Automated Highlight Identification in a Data Profiling System

Christodoulos Antoniou

Diploma Thesis

Supervisor: Prof. P. Vassiliadis

Ioannina, July 2023

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Acknowledgments

This thesis would not be possible without the guidance of my supervisor, professor Panos Vassiliadis. I am grateful for his ongoing mentorship throughout the entirety of the thesis. Despite his tough schedule and abundance of other tasks, he was always eager to provide me with not only invaluable assistance and guidance in the development of the thesis, but also general advice and guidance for my career as a computer scientist. It would be unfair had I not acknowledged him as an exemplar of a professor and a very kind human.

Also, I would like to thank my family members and all the people close to me for always being there for me whenever I was in need. Each of them has been supportive and helpful to me in their own way. And specifically, I would like to express my gratitude towards Irene. Chances are I would not be here today without her, and this is the least I can say.

I believe that this chapter must not be very long, as gratitude and appreciation are better expressed through different means rather than plain words on a piece of paper. Thus, the above will have to suffice for the time being.

To all the people mentioned above, I am forever grateful.

July 2023

Christodoulos Antoniou

Abstract

The increasing complexity and size of data have made it evident that data holds significant value since the 1960s. Exploratory Data Analysis (EDA) is a procedure used by data scientists to analyze and summarize data sets, enabling them to discover patterns, identify anomalies, and test hypotheses. However, the lack of automation in EDA poses challenges and hinders the objective of extracting valuable insights from data. To address this issue, Pythia, a Java application that utilizes the Apache Spark engine, was developed to facilitate automated EDA. The general idea of the system is that it accepts a dataset as input and automatically produces valuable insights about it. In the context of this thesis, Pythia is extended to automatically identify highlight patterns in the input dataset. The data set passes by a set of highlight extractor modules which check for both holistic highlights that concern an entire area of the dataspace and point-based highlights which only concern an individual point in the dataspace. Specifically, Pythia is extended to identify dominance and outlier highlights. Dominance is a holistic highlight pattern that allows analysts to identify partial or total dominance occurrence for coordinates of the input dataset over a selected measurement. Outlier is a point-based highlight pattern that allows analysts to detect numerical data points whose value is notably different from the others in the input dataset. The identified highlight patterns are recorded on main memory and exported as part of the generated reports. Pythia is also augmented with the ability to allow for selection of the data analysis techniques that should be executed for each input dataset. Lastly, the contributed extensions to the system are thoroughly tested and extensively experimented. Following the conventions of the existing project, the contributed code is written in Java using Apache Spark for fast analytics computations. The code is also easily scalable with additional highlight patterns.

Keywords: Data Science, Dataset Profiler, Exploratory Data Analysis, Data Patterns, Java, Apache Spark

Περίληψη

Από τις αρχές της δεκαετίας του 1960, έχει γίνει σαφές ότι τα δεδομένα έχουν σημαντική αξία. Στις σύγχρονες μέρες, όλοι μας παράγουμε και συλλέγουμε δεδομένα με κάποιο τρόπο και αυτό έχει ως αποτέλεσμα τα δεδομένα να γίνονται όλο και πιο κολοσσιαία σε μέγεθος και πολύπλοκα σε δομή. Η Διερευνητική Ανάλυση Δεδομένων (Exploratory Data Analysis - EDA) είναι μία διαδικασία που χρησιμοποιείται από επιστήμονες δεδομένων για την ανάλυση και κατανόηση συνόλων δεδομένων. Περιλαμβάνει διάφορες μεθόδους ανάλυσης, όπως οι ομαδοποιήσεις, οι οπτικοποιήσεις δεδομένων και οι στατιστικές αναλύσεις, επιτρέποντας την ανακάλυψη προτύπων και την ανίχνευση ανωμαλιών. Ωστόσο, η έλλειψη αυτοματοποίησης στην Διερευνητική Ανάλυση Δεδομένων μπορεί να την κάνει ιδιαίτερα κουραστική, εμποδίζοντας την εξαγωγή πολύτιμων ερευνητικών αποτελεσμάτων και αποθαρρύνοντας τη συνέχιση της ανάλυσης.

Για να αντιμετωπιστεί αυτή η πρόκληση, αναπτύχθηκε το Pythia, ένα πρόγραμμα σε Java που χρησιμοποιεί την μηχανή Apache Spark για να επεξεργαστεί γρήγορα σύνολα δεδομένων μεγάλου όγκου. Το Pythia αποδέχεται ένα σύνολο δεδομένων ως είσοδο, το επεξεργάζεται, και παράγει αυτόματα πολύτιμες πληροφορίες για αυτό. Το σύστημα Pythia έχει τη δυνατότητα να αξιολογεί την ποιότητα των δεδομένων, να υπολογίζει στατιστικά, ακόμα και να παράγει δένδρα αποφάσεων. Όλα αυτά γίνονται αυτόματα και τα αποτελέσματα παρουσιάζονται στον αναλυτή μέσω μιας λεπτομερούς αναφοράς.

Στο πλαίσιο αυτής της διατριβής, το Pythia επεκτείνεται ώστε να εξετάζει το σύνολο δεδομένων για την ύπαρξη μοτίβων. Ο πυρήνας της αυτόματης ανάλυσης δεδομένων του συστήματος εμπλουτίζεται με μονάδες εξαγωγής μοτίβων, οι οποίες εξετάζουν το σύνολο δεδομένων και αναγνωρίζουν τόσο ολικά όσο και σημειακά μοτίβα. Τα ολικά μοτίβα αφορούν μεγάλες περιοχές στο χώρο δεδομένων του συνόλου, ενώ τα σημειακά μοτίβα αφορούν συγκεκριμένα σημεία. Συγκεκριμένα, το Pythia επεκτείνεται για να αναγνωρίζει μοτίβα κυριαρχίας και μοτίβα ακραίων σημείων. Τα μοτίβα κυριαρχίας είναι ολικά μοτίβα, τα οποία επιτρέπουν στους αναλυτές να εντοπίζουν μερική ή απόλυτη εμφάνιση κυριαρχίας σε συντεταγμένες του συνόλου δεδομένων για μία επιλεγμένη μετρική. Αντίθετα, τα μοτίβα ακραίων σημείων είναι σημειακά μοτίβα, τα οποία επιτρέπουν στους αναλυτές να ανιχνεύουν αριθμητικά σημεία δεδομένων των οποίων η τιμή διαφέρει σημαντικά από τα άλλα σημεία στο σύνολο δεδομένων. Τα εντοπισμένα μοτίβα καταγράφονται στην κύρια μνήμη ως αντικείμενα Java και παρουσιάζονται ως μέρος της δημιουργηθείσας αναφοράς ευρημάτων.

Το σύστημα επίσης επαυξάνεται με τη δυνατότητα καθορισμού των τεχνικών ανάλυσης δεδομένων που εκτελούνται για κάθε σύνολο δεδομένων. Τέλος, οι προαναφερόμενες επεκτάσεις στο σύστημα δοκιμάζονται με διεξοδικούς ελέγχους και αξιολογούνται με εκτενείς πειράματα. Σύμφωνα με τη σύμβαση του υπάρχοντος έργου, ο συνεισφερόμενος κώδικας γράφεται σε Java και είναι εύκολα επεκτάσιμος και κλιμακούμενος με επιπλέον μονάδες εξαγωγής μοτίβων.

Λέξεις Κλειδιά: Επιστήμη Δεδομένων, Δημιουργία Προφίλ για Σύνολα Δεδομένων, Διερευνητική Ανάλυση Δεδομένων, Μοτίβα Δεδομένων, Java, Apache Spark

Table of Contents

Chapter 1. Introduction

This chapter presents a brief overview of the subject and the general field of study of the thesis, as well as an overview of the structure of the following sections of the thesis paper.

1.1 Thesis subject

Since the early 1960s, it has become clear that data holds significant value. Nowadays, everybody is collecting and producing data in some way and as a result data is getting increasingly complex, interconnected, and colossal in size. Naturally, some sets of data might hold more value than others. The selection of a suitable data set for a certain purpose can be a challenge of its own. But, even if we were to obtain an incredibly valuable set of data, having possession of the data alone is not synonymous with acquiring the value it contains. Data must be broken down and analyzed in order to summarize its characteristics, assess its quality, and determine whether it contains valuable information. This procedure is known as Exploratory Data Analysis (EDA) [Agga15].

Essentially, EDA is a procedure used by data scientists to analyze data sets and summarize their characteristics and it helps determine how to manipulate data to get the insights one might need, such as discovering patterns, spotting anomalies, or testing hypotheses. Scientists can identify obvious errors, better understand patterns within the data and find interesting relations among the variables of the data set [IBMC20].

EDA usually involves various analysis methods such as clustering, data visualization and statistical analysis. For instance, an analyst might want to detect and exclude outliers from a data set to decrease its variability. Data visualization is also a very common method, as it transforms data into a form that is easy to understand, thus allowing analysts to quickly gain insights into large amounts of data. There are numerous tools that provide the above-mentioned analysis methods. However, such analysis methods are typically conducted in an interactive manner, i.e., the analyst has to specify the desired analysis method along with any required parameters and perform any potential data preprocessing in order to extract the desired insights. Afterwards, the analysis results which are produced can be evaluated, reviewed, and used for follow-up analysis, repeating the above process.

Lack of automation in the context of data exploration is a severe issue for several reasons. Data exploration can be a tiresome task, especially for larger data sets as the extracted data is highly likely to lack structure and organization. Therefore, lack of automation

hinders the main objective of EDA, which is to gain valuable insights from the data while it also discourages any potential follow-up analysis [PRSD21]. In an attempt to solve this issue, a system called Pythia [Alex22] was developed. The main objective of the system is to facilitate automated EDA, accepting a data set as input which is processed and a valuable insights overview about it is automatically generated. Pythia is a Java application which utilizes the Apache Spark engine to process the data set and generate the results in an automated manner. Apache Spark [Apac21] is an optimized and unified engine for executing data engineering and analytics on large scale data. It is widely used in the general field of data science as it is capable of rapidly processing vast amounts of data.

In its state prior to this thesis, Pythia was relatively limited to assessing the quality of the given data set. In essence, the system was still in an early stage of development. More specifically, it was capable of accepting a data set as input and generating a detailed statistics profile about it including automated correlation calculation. The system was also capable of generating decision trees for labeled fields of the data set in a simplistic manner. All of the above is done automatically, and the results are presented to the analyst in a detailed report [Alex22].

In the context of this thesis, Pythia is extended to not only assess the quality of the input dataset but also produce valuable insights about it in an automated manner. The system is extended with highlight extractor modules that examine the dataset and identify both holistic highlights and point-based highlights. Holistic highlights concern an entire area of the dataspace and point-based highlights only concern a specific point in the input data. Each highlight extractor module consists of data preparation procedures such as querying the dataset and a highlight identification algorithm which actually checks for the existence or absence of the highlight pattern at hand. Identification results are generated and stored in concrete result objects on main memory. In the end, once all the automated data analysis procedures of the system have finished and the result objects have been generated, the highlight identification results are exported to report files, that contain insightful information.

In further detail, Pythia is extended to identify dominance and outlier highlights. Dominance is a holistic highlight pattern that allows analysts to identify partial or total dominance occurrences for specific coordinates of the input dataset w.r.t. a selected measurement. For instance, an analyst might obtain insights such as which occupation has worked the most or least hours per week among different countries. Another example would be a highlight such as which car model had the highest or lowest price among different years. Pythia is also augmented with the ability to accept input parameters that

determine whether the selection of dominance measurement and coordinate columns should be performed automatically or manually by the data analyst. On the other hand, outlier is a point-based highlight pattern that allows analysts to detect numerical data points whose value is notably different from the others in the input dataset.

Overall, Pythia is extended with the ability to identify highlight patterns such that valuable insights about the input dataset are produced in an automated manner. The development of the newly added features is done in a flexible way such that the system can be easily extended with more highlight extractor modules. Lastly, in order to improve the scalability and usability of the system, Pythia is also augmented with the ability to accept parameters regarding which parts of the automated data analysis pipeline should be executed, such that certain data analysis parts can be excluded or executed individually.

1.2 Thesis structure

The following chapters showcase the contributions that are implemented in the context of this diploma thesis in great detail.

More specifically, chapter 2 describes the thesis objective and presents the scientific background that is required to properly comprehend the thesis, describing data science terms and techniques as well as technologies used in the system, such as Apache Spark. The state of the Pythia system prior to this thesis is also described.

Chapter 3 presents a higher-level overview of the software architecture and execution flow of the Pythia dataset profiling system as well as an in-depth description of the software design and implementation regarding the automated highlight identification problem. Details regarding software testing, installation and execution are also provided, along with information about maintenance and scalability of the software.

Chapter 4 describes the methodologies used for the experimental evaluation of the system which involved measurement of execution times for three (3) different levels of abstraction. It also presents the results of the experimental evaluation in detail, in the form of tables and bar charts.

Lastly, Chapter 5 contains a synopsis of the contributions of this thesis and presents a list of suggestions with future extensions for the system. In the end of the document, there are references to all bibliography and all web links that were visited during conduction of the thesis.

Chapter 2. Subject Description

2.1 Thesis objective

The objective of this thesis is to contribute to and extend the Pythia system. The main objective of the Pythia system is to allow an analyst to obtain valuable insights of a data set in an automated manner. The system accepts a data set as input and generates a detailed statistical profile with insights of the data set. The contribution revolved around the enrichment of the generated statistical profile.

In detail, the added requirements to the system are organized as follows:

- After a data set has been successfully loaded into the system, the system should be able to check for patterns among the fields of the data set and identify interesting subsets of the data set that pertain to these patterns. The term "highlights" is used to refer to these patterns that actually exist in the data set. The highlight extractor modules must be easily extendable to allow for pattern additions without any significant maintenance of the pattern manager class. Patterns can be of different types:
    - **Two-dimensional highlight-related patterns**. This pattern type consists of a measurement column M and a regulator column X. We are looking for properties such as which values of X produce the top-K values of M, outliers, or trend change patterns in cases where X describes a time attribute.
    - **Two-dimensional global property patterns**. This pattern type also consists of a measurement column M and a coordinate column X. However, it does not concern specific values of X but rather, general properties such as whether M follows a distribution in relation to the values of X, or whether a timeseries trend exists in cases where X describes a time attribute.
    - **Three-dimensional highlight-related patterns**. This pattern type consists of a measurement column M and two regulator columns X and Y respectively. The behavior of M is examined in relation to how the values of X and Y change. We are looking for top-K values of M, outliers and X or Y values that dominate a result.

- The system should be able to retain all the results of the evaluated patterns the data set was checked against as objects during runtime, using the respective classes.

Note that since some of the added operations are performed internally within the system, as part of the data set processing, the analyst can not interact with them directly.

2.2 Background

In this section, the background required to comprehend the thesis is presented in greater detail. The state of the Pythia system prior to this thesis is also described.

2.2.1 Data sets & multi-dimensional data

Data refers to information that comes from observations or measurements which are recorded and represented as text or numbers [Wiki22]. A data set is a structured collection of information which can be manipulated by a computer. This information may originate from various sources of measurement and understanding it is essential in order to use it and extract value from it.

In the majority of cases, data sets hold multi-dimensional data, which are typically organized in tabular format, where each row corresponds to a record and each column is an attribute. The data set lists values for each of the attributes. For instance, height and width of an object, for each row of the data set. In general, attributes on columns can be distinguished between two different types: dimensions and measures. Dimensions can be categorical (e.g., “Country”) or temporal (e.g., “Month”) and they can be used to group records. As an example, one could select only the records (rows) of a given data set whose “Month” attribute has the value of “July”. On the other hand, measures are numerical columns (e.g., “Temperature”) on which certain aggregate functions can be performed such as mean, median or sum [PRSD21].

The rows and columns of an entire data set with multi-dimensional data define a space which we will call dataspace. However, on many occasions, we do not need to examine the entire dataspace but only a small fraction of it. Based on that, the data space can be divided into subspaces. Subspaces can be as small as a single data point, meaning a single attribute on a single row. On the other hand, a subspace may consist of one or more rows and one or more attribute columns.

Data sets are typically stored and maintained in databases. However, they can be individually exported for specific analysis such as in the case of a data set profiler. A

common format in which data sets can be exported is a plain text file, where each line is a record with its attributes, which are separated using a delimiter such as a tab or a comma.

2.2.2 Pythia prior to this thesis

As it was mentioned above, the Pythia[1] system, in its state prior to this thesis, is relatively limited to assessing the quality of the given data set. The end goal of the system is to facilitate fast automated EDA, by generating a valuable insights overview about a given data set. The system can be easily imported as a dependency into other Java projects as it is developed in Java too and is distributed in a JAR package. Pythia utilizes the Apache Spark engine to quickly process the data set. Apache Spark is an optimized and unified engine for executing data engineering and analytics on large scale data.

The capabilities of the system prior to this thesis are described in further detail below [Alex22]:

- **Data set loading**: The system is capable of accepting a data set via programming to declare the fields of the data set and their type respectively (integer, double, Boolean, etc.).
- **Descriptive statistics**: Any data set can be summarized using descriptive statistics and that is usually the first step towards analysis. Pythia is capable of automatically calculating mean, median, standard deviation, min, max and multitude of values for all columns, even for non-numerical data.
- **All pairs correlation**: Another important metric in data sets is the correlation between columns, as it can be useful in cases where predictive algorithms are applied. Pythia is capable of calculating the correlation between all columns of the given data set.
- **Labeling & decision trees**: Pythia allows for optional labeling rules to be applied to any desired columns. If the analyst specifies any labeling rules for a column, Pythia automatically labels its values and creates a new column with the labeled values. Subsequently, a simple decision tree is also generated in an automated manner based on the labeled column with the help of Apache Spark.
- **Reporting**: After all the data processing is complete, the analyst can export the results in plain text or JSON format. JSON (JavaScript Object Notation) is an easy-

[1] The system was named Pythia after the generic title of the high priestesses of the Temple of Apollo at Delphi, known as the Oracle of Delphi. The Pythia were renowned for their oracular capabilities, as they delivered enigmatic prophecies in a frenzied state induced by vapors rising from a chasm in the rock.

to-understand format to transfer and display data [Orac22]. It is widely used in many fields of computer science and especially in web technologies.

2.2.3 Highlight patterns

In general, the term highlight is used to describe an interesting property that is identified in a subspace of a data set. In essence, identifying a highlight is a way of extracting valuable insights from data. There are numerous types of highlights, heavily depending on what one might consider interesting. However, in the context of this thesis, we are examining highlights in the form of patterns within the data. The term "highlight patterns", or simply "highlights", is used to refer to these patterns that actually exist in the analyzed data set. Highlight patterns can be classified between two categories based on the spatial context they are identified at: holistic and point based.

- **Holistic highlight patterns**. This pattern refers to a property that corresponds to the entire subspace under examination. The dataspace is either true or false w.r.t. the highlight's occurrence.
- **Point-based highlight patterns**. This pattern refers to a highlight property that corresponds to a specific point in the subspace under examination. This data point is characterized by one or more coordinates which induce a property to the measurement.

In both cases, the subspace consists of a measurement column M and one or more coordinate columns. In general, a subspace of a data set can have multiple coordinate (or regulator) columns that make the highlight predicate true, depending on the nature of the highlight itself. In the context of this thesis, we examine holistic cases of one coordinate column X and point-based cases of one coordinate column X or two coordinate columns X and Y respectively. The coordinate column(s) induce a property to the measurement column.

