Data Profiling Enhancement through Machine Learning Techniques

Georgios Karathanos

Diploma Thesis

Supervisor: Prof. P. Vassiliadis

Ioannina, June 2024

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Acknowledgments

I am profoundly grateful for the indispensable guidance and unwavering support provided by my supervisor, Professor Panos Vassiliadis. His mentorship has been instrumental in shaping the trajectory of this thesis, offering invaluable insights and direction every step of the way. Despite his busy schedule and multitude of obligations, he consistently prioritized providing expert guidance and support throughout the thesis development process. I am immensely grateful for his outstanding commitment and generosity, which have left me indebted to him.

I would like to extend my heartfelt gratitude to my family members and all those closest to me for their unwavering belief in me and steadfast support throughout this journey. Their encouragement and guidance have been a constant source of strength, motivating me to overcome challenges and pursue my goals with determination. Each of them has played a unique and invaluable role in my life, offering their support and assistance whenever I needed it most. I am truly blessed to have such a loving and supportive network of family and friends, and I am deeply thankful for their unwavering presence in my life.

Lastly, I want to thank the Academic Staff of the Computer Science & Engineering Department, as each one of them contributed to the achievement of my academic background.

As I conclude my undergraduate studies, I am thankful to all those mentioned above.

June 2024

Georgios Karathanos

Abstract

It is a fact that in recent decades, data analysis has become one of the most popular research fields. This is quite expected, as data storage is a routine this is met in almost all scientific fields and so their processing to draw specific conclusions is a necessary technique so that their value is not lost. Most of the time, this processing is a difficult procedure since the large amount of data makes this analysis inefficient. Also, an important problem with data analysis is the appropriateness of the techniques and algorithms that are selected depending on the dataset that is under study. Data analysis should be broken down into individual steps so that the final analyst will only have to deal with drawing conclusions based on the results of the algorithms applied to the data and not take on more responsibilities, such as collecting them.

The Pythia system provides a solution to the above problem. Pythia is a data analysis software, implemented in Java and based on the integrated Apache Spark engine, that comes to apply the analysis algorithms efficiently on a large amount of data. Pythia accepts a dataset as input, processes it, and automatically produces valuable information about it. The Pythia system can detect hidden patterns inside the data, calculate statistics, and even produce decision trees. All this is done automatically, and the results are presented to the analyst through a detailed report.

In the context of this thesis, the Pythia system is extended in order to integrate Machine Learning techniques for further deriving conclusions from the datasets it processes. More specifically, the goal of this diploma thesis is to extend Pythia with two of the most popular Machine Learning techniques, regression, and clustering. Through regression we want to know what the correlation is between two or more features of the data. In terms of clustering, different families of algorithms of this technique are tested with the aim of the analyst obtaining the final clusters resulting from the basis data of the optimal algorithm per case.

The extensions to the system are tested with thorough checks tests and evaluated with experiments. It is worth noting that the evaluation of Machine Learning models is a difficult and well-studied scientific subject, so the evaluation carried out in the context of this thesis concerns an important part of it. According to the convention of the existing project, the contributed code is written in Java and is easily extensible since it follows the design patterns of the existing code.

Keywords: Data Analytics, Dataset Profiling, Data Science, Machine Learning, Clustering, Regression, Java, Apache Spark.

Περίληψη

Είναι γεγονός πως τις τελευταίες δεκαετίες το ευρύ φάσμα της ανάλυσης δεδομένων αποτελεί ένα από τα πιο δημοφιλή ερευνητικά πεδία. Κάτι τέτοιο είναι απόλυτα λογικό καθώς η καταγραφή δεδομένων είναι μια ρουτίνα η οποία συναντάται σχεδόν σε όλες τις επιστήμες και έτσι η επεξεργασία τους για την εξαγωγή συγκεκριμένων συμπερασμάτων είναι απαραίτητη τεχνική έτσι ώστε να μη χαθεί η αξία τους. Τις περισσότερες φορές αυτή η επεξεργασία είναι μια δύσκολη διαδικασία αφού ο μεγάλος όγκος των δεδομένων σε συνδυασμό με την ακατέργαστη μορφή τους κάνει αυτή την ανάλυση μη αποδοτική. Επίσης, ένα σημαντικό πρόβλημα με την ανάλυση δεδομένων είναι και η κατάλληλη επιλογή τεχνικών και αλγορίθμων ανάλογα με το υπό μελέτη σύνολο δεδομένων. Είναι κατανοητό, λοιπόν, πως η ανάλυση δεδομένων πρέπει να σπάσει σε επιμέρους κομμάτια με σκοπό ο τελικός αναλυτής να έχει να αντιμετωπίσει μόνο την εξαγωγή συμπερασμάτων από το αποτελέσματα των αλγορίθμων που έχουν εφαρμοστεί στα δεδομένα και να μην αναλάβει περισσότερες ευθύνες, όπως την περισυλλογή τους.

Λύση στο παραπάνω πρόβλημα έρχεται να δώσει το σύστημα Pythia. Το Pythia είναι ένα λογισμικό αναλυτικής δεδομένων, υλοποιημένο σε Java και βασισμένο στην ενοποιημένη μηχανή Apache Spark, το οποίο έρχεται να εφαρμόσει τους αλγορίθμους ανάλυσης αποδοτικά σε μεγάλο όγκο δεδομένων. Το Pythia αποδέχεται ένα σύνολο δεδομένων ως είσοδο, το επεξεργάζεται, και παράγει αυτόματα πολύτιμες πληροφορίες για αυτό. Το σύστημα Pythia έχει τη δυνατότητα να εντοπίζει μοτίβα που κρύβονται μέσα στα δεδομένα, να υπολογίζει στατιστικά, ακόμα και να παράγει δένδρα αποφάσεων. Όλα αυτά γίνονται αυτόματα και τα αποτελέσματα παρουσιάζονται στον αναλυτή μέσω μιας λεπτομερούς αναφοράς.

Στα πλαίσια της συγκεκριμένης διατριβής, το σύστημα Pythia επεκτείνεται με σκοπό την ενσωμάτωση τεχνικών Μηχανικής Μάθησης για περαιτέρω εξαγωγή συμπερασμάτων από τα σύνολα δεδομένων που επεξεργάζεται. Πιο συγκεκριμένα, στόχος αυτής της επέκτασης είναι η εφαρμογή δύο από τις πιο δημοφιλές τεχνικές Μηχανικής Μάθησης και η αξιολόγηση τους, η παλινδρόμηση και η ομαδοποίηση. Μέσω της παλινδρόμησης (regression) θέλουμε να μάθουμε ποια είναι η συσχέτιση μεταξύ δύο η περισσότερων χαρακτηριστικών των δεδομένων. Έτσι μπορούμε να δούμε κατά πόσο ένα χαρακτηριστικό των δεδομένων επηρεάζεται από τις τιμές που παίρνει ένα άλλο χαρακτηριστικό και κατ' επέκταση να μπορούμε να προβλέψουμε τιμές του συγκεκριμένου χαρακτηριστικού δοθέντος τις τιμές από τα υπόλοιπα με τα οποία έχει αποδειχθεί πως συσχετίζεται σε υψηλό βαθμό. Όσο αναφορά την ομαδοποίηση,

δοκιμάζονται διάφορες οικογένειες αλγορίθμων της συγκεκριμένης τεχνικής με σκοπό ο αναλυτής να λάβει τις τελικές ομάδες που προκύπτουν από τα δεδομένα βάση του βέλτιστου αλγορίθμου ανά περίπτωση.

Οι προαναφερόμενες επεκτάσεις στο σύστημα δοκιμάζονται με διεξοδικούς ελέγχους και αξιολογούνται με πειράματα. Αξίζει να σημειωθεί πως η αξιολόγηση των μοντέλων Μηχανικής Μάθησης είναι ένα δύσκολο και αρκετά μελετημένο επιστημονικό αντικείμενο οπότε η αξιολόγηση που γίνεται στα πλαίσια αυτής της διατριβής αφορά ένα σημαντικό κομμάτι της. Σύμφωνα με τη σύμβαση του υπάρχοντος έργου, ο συνεισφερόμενος κώδικας γράφεται σε Java και είναι εύκολα επεκτάσιμος αφού ακολουθεί τα σχεδιαστικά μοτίβα του υπάρχοντος κώδικα.

Λέξεις Κλειδιά: Αναλυτική Δεδομένων, Προφίλ Συνόλων Δεδομένων, Μηχανική Μάθηση, Ομαδοποίηση, Παλινδρόμηση, 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

Data has become a key input for driving growth, enabling businesses to differentiate themselves and maintain a competitive edge. Given the growing importance of data to companies, should managers measure its value? Is it even possible for a company to effectively measure the value of its data? An increasing number of institutions, academics, and business leaders have begun tackling these questions, leaving managers with many alternatives for assessing the value of data. None are yet generally accepted, nor completely satisfactory, but they can help organizations realize more value from their data [AkSa18].

In today's data-driven landscape, machine learning has emerged as a pivotal component in the realm of data profiling systems. With the exponential growth in data generation and collection, businesses are seeking to extract meaningful insights and actionable intelligence from their datasets. Machine learning, a subset of artificial intelligence, has proven to be a powerful tool in this pursuit. It offers the capability to sift through vast quantities of data, identify patterns, and make predictions or recommendations based on historical and real-time information. This fields' algorithms can automatically uncover hidden correlations, segment data into meaningful clusters, and predict future trends, all of which contribute significantly to data profiling. By applying machine learning techniques, organizations can gain a deeper understanding of their data, identify outliers and anomalies, and optimize their data management strategies. This not only enhances the quality and accuracy of data but also contributes to better decision-making processes, cost-efficiency, and competitive advantage in the market.

In the context of the current thesis, the machine learning techniques that are chosen to be new features to the profiling system Pythia are regression and clustering. Pythia is a data analysis software, implemented in Java and based on the integrated Apache Spark engine, that comes to apply the analysis algorithms efficiently on a large amount of data. Pythia accepts a dataset as input, processes it, and automatically produces valuable information about it. The Pythia system can detect hidden patterns inside the data, calculate statistics,

and even produce decision trees. All this is done automatically, and the results are presented to the analyst through a detailed report. By introducing clustering techniques, the profiling system gains the ability to automatically group similar data points together, facilitating a deeper understanding of data structures and patterns. This not only aids in the identification of data outliers but also enhances the process of segmenting and categorizing data, which is instrumental in various applications, such as customer segmentation for recommendation systems, anomaly detection, and market trend an analysis. Clustering within a data profiling system provides a powerful means to extract insights that might remain hidden with traditional profiling methods. Furthermore, the inclusion of regression features allows the development of predictive models within the profiling system. Regression analysis enables the system to make data-driven forecasts, estimate relationships between variables, and provide insights into future trends. This is especially significant for businesses aiming to optimize resource allocation, forecast demand, and make informed decisions regarding pricing strategies and inventory management. The ability to predict future outcomes based on historical data is a game-changer in terms of strategic planning and competitiveness.

When a data profiling system is enhanced with machine learning techniques new kinds of problems occur and most of them have not a proven solution. For example, in the clustering methods, the major problems are the initialization criteria of the algorithm's parameters, the choice of the number of clusters, and the presence of outliers. The number of clusters is often the most difficult to determine using automated methods. As the number of clusters is not known a priori, it may sometimes be desirable to use many clusters than the analyst's "guess" about the true natural number of clusters in the data. This will result in splitting some of the data clusters into multiple representatives. In addition, cluster validation is difficult in real datasets because the problem is defined in an unsupervised way. Therefore, no external validation criteria may be available to evaluate a clustering. Thus, a few internal validation criteria may be defined to evaluate the quality of the clustering [Agga15].

While regression analysis undeniably offers valuable insights by identifying relationships between variables, it's crucial to acknowledge its inherent limitations. The primary challenge lies in distinguishing correlation from causation. Establishing a statistical connection between two variables does not necessarily imply that one directly influences the other: there might be confounding factors at play. So, it still takes critical thinking and careful studies to locate meaningful cause-and-effect relationships in the world. But at a minimum, regression analysis helps establish the existence of connections that call for closer investigation [Dizi10]. Nonetheless, while regression analysis may not provide

conclusive answers regarding causality, it plays a pivotal role in hypothesis generation and identification of potential relationships that warrant in-depth scrutiny.

Overall, Pythia is extended with 2 major abilities. It can now predict future values for some of the columns of the datasets via regression analysis and divide the dataset into groups of entities with various clustering algorithms. All these are implemented based on Apache Spark in order to avoid the implementation of the algorithms as they have trivial implementation this is not the point of this thesis. The experiments and testing conducted have demonstrated that Apache Spark performs admirably in handling the computational demands of regression analysis and clustering algorithms, affirming its suitability for supporting Pythia's enhanced functionalities.

1.2 Thesis structure

The thesis consists of 4 more chapters in which the contributions are presented extensively.

Chapter 2 describes the current state of the Pythia system and the goal of this thesis to enhance it. In this chapter, the scientific background that is required for someone to understand this thesis is also described, and mainly it has to do with terms of data science and machine learning. Lastly, this is the chapter where the technologies that are used are presented.

Chapter 3 is the heart of the thesis. It presents the technical definition of the problem that the Pythia system must deal with and the list of algorithms that are chosen to solve this problem. In this chapter, the design of the software architecture is discussed as well as the testing that guarantees the correct functionality of the final code. In Chapter 3, installation and implementation details are also described along with the maintenance and scalability of the software.

Chapter 4 analyzes the experimental evaluation of the system in terms of its extensions. The results of the experiments are presented in detail, in the form of both table and graphic representations.

In chapter 5, a synopsis of the enhancement of the system Pythia is described along with some conclusions about the importance of this contribution. Chapter 5 also presents some suggestions for future extensions of the system. In the end of the document, there are references to all bibliography, articles, and web links there were used during the implementation of the thesis.

Chapter 2. Subject Description

2.1 Thesis objective

The objective of this thesis is to extend the Pythia system with regression and clustering features. The main objective of the Pythia system is to allow an analyst to obtain valuable insights of a dataset in an automated manner. The system accepts a dataset as input and generates a detailed statistical profile with insights of the data set [Anton23].

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

- The first significant extension to the Pythia system is the integration of a regression feature. With this addition, users are empowered to select a specific column from the dataset as the dependent variable, for which they seek future value predictions. Additionally, they can choose one or more columns as independent variables, based on which the regression model will make these predictions. This flexibility allows analysts to tailor the regression analysis to their specific research questions and objectives. Upon initiating the process, the Pythia system will perform regression analysis, establishing relationships between the chosen dependent and independent variables. This extension aims to automate the otherwise complex and time-consuming task of conducting regression analysis. The Pythia system will generate a comprehensive report, complete with statistical metrics, regression coefficients, and graphical representations. These visual aids assist the analyst in understanding the data's underlying patterns and relationships. The report will serve as a valuable output, offering insight into the predictive capabilities of the regression model. In addition to providing prediction results, the reliability and quality of the regression model will also be evaluated. The analyst will be informed of how well the model fits the data, and metrics like R-squared and Mean Absolute Error will be included in the report to assess the accuracy and precision of the predictions.
  The inclusion of the regression feature in the Pythia system is a significant step forward in enabling data analysts to harness the power of regression analysis for informed decision-making. This enhances the system's automation capabilities, making it a more versatile and user-friendly tool for data exploration and hypothesis testing.

- Another pivotal augmentation to the Pythia system is the incorporation of clustering features. The Pythia system will support various clustering algorithms, granting analysts the ability to select the most appropriate method to their specific analysis. The clustering feature will offer analysts the flexibility to choose from an array of clustering techniques, such as K-Means, Hierarchical, and DBSCAN, among others. This flexibility is invaluable, as different clustering algorithms are better suited for specific types of data or research question. Analysts can select the clustering method that aligns with their objectives and the nature of their dataset.

