Deciphering Patterns in High-Profile GitHub Open-Source Projects

Authors: Inna Krmoyan and Gor Mkrtchyan

Major: BS in Data Science

Supervisor: Arman Asryan

Date: 09.05.24

Abstract

This capstone project, "Deciphering Patterns in High-Profile GitHub Open Source Projects", aims to discover hidden underlying patterns among popular repositories that contribute to their success. Those repositories that are included in our study are open-source projects taken from GitHub. GitHub is a very popular platform for version control and collaborative development of projects, that provides a diverse set of repositories. As it is a very well-known and popular platform for software developers, it is crucial to know what features contribute to the popularity of a repository. This study attempts to discover and explore patterns within the repositories by conducting a thorough data analysis followed by machine learning techniques in order to provide meaningful insights into the features that influence the popularity of the projects, including various variables such as the programming language, community engagement, etc.

Introduction

This project is important to the software development community as the core idea lies in the potential to discover ways to improve repository development in GitHub and help guide the developers who want to have a repository that becomes successful and useful to many. It aims to identify possible connections and correlations between the attributes of the repositories that are available on GitHub. By using our domain knowledge one can assume that the main components of a repository that makes it spark are Stargazers, Forks, Downloads, and so on. We aim to test our intuition and see whether the main variables that contribute to the success are the ones that immediately come to mind.

Ultimately, this capstone project aspires to contribute to the broader understanding of successful software development on GitHub by providing developers with useful insights that can guide and help them throughout their software development journey towards creating repositories/projects that not only meet their own immediate goals but also can also inspire the GitHub community. The outcomes of this project and research have the potential to create best practices in the software development community guide them to become better, and foster a collaborative ecosystem on the GitHub platform.

During this capstone project, there were several crucial stages that the work went through. As the goal of our project is to determine underlying patterns that contribute to the success of the repositories, there are three main stages that the work was divided into. The first stage is the data retrieval process, followed by data analysis procedures and machine learning algorithms, and the last stage is for Power Bi analysis, the creation of a user-friendly and informative dashboard for easy usage and access. There are bridges that connect the crucial stages so that the processes are automated as much as possible. Those bridges are pipelines that start from the database collection, connect to the data analysis and machine learning part, and flow to the dashboard. It was very important to create a smooth data transfer and analysis process. In the following chapters, we will discuss each stage and process more thoroughly.

Problem Statement

The problem that our capstone project aims to resolve is the need to enhance repository development practices on GitHub platform in order to help developers understand what helps repositories become successful. Even though there are many repositories present in GitHub there is no such indicator of success. Intuitively it can be thought that Stargazers, Forks, Subscribers, and other such features are the ones who contribute to the spiking of the repository. That is a dilemma for the software community that no one can define how the repositories go viral, become successful, and be used by many.

Literature Review

1. ***“Understanding the Factors that Impact the Popularity of GitHub Repositories”*** By Hudson Borges, Andre Hora, Marco Tulio Valente.

   The hypothesis of the paper titled "The Impact of Social Features on GitHub Repositories" revolves around the idea that social features such as interactions between users significantly influence the success and popularity of GitHub repositories. Specifically, the hypothesis states that repositories with higher levels of social engagement, as indicated by metrics like stars, forks, and comments, are more likely to attract attention, contributions, and usage from the GitHub community. As this paper explores the influence of social features such as stars, forks, and comments on the success of the GitHub repositories, it inspired and helped us to finalize our decision on using the available features in our dataset such as stargazers, forks, subscribers, etc. Whereas our project is more focused on the attributes mentioned before this paper focuses on the impact of social interactions and community engagement on the repository’s popularity.

“We identified four patterns of popularity growth, which were derived after clustering the time series that describe the number of stars of the systems in our dataset. We found that slow growth is the most common pattern (65.7%) and that very few systems present viral behavior (1.6%). Slow growth is more common in case of overpopulated application domains (as web libraries and frameworks) and for old repositories.” (Hudson Borges, Andre Hora, Marco Tulio Valente, 2016, p. 10). We found this result interesting and expectable from our analysis as well, even though it highly depends on the data we have.

2. ***"A Comprehensive Study of Software Forks: Dates, Reasons and Outcomes"*** by Robles et al. (2016)

This paper investigates software forks on GitHub. It examines temporal patterns, reasons for forking, and outcomes of forked projects. Through statistical analysis and classification methods, the paper identifies common reasons for forking like bug fixes, and feature enhancements. It also explores factors associated with successful or unsuccessful outcomes. It was interesting to see how machine learning algorithms were used in such cases. This study helped us to be sure of the final choice of our model selection, even though it highly depends on the performance of the models. They used decision trees to analyze their data, which gave us the idea of trying a random forest classifier on our analysis that actually yielded the best results out of all.

3. ***“Predicting the Popularity of GitHub Repositories”*** by Hudson Borges, Andre Hora, Marco Tulio Valente

    This project focuses on predicting the number of stars of GitHub repositories using multiple linear regression, which helped with our model selection. As it is defined in the paper “For example, GitHub users can show appreciation to projects by adding stars to them. Therefore, the number of stars of a repository is a direct measure of its popularity.” (Hudson Borges, Andre Hora, Marco Tulio Valente
    ,2016, p.1). Basically, stars serve as a measure of a repository's popularity, and accurate predictions are valuable for both repository owners and clients seeking insights into project performance in the competitive open-source development market. While seeking for the right choice of the target variable this study was a catch to determine that. The study demonstrates that the proposed models achieve accurate predictions, particularly when trained with data from the last six months of star counts. This helped us understand that this model will not be suitable for our case as the data that we have is very large and the differences between the stars of the repositories are very random as the project recommends specific models tailored to repositories with slow growth or fewer stars.