According to the terminology and taxonomical clarification of MetaInsight [PRSD21], highlight patterns can be further classified. In case of holistic highlights, patterns can be distinguished between the following types:

- **Statistical highlight patterns**. In statistical highlight patterns, the data subspace is examined for general properties such as whether M follows a distribution in relation to the values of X.
- **Trend highlight patterns**. Trend is a time series highlight pattern, which means that the regulator column X describes a time attribute. This pattern describes a continuous increase or decrease over time of the measurement column M. On a

graph, the pattern usually resembles a straight line, although, special cases of trends exist, such as exponential trends.

- **Seasonality highlight patterns**. Seasonality is also a time series highlight pattern, meaning that the regulator column X describes a time attribute. The highlight presents a repeated pattern in a time series. One can identify a seasonality highlight pattern when fluctuations repeat over a relatively fixed period of time.

In case of point-based highlights, patterns can be further distinguished between the following types [PRSD21]:

- **Coordinate highlight patterns**. In coordinate highlights the behavior of the measurement column is examined in relation to how the values of the coordinate column(s) change. We are looking to identify properties such as outliers or which values of X produce the top-K values of M.
- **Change point highlight pattern**. Change point is a time series highlight pattern, which means that the regulator column X describes a time attribute. The highlight refers to a point of interest in the data subspace where a change on the measurement column M occurs. In further detail, the mean value of M before and after the highlight changes significantly.
- **Unimodality highlight pattern**. Unimodality is also a time series highlight pattern, which means that the regulator column X describes a time attribute. This highlight refers to a point of maximum or minimum value in a time series that has a peak shape or a U-shape.

2.2.4 Outliers

An outlier is a data point or observation whose value is notably different from the others in the analyzed data set [Agga15][Bosl12]. This is sometimes described as a data point that seems to come from a different source or is outside the typical pattern of the other data points. In the case of numerical data, an outlier refers to a value that is either much smaller or much larger than most of the other values of the data set. For instance, suppose we are examining the sales of different departments across different cities for a specific time period. All departments have made sales between the values of 5.000 and 6.000, with the exception of two departments that made 3.000 and 8.500 sales respectively. The last two cases are most likely considered outliers as their values are far from the other data in the sample.

Identification of outliers is important in many types of data analysis because the presence of just a few outliers can completely distort the value of some simple statistics, such as the mean. It is also important to identify outliers because sometimes they represent data entry errors in the measuring process. Once identified, it is possible to remove cases with outliers from the data before any further analysis is performed, but the acceptability of such practices varies. Alternatively, in cases where it is not possible or acceptable to remove outliers, the very knowledge of their presence can have a significant impact on the methods used for further analysis [Bosl12].

2.2.5 Clustering analysis

Clustering analysis involves a set of techniques that allow data to be grouped together into clusters in such a way that data belonging to the same cluster are more similar to each other than data belonging to another cluster [Bosl12]. Clustering analysis is performed based on the value of one or more attributes of the data set. Clustering is commonly used in various fields of computer science and very often in Exploratory Data Analysis. It is worth noting that clustering analysis itself does not refer to one specific algorithm but rather, the general task to be solved.

There are several clustering algorithms and choosing the right one for each occasion depends heavily on whether the expected number of clusters is known or not. For instance, in cases where we know the number of expected clusters, we can pass this number to the algorithm and let it take care of the allocation, such as in the case of the k-means algorithm. On the other hand, if the number of expected clusters is unknown, we may have to use a different algorithm to estimate how many clusters there are.

Clustering works by taking an input vector V with n cases and p attributes, iterating over the cases, and allocating them to one cluster based on the similarity of the case with the cluster. The criterion for assessing similarity varies among different clustering algorithms and each specific use case. A very common approach for numerical data is to measure their distance using a formula such as the Euclidean distance. Euclidean distance is the geometric distance between two points in a multidimensional space. As the algorithm execution carries on, cases are moved between clusters to minimize within-cluster variability and maximize between-cluster variability. The process continues until it converges according to some predefined criterion [Bosl12].

Clustering analysis is relatively empirical, and its outcome highly depends on the quality of the given data set. Note that in most algorithms, some randomness is introduced due to the initial assignment of the clusters. As a result of that, results are highly likely to vary

even when the selected clustering algorithm, the data set and any other predefined parameter remain the same. Therefore, if results are not satisfactory, the process of clustering may be repeated.

2.2.6 Statistical data distributions

The process of statistical analysis often involves examining the way data is distributed. The values of each column of a data set will form a distribution. Mathematically, this distribution is often described in the form of a parameterized function that can be used to calculate the probability for any individual observation in the sample subspace. This function describes the density of the observations and is known as the probability density function [Bosl12] [Mach19]. There are numerous types of distributions, with the most common one being the Gaussian distribution, often called the Normal distribution. Some common types of data distributions are described in further detail, below [Bosl12]:

- **Gaussian distribution**. The Gaussian distribution is arguably the most commonly used in statistics, mainly because it provides a reasonable description regarding how data is distributed in reality; hence, it is also known as the Normal distribution. There is an infinite number of normal distributions, all of which have the same basic shape and are described by the same two parameters: mean (μ) and standard deviation (σ).
- **Flat distribution**. The flat distribution, also known as uniform distribution, is a probability distribution where all outcomes or values within a given range are equally likely to occur. In other words, the flat distribution has a constant probability density function across its entire range. Visually, a flat distribution appears as a rectangle when represented on a graph, where the height of the rectangle corresponds to the constant probability density.

2.2.7 Data visualization

While all the data set analysis methods described above are very important, their output usually comes in numerical or tabular format, which can be tiresome for humans to go through. Presenting analysis results in a way that is meaningful, quick, and easy to understand is very important too. The most common method to achieve this – widely used in EDA – is data visualization. Data visualization is the graphical representation of information and data. It provides a quick and effective way to communicate information in a universal manner. The main goal of data visualization is to make it easier for the human brain to identify properties of the data such as patterns, trends, and outliers,

especially in larger data sets. In essence, it provides a quick and valuable insight that allows an analyst to better understand the data.

Data visualization is usually performed using some type of chart. Some of the most commonly used charts are presented in greater detail, below [Bosl12]:

- **Line chart**. Line charts or line graphs are used to display the relationship between two attributes, with one attribute depicted on the x-axis and the other on the y-axis, accordingly. The main purpose of line charts is to demonstrate the effect of the x-axis attribute on the y-axis attribute as the first one increases. One requirement for a line chart is that there can only be one x-value for each y-value. These two values make up a point in the space of the chart. Points can be connected with lines, as suggested by the name of the chart. Oftentimes, the x-axis is used to depict a time attribute and line charts are typically used to visualize trends or temporal data, such as a timeseries.
- **Bar chart**. Bar charts are particularly appropriate for displaying discrete data with only a few categories, using bars whose height and width are analogous to the attribute values they represent. The main purpose of bar charts is to allow for easy comparisons across discrete data. Bars are separated from each other, so they do not suggest continuity even if the represented attribute is continuous. Bars can be drawn horizontally or vertically. One axis is used to depict the categories being compared and the other depicts the measured value of the attribute. Bar charts can also be combined into grouped bar charts. A grouped bar chart extends the bar chart by plotting bars for the values of two or more categorical attributes instead of one. However, note that it is not a good practice to depict more than two or three categorical attributes in a grouped bar chart, as the resulting chart will end up being overcharged with the attribute data.
- **Scatter plot**. Scatter plots are used to display the relationship between pairs of attributes. A scatter plot is a graph of two continuous attributes. Similarly to line charts, one attribute value is graphed on the x-axis and the other on the y-axis. These two values make up a point in the graph, described by the x and y cartesian coordinates. Scatter plots can be useful in depicting the overall relationship between two attributes, with properties such as direction (positive or negative), and shape (linear, quadratic, etc.). On top of that, scatter plots are a good way to get the general sense of the range of data and identify any potential outliers.
- **Histogram**. Histograms are visually similar to bar charts. However, in a histogram the bars - which are also called bins in histograms - are used to depict the distribution of data. In other words, the range of attribute values is divided into

groups. Each group is depicted as a bar. The height of the bar is proportionate to the amount of attribute values it contains. The width of the bars is usually identical and contrary to bar charts, bars in histograms touch each other.

2.2.8 Apache spark

Apache Spark [Amaz23] [Apac21] is an optimized and unified engine and a set of libraries for executing distributed data engineering and analytics on large scale data. It was developed at the University of California, Berkeley, and is currently maintained by the Apache Software Foundation. Spark is one of the most actively developed open-source engines for complex large-scale data analytics and is widely used in the general field of data science as it is capable of rapidly processing vast amounts of data. Apache Spark is efficient due to the fact that it utilizes in-memory caching, it has optimized query execution for fast queries against data of any size, and it can also distribute data processing tasks across multiple computers.

Spark Core is the foundation of the engine and the respective libraries. It has multiple responsibilities such as memory management, fault recovery, task distribution, monitoring, and storage system interactions. It integrates libraries for interactive queries (Spark SQL), machine learning algorithms (MLlib), real-time streaming analytics (Spark Streaming) and distributed graph processing (GraphX). Spark Core is exposed in the form of an application programming interface (API) for the following programming languages: Java, Scala, Python and R. This API hides the complexity of the engine's internal tasks and provides high-level operators, making it easy to execute complex distributed data engineering and analytics with few lines of code.

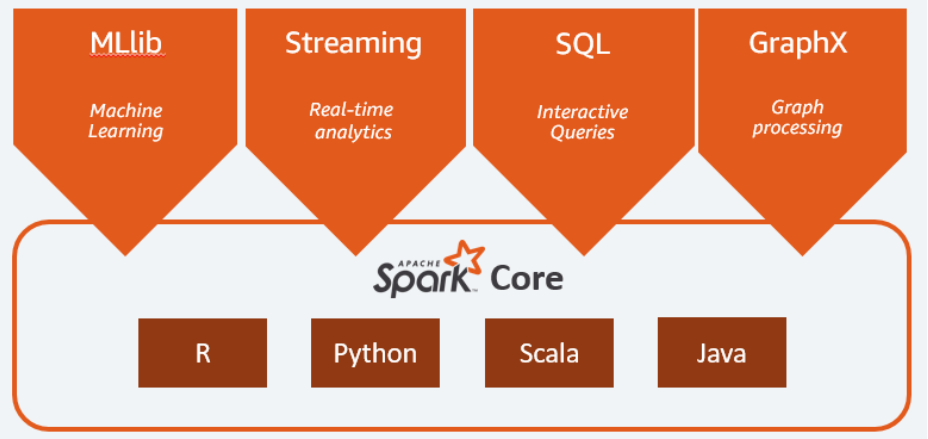

Figure 1. Spark Core integrated libraries and the provided APIs. [Amaz23]

Some of the libraries and classes of Apache Spark which are particularly used in the Pythia system are described in further detail below:

- **Spark SQL**. Apache Spark SQL is a library in the Apache Spark engine that provides a programming interface for working with structured and semi-structured data using SQL-like syntax. It allows users to read, write and load data from a wide range of sources including data files of various formats (such as CSV, TSV and JSON), external relational database management systems as well as streaming data sources. With Spark SQL, users can perform SQL queries via programming in order to filter, aggregate and transform data. Overall, it is a powerful library for analyzing and manipulating large-scale data in a distributed computing environment.
- **MLlib**. Apache Spark MLlib is also a library that is part of the Apache Spark system. It provides a scalable and distributed interface for implementing machine learning algorithms on large amounts of data. MLlib includes numerous algorithms for common machine learning tasks, such as regression, classification, clustering, and pattern mining. These algorithms are designed to work seamlessly with Spark's distributed computing, making them very efficient at large-scale data processing. Some of the algorithms available in MLlib include linear regression, logistic regression, decision trees and k-means clustering. Additionally, MLlib also includes tools for data preprocessing, feature extraction, and model evaluation.
- **Spark Session**. The SparkSession class is a top-level interface which provides a unified entry point to Apache Spark's features and functionalities, including Spark Streaming, MLlib and Spark SQL. It is the starting point for any Spark application and provides a way to configure Spark settings and access resources. SparkSession is designed to support top-level operations on structured data and is the recommended way to work with Apache Spark in most cases, as it encapsulates other high-level classes such as SparkContext and SQLContext.
- **Resilient Distributed Dataset**. The Apache Spark Resilient Distributed Dataset (RDD) class is the fundamental data abstraction in Spark. It represents a distributed collection of objects that can be processed in parallel across a cluster of machines. RDDs can be created by reading data from a file system or a distributed storage system. They can also be created as a result of data manipulation operations such as map, filter and reduce. RDDs are fault tolerant, meaning that if a node in the cluster fails, Spark can recover the lost data and proceed with data processing. Moreover, RDDs are designed to be stored in main

memory and their operations are evaluated lazily, meaning that Spark will not execute any computation until it is necessary.

- **Spark DataFrame**. The Apache Spark DataFrame class is used to describe a distributed collection of data into named columns, similar to a table in a relational database schema. It is a higher-level abstraction from RDDs. Each column has a specific data type such as integer, string, double, boolean, timestamp, etc. This structured data representation allows the DataFrame class to provide a rich set of functions, including filtering, aggregating, joining, and transforming operations. DataFrames are optimized for performance and integrate seamlessly with other Spark libraries and components, making it easy to build data processing pipelines.
- **Spark Dataset**. The Apache Spark Dataset class is the latest class which describes a strongly typed distributed collection of data, starting from Spark 1.6, integrating the benefits of RDDs, such as lambda functions. Contrary to the DataFrame class, a Dataset is a collection of strongly typed JVM objects, which provides type safety at compile time and allows for fewer runtime errors. As a result, Datasets are supported only in Java and Scala which are type safe programming languages. Operations of the Dataset class can be distinguished between actions and transformations. Actions refer to operations such as count, show and write, while transformations are operations such as select, map and aggregate and they produce new Datasets. Dataset is the class used to represent data collections in the Pythia system. Lastly, it should be noted that Spark provides functions to easily switch between Datasets, DataFrames and RDDs.
- **Spark Column**. The Apache Spark Column class is a representation of a column of data in a Spark DataFrame. It provides a set of functions that can be used to manipulate and transform data in a column, such as selecting a subset of rows, computing aggregate functions or user-defined functions.
- Spark Row. The Apache Spark Row class is responsible for describing a row of data in a Spark DataFrame. It is a general object, containing one or more Spark Column objects, in an array-like syntax and allows for each Column to have a different data type. Similarly to Spark Column, the Row class can be used to perform operations like filtering and mapping on individual rows.

2.2.9 Git & GitHub

Git [ChSt22] is version control system (VCS) that allows multiple developers to easily collaborate in parallel on a project and track changes to the source code. It was created by Linus Torvalds in 2005 and has become one of the most popular VCS used nowadays.

Git tracks changes to a project by creating snapshots of the source code which are called commits. Each commit records changes made to the code since the previous commit, including any file additions, file modifications or file deletions. Git also allows developers to create branches, which are independent copies of the source code, that can be worked on independently without affecting the main branch. Branches can be merged back into the main source code once changes have been reviewed.

All this information is stored in a repository by Git. A repository is simply a directory that contains the source code, the commits, the branches, and various other metadata. Repositories can be stored on remote cloud servers such as GitHub and GitLab, allowing developers to easily collaborate regardless of their physical location.

Git and GitHub, as well as the above procedures and methodologies were heavily used in the development of Pythia. The above information may not be required in order to comprehend the features of the system, but it would be extremely important for those interested in the development process.

Chapter 3. Design & Implementation

This chapter presents a higher-level overview of the overall software architecture and execution flow of the Pythia dataset profiling system as well as an in-depth description of the software design and implementation regarding the automated highlight identification problem. All of the above is accompanied by UML class and package diagrams in order to assist with comprehension of the software. Details regarding software testing, installation and execution are also provided. The last section provides information about maintenance and scalability of the software.

3.1 Problem definition & resolution

As mentioned in previous chapters, lack of automation in EDA is a severe issue because it makes the process tiresome and time-consuming, and therefore hinders the main objective of data analysis, which is to extract valuable insights from data. Prior to the contributions of this thesis, the Pythia system was still in an early stage of development, limited to automatically assessing the quality of the given data set by generating descriptive statistics, correlations, and decision trees for columns of the dataset. In essence, the Pythia system set the foundation for a highly extendable system that facilitates automated EDA, producing valuable insights about input datasets.

This thesis aims to extend the functionality of Pythia beyond simply assessing the quality of a given data set by enriching the generated statistical profile with valuable insights. To accomplish this, Pythia is augmented with highlight extractor modules which examine the data set for both holistic and point-based highlights.

Each highlight extractor module has its own dedicated sub-package in the software architecture of the system and consists of the following:

1. **Data preparation**. Data preparation is an optional step which takes place before the identification algorithm. It refers to procedures such as querying the data set and selecting valid and interesting columns for passing against an identification algorithm.
2. **Highlight identification algorithm**. Once data preparation is complete, the selected data is passed by the highlight identification algorithm which processes the data and generates results regarding the existence or absence of highlight patterns for the algorithm at hand. Some algorithms may have multiple variations,

in which case all the different variations are executed and produce respective results. It should be noted that in many cases the size of the results is highly likely to exceed what one might expect to be easy-to-read and therefore, some algorithms implement mechanisms such as Top-K filtering, in order to limit the number of results, keeping only the most important ones.

3. **Highlight identification result object(s)**. For each algorithm execution, respective highlight identification results are saved in a separate concrete result object. Apart from the existence or absence of a highlight, result objects may also contain information such as the location of the highlight in the data set, scoring for the involved data, which variation of the algorithm was executed, and which measurement and coordinate columns were involved. Highlight identification algorithms greatly vary from one another and therefore each algorithm has its own result class.
4. **Reporting mechanisms**. In the end, once all involved data has passed by the previous phases and the respective result objects have been generated, the highlight identification results are exported to report files, which contain all the information saved in the result objects, as well as natural language descriptions about the identified highlights.

All of the above is performed automatically by Pythia, as part of the data analysis pipeline.

Furthermore, in order to improve the usability, extensibility and scalability of the system, Pythia is augmented with the ability to accept input parameters. After loading a data set into the system, analysts can declare parameters to specify which parts of the automated data analysis pipeline should be executed, as well as specify points of interest, such as columns of the data set, that should be involved specifically in highlight pattern identifications.

Overall, this thesis aims to extend Pythia and enhance its ability to provide valuable insights to analysts by setting the foundation for highlights identification in an automated manner. The developed highlight extractor modules implement the parameterized Factory [Knoe01] software design pattern, making them easily extendable and allowing for additional modules to be developed without any significant maintenance of the pattern manager class. Following the conventions of the existing project, the contributed code is developed in Java and utilizes the Apache Spark engine for rapid data processing. The Pythia system can be easily imported into other Java projects, as it is distributed in a JAR package.

In essence, the contributed code builds upon the following user story:

- As a data analyst, the system must provide me with the ability to automatically calculate the statistical profile for a data set such that I can easily extract its statistical properties [Alex22].

Prior to describing the software design and architecture in detail, the contributed features to the system are listed below in the form of user stories:

- [US1] As a data analyst, the system must provide me with the ability to declare parameters regarding which parts of the overall automated data analysis pipeline should be executed such that I can exclude or include each part of the data analysis pipeline individually.
- [US2] As a data analyst, the system must provide me with the ability to automatically examine the data set for high and low dominance highlights, using a measurement column and one or two coordinate columns, such that I can easily gain insights regarding which attribute values dominate the other values on the given measurement column.
- [US3] As a data analyst, the system must provide me with the ability to choose among different modes regarding the automatic selection of columns of the data set that will be involved in dominance examination such that I can control the extensiveness of the dominance examination to be performed.
- [US4] As a data analyst, the system must provide me with the ability to manually declare specific measurement and coordinate columns of the data set that should be involved in dominance examination such that I can ensure that dominance examination will be performed for the manually declared columns.
- [US5] As a data analyst, the system must provide me with the ability to automatically examine the numerical columns of the data set and detect outliers such that I can easily obtain an overview regarding outlier existence or absence.