  Following the selection of clustering parameters, the Pythia system will proceed to perform the clustering analysis. It will generate a comprehensive report encompassing the clustering results. The report will include visual representation of the clustered data. Additionally, descriptive statistics for each cluster, including means, variances, and sizes, will be provided to offer insights into the characteristics of each group.

  An essential aspect of the clustering feature is the evaluation of the clustering quality. The Pythia system will employ various internal and external clustering validations metrics to assess the quality and coherence of the generated clusters. These metrics will assist the analyst in determining the suitability of the chosen clustering approach for their dataset.

  The inclusion of the clustering feature in the Pythia system represents a substantial enhancement, empowering analysts to gain deeper insights into their data by uncovering inherent structures and patterns. This addition streamlines the process of clustering analysis, automating a task that can be extremely complex. The comprehensive report, including visualizations and validation metrics, will equip the analyst with the necessary tools to make informed decisions based on clustering results.

2.2 Background

In this subject, the background required to understand the thesis is presented in detail.

2.2.1 Datasets – Multidimensional data

Datasets serve as the fundamental building blocks of data analysis and play a pivotal role in numerous scientific and business domains. A dataset, in its essence, is a structured collection of data points or observations, often organized into rows and columns. Each row typically represents an individual entity or observation, while columns, also referred to as attributes or features, delineate specific characteristics or measurements associated with each entity. Datasets come in various forms, ranging from simple tables in tabular data to more complex structures, including times series data, images, text and other. Currently, Pythia accepts datasets in the form of csv files.

A multidimensional dataset is a specialized form of dataset where observations are defined not only by individual attributes but also by their relationships with respect to multiple dimensions. This type of data often arises in scientific experiments, geographical information systems, sensor networks, and various other contexts where measurements depend on several independent variables. For instance, in a climate study, multidimensional data might encompass spatial dimensions (latitude and longitude), temporal dimensions (time), and various atmospheric attributes (temperature, humidity, pressure). These multiple dimensions result in data points existing in a multi-dimensional space, which can be challenging to explore and comprehend using conventional data analysis techniques.

Understanding datasets and multidimensional data is imperative for the ensuing chapters of this thesis. These data structures form the foundation upon which various data profiling, regression, and clustering techniques are applied. Therefore, a clear grasp of the concepts of surrounding datasets and multidimensional data is essential for comprehending the methodologies and results presented in the subsequent sections of this thesis.

2.2.2 Pythia

As it was mentioned above, the Pythia system, in its state prior to this thesis, is relatively limited to assessing the quality of the given data set and extracting some hidden information from the data. The end goal of the system is to facilitate fast automated Exploratory Data Analysis (EDA) [Agga15], by generating a valuable insights overview of a given dataset. The system can be easily imported as a dependency into other Java projects as it is developed in Java and is distributed in a JAR package. Pythia utilizes the Apache Spark engine to quickly process the data set [Anton23].

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

- **Data set loading**: The system can accept a data set via programming to declare the fields of the data set and their type respectively (integer, double, Boolean, etc.).
- **Descriptive statistics**: Any dataset 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 datasets is the correlation between columns. Pythia can calculate the correlation between all columns of the given dataset. That is a feature that will concern us in this thesis due to regression.
- **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.
- **Dominance Patterns**: Those patterns break down to two categories: Low-Dominance Patterns and High-Dominance Patterns. For each category a separate detailed report is generated. This feature is explained later as it belongs to a part of a system that has been refactored with outlier detection.
- **Outlier Detection**: Pythia supports outlier detection by calculating the z-score for each data point and setting a threshold.
- **Reporting**: After all the data processing is completed, the analyst can export the results in plain text or JSON format. JSON (JavaScript Object Notation) is an easy-to-understand format to transfer and display data [Orac22]. It is widely used in many fields of computer science and especially in web technologies.

The user can choose which of the above features he wants to be applied to the data he gives to Pythia via client classes. The interface of Pythia is explained in the next Chapter.

2.2.3 Regression

Regression is a statistical method that attempts to determine the strength and character of the relationship between one dependent variable and a series of other variables (known as independent variables) [BeePot23].

There are various types of regression. Some of them that are important in data analysis are the following:

- **Linear Regression:** This regression type is based on the general linear model $y = ax + b$. So, considering x the independent variable and y the dependent variable

we can predict the y values given the x. The only problem is to determine the model's parameters a, b. Most of the time this happens by finding the min value of the SSE (sum of squared errors) function [BoWa08].

$$SSE = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

In this function we use $\hat{y}_i$ which is the mean of the y_i values. The solution at finding the min value arises by taking the partial derivatives of SSE with respect to both coefficients (a and b), setting them equal to zero. We calculate a and b from the resulting equations.

- **Multiple Linear Regression:** This is the type of regression that is used when we have more than one of independent variables X_1, X_2, …., X_n. To predict future values of the dependent variable Y given the values of the independent variables the new model is:

$$Y = \beta + \alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n$$

Once again, the goal is to minimize the value of the SSE function in order to find the values of a_1, a_2, …, a_n [BoWa08].

- **Polynomial Regression:** In real world problems, a linear relationship between the dependent and independent variables does not always exist. Thus, a polynomial model is necessary so it can have curves that will fit better to the dataset relationships. This regression is based on the following model (consider Y the dependent variable and X_1 the independent) [Agra23]:

$$Y = \beta + \alpha_1 X + \alpha_2 X^2 + \cdots + \alpha_n X^n$$

2.2.4 Clustering

Clustering is about separating the dataset into cohesive groups. To achieve good clustering, each group should have entities that are 'similar'. There are various kinds of clustering techniques to follow. The 4 families for clustering are:

- **Representative-based Algorithms (flat algorithms)**: In this algorithmic family each cluster has a representative. The goal is to minimize the loss function O, meaning that the sum of the distances of the different data points to their closest representative needs to be minimized.

$$O = \sum_{i=1}^{n} \left[min_j Dist\left(\bar{X}_i, \bar{Y}_j\right) \right]$$

The representatives can be selected from the dataset (not optimal). The algorithmic issue is to find the optimal representatives that are unknown a priori. This algorithm has 2 steps: assign step (assigning each data point to its closest representative using a distance metric) and optimize step (select new optimal representative for each cluster). These algorithms stop after a max number of iterations, or better, when the loss function has reached an optimal threshold. The main differences between the algorithms of this family are the distance criteria. Some famous algorithms of this kinds are [Agga15]:

1. *K-means:* Uses Euclidian or Mahalanobis Distance and means as representatives.
2. *K-medians*: Uses Manhattan Distance and medians as representatives.
3. *K-medoids*: Keeps a set S with all the representatives and updates it with optimal comparisons.

- **Hierarchical Algorithms:** In this algorithmic family there are 2 categories [Agga15]:
    1. *Bottom-up (agglomerative) methods*: We begin with a cluster for each entity. In each iteration we merge 2 clusters, so that the total number of clusters is reduced by 1. In order to choose which clusters will be merged a matrix is used that is updated after each agglomeration. This matrix is 2-dimensional and holds the distance between all groups. Some of the distances that can be used for this purpose are:
        a. ***Best (single) linkage*:** Minimum distance between all pairs.
        b. ***Worst (complete) linkage*:** Max distance between all pairs.
        c. ***Group average linkage*:** Average distance of all possible pairs.
        d. ***Closest centroid*:** The closest centroids are merging their clusters.
    2. *Top-down (divisive) methods:* These methods use the flat clustering algorithms as a subroutine. These methods start with all the entities as one cluster and during division it creates a top-down tree. In each iteration the 'tree' split into more nodes (new clusters). The advantage here is that the tree structure is better controlled. More detailed, in each iteration the following things happen:
        a. *The algorithm selects which node to extend based on pre-defined criteria.*
        b. *Then uses a flat algorithm to divide the selected node.*
        c. *Adds the new nodes as children of the selected node.*

A very famous top-down method is the Bisecting K-means that uses the invocation of a 2-means algorithm as subroutine.

- **Grid-based Algorithms:** These algorithms involve dividing the data space into a finite number of cells, creating a grid-like structure that simplifies the clustering process. Data points are assigned to specific cells based on their attribute values, enabling the algorithms to efficiently identify regions with high data density. By minimizing the computational complexity through the grid-based approach, these algorithms offer an effective means of handling large-scale and high-dimensional datasets, making them particularly useful for tasks that require fast and scalable clustering solutions. An advantage of these methods is that the number of clusters must not be declared as the algorithm finds it. A very well-known grid-based clustering algorithm is DBSCAN [Agga15].
- **Graph-based Algorithms:** In these methods, the first thing that is done is the encoding of the dataset into a graph. In order to do that, some problems occur, such as which distance function will be used to find the graph's edges' weights. Ater this step the only thing left is to apply a community-detection algorithm to the graph and as a result the detected communities represent the clusters of the dataset.

2.2.5 Apache Spark

Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters [ApSpa23]. Spark provides a unified and comprehensive framework for processing large-scale data sets with speed and efficiency. One of the key features of Apache Spark is its in-memory data processing, which enables it to cache data in memory, thereby significantly accelerating iterative algorithms and interactive data analysis. This capability makes Spark particularly well-suited for applications that require fast and iterative processing, such as real-time data analytics, ETL (Extract, Transform, Load) workflows, and complex data manipulation tasks. Its ability to distribute data processing tasks across a cluster of machines in a fault-tolerant manner further enhances its scalability and reliability, making it a popular choice for handling big data workloads.

Moreover, Apache Spark offers a rich ecosystem of libraries and tools that cater to various data processing and analytics needs. Very famous components of Apache Spark are Spark SQL, Spark Streaming, MLlib and GraphX. Its versatile nature and extensive library support make Apache Spark a powerful tool for a wide range of data-centric applications,

cementing its position as a leading big data processing framework in the industry of data management.

The components of Apache Spark are explained in the following list:

- **RDD (Resilient Distributed Dataset)**: RDDs are an immutable, resilient, and distributed representation of a collection of records partitioned across all nodes in the cluster. In Spark programming, RDDs are the primordial data structure. Datasets and DataFrames are built on top of RDD [Santia23].

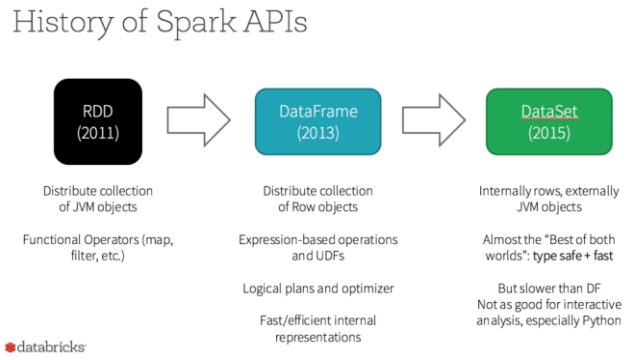


Figure 1. Evolution of Spark APIs [databr23].

- **Spark SQL:** One of the most common data processing paradigms is relational queries. Spark SQL implements such queries on Spark, using technologies similar to analytical databases. For example, these systems support columnar storage, cost-based optimization, and code generation for query execution. The goal is to use the same data layout as analytical databases inside RDDs. In Spark SQL, each record in an RDD holds a series of rows stored in binary format, and the systems generate code to run directly against this layout [].
- **Spark Streaming:** This component implements incremental stream processing using a model called 'discretized streams". To implement streaming over Spark, we split the input data into batches (such as every 200 milliseconds) that we regularly combine with state stored inside RDDs to produce new results. Running streaming computations this way has several benefits over traditional distributed streaming systems. For example, fault recovery is less expensive due to using lineage, and it is possible to combine streaming with batch and interactive queries.
- **MLlib:** This is Spark's Machine Learning library which serves a crucial component within its ecosystem. Its primary objective is to enable scalable simplified machine learning, thereby making complex data analysis more accessible and

efficient. At its core, MLlib offers a comprehensive set of tools and functionalities that cater to diverse machine learning requirements. These include an array of common algorithms for tasks such as classification, regression, clustering, and collaborative filtering. Additionally, MLlib provides robust support for various aspects of data preprocessing, including feature extraction, transformation, dimensionality reduction, and selection. Its built-in tools for constructing, evaluating, and tuning ML Pipelines allow for streamlined and efficient development of machine learning workflows. MLlib also offers seamless persistence options, enabling users to save and load algorithms, models, and Pipelines effortlessly. Furthermore, MLLib encompasses a range of utility functions for tasks such as linear algebra, statistics, and data handling, solidifying its position as a complete and user-friendly library within the Apache Spark environment. The enhancement of the Pythia system is based on this library [ApSpa23].

- **GraphX**: This is a graph processing framework for big data. It is used for analyzing and processing large graphs in a distributed fashion. GraphX can solve various data science problems, including machine learning, social network analysis, recommendation systems, and bioinformatics [Simpli23]. GraphX extends the Spark RDD by introducing a new Graph abstraction: a direct multigraph with properties attached to each vertex and edge. It is one of the latest components in Spark and supports graph-parallel computation [ApSpa23].

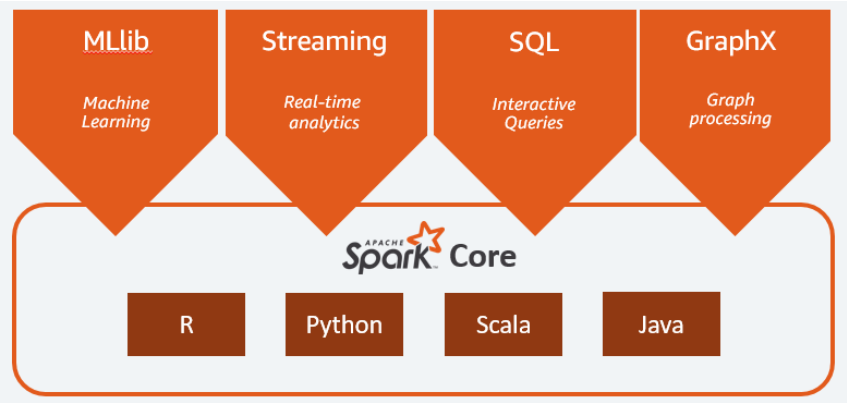


Figure 2. Main Libraries of Spark [Amaz23].

Chapter 3. Design & Implementation

This chapter offers a comprehensive glimpse into the Pythia dataset profiling system's software architecture and execution flow. A detailed exploration of the software design and implementation related to the automated highlight identification issue is presented. To enhance understanding, UML class and package diagrams are included.

3.1 Refactoring the existing system

Before designing the additions related to Machine Learning there were some issues with the semantic integrity of the Highlight Patterns Identification feature of Pythia. More specifically it was noticed that this functionality was hiding two operations that were different enough to be considered different 'use cases'. Also, the name 'Highlight Patterns' was not representative of those features, so it was eliminated.

The first feature is Dominance Patters Identification. 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 [Anton23]. The part of the system that is responsible for the functionality of this feature stays the same during the implementation of this thesis. The only thing that changes is the names of the methods and fields that used to arise from the term 'Highlight' when, in reality, they were parts of the system that had to do with the term 'Dominance'. Overall, this was a refactoring about naming.

The second feature that needed to be refactored was Outlier Detection. Outlier Detection is a process that finds values that are so strange that it's like they should not belong in the dataset. Pythia supports Outlier Detection for each column of the dataset that is analyzed based on z-score. More specifically, Pythia assumes that the values of each column come from the Gaussian Distribution and calculates the z-score, for each value, which represents how many standard deviations an observation is away from the mean of the distribution [WaLMo21]. There are two reactors that affect the Outlier Detection process.

The first reactor has to do with the fact that the term 'Highlights' should not exist and therefore as a process Outlier Detection should be independent of Dominance Patterns.