4. ***"Scoring Popularity in GitHub”*** by Abduljaleel Al-Rubaye, Gita Sukthankar

This is a study on popularity in GitHub that is focusing on the relationship between forking, watching, and starring repositories as quantitative measures of popularity. While we were thinking about how to define the target variable we even though of incorporating of some formula or calculate weights to give to the features we find useful. The paper proposes a weight-based popularity score (WTPS) derived from the history of repository popularity indicators on GitHub. This score aims to provide a comprehensive measure of a repository's popularity based on user interactions such as forking, watching, and starring. While this is a great way to have a popularity score measure, unfortunately, this helped us understand that we cannot do this unless we have the right data to do so. The data used includes every update about the repository since the day it was created. Therefore, it was possible to calculate the growth of the repository, while in our case we do not have that kind of data.

Methodology

---

1. **Exploratory Data Analysis (EDA):**
   - Used Python analysis techniques such as understanding the types of our features, fixing the shape of the dataset, filtering some values to see what repositories include those, etc.
   - Used different kinds of visualizations to better grasp the data.
2. **Time Series Analysis:**
   - For capturing trends and identifying patterns we used scatterplots incorporated with different features.

3. **Feature Engineering:**
   - For feature engineering several columns were created: Repository_Age, Updated_In_Days, Forks-to-Network_Ratio, and Issues-to-Forks_Ratio.
   - Some variables were time-based so those were converted to datetime.
4. **Supervised Learning Models:**
   - For supervised learning linear regression, gradient boosting, and random forest regressor were used.
   - These were selected based on the literature reviews and browsing the internet as they suited our dataset the most.
5. **Unsupervised Learning Models:**
   - For unsupervised learning we used K-Means, DBSCAN, Mean Shift Clustering, and Gaussian Mixture models (GMM).
6. **Software and Tools:**
   - The core language was Python, we used MySQL for databases, Power BI for dashboard creation, and Database works with ELT.

Results

Data Retrieval

The first stage was to collect the required data for our analysis. Since we were dealing with a vast amount of data, it was crucial to find an efficient approach.

Initially, we attempted to retrieve data directly using the GitHub API. However, due to its limitations in restricting large-scale data retrieval, we came up with an alternative approach. Additionally, there were no applicable open-source datasets related to GitHub available on the web. To overcome these difficulties, a Python script was developed to interact with the GitHub API using a personal access token and ultimately extract detailed data related to users and repositories as shown in *Table 1* below.

Column Name	Data Type	Description
User_ID	INTEGER	Unique identifier for the user.
User_Node_ID	STRING	Node identifier for the user.
User_Avatar_URL	STRING	URL for the user's avatar image.
User_HTML_URL	STRING	URL for the user's profile page.
User_Type	STRING	Type of user (e.g., "User", "Organization").
User_Site_Admin	BOOLEAN	Indicates if the user is a site administrator.

User_Name	STRING	Username of the user.
User_Company	STRING	Company affiliation of the user.
User_Blog	STRING	URL of the user's blog.
User_Location	STRING	Location information provided by the user.
User_Email	STRING	Email address of the user.
User_Bio	STRING	Biography or description provided by the user.
User_Twitter_Username	STRING	Twitter username of the user.
User_Public_Repos	INTEGER	Number of public repositories owned by the user.
User_Followers	INTEGER	Number of followers of the user.
User_Following	INTEGER	Number of users followed by the user.
User_Created_At	TIMESTAMP	Date and time when the user account was created.
User_Updated_At	TIMESTAMP	Date and time when the user account was last updated.
ID	INTEGER	Unique identifier for the repository.
Node_ID	STRING	Node identifier for the repository.
Name	STRING	Name of the repository.

Full_Name	STRING	Full name of the repository.
Owner_ID	INTEGER	ID of the user or organization that owns the repository.
HTML_URL	STRING	URL for the repository's HTML page.
Description	STRING	Description of the repository.
Fork	BOOLEAN	Indicates if the repository is a fork.
Created_At	TIMESTAMP	Date and time when the repository was created.
Updated_At	TIMESTAMP	Date and time when the repository was last updated.
Pushed_At	TIMESTAMP	Date and time of the last push to the repository.
Git_URL	STRING	URL for the repository's Git endpoint.
Homepage	STRING	URL of the repository's homepage.
Size	INTEGER	Size of the repository in kilobytes.
Stargazers_Count	INTEGER	Number of users who have starred the repository.
Language	STRING	Primary programming language used in the repository.
Has_Issues	BOOLEAN	Indicates if the repository has open issues.
Has_Projects	BOOLEAN	Indicates if the repository has projects.

Has_Downloads	BOOLEAN	Indicates if the repository has downloads.
Has_Wiki	BOOLEAN	Indicates if the repository has a wiki.
Has_Pages	BOOLEAN	Indicates if the repository has GitHub Pages enabled.
Has_Discussions	BOOLEAN	Indicates if the repository has discussions enabled.
Forks_Count	INTEGER	Number of forks of the repository.
Archived	BOOLEAN	Indicates if the repository is archived.
Open_Issues_Count	INTEGER	Number of open issues in the repository.
License_Name	STRING	Name of the license under which the repository is distributed.
License_SPDX_ID	STRING	SPDX identifier for the license.
License_URL	STRING	URL for the license information.
License_Node_ID	STRING	Node identifier for the license.
Is_Template	BOOLEAN	Indicates if the repository is a template.
Web_Commit_Signoff_Required	BOOLEAN	Indicates if web commits require sign-off.
Default_Branch	STRING	Default branch of the repository.
Network_Count	INTEGER	Number of forks of the repository in the network.

Subscribers_Count	INTEGER	Number of users subscribed to notifications for the repository.
Ingestion_Date	TIMESTAMP	Date and time when the data was ingested or imported into the system.

Table 1: Fetched Columns with Data Types and Description

Furthermore, to expand the dataset, another script was developed to extract followers and followings of users already present in the dataset and saved into a TXT file. These usernames are later used to discover and retrieve information on even more repositories, creating a process that could potentially expand the dataset indefinitely.

Data Analysis

During the second stage, that is the data analysis section, the retrieved data was analyzed through Python. In order to have a good understanding of what we are working with it is important to know the datatypes of the variables, the size of the data, and column names that you can find in the analysis file. Very essential part of an analysis is finding and identifying missing values in the columns that will be used during the process of both data analysis and machine learning.