It should be noted that some of the added features are performed internally within the system, as part of the data set analysis, and thus, it is not feasible for the analyst to interact with them directly. These features are performed automatically once a data set has been loaded and its profile is requested, provided that highlight patterns analysis is included in the overall data analysis pipeline.

3.2 Software design & architecture

As mentioned in the previous sections, Pythia aims to establish the foundation for a highly extendable system that facilitates automated EDA, producing significant insights for given data sets. In order to provide a better understanding of the software architecture, this

section provides a short overview of the overall architecture of the system, before specifically describing the components involved in the automated highlight identification which were developed in the context of this thesis.

3.2.1 Overall architecture overview

The different features of the Pythia system are developed in respective classes, which are organized into separate packages. In most of the packages, a parameterized Factory [Knoe01] software design pattern is implemented. The parameterized Factory is a creational design pattern where object creation is performed based on parameters by a separate factory class. Furthermore, the parameterized Factory pattern is used in combination with an interface which is implemented by all the different classes that get instantiated by the factory class. In essence, the factory class creates objects based on one input parameter and returns them as an interface type. The interface defines a set of methods that can be used to interact with the created objects in a consistent manner, regardless of the internal logic in each specific implementation. This provides a flexible approach to creating objects, improving the overall maintainability and extensibility of the system as the code can be easily modified or extended with additional classes implementing the interface.

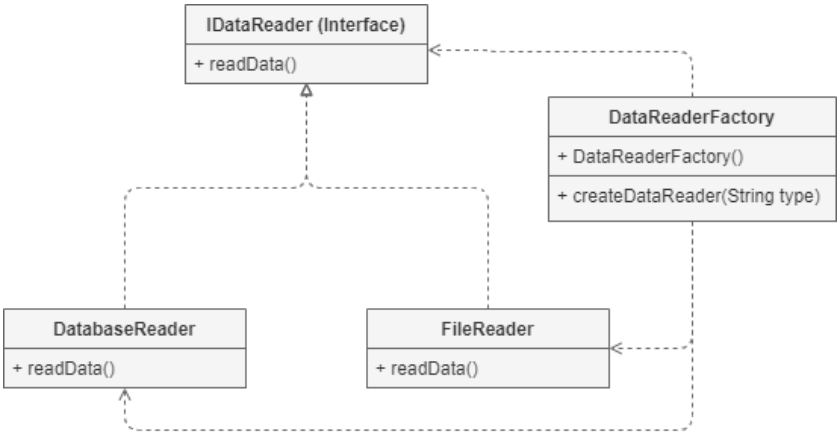

Figure 2. UML diagram with a demonstrative implementation of a parameterized Factory pattern in combination with an interface where expected behavior is defined

Figure 2 provides a complete example of an implementation of the parameterized Factory pattern in combination with an interface. Suppose we have a program that needs to read data from multiple sources, such as a file or a database. We want to abstract away the details of reading from each data source and provide a unified way of reading data from any source. To achieve this we can use a parameterized factory in combination with an interface.

In the example above, the IDataReader interface defines a method for reading data. The DataReaderFactory is the concrete factory class which creates and returns objects of type IDataReader. The "type" parameter determines which specific implementation will be instantiated. The DatabaseReader and FileReader classes implement the IDataReader interface and contain a specific implementation for reading data from their respective sources. In this way, if the program needs to read data from a different source in the future, the code can be easily extended by developing an additional reader class implementing the IDataReader interface.

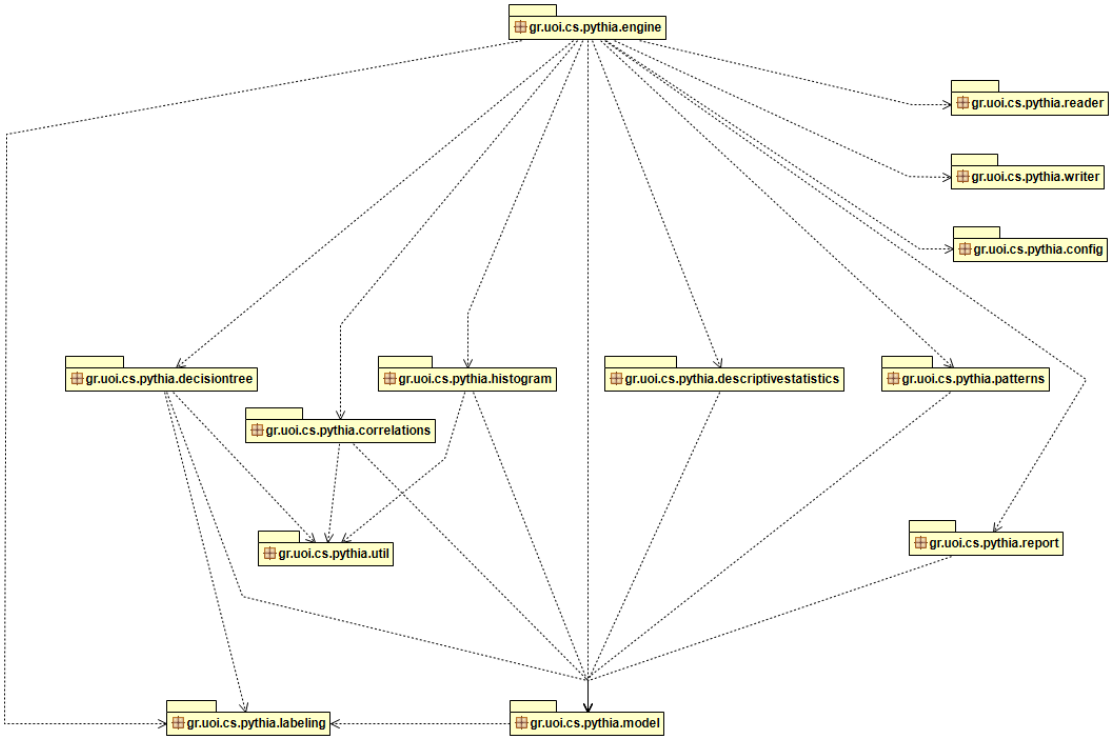

Figure 3. UML diagram with the main packages of Pythia.

A brief overview for each of the different packages of Pythia is provided below [Alex22]:

- **Engine package**. This package contains the classes responsible for the basic functionalities of the software. It is highest level package that defines and exposes the operations of the system in the form of an application programming interface (API). In essence, it provides the central control module for coordinating and managing the execution flow of the system. In the context of this thesis, this package is augmented with the capability to accept input parameters regarding which parts of the data analysis pipeline should be executed. The engine package is described in further detail in a separate sub-section below.
- **Config package**. This package contains a single class (SparkConfig) which is responsible for configuring the Apache Spark parameters required for

instantiating a Spark Session. These parameters are stored in a properties file under the src/main/resources directory.

- **Correlations package**. The correlations package has the classes responsible for the correlation calculation among all the columns of the input dataset. It implements the parameterized Factory pattern that was described above, and the main interface of the package is named ICorrelationsCalculator.
- **Labeling package**. This package contains classes responsible for creating labeled columns based on a user defined rule. The Rule class is responsible for creating a single rule for an existing column of the dataset while the RuleSet class is responsible for the creation of a complete expression based on which a new labeled column is generated.
- **Model package**. The model package contains the domain classes required for storing data that is used or produced by the system. The top-level domain class is called DatasetProfile which encapsulates various other domain classes of the package that correspond to the different data analysis parts. In the context of this thesis, this package is augmented with the PatternsProfile class which is responsible for storing all the data regarding highlight patterns identification.
- **Reader package**. This package contains the classes responsible for reading the input datasets and loading them into the system. There is a central interface named IDatasetReader which is implemented by different concrete classes such that multiple file types are supported for loading. Currently, the system is capable of loading datasets in JSON, CSV and TSV formats.
- **Report package**. This package contains the classes responsible for the creation of the report containing all the analysis results, such as descriptive statistics, decision trees and highlight patterns. In the context of this thesis, this package was modified such that each highlight pattern result is exported to a different report file. All report files are exported under the same directory.
- **Util package**. The util package has classes containing utility-methods which are required and used in different parts of the software.
- **Writer package**. The writer package contains classes responsible for exporting the loaded dataset back to the disk, in cases where the analyst wants to save a modified dataset for future use. The package has a main interface called IDatasetWriter which is implemented by the following two concrete classes. The NaiveDatasetWriter class which is a simplistic line-by-line writer approach that can be used for smaller data sets and the HadoopDatasetWriter class which is

powerful writer utilizing Hadoop Distributed File System (HDFS), capable of rapidly exporting large datasets to disk.

- **Descriptive statistics package**. This package contains the classes responsible for the calculation of descriptive statistics. The main interface is called IDescriptiveStatistics and supports automated calculation for mean, median, standard deviation, min, max and multitude of values for all columns of the dataset, even for non-numerical data.
- **Decision tree package**. This package encapsulates the classes and sub-packages responsible for the extraction and the visualization of decision trees. The top-level manager class is called DecisionTreeManager and there are classes regarding input parameters, data preparation as well as generation and visualization of decision trees. The package is described in great detail in the "Automated Extraction of Decision Trees in a Data Profiling System" [Char23] thesis.
- **Histograms package**. This package contains all classes responsible for the calculation of histograms for the numerical columns of the data set. The top-level manager class is called HistogramManager. This package is also described in great detail in the "Automated Extraction of Decision Trees in a Data Profiling System" [Char23] thesis.
- **Patterns package**. This package encapsulates the classes responsible for the highlight patterns identification. It also implements the parameterized Factory design pattern and has a main engine interface named IPatternManager. This is the main package that was developed in the context of this thesis and is thoroughly described in a separate subsection below.

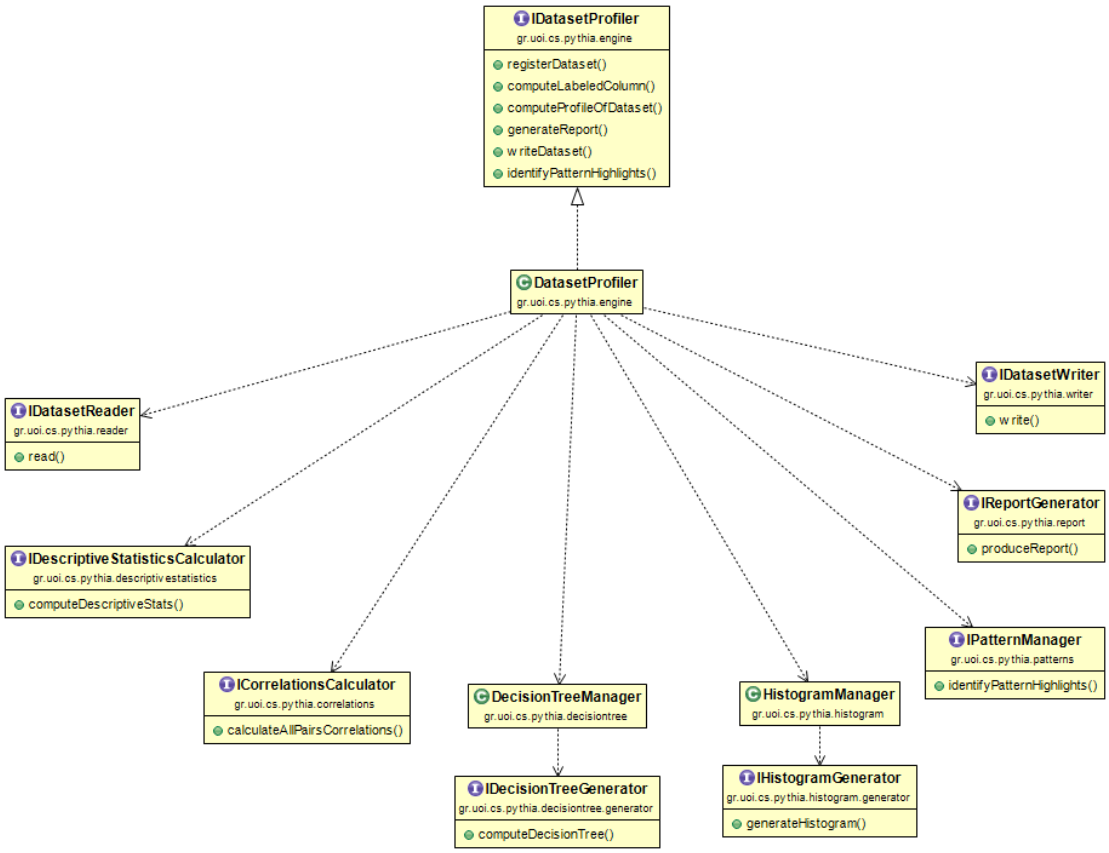

Figure 4. UML diagram with the main interfaces of Pythia.

3.2.2 Execution flow

The operations of Pythia are exposed as methods in the IDatasetProfiler interface of the engine package. In general, the basic way of working with the Pythia system to analyze a dataset and obtain statistics and other valuable insights is as follows:

1. **Register Dataset**. The analyst must programmatically provide the file path of the dataset, the names of each column along with its data type such that the dataset is loaded into the system.
2. **Add labeled column**. Then, the analyst must specify a column name and a rule set based on which a new labeled column is generated. Labeled columns are used later on in the execution flow for analysis mechanisms such as decision trees generation. This step can be repeated multiple times.
3. **Declare dominance parameters**. Next, the analyst must specify the mode, based on which columns involved in dominance examination will be automatically selected and/or manually specify the names of coordinate and measurement columns involved in dominance examination. This step is optional; however, it is a prerequisite for highlight patterns identification. It should be noted that the order of this step is interchangeable with that of step 2.

4. **Compute dataset profile**. Once the previous steps are complete, the analysis of the loaded dataset can be performed. In the current state of the system, this step integrates the following sub-steps, executed internally as part of the automated data analysis procedure. In the context of this thesis, the analyst can pass parameters to select which of the analysis sub-steps will be executed.
    a. **Calculate descriptive statistics**. This step refers to the calculation of mean, median, standard deviation, min, max and multitude of values for all columns of the dataset, including columns with non-numerical data type.
    b. **Calculate all histograms**. In this step, the system calculates histograms for all the columns of the dataset with a numerical data type. This step is described in great detail in [Char23].
    c. **Calculate all pairs correlations**. During this step, Pythia calculates correlations among all the columns of the dataset.
    d. **Extract decision trees**. This step refers to the extraction of decision trees based on labeled columns. More specifically, for each labeled column that was specified in step 2, the system generates a respective decision tree. This step is described in great detail in [Char23].
    e. **Identify highlight patterns**. In this step, Pythia examines the dataset for highlight patterns. For each implemented pattern type, the system determines whether the whole dataspace or a subspace of the dataset will be involved and afterwards, the selected data is passed by the respective highlight identification algorithms.
5. **Generate report**. Finally, once all the sub-steps involved in the data analysis are finished, the system is capable of generating a report for each of the analysis sub-steps respectively. Each generated report consists of multiple files and is organized in a separate dedicated directory, named after the dataset name, along with a timestamp of the specific execution time.
6. **Write dataset to disk**. This step is optional. It refers to the capability of writing a modified dataset (e.g. a registered dataset with labeled columns added to it) back to the disk, such that it can be used for future analysis.

3.2.3 Engine package

As described above, the engine package is the highest-level package of the system with classes responsible for the basic functionalities of the software. It provides the central control module for coordinating and managing the execution flow of the system, ensuring

that the internal tasks are executed in a well-defined order. The package implements the parameterized Factory pattern in combination with an interface. The central interface is called IDatasetProfiler. It defines the operations of the system and exposes them in the form of an application programming interface (API) such that they can be invoked from other projects where Pythia is imported.

In the context of this thesis, this package is augmented with the DatasetProfilerParameters class which is responsible for holding input parameters regarding which parts of the data analysis pipeline should be executed. The class has an additional parameter that defines the auxiliary data output directory. Auxiliary data refers to any data that is generated during analysis of the data set, e.g. images of decision trees. The computeProfileOfDataset method of the DatasetProfile class is the main method that automatically executes the data analysis pipeline. The method is modified to accept a DatasetProfilerParameters object. It should be noted that computation of descriptive statistics is a prerequisite for highlight pattern identification. Therefore, while it is practically feasible to include descriptive statistics and exclude highlight patterns from the overall analysis pipeline, if such a scenario is encountered, descriptive statistics are calculated nevertheless before the execution proceeds with highlight patterns identification.

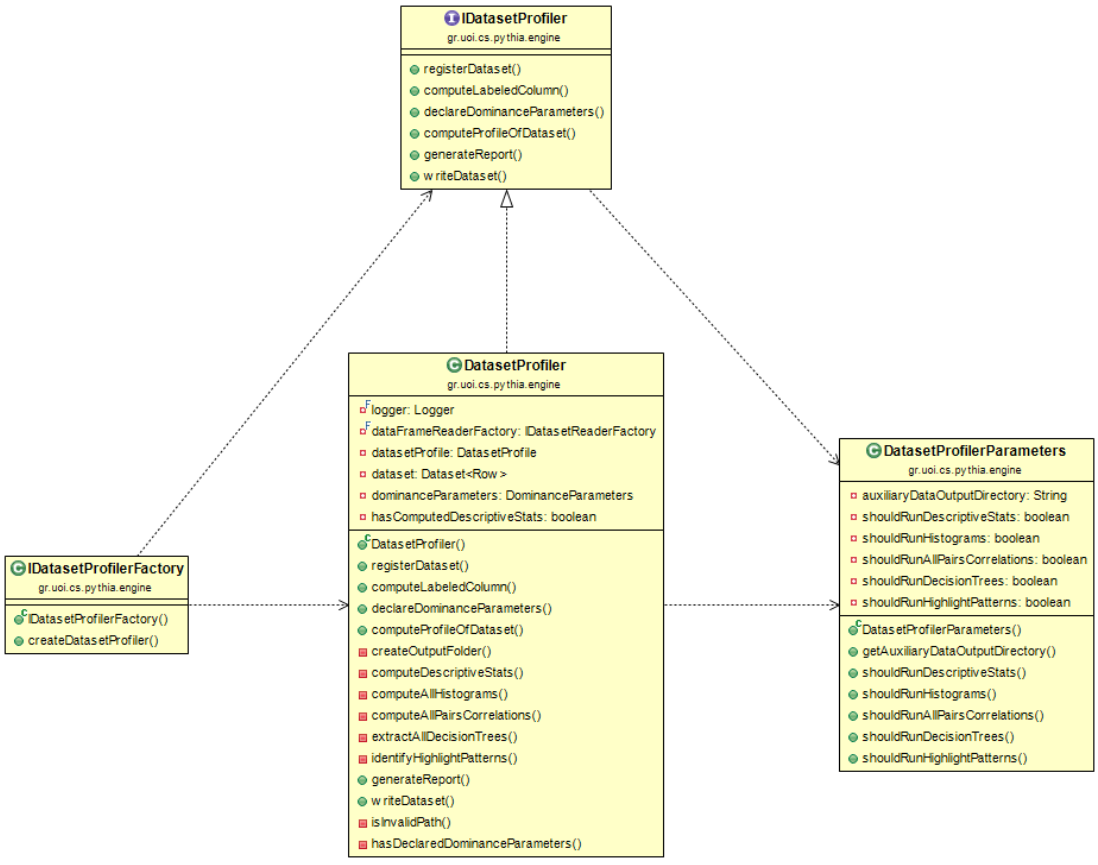

Figure 5. UML diagram with the classes of the engine package.