Necessary changes in the system were made in order to achieve this, and now the user can choose if he wants Dominance Patterns Identification or Outlier Detection to happen during the data analysis. Of course, he can choose both features.

The second refactoring has to do with the extension of Outlier Detection with more than 1 algorithm. The users of Pythia are supposed to have knowledge of Data Analytics, so they will probably expect from a tool like Pythia to give them the convenience to choose the algorithm to detect outliers. As mentioned, Pythia at its previous state supported Outlier Detection only based on z-score. In order to parametrize the algorithm that detects outliers, the strategy pattern is followed. Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable [ReGu23]. Below there is a UML diagram of the package that contains all the classes related to the strategy pattern. This figure is just an insight to make clear the refactoring that happened. The new code of Pythia is completely analyzed in the following sections.

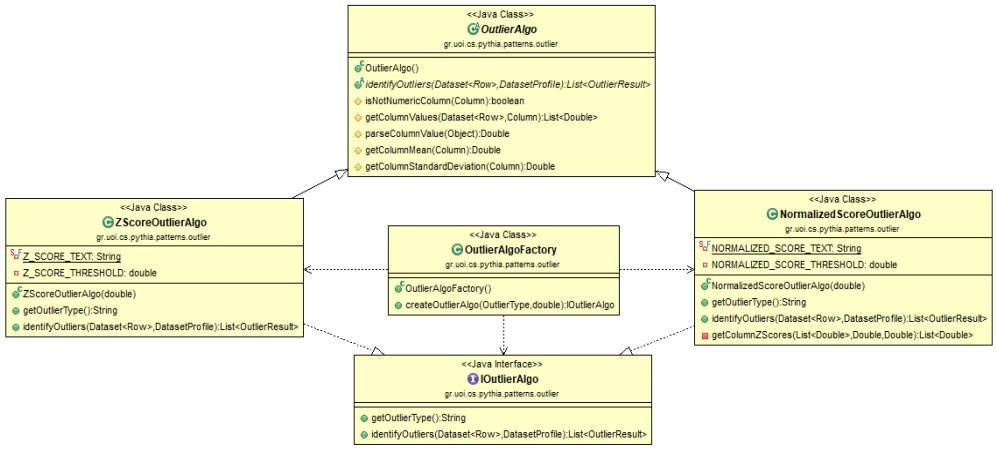


Figure 3. UML for the classes of package 'patterns/outlier'.

The pattern is pretty simple. Before explaining the package, it should be mentioned that the normalized version of z-score is the alternator that was added the standard z-score. Also, in each case, a threshold is necessary which when exceeded by the corresponding score the value is considered an outlier.

The API of this package is the method identifyOutliers() that is contained in the interface 'IOutlierAlgo', so an object of this type is used to implement this functionality. The object is initialized with the createOutlierAlgo() method of the factory 'OutlierAlgoFactory'. This method will determine the type of score that will be measured for each value and the threshold. So the createOutlierAlgo() method returns either a 'ZScoreOutlierAlgo' object or a 'NormalizedScoreOutlierAlgo' object. Both classes implement the 'IOutlierAlgo'

interface. The ‘OutlierAlgo’ class is used to erase the duplicate code that occurred during the development between the two classes that implement the score calculations.

3.2 Problem definition & resolution

Integrating machine learning capabilities, specifically regression and clustering, into Pythia represents a significant advancement in its data profiling capabilities. However, the seamless integration of such complex features demands careful consideration and a structured approach. Building a robust tool like Pythia requires a systematic framework to ensure not only the successful incorporation of new features but also the preservation of the tool's efficiency and reliability. Pythia currently operates through a well-defined process of four standard steps during the analysis of user requests. As we embark on enhancing Pythia with regression and clustering functionalities, it is imperative to establish a standardized framework that harmonizes these advanced features with the existing architecture. This chapter delves into the phases associated with extending Pythia's capabilities, highlighting the need for a cohesive structure to accommodate novel features while maintaining the tool's integrity and user-friendly design.

Each phase within Pythia’s data analysis pipeline is organized into dedicated sub-packages within its software architecture, following a structured approach [Anton23]. The process comprises the following key steps:

1. **Data Preparation**: This initial phase is optional and occurs before engaging the identification algorithm. It involves tasks such as querying the dataset and selecting pertinent columns to be passed through the identification algorithm.
2. **Highlight Identification Algorithm**: After completing data preparation, the selected dataset undergoes processing by the highlight identification algorithm. This algorithm examines the data, producing results that indicate the presence or absence of highlight patterns. Some algorithms offer multiple variations, necessitating the execution of all possibilities to generate respective results. To manage potentially voluminous outcomes, certain algorithms incorporate mechanisms like Top-K filtering, limiting the displayed results to the most significant.
3. **Highlight Identification Result Object(s)**: For each algorithm execution, the corresponding results are stored in distinct, concrete result objects. Beyond highlighting presence or absence, these result objects may encompass additional information, such as the highlight's location in the dataset, scoring for the

implicated data, the executed algorithm's variation, and the involved measurement and coordinate columns. Due to substantial differences among highlight identification algorithms, each algorithm possesses its own result class.

4. **Reporting Mechanisms**: Finally, as data passes through the preceding phases and result objects are generated, the highlight identification results are exported to report files. These files encapsulate all information stored in the result objects, augmented by natural language descriptions elucidating the identified highlights.

All these intricacies are seamlessly orchestrated by Pythia, underscoring its role as an automated data analysis pipeline.

Furthermore, to bolster the usability, extensibility, and scalability of the system, Pythia incorporates a flexible parameterization feature. This enhancement empowers analysts to optimize the automated data analysis pipeline according to their specific needs. Upon loading a dataset, analysts can declare input parameters to guide the execution of Pythia. These parameters allow users to pinpoint which segments of the data analysis pipeline should be activated, as well as identify specific points of interest—such as particular columns—pertinent to highlight pattern identifications.

Beyond its foundational capabilities, Pythia currently boasts a repertoire of sophisticated features, elevating its analytical prowess. The system seamlessly provides statistical profiling, leveraging its ability to distill essential statistical summaries for datasets. Additionally, Pythia harnesses decision trees to unravel complex relationships within data, enabling insightful decision-making. Furthermore, the system excels in outlier detection, pinpointing data points that deviate significantly from the norm. Pythia's advanced capabilities extend to the identification of dominance patterns, elucidating hierarchical relationships within datasets.

The implementation of these features adheres to best practices, incorporating the parameterized Factory [Knoe01] software design pattern. This design choice not only ensures seamless integration of the developed highlight extractor modules but also facilitates extensibility. The use of the strategy pattern further enhances flexibility, allowing for the addition of new modules without significant maintenance of the pattern manager class. Following established conventions, the contributed code is written in Java and optimized for rapid data processing using the Apache Spark engine.

Before delving into an intricate exploration of the software design and architecture, let's articulate the system's augmented functionalities by presenting the contributed features in the context of user stories.

[US1] Regression Variety: As a data analyst, I want the capability to perform diverse types of regression analysis on my dataset. This includes polynomial regression, automated regression, multiple linear regression, and traditional linear regression. By having this flexibility, I can tailor my analysis to the specific characteristics of the data and gain a nuanced understanding of the relationships between variables.

[US2] Clustering Proficiency: As a data analyst, I seek the ability to conduct clustering analysis on my dataset. This functionality empowers me to identify inherent patterns and groupings within the data, facilitating the discovery of hidden insights. The system should provide options for selecting appropriate clustering algorithms and configuring parameters to ensure a customizable and insightful clustering process.

[US3] Independent Outlier Detection: As a data analyst, I aim to conduct outlier detection independently from dominance patterns. This requires a refactoring of the system to decouple outlier detection from dominance examination. By doing so, I can pinpoint data points that deviate significantly from the norm, enhancing the precision of my analysis. The system should offer a seamless and independent outlier detection process, allowing for efficient identification and analysis of anomalous data points.

These features are performed automatically once a data set has been loaded and its profile is requested.

3.3 Software design & architecture

As highlighted earlier, Pythia strives to establish the groundwork for a remarkably adaptable platform, simplifying the process of Exploratory Data Analysis (EDA) by extracting valuable insights from diverse datasets. To foster a deeper comprehension of Pythia's inner workings, this section offers a concise glimpse into the software architecture. Subsequently, it provides a detailed explanation of the essential components involved in the addition of the Machine Learning techniques as new features.

3.3.1 Execution flow

The standard execution flow of Pythia comprises four major stages. These stages include registering data, declaring parameters, selecting algorithms for execution, computing the dataset profile, and generating the output. This structured framework serves as the backbone of Pythia's data analysis process, ensuring consistency and reliability in its

operations. In this subjectivity we will examine each step, in order to understand the whole process.

- **Register Data:** Firstly, the analyst must provide the input dataset. For now, this happens programmatically by creating a client class. In the class the dataset is registered as a Spark Schema [SpaExa23] and all the columns are labeled with their type.
- **Declare Parameters (& select algorithms)**: After registering the dataset the analyst should determine the algorithms that he wants Pythia to perform and declare their parameters. For each algorithm there are implemented methods that the client can use to declare their parameters.
- **Compute Dataset Profile**: After declaring the necessary parameters, the client executes the method that computes the Dataset Profile. This method knows which algorithms must be executed for the data profiling process based on some Boolean variables that represent if the user chose the corresponding algorithm. There is a Boolean variable for each available feature of Pythia and currently they are the following:
    - Statistical Analysis
    - All Pairs Correlation Computation (for numerical columns)
    - Histograms
    - Decision Trees
    - Dominance Patterns
    - Outlier Detection
    - Regression
    - Clustering

- **Generate output**: To save the output of the data profiling process somewhere in the memory disk, the client calls methods that generate .md and .txt files representing an analytical report about the dataset based on the results of the algorithms executed. For most of the algorithms the .txt files are extended with details while .md files contain a summary of the data profile in a more optically friendly way.

3.3.2 Overall architecture overview

Here we will dive into the architecture of the software by explaining each package. For the packages that were modified or created during this thesis a UML diagram with further explanation will be provided. After examining each one of the packages that Pythia consists of, then the architecture will be crystal clear.

Package 'client': This is the only package that the user of Pythia uses to interact with the system. In order to use Pythia, the analyst should make a class in this package, implementing all the steps mentioned previously in the execution flow section.

Package 'config': 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 propertied file under the src/main/resources directory [Anton23].

Package 'util': Within this package, you'll find classes housing essential methods utilized across different facets of the software. These utility methods play a crucial role in facilitating various operations within the system, contributing to its efficiency and coherence.

Package 'reader': Within this package, you'll find classes dedicated to retrieving and importing input datasets into the system. At its core lies the 'IDatasetReader' interface, serving as a central point for dataset loading operations. Various concrete implementations of this interface enable support for multiple file formats, including JSON, CSV, and TSV, enhancing the system's versatility in handling diverse data sources.

Package 'writer': Within this package reside classes tasked with exporting the loaded dataset back to the disk, particularly useful when analysts wish to preserve modified datasets for future reference. The cornerstone of this package is the 'IDatasetWriter' interface, serving as the primary interface for dataset writing operations. It is implemented by two concrete classes: the 'NaiveDatasetWriter', employing a simplistic line-by-line approach suitable for smaller datasets, and the 'HadoopDatasetWriter', a robust solution leveraging the Hadoop Distributed File System (HDFS) to swiftly export large datasets to disk.

Package 'engine: This package contains the classes responsible for the basic functionalities of the software. It is the 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. It communicates with every other package that implements a data profiling feature. Inside this package there is a class named 'DatasetProfiler' that implements the important method that computes the dataset profile. This method is basically calculating the data profile by using each Pythia feature one by one. For example, if the first thing that was asked to happen was to cluster the data, then inside this method there is another method called that performs this operation. For each feature there is a separate package with a standard architecture that is explained later. The important thing for now is that the API offered from each package is an interface with the method that 'DatasetProfiler' uses to get the result of the feature. Below there is a UML diagram that helps understanding better the usability of the engine package and its connection with the other interfaces. As it can be seen, the 'DatasetProfiler' itself implements an interface.

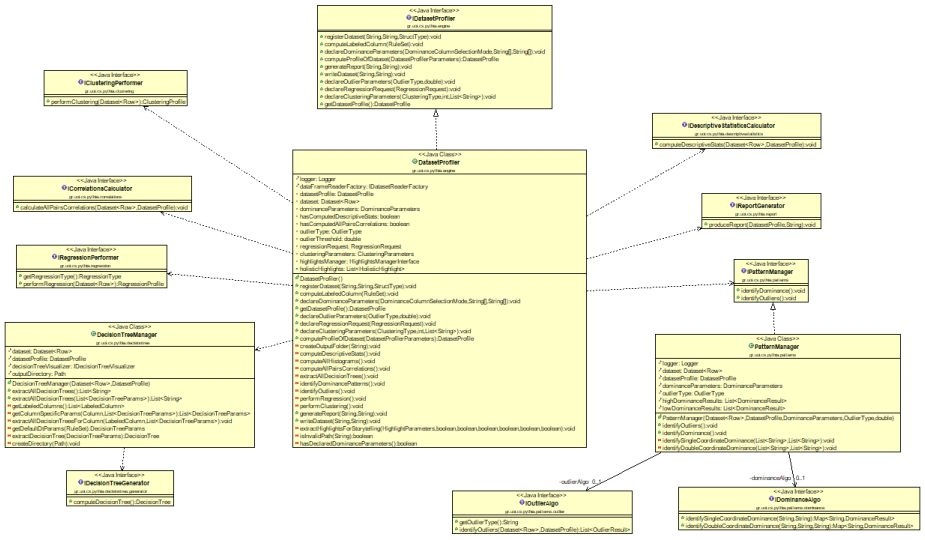


Figure 4. UML for the relations between 'DatasetProfiler' with features/interfaces.

Package 'correlations': The correlations package has the classes responsible for the correlation calculation among all numerical columns of the input dataset based on Pearson's correlation. It implements the parametrized Factory pattern, so it's easy to enhance this package with more than one implementation. Almost each one of the packages representing a feature is based on the parametrized Factory pattern. The main interface of the package is named 'ICorrelationsCalculator'.

Package 'regression': This package consists of each class necessary to implement the regression feature. This package was created during this thesis, so it is further explained later. The main interface of the package is named 'IRegressionPerformer'.

Package 'clustering': This package consists of each class necessary to implement the clustering feature. This package was created during this thesis so it is further explained later. The main interface of the package is named 'IClusteringPerformer'.

Package 'labeling': 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. The' RuleSet' class is responsible for the creation of a complete expression based on which a new labeled column is generated.

Package 'model': This package encompasses domain classes essential for storing data utilized or generated by the system. At its core is the 'DatasetProfile' class, serving as the top-level domain entity. It encapsulates various other domain classes within the package, each corresponding to different aspects of data analysis. In the context of this thesis, the package is expanded with the 'PatternsProfile' class, specifically designed to manage data related to highlight patterns identification.

Package 'report': This package houses functionalities related to generating comprehensive reports containing analysis outcomes, ranging from descriptive statistics to decision trees and highlight patterns. Notably, for the purposes of this thesis, modifications were made to export each highlight pattern result into separate report files. All generated report files are conveniently organized within a unified directory structure.

Package 'descriptivestatistics': This package hosts classes dedicated to computing descriptive statistics for datasets. The central interface, 'IDescriptiveStatistics', supports automated calculation of mean, median, standard deviation, minimum, maximum, and various other values for all dataset columns, including non-numerical data.

Package 'desiciontree': This package encapsulates classes and sub-packages responsible for extracting and visualizing decision trees. At its helm is the 'DecisionTreeManager' class, overseeing input parameters, data preparation, and decision tree generation and visualization. A comprehensive description of this package can be found in the "Automated Extraction of Decision Trees in a Data Profiling System" [Char23] thesis.

