of interviews. The insights were summarized by first identifying each opinion as a code. The codes are then labeled into labels. A full list of code-label pairs can be found in Appendix A.4. Themes are then generated and formed by the labels. The themes are then discussed and summarized in regards to the RQs.

Questions Q2.1 - Q2.4 were simple enough and did not provide any additional insights and does not require any kind of thematic analysis and therefore were skipped in the Appendix, as it already provided necessary output for Table 4.15 above.

A few examples of interview questions and codes:

Q2.6: Based on your own experience in developing ML projects: How would you rate the usefulness of the query language on a scale from 1 (not useful at all) to 5 (very useful)?

> **Code:** "Uhh, I have to think. So, I mean like, versions for like different models you trained with different hyperparameter setups is very important for machine learning. So, like the versioning part of it, like storing the history is definitely useful. I'm not sure... Like so, I think using the query language to actually train models and develop machine learning systems, then you would need some models that using existing methods. So somebody that is like working for a company maybe and then working on a particular application. And then they are only going to use existing methods. But for me as a PhD student who's researching stuff, making my own models and so on. Then it would be hard to use this query language, right. Cuz, I won't be able to use just any machine learning model. So, for me personally probably not so useful for that reason, but in general it looks interesting. So like usefulness in general is, I would give it maybe a 4. But for me is probably like a 1." - Participant 4

Labels: Rating on the usefulness of the query language, Agreement on versioning functionality, Possible improvement by adding support for any arbitrary models, Hard to use the query language

Q2.9: Which of the features need improvement? For each of them: what kind of improvement?

> **Code:** "It's difficult, because, you know, for a short amount of time, you have to get a very deep understanding of all of them and check if any needs improvement. But I'd say right now I don't think any of them do." - Participant 1

Labels: Approval of no features need improvement, More time required to understand

Labels to Themes:

Labels	**Themes**
Rating on own understanding of the query language	Ease of Use
Rating on the usefulness of the query language	Usefulness
Subjective rating on most useful feature	Usefulness
Agreement on versioning functionality	Agreement on Implemented Features
Agreement on reproducibility functionality	Agreement on Implemented Features
Agreement on reusability functionality	Agreement on Implemented Features
Agreement on calculation feature	Agreement on Implemented Features
Agreement on filtering feature	Agreement on Implemented Features
Possible improvement by adding support for any arbitrary models	Elicitation of New Features
Hard to use the query language	Ease of Use
Approval of no feature to be least useful	Agreement on Implemented Features
Lack of experience	Validity
Approval of no features need improvement	Agreement on Implemented Features
More time required to understand	Validity
Disapproval of no features need improvement	Potential Improvement of Implemented Features
Possible improvement by easing the process of comparison between values produced by different models	Potential Improvement of Implemented Features
Possible improvement by reducing the need for a lot of scrolling in command console to find the correct version of a model	Potential Improvement of Implemented Features
Possible improvement by adding support for other libraries	Elicitation of New Features
Approval of no new features can improve the usefulness	Agreement on Implemented Features
Disapproval of no new features can improve the usefulness	Elicitation of New Features
Possible improvement by improving the current features to save values in a file separately	Potential Improvement of Implemented Features
Possible improvement by adding support for data analysis	Elicitation of New Features

Table 4.16: Displays conversion from code labels from second phase of interviews to themes.

List of Generated Themes (For Second Phase of Interviews):

- Agreement on Implemented Features
- Potential Improvement of Implemented Features
- Elicitation of New Features
- Validity
- Ease of Use
- Usefulness

Each following paragraph below is about each theme such that it relates to the RQs. Each theme description is according to the interviewees' opinions.

Agreement on Implemented Features (RQ2)
The majority agreed on the current set of features. Participant 4 thought a few features could be improved and few new potential features could be implemented.

Potential Improvement of Implemented Features (RQ2)
Participant 4 thought that the features that relate to training models and predicting models which are "fit" and "predict" were the least useful as their use are restricted to existing Scikit-learn class models. An improvement would be to add support for any arbitrary models e.g. models that are defined in a Python class by the users as long as it follows the correct paradigm such that it follows the current features of the query language e.g. "fit" and "predict". Another improvement would be to add support for other libraries such as PyTorch and TensorFlow, as they offer great functionalities as well. Participant 1 thought of an improvement could be to improve the current features to be able to save values such as hyperparameter values or performance metric values in a file separately. Participant 2 thought of an improvement that is to ease the process of comparison between values produced by different models. Participant 3 thought of another minor improvement would be to reduce the need for a lot of scrolling in the command console to find the correct version of a model by making the model metadata output more compact.

Elicitation of New Features (RQ2)
Participant 4 thought of an improvement would be to add a new feature that supports the ability to analyse data. Feature commands that plots different statistics regarding the data, manipulates data in different ways and normalises the data would improve the usefulness of the query language. In practice, dealing with datasets with millions of entries usually have some missing or empty values which should be handled as well. A feature that supports data analysability can also help with finding outliers.

Validity (RQ2)
Participant 1 mentioned its requirement of more time to be able to understand more about the query language which might affect the validity of the interview. Participant 2 mentioned its lack of experience in the topic of this thesis which might affect

the validity of the interview as well.

Ease of Use (RQ2)
The mean value of participants' understanding of the query language is 3.875.

Usefulness (RQ1, RQ2)
The mean value of participants' rating on the usefulness of the query language is 2.875.
The feature that was the most useful according to a subjective rating by the participants is the feature "continue" that allows reproducing a model of a different version.

Participant 4 thought that the versioning part of the query language is very useful. Although the versioning part is useful, the query language would not be very useful if companies were required to use existing methods. For researchers, it would be hard to use this query language, because they want to use any kind of machine learning models and not only Scikit-learn class models that are created by the query language. The current state of the query language is not very useful for advanced users, however has a lot of potential if more features were to be implemented. Participant 1 thought that it is good to switch between different versions of a model. All of the features were good, as each is a part of the whole procedure of managing machine learning assets. Participant 2 thought that the feature "calculate" is useful. The filtering feature was also found pretty neat and helpful such that it supports working with multiple models at the same time. All the features that related to saving histories, versioning and reproducing results were the most useful. The majority of the participants did not have any feature to be the least useful in mind as all of the features were useful.

GQM Results:
The metrics of the GQM-approach used in this thesis (see Section 3.6, Interviews Second Phase) are calculated as well.

- **Metric M1**: Developers' rating on their own experience in machine-learning-based systems. Mean rating score of 2.375.
- **Metric M2**: Developers' rating on their own experience in query languages. Mean rating score of 2.875.
- **Metric M3**: Developers' rating on the usefulness of the query language of this thesis project. Mean rating score of 2.875.
- **Metric M4**: Developers' subjective rating on the most useful feature. Feature "continue".
- **Metric M5**: Developers' rating on their own understanding of the developed query language of this thesis project. Mean rating score of 3.875.

5 Artifact Usage Scenarios

The purpose of this chapter is to provide further understanding on how the querying commands works by providing execution examples of the available querying commands. A querying command is valid if it follows the rules of the defined grammar for the query language. Additionally, a querying command is also valid if it is a valid Git-command without following the defined grammar as the implementation design of the query language is integrated with Git.

Initially, the folder structure only contains the program file written in Python "query.py" and the textX grammar file "grammar.tx".

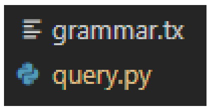
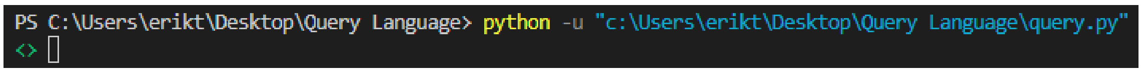


Figure 5.1: Initial folder structure.

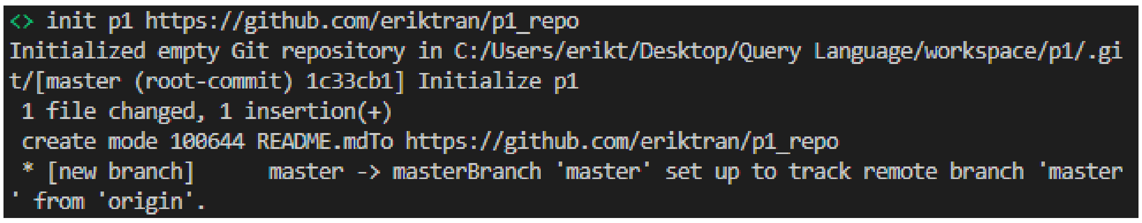
To begin, open a command terminal, start by changing the current directory to the same directory that contains the file "query.py". Next, execute the file "query.py".



```
PS C:\Users\erikt\Desktop\Query Language> python -u "c:\Users\erikt\Desktop\Query Language\query.py"
<> 
```

Figure 5.2: Example execution command of main program in Visual Studio Code.