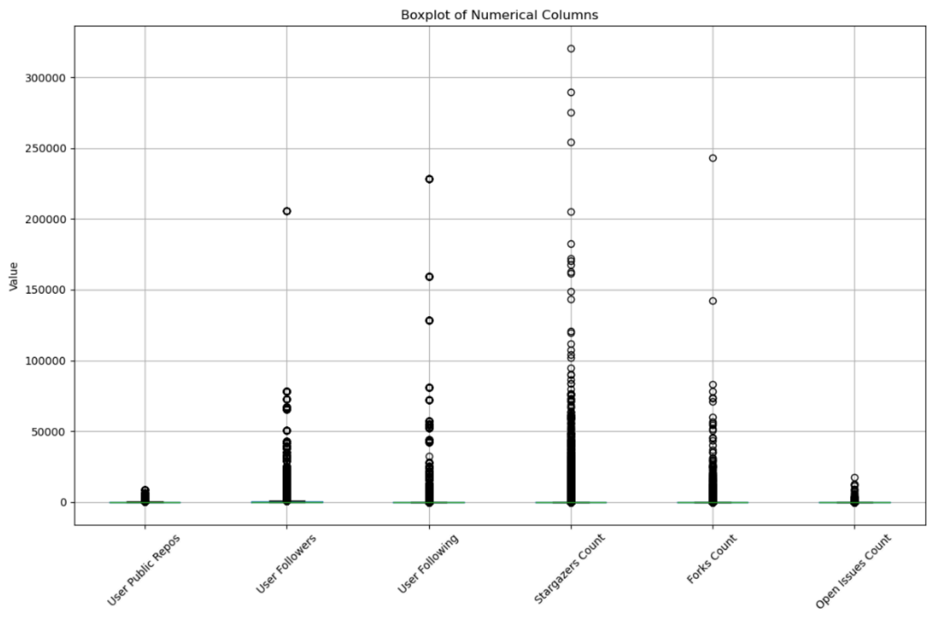


Fig. 1. Boxplot of Numerical Columns

As one can deduce from Fig. 1 there are clear outliers in all of the columns. It is worth mentioning that in some columns, comparably to other ones, there are more severe and noticeable outliers that are vividly present. After checking those eye-catching data points it was clear that those outliers are big projects for example the "Free Ebook Foundation" which has the most stargazers and is a big ebook organization. Overall, in this case, all of the outliers were meaningful and not just random noises in the data.

The next step here was the time series analysis. For this, we used the "Creation Date" from the dataset in order to get to know the variables and overall data that we have better. The aim of this was to get one step closer to understanding the possible trends that can be present in the data.

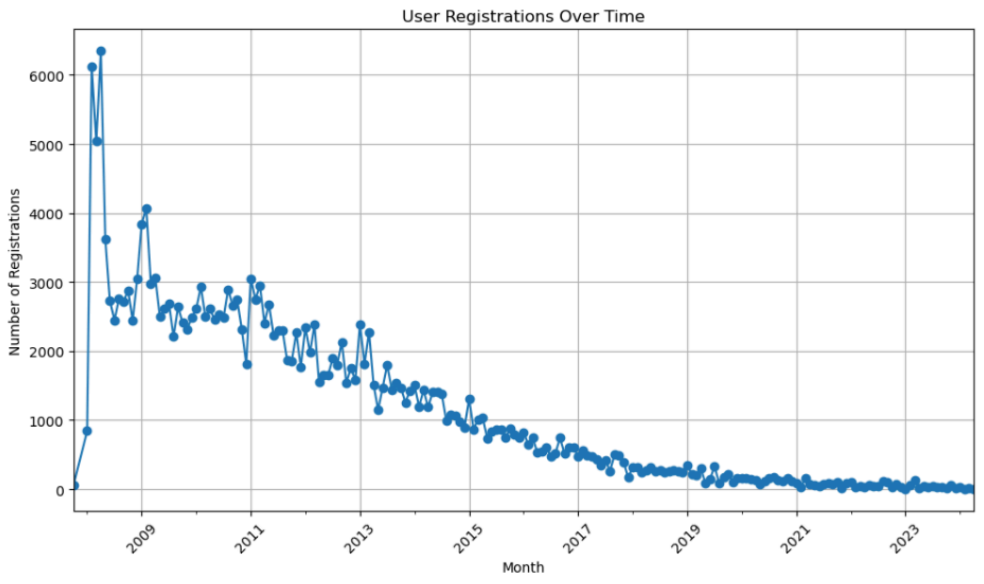


Fig. 2. Time Series Scatter Plot of User Registrations Over Time.

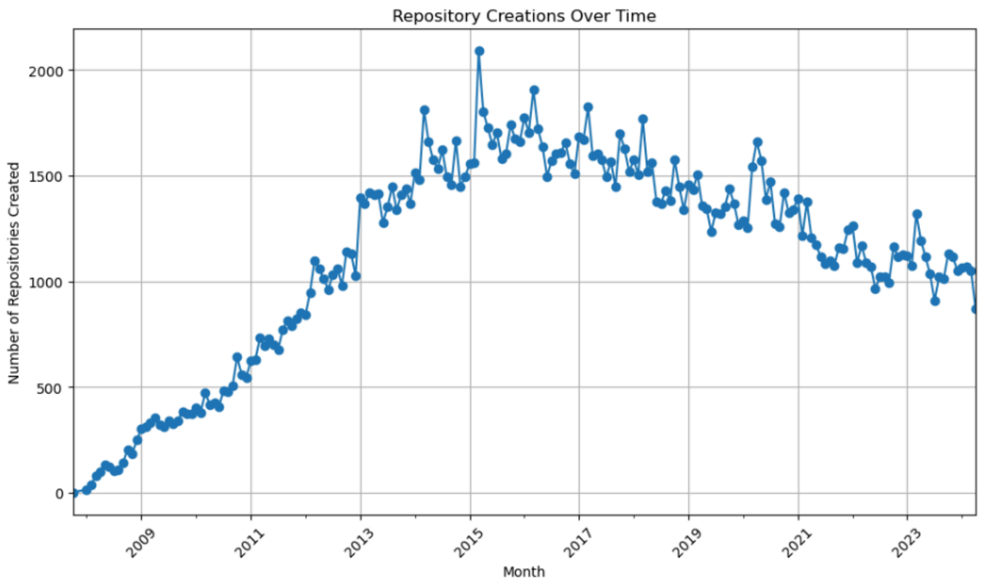


Fig. 3. Time Series Scatter Plot of Repository Creations Over Time.