Package 'patterns': This package is responsible for the Dominance Patterns Identification but also for the Outlier Detection. As it contains two different jobs within itself it a part of out future work to split this package to two different packages 'dominace' and 'outlier'. For now, those packages are inside the patterns package so it gives to the overall UML diagram an extra complexity that should not exist.

Package 'histogram': Within this package reside classes dedicated to computing histograms for numerical columns of datasets. The top-level manager class,

'HistogramManager', orchestrates histogram calculations and operations. Further details about this package can be found in the "Automated Extraction of Decision Trees in a Data Profiling System" [Char23] thesis.

3.3.3 Regression implementation

As it is already mentioned, this feature was entirely implemented during the thesis. The steps of execution flow can help us understand which steps someone should follow to add a new feature like regression in the system. Before anything else is explained, the UML diagram for the classes inside the regression package is presented. It is observed on the top of the diagram the 'IRegressionPerformer' interface which is used to communicate with 'DatasetProfiler'.

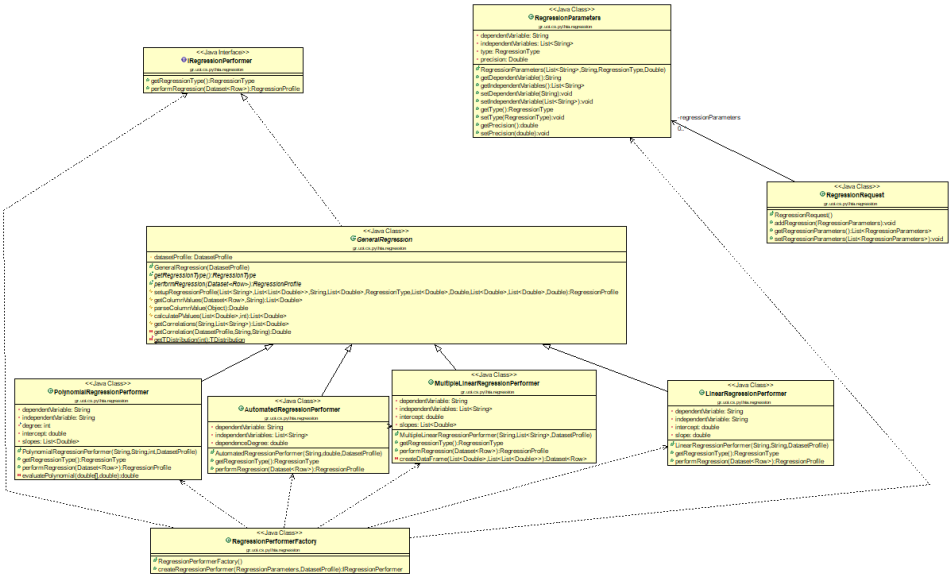


Figure 5. UML for the regression package.

Firstly, the analyst declares the parameters of Regression so the corresponding entity should exist. The class responsible for storing the declared parameters is the class 'RegressionParameters'. Let's break down each one of the parameters necessary to determine a specific regression process:

- **Type:** The type of the regression to be computed. Available regression choices for Pythia are linear, multiple linear, polynomial, and automated.
- **Dependent variable:** The dependent variable of regression.
- **Independent variables:** The independent variables of regression. In case of linear and polynomial it's only one.
- **Precision:** This is a constant that presents the power in case of polynomial regression and p-value threshold in case of automated regression.

The next step is to create the class 'RegressionRequest'. By using objects of this class, the user can request as many regressions performances as he wants, in a single call. All he has to do is to declare parameters for each one of the requests. So, all 'RegressionRequest' needs to know is a list of objects of type 'RegressionParameters'. With these classes, the interaction of regression feature with the user is closed.

The next step is to implement regression algorithms. Based on the parametrized factory pattern, the first thing is to create the class 'RegressionPerformerFactory' that initializes the interface object inside 'DatasetProfiler' based on the given parameters. The interface is implemented by a class named 'GeneralRegressionPerformer' that is used to reduce duplicate code between child-classes and simplify the UML diagram.

The final step is to implement each variation of regression in concrete classes extending 'GeneralRegressionPerformer'. Linear regression, multiple linear regression and polynomial regression are implemented using Apache library. The automated variant was implemented to give a solution to the analyst in case he does not know the independent variables a priori. This is how it works:

Automated Regression:

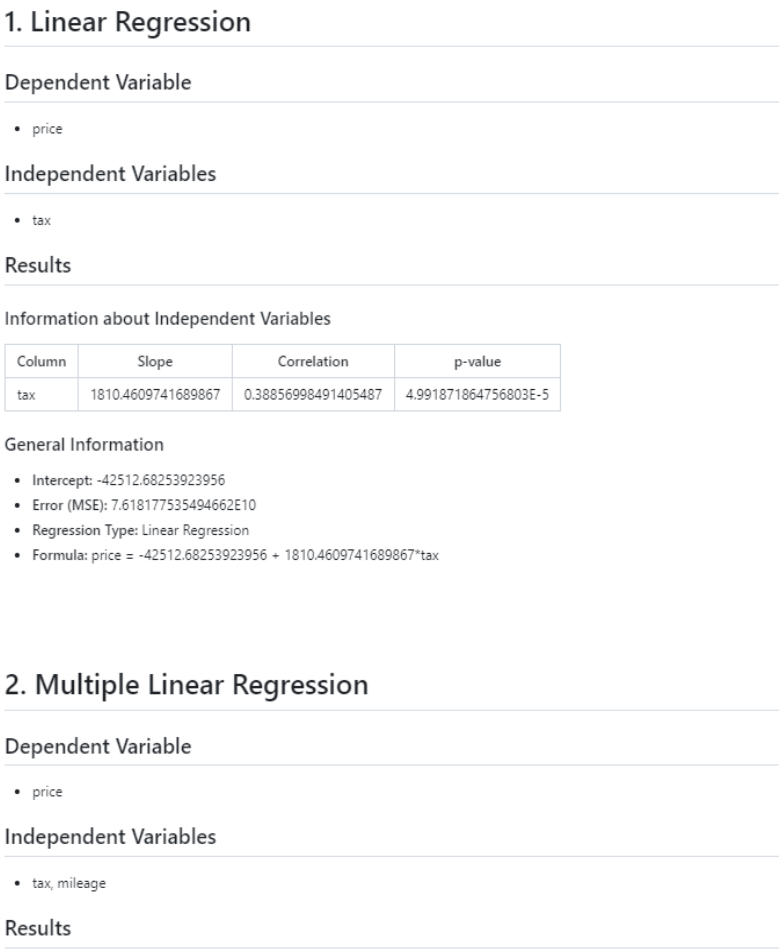
1. Calculate all correlations between dependent variable and the other columns.
2. Calculate p-value for each one of those correlations with the null hypothesis being that those two columns are not correlated.
3. For each candidate column:
    a. If p-value is under the determined threshold then add it as independent variable.
4. Perform multiple linear regression with the chosen independent variables.

The results after a regression execution are saved in an object of type 'RegressionProfile'.
The results kept for a single regression execution are the following:

- **Name and values of the dependent variable.**
- **Names and values of the independent variables.**
- **Type of regression.**
- **Slopes of the output function.**
- **Intercept of the output function.**
- **Regression error.**
- **List of correlations of dependent variable with all the independent variables.**
- **List of p-values of dependent variable with all the independent variables.]**

After a multiple regression request by the analyst, a list of 'RegressionProfile' objects occurs for each regression requested. This list is saved as a field to a global object of type 'DatasetProfile'. 'DatasetProfile' is a class that holds all the necessary information to compute outputs or to save outputs. Each feature saves its output somehow in the global 'DatasetProfile' object. This happens because the 'DatasetProfile' is the only accessible object for the final stage of the execution flow.

The last thing remaining in the execution flow of regression is the production of the report. Both .md and .txt hold the same information. For each regression execution there is a concrete section in the report showing all the information the analyst needs to know. Below there is a screenshot of the beginning of possible report that can be produced:



1. Linear Regression

Dependent Variable

- price

Independent Variables

- tax

Results

Information about Independent Variables

Column	Slope	Correlation	p-value
tax	1810.4609741689867	0.38856998491405487	4.991871864756803E-5

General Information

- **Intercept:** -42512.68253923956
- **Error (MSE):** 7.6181775354946 62E10
- **Regression Type:** Linear Regression
- **Formula:** price = -42512.68253923956 + 1810.4609741689867*tax

2. Multiple Linear Regression

Dependent Variable

- price

Independent Variables

- tax, mileage

Results

Figure 6. Regression Report.

The first request was a linear regression, and the second request was a multiple linear regression. The output of the second request is not fully visible by the screenshot.

With the production of the report, the execution flow of regression comes to an end.

3.3.4 Clustering implementation

Clustering was the second feature that was introduced during this thesis. The logic behind implementing clustering feature is quite similar to the regression one. There in an interface named 'IClusteringPerformer' which is responsible for the communication with 'DatasetProfiler' and a class named 'GenerealClusteringPerformer' implementing it. Again, the general class exists to avoid duplicate code between child-classed (like the computation of the clustering error) and make simpler the UML diagram that occurs.

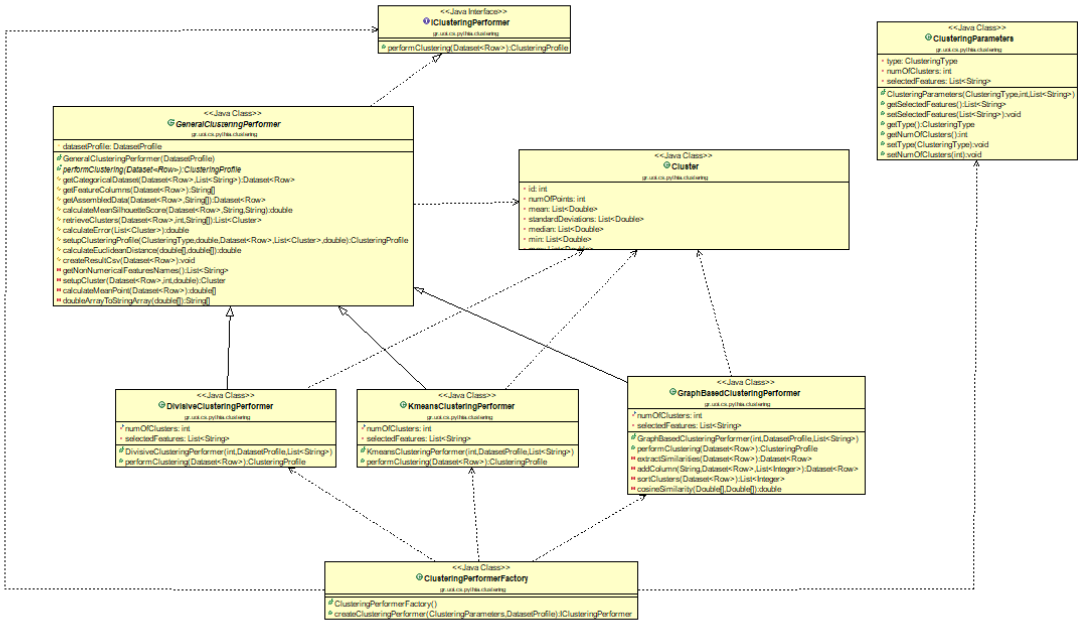


Figure 7. UML for the clustering package.

As defined by the execution flow, the first thing to do in the clustering process is to declare parameters. The parameters for clustering are stored in an object of type 'ClusteringParameters' and in contrast with regression only one clustering request can be asked by the analyst in a single data profiling computation. The parameters that need to be determined are the following:

- **Type:** The type of clustering to be performed by the system. Available clustering choices for Pythia are K-Means, Top-Down Clustering, and Power Iteration Clustering [LinCo10].
- **Number of clusters:** The number of clusters/data groups.
- **Selected Features:** The columns that will be considered during the clustering process. If a column is neither numerical nor selected, then it does not contribute to the clustering computation.

After declaring the parameters, the next step is to compute clustering, so the parametrized Factory pattern is used again. The class 'ClusteringPerformerFactory' initializes an object of type 'IClusteringPerformer' inside the 'DatasetProfiler' based on the clustering parameters. The 'GeneralClusterngPerformer' class is extended by 3 variants:

GraphBasedClusteringPerformer: This class implements the Power Iteration Clustering (PIC) using the Mlib of Apache Spark. PIC is a graph-based clustering algorithm, meaning that the first step is to transform the dataset to a graph and then extract the clusters from the generated nodes.

Graph Based Clustering:

1. *Create graph:*
    a. *Create an array with 3 columns representing the graph.*
    b. *For each possible pair of rows from the origin dataset add the following row to the graph-array: [id of row1, id of row2, cosine similarity of row1 and row2].*
2. *PIC applies a power iteration method like the one used in eigenvector computations. It iteratively applies the following steps:*
    a. *Initialize a random vector r as the initial cluster indicator vector.*
    b. *Update the cluster indicator vector using the power iteration method, which involves multiplying the affinity matrix A with the current cluster indicator vector r.*
    c. *Normalize the updated cluster indicator vector r to have unit length.*
    d. *Repeat steps b and c until convergence criteria are met (e.g., the change in the cluster indicator vector falls below a threshold).*
3. *After convergence, use the final cluster indicator vector to assign data points to clusters. Typically, the entries of the final cluster indicator vector represent the membership strength of each data point to each cluster. You can threshold these values or use a clustering algorithm to determine the final clusters.*

KmeansClusteringPerformer: This class implements the K-Means algorithm using the Mlib library of Apache Spark. This is the most famous clustering algorithm and belongs to the k-representative algorithmic family, mentioned in the background section of this thesis.

DivisiveClusteringPerformer: This class implements the Bisecting K-Means algorithm using the Mlib library of Apache Spark. Bisecting K-Means is a famous top-down hierarchical clustering algorithm that its logic was explained in detail in the background section.

After clustering is executed, the results are saved in a 'ClusteringProfile' object. Like regression, an object of this type is a field to 'DatasetProfile' so there is access to it while generating the report. The results kept for clustering are the following:

- **Type:** The type of the clustering

- **Error:** The SSE error of the clustering. SSE is the sum of the squared differences between each observation and its group's mean. It can be used a measure of variation within a cluster [StanUni24]. The formula for SSE is:
  - **SSE =** $\sum_{k=1}^{K} \sum_{for\ each\ x_i in\ Ck} \|x_i - \mu_k\|^2$

- **Result:** The input dataset with an extra column at the end named 'cluster' to label each row with the id of the cluster it belongs.

- **Average Silhouette Score:** The Silhouette Score serves as a valuable metric for evaluating the performance of clustering algorithms. It plays a crucial role in assessing the quality and effectiveness of clustering results, particularly in scenarios where clear labels for validation are unavailable. As clustering is inherently an unsupervised learning task, internal validation metrics like the Silhouette Score are indispensable for gauging the appropriateness of clustering outcomes. This metric quantifies the cohesion and separation of data points within clusters, offering insights into the overall well-definedness and distinctiveness of the clusters produced by the algorithm [Educ24]. To compute the Silhouette Score for a dataset, you can follow these steps:
  - **Calculate the average distance:**
    - For each data point i, compute:
      - $\mathbf{a_i}$**:** The average distance of point i to all other points in the same cluster (intra-cluster distance).
      - $\mathbf{b_i}$**:** The average distance of point i to all points in the nearest cluster (inter-cluster distance).
  - **Compute the Silhouette Score for each point:**
    - Calculate the Silhouette Score for each data point i using the formula:
      - **Silhouette Score(*i*) =** $\frac{b_i - \alpha_i}{max(\alpha_i, b_i)}$
  - **Determine the overall Silhouette Score:**

- Calculate the overall Silhouette Score for the clustering result by averaging the individual Silhouette Scores of all points:
  - **Silhouette Score =** $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{Silhouette\ Score}\ (\boldsymbol{i})$

Where n is the total number of data points.