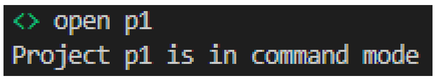
Initialize a project, by specifying a name and a GitHub repository for it.



```
<> init p1 https://github.com/eriktran/p1_repo
Initialized empty Git repository in C:/Users/erikt/Desktop/Query Language/workspace/p1/.gi
t/[master (root-commit) 1c33cb1] Initialize p1
 1 file changed, 1 insertion(+)
 create mode 100644 README.mdTo https://github.com/eriktran/p1_repo
 * [new branch]      master -> masterBranch 'master' set up to track remote branch 'master
' from 'origin'.
```

Figure 5.3: Example execution of init-command with its output. Initializes a new project called p1.

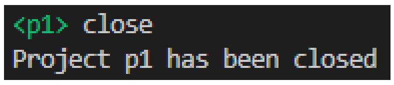
Open the project.



```
<> open p1
Project p1 is in command mode
```

Figure 5.4: Example execution of open-command with its output. Sets project p1 to be in command-mode.

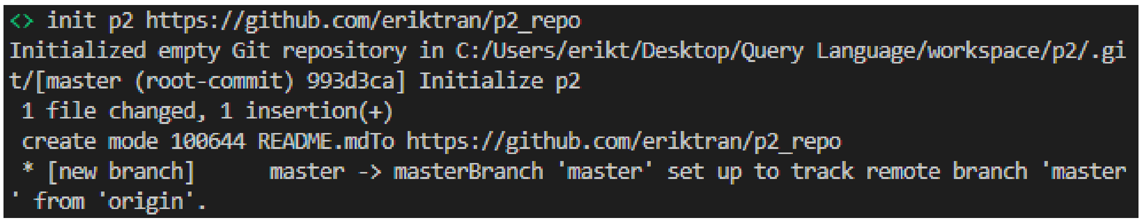
To effortlessly switch between projects, use the same open-command. To show this, another project needs to be created first. So, close the current project first.



```
<p1> close
Project p1 has been closed
```

Figure 5.5: Example execution of close-command with its output. Unsets project p1 to be in command-mode.

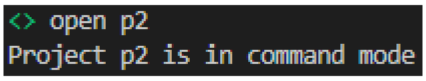
Initialize another project.



```
<> init p2 https://github.com/eriktran/p2_repo
Initialized empty Git repository in C:/Users/erikt/Desktop/Query Language/workspace/p2/.gi
t/[master (root-commit) 993d3ca] Initialize p2
 1 file changed, 1 insertion(+)
 create mode 100644 README.mdTo https://github.com/eriktran/p2_repo
 * [new branch]      master -> masterBranch 'master' set up to track remote branch 'master
' from 'origin'.
```

Figure 5.6: Example execution of init-command with its output. Initializes a new project called p2.

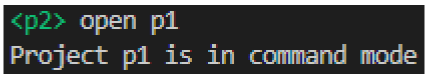
Open the project.



```
<> open p2
Project p2 is in command mode
```

Figure 5.7: Example execution of open-command with its output. Sets project p2 to be in command-mode.

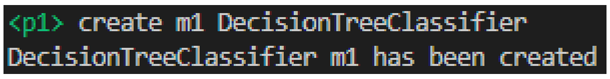
Switch to another project using the same open-command.



```
<p2> open p1
Project p1 is in command mode
```

Figure 5.8: Example execution of open-command with its output. Unsets project p2 to be in command-mode and sets project p1 to be in command-mode.

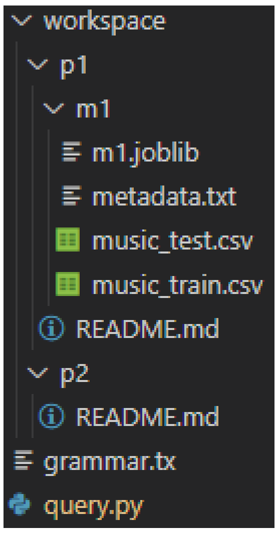
Let's create a new machine learning model as a DecisionTreeClassifier (a Scikit-learn class).



```
<p1> create m1 DecisionTreeClassifier
DecisionTreeClassifier m1 has been created
```

Figure 5.9: Example execution of create-command with its output. Creates a machine learning model of class DecisionTreeClassifier called m1.

To train the model, add external training dataset and testing datasets into the model folder manually.

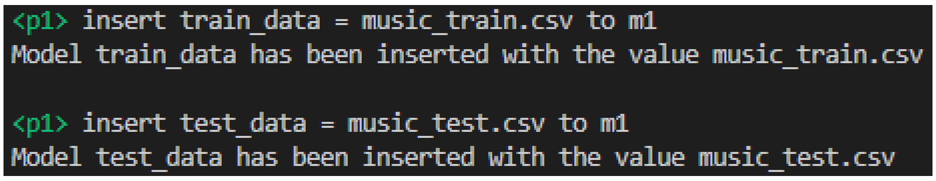


```
workspace
  p1
    m1
      m1.joblib
      metadata.txt
      music_test.csv
      music_train.csv
    README.md
  p2
    README.md
grammar.tx
query.py
```

Figure 5.10: Folder structure after manually moving datasets into model folder.

Note that creating a machine learning model generated a binary file called "m1.joblib" and a metadata file called "metadata.txt" for the created model.

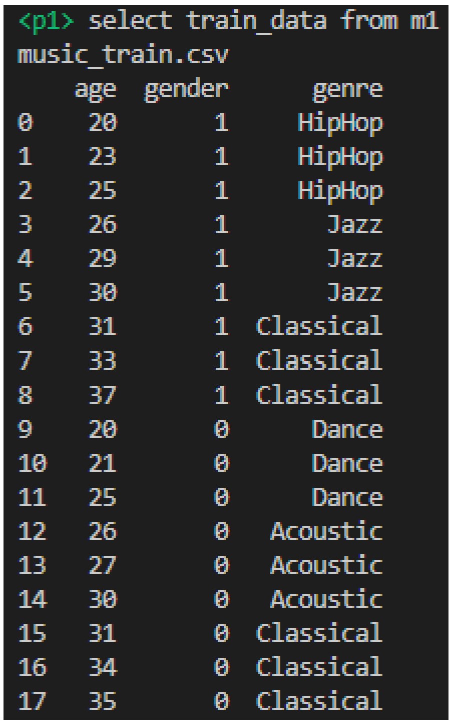
Choose the external training dataset as the training data asset of the model with the insert-command. Do the same for the testing dataset.



```
<p1> insert train_data = music_train.csv to m1
Model train_data has been inserted with the value music_train.csv

<p1> insert test_data = music_test.csv to m1
Model test_data has been inserted with the value music_test.csv
```

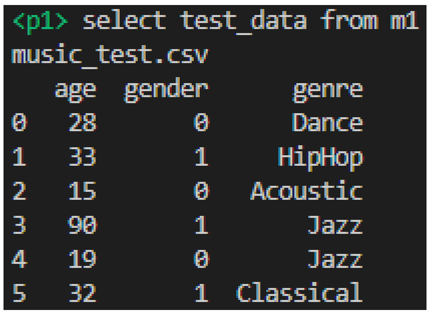
Figure 5.11: Example executions of insert-command with its outputs. Inserts training and testing datasets as the assets of model m1.

Let's display how the datasets look like by using the select-command.



```
<p1> select train_data from m1
music_train.csv
    age  gender      genre
0    20       1     HipHop
1    23       1     HipHop
2    25       1     HipHop
3    26       1       Jazz
4    29       1       Jazz
5    30       1       Jazz
6    31       1  Classical
7    33       1  Classical
8    37       1  Classical
9    20       0      Dance
10   21       0      Dance
11   25       0      Dance
12   26       0   Acoustic
13   27       0   Acoustic
14   30       0   Acoustic
15   31       0  Classical
16   34       0  Classical
17   35       0  Classical
```

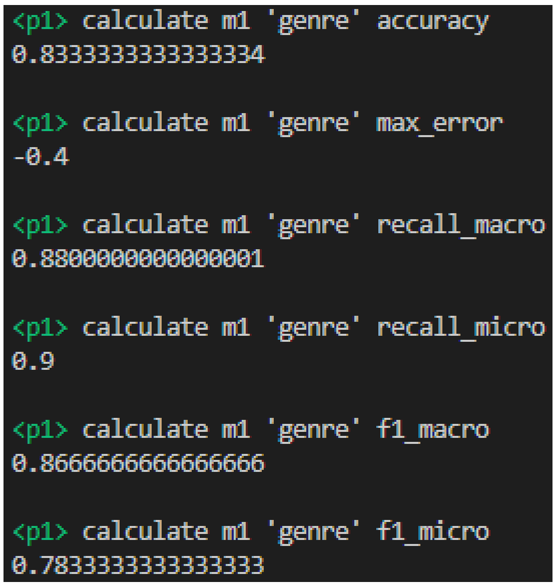
Figure 5.12: Example execution of select-command with its output. Showcasing content of a training dataset.



```
<p1> select test_data from m1
music_test.csv
   age  gender      genre
0   28       0      Dance
1   33       1     HipHop
2   15       0   Acoustic
3   90       1       Jazz
4   19       0       Jazz
5   32       1  Classical
```

Figure 5.13: Example execution of select-command with its output. Showcasing content of a testing dataset.