From fig. 2 we can imply that there has been a decline in the user registration. This can be because of the fact that GitHub was founded in 2008 and at first there was a huge flow of software developers that registered, as the years went by the user registrations went down accordingly. From fig. 3 we can see some spikes in the repository creations, therefore if we want to have an overall understanding of the user activity, then we should rely on Fig. 2 more and state that the activity is stable.

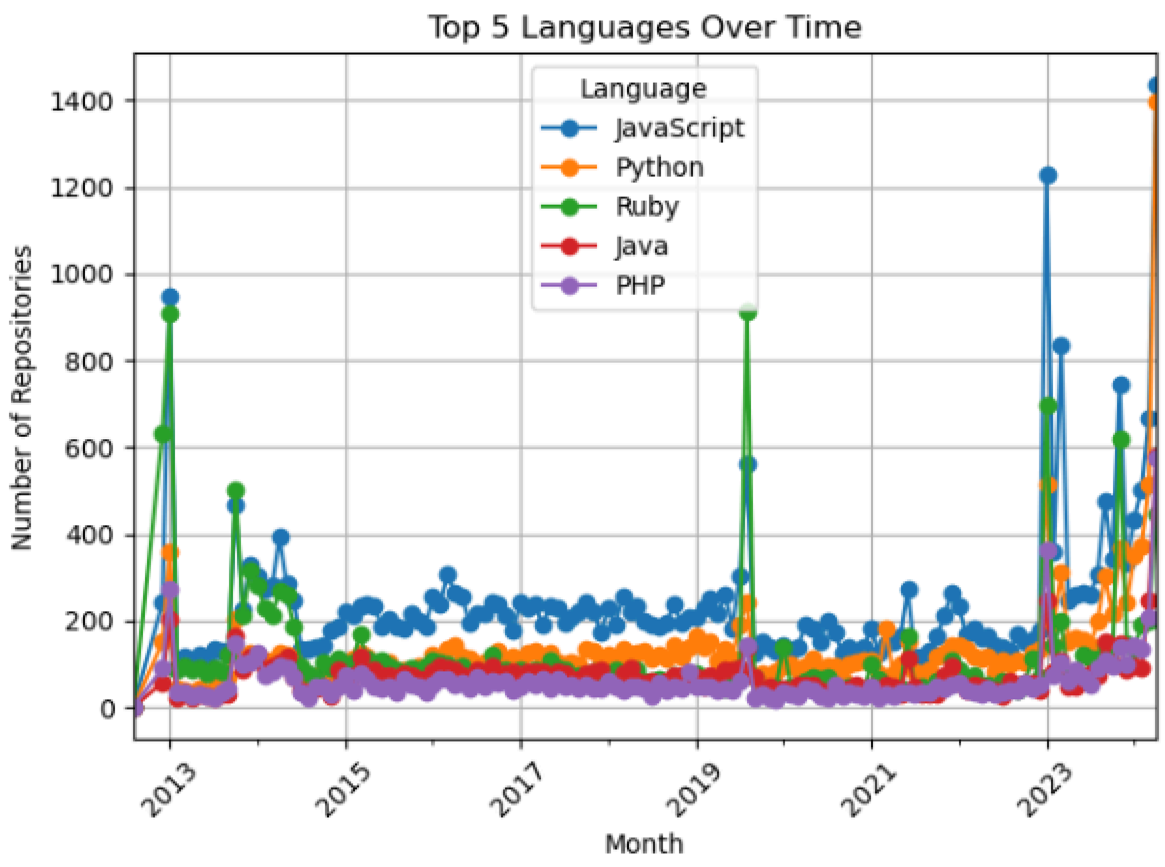


Fig. 4. Time Series Scatter Plot of The Top Used Languages Over Time.

In order to understand the programming language trends, we can take a look at Fig. 4 and deduce that the most used programming language is javascript which doesn’t lose its popularity followed by Python. This was further analyzed in the Power BI Dashboard.

Machine Learning

During the Machine Learning stage we have separated the necessary variables for our machine learning models, which are numerical columns: **'User Followers', 'User Following', 'Size', 'Stargazers Count', 'Forks Count', 'Open Issues Count', 'Subscribers Count', 'Repository Age', 'Updated In Days'**. Additionally, there were variables with boolean values that were encoded: **'User Type', 'Fork', 'Has Issues', 'Has Projects', 'Has Downloads', 'Has Wiki', 'Has Pages', 'Has Discussions', 'Archived'**.