3.2.4 Patterns package

As mentioned above, this package encapsulates the classes responsible for the implementation of the highlight patterns identification features. The package itself is internally organized, distributing most of its classes to sub-packages, thus, forming its own internal architecture. This is the main package developed in the context of this thesis.

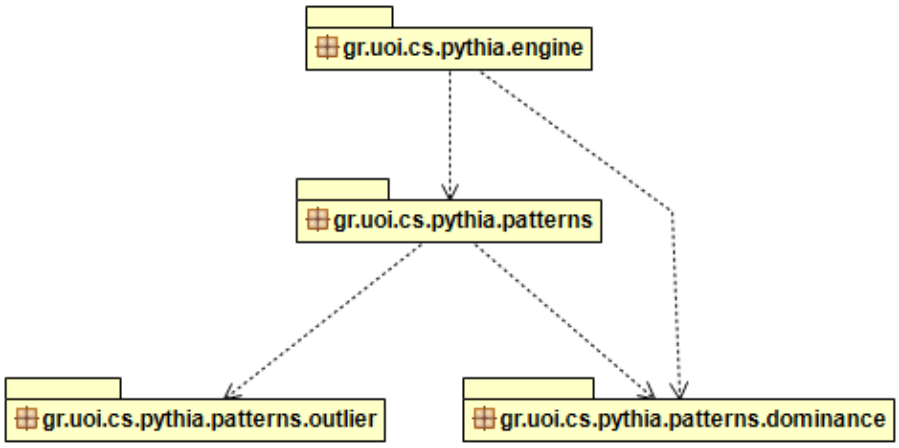

Figure 6. UML diagram with the sub-packages in the patterns package.

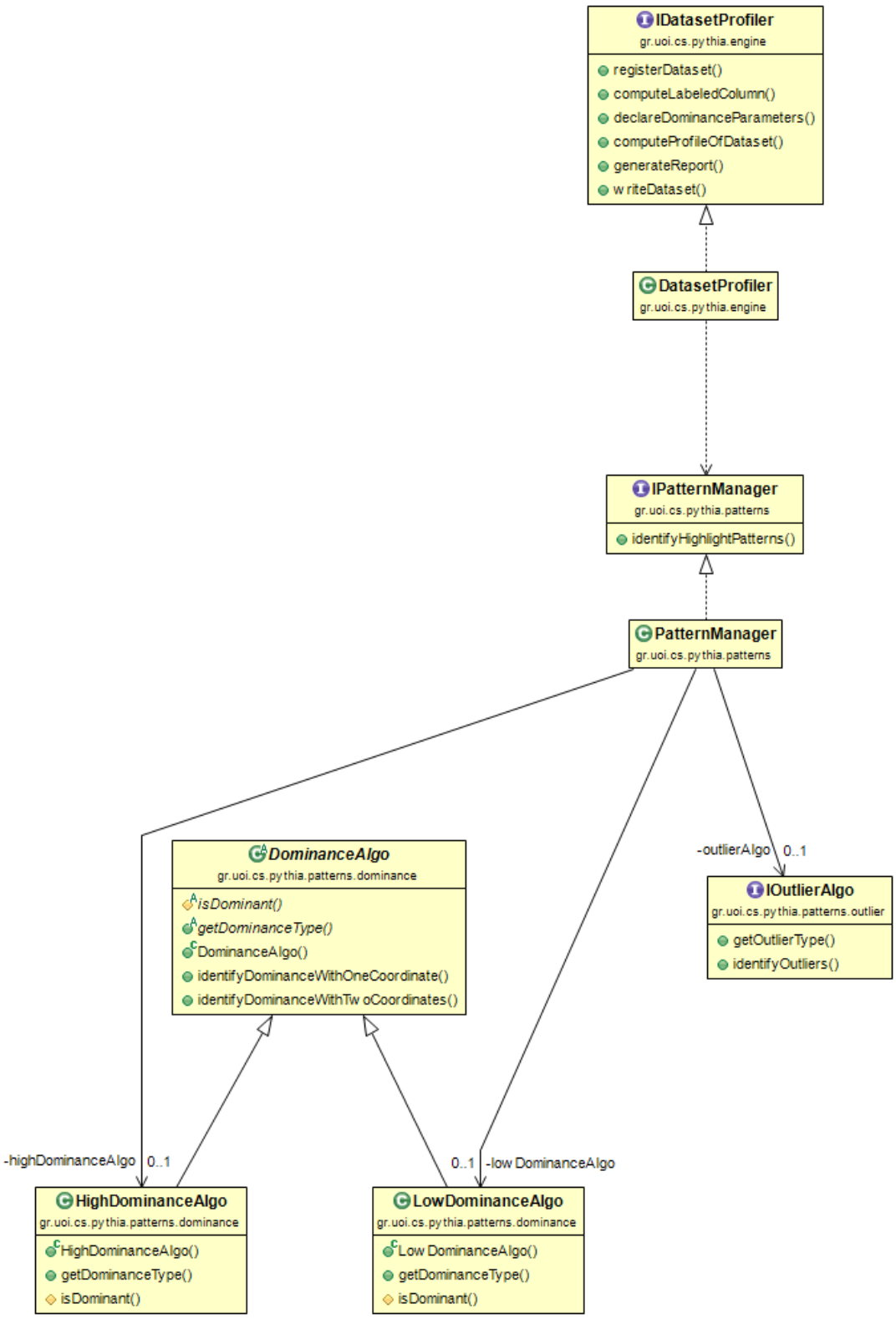

Figure 7. UML diagram with the interfaces and abstract classes within the patterns package.

The patterns package contains the central control and management interface called IPatternManager which is developed using the Factory design pattern. Currently, there is only one class implementing the interface, named PatternManager. The pattern manager interface is responsible for handling the overall highlight pattern identification procedure

at a higher level and includes information such as which pattern types are supported by the system.

The identifyHighlightPatterns method is the main method regarding highlight pattern identification in Pythia. Internally, the method calls a dedicated method for each supported pattern, where data preparation, such as measurement and coordinate column selection, might be performed. Depending on the pattern, the selected data or the entire dataset is then passed by the respective highlight identification algorithms.

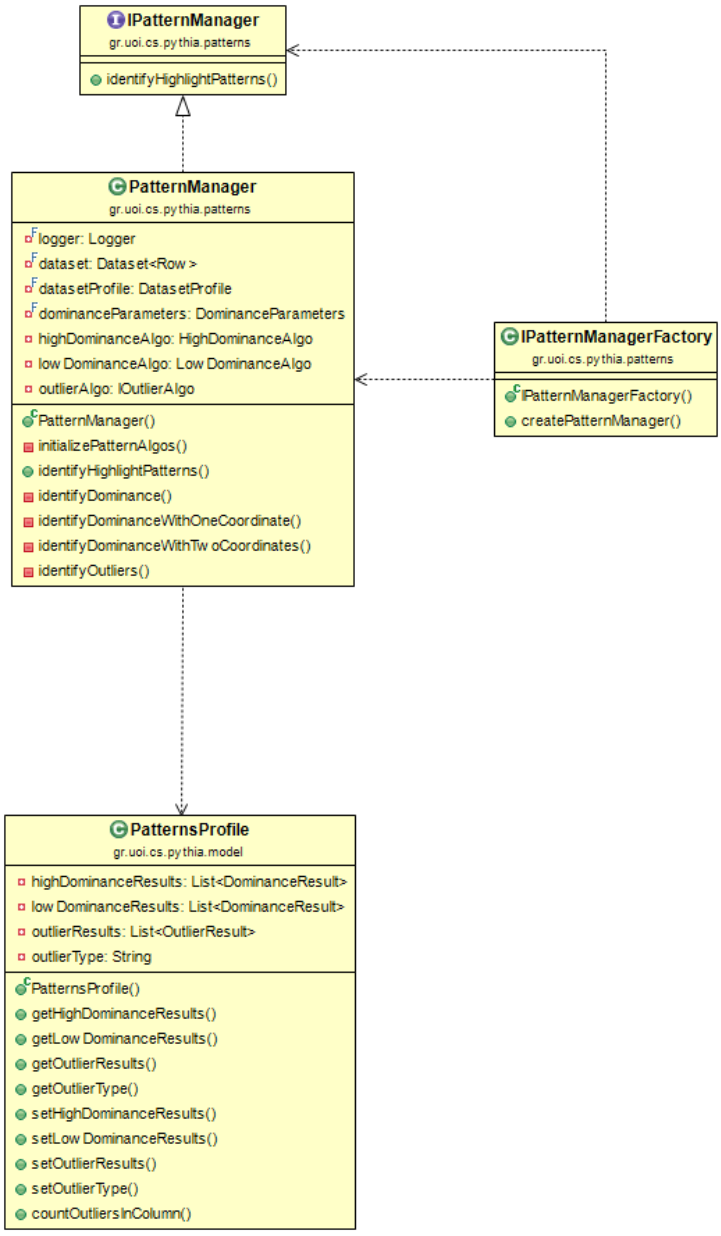

Figure 8. UML diagram with the classes directly under the patterns package.

Each pattern featured in Pythia is organized in a dedicated sub-package. Implementation of the parameterized Factory pattern that was described above is decided separately for each pattern depending on its requirements and specific features. Each sub-package

contains all classes that make up a highlight extractor module, including pattern identification algorithms, a result object, and any classes regarding parameters and data preparation, such as coordinate and measurement column selection. Some algorithm implementations may contain variations (e.g. dominance pattern algorithm) and therefore, might be implemented using multiple classes.

Each featured pattern has a respective result class that is responsible for describing highlight identification results. An algorithm execution may generate one or multiple result objects depending on the specific pattern. Result objects contain information regarding the existence or absence of highlights and additional details such as which variation of the algorithm was executed, the location of any identified highlight within the data, which columns were involved and any scoring for the algorithm at hand. The pattern manager class is responsible for adding all the generated results to respective lists of result objects in the PatternsProfile model class. In this manner, results of all pattern identifications are stored in the PatternsProfile class.

3.2.5 The dominance pattern

The dominance pattern is the first pattern that was developed for Pythia. In the context of this thesis, the dominance algorithms are developed under the homonymous dominance sub-package and implement four separate variants:

a) High dominance with one (1) coordinate.
b) Low dominance with one (1) coordinate.
c) High dominance with two (2) coordinates.
d) Low dominance with two (2) coordinates.

It should be noted that all variants involve one measurement column. In essence, a dominance highlight identification allows analysts to identify partial or total dominance occurrences for specific coordinate value(s) against the other coordinate values w.r.t. the given measurement column.

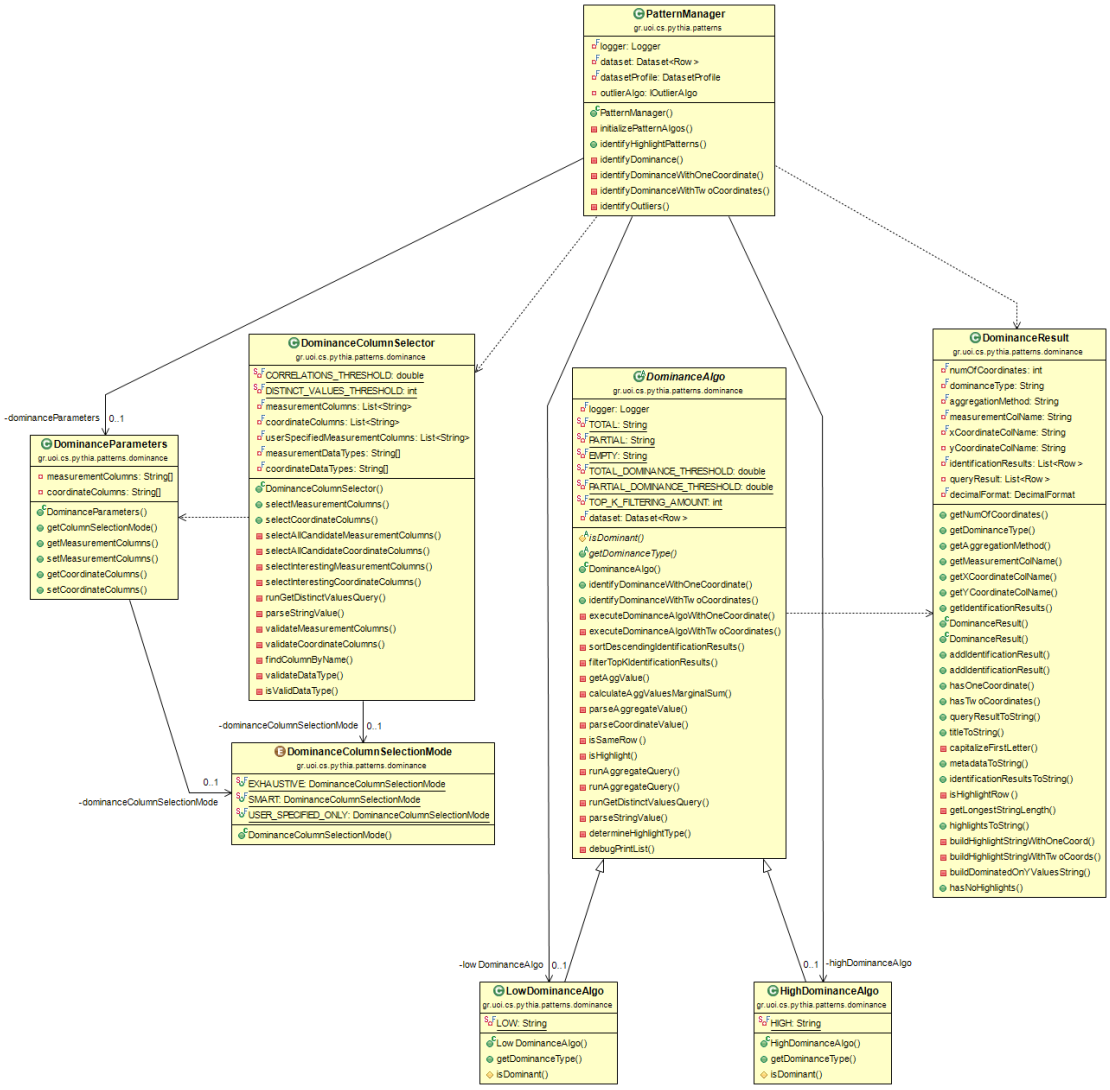

Figure 9. UML diagram with the classes of the dominance sub-package.

Suppose we have registered a dataset into the system which contains information regarding the prices of different used cars. Among the columns of the dataset there is a numerical measurement column named “price”, a coordinate column named “car model” and a second coordinate column named “year”. A dominance highlight with one coordinate may generate insights such as:

- Car model “A” has an aggregate (sum) value of 2865320 (price) and a partial high dominance of 87.5% over the aggregate values of the other car models.

In a similar manner, a dominance highlight with two coordinates may generate insights such as:

- Car model “A” presents a total high dominance on the price measurement over the car models “B”, “C” and “D”. In detail, the aggregate values of “A” dominate the car models: “B” on the year(s): 2014, 2015 and 2016. “C” on the year(s): 2014 and

2015. “D” on the year(s): 2016, 2017 and 2019. Overall, “A” has a dominance percentage score of 100%.

The first step in dominance highlights identification is the selection of measurement and coordinate columns to be involved. For this purpose, the IDatasetProfiler interface (refer to Figure 5) is augmented with the ability to accept parameters regarding dominance identification. In cases where dominance parameters are not specified, highlight patterns identification is not initiated even if requested. The DominanceParameters class is responsible for storing the parameters and the DominanceColumnSelector is responsible for examining the columns of the dataset regarding their validity as measurement or coordinate columns based on the specified parameters. In detail, the analyst can choose among the following column selection modes:

- **Exhaustive mode.** In this mode columns are automatically examined and selected solely based on their data type. This is the mode that selects the most columns for dominance examination and therefore produces the highest volume of results.
- **Smart mode.** In this mode columns are automatically examined and selected based on descriptive statistics and correlations. Columns that qualify as measurements based on their data type but are highly correlated are pruned, such that only one of the two columns is selected for dominance check. Coordinate columns with more than 20 distinct values are also pruned. The number 20 was selected as a threshold for coordinate distinct values in an empirical manner, due to the fact that dominance identification results appeared to be rather complicated and cluttered with coordinate with more than 20 distinct values.
- **User specified only mode.** In this mode the names of columns that should be involved in dominance identification are manually declared by the analyst in the form of input dominance parameters.

It should be noted that the analyst has the ability to manually declare the names of columns involved in dominance examination regardless of the chosen column selection mode. For all manually declared columns, the analyst is responsible for ensuring that the declared columns are of a valid data type.

Dominance column selection produces a list of measurement columns and a list of coordinate columns respectively. Afterwards, all combinations of measurement and coordinate(s) are passed by the four dominance algorithm variations. A query is executed which aggregates based on the measurement column and groups by the coordinate columns. And finally, the query results are examined for dominance existence or absence depending on the specific dominance algorithm variation. It should be noted that missing

values on the input datasets may result in missing values in the query results respectively. Such missing values are simply ignored by the algorithms.

As mentioned above dominance highlight identification refers to the identification of total or partial dominance occurrences for specific coordinate value(s) against the other coordinate values w.r.t. the given measurement column. As the name suggests, a total dominance occurrence corresponds to a case where a coordinate value dominates all the other (100%) coordinate values of the column at hand. On the other hand, partial dominance refers to a case where a coordinate value dominates only some of the other coordinate values. Suppose we define partial dominance A, where $A \in (0.5 \ldots 1.0]$. During the development of this thesis, A was set to 0.75 (75%) as a rational middle point for partial dominance.

The dominance algorithm check, with one coordinate, for both low and high dominance is described below:

1. For each measurement aggregate value M of a coordinate value X:
    a. Check whether M is higher (or lower) than the measurement aggregate values of the other coordinate values.
    b. Calculate the dominance percentage score of the coordinate value X against the other coordinate values. A dominance score of 100% indicates a total dominance highlight of X. A dominance score that is equal to or greater than 75% indicates a partial dominance highlight of X.

Similarly, the dominance algorithm check, with two coordinates, for both low and high dominance is described below:

1. For each first coordinate value X_1:
    a. For each first coordinate value X_2 ($X_1 \neq X_2$):
        i. If X_1 has a higher (or lower) measurement aggregate value than X_2 for all second coordinate values, then X_1 appears to have an upper (or lower) dominance over X_2.
    b. Calculate the dominance percentage score of coordinate value X_1 against the other coordinate values. A dominance score of 100% indicates a total dominance highlight of X_1. A dominance score that is equal to or greater than 75% indicates a partial dominance highlight of X_1.

With a quick examination of the algorithms it is easy to observe that low and high algorithms are greatly alike. Specifically, the only difference is the check for a lower aggregate value on low dominance or a higher aggregate value on high dominance. For this reason, the classes responsible for the dominance algorithm variants are

implemented using the Template Method [Refa23] software design pattern. The Template Method is a behavioral pattern that defines the overall structure of an algorithm in a base abstract class, allowing concrete subclasses to override specific steps of the algorithm. The overall algorithm structures for both dominance with one coordinate and dominance with two coordinates is defined in the DominanceAlgo abstract class while the HighDominanceAlgo and LowDominanceAlgo concrete subclasses are used to override the "isDominant" method which checks for high or low dominance respectively.