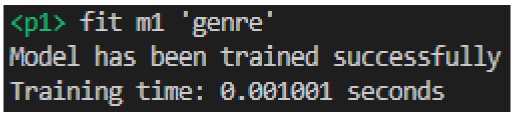
Let's calculate several performance metric values for an existing column of the datasets. A performance metric is class-specific, meaning a performance metric value is only calculate-able if the model of the specified class supports it. Specifying a nonexistent performance metric will print out a list of available performance metrics.



```
<p1> calculate m1 'genre' accuracy
0.8333333333333334

<p1> calculate m1 'genre' max_error
-0.4

<p1> calculate m1 'genre' recall_macro
0.8800000000000001

<p1> calculate m1 'genre' recall_micro
0.9

<p1> calculate m1 'genre' f1_macro
0.8666666666666666

<p1> calculate m1 'genre' f1_micro
0.7833333333333333
```

Figure 5.14: Example executions of calculate-command with its outputs. Calculates several performance metric values.

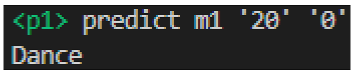
Let's train the model to learn a specified column of the datasets.



```
<p1> fit m1 'genre'
Model has been trained successfully
Training time: 0.001001 seconds
```

Figure 5.15: Example execution of fit-command with its output. Trains model m1 to learn a column of its datasets.

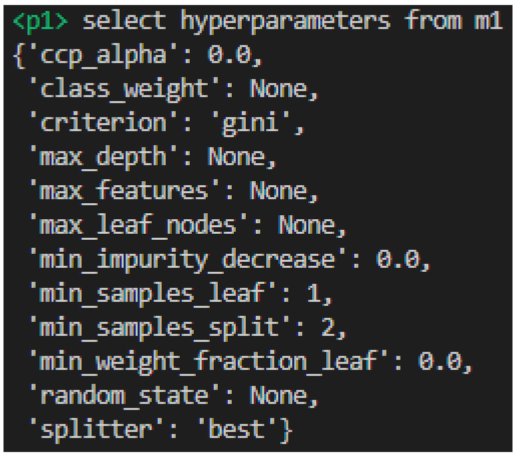
Let's see what the model prediction is by specifying values for all columns except the specified column used for training.



```
<p1> predict m1 '20' '0'
Dance
```

Figure 5.16: Example execution of predict-command with its output. A male of age 20 enjoys the dance music genre, based on the training dataset and testing dataset.

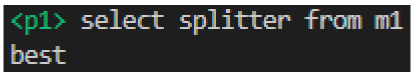
Let's check the available and current hyperparameters of the model.



```
<p1> select hyperparameters from m1
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Figure 5.17: Example execution of select-command. Displays hyperparameters of a model.

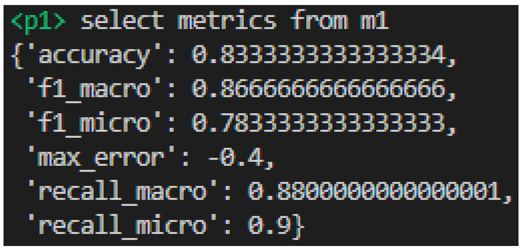
Retrieving the value of a specified hyperparameter is also possible.



```
<p1> select splitter from m1
best
```

Figure 5.18: Example execution of select-command. Outputs a specific hyperparameter value of a model.

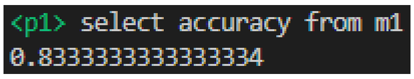
Let's check the available and current performance metrics of the model.



```
<p1> select metrics from m1
{'accuracy': 0.8333333333333334,
 'f1_macro': 0.8666666666666666,
 'f1_micro': 0.7833333333333333,
 'max_error': -0.4,
 'recall_macro': 0.8800000000000001,
 'recall_micro': 0.9}
```

Figure 5.19: Example execution of select-command. Displays performance metrics of a model.

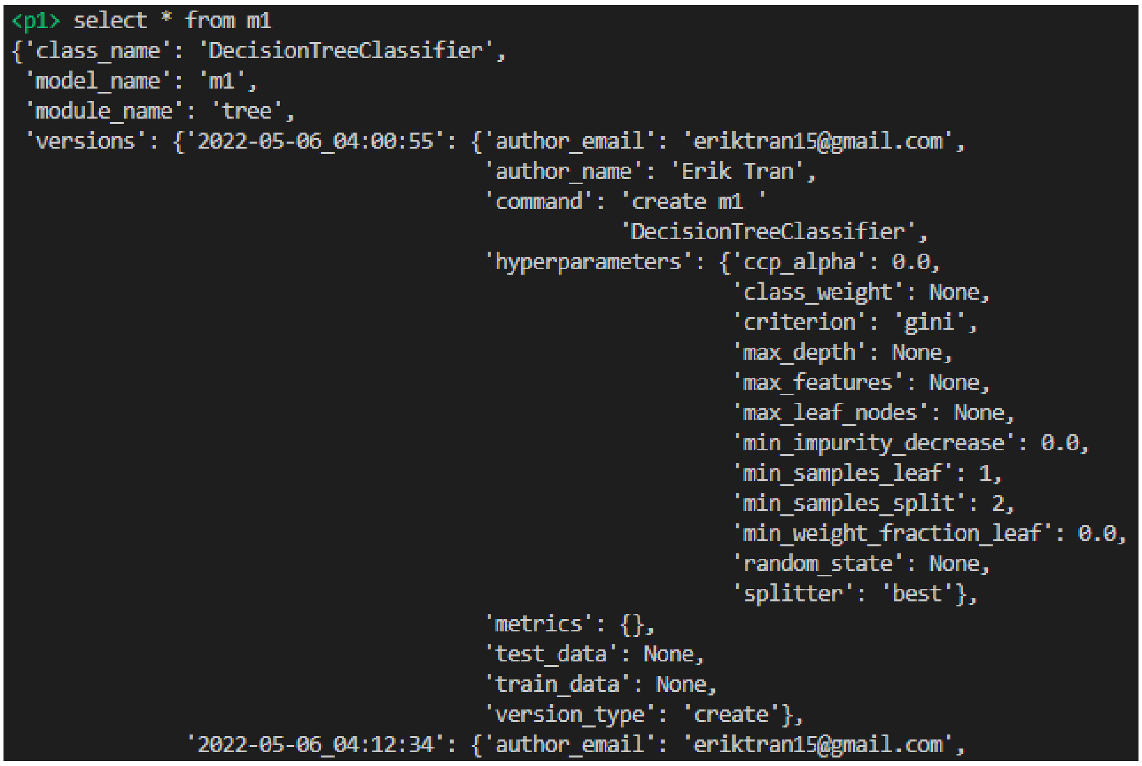
Retrieving the value of a specified performance metric is also possible.