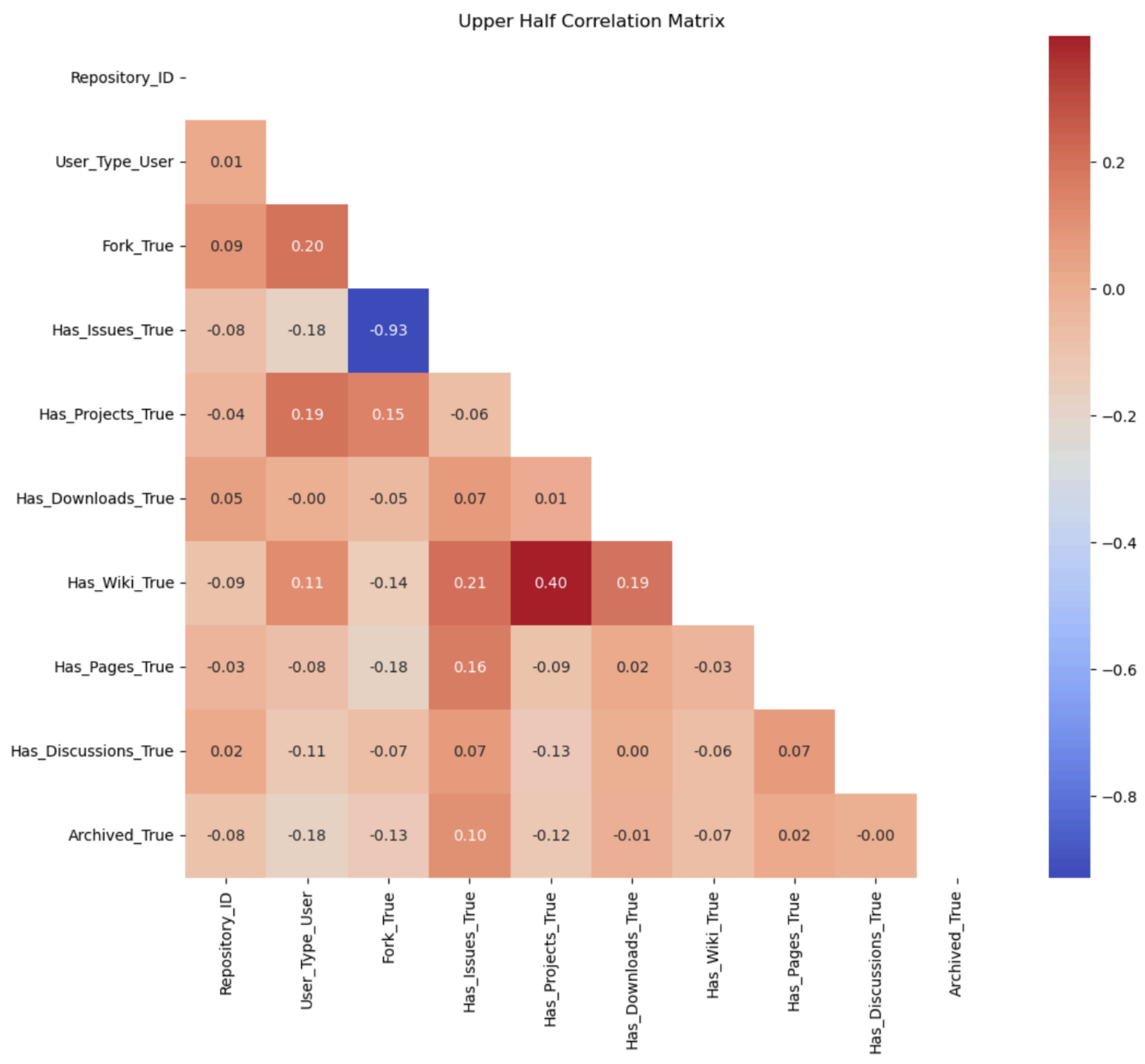
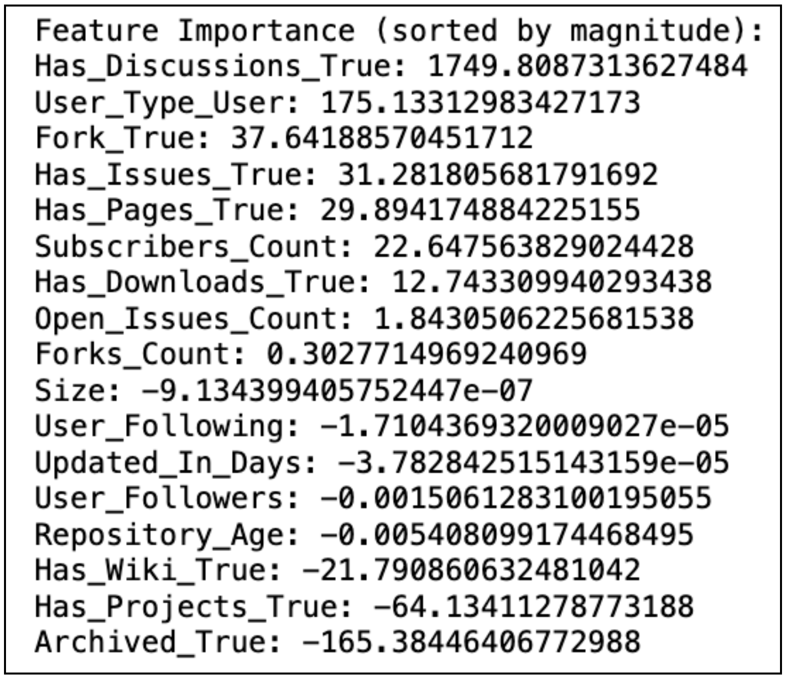


Fig. 5. Correlation Matrix for the Machine Learning variables.

From the above Fig. 5, we can say that there are no significant correlations present, the most noticeable one is between "Has Wiki_True" and "Has Projects_True". Therefore we could not rely much on the correlations.

For this project, as we could approach it from both a supervised learning perspective and not supervised one, we decided to try both and see what patterns we could find. The supervised learning consisted of taking "Stargazers Count" as the target variable for the prediction and classification. The choice lies within the fact that the stargazers are the starred favorites of users, which implies that if a repository has many stargazers therefore it is a crowd favorite. Starting off with **Linear Regression** we obtained the result of **R-squared: 0.817650837896611**, this indicates that about 81% of the target variable, that is in our case the "Stargazers Count", can be explained by the independent variables in the regression model. In order to understand each variable's contribution to the model, we extracted how the regression model gave feature importance to the independent variables.