- **List of the clusters:** A list with all the clusters. Clusters are entities of the 'Cluster' class of the system. The objects of this class have the following features:
  - **Id:** Each cluster has its own id. First cluster has id = 0 and the last cluster has id = (number of clusters)-1.
  - **Number of Points:** The number of rows of the input dataset to be clustered.
  - **Error:** The part of the SSE for each cluster. Overall SSE occurs from the sum of each cluster's SSE.
  - **Means:** A list with the means for each column of the cluster.
  - **Standard Deviations:** A list with the standard deviations for each column of the cluster.
  - **Medians:** A list with the medians for each column of the cluster.
  - **All Min Values:** A list with the min values of each column of the cluster.
  - **All Max Values:** A list with the max values of each column of the cluster.

When the 'ClusteringProfile' is completely computed then the execution flow continues with the production of the report. At the beginning of the report there is the title that indicates the type of clustering that has been performed following a brief of the selected clustering technique. Under that there is some general information about the overall clustering. Lastly, for each cluster there is a table will all the calculated statistics for each column. Below there is a screenshot of the beginning of possible report that can be produced:

K-Means Clustering

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a predetermined number of clusters. The algorithm aims to minimize the variance within clusters by iteratively assigning data points to the nearest centroid and updating the centroids based on the mean of the data points assigned to each cluster. K-means is widely used for data exploration, pattern recognition, and segmentation tasks, offering simplicity, scalability, and efficiency in handling large datasets.

Clustering Information

- Number of Clusters: 4
- Error: 6.804371448197215E10
- Average Silhouette Score: 0.8816893414335903

Cluster 1 (with 12 points)

Column	Mean	Standard Deviation	Median	Min	Max
year	2015.8333333333333	1.4668044012461452	2016.0	2014.0	2019.0
price	917400.0	3849.4391563163877	916200.0	912300.0	928000.0
mileage	31739.333333333332	11643.324816729919	32011.0	4000.0	47482.0
tax	165.0	28.20380374088831	150.0	125.0	205.0
mpg	48.724999999999994	2.7290108097990378	47.9	42.8	53.3
engineSize	1.8	0.2954195783503986	2.0	1.4	2.0
manufacturer_index	0.0	0.0	0.0	0.0	0.0
transmission_index	0.6666666666666666	0.49236596391733095	1.0	0.0	1.0
fuelType_index	0.3333333333333333	0.4923659639173309	0.0	0.0	1.0

Cluster 2 (with 69 points)

Column	Mean	Standard Deviation	Median	Min	Max
year	2016.4492753623188	1.3343453107627652	2016.0	2013.0	2019.0
price	9159.898550724638	8002.517771347539	10600.0	500.0	31000.0
mileage	31971.08695652174	21549.837781386952	28324.0	1998.0	97440.0
tax	93.33333333333333	61.64811591669637	125.0	0.0	200.0
mpg	58.542028985507265	9.731912420079036	58.9	33.2	76.3

Figure 8. Clustering Report.

3.4 Software test design & results

Before this thesis, Pythia already had comprehensive test cases covering all its functionalities. To validate the newly introduced features, the established conventions and methodologies of Pythia's development were followed. Specifically, unit tests were developed using the black box testing approach [Alex22].

Black box testing is a software testing technique that focuses solely on the functionality of a system without delving into its internal workings. In the realm of unit testing, black box testing involves crafting test cases that examine the inputs and outputs of a code unit (often a method) without considering its implementation details. Test inputs are fed into the code under scrutiny, and the system is expected to produce predetermined outputs. A successful test yields the expected outcome, while a deviation indicates failure. Throughout this process, the system is treated as a "black box".

To maintain consistency with Pythia's existing framework, tests were implemented using the JUnit 4 framework [JUni21]. JUnit is a popular open-source testing framework tailored for Java, designed to streamline the process of writing, and executing unit tests. Leveraging Java annotations, JUnit offers a plethora of features including test setup, execution, and output assertion.

For the newly contributed features, detailed descriptions of the developed test cases are provided in Figures 9-16, showcasing their validation process.

Test Case	**Correct execution test for Linear Regression.**
Involved user story	US1
Method under test	performRegression()
Class under test	LinearRegressionPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Regression type has been selected a Linear. • Dependent variable has been selected. • Independent variable has been selected.
Input	The loaded data set, dependent variable name, independent variable name.
Output	A RegressionProfile object with the calculated measurements.
Assert that	The generated RegressionProfile object is equal with the expected RegressionProfile object.

Figure 9. Description of the unit test regarding correct execution of Linear Regression.

Test Case	**Correct execution test for Multiple Linear Regression.**
Involved user story	US1
Method under test	performRegression()
Class under test	MultipleLinearRegressionPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Regression type has been selected as Multiple Linear. • Dependent variable has been selected. • Independent variables have been selected.
Input	The loaded data set, dependent variable name, list of independent variables' names.
Output	A RegressionProfile object with the calculated measurements.
Assert that	The generated RegressionProfile object is equal with the expected RegressionProfile object.

Figure 10. Description of the unit test regarding correct execution of Multiple Linear Regression.

Test Case	**Correct execution test for Polynomial Regression.**
Involved user story	US1
Method under test	performRegression()
Class under test	PolynomialRegressionPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Regression type has been selected as Polynomial. • Dependent variable has been selected. • Independent variable has been selected. • The power of the polynomial has been selected as an integer number.
Input	The loaded data set, dependent variable name, independent variable name, the power of the polynomial.
Output	A RegressionProfile object with the calculated measurements.
Assert that	The generated RegressionProfile object is equal with the expected RegressionProfile object.

Figure 11. Description of the unit test regarding correct execution of Polynomial Regression.

Test Case	**Correct execution test for Automated Regression.**
Involved user story	US1
Method under test	performRegression()
Class under test	AutomatedRegressionPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Regression type has been selected as Automated. • Dependent variable has been selected. • Threshold has been selected.
Input	The loaded data set, dependent variable name, independent variable name.
Output	A RegressionProfile object with the calculated measurements.
Assert that	The generated RegressionProfile object is equal with the expected RegressionProfile object.

Figure 12. Description of the unit test regarding correct execution of Automated Regression.

Test Case	**Correct execution test for K-Means Clustering.**
Involved user story	US2
Method under test	performClustering()
Class under test	KmeansClusteringPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Clustering type has been selected as K-Means. • The number of clusters has been selected. • The categorical columns have been selected.
Input	The loaded data set, the number of clusters, a list with the categorical columns' names.
Output	A ClusteringProfile object with the calculated measurements.
Assert that	The generated ClusteringProfile object is equal with the expected ClusteringProfile object.

Figure 13. Description of the unit test regarding correct execution of Automated Clustering.

Test Case	**Correct execution test for Top-Down Clustering.**
Involved user story	US2
Method under test	performClustering()
Class under test	DivisiveClusteringPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Clustering type has been selected as Divisive. • The number of clusters has been selected. • The categorical columns have been selected.
Input	The loaded data set, the number of clusters, a list with the categorical columns' names.
Output	A ClusteringProfile object with the calculated measurements.
Assert that	The generated ClusteringProfile object is equal with the expected ClusteringProfile object.

Figure 14. Description of the unit test regarding correct execution of Top-Down Clustering.

Test Case	**Correct execution test for Graph Based Clustering.**
Involved user story	US2
Method under test	performClustering()
Class under test	GraphBasedClusteringPerformer
Prerequisite condition	• A data set successfully loaded into the system. • Clustering type has been selected as Graph Based. • The number of clusters has been selected. • The categorical columns have been selected.
Input	The loaded data set, the number of clusters, a list with the categorical columns' names.
Output	A ClusteringProfile object with the calculated measurements.
Assert that	The generated ClusteringProfile object is equal with the expected ClusteringProfile object.

Figure 15. Description of the unit test regarding correct execution of Graph Based Clustering.

Test Case	**Correct execution test for Normalized Outlier Detection.**
Involved user story	US3
Method under test	identifyOutliers()
Class under test	NormalizedScoreOutlierAlgo
Prerequisite condition	• A data set successfully loaded into the system. • Outlier detection type has been selected as Normalized. • A threshold has been selected.
Input	The loaded data set, the outlier threshold.
Output	A list of OutlierResult objects with the found outliers.
Assert that	The generated list with the OutlierResult objects has the same elements with the expected list of OutlierResult objects.

Figure 16. Description of the unit test regarding correct execution of Normalized Outlier Detection.

3.5 Installation & implementation details

This section offers insights into Pythia's specifications, detailing the development tools and technologies utilized, along with instructions for setting up the system as a developer. Below is a succinct overview of the various technologies employed in Pythia's development:

Git & GitHub: Pythia's version control system relies on Git and GitHub. The project is open source, with its source code accessible via the GitHub repository referenced in [DAIN23]. Further details on Git and GitHub can be found in section 2.2.9.

Java: Pythia is crafted using Java 8 as its primary programming language. Java's widespread usage and platform independence make it a versatile choice for building diverse systems and applications.

Eclipse: Development of Pythia was facilitated using Eclipse, an open-source integrated development environment (IDE) renowned for its robust features tailored for Java development. While Eclipse was the chosen IDE, Pythia is compatible with other Java-supporting IDEs like IntelliJ or Visual Studio Code.

Maven: Pythia employs Maven, an open-source build automation tool tailored for Java applications. Maven simplifies the build process and manages external dependencies using a declarative XML file known as Project Object Model (POM).

Apache Spark: Pythia harnesses the power of Apache Spark for efficient processing of large datasets. Apache Spark serves as a high-performance, unified engine optimized for executing data engineering and analytics tasks at scale. Further elaboration on Apache Spark can be found in section 2.2.8.

Below are the instructions for installing Pythia on a new machine for development purposes, as outlined by [Alex22]:

1. Begin by downloading and installing Java 8 on the new machine. After installation, ensure to update the path of the JAVA_HOME environment variable to point to the newly installed Java directory.
2. Next, download and install an Integrated Development Environment (IDE) suitable for Java applications, such as Eclipse or IntelliJ.
3. (Optional, but recommended) If not already installed, download, and install Git and create a GitHub account. Configure global git username and email, along with any other necessary first-time settings to enable cloning repositories from GitHub to the local machine.

4. Clone the Pythia source code from the repository referenced in [DAIN23] to the local machine. Alternatively, the source code can be downloaded directly as a compressed zip file without using Git.
5. The source code includes a Maven Wrapper, eliminating the need for manual Maven installation.
6. Download and install Apache Hadoop 3.2.2 from the official Hadoop website. Extract the downloaded tar.gz compressed file and update the path of the HADOOP_HOME environment variable to point to the extracted directory.
    a. For Windows operating systems, installation of WinUtils for Hadoop 3.2.2 is necessary. Download the binary files from the repository referenced in [WinU21] and place them within the bin directory of the newly installed Hadoop.
7. Once the above steps are complete, open the source code using the installed IDE to begin development.

Maven is designed to automatically fetch external dependencies and build the application, making it ready for execution within the Integrated Development Environment (IDE). Most IDEs offer features that simplify building and running applications, including tests. However, for comprehensive instructions, key terminal commands for building and running the Pythia system are provided below. It's important to execute these commands after navigating to the root directory of the system, where the mvnw.cmd and mvnw executable files for Windows and Unix-based operating systems are located.

- Building on Windows: > ./mvnw.cmd clean install
- Building on Unix systems: > ./mvnw clean install
- Running tests on Windows: > ./mvnw.cmd test
- Running tests on Unix systems: > ./mvnw test

The build command generates two Java archive (JAR) files named "Pythia-x.y.z-all-deps.jar" and "Pythia-x.y.z.jar". Either JAR can be imported into other Java applications as an external library. The first JAR (all-deps) includes all external dependencies precompiled and integrated with Pythia's executable source code. This facilitates easy integration into other applications regardless of their dependencies. In contrast, the second JAR contains only Pythia's executable source code without external dependencies. Importing this JAR into other Java applications requires importing Pythia's dependencies separately. The second JAR is recommended for applications with existing Pythia dependencies to conserve disk space.

3.6 Software scalability

The software architecture of Pythia is crafted with methodologies aimed at ensuring effortless maintenance and scalability of the system.

As previously mentioned, each feature of Pythia is meticulously developed and organized into distinct packages. Most of these packages employ the parameterized Factory software design pattern, complemented by an interface implemented by all instantiated classes within the factory class. This approach allows for consistent interaction among created objects, regardless of their internal logic. Consequently, the system can be seamlessly extended with additional logic by simply introducing a new class that implements the interface, along with the necessary parameter distinguishing it from other implementations. This streamlined process eliminates the need for further refactoring.

The clustering and regression features introduced in this thesis are built upon the foundation of the parameterized Factory software design pattern, ensuring scalability and ease of maintenance within Pythia's architecture. By adhering to this pattern, the clustering and regression functionalities are seamlessly integrated into the system's framework, allowing for consistent interaction with other components. This approach fosters scalability, as additional clustering and regression algorithms can be effortlessly incorporated by creating new classes that implement the established interface. Moreover, the parameterized nature of the Factory pattern facilitates the customization of these features based on specific requirements or preferences, without necessitating extensive modifications to existing code. Overall, leveraging this pattern not only ensures the extensibility of Pythia but also enhances its ability to adapt to evolving analytical needs with minimal overhead.

Lastly, an extra feature can also be easily introduced to the system following the same steps that were presented for the implementation of regression and clustering features. Since the execution flow of Pythia is a standard scalable framework then it’s very easy for someone to adapt and interact with the system’s architecture.

Chapter 4. Experimental Evaluation

This chapter offers a comprehensive exploration of the experimental assessment conducted on the recently introduced features of Pythia. The initial section delves into the methodologies employed in the experiments, outlining the chosen input dataset and the environment where the experiments took place. Following this, the subsequent section meticulously presents the findings of the experimental evaluation, offering an extensive breakdown of results through the utilization of tables and bar charts.

4.1 Experimental Methodology

The experiments conducted encompassed the measurement of execution time across all stages of the data analysis process associated with regression and clustering algorithms. This includes the registration of datasets, the data profile computation, and the generation of reports. There are separate experimental methodologies for regression and clustering analysis. In the case of clustering analysis, the quality of the clustering is also tested with the very well-known metric: Silhouette Score.

The measured execution time can be categorized into two levels of abstraction, corresponding to the interfaces that implement the different operations. This categorization aims to offer a comprehensive understanding of the time allocation across various components of the data analysis process.

IDatasetProfiler: 'IDatasetProfiler' serves as the highest level of abstraction within the system. As elaborated in the preceding chapter, 'IDatasetProfiler' functions as the central control unit in Pythia. At this level, the execution time is gauged for fundamental functionalities of the system, encompassing tasks like computing all pairs correlations and detecting highlights. Additionally, to provide a comprehensive overview, the total execution time is also incorporated at this level. Specifically, the execution time was monitored for the following operations performed by 'IDatasetProfiler':

1. Register data.
2. Declare necessary parameters.
3. Compute all pairs correlations (only in case of regression analysis)

4. Identify highlights of the data profile.
5. Generate report in txt format.
6. Generate report in markdown format.

IRegressionPerformer: Only in case of regression analysis, 'IRegressionPerformer' is responsible to fulfil the entire user's regression request. The overall time IRegressionPerformer needs is the sum of every single regression performance time that occurs from the request.

IClusteringPerformer: Only in case of clustering analysis, 'IClusteringPerformer' performs the clustering analysis on the dataset and causes the time cost for extracting the object with the information Pythia needs to generate the reports.

For regression experiments all four regressions variants Pythia currently supports are tested. For each one of them the time is measured while changing a single parameter.