```
<p1> select accuracy from m1
0.8333333333333334
```

Figure 5.20: Example execution of select-command. Outputs a specific performance metric value of a model.

Let's check the full content of the metadata and all the tracked versions of the model that the metadata contains.

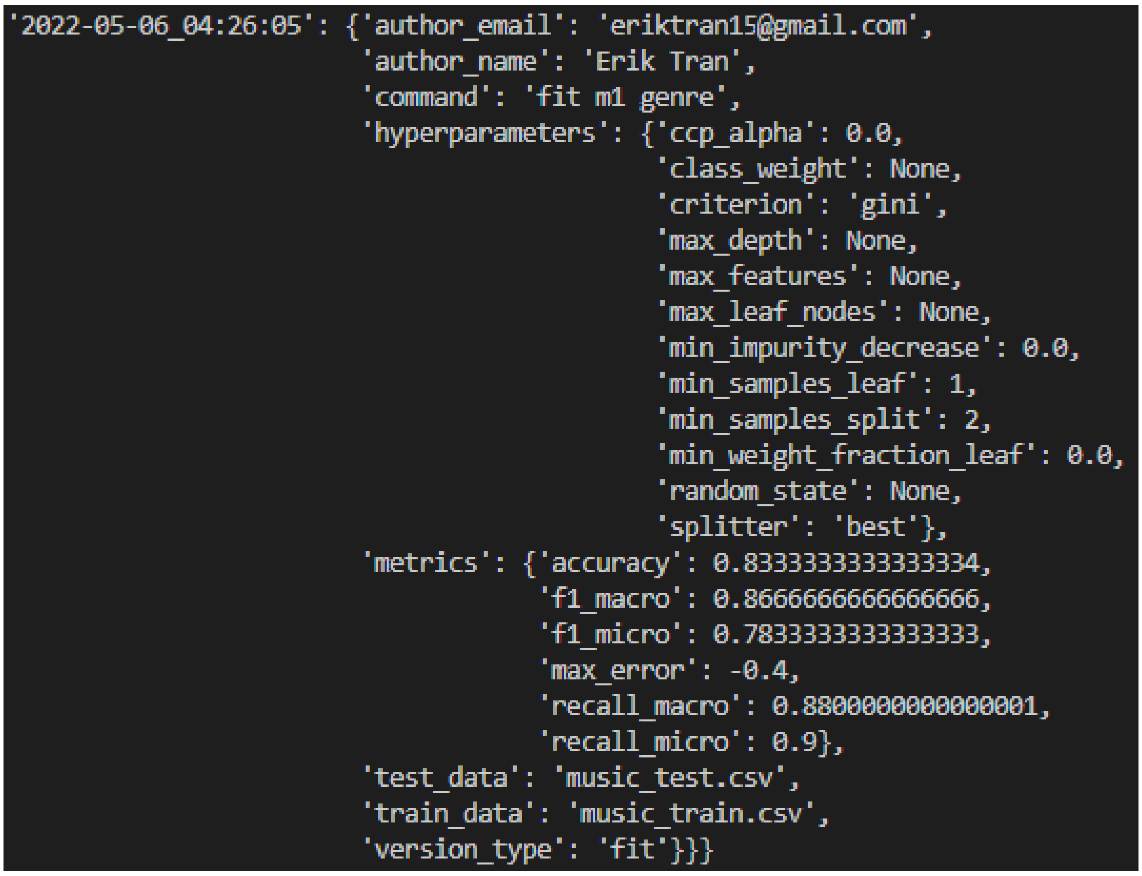


```
<p1> select * from m1
{'class_name': 'DecisionTreeClassifier',
 'model_name': 'm1',
 'module_name': 'tree',
 'versions': {'2022-05-06_04:00:55': {'author_email': 'eriktran15@gmail.com',
                                      'author_name': 'Erik Tran',
                                      'command': 'create m1 '
                                                 'DecisionTreeClassifier',
                                      'hyperparameters': {'ccp_alpha': 0.0,
                                                          'class_weight': None,
                                                          'criterion': 'gini',
                                                          'max_depth': None,
                                                          'max_features': None,
                                                          'max_leaf_nodes': None,
                                                          'min_impurity_decrease': 0.0,
                                                          'min_samples_leaf': 1,
                                                          'min_samples_split': 2,
                                                          'min_weight_fraction_leaf': 0.0,
                                                          'random_state': None,
                                                          'splitter': 'best'},
                                      'metrics': {},
                                      'test_data': None,
                                      'train_data': None,
                                      'version_type': 'create'},
              '2022-05-06_04:12:34': {'author_email': 'eriktran15@gmail.com',
```

Figure 5.21: Example execution of select-command with its output. Partially displays a few first lines of the content of the metadata of a model.

As seen in the figure above, the top-level of the metadata contains class name, model name, module name and versions. Each version contains a date time. The first version is tracked using the create-command.

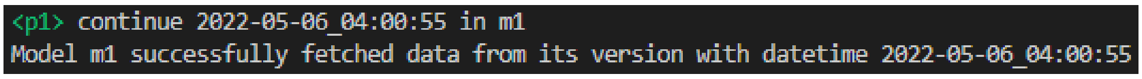
The latest version is printed at the end of the output.



```
'2022-05-06_04:26:05': {'author_email': 'eriktran15@gmail.com',
                        'author_name': 'Erik Tran',
                        'command': 'fit m1 genre',
                        'hyperparameters': {'ccp_alpha': 0.0,
                                            'class_weight': None,
                                            'criterion': 'gini',
                                            'max_depth': None,
                                            'max_features': None,
                                            'max_leaf_nodes': None,
                                            'min_impurity_decrease': 0.0,
                                            'min_samples_leaf': 1,
                                            'min_samples_split': 2,
                                            'min_weight_fraction_leaf': 0.0,
                                            'random_state': None,
                                            'splitter': 'best'},
                        'metrics': {'accuracy': 0.8333333333333334,
                                    'f1_macro': 0.8666666666666666,
                                    'f1_micro': 0.7833333333333333,
                                    'max_error': -0.4,
                                    'recall_macro': 0.8800000000000001,
                                    'recall_micro': 0.9},
                        'test_data': 'music_test.csv',
                        'train_data': 'music_train.csv',
                        'version_type': 'fit'}}}
```

Figure 5.22: Example execution of select-command with its output. Displays the latest version in the metadata of a model.

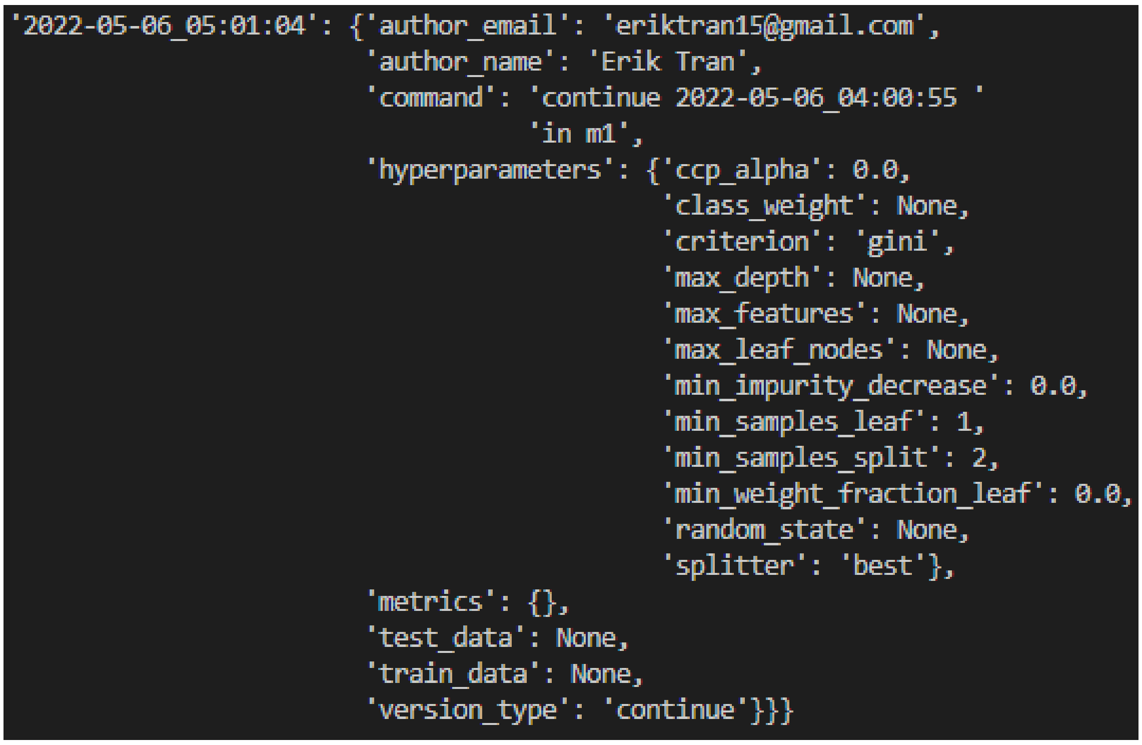
To retrieve model assets from an older version i.e. reproduce a model of an older version, use the continue-command by specifying a date time that is tracked in the metadata file of the model.



```
<p1> continue 2022-05-06_04:00:55 in m1
Model m1 successfully fetched data from its version with datetime 2022-05-06_04:00:55
```

Figure 5.23: Example execution of continue-command with its output. Reproduces a model of an older version.

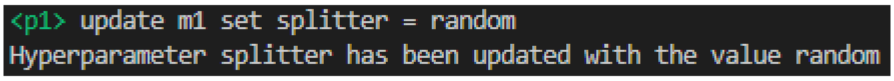
See the latest version at the end of the output message by selecting all assets of the model again, and check that the latest version has been updated with an older version's data.