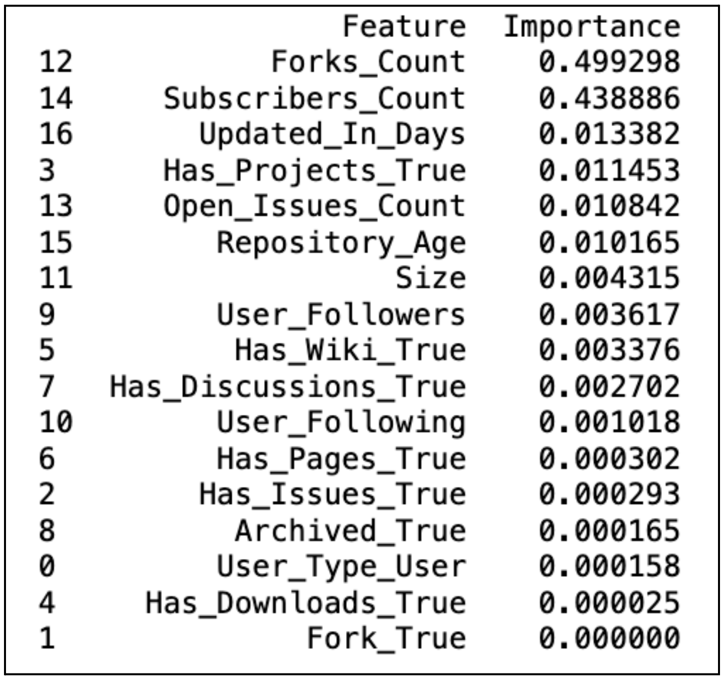


```
Feature Importance (sorted by magnitude):
Has_Discussions_True: 1749.8087313627484
User_Type_User: 175.13312983427173
Fork_True: 37.64188570451712
Has_Issues_True: 31.281805681791692
Has_Pages_True: 29.894174884225155
Subscribers_Count: 22.647563829024428
Has_Downloads_True: 12.743309940293438
Open_Issues_Count: 1.8430506225681538
Forks_Count: 0.3027714969240969
Size: -9.134399405752447e-07
User_Following: -1.7104369320009027e-05
Updated_In_Days: -3.782842515143159e-05
User_Followers: -0.0015061283100195055
Repository_Age: -0.005408099174468495
Has_Wiki_True: -21.790860632481042
Has_Projects_True: -64.13411278773188
Archived_True: -165.38446406772988
```

Fig. 6. Feature Importances of Linear Regression Model.

The distribution of importances looks intuitive and insightful (fig. 6), by using these we can say that if the repository has discussions, forks, issues, pages, downloads, etc. that shows a popularity pattern. The higher these values the more popular the repository gets. The first pattern is found!

As it is crucial to try different models before making conclusions, we continued with **Gradient Boosting**, which gave the result **R-squared: 0.8710061090160807**. This shows that about 87% of the target variable, that is in our case is again the “Stargazers Count”, can be explained by the independent variables in the gradient boosting model.



```
                 Feature  Importance
12           Forks_Count    0.499298
14     Subscribers_Count    0.438886
16        Updated_In_Days   0.013382
3      Has_Projects_True    0.011453
13     Open_Issues_Count    0.010842
15        Repository_Age    0.010165
11                  Size    0.004315
9         User_Followers    0.003617
5          Has_Wiki_True    0.003376
7   Has_Discussions_True    0.002702
10        User_Following    0.001018
6         Has_Pages_True    0.000302
2        Has_Issues_True    0.000293
8          Archived_True    0.000165
0         User_Type_User    0.000158
4     Has_Downloads_True    0.000025
1              Fork_True    0.000000
```

Fig. 7. Feature Importances of Gradient Boosting Model.

This feature importance distribution (fig. 7) is quite different from the previous one. The importance measures that are given to the variables did not meet our expectations as there are many columns that by judging our domain knowledge should have been higher ranked.

For the last model we used Random Forest Regressor, whereas previously for the two models, we again used as target the "Stargazer Count". From this model, we obtained **R-squared (R^2): 0.8834127055318827,** which indicates that the independent variables explain the target with about 88% accuracy.

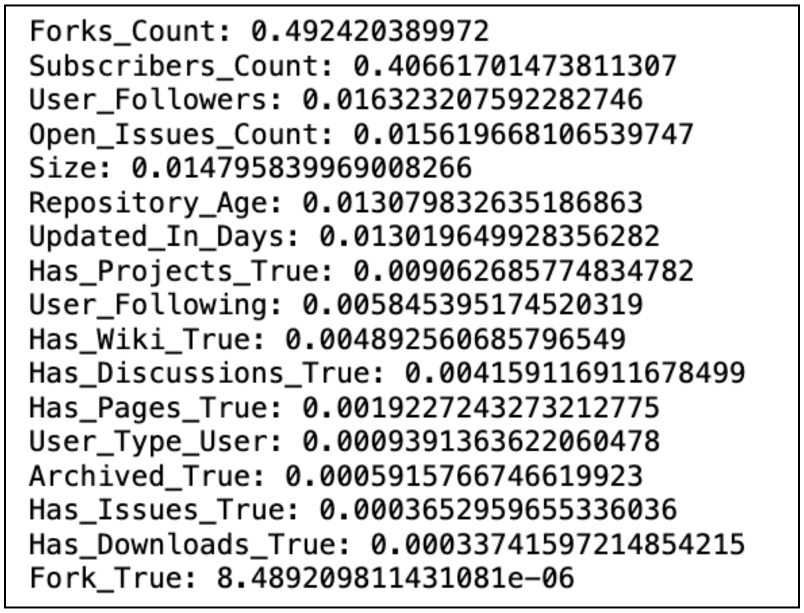
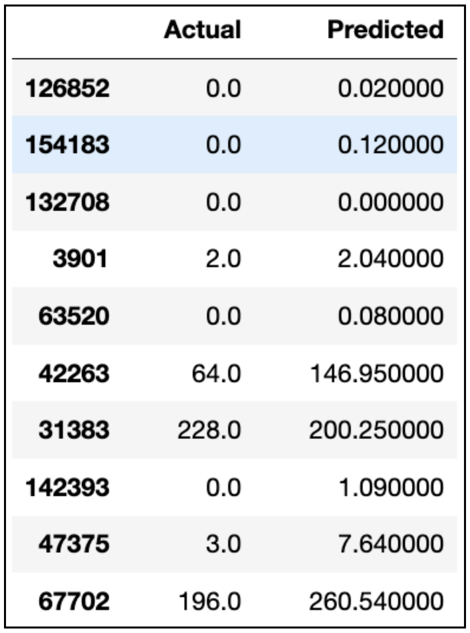

```
Forks_Count: 0.492420389972
Subscribers_Count: 0.40661701473811307
User_Followers: 0.016323207592282746
Open_Issues_Count: 0.015619668106539747
Size: 0.014795839969008266
Repository_Age: 0.013079832635186863
Updated_In_Days: 0.013019649928356282
Has_Projects_True: 0.009062685774834782
User_Following: 0.005845395174520319
Has_Wiki_True: 0.004892560685796549
Has_Discussions_True: 0.004159116911678499
Has_Pages_True: 0.0019227243273212775
User_Type_User: 0.0009391363622060478
Archived_True: 0.0005915766746619923
Has_Issues_True: 0.0003652959655336036
Has_Downloads_True: 0.0003374159721485421 5
Fork_True: 8.489209811431081e-06
```