Moreover, due to the fact that large datasets can produce a big volume of results, the algorithms also perform Top-K filtering on the results. To this end, instead of keeping the dominance identification results of all coordinates, the algorithms filter the identification results such that results are retained for only the six (6) highest scoring coordinates.

Each algorithm execution generates a respective dominance result object with all the information regarding the specific dominance identification. The DominanceResult class is located under the homonymous dominance sub-package and is responsible for describing a dominance result. The class contains information regarding the involved columns, the number of coordinates, the aggregation method, dominance score and highlight existence or absence for each coordinate.

Lastly, a complete demonstration of a high dominance pattern identification with one and two coordinate columns is described in detail below. Suppose the data set depicted in Figure 10.

Car Model	Year	Price
A	2016	20000
A	2017	15000
B	2016	2000
B	2016	500
B	2017	500
C	2016	5000
C	2017	8000
D	2016	3000
D	2017	500
E	2016	4000

Figure 10. Table with a demonstrative data set containing the prices of various car models for multiple years.

The data set is registered into Pythia and is ready for analysis. A single-coordinate high-dominance pattern identification, with the "price" column as a measurement and the "car model" column as coordinate, would produce the results shown in Figure 11.

Car Model	Price (Sum)	Dominance score (%)	Is highlight?	Highlight Type
A	35000.0	100.0	true	total high
C	13000.0	75.0	true	partial high
E	4000.0	50.0	false	-
D	3500.0	25.0	false	-
B	3000.0	0.0	false	-

Figure 11. Table with demonstrative results of a single-coordinate high-dominance identification.

As described above, the "Price (Sum)" column presents the result of the group by-aggregate query that was executed on the data set. The dominance score shows the percentage of measurement values (car models) that are dominated by each individual measurement value. A dominance score of 100% indicates a total high highlight, while a dominance score that is equal to or greater than 75% indicates a partial high highlight. The above dominance pattern identification will also generate natural language descriptions for the identified highlights as follows:

- Coordinate: A (car model) has an aggregate (sum) value of 35000 (price) and a total high dominance of 100% over the aggregate values of the other car models.
- Coordinate: C (car model) has an aggregate (sum) value of 13000 (price) and a partial high dominance of 75% over the aggregate values of the other car models.

In a similar manner, a double-coordinate high-dominance pattern identification, with the "price" column as a measurement and the "car model" and "year" columns as first and second coordinates respectively, would produce the results shown in Figure 12.

Car Model	Dominates the car model(s)	Dominance score (%)	Is highlight?	Highlight Type	Aggregate Marginal Sum (Price)
A	B, C, D, E	100.0	true	total high	35000.0
C	B, D, E	75.0	true	partial high	13000.0
E	B, D	50.0	false	-	4000.0
B	-	0.0	false	-	3000.0
D	-	0.0	false	-	3500.0

Figure 12. Table with demonstrative results of a double-coordinate high-dominance identification.

Double coordinate dominance checks produce a list of dominated measurement values (car models) for each measurement value of the data set under the column named as "Dominates the car model(s)". The dominance score is calculated as described above for the single coordinate dominance pattern identification. The generated columns "Is highlight?" and "Highlight Type" also follow the same principles as described above. Furthermore, the last column of the results contains the aggregate marginal sum of each measurement value for both coordinates. In essence, the aggregate marginal sum is the

same as the result of the group-by aggregate query executed on single coordinate dominance.

Lastly, the high dominance pattern identification with two coordinates, will also generate natural language descriptions for the identified highlights as follows:

- Coordinate “A” (car model) presents a total high dominance over the car models: “B”, “C”, ”D”, ”E”. In detail, the aggregate values of “A” dominate the car models:
    - ”B”, ”C”, ”D” on the year(s): 2016, 2017.
    - ”E” on the year(s): 2016.

  Overall, ”A” has a dominance percentage score of 100% and an aggregate marginal sum of 35000 (price).
- Coordinate “C” (car model) presents a partial high dominance over the car models: ”B”, ”D”, ”E”. In detail, the aggregate values of ”C” dominate the car models:
    - ”B”, ”D” on the year(s): 2016, 2017.
    - ”E” on the year(s): 2016.

  Overall, ”C” has a dominance percentage score of 75% and an aggregate marginal sum of 13000 (price).

3.2.6 The outlier pattern

The outlier pattern is the second pattern that was developed for Pythia. It is developed under the homonymous outlier sub-package. Due to the fact that there are numerous outlier detection algorithms, the package implements the parameterized Factory pattern along with an interface, such that additional outlier algorithms can be easily added into the system. The main interface is called IOutlierAlgo that defines a single method responsible for outlier detection, as well as a simple method that returns the type of the concrete algorithm class implementing the interface. Any data preparation, such as examination of the columns regarding their validity for outlier detection is performed separately -within the detection method- for each specific outlier algorithm implementation.

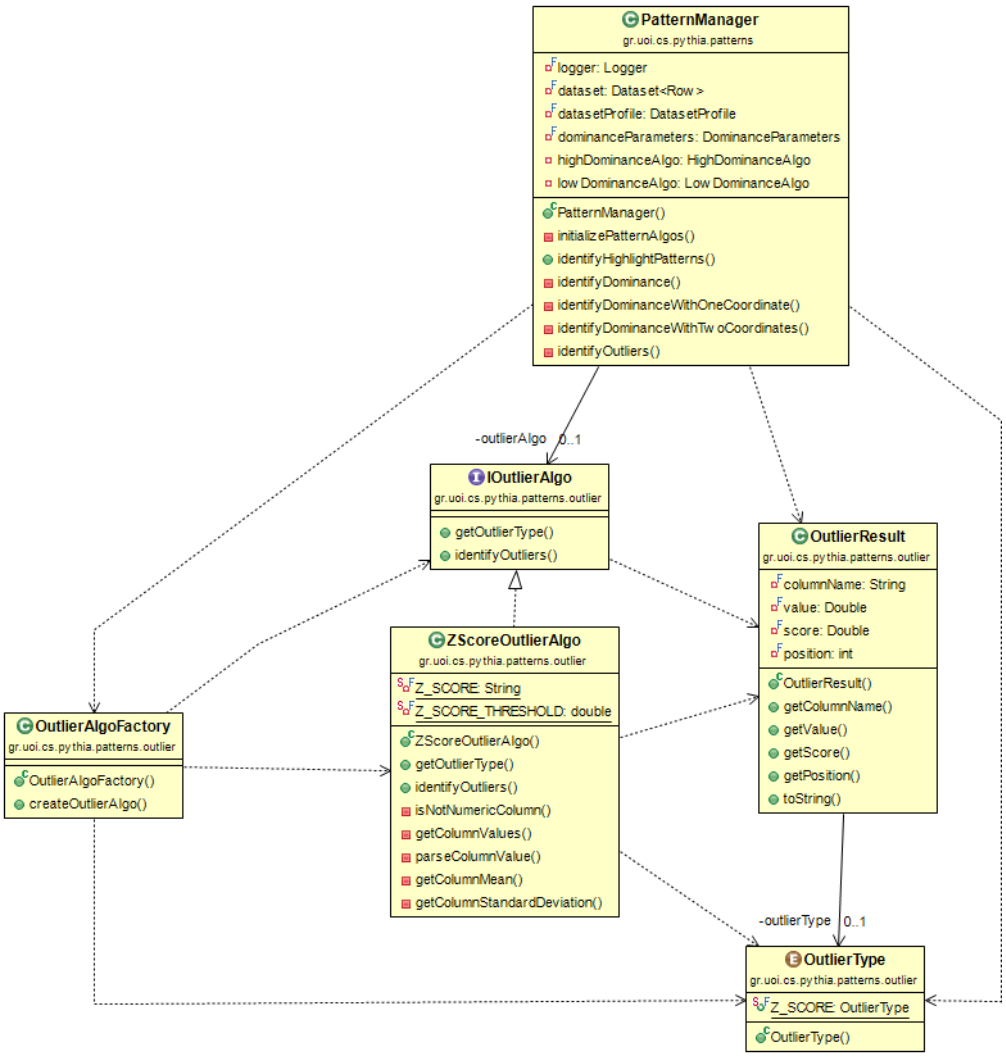

Figure 13. UML diagram with the classes of the outlier sub-package.

In the context of this thesis, the implemented algorithm utilizes the Z-Score outlier detection method [Bosl12]. The Z-Score method is a statistical approach for detecting outliers in the numerical columns of a data set. It works by calculating the Z-Score for each data point, which measures how many standard deviations a data point is from the mean of the column at hand of the data set. A Z-Score equal to or higher than 3 (or equal to or lower than -3 respectively) indicates that a given data point is an outlier. The Z-score method is a simple, yet effective way to detect outliers in a data set. However, it assumes that the data follow the Gaussian distribution and is highly likely to not be as effective for data with non-Gaussian distributions. The formula for Z-Score calculation is presented below in Figure 14.

$$Z = \frac{x - \mu}{\sigma}$$

Where x refers to the value of a data point, μ refers to the mean of the specific column of the data set, and σ is the standard deviation of the specific column of the data set.

Figure 14. Formula for Z-Score calculation [Bosl12].

Z-Score outlier detection in Pythia is performed in the ZScoreOutlierAlgo class and is executed as follows. Initially, the columns of the data set are examined regarding their data type. Numerical columns are selected for outlier examination. The Z-Score is calculated for each value of the selected columns, using the mean and standard deviation, that was already calculated in the descriptive statistics data analysis step (hence, descriptive statistics computation is a prerequisite for highlight patterns identification, as described above). Values with an absolute Z-Score that is equal to or higher than 3 are stored as identified outliers.

An object of the OutlierResult class is created for each identified outlier. The result object contains information such as the Z-Score, the value of the data point, the position of the data point in the column, and the column it was detected in. Once outlier examination is complete for all selected columns, a list of all the results with identified outliers is returned to the pattern manager.

3.2.7 Highlight patterns reporting

As mentioned above, each pattern featured in Pythia has a respective result class that is responsible for storing the highlight identification results. Depending on the pattern at hand, an algorithm execution may generate one or multiple result objects. More specifically, regarding the patterns developed during this thesis, the DominanceResult class is responsible for describing the result of a dominance identification and the OutlierResult class is responsible for storing information about identified outliers. Generally, result objects contain information regarding the existence or absence of highlights and additional details such as the location of any identified highlight within the dataset. During highlights identification, the generated results are returned to the PatternManager class which is responsible for adding them to respective lists of result objects in the PatternsProfile model class. In this manner, results of all pattern identifications are stored in the PatternsProfile class.

Once the procedure of highlights identification has completed for all patterns, Pythia is capable of generating reports with all the information of the performed identifications. Report generation is performed in the homonymous report package. In its current state,

the system supports reporting in plain text and markdown format. As far as dominance results are concerned, the two report types have been separated due to the large volume of dominance results. The report types haven been separated into:

a. An extensive txt report.
b. A concise markdown report.

The concise markdown report only contains results that have been identified as actual dominance highlights, along with scoring, metadata, and a natural language description for each highlight. The extensive txt report contains all the above but might also contain results that have not been identified as highlights. However, the number of results is limited by the top-K filtering threshold. The extensive txt report also contains the results of the query that was executed on the dataset for dominance identification, although, that is mainly for debugging purposes. On the contrary, outlier identification reports contain the same information in both txt and markdown, regarding outlier existence or absence in each column of the input dataset.

The reports are generated by the MdReportGenerator and TxtReportGenerator classes respectively as shown in the UML diagram in Figure 15. The two report classes are implementations of the IReportGenerator interface, which is the interface that describes report generation. The classes are responsible for acquiring the related result data from the result objects via the PatternsProfile class. The main method responsible for the generation of all reports is produceReport, which internally calls a dedicated method for the generation of the overall statistical report and another method for the generation of highlight pattern related reports.

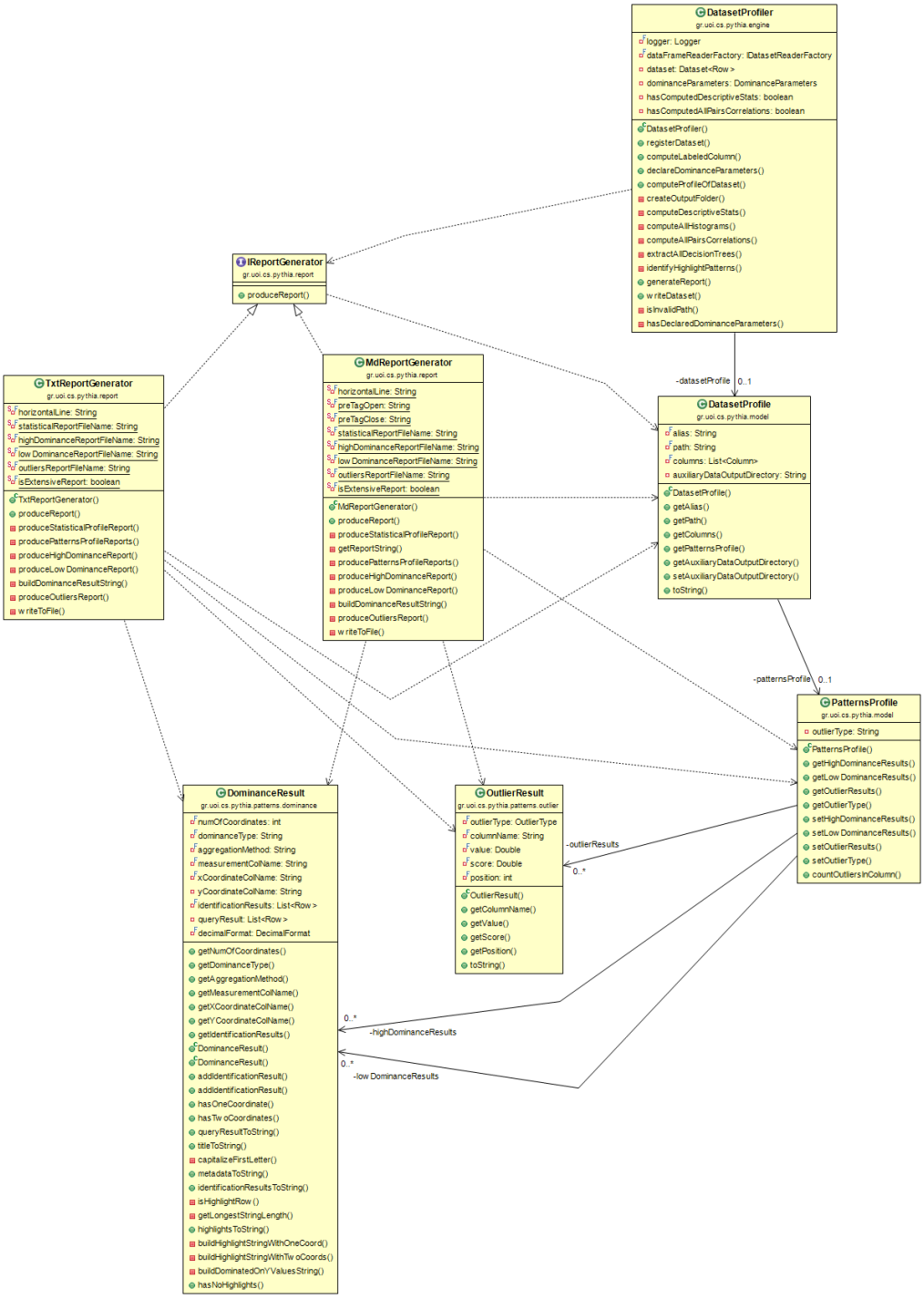

Figure 15. UML diagram with the classes involved in highlights report generation.

3.3 Software tests design & results

Prior to this thesis, Pythia was already equipped with test cases that covered the entirety of the functionalities of the system. To test the newly contributed features to Pythia, the existing conventions and methodologies for the development of the system were used. Namely, testing was achieved by developing unit tests that utilize the black box testing method [Alex22].

Black box testing is a software testing method that focuses on testing the functionality of a system without knowledge of its internal implementation details. In the context of unit testing, black box testing involves creating test cases that only consider the inputs and outputs of a unit of code -oftentimes a method- and do not take into account how the code is implemented. Input is entered into the code unit under test and the system is expected to generate a specific predefined output. If the system generates the expected output, then the test is successful, otherwise the test is a failure. In this process, the system itself is treated as a "black box".

Following the existing dependencies of the system, tests were developed using the JUnit 4 framework [JUni21]. JUnit is an open-source programmer-oriented testing framework for Java. It is designed to help developers with writing and running unit tests. It is based on Java annotations, which are special markers added before test methods, that provide numerous features, including test preparation, test runners and output assertion.

Regarding testing of the features that were contributed to the system during this thesis, the developed test cases are described below throughout Figures 15-31.

Test Case	**Correct execution test for single-coordinate high dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithOneCoordinate
Class under test	HighDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, and a coordinate column name.
Output	A DominanceResult object with the identification results.
Assert that	The generated DominanceResult object is equal with the expected DominanceResult object.

Figure 16. Description of the unit test regarding correct execution of single-coordinate high dominance identification.

Test Case	**Correct execution test for single-coordinate low dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithOneCoordinate
Class under test	LowDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, and a coordinate column name.
Output	A DominanceResult object with the identification results.
Assert that	The generated DominanceResult object is equal with the expected DominanceResult object.

Figure 17. Description of the unit test regarding correct execution of single-coordinate low dominance identification.

Test Case	**Correct execution test for double-coordinate high dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithTwoCoordinates
Class under test	HighDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, and two coordinate column names.
Output	A DominanceResult object with the identification results.
Assert that	The generated DominanceResult object is equal with the expected DominanceResult object.

Figure 18. Description of the unit test regarding correct execution of double-coordinate high dominance identification.

Test Case	**Correct execution test for double-coordinate low dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithTwoCoordinates
Class under test	LowDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, and two coordinate column names.
Output	A DominanceResult object with the identification results.
Assert that	The generated DominanceResult object is equal with the expected DominanceResult object.

Figure 19. Description of the unit test regarding correct execution of double-coordinate low dominance identification.

Test Case	**Invalid coordinate exception test for single-coordinate low & high dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithOneCoordinate
Class under test	HighDominanceAlgo, LowDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, and a coordinate column name that does not belong to the loaded data set.
Output	-
Assert that	AnalysisException is thrown.

Figure 20. Description of the unit test regarding invalid coordinate exception in single-coordinate low & high dominance identification.

Test Case	**Invalid coordinate exception test for double-coordinate low & high dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithTwoCoordinates
Class under test	HighDominanceAlgo, LowDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name, a coordinate column name, and a second coordinate column name that does not belong to the loaded data set.
Output	-
Assert that	AnalysisException is thrown.

Figure 21. Description of the unit test regarding invalid coordinate exception in double-coordinate low & high dominance identification.

Test Case	**Invalid measurement exception test for double-coordinate low & high dominance identification.**
Involved user story	US2
Method(s) under test	identifyDominanceWithTwoCoordinates
Class under test	HighDominanceAlgo, LowDominanceAlgo
Prerequisite condition	A data set successfully loaded into the system and measurement and coordinate columns are selected.
Input	The loaded data set, a measurement column name that does not belong to the loaded data set, and two coordinate column names.
Output	-
Assert that	AnalysisException is thrown.

Figure 22. Description of the unit test regarding invalid measurement exception in double-coordinate low & high dominance identification.

Test Case	**Correct execution test for dominance column selection with exhaustive mode.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the exhaustive mode, an empty list of measurement column names, and an empty list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected measurement column names and the selected coordinate column names are equal with the expected coordinate column names.