```
'2022-05-06_05:01:04': {'author_email': 'eriktran15@gmail.com',
                        'author_name': 'Erik Tran',
                        'command': 'continue 2022-05-06_04:00:55 '
                                   'in m1',
                        'hyperparameters': {'ccp_alpha': 0.0,
                                            'class_weight': None,
                                            'criterion': 'gini',
                                            'max_depth': None,
                                            'max_features': None,
                                            'max_leaf_nodes': None,
                                            'min_impurity_decrease': 0.0,
                                            'min_samples_leaf': 1,
                                            'min_samples_split': 2,
                                            'min_weight_fraction_leaf': 0.0,
                                            'random_state': None,
                                            'splitter': 'best'},
                        'metrics': {},
                        'test_data': None,
                        'train_data': None,
                        'version_type': 'continue'}}}
```

Figure 5.24: Example execution of select-command with its output. Displays the latest version in the metadata of a model, specifically after execution of continue-command.

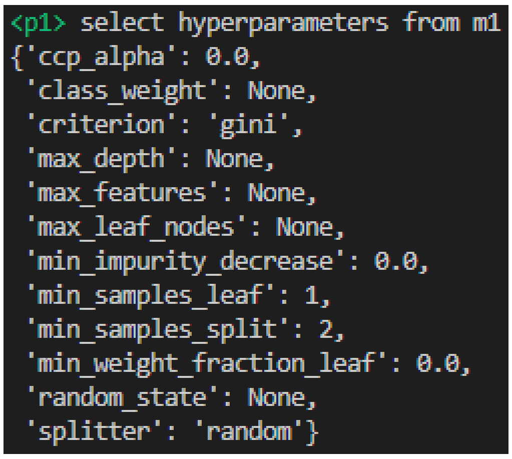
Let's update a hyperparameter of a model.



```
<p1> update m1 set splitter = random
Hyperparameter splitter has been updated with the value random
```

Figure 5.25: Example execution of update-command with its output. Updates a hyperparameter of a model.

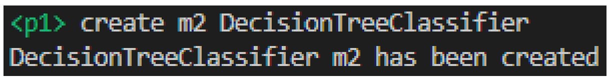
Check the updated hyperparameter.



```
<p1> select hyperparameters from m1
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'random'}
```

Figure 5.26: Example execution of select-command with its output. Displays hyperparameters of a model.

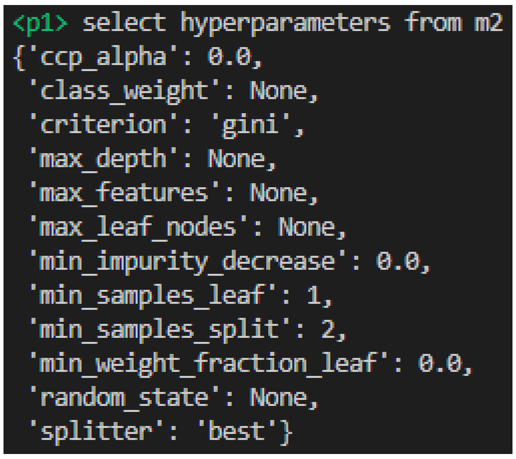
To showcase the filtering feature, let's create another model for demonstrating purposes.



```
<p1> create m2 DecisionTreeClassifier
DecisionTreeClassifier m2 has been created
```

Figure 5.27: Example execution of create-command with its output. Creates a machine learning model of class DecisionTreeClassifier called m2.

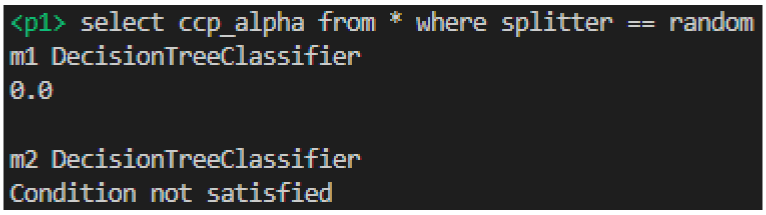
Display the current hyperparameter values of the new model.



```
<p1> select hyperparameters from m2
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Figure 5.28: Example execution of select-command with its output. Displays hyperparameters of a model.

Now, select a specified hyperparameter (or any other choice of model asset) of every model that satisfies a specified condition.



```
<p1> select ccp_alpha from * where splitter == random
m1 DecisionTreeClassifier
0.0

m2 DecisionTreeClassifier
Condition not satisfied
```

Figure 5.29: Example execution of select-command with its output. Showcasing the filtering feature.

Everything that has been shown right now is partially what the query language can do. It is possible to work with other machine learning model classes (Scikit-learn classes) as well. Working with multiple machine learning models at the same time is also possible by utilizing the filtering feature which increases efficiency and usability. It is possible to validate machine learning models, but is not showcased in this example as DecisionTreeClassifier is not a "validate-able" class. A class that is "validate-able" is defined by Scikit-learn. Git is not showcased here, as it is used as the regular way software engineers routinely use.

6 Discussion

This chapter discusses different ways to support several ideas provided by the interviewees during evaluation in both first phase and second phase, and ends with a discussion regarding different threats to validity of the thesis.

6.1 Support for Scalability

Scalability is a potential requirement formed by the insights from the first phase of interviews. A high scalability should be able to achieve a sufficient performance efficiency when working large datasets. A sub-requirement to support scalability is to have a scalable data storage capacity such as a database. Relying only on the data storage capacity of a personal computer would not be sufficient as it is limited. The query language would require relying on an external database to support scalability. MongoDB is an example of a database that could be supported [31] in order to achieve a scalable data storage capacity. When working with larger datasets, it would require another process which is data mining. The process of data mining needs to be efficient as well. There are several efficient data analytics methods for data mining discussed in a survey paper published by Chun-Wei et al. [32]. The process of data mining might encounter some kind of problem when working with larger datasets such as clustering, classification, association rules and sequential patterns. Each problem type contains different methods as solutions e.g. BIRCH [32] for clustering problems. These solution methods could improve the scalability and potentially be supported in the query language of this thesis.

6.2 Support for Data Analysability

Data analysability is a potential requirement formed by the insights from the first phase of interviews. A high data analysability should be able to visualize and plot different types of graphs such as bar graphs, pie charts, box plots, histograms, line charts and scatter plots [33] by utilizing a library called Matplotlib [34]. This improves the interpretability of datasets and allows the users to detect outliers which are data points in a dataset that differs significantly from other data in the dataset which could be crucial information [35]. Data analysability also helps with detecting empty or missing values in entries when working with large datasets.

6.3 Support for Any Arbitrary Models

Support for any arbitrary models of any class is a potential improvement suggested by an interviewee from the second phase of interviews. The support for this expands the query language to more advanced users as they might need to use their own defined class in the query language instead of existing Scikit-learn classes.

6.4 Support for Other Data Types

Support for other data types such as image data or audio data could be implemented in the future. The current supported data types for now are text-based data (numerical or string) in table-structured datasets.

6.5 Threats to Validity

The threats to validity are discussed in three different aspects: internal validity, construct validity and external validity.

6.5.1 Internal Validity

A threat to internal validity relates to the elicitation of requirements. A few requirements were already defined before the first phase of interviews. There are trade-offs by defining the requirements before versus after the first phase of interviews. Defining requirements before will allow the participants to confirm the requirements. This might have biased the results more towards remarks on our particular solution, instead of the intended users' needs. On the other hand, it avoided the risk of obtaining too sporadic comments, in which case it would have been more difficult to form a consistent set of requirements. Defining requirements after have a benefit of knowing what the participants' want in query language from scratch. A problem by letting participants having a high impact on the requirements, might lead to a problem if the participants' experience is not sufficient in the topic of this thesis. The participants might give different answers for requirements which might be contradicting. Therefore, a few requirements for query language which are elicited from research papers are defined in prior. However, there are also some open-ended questions asked during the first phase of interviews such as "*Are there any missing requirements that would be good to have, based on your own experiences?*" and "*What do you think about the usage of the query language, based on what has been shown? What do you like, what needs improvement?*" which enables the participants to think more critically.

Another threat to internal validity relates to the feature implementation. The features are implemented and then evaluated during the second phase of interviews. As the same reason for eliciting the requirements before versus after interviews, the features implemented beforehand could have biased the results more towards remarks on our particular solution. However, there are several open-ended questions asked during second phase of interviews such as "*Do you have ideas of new features that can further improve the usefulness of the query language?*" which enables the participants to think more critically as well.

Another threat to internal validity relates to the systematic thematic analysis of interviews. The codes might not be correctly labeled. The labels might not be correctly converted into themes, which might affect the internal validity.

6.5.2 Construct Validity

A threat to construct validity could possibly be potential misunderstandings of the terms from the interview questions by the interviewees. An example is that Question Q1.4 is supposed to be about the model interpretability and not interpretability of the query language.

Another threat to construct validity is that the GQM metrics chosen to be answered during interviews might not actually answer the stated GQM questions. An example is that Metric M3 "Developers' rating on the usefulness of the query language of this thesis project" does not exactly answer the Question Q1 "Who would hypothetically use the querying tool?". But it does provide some ideas about why it may be used or not based on its usefulness.