Fig. 8. Feature Importances of Random Forest Model.

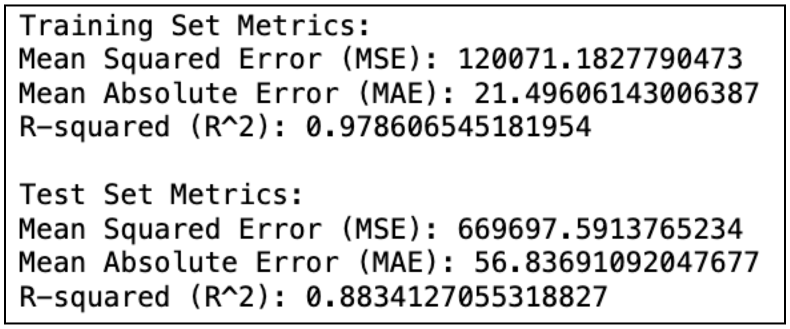
This model had promising results, as it both was the best-performing one amongst the three and the feature importance distribution (fig. 8) is quite intuitive and meets our domain knowledge's expectations. Therefore, we moved on with prediction and actual value comparison.



	Actual	Predicted
126852	0.0	0.020000
154183	0.0	0.120000
132708	0.0	0.000000
3901	2.0	2.040000
63520	0.0	0.080000
42263	64.0	146.950000
31383	228.0	200.250000
142393	0.0	1.090000
47375	3.0	7.640000
67702	196.0	260.540000

Fig. 9. Random Forest's Prediction comparison.

The test result was very impressive therefore we moved on with the predictions over the dataset. The result we obtained for that is **R-squared (R^2): 0.9545406562830735**. This needed a check for overfitting or underfitting. We evaluated the performance of our model which helps in understanding how well the model generalizes to unseen data (test set) and whether it is overfitting or underfitting (by comparing performance on training and test sets).



```
Training Set Metrics:
Mean Squared Error (MSE): 120071.1827790473
Mean Absolute Error (MAE): 21.49606143006387
R-squared (R^2): 0.978606545181954

Test Set Metrics:
Mean Squared Error (MSE): 669697.5913765234
Mean Absolute Error (MAE): 56.83691092047677
R-squared (R^2): 0.8834127055318827
```

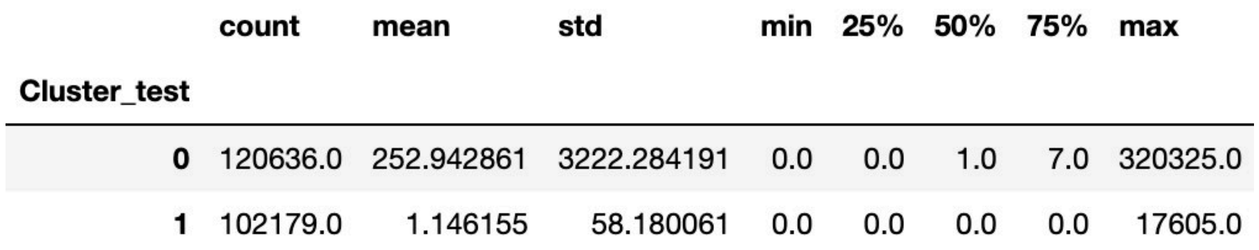
Fig. 10. Performance Evaluation Metrics.

Overall, while there are signs of overfitting (lower performance on the test set compared to the training set), the magnitude of the difference between the training and test metrics is not extreme. As there is some degree of overfitting present it would be good to further investigate and tune the model to reduce overfitting. After conducting cross-validation the results that we obtained are **Mean Train R-squared (R^2): 0.9776625589347626** and **Mean Test R-squared (R^2): 0.8834127055318827.** These results suggest that the model is overfitting to the training data and does not generalize well to unseen data. Even though the overfitting issue is still present we can try to reduce overfitting by restricting the depth or complexity of individual trees and limiting the number of features considered at each split. After interchanging the variables and the depth of the trees to 8, we obtained a better result **R-squared (R^2): 0.8273635791714308** and **R-squared (R^2): 0.9055433843025674**, the overfitting reduced significantly.

The next step in the machine learning analysis is the unsupervised learning section. As in our dataset, there is no certain column for a popularity indicator, the best thing to do here is to conduct both supervised and unsupervised learning techniques. Before now, we used supervised learning and chose the "Stargazers Count" as the target variable, but there is no certainty that the chosen column is the perfect indicator for the success metric. As there is no certainty here, it is useful to investigate the problem from different views. The unsupervised learning methods help to detect clusters that can potentially have hidden patterns that tell you the story of a repository.