Figure 23. Description of the unit test regarding correct execution of dominance column selection with exhaustive mode.

Test Case	Correct execution test for dominance column selection with smart mode.
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the smart mode, an empty list of measurement column names, and an empty list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected measurement column names and the selected coordinate column names are equal with the expected coordinate column names.

Figure 24. Description of the unit test regarding correct execution of dominance column selection with smart mode.

Test Case	Correct execution test for dominance column selection with user specified only mode.
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the user specified only mode, a list of measurement column names, and a list of coordinate columns names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected (input) measurement column names and the selected coordinate column names are equal with the expected (input) coordinate column names.

Figure 25. Description of the unit test regarding correct execution of dominance column selection with user specified only mode.

Test Case	**Correct execution test for dominance column selection with user specified only mode and no columns specified.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the user specified only mode, an empty list of measurement column names, and an empty list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The list of selected measurement column names is empty, and the list of selected coordinate column names is empty.

Figure 26. Description of the unit test regarding correct execution of dominance column selection with user specified only mode and no columns specified.

Test Case	**Correct execution test for dominance column selection with exhaustive mode and partial user input.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the exhaustive mode, a list of measurement column names, and a list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected measurement column names and the selected coordinate column names are equal with the expected coordinate column names.

Figure 27. Description of the unit test regarding correct execution of dominance column selection with exhaustive mode and partial user input.

Test Case	**Invalid column data type exception test for dominance column selection.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the user specified only mode, a list of measurement column names that includes column names with an invalid data type, and a list of coordinate column names that includes column names with an invalid data type.
Output	-
Assert that	IllegalArgumentException is thrown.

Figure 28. Description of the unit test regarding invalid column data type exception in dominance column selection.

Test Case	**Invalid column names exception test for dominance column selection.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the user specified only mode, a list of measurement column names that includes column names that do not belong to the loaded data set, and a list of coordinate column names that includes column names that do not belong to the loaded data set.
Output	-
Assert that	IllegalArgumentException is thrown.

Figure 29. Description of the unit test regarding invalid column names exception in dominance column selection.

Test Case	**Null column names lists safety test for dominance column selection.**
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, an enum value corresponding to the exhaustive mode, a null value instead of a list of measurement column names, and a null value instead of a list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected measurement column names and the selected coordinate column names are equal with the expected coordinate column names.

Figure 30. Description of the unit test regarding null safety for null column names lists in dominance column selection.

Test Case	Null safety test for dominance column selection.
Involved user story	US3, US4
Method(s) under test	selectMeasurementColumns, selectCoordinateColumns
Class under test	DominanceColumnSelector
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile object of the data set that is loaded into the system, a null value instead of an enum value that corresponds to the column selection mode, a null value instead of a list of measurement column names, and a null value instead of a list of coordinate column names.
Output	A list with the selected measurement column names, and a list with the selected coordinate column names.
Assert that	The selected measurement column names are equal with the expected measurement column names and the selected coordinate column names are equal with the expected coordinate column names.

Figure 31. Description of the unit test regarding overall null safety in dominance column selection.

Test Case	Correct execution test for Z-Score outlier identification.
Involved user story	US5
Method(s) under test	identifyOutliers
Class under test	ZScoreOutlierAlgo
Prerequisite condition	A data set successfully loaded into the system.
Input	The DatasetProfile and Dataset objects of the data set that is loaded into the system.
Output	An OutlierResult object with the identification results.
Assert that	The generated OutlierResult object is equal with the expected OutlierResult object.

Figure 32. Description of the unit test regarding correct execution of Z-Score outlier identification.

3.4 Installation & implementation details

This section provides information about the specifications of Pythia, such as the development tools and technologies that were used, as well as installation instructions for setting up the system as a developer.

A brief overview for each of the different technologies used in the development of Pythia is provided below:

- **Git & GitHub**. As described above, the system was developed using Git [ChSt22] and GitHub for version control. The system is open-source, and the source code can be found at the GitHub repository link referenced at [DAIN23]. Additional information regarding Git and GitHub can be found under section 2.2.9.
- **Java**. Pythia was developed using Java 8 as a programming language. Java is a popular and widely used programming language that allows developers to build

a wide range of systems and applications. It is a general-purpose, object-oriented, and platform-independent programming language, meaning that Java code can be run on any platform that has a Java Virtual Machine (JVM) installed.

- **Eclipse**. Eclipse is an open-source integrated development environment (IDE) that was used for the development of Pythia. It is a powerful and widely used IDE for Java development that offers numerous features such as syntax highlighting, refactoring automations, code completion, code navigation, testing and debugging assistance tools and version control integration. However, it should be noted that Pythia is capable of running on any IDE that supports Java, such as IntelliJ or Visual Studio Code.
- **Maven**. Maven is an open-source build automation tool for Java applications that is used to simplify the build process of Pythia as well as to manage the external dependencies of the system. It uses a declarative Extensible Markup Language (XML) file called Project Object Model (POM) to describe specifications of the application such as project structure, dependencies and build processes.
- **Apache Spark**. As described above, Pythia utilizes the Apache Spark [Apac21] engine to quickly process large data sets. Apache Spark is an optimized and unified engine for executing data engineering and analytics on large scale data. It is described in further detail under section 2.2.8.

Installation instructions for setting up Pythia as a developer in a new machine are provided below [Alex22]:

1. Download and install Java 8 on the new machine. Once installation is complete, it is important to edit the path of the JAVA_HOME environment variable such that it points to the newly installed Java directory.
2. Download and install an Integrated Development Environment for Java applications, such as Eclipse or IntelliJ.
3. (Optional, but recommended) Download and install Git and create an account on GitHub. Afterwards, set a global git username and email and perform any other required first-time configurations such that cloning repositories from GitHub to the local machine is possible.
4. Clone the Pythia source code from the repository referenced at [DAIN23] to the local machine. Alternatively, it is also feasible to simply download the source code in the form of a compressed zip file without the usage of Git.
5. That source code includes a Maven Wrapper which integrates an installation of Maven into the system and therefore, no action is required regarding Maven installation.

6. Download and install Apache Hadoop 3.2.2 from the official Hadoop website. Hadoop installation is performed by extracting the downloaded tar.gz compressed file and editing the path of the HADOOP_HOME environment variable such that it points to the newly extracted directory.

    6.1. If the new machine has a Windows operating system, installation of WinUtils for Hadoop 3.2.2 is required. Installation is done by downloading the binary files found at the repository referenced at [WinU21] and placing them within the bin directory of the newly installed Hadoop.

7. Once the above is complete, open the source code using the installed IDE.

Maven should automatically fetch the external dependencies of the system and build the application such that it is ready for execution within the IDE. Most IDEs provide features that allow developers to easily build and run the application, as well as its tests. However, for the sake of complete instructions, important terminal commands for building and running the Pythia system are provided below. Note that any of the following commands should be executed after navigating to the root directory of the system. The root directory contains the mvnw.cmd and mvnw executable files for Windows and Unix based operating systems respectively.

- Build on Windows: **`> ./mvnw.cmd clean install`**
- Build on Unix systems: **`> ./mvnw clean install`**
- Run tests on Windows: **`> ./mvnw.cmd test`**
- Run tests on Unix systems: **`> ./mvnw test`**

The build command produces two Java archive (JAR) files named “Pythia-x.y.z-all-deps.jar” and “Pythia-x.y.z.jar”. Either of the JAR files can be imported into other Java applications in the form of an external library. The first JAR (all-deps) has precompiled and integrated all the external dependencies of Pythia along with the executable source code of Pythia itself such that it can be easily imported into other applications regardless of the dependencies of the other application. On the contrary, the second JAR only contains the executable source code of Pythia, without the external dependencies of the system. Importing this JAR in other Java applications means that the external dependencies of Pythia are required to be imported into the other application in order for Pythia to function. Usage of the second JAR is recommended in applications that already contain the external dependencies of Pythia in order to save disk space.

3.5 Software scalability

The software architecture of Pythia is designed and developed using methodologies that allow for easy maintenance and scalability of the system.

As described above, each of the different features of Pythia is developed and organized into a separate package. The parameterized Factory [Knoe01] software design pattern is implemented in most of the packages, combined with an interface which is implemented by all the classes that get instantiated by the factory class. The factory class is responsible for creating objects based on an input parameter and the interface is responsible for defining a set of methods such that created objects can interact in a consistent manner regardless of the internal logic of each implementation. Therefore, if the need arises, the software can be easily extended with additional logic simply by creating a new class implementing the interface, along with the parameter that separates it from the other implementations. No further refactoring is required.

As far as highlight patterns identification is concerned, the patterns package forms its own internal architecture. The top-level pattern manager is placed directly under the patterns package and is responsible for the overall patterns identification execution flow. The pattern manager implements the parameterized Factory software design pattern design in the previous paragraph, and thus, can be easily extended as described above. The general convention of the patterns package is that the details of each pattern featured in Pythia are organized in a separate sub-package. Additional patterns can be easily contributed to the system by creating a new dedicated sub-package along with a new high-level method in the pattern manager class that is responsible for calling the newly added pattern in the overall highlight patterns identification procedure. An extra modification required for new pattern additions would be updating of the PatternsProfile model class, as well as the classes responsible for reporting such that they include the newly added pattern.

The dominance pattern is placed under the homonymous dominance sub-package and is developed using the Template Method software design pattern to avoid code duplication among the high and low dominance algorithm variations. Contributions to the dominance algorithms are not predicted or expected since -as described above- Pythia already features four (4) variations of the algorithm. However, the dominance column selection procedure could be easily extended with additional logic by adding a new enum value corresponding to the new column selection mode and two additional methods in the DominanceColumnSelector class with the added logic for measurement and coordinate column selection respectively.

Lastly, in a similar manner, the outlier pattern is placed under the homonymous outlier sub-package. The parameterized Factory pattern in combination with an interface is also implemented to support multiple algorithms for outlier detection. It should be noted that Pythia currently features only Z-Score outlier detection and is hard-coded to select this outlier detection algorithm. Additional outlier detection algorithms can be added to the system by creating a new class implementing the IOutlierAlgo interface along with an enum value parameter dedicated to the new algorithm. However, in this case, the analyst should be able to declare the desired outlier detection algorithm.

Chapter 4. Experimental Evaluation

This chapter presents an in-depth description of the experimental evaluation of the newly contributed features to Pythia. The first section provides details regarding the methodologies of the experiments, the selected input dataset, and the environment the experiments were conducted in. The second section presents the results of the experimental evaluation in great detail, in the form of tables and bar charts.

4.1 Experiments methodology

The conducted experiments involved measurement of the execution time for all steps of the data analysis procedure that are related to highlight patterns identification, including dataset registration and report generation. The measured execution times are divided into the following three (3) levels of abstraction, aiming to provide an in-depth overview of the execution times of the different parts of the data analysis procedure:

1. **IDatasetProfiler**. This is the top level of abstraction. As described in the previous chapter, IDatasetProfiler is the central control unit of Pythia. In this level, execution time is measured for core features of the system such as computation of descriptive statistics and identification of highlight patterns. For the sake of simplicity, total execution time is also included in this level. More specifically, execution time was measured for the following IDatasetProfiler operations:
    - Register dataset.
    - Compute descriptive statistics.
    - Compute all pairs correlations.
    - Identify highlight patterns.
    - Generate report in txt format.
    - Generate report in markdown format.
2. **PatternManager**. This is the middle level of abstraction. The PatternManager class is responsible for all operations involved in highlight patterns identification. The operations of this level are included as part of the overall identify highlight patterns operation of the IDatasetProfiler level. Execution time was measured for the following operations:
    - Identify outliers.

- Select measurement and coordinate columns involved in dominance highlights identification.
- Identify dominance highlights with one coordinate column.
- Identify dominance highlights with two coordinate columns.

3. **DominanceAlgo**. This is the lowest level of abstraction. The DominanceAlgo class is responsible for all operations regarding single-coordinate and double-coordinate dominance identification. Time measurements were taken separately for operations that are integrated into both high and low dominance. Namely, execution time was measured for the following operations:
    - Execute get distinct values query. (This operation is only performed in double-coordinate dominance identification.)
    - Execute aggregate query.
    - Execute high dominance algorithm check.
    - Execute low dominance algorithm check.
    - Execute top-K filtering on the identification results.

It should be noted that execution time for the identify outliers operation of the PatternManager abstraction level appears to be sufficient for the experimental evaluation of the outlier pattern due to the simplicity of the outlier detection algorithm.

The goal of the experimental evaluation was to observe the performance and the efficiency of the system in realistic scenarios where Pythia is expected to compute statistics and insights for large-scale datasets. To achieve this, the performance of the system was measured with both input datasets of different record sizes as well as input datasets with different number of column distinct values. For this purpose, the input dataset was transformed into five (5) equally distanced sizes regarding the number of records. Moreover, the dataset was also transformed into three (3) equally distanced sizes regarding the number of distinct values of a column for a fixed record size, to specifically examine how execution time is affected by a column's domain. Execution times were measured separately for each variation of the dataset and are organized into tables and charts in order to provide a better overview of the capabilities of the system.

The dataset that was selected for usage in the experimental evaluation of the system is a variation of the "Adult" dataset [BeKo96]. The "Adult" dataset has been commonly used for educational and research purposes, notably for predicting whether a person's yearly income exceeds 50 thousand dollars. It includes information such as age, work class, education, occupation, race, gender, and work hours per week. The original dataset consists of 48.842 instances with 14 attributes, most of which are of categorical type. The

large number of categorical attributes of the "Adult" dataset make it ideal for dominance highlights identification. The modified variation of the dataset that was used for the experiments on Pythia was cleaned up of duplicate records in [Pila10] and was artificially modified to consist of a total of 10 million records.

All of the experiments were conducted with the same input parameters on Pythia. The parts of the automated data analysis procedure that were executed in the context of the experiments are: descriptive statistics computation, all-pairs correlations computation, and of course, highlight patterns identification. Decision trees generation and histogram computation were omitted from execution due to the fact that these operations are totally uninvolved in highlight patterns identification, which is the main emphasis of this thesis. As far as dominance highlights identification is concerned, the column selection mode was set to user specified only mode. The declared measurement column was "work hours per week" and the declared coordinate columns were "native country", "occupation", and "gender". It should be noted that dominance identification is performed separately for all possible combinations of measurement and coordinate columns.

For the conduction of the experimental evaluation, Apache Spark was configured to utilize the maximum processing capacity (CPU cores) of the physical machine and 75% of the available RAM. The hardware specifications of the machine where the experiments were conducted are presented below:

- **Operating System**: Microsoft Windows 10 Home Version 10.0.19045 Build 19045.
- **CPU**: AMD Ryzen 5 2600X Six-Core Processor, 3600 MHz, 6 Cores, 12 Logical Processors.
- **RAM**: G. Skill Ripjaws V 16GB (2x8GB) DDR4 RAM 3200 MHz
- **SSD**: Samsung 970 Evo Plus SSD 500GB M.2 NVMe PCI Express 3.0. Sequential reads speed up to 3500 MB/s. Sequential writes speed up to 3300 MB/s.

4.2 Detailed results presentation

As mentioned above, the measurement of execution times is divided into three (3) levels of abstraction. This section provides a detailed presentation of the measurement results for each level of abstraction separately. The presentation begins from the top level with measurements of total execution time of Pythia and ends on the lowest level with measurements of specific operations of the dominance pattern.

4.2.1 The IDatasetProfiler level

This subsection presents the results of the highest level of abstraction of the experimental evaluation of Pythia. The following Figures present the total execution time of Pythia over a variable number of records on the input dataset. More specifically, Figure 33 contains a table with the measured total execution times and Figure 34 presents the same information in the form of a bar chart. In both Figures, it is fairly obvious to observe a steady increase of the total execution time as the number of records on the input dataset increases.

Total execution time over number of records					
Number of records (millions)	2	4	6	8	10
Total execution time (milliseconds)	132357	222636	305330	402054	487535

Figure 33. Table of measured total execution time of Pythia over number of records.

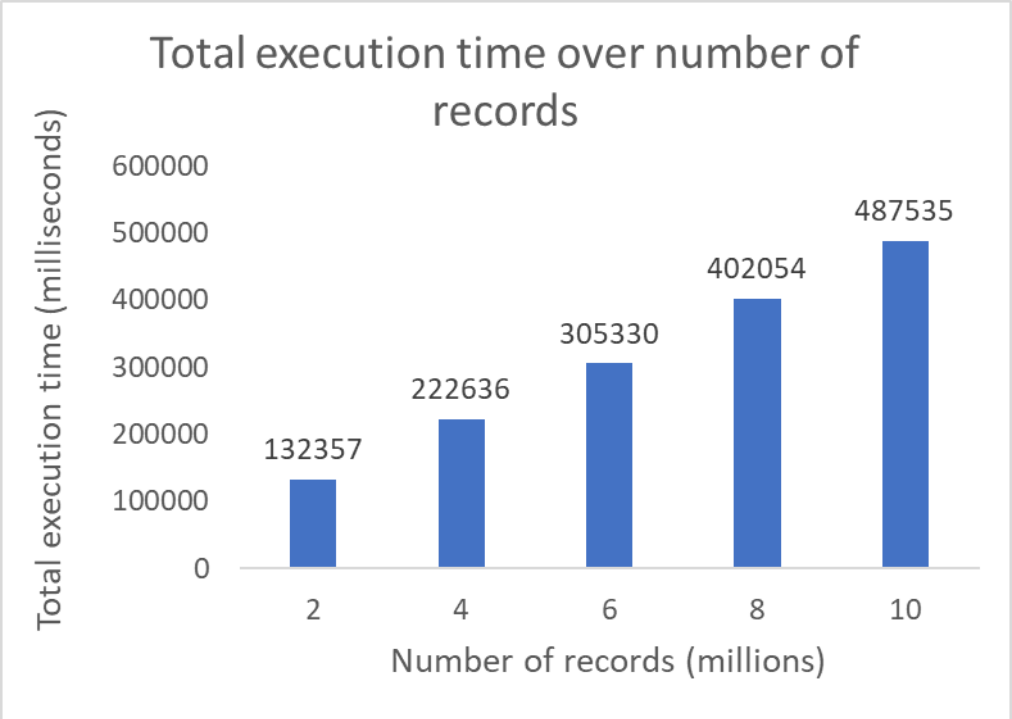

Figure 34. Bar chart of measured total execution time of Pythia over number of records.

The next Figures present a break-down of the total execution time into the different core operations of the IDatasetProfiler class that were performed during execution. Figure 35 contains a table with the measured execution times over the number of records of the input dataset for the core methods of the IDatasetProfiler. Figure 36 contains the same information in the form of a bar chart, which is rather insightful as it showcases how time consuming each of the different operations is. It should be noted that the descriptive statistics calculation and the highlight patterns identification methods appear to be

significantly more time consuming than the other methods. Computation of all-pairs correlations seems to be much less time consuming, which is most likely due to the fact that the "Adult" dataset has mostly categorical attributes. The measurements showcase a steady increase of execution time for descriptive statistics calculation and highlight patterns identification as the number of records on the input dataset increases. On the contrary, the number of records does not seem to have any significant impact on the dataset registration and the report generation methods.

Execution time over number of records for methods of the IDatasetProfiler class					
Number of records (millions)	2	4	6	8	10
registerDataset (milliseconds)	1887	1854	2530	2103	1879
computeDescriptiveStats (milliseconds)	72899	139753	201537	271421	338976
computeAllPairsCorrelations (milliseconds)	3770	6258	8062	11576	12545
identifyHighlightPatterns (milliseconds)	47526	68533	86583	110374	127839
generateReport - txt (milliseconds)	29	35	33	60	37
generateReport - markdown (milliseconds)	16	17	16	83	16

Figure 35. Table of measured execution times over number of records for methods of the IDatasetProfiler class.