6.5.3 External Validity

A threat to external validity is the experience of the participants. The interviewees that participated during the evaluation of the query language should have the required knowledge and experience in order to be able provide qualitative insights. During the first phase of interviews, most of the interviewees' experiences have not been measured in some kind of concise way. An assumption that most of them are experienced enough is the fact that they have been lectured or/and supervised by my supervisor who is experienced in AI engineering [25]. Most of the participants are attending / have attended a software engineering course called "Software engineering for data-intensive AI applications" which is in the same context as this thesis of developing an AI-enabled software system. During the second phase of interviews, the participants' experience has been measured by their own ratings on topics related to this thesis. A partial threat to this external validity is their biases on their rating scores on their own experiences. The identity of a particular interviewee during both first phase and second phase of interviews were not known to the thesis supervisor, which avoided an additional source of bias.

Another threat to external validity is the fact that two of the participants during the second phase of interviews have also participated during the first phase of interviews. It is possible that the results are affected, which is hard to avoid in this evaluation setup. However, the positive side with it is that they are well-informed about the query language from the first phase of interviews, which potentially gives them additional time to think critically during the second phase of interviews and are able to provide more qualitative insights.

Another threat to external validity is the low amount of participants for interviews. It is necessary to thrive for an accurate evaluation as possible from the general experienced users. It is also necessary to thrive for as unbiased opinions as possible by including more participants. A survey could helped with quantitative answers from the participants, however interviews provided more qualitative insights which would be fair to have a low amount of participants, which were more focused for this thesis.

7 Conclusion

The need for a new tool with improved versioning support for machine-learning-based systems dedicated for software engineers has been indicated in several studies. Therefore, a query language that is able to manage machine learning assets with improved versioning support has been developed that is more focused towards software engineers in this thesis. The developed query language has been evaluated by four participants in each phase of interviews.

The first phase was about requirements, and confirming the prior defined requirements for the query language. The prior defined requirements were overall a good set of requirements. An interviewee helped with forming a requirement of data analysability which would improve the usefulness of the query language. Another interviewee helped with forming a requirement of scalability which would allow the query language to be able to work with larger datasets. These requirements are discussed further in this chapter in Section 7.2 about potential solutions of features that would help with achieving these requirements.

The second phase of interviews was about features. The set of questions during the second phase of interviews included questions to assess the experience of the interviewees as well as questions about the features. The set of questions contained a mix of open-ended, closed-ended, rating and multiple choice questions. The questions helped with calculating a set of metrics of the GQM approach applied for the second phase of interviews. The effectiveness of the querying tool is evaluated. Based on the GQM results, those who would hypothetically use this query language are users who are in the beginner-level in management of machine learning assets and average-experienced users in general query languages. The usefulness of the query language of this thesis is at a medium-level. It is more useful for practitioners and less useful for researchers. The feature that was the most useful according to the GQM results was the "continue" feature that has the reproducibility functionalities. The ease of using the query language of this thesis is at a high-level meaning the query language is very easy to use.

7.1 Limitations and Delimitations

The whole implementation process of the query language is done in a single type of operating system: Windows. Specifically, the query language is developed in Windows 11. This is a delimitation as Windows is the only accessible operating system to the author during the development. Testing of the query language is not conducted in other operating systems.

7.2 Future Work

The usefulness of the query language could be further improved by implementing new potential features that support the ideas provided by the interviews which are discussed in Chapter 6, such as support for scalability, data analysability, any arbitrary models and other data types. In the future, it could also potentially be developed in other operating systems other than Windows which would expand to more users.

Bibliography

[1] "Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency." [Online]. Available: https://git-scm.com/

[2] "Query Language." [Online]. Available: https://www.techopedia.com/definition/3948/query-language

[3] "Query." [Online]. Available: https://www.techopedia.com/definition/5736/query

[4] S. Idowu, O. Osman, D. Strüber, and T. Berger, "On the Effectiveness of Machine Learning Experiment Management Tools," 2022.

[5] "PostgreSQL: The World's Most Advanced Open Source Relational Database." [Online]. Available: https://www.postgresql.org/

[6] "Experiment tracking and model registry for teams doing ML at a reasonable scale." [Online]. Available: https://neptune.ai/

[7] S. Idowu, D. Strüber, and T. Berger, "Asset Management in Machine Learning: A Survey," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 51–60.

[8] S. Huang, "Effective data versioning for collaborative data analytics," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1925–1938.

[9] L. Xu, S. Huang, S. Hui, A. J. Elmore, and A. Parameswaran, "Orpheusdb: a lightweight approach to relational dataset versioning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1655–1658.

[10] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1723–1726.

[11] A. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran, "Datahub: Collaborative data science & dataset version management at scale," *arXiv preprint arXiv:1409.0798*, 2014.

[12] A. Barrak, E. E. Eghan, and B. Adams, "On the co-evolution of ml pipelines and

source code-empirical study of dvc projects," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER).* IEEE, 2021, pp. 422–433.

[13] A. Kumar, M. Boehm, and J. Yang, "Data Management in Machine Learning: Challenges, Techniques, and Systems," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1717–1722. [Online]. Available: https://doi.org/10.1145/3035918.3054775

[14] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards effective foraging by data scientists to find past analysis choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[15] "Google Colab." [Online]. Available: https://colab.research.google.com/

[16] "Google Docs." [Online]. Available: https://www.google.com/docs/about/

[17] 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.

[18] "Project Jupyter." [Online]. Available: https://jupyter.org/

[19] P. Sugimura and F. Hartl, "Building a reproducible machine learning pipeline," *arXiv preprint arXiv:1810.04570*, 2018.

[20] E. Knauss, "Constructive master's thesis work in industry: Guidelines for applying design science research," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET).* IEEE, 2021, pp. 110–121.

[21] "ISO/IEC 25010:2011," International Organization for Standardization, Standard, Mar. 2011. [Online]. Available: https://www.iso.org/standard/35733.html

[22] "textX." [Online]. Available: https://textx.github.io/textX/3.0/

[23] S. Counsell, "Do student developers differ from industrial developers?" in *ITI 2008-30th International Conference on Information Technology Interfaces.* IEEE, 2008, pp. 477–482.

[24] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert, and J. Siegmund, "Efficiency of Projectional Editing: A Controlled Experiment," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 763–774. [Online]. Available: https://doi.org/10.1145/2950290.2950315

[25] D. Strüber, "Daniel Strüber." [Online]. Available: https://www.danielstrueber.de/

[26] V. R. B. G. Caldiera and H. D. Rombach, “The goal question metric approach,” *Encyclopedia of software engineering*, pp. 528–532, 1994.

[27] C. Kästner, “Versioning, Provenance, and Reproducibility in Production Machine Learning.” [Online]. Available: https://ckaestne.medium.com/versioning-provenance-and-reproducibility-in-production-machine-learning-355c48665005

[28] Z. C. Lipton, “The Mythos of Model Interpretability,” *CoRR*, vol. abs/1606.03490, 2016. [Online]. Available: http://arxiv.org/abs/1606.03490

[29] “sklearn.model_selection.cross_validate.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html

[30] “Regular Expressions 101.” [Online]. Available: https://regex101.com/

[31] “Database. Deploy a multi-cloud database.” [Online]. Available: https://www.mongodb.com/atlas/database

[32] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, “Big data analytics: a survey,” *Journal of Big data*, vol. 2, no. 1, pp. 1–32, 2015.

[33] A. Bhandari, “A Beginner’s Guide to matplotlib for Data Visualization and Exploration in Python.” [Online]. Available: https://www.analyticsvidhya.com/blog/2020/02/beginner-guide-matplotlib-data-visualization-exploration-python/

[34] “Matplotlib: Visualization with Python.” [Online]. Available: https://matplotlib.org/

[35] C. Y. Wijaya, “Outlier — Why is it important?” [Online]. Available: https://towardsdatascience.com/outlier-why-is-it-important-af58adbefecc

A

Appendix

A.1 First Phase Interview Questions

Q1.1: Based on your own experiences, would the requirement model versioning be useful?

Q1.2: Would the requirement model reproducibility be useful?

Q1.3: Would the requirement data provenance be useful?

Q1.4: How important is the requirement interpretability?

Q1.5: How important is the requirement performance efficiency?

Q1.6: How important is the requirement interoperability? Is it important to integrate with Git?

Q1.7: Are there any missing requirements that would be good to have, based on your own experiences?

Q1.8: What do you think about the usage of the query language, based on what has been shown? What do you like, what needs improvement?

A.2 Second Phase Interview Questions

Q2.1: What are you studying?

Q2.2: Which study degree level?

Q2.3: How would you rate your own experience in managing machine learning assets (code, data, models...) on a scale from 1 (very inexperienced) to 5 (very experienced)?

Q2.4: How would you rate your own experience in query languages on a scale from 1 (very inexperienced) to 5 (very experienced)?

Q2.5: How would you rate your own understanding of the query language on a scale from 1 (very low) to 5 (very high)?

Q2.6: Based on your own experience in developing ML projects: How would you rate the usefulness of the query language on a scale from 1 (not useful at all) to 5 (very useful)?

Q2.7: Which features are the most useful?

Q2.8: Which features are the least useful?