- **Linear Regression:** For Linear Regression there is one experiment.
    - *Parameter under test:* Number of Rows.
- **Multiple Linear Regression:** For Multiple Linear Regression there are two experiments.
    - *First parameter under test:* Number of Rows.
    - *Second parameter under test:* Number of Columns (Independent Variable).
- **Polynomial Regression:** For Polynomial Regression there are two experiments.
    - *First parameter under test:* Number of Rows.
    - *Second parameter under test:* The Degree of the Polynomial.
- **Automated Regression:** For Automated Regression there is one experiment.
    - *Parameter under test:* Number of Rows.

The dataset that is used originally had one million rows and ten columns. All columns contain float numbers, so in Java they are considered as doubles [FLePla21]. For the sake of the experiments that test the effect of the number of rows on the execution time, the first *n* rows were cut for each test point and the time measurement was performed on them. Specifically, we measured time by executing regression five times on the first *n* rows of the dataset and capturing the mean of the observed times. The selected number of rows selected to be under test where *n* = {100, 1K, 10K, 50K, 100K, 500K, 1M}. In the case of Multiple Linear Regression where the effect of the number of columns on time is also tested there is no need to do any modification on the dataset since the selection of

independent variables happens programmatically in the client classes. The selected number of columns selected to be under test were {2, 3, 4, 5, 6, 7, 8}. Lastly, when experimenting with Polynomial Regression, the degree of the polynomial is another parameter that is under test for its effect on the execution time. The selected degrees for this experiment are {2, 3, 4, 5, 6}. There is no need to go further than 6 since in real life problems even the 6th degree is rare to express a phenomenon and five possible degrees are more than enough to test if they have any effect on execution time.

For the experiments on the clustering analysis, the three clustering techniques Pythia has been enhanced with during this thesis are tested. For each one of them execution time and Silhouette Score are measured while changing a single parameter.

- **K-Means:** For K-Means there are three experiments.
    - *First parameter under test:* Number of Rows.
    - *Second parameter under test:* Number of Columns (features).
    - *Third parameter under test:* Number of Clusters (K).
- **Divisive:** For Divisive clustering there are three experiments.
    - *First parameter under test:* Number of Rows.
    - *Second parameter under test:* Number of Columns (features).
    - *Third parameter under test:* Number of Clusters (K).
- **Power Iteration Clustering (PIC):** For PIC there are three experiments.
    - *First parameter under test:* Number of Rows.
    - *Second parameter under test:* Number of Columns (features).
    - *Third parameter under test:* Number of Clusters (K).

The original dataset for those experiments is the same as the one used at regression. Apparently, there are three parameters that influence time and quality that need to be tested and this leads to three concrete experimental groups about clustering. In each one of those experimental groups the three algorithms will be compared with each other, since in contrast with regression, all of them try to solve the same problem: to cluster the dataset.

Experimental Group 1: In those experiments time and quality are measured while changing the number of rows of the dataset. The dataset has 7 columns. The six of them have float values and one of them has Boolean value ('TRUE' or 'FALSE'). In each experiment in this group the number of clusters in set to K = 4. The number of rows to be tested are *n* = {100, 1K, 10K, 50K, 100K, 500K, 1M}.

Experimental Group 2: In those experiments time and quality are measured while changing the number of columns of the dataset. The dataset has 1 thousand rows. Once again, in each experiment in this group the number of clusters in set to K = 4. The number of columns to be tested are m = {2, 3, 5, 7, 10, 12}. The categorical column is always considered in the clustering and it's always one, so, each experiment differs on the number of numerical columns that it has. For instance, when m=2, the dataset has 1 numerical column and 1 categorical but when m=3, the dataset has 2 numerical columns and 1 categorical etc.

Experimental Group 3: In those experiments time and quality are measured while changing the number of clusters. The dataset has 1 thousand rows and 7 columns (again one categorical-Boolean and 6 numerical). The number of clusters to be tested are K = {4, 10, 50, 100, 200}.

4.2 Detailed Results Presentation

Since all the experimental methodologies have been completely demonstrated, this section provides a detailed presentation of the measurement results in chart bar form. Regression and clustering experimental analysis are presented separately.

4.2.1 Regression Experiments Presentation

As mentioned, there are experiment results for all four supported variants of Regression.

For **Linear Regression** there is only one parameter under test, and it's the number of rows of the dataset. Below there is a bar chart showing the results of this experiment:

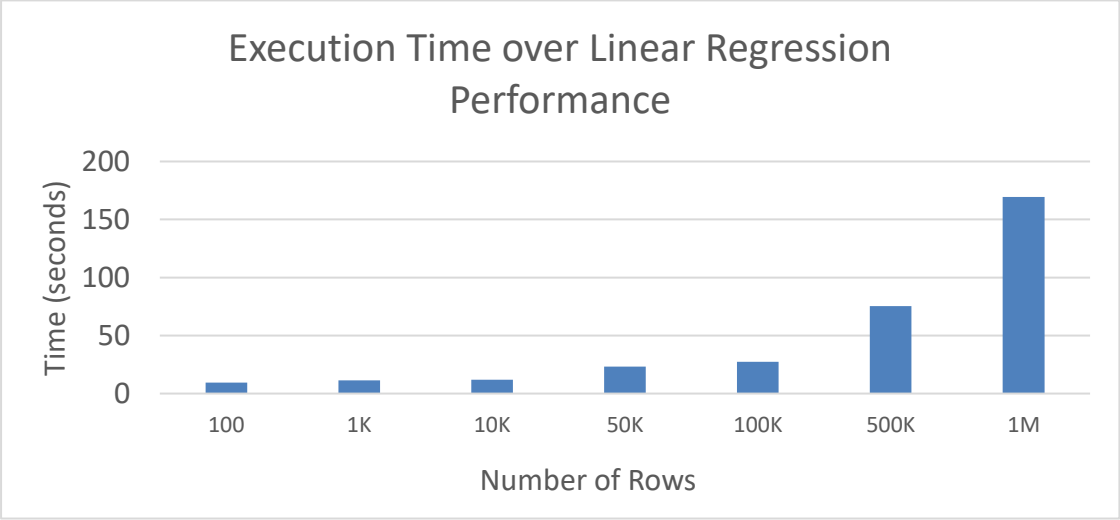


Figure 17. Bar chart showing how the number of rows affects the execution time of Linear Regression.

From the results of this experiment, it can be concluded that the number of rows has an important effect on the execution time. This makes total sense since the computational complexity of Linear Regression is sensitive to the size of the dataset. Even the error that Linear Regression tries to minimize is calculated slower as the number of rows increases.

Next, **Multiple Linear Regression** has 2 parameters to be tested. Like in Linear Regression the number of rows is also tested. Below there is a bar chart showing the results of the experiment testing the effect of the number of rows on the execution time:

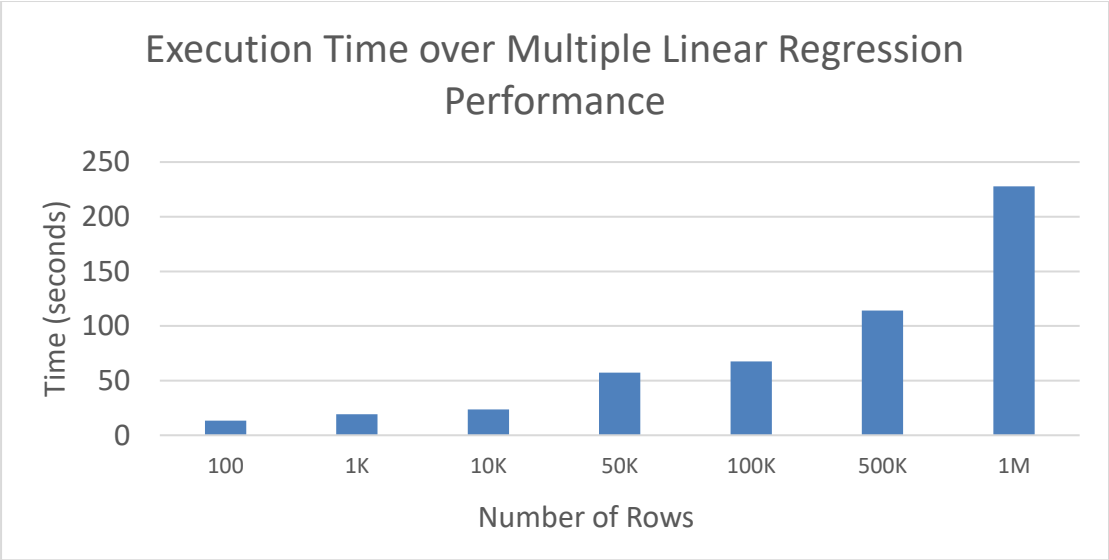


Figure 18. Bar chart showing how the number of rows affects the execution time of Multiple Linear Regression.

With more rows, the algorithm must process a larger amount of data during training. For each iteration of the training process, the algorithm computes predictions, calculates errors, and updates coefficients based on the entire dataset. As the number of rows increases, the algorithm needs to perform these calculations more times, leading to longer training times. Overall, the number of rows in the dataset affects execution time in Multiple Linear Regression because it influences the amount of data processed, memory usage, computational complexity of matrix operations, and convergence behavior of optimization algorithms. These theories are verified by the above experiment as it's clearly observed that as the number of rows increases so does the execution time.

The second parameter under test for Multiple Linear Regression is the number of columns considered as independent variables. In the context of Multiple Linear Regression, the computational complexity is primarily determined by the number of rows rather than the

number of columns. Both training and prediction involve matrix operations, where the size of the dataset (number of rows) typically has a greater impact on the computational workload than the number of features (columns). The algorithm is tested for a small range of independent variables that represents its general behavior. Below there is a bar chart which corroborates this theory:

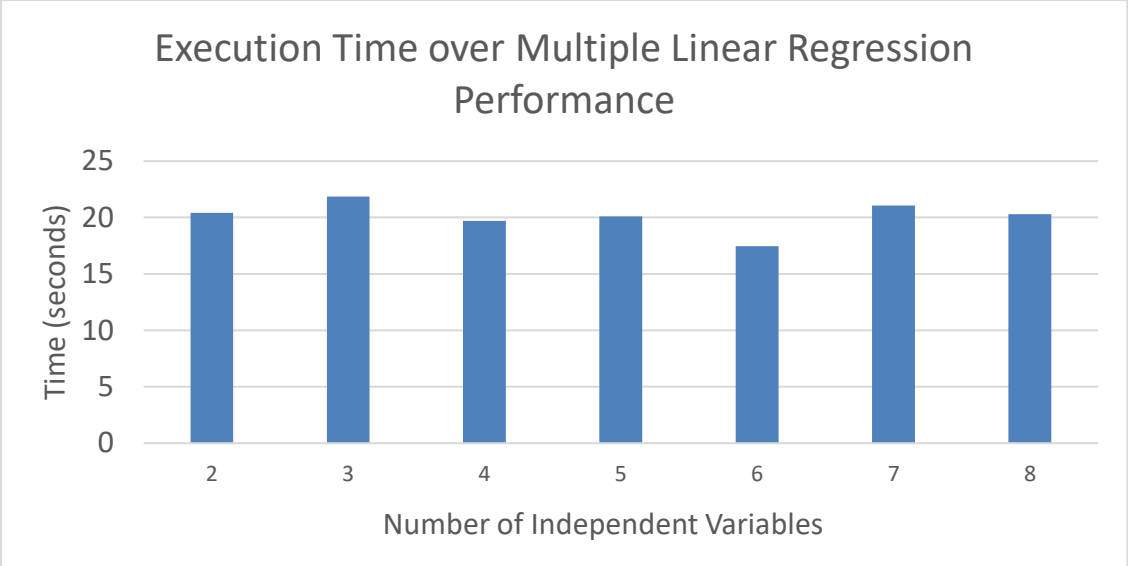


Figure 19. Bar chart showing how the number of Independent Variables affects the execution time of Multiple Linear Regression.

While it may seem that increasing the number of columns in linear regression does not significantly affect execution time, it's essential to recognize that additional columns do introduce a small increase in algorithmic complexity. Although this impact is typically overshadowed by the larger influence of increasing the number of rows, the additional features contribute to the computational workload, albeit to a lesser extent. Therefore, while the primary driver of execution time remains the number of rows, the number of columns still plays a role in determining the overall complexity of the algorithm, something that cannot be observed by the above chart bar.

Polynomial Regression has 2 parameters to be tested. Once again, the first one is the number of rows. Below there is a bar chart showing the results of the experiment testing the effect of the number of rows on the execution time:

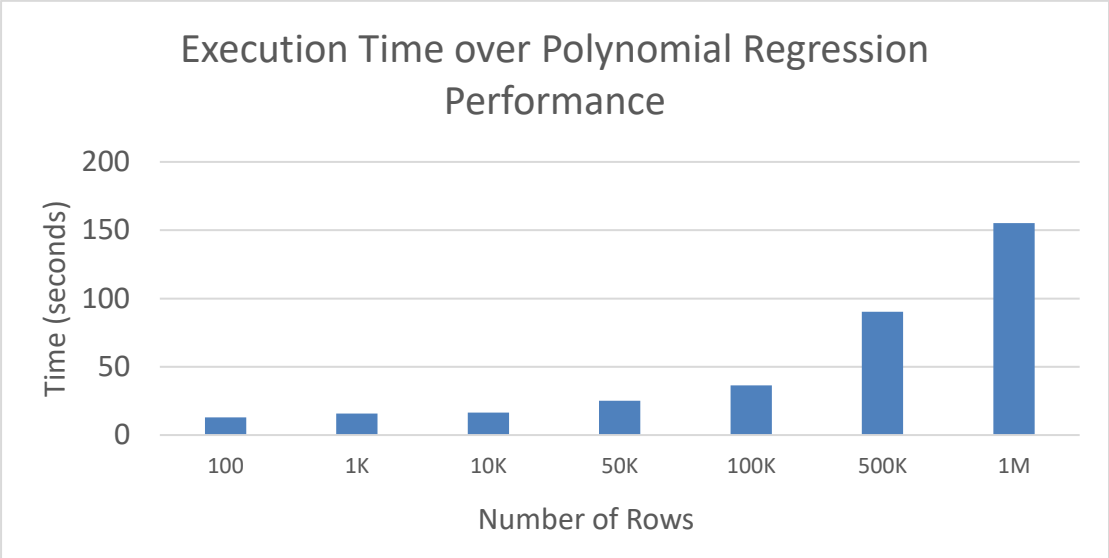


Figure 20. Bar chart showing how the number of rows affects the execution time of Polynomial Regression.

Polynomial regression algorithms need to process each data point during training, which involves computing predictions, errors, and model parameter updates. With more rows, the algorithm requires more calculations, leading to longer training times. This is also observed by the above experiment.