Figure 36. Bar chart of measured execution times over number of records for methods of the IDatasetProfiler class.

In a similar manner, the following Figures present the measurements of execution times over a variable number of column distinct values for a fixed number of records. In detail, Figure 37 contains a table with the measured total execution times and Figure 38 presents the same information in the form of a bar chart. In both Figures, it is observed that the increase of a column's domain appears to slightly increase the total execution time. However, as it is expected, the total execution time increase is not as significant as that of the experiments with a variable number of records.

Total execution time over number of column distinct values			
Number of distinct values of the 'native_country' column	10	25	40
Total execution time (milliseconds)	126511	131908	132279

Figure 37. Table of measured total execution time of Pythia over number of column distinct values.

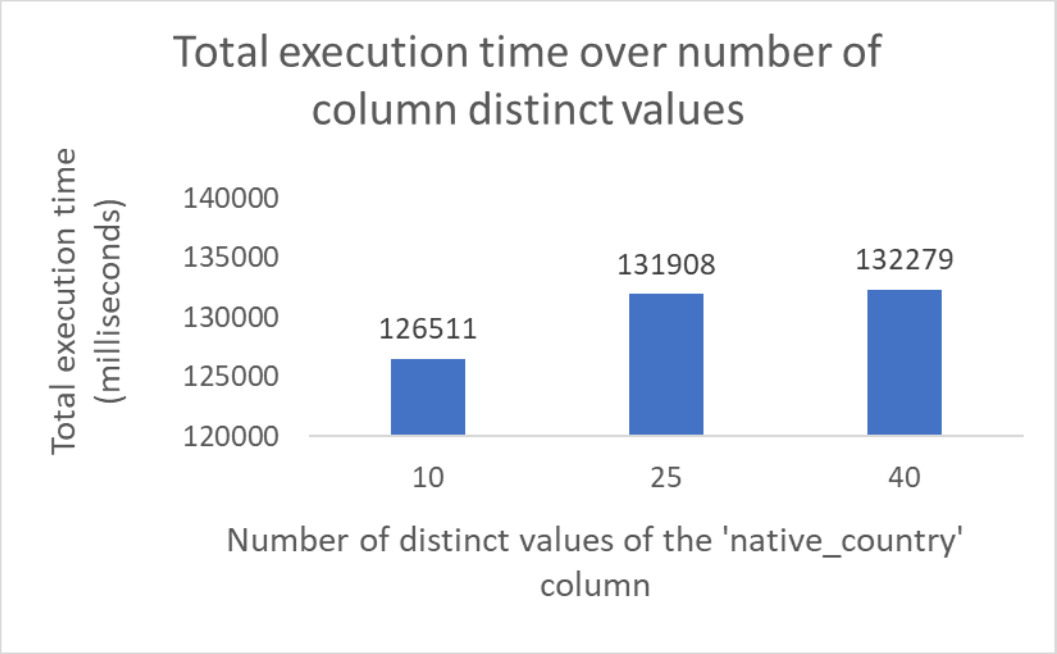

Figure 38. Bar chart of measured total execution time of Pythia over number of column distinct values.

Similarly, the following Figures present a break-down of the total execution time into the different core operations of the IDatasetProfiler. Figure 39 contains a table with the measured execution times over the number of distinct values of the 'native_country' column. Figure 40 contains the same information in the form of a bar chart, which is rather insightful as it showcases the duration of each operation. Similarly to the above experiments, the descriptive statistics calculation and the highlight patterns identification methods appear to be significantly more time consuming than the other methods. Computation of all-pairs correlations seems to be much less time consuming, which is most likely due to the fact that the "Adult" dataset has mostly categorical attributes. It should be noted that the variable number of column distinct values does not seem to affect any operation other than the identification of highlight patterns. An increase of column distinct values appears to slightly increase the duration of the identify highlight patterns method. However, as it is expected, the execution time increase is not as significant as that of the experiments with a variable number of records.

Execution time over number of column distinct values for methods of the IDatasetProfiler class			
Number of distinct values of the 'native_country' column	10	25	40
registerDataset (milliseconds)	1874	1874	1889
computeDescriptiveStats (milliseconds)	73491	71220	70591
computeAllPairsCorrelations (milliseconds)	4224	4086	4152
identifyHighlightPatterns (milliseconds)	40704	48242	49442
generateReport - txt (milliseconds)	24	251	32
generateReport - markdown (milliseconds)	13	15	13

Figure 39. Table of measured execution times over number of column distinct values for methods of the IDatasetProfiler class.

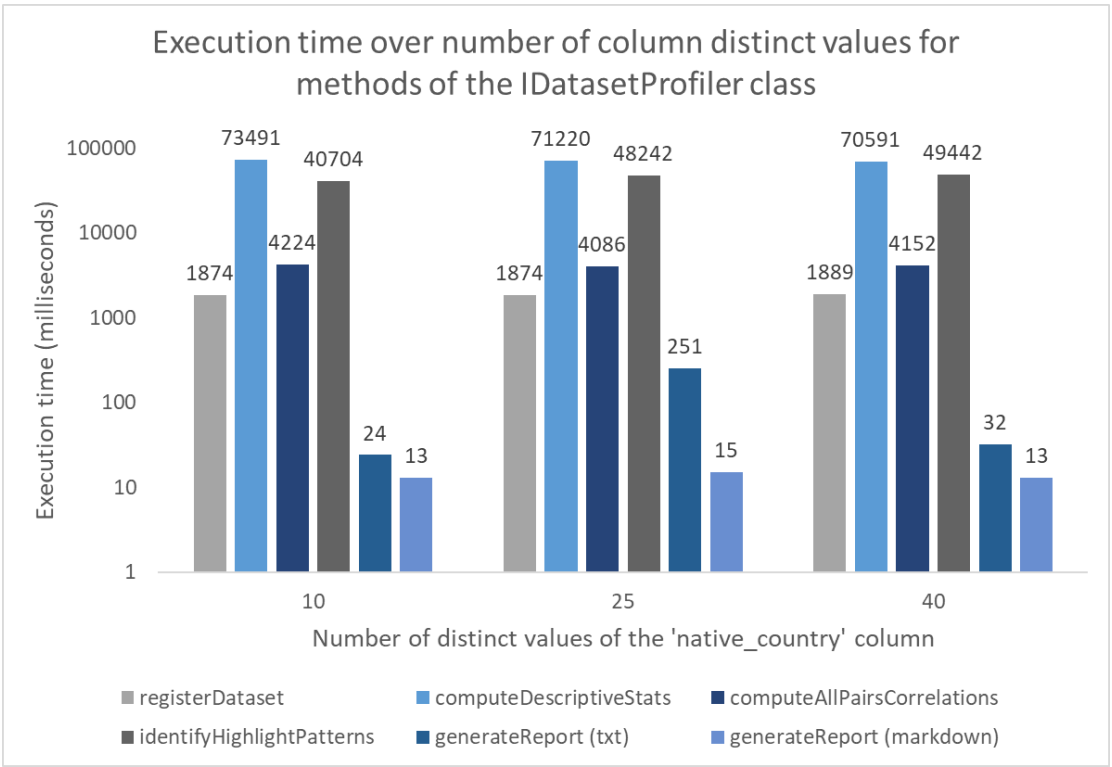

Figure 40. Bar chart of measured execution times over number of column distinct values for methods of the IDatasetProfiler class.

4.2.2 The PatternManager level

This subsection presents the results of the middle level of abstraction of the experimental evaluation of Pythia. The next Figures present a break-down of the identify highlight patterns method execution time into the different operations of the PatternManager class. Figure 41 contains a table with the measured execution times over the number of records of the input dataset and Figure 42 presents the same information in the form of a bar chart. In both Figures it is fairly obvious to observe that the double-coordinate dominance identification method seems to be significantly more time consuming than the other methods. An increase of the number of records on the input dataset appears to steadily increase the duration of all operations with the exception of the selection of columns for dominance identification. However, the increase is much more significant in the double-coordinate dominance identification method.

Execution time over number of records for methods of the PatternManager class					
Number of records (millions)	2	4	6	8	10
identifyOutliers (milliseconds)	2883	5942	9000	12051	14973
select dominance columns (milliseconds)	1	0	1	8	1
identifyDominanceWithOneCoordinate (milliseconds)	4956	7381	9451	12662	14490
identifyDominanceWithTwoCoordinates (milliseconds)	39679	55205	68126	85587	98369

Figure 41. Table of measured execution times over number of records for methods of the PatternManager class.

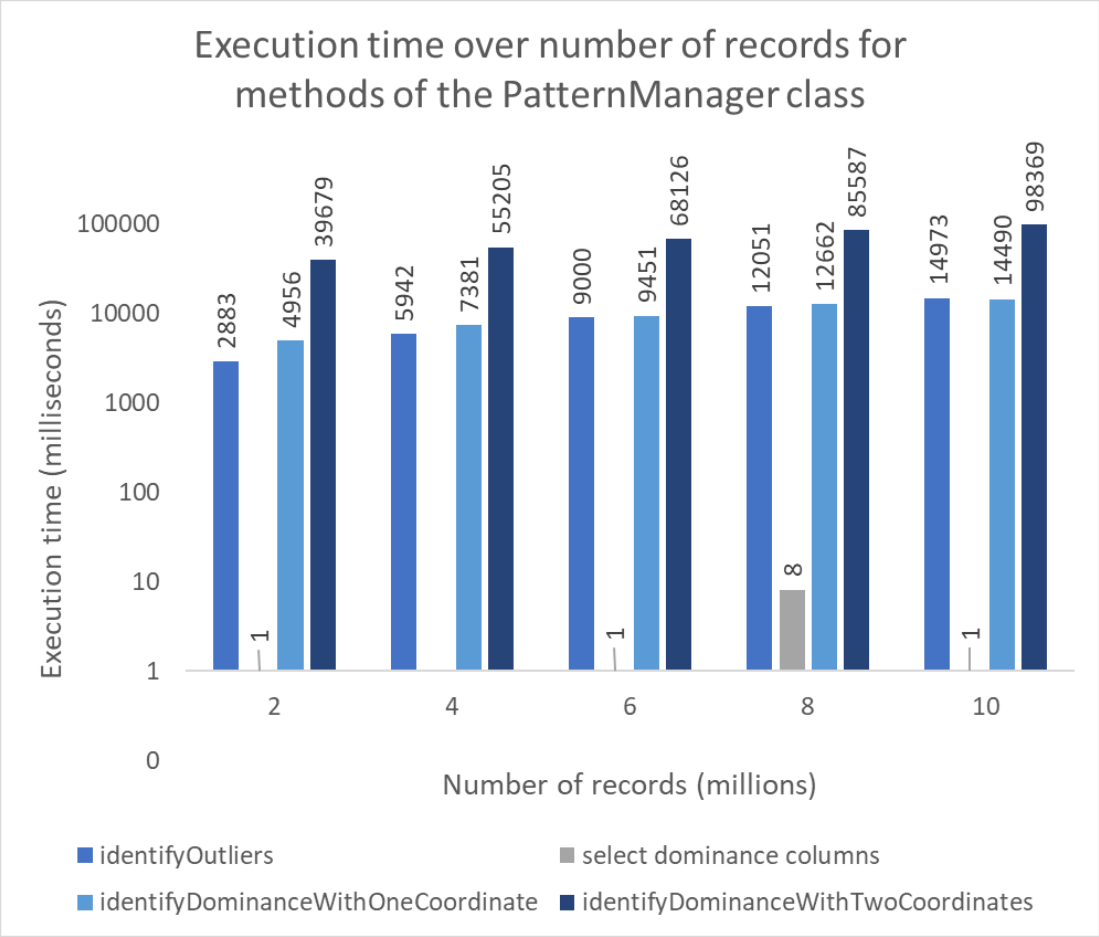

Figure 42. Bar chart of measured execution times over number of records for methods of the PatternManager class.

In a similar manner, the next Figures also present a break-down of the identify highlight patterns method execution time into the different operations of the PatternManager class. The Figures present the measurements of execution times over a variable number of column distinct values for a fixed number of records. Figure 43 contains a table with the measured execution times and Figure 44 presents the same information in the form of a bar chart. In both Figures, it is observed that the increase of a column's domain appears to only increase the duration of the double-coordinate dominance identification method. The number of column distinct values is expected to affect the double-coordinate dominance identification, since the algorithm internally executes a get distinct values query against the input dataset. The other methods of the PatternManager class appear to be unaffected by the variable number of column distinct values.

Execution time over number of column distinct values for methods of the PatternManager class			
Number of distinct values of the 'native_country' column	10	25	40
identifyOutliers (milliseconds)	3248	3118	3042
select dominance columns (milliseconds)	0	1	1
identifyDominanceWithOneCoordinate (milliseconds)	4520	5100	5019
identifyDominanceWithTwoCoordinates (milliseconds)	32930	40018	41376

Figure 43. Table of measured execution times over number of column distinct values for methods of the PatternManager class.

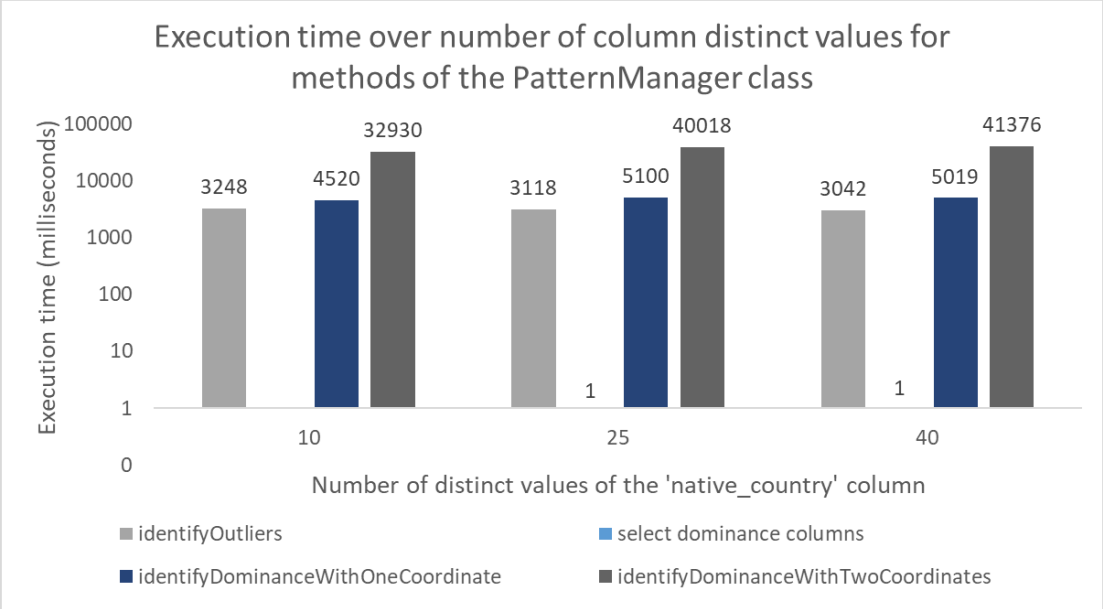

Figure 44. Bar chart of measured execution times over number of column distinct values for methods of the PatternManager class.

4.2.3 The DominanceAlgo level

This subsection presents the results of the lowest level of abstraction of the experimental evaluation of Pythia, with operations specifically related to the dominance pattern. The operations of this level are a break-down of the single and double coordinate dominance identification methods of the PatternManager level. Due to the fact that dominance identification is generally performed for all possible combinations of the declared (or selected) measurement and coordinate columns, the presented results only concern dominance identification with the 'hours_per_week' measurement column, the 'native_country' as a first coordinate column and the 'occupation' as a second coordinate column. As expected, the results of high and low dominance methods bear great similarity to each other in both single and double coordinate checks and are therefore grouped together w.r.t the two different experiment types that were conducted.

Figures 45 and 46 present the measured execution times for single-coordinate high and low dominance identification respectively, over a variable number of records on the input dataset. Figure 47 presents the same information in the form of two bar charts. The bar charts are grouped to ease comparison as the measured times appear to be rather similar. In all Figures, it is easy to observe that the duration of the data aggregation query is magnitudes higher than the rest of the dominance identification related operations. An increase of the number of records appears to slightly but steadily increase the execution time of the aggregation query. The number of records does not seem to affect the duration of the other operations which is expected since the data used by the other dominance related operations is the prepared data of the aggregation query. It should be noted that the difference between the durations of the top-K filtering operations between low and high dominance is due to the fact that top-K filtering is performed on identified dominance highlights. In cases where no dominance highlights were identified, it is expected that the duration of top-K filtering is minimum and negligible.

Execution time over number of records for single-coordinate dominance methods of the HighDominanceAlgo class					
Number of records (millions)	2	4	6	8	10
runAggregateQuery (milliseconds)	1491	1822	2151	2757	3147
high dominance algorithm check (milliseconds)	1	2	3	2	4
top-K filtering (milliseconds)	9	3	7	6	6

Figure 45. Table of measured execution times over number of records for single-coordinate dominance methods of the HighDominanceAlgo class for measurement 'hours_per_week' and coordinate 'native_country'.

Execution time over number of records for single-coordinate dominance methods of the LowDominanceAlgo class					
Number of records (millions)	2	4	6	8	10
runAggregateQuery (milliseconds)	881	1264	1609	2245	2491
low dominance algorithm check (milliseconds)	3	2	2	2	2
top-K filtering (milliseconds)	1	0	1	1	0

Figure 46. Table of measured execution times over number of records for single-coordinate dominance methods of the LowDominanceAlgo class for measurement 'hours_per_week' and coordinate 'native_country'.

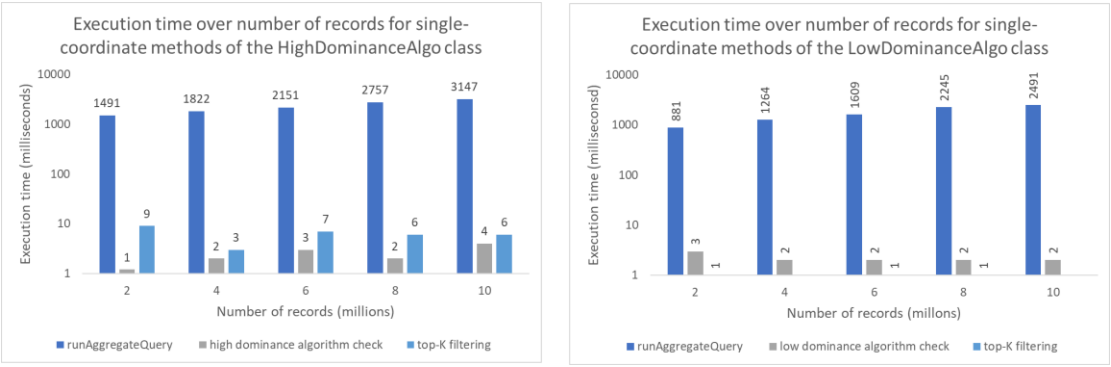

Figure 47. Bar charts of measured execution times over number of records for single-coordinate dominance methods of the HighDominanceAlgo class (left) and the LowDominanceAlgo (right) for measurement 'hours_per_week' and coordinate 'native_country'.