Q2.9: Which of the features need improvement? For each of them: what kind of improvement?

Q2.10: Do you have ideas of new features that can further improve the usefulness of the query language?

A.3 First Phase Interview Code-Label Pairs

Q1.1: Based on your own experiences, would the requirement model versioning be useful?

> **Code:** "Eh, I believe so. So, I'm really feeling right now that I probably am not qualified. But, like, yes, it seems like interesting and useful." - Participant 1

Label: Agreement on model versioning

> **Code:** "Eh, I think so. I mean the model versioning you are talking about, different, if you train different models just to see, and give it different versions and the query language supports that. Yeah, for sure. I'd say that is one of the key points of machine learning modeling. So, yes." - Participant 2

Label: Agreement on model versioning

> **Code:** "I think it's always useful to have that history, to be able to sort of, backtrace what you've done. That's always helpful no matter what. I think model versioning is absolutely something good to have, for sure." - Participant 3

Label: Agreement on model versioning

> **Code:** "Yes, when working with machine learning and training models, it is clear that it is important to keep an eye on the different versions." - Participant 4
> **Note:** Transcript translated from Swedish to English

Label: Agreement on model versioning

Q1.2: Would the requirement model reproducibility be useful?

> **Code:** "Uh, yeah, I think so." - Participant 1

Label: Agreement on model reproducibility

> **Code:** "Yeah, for sure. I'd say it's important. In my opinion, then you can improve accuracy." - Participant 2

Label: Agreement on model reproducibility

Code: "Yeah, absolutely." - Participant 3

Label: Agreement on model reproducibility

Code: "So, yes it is important in general." - Participant 4
Note: Transcript translated from Swedish to English

Label: Agreement on model reproducibility

Q1.3: Would the requirement data provenance be useful?

Code: "Uh, yeah, it's like very important, because like you would need to... It is always important to be able to track changes, ehm, who made them, and go back, see what went wrong, and talk to them and find out how to solve things. So yeah, absolutely." - Participant 1

Label: Agreement on data provenance

Code: "Eh, yes. I'd say important. We worked a bit with traceability. And especially in projects is really useful to see who touches it, and what the latest, and just get all the information you can regarding that" - Participant 2

Label: Agreement on data provenance

Code: "Yes, I only have experience from working with these things in very very small group. So you kinda always have a sense of what everyone is doing and who is doing what. But, if you are working with models in larger team. If you have like commits everyday by several people. This is something going to be turned out useful." - Participant 3

Label: Agreement on data provenance

Code: "Of course it is, for the same reason to keep track of... So, GitHub keeps track of commits in software development in general. But you can see similar things when it comes to machine learning, that you keep track of who does what and train which models, I guess." - Participant 4
Note: Transcript translated from Swedish to English

Label: Agreement on data provenance

Q1.4: How important is the requirement interpretability?

Code: "Yes, I believe so, yes, absolutely." - Participant 1

Label: Agreement on interpretability

Code: "Aha, hmm, depending on who sees it, if it's just software developers and people who understands it, then it's useful, but it can be kinda redundant for people who doesn't." - Participant 2

Label: Depends on the type of users when it comes to interpretability

Code: "Yeah, it's important. I mean it's difficult enough with machine learning models to sort of get a grip of why stuff is happening in the first place, so every ounce of interpretability is gonna be good." - Participant 3

Label: Agreement on interpretability

Code: "Uh, it depends on what applications it's about. For example, I predict which medications a patient should take, for example, to determine whether that patient has cancer or not. When you make such decisions, you want to understand how the machine learning model processed, when it came to that decision. Because patients want to know... Like, the model says I have cancer, but I also want to know why it says so. For such applications, it is important. But then there are other things, such as predicting an email is spam or not, which is not important." - Participant 4
Note: Transcript translated from Swedish to English

Label: Depends on the type of applications when it comes to interpretability

Q1.5: How important is the requirement performance efficiency?

Code: "It would be right? Yeah, I find myself not being able to... But, but, I'm pretty sure that would be important yeah." - Participant 1

Label: Agreement on performance efficiency

Code: "I'd say very important. You can improve the versions by retraining and with more performance, then you can decrease the time it takes." - Participant 2

Label: Agreement on performance efficiency

> **Code:** "Well... I don't know, both yes and no, I think. Cuz, I mean performance is important overall. But then again, we train machine learning model using Python, which is not a very performant language right? But you don't want the performance to go down the drain, just because you are using like a query language, like a special tool, for inserting model data and training data. You don't want to query language to be slow, you don't want it to make like stupid decisions about where to put things and make things even slower. So yeah, it's important. In the case of versioning control, maybe not as important. Cuz, that's mostly about seeing like who did what. But it's important, but less to some aspects." - Participant 3

Label: Depends on what kind of aspect when it comes to performance efficiency

> **Code:** "So if we take a comparison with Git which is a version control system. Of course I care if Git is slow or fast. So if Git is very slow, if I run a Git push for example, and it is very slow and I have to sit and wait, it's clear that I get annoyed by it, so it has an impact. The training time has to do with the fact that you use Scikit-learn, for example. So the training time is involved in making which library you use you use to train the model. For the query language as a whole, it is generally important." - Participant 4
> **Note:** Transcript translated from Swedish to English

Label: Agreement on performance efficiency

Q1.6: How important is the requirement interoperability? Is it important to integrate with Git?

> **Code:** "Yeah, absolutely, you would be working with that all the time. Yeah, there is not doubt about that." - Participant 1

Label: Agreement on interoperability

> **Code:** "For me, I somewhat agree. But it won't be the most important non-functional requirement." - Participant 2

Label: Agreement on interoperability

> **Code:** "Eh, I mean yeah. Maybe some parts of Git would actually be beneficial to enrich this versioning. Like you could connect different types of commits and different types of histories. Like your query history and Git history, and connect commits, sort of. Like wrap them together, or entangle them a bit to be able track what's going on." - Participant 3

Label: Agreement on interoperability

> **Code:** "So that the version not only keeps track of your code files that GitHub keeps track of, but it is also in combination with also keeping track of history from the query language. Yes, I guess so. Sees no reason why it would not be useful." - Participant 4
> **Note:** Transcript translated from Swedish to English

Label: Agreement on interoperability

Q1.7: Are there any missing requirements that would be good to have, based on your own experiences?

> **Code:** "Uhm, right now, I haven't thought of any. I think the ones you listed are perfect, yeah." - Participant 1

Label: Approval of no missing requirements

> **Code:** "Cuz, the requirements you have are very broad, cuz, I think they capture the most important things." - Participant 2

Label: Approval of no missing requirements

> **Code:** "No, I think this is already a solid list. I can't come up with any more on the spot. I think this is a very good start of requirement specification." - Participant 3

Label: Approval of no missing requirements

> **Code:** "I do not think so. But it is possible that there are stuff." - Participant 4
> **Note:** Transcript translated from Swedish to English

Label: None

Q1.8: What do you think about the usage of the query language, based on what has been shown? What do you like, what needs improvement?

> **Code:** "I really like, ehm..., I think it's..., yeah. Hmm, there was nothing that stands out to me, but me like, I'm constantly repeating, like my lack of experience, that just..., so I have nothing to contribute here." - Participant 1

Label: Lack of experience

Code: "I like that it is very easy to use. Like the syntaxes, it follows all the standard query languages. So that was my favorite part. I can see the usefulness for when doing in projects. Ehm, I mean cuz when we did the training, cuz you had a small amount of data, but if the data was bigger, does it display any sort of how long it would take or... If we working with big data and it would take like couple of hours, then I'd say it would be very useful. Cuz then you can..., you don't have to cancel, cuz you don't know what's going on in the other side of the console. It's nice to see some sort of, oh it's alive, it's still working." - Participant 2

Labels: Easy to use, Follows standard sequel commands, How to handle big data

Code: "Ehm, I think it's good that it sort of strives to look like regular sequel commands a little bit. I think that's a good thing, cuz you always feels right at home, you know what it's about, that's good. Although that comes with the backside if your command were to behave differently in some way from regular sequel command. If it were to perform a function that it is completely, like, it does not look like or feel like regular sequel functions at all, then that might be a bit of a surprise. So there might be small learning bumps. So that might be the backside of it. So there's the feeling, I know this, I've seen this. And there is also, wow this didn't behave the way I expected it to. So it might be a two-edged sword, so to speak. But I like the idea to go for a sequel kind of style." - Participant 3

Labels: Follows standard sequel commands, Possibly can behave differently compared to regular sequel commands, Possible learning bumps

Code: "I think I understand roughly what it's about, but not exactly in detail. It is an SQL-like language. You have made a program that takes care of all the steps that you normally need to do manually, such as starting a GitHub repository, defining your model, training the model. Improvement then would be to implement support for more libraries, so not only for Scikit-learn, but your language could also support PyTorch, Keras or TensorFlow. When it comes to data analysis and stuff like that, because now you just load the data and train. You have to prepare the data and normalize it as well." - Participant 4
Note: Transcript translated from Swedish to English

Labels: Easy to understand the idea, Possible improvement by adding support for more libraries, Possible improvement by adding support for data analysis

A.4 Second Phase Interview Code-Label Pairs

Q2.5: How would you rate your own understanding of the query language on a scale from 1 (very low) to 5 (very high)?