The second parameter under test for Polynomial Regression is the degree of the polynomial. Below there is a bar chart showing the results of the experiment testing the effect of the polynomial degree on the execution time:

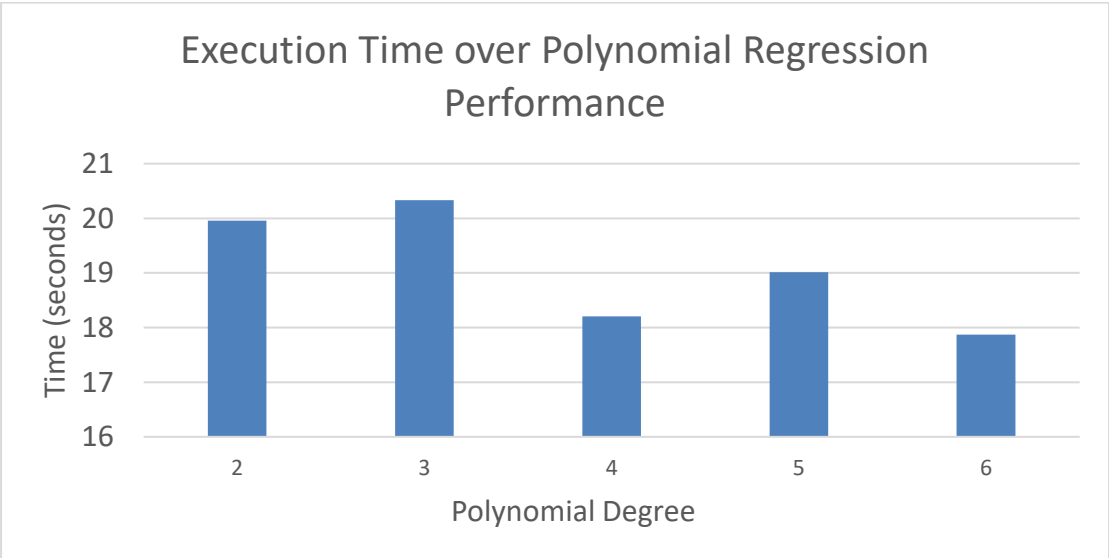


Figure 21. Bar chart showing how the degree affects the execution time of Polynomial Regression.

While the degree of the polynomial does affect the complexity of the model and the number of features (columns) in the dataset, the impact on execution time may not be as pronounced compared to the number of rows (samples). The primary determinant of execution time in polynomial regression is often the size of the dataset rather than the degree of the polynomial. That's why in each degree its is observed approximately the same time. However, it would be more realistic to observe a small increase on the execution time.

Lastly, in the case of **Automated Regression**, the only parameter under test is the number of rows. Below there is a bar chart showing the results of the experiment testing the effect of the number of rows on the execution time:

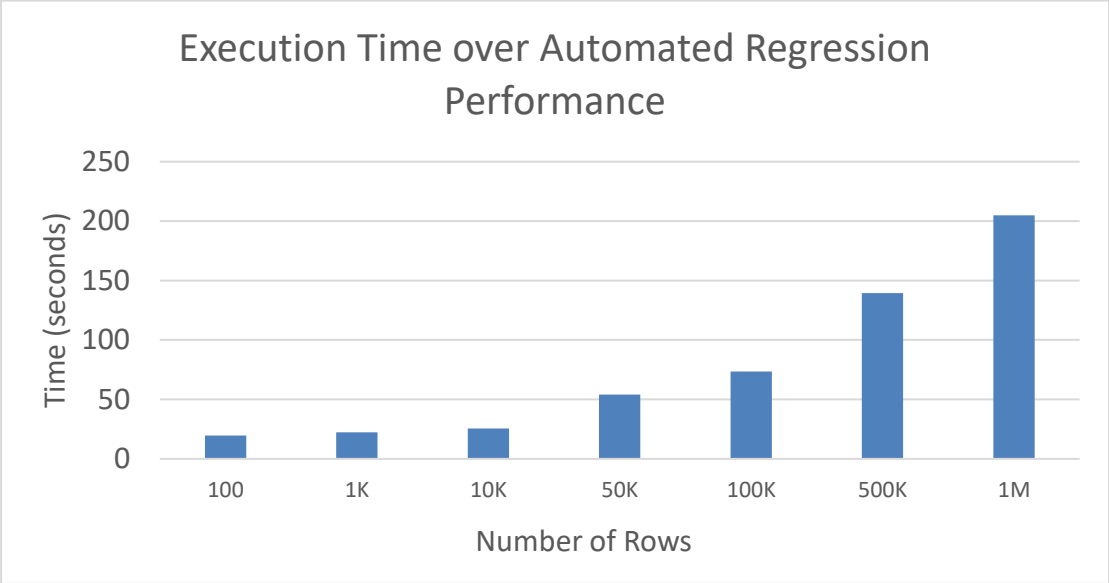


Figure 22. Bar chart showing how the number of rows affects the execution time of Automated Regression.

Automated Regression is practically Multiple Linear Regression with the extra overhead of computing the independent variables. Considering that, the number of rows affects execution time of Automated Regression for the same reasons as Multiple Linear Regression. Therefore, the observed result is justified by the Multiple Linear Regression complexity dependence from the number of rows.

4.2.2 Clustering Experiments Presentation

Clustering analysis consists of the three experimental groups mentioned in the previous section. It is reminded that the three algorithms under test are: K-Means, Divisive, PIC.

Experimental Group 1: In this experimental group the parameter under test is the number of rows of the dataset.

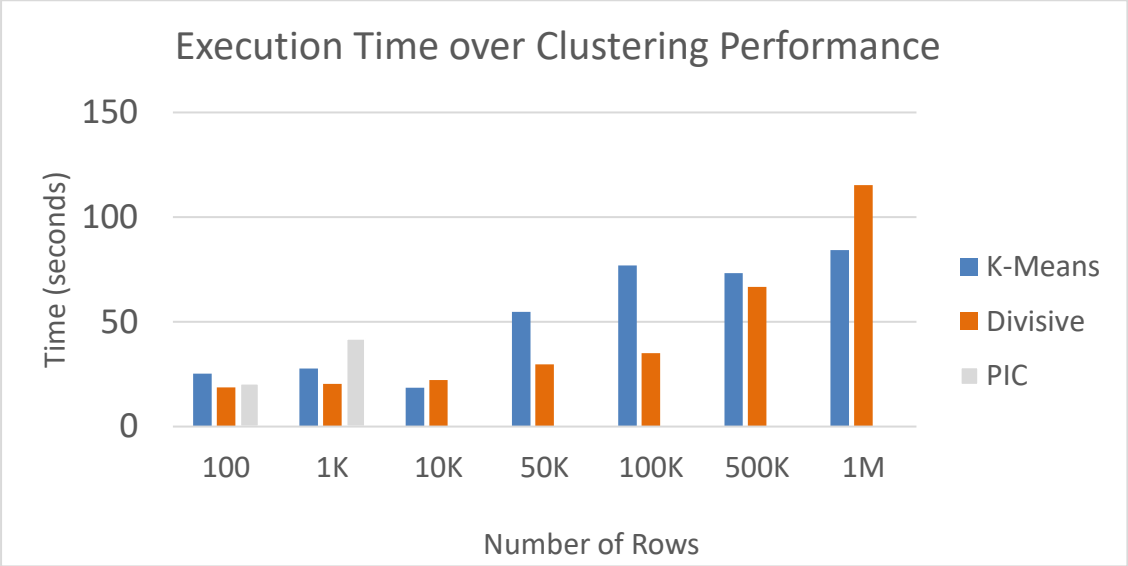


Figure 23. Bar chart comparing the effect of number of rows on the clustering algorithms' time.

This observation underscores the computational complexity associated with processing larger datasets in clustering tasks. With more data points to analyze, the algorithms require additional computational resources and processing time to iteratively partition the data and update cluster centroids. The interesting think to emphasize, is the spatial complexity of Power Iteration Clustering. Due to the large arrays that Power Iteration Clustering creates during its steps, it is impossible for the machine to store all of them in the memory, so the program exits with spatial issue warnings. Just for the array that holds the cosine similarities for each pair of rows it is required $O(N^2)$ space. And that is just one of the arrays this algorithm needs to perform clustering. On the other side, it is observed that Divisive's execution time increases faster than K-Means'. This happens because Bisecting K-Means executes K-Means internally, so its execution time increases multiplicatively with K-Means time based on the number of clusters (K) which is fixed in this experiment.

Continuing with the effect of the number of rows on the Average Silhouette Score the results that occur are presented below:

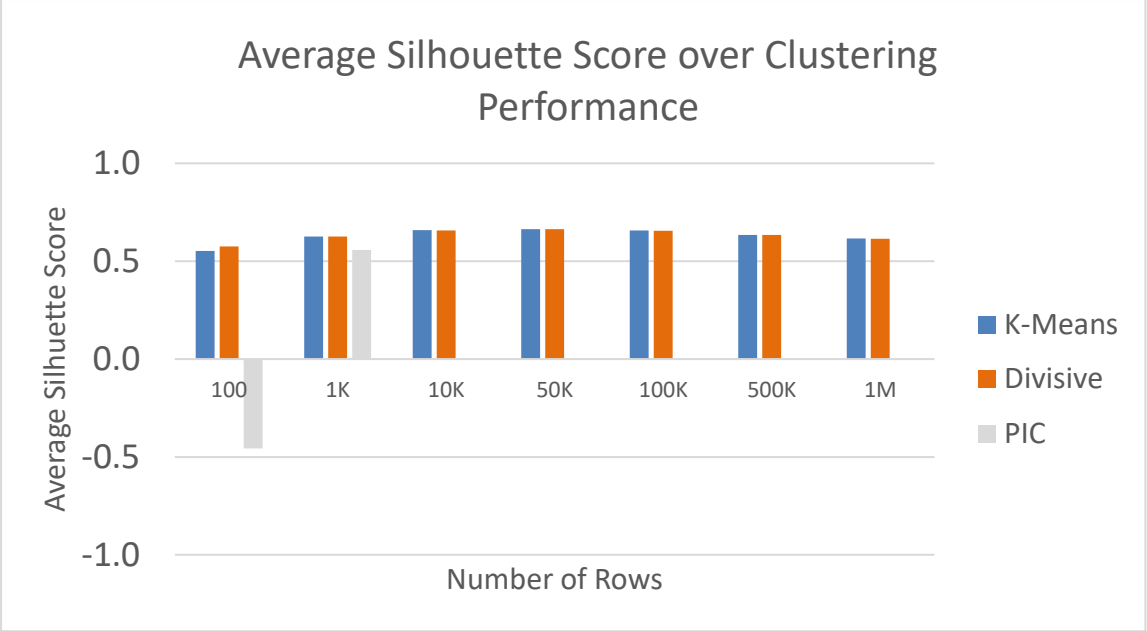


Figure 24. Bar chart comparing the effect of number of rows on the Average Silhouette Score.

Number of Rows should not have an affect on the Average Silhouette Score, since the data inside a dataset of 500 thousand rows are unknown. There is no way to know a priori for random datasets if they can be clustered good a priori. Once again, PIC was measured for a max of 1 thousand rows because of the memory issue. It is observed that for 100 rows PIC gave negative Average Silhouette Score, meaning that the clustering was not good. This indicates that PIC is less suitable for this dataset compared to the other algorithms.

Experimental Group 2: In this experimental group the parameter under test is the number of columns of the dataset.

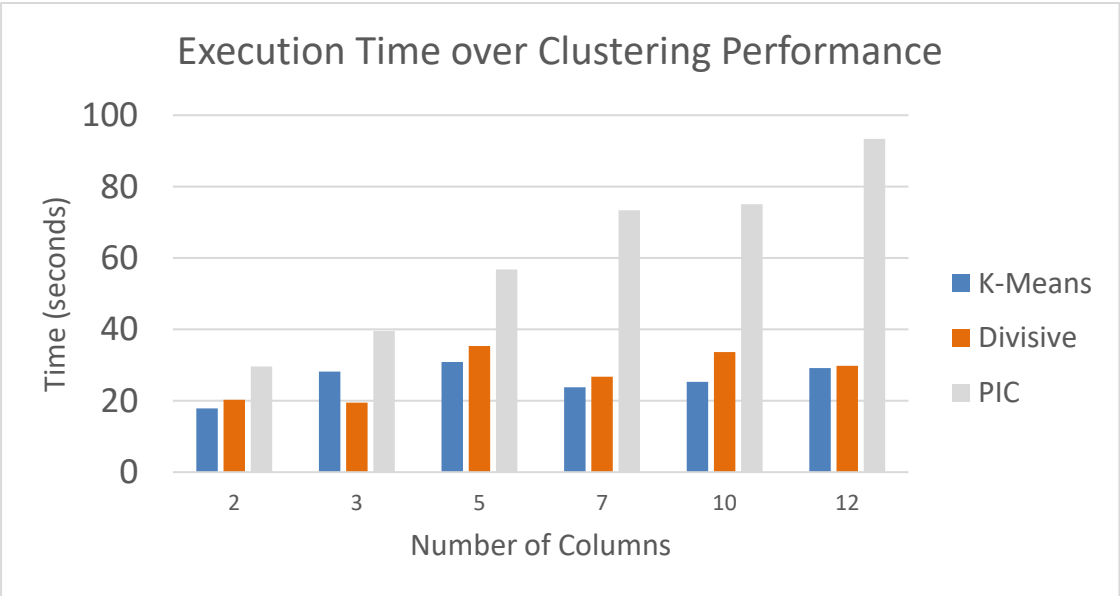


Figure 25. Bar chart comparing the effect of number of columns on the clustering algorithms' time.

The experiments conducted on clustering algorithms, including K-Means, Bisecting K-Means, yielded a notable finding: despite increasing the number of columns in the dataset, the execution time of the algorithms remained relatively stable. This observation suggests that the computational complexity of these clustering algorithms is primarily influenced by the number of rows rather than the number of columns. On the other hand, observe that PIC's execution time is sensitive to the number of columns compared to the other algorithms. One reason leading to this result is the computation of cosine similarity for each row-pair. Increasing the number of columns means an increase to the dimensionality of the data, leading to a cost on the computation of cosine similarity.

Next, the effect of the number of columns on the Average Silhouette Score is presented on the chart bar below:

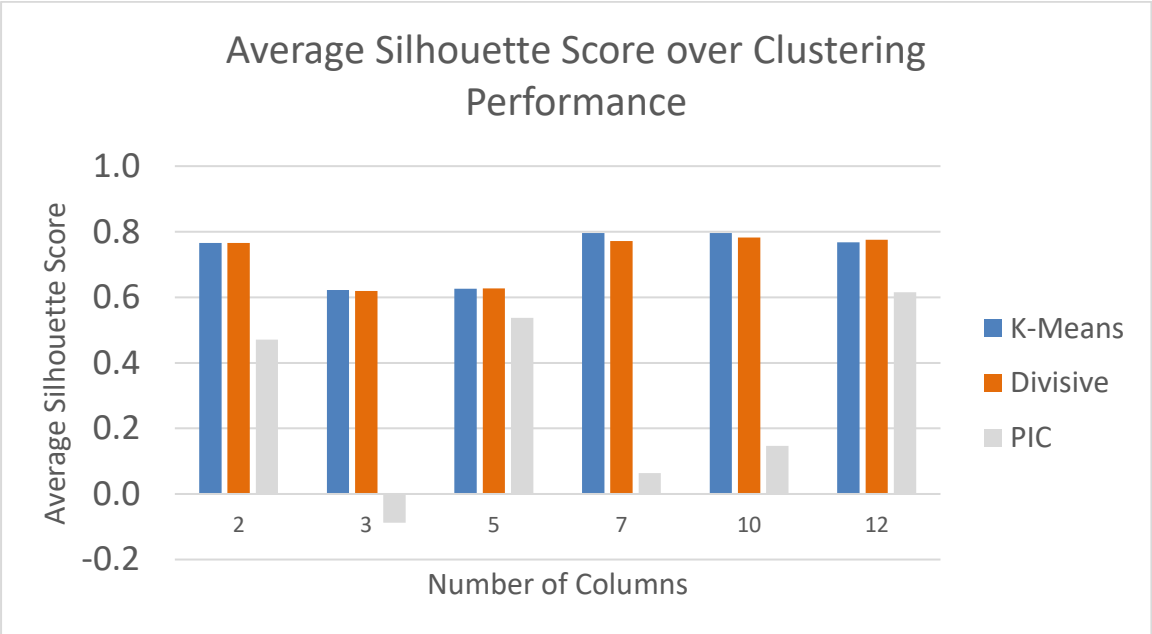


Figure 26. Bar chart comparing the effect of columns on the Average Silhouette Score.