Figures 48 and 49 present the measured execution times for single-coordinate high and low dominance identification, over a variable number of column distinct values. As mentioned above, the 'native_country' was selected to be the coordinate column with a variable number of distinct values due to the fact that it had the most distinct values compared to the other columns of the "Adult" dataset. Figure 50 presents the same information in the form of two bar charts, for high and low dominance respectively. The observations are rather similar to the experiments with a variable number of records. The duration of the data aggregation query appears to be orders of magnitude higher than the duration of the other dominance related operations. An increase of a columns distinct values appears to slightly increase the execution time of the aggregation query. However, the increase in execution time is less significant than that of the experiments with a variable number of records. Furthermore, the number of column distinct values does not seem to affect the duration of the other operations.

Execution time over number of column distinct values for single-coordinate dominance methods of the HighDominanceAlgo class			
Number of distinct values of the 'native_country' column	10	25	40
runAggregateQuery (milliseconds)	1067	1279	1338
high dominance algorithm check (milliseconds)	1	3	3
top-K filtering (milliseconds)	1	3	4

Figure 48. Table of measured execution times over number of column distinct values for single-coordinate dominance methods of the HighDominanceAlgo class for measurement 'hours_per_week' and coordinate 'native_country'.

Execution time over number of column distinct values for single-coordinate dominance methods of the HighDominanceAlgo class			
Number of distinct values of the 'native_country' column	10	25	40
runAggregateQuery (milliseconds)	696	865	897
high dominance algorithm check (milliseconds)	0	1	1
top-K filtering (milliseconds)	1	2	0

Figure 49. Table of measured execution times over number of column distinct values for single-coordinate dominance methods of the LowDominanceAlgo class for measurement 'hours_per_week' and coordinate 'native_country'.

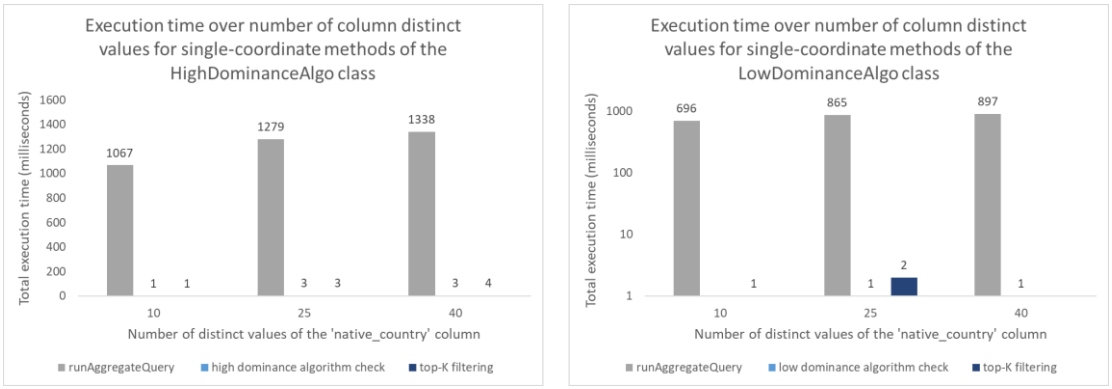

Figure 50. Bar charts of measured execution times over number of column distinct values for single-coordinate dominance methods of the HighDominanceAlgo class (left) and the LowDominanceAlgo (right) for measurement 'hours_per_week' and coordinate 'native_country'.

Similarly, the following Figures present the measurement results of double-coordinate dominance operations over a variable number of records. Figures 51 and 52 present the measurement in the form of tables, while Figure 53 presents the same information in the form of two bar charts, for high and low dominance respectively. Note that, in double-coordinate dominance identification, an additional query is executed against the registered dataset which fetches the distinct values of the coordinate columns. As with single-coordinate dominance, the measurements showcase that the duration of the two queries is orders of magnitude higher than the duration of the other operations. An increase of the number of records on the input dataset appears to steadily increase the execution time of both queries. The number of records does not seem to have an impact on the execution time of the other operations.

Execution time over number of records for double-coordinate dominance methods of the HighDominanceAlgo class					
Number of records (millions)	2	4	6	8	10
runGetDistinctValuesQuery (milliseconds)	1688	2435	3172	4144	4677
runAggregateQuery (milliseconds)	3970	4511	4813	5542	5616
high dominance algorithm check (milliseconds)	56	68	63	68	70
top-K filtering (milliseconds)	0	0	1	1	0

Figure 51. Table of measured execution times over number of records for double-coordinate dominance methods of the HighDominanceAlgo class for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

Execution time over number of records for double-coordinate dominance methods of the LowDominanceAlgo class					
Number of records (millions)	2	4	6	8	10
runGetDistinctValuesQuery (milliseconds)	1446	2316	3102	4016	4532
runAggregateQuery (milliseconds)	3538	3946	4392	5206	5255
low dominance algorithm check (milliseconds)	12	13	14	14	13
top-K filtering (milliseconds)	1	1	0	1	1

Figure 52. Table of measured execution times over number of records for double-coordinate dominance methods of the LowDominanceAlgo class for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

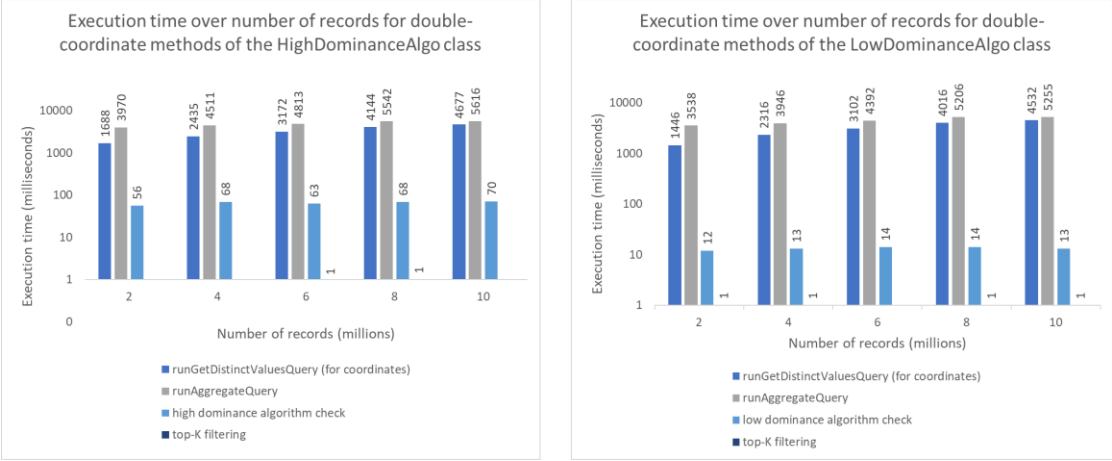

Figure 53. Bar charts of measured execution times over number of records for double-coordinate dominance methods of the HighDominanceAlgo class (left) and the LowDominanceAlgo (right) for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

The measurement results for double-coordinate dominance operations with a variable number of column distinct values are rather similar. Figures 54 and 55 present the results in the form of tables, while Figure 56 present the same information in the form of two bar charts. As with experiments with a variable number of records, the results here showcase that the execution time of the two queries is orders of magnitude higher than the duration of the other dominance related operations. An increase of column distinct values appears to slightly increase the duration of the two queries. However, the duration increase is less

significant than that of the experiments with a variable number of records. It should be noted that the execution time did not increase between 25 and 40 column distinct values in Figure 52, and that is most likely due to the fact that the duration increase of the query that fetches distinct values is rather small. Furthermore, an increase of column distinct values appears to also increase the duration of the dominance check. However, the duration of the dominance check is still negligible compared to the duration of the two queries.

Execution time over number of column distinct values for double-coordinate dominance methods of the HighDominanceAlgo class			
Number of distinct values of the 'native_country' column	10	25	40
runGetDistinctValuesQuery (milliseconds)	1501	1741	1741
runAggregateQuery (milliseconds)	2725	3904	3984
high dominance algorithm check (milliseconds)	4	16	53
top-K filtering (milliseconds)	0	0	0

Figure 54. Table of measured execution times over number of column distinct values for double-coordinate dominance methods of the HighDominanceAlgo class for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

Execution time over number of column distinct values for double-coordinate dominance methods of the LowDominanceAlgo class			
Number of distinct values of the 'native_country' column	10	25	40
runGetDistinctValuesQuery (milliseconds)	1415	1496	1537
runAggregateQuery (milliseconds)	2149	3431	3576
high dominance algorithm check (milliseconds)	2	14	16
top-K filtering (milliseconds)	0	0	0

Figure 55. Table of measured execution times over number of column distinct values for double-coordinate dominance methods of the LowDominanceAlgo class for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

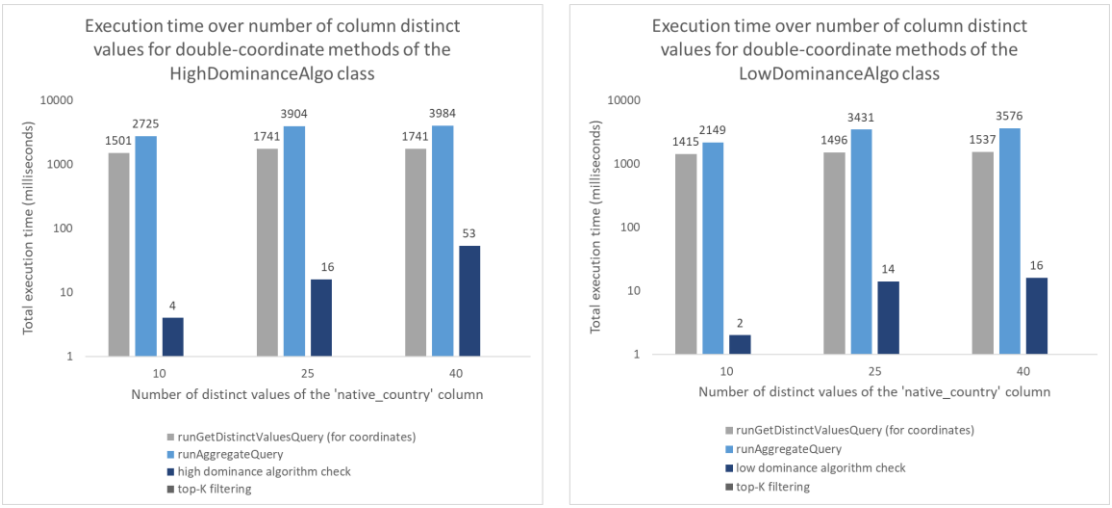

Figure 56. Bar charts of measured execution times over number of column distinct values for double-coordinate dominance methods of the HighDominanceAlgo class (left) and the LowDominanceAlgo (right) for measurement 'hours_per_week' and coordinates 'native_country' and 'occupation'.

Overall, the experiments on the DominanceAlgo level showcase how the duration of any query against the dataset is orders of magnitude higher than the duration of any other dominance operation. The methods of single-coordinate dominance identification are less time consuming than those of double-coordinate dominance identification, which is expected due to the fact that a second coordinate increases the complexity of both the aggregate query and the identification algorithm. Also, in double-coordinate dominance identification, an additional query is executed that fetches distinct values of coordinates. The additional query is of equal overload with the aggregate query, which has a significant impact on the increase of the overall execution time of double-coordinate dominance.

Moreover, the number of records appears to have a greater impact on the increase of execution time compared to the number of column distinct values. Naturally, an even greater (or extreme) number of column distinct values would be expected to also greatly increase execution time of the queries. However, an extreme number of column distinct values would not be suitable in the context of dominance identification as it would clutter the identification results with too much information and most likely make them incomprehensible for the data analyst.

Chapter 5. Epilogue

5.1 Synopsis and conclusions

Exploratory Data Analysis (EDA) is a procedure used by data scientists to analyze complex data sets and obtain insights such as patterns and anomalies. However, the lack of automation in EDA hinders the acquisition of such insights, due to the fact that analysts have to manually execute the desired analysis methods and perform any potential data preprocessing. To address this challenge, a system called Pythia [Alex22] was developed with the aim to facilitate automated EDA. The general idea of Pythia is that the system accepts a dataset as input, processes it, and a valuable insights overview about it is automatically generated. Pythia is a Java application which utilizes the Apache Spark engine to rapidly process large datasets.

Prior to this thesis, Pythia was in its early stages of development and had certain limitations in its functionalities. Pythia was capable of accepting a data set as input and generating a detailed statistics profile about it including automated correlation calculation. The system was also capable of generating decision trees for labeled fields of the data set in a simplistic manner. Essentially, the system was primarily focusing on evaluating the quality of the input dataset.

In the context of this thesis, Pythia was extended to not only evaluate the quality of the input dataset but also to produce valuable insights about it in an automated manner. The system was extended with highlight extractor modules that actively search for and identify both holistic highlights and point-based highlights in the input data. Holistic highlights concern an entire area of the dataspace and point-based highlights only concern a specific point in the input data. Each highlight extractor module was equipped with the following procedures:

1. **Data preparation**. Data preparation refers to any procedures, such as queries, that prepare or select the data for identification of the highlight pattern at hand.
2. **Highlight identification algorithm**. The selected data is passed by the highlight identification algorithm which processes it and generates results regarding the existence or absence of highlight patterns.
3. **Highlight identification result object(s)**. Each identification algorithm produces respective results that are stored in separate concrete result objects.

4. **Reporting mechanisms**. In the end, once all data analysis procedures of Pythia have finished and the respective result objects have been generated, the highlight identification results are exported to report files, which contain all the insightful information saved in the result objects.

In further detail, Pythia was extended to identify dominance and outlier highlights. Dominance is a holistic highlight pattern that allows analysts to identify partial or total dominance occurrences for specific coordinates of the input dataset w.r.t. a selected measurement. For instance, an analyst might obtain insights such as which occupation has worked the most or the least hours per week among different countries or which car model had the highest or lowest price among different years. Pythia was also augmented with the ability to accept input parameters that determine whether the selection of dominance measurement and coordinate columns should be performed automatically or manually by the data analyst. On the other hand, outlier is a point-based highlight pattern that allows analysts to detect numerical data points whose value is notably different from the others in the input dataset. Outlier identification was achieved using the Z-Score algorithm.

Overall, Pythia was extended with the ability to identify highlight patterns such that valuable insights about the input dataset are produced in an automated manner. The development of the newly added features was done in a flexible way such that the system can be easily extended with more highlight extractor modules. Lastly, in order to improve the scalability and usability of the system, Pythia was also augmented with the ability to accept parameters regarding which parts of the automated data analysis pipeline should be executed, such that certain data analysis parts can be excluded or executed individually.

5.2 Future extensions

Finally, this section presents a list of suggestions with potential extensions that would benefit Pythia:

- **Additional highlight patterns.** In the context of this thesis, the most obvious contribution to Pythia would be the contribution of additional highlights patterns. As mentioned above, the highlight extractor modules were developed in a manner that supports software scalability and extensibility. Pythia could be extended with additional algorithms for outlier detection or entirely new algorithms that search for and identify highlight patterns, such as statistical highlights, trends, or seasonality highlight patterns.

- **Clustering analysis.** The data analysis pipeline of Pythia could be extended with additional analysis techniques, such as clustering. Clustering analysis could be performed based on values of the attributes of the input dataset, such that the attribute values of the input dataset are grouped into clusters and an additional statistical insight is automatically generated by Pythia.
- **A front-end interface.** The implementation of a front-end interface for Pythia would be a rather significant contribution to the project. It would greatly improve the usability of the system, especially for analysts that are unfamiliar with Java or programming in general. In its current state, the system can only be used as a Java application or as a JAR dependency into other Java projects. A front-end interface could be a terminal-only application, a Desktop application where the analyst would be able to use Pythia via a user interface, or even a web application that uses Pythia as a back-end. In theory, a front-end application could also extend the capabilities of Pythia by visualizing data in charts, in cases where it is applicable.
- **Evaluation of the highlight results and improvement of the generated reports.** Currently, the system generates multiple reports due to the large volume of generated results from the identification of highlight patterns. A significant and rather necessary contribution would be the evaluation of the generated results such that a more concise report is generated with results of all data analysis techniques as well as the highlight pattern results that are evaluated as interesting.
- **Optimization of the dominance pattern.** As shown in the experiments in chapter 4, the most time-consuming operation during dominance identification is the execution of data preparation queries. Queries for low and high dominance respectively are identical and could therefore be executed only once. In theory, such an optimization would roughly reduce the execution time of dominance identification to half. Moreover, the get distinct values query for the coordinates in double-coordinate dominance identification, could be avoided, such that distinct values are fetched from the results of the aggregate query. Another optimization would be a smarter selection of dominance measurement and coordinate columns, such that columns that do not make logical sense are not selected for dominance identification as such operations practically take up execution time for results that are most likely not interesting.

References

[Agga15] Charu C. Aggarwal. Data Mining: The Textbook. Springer, 2015.

[Alex22] Alexandros Alexiou. Automated Generation of Statistical Profiles for Data Sets (Diploma Thesis). Department of Computer Science & Engineering, University of Ioannina, Greece. March 2022.

[Amaz23] Amazon Web Services. Introduction to Apache Spark. Available at https://aws.amazon.com/big-data/what-is-spark/, January 2023.

[Apac21] Apache Spark – Unified engine for large-scale data analytics. Apache Spark 3.2.0 Documentation. Available at https://spark.apache.org/docs/3.2.0, October 2021.

[BeKo96] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository. Available at https://doi.org/10.24432/C5XW20, 1996.

[Bosl12] Sarah Boslaugh. Statistics in a Nutshell, 2nd Edition. O'Reilly Media, November 2012.

[Char23] Alexandros Charisis. Automated Extraction of Decision Trees in a Data Profiling System (Diploma Thesis). Department of Computer Science & Engineering, University of Ioannina, Greece. March 2023.

[ChSt22] Scott Chacon, Ben Straub. Pro Git, Everything You Need To Know About Git, Second Edition. Apress. Available online at https://git-scm.com/book/en/v2, February 2022.

[DAIN23] DAta INTensive Information EcoSystemS Group (DAINTINESS). Pythia source code GitHub repository. Department of Computer Science & Engineering, University of Ioannina, Greece. Available at https://github.com/DAINTINESS-Group/Pythia, June 2023.

[IBMC20] IBM Cloud Education. Exploratory Data Analysis. Available at https://www.ibm.com/cloud/learn/exploratory-data-analysis, August 2020.

[JUni21] JUnit. JUnit 4 Project Documentation. Available at https://junit.org/junit4/, February 2021.

[Knoe01] Kirk Knoernschild. Java™ Design: Objects, UML, and Process. Addison Wesley, December 2001.

[Mach19] Machine Learning Mastery. Statistical Data Distributions. Available at https://machinelearningmastery.com/statistical-data-distributions, August 2019.

[Orac22] Oracle. JSON Defined. Available at https://www.oracle.com/database/what-is-json, November 2022.

[Pila10] Alexandra Pilalidou. Online Negotiation for Privacy Preserving Data Publishing (MSc Thesis). Department of Computer Science, University of Ioannina, Greece. July 2010.

[PRSD21] Pingchuan Ma, Rui Ding, Shi Han, Dongmei Zhang. MetaInsight: Automatic Discovery of Structured Knowledge for Exploratory Data Analysis. SIGMOD '21, China, June 2021.

[Refa23] Refactoring Guru. The Catalog of Design Patterns. Available at https://refactoring.guru/design-patterns/catalog, March 2023.

[Wiki22] Wikipedia. Data. Available at https://en.wikipedia.org/wiki/Data, November 2022.

[WinU21] WinUtils GitHub repository with binary files required for Hadoop 3.2.2 in Windows Operating Systems. Available at https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.2/bin, September 2021.