> **Code:** "That's basically the same thing I been working before with, so I would say 4, I understand it." - Participant 1

Label: Rating on own understanding of the query language

> **Code:** "I think I could follow along fairly good, you were getting statistics, uhh, I mean or not statistics per se, but information from the runs that you did using queries basically. Say maybe a 2 or 3." - Participant 2

Label: Rating on own understanding of the query language

> **Code:** "I'd say I understand it pretty well, so a 4." - Participant 3

Label: Rating on own understanding of the query language

> **Code:** "Uhh, I mean 5. I guess." - Participant 4

Label: Rating on own understanding of the query language

Q2.6: Based on your own experience in developing ML projects: How would you rate the usefulness of the query language on a scale from 1 (not useful at all) to 5 (very useful)?

> **Code:** "4." - Participant 1

Label: Rating on the usefulness of the query language

> **Code:** "Okay, based... I would say in just a number short, it's so hard from my perspective, but a 3, a 3 at least." - Participant 2

Label: Rating on the usefulness of the query language

> **Code:** "I think would got it somewhere 3 and 4 region, perhaps" - Participant 3

Label: Rating on the usefulness of the query language

> **Code:** "Uhh, I have to think. So, I mean like, versions for like different models you trained with different hyperparameter setups is very important for machine learning. So, like the versioning part of it, like storing the history is definitely useful. I'm not sure... Like so, I think using the query language to actually train models and develop machine learning systems, then you would need some models that using existing methods. So somebody that is like working for a company maybe and then working on a particular application. And then they are only going to use existing methods. But for me as a PhD student who's researching stuff, making my own models and so on. Then it would be hard to use this query language, right. Cuz, I won't be able to use just any machine learning model. So, for me personally probably not so useful for that reason, but in general it looks interesting. So like usefulness in general is, I would give it maybe a 4. But for me is probably like a 1." - Participant 4

Labels: Rating on the usefulness of the query language, Agreement on versioning functionality, Possible improvement by adding support for any arbitrary models, Hard to use the query language

Q2.7: Which features are the most useful?

> **Code:** "Uhh, I think it is good to switch between different versions of models. It is good to... I think all of them are quite useful, cuz each of them is a part of the model processing or training or performing the prediction. It is a part of what you're doing, it's a part of the whole procedure. But yeah, if I were to prioritize or pick one, it would be the Continue-feature." - Participant 1

Labels: Subjective rating on most useful feature, Agreement on reproducibility functionality

> **Code:** "Mmm, I would think the reusability feature. Just from my understanding, that's what's been a big struggle when you have involvement in people developing, and running different models. Yeah, it's difficult to reproduce results from other teams not within your own, it's been a big issue from my understanding. So yeah, that seems really good." - Participant 2

Labels: Subjective rating on most useful feature, Agreement on reusability functionality

> **Code:** "The calculation stuff is pretty useful. Seems like a nice sort of shortcut to get what you actually after, and what you actually interested in. I also thought that the filtering was pretty neat. So I think that is going to be very helpful as well." - Participant 3

Labels: Subjective rating on most useful feature, Agreement on calculation feature, Agreement on filtering feature

> **Code:** "Ok, by the way when I said 1 just for the previous question, I meant like with respect to the current state of the system, but if you kept developing and added more features, then maybe it would be more than 1. Ok so, the most useful feature, for me it probably would be the way of being able to save the history. Anything that has to do with storing like the history of trained models and also reproducing results and so on." - Participant 4

Label: Subjective rating on most useful feature, Agreement on reproducibility functionality

Q2.8: Which features are the least useful?

> **Code:** "Ehm, I don't know, do I have to pick one that is considered the least useful? I don't have anything in mind, I think all of them are useful." - Participant 1

Label: Approval of no feature to be least useful

> **Code:** "Uhh, hmm... I don't know, it depends... I mean... the... The select, but I mean it's also one of the key features but it doesn't do as much as the reproducibility one. It's really hard. Do I have to say something? Because my opinion is not gonna be... It's gonna be mostly a guess work at this point. I don't think I can provide much value for you actually." - Participant 2

Labels: Approval of no feature to be least useful, Lack of experience

> **Code:** "I don't think there is a feature that is like unwanted in this set of features. So I couldn't really say like that this one should probably go. Nah I think this set of features, this set of functionality is like a good foundation of what we would want to do with a query language like this. Like there are no obvious, like no get rid of this, like this is not useful at all." - Participant 3

Label: Approval of no feature to be least useful

> **Code:** "Probably, well the stuff that had to do with like training models and predicting and so on. Cuz when I do that I would be restricted to existing models, like Scikit-learn models and so on. That would be least useful for me I would say." - Participant 4

Label: Possible improvement by adding support for any arbitrary models

Q2.9: Which of the features need improvement? For each of them: what kind of improvement?

> **Code:** "It's difficult, because, you know, for a short amount of time, you have to get a very deep understanding of all of them and check if any needs improvement. But I'd say right now I don't think any of them do." - Participant 1

Labels: Approval of no features need improvement, More time required to understand

> **Code:** "Mhmm, maybe that you would be able to compare results, if you select two different parameters from two different models if you compare, and... Like if you select one run and then you compare result with another one, then maybe you would see okay this run had this much of an improvement in terms of this but decreases the accuracy here, ehh, something like that maybe. If you train two models on the same datasets, maybe it would be interesting to have an easy view of how they performed differently, or what their parameters were differently." - Participant 2

Labels: Disapproval of no features need improvement, Possible improvement by easing the process of comparison between values produced by different models

> **Code:** "Maybe one would want to tidy up, like if you print a lot of stuff to the console, maybe that needs to be prettified in some way, so that it's more easy to read like, to actually find what you are looking for, so you are not jumping across the screen because the implementation. But that's kinda minor thing I guess." - Participant 3

Labels: Disapproval of no features need improvement, Possible improvement by reducing the need for a lot of scrolling in command console to find the correct version of a model

> **Code:** "So all the features that correspond to using existing models. So I guess... Your models. All models are in Scikit-learn, right. So if you added a way to use any arbitrary models that I can define with my own like Python class for example, as long as it follows like the correct paradigm, like it has a predict function and a fit function and so on. Like, if I can define my own class with those functions and then use them with your framework, then it would be useful. And also support for other libraries other than Scikit-learn like PyTorch, or TensorFlow and so on." - Participant 4

Labels: Disapproval of no features need improvement, Possible improvement by adding support for any arbitrary models, Possible improvement by adding support for other libraries

Q2.10: Do you have ideas of new features that can further improve the usefulness of the query language?

> **Code:** "So my thought is, maybe a user want to retrieve certain information and save them separately. Like you are looking for something, you filter the metric for example and you want this piece of information to be saved to somewhere. Otherwise I don't think uh... Like what I've seen, as I said before, because it was short time to get understanding for all of these features, but from this overview, I don't think any other features are needed." - Participant 1

Labels: Approval of no new features can improve the usefulness, Possible improvement by improving the current features to save values in a file separately, More time required to understand

> **Code:** "I think it would be somewhere based around that, like as for me as I am so new to it, that would be one of the things that I would find interesting, like to easy be able to see the difference between what parameters you are using." - Participant 2

Labels: Disapproval of no new features can improve the usefulness, Possible improvement by easing the process of comparison between values produced by different models, Lack of experience

> **Code:** "I wouldn't say new features, cuz I think these you could get pretty far with these features that already exists, so I guess, to make the language more useful. Maybe just expand on functionality that is already there and refine it. And maybe add more options to these commands that are already in there. I think the main point is that I don't really see an obvious new piece of functionality, so I think, if you want to make it more useful, then like focus on just expanding the functionality that is already there. I don't think you would need to throw in a bunch of new commands just to make it more useful, cuz this would already get you pretty far." - Participant 3

Label: Approval of no new features can improve the usefulness

> **Code:** "So I guess I just mentioned, right. Just more support for more models, and I mean I guess we talked about it in the previous interview with like data analysis. Being able to analyse the data, because that is the very important part in machine learning. Maybe some command that like after you loaded your data, you can plot different statistics regarding the data, and manipulate the data, and normalise and so on. The datasets that you show me now were just example toy datasets, right. But in practice, you sometimes deal with data with millions of entries. In some of the columns or rows there might be missing values and so on, so you might have to remove some data to prevent that and so on." - Participant 4

Labels: Disapproval of no new features can improve the usefulness, Possible improvement by adding support for any arbitrary models, Possible improvement by adding support for data analysis