As Average Silhouette Score is influenced by the geometrical distribution of the clusters, increasing their dimensionality (by increasing the number of columns with unknown data) cannot create a general pattern. Each dataset must be tested with a vary of dimensions and the analyst should keep the best based on a metric like Silhouette Score. The only thing that can be observed by this experiment is that PIC gives the lowest Silhouette Score on average, so it is not a good choice for this dataset.

Experimental Group 3: In this experimental group the parameter under test is the number of clusters.

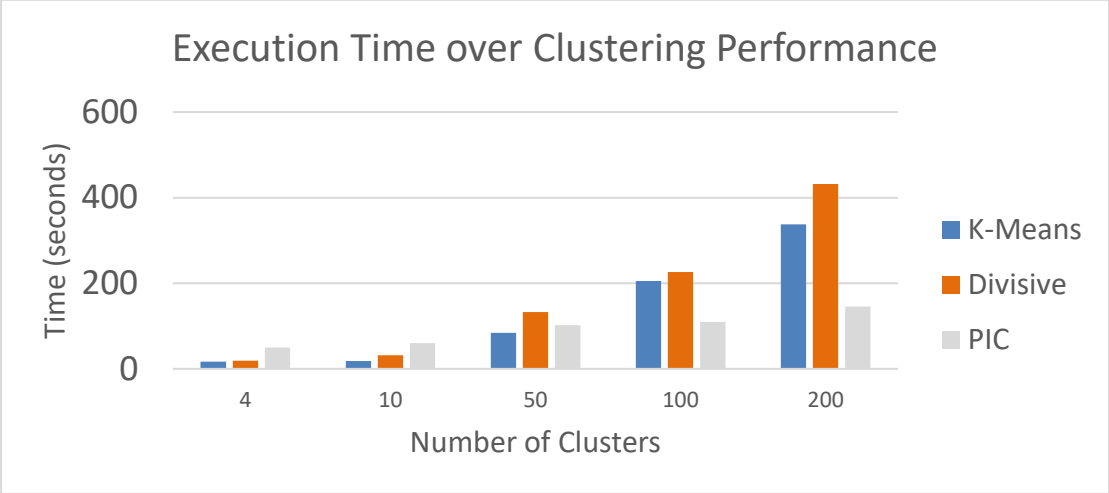


Figure 27. Bar chart comparing the effect of number of clusters on the clustering algorithms' time.

This experiment revealed a consistent trend: as the number of clusters in the dataset increased, the execution time of the algorithms also increased. This observation highlights the computational complexity inherent in clustering tasks, particularly when attempting to partition data into a larger number of clusters. With more clusters to compute and update, the algorithms require additional computational resources and processing time, resulting in longer execution times. It should be noted that the execution time says nothing about the ideal number of clusters which is unknown a priori. The only important observation of this experiment is the comparison between the three algorithms. Even though all algorithms are sensitive to the number of clusters, it is clear that PIC is more efficient than the others. This happens because the number of clusters does not affect the expensive step (in terms of time) of similarities array. Therefore, PIC is not as sensitive to the number of clusters as the other two algorithms when it comes to the execution time.

Lastly, the effect of the number of columns on the Average Silhouette Score is presented on the chart bar below:

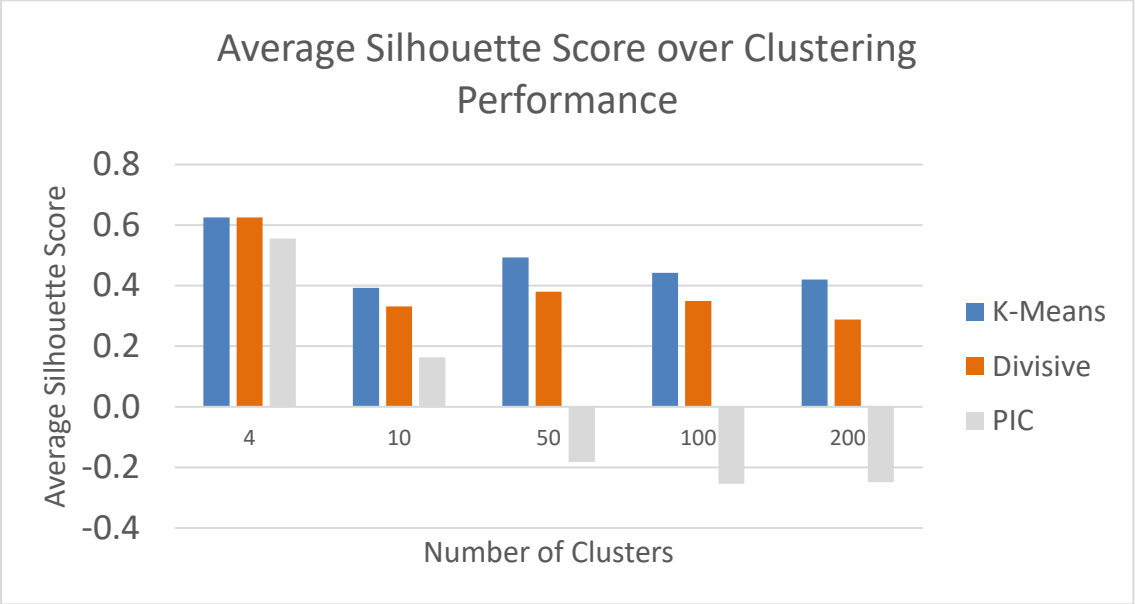


Figure 28. Bar chart comparing the effect of number of clusters on the Average Silhouette Score.

The average silhouette score of clustering algorithms remains unaffected by changes in the number of clusters due to the intrinsic nature of the silhouette score metric. The silhouette score evaluates the compactness and separation of clusters based on distances between data points within and across clusters. Regardless of the number of clusters specified, the silhouette score assesses the quality of clustering by considering the distances between data points and their respective cluster centroids. Therefore, as the number of clusters varies, the silhouette score adjusts accordingly to reflect the relative cohesion and separation of clusters, ensuring a consistent evaluation of clustering quality irrespective of cluster count. In this experiment it is observed that PIC has the lowest clustering quality based on Average Silhouette Score, with most of the times being negative. In conclusion, none of the parameters under test influenced the silhouette score.

Chapter 5. Epilogue

5.1 Synopsis and conclusions

Exploratory Data Analysis (EDA) serves as a crucial methodology for data scientists, allowing them to delve into intricate datasets to uncover patterns and anomalies. However, the absence of automation within EDA poses a hurdle in swiftly extracting such insights, as analysts are required to manually execute analysis methods and handle any necessary data preprocessing tasks. To overcome this challenge, an innovative solution called Pythia [Alex22] was devised with the goal of streamlining automated EDA processes. Pythia operates as a Java application harnessing the power of the Apache Spark engine to efficiently process vast datasets, providing users with a comprehensive overview of valuable insights derived from the data.

Prior to this thesis, Pythia had less functionalities. Pythia showcased its prowess by ingesting datasets and crafting comprehensive statistical profiles, complete with automated correlation calculations. Furthermore, the system demonstrated its versatility by effortlessly constructing decision trees for labeled fields within the dataset, simplifying the process considerably. Beyond these functionalities, Pythia notably excelled in identifying dominant patterns, thus prioritizing the evaluation of input dataset quality. Outlier detection was enhanced with another implementation giving the analysts the choice to process data with a normalized version.

The principal contribution of this thesis to Pythia lies in the integration of advanced machine learning techniques, significantly enhancing its analytical capabilities, and introducing applications of the contemporary field of Artificial Intelligence (AI). Specifically, the addition of regression and clustering algorithms has empowered Pythia to offer users a more comprehensive toolkit for data analysis. With regards to regression, the thesis introduced several types of regression models, including Linear Regression, Multiple Linear Regression, Automated Regression, and Polynomial Regression. These diverse regression techniques afford users the flexibility to choose the most suitable model for their data, enabling precise predictions and insights extraction. Additionally, the incorporation of various clustering algorithms, such as K-Means, Divisive Clustering, and Power Iteration Clustering, extends Pythia's functionality in uncovering meaningful

patterns and structures within datasets. By leveraging these machine learning techniques, Pythia equips analysts with powerful tools for exploring and understanding complex datasets, thereby advancing its utility and effectiveness in exploratory data analysis using modern solutions.

Each new feature introduced in Pythia adheres to the established execution flow, following a consistent framework and pattern methodology. This meticulous approach ensures that Pythia remains a scalable software solution, capable of accommodating additional functionalities seamlessly. Moreover, it enhances the software's overall comprehensibility, facilitating ease of understanding for users and developers alike.

5.2 Future extensions

In this concluding section dedicated to Pythia's development, suggestions for future extensions that can amplify its capabilities and utility are explored.

- **Data Preprocessing/Preparation:** These tools are essential for ensuring that algorithms like regression can be executed effectively without encountering issues stemming from incomplete or erroneous data. Pythia could benefit from incorporating robust data preprocessing capabilities to facilitate tasks such as missing value imputation, outlier detection, and feature scaling. By providing users with tools to clean and prepare their data before running regression algorithms, Pythia can improve the accuracy and reliability of regression model outputs. Additionally, efficient data preprocessing workflows can streamline the overall analysis process, allowing users to focus more on deriving insights from their data rather than spending time addressing data quality issues. Therefore, integrating data preprocessing functionalities into Pythia would enhance its usability and enable users to perform more reliable and insightful data analyses.

- **Graphic User Interface (GUI):** The implementation of GUI for Pythia would represent a significant advancement in the project's usability and accessibility. Currently, analysts can only interact with Pythia through programming the client class, limiting its accessibility primarily to Java-savvy users. Introducing a GUI would democratize access to Pythia, making it more user-friendly, particularly for analysts who may not be proficient in Java or programming in general. By providing a visual interface, analysts can interact with Pythia more intuitively, reducing the learning curve and enabling a broader range of users to leverage its

powerful data profiling capabilities. Moreover, a GUI could extend Pythia's functionality by incorporating data visualization tools, allowing analysts to visualize insights and patterns directly within the application. Whether as a terminal-only application, a desktop application, or a web-based platform, a front-end interface has the potential to enhance Pythia's usability and broaden its appeal to analysts across diverse skill levels and backgrounds.

- **Integration with External Data Sources:** Enabling Pythia to seamlessly integrate with external data sources, including databases, APIs, and streaming data sources, would significantly enhance its capabilities and utility. By facilitating direct access to diverse data sources, users can effortlessly import and analyze data from multiple sources without the need for manual data preprocessing or transformation steps. This seamless integration streamlines the data analysis process, saving time and effort while ensuring that users have access to the most up-to-date and relevant data for their analyses. Additionally, integration with external data sources expands Pythia's versatility, allowing it to accommodate a wide range of data types and formats, from structured databases to unstructured streaming data. Overall, this feature empowers users to conduct more comprehensive and insightful analyses by leveraging a broader array of data sources within Pythia's familiar and user-friendly environment.

- **Distributed Computing:** Optimizing Pythia's algorithms and processing pipeline for distributed computing environments, such as Apache Spark clusters, presents a significant opportunity to enhance its scalability and performance for big data analytics. Leveraging the existing foundation on Apache Spark provides a solid framework for parallel and distributed computing, making it well-suited for handling large-scale datasets efficiently. By optimizing Pythia's algorithms and workflows for distributed computing, it can effectively harness the computational power of Spark clusters, enabling it to process and analyze even larger datasets with ease. This optimization not only improves Pythia's ability to handle big data analytics tasks but also enhances its scalability, allowing it to scale seamlessly as data volumes grow. Overall, optimization for distributed computing environments strengthens Pythia's position as a powerful tool for data analysis and enables it to meet the evolving needs of data-intensive applications.

- **Optimization and Evaluation of the clustering analysis:** The optimization and evaluation of clustering analysis in Pythia present opportunities to enhance its functionality and provide users with a broader range of options for data exploration. One avenue for improvement is the addition of new algorithmic families for clustering, such as grid-based algorithms like DBSCAN or neural network-based approaches. By incorporating these alternative clustering algorithms, users gain access to a diverse set of techniques for partitioning data into meaningful clusters, allowing for more nuanced analysis and interpretation of complex datasets. Additionally, Pythia's inclusion of Power Iteration Clustering (PIC), a graph-based algorithm, opens the door to further exploration and optimization using graph theory principles. PIC's inherent graph structure facilitates community detection within datasets, offering valuable insights into the underlying relationships and structures present in the data. By leveraging graph theory techniques to analyze and optimize PIC, Pythia can enhance its clustering capabilities, providing users with more accurate and efficient clustering results. Overall, the optimization and expansion of clustering analysis in Pythia empower users to explore data from different perspectives and extract deeper insights from their datasets.

References

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

[Agra23] Raghav Agrawal. *Master Polynomial Regression*. Available online at https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression. July 2023.

[AkSa18] John Akred and Anjali Samani. *Your Data Is Worth More Than You Think*. Available online at https://sloanreview.mit.edu/article/your-data-is-worth-more-than-you-think, January 2018.

[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. *What is Apache Spark?* Available online at https://aws.amazon.com/what-is/apache-spark, January 2023.

[Anton23] Christodoulos Antoniou. *Automated Highlight Identification in a Data Profiling System (Diploma Thesis).* Department of Computer Science & Engineering, University of Ioannina, Greece, July 2023.

[ApSpa23] Website of Apache Spark™. Available online at https://spark.apache.org.

[BeePot23] Brian Beers, reviewed by Charles Potters. *What is Regression? Definition, Calculation and Example.* Available online at https://www.investopedia.com/terms/r/regression.asp. March 2023.

[BoWa08] Sarah Boslaugh and Paul Andrew Watters. *Statistics in a nutshell (1st editiot)*. July 2008.

[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.

[databr23] Official website of databricks. Available online at https://www.databricks.com/glossary/what-is-rdd.

[Dizi10] Peter Dizikes. *Explained: Regression analysis*. MIT News, March 2010.

[Educ24] Educative Website. *What is Silhouette Score?* Available online at https://www.educative.io/answers/what-is-silhouette-score.

[FLePla21] Free Learning Platform For Better Future. *Java double keyword.* Available online at https://www.javatpoint.com/double-keyword-in-java. Nodia, UP, India.

[LinCo10] Frank Lin, William W. Cohen. *Power Iteration Clustering*. Carnegie Mellon University. Proceedings of the 27 th International Conference on Machine Learning, Haifa, Israel, 2010.

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

[Regu23] Refactoring Guru. Strategy. Available online at https://refactoring.guru/design-patterns/strategy.

[Santia23] Lucas Santiago Cardoso. *What Is an Apache Spark RDD?* Available online at https://www.baeldung.com/scala/apache-spark-rdd, Baeldung, September 2023.

[Simpli23] Simplilearn. *What is Spark GraphX? Everything You Need to Know*. Available online at https://www.simplilearn.com/spark-graphx-article#graph_algorithms, February 2023.

[SpaExa23] Naveen (NNK). *Spark Schema – Explained with Examples.* Available online at https://sparkbyexamples.com/spark/spark-schema-explained-with-examples, January 2023.

[StanUni24] Stanford University Website. *Error Sum of Squares (SSE).* Available online at https://hlab.stanford.edu/brian/error_sum_of_squares.html.

[WaLMo21] Wayne W. LaMorte. *Z Scores and the Standard Normal Dstribution.* Boston University School of Public Health. Available online at https://sphweb.bumc.bu.edu/otlt/mph-modules/ph717-quantcore/r-for-ph717/R-for-PH7179.html, December 2021.

[ZaXin16] Matei Zaharia, Reynold X. Sin. *Apache Spark: A Unified Engine for Big Data Processing.* Available online at https://people.eecs.berkeley.edu/~matei/papers/2016/cacm_apache_spark.pdf. November 2016.