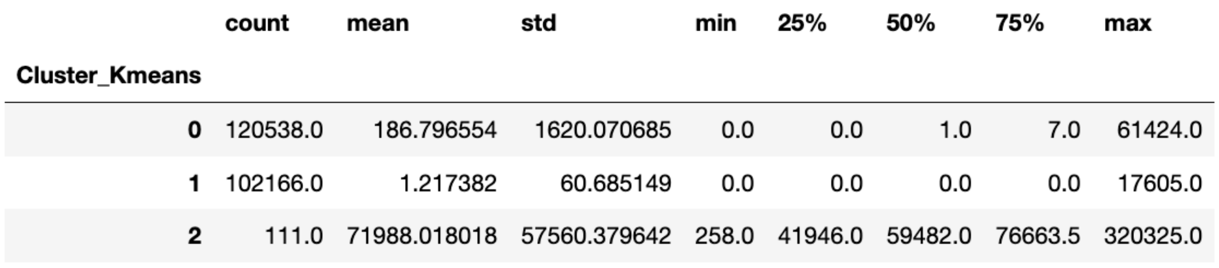
The first algorithm we tried was **K-Means**. The trickiest part of this was choosing the optimal k for the algorithm. We conducted GridSearch for this which yielded the result of having k set to 2. After identifying the clusters, the number of k clusters did not do the job well as the 2 clusters were very inefficient. The mean of the first cluster is **252.94** and for the second one is

1.15. This wasn’t a good-looking result of two clusters as there are many repositories that have very high stargazers (e.g. 150.000+) which does not make sense to include in a cluster with a mean value of 252. Therefore, we tried with k as 3,4,5, etc. The best outcome was **k = 3** as it provided clusters that can be categorized as High, Medium, and Low. The mean values that we got accordingly are **71988.018, 186.796554, and 1.217382.**



	count	mean	std	min	25%	50%	75%	max
Cluster_test								
0	120636.0	252.942861	3222.284191	0.0	0.0	1.0	7.0	320325.0
1	102179.0	1.146155	58.180061	0.0	0.0	0.0	0.0	17605.0

Table 2. K-Means Descriptive Results K = 2.

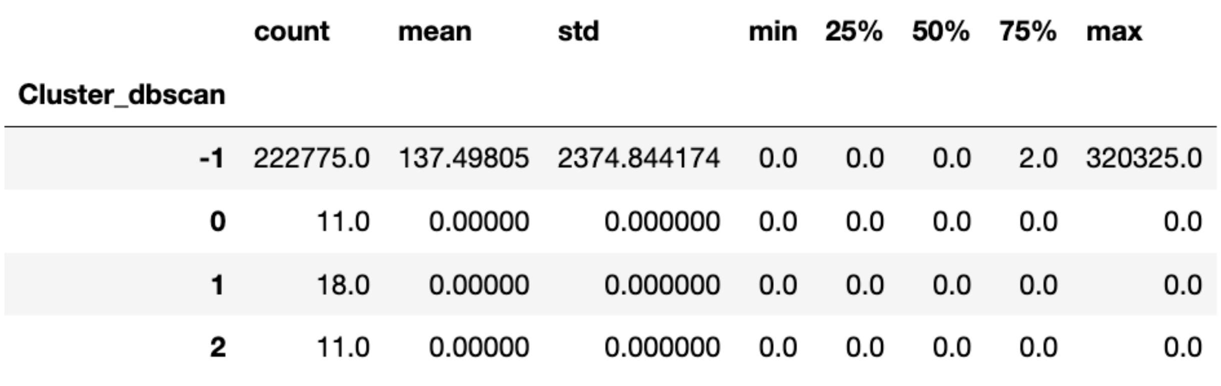


	count	mean	std	min	25%	50%	75%	max
Cluster_Kmeans								
0	120538.0	186.796554	1620.070685	0.0	0.0	1.0	7.0	61424.0
1	102166.0	1.217382	60.685149	0.0	0.0	0.0	0.0	17605.0
2	111.0	71988.018018	57560.379642	258.0	41946.0	59482.0	76663.5	320325.0

Table 3. K-Means Descriptive Results K = 3.

The second clustering algorithm we tried was **DBSCAN**. Which did not perform well on the data, even after modifying the metrics. We had 4 clusters and the mean values of the

Stargazers for each were **137.5, 0, 0, 0**.

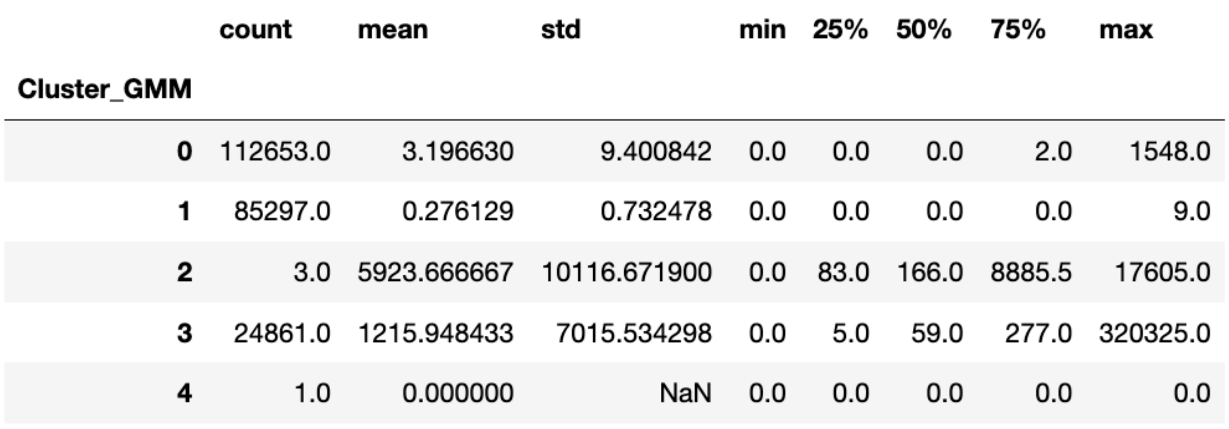


Cluster_dbscan	count	mean	std	min	25%	50%	75%	max
-1	222775.0	137.49805	2374.844174	0.0	0.0	0.0	2.0	320325.0
0	11.0	0.00000	0.000000	0.0	0.0	0.0	0.0	0.0
1	18.0	0.00000	0.000000	0.0	0.0	0.0	0.0	0.0
2	11.0	0.00000	0.000000	0.0	0.0	0.0	0.0	0.0

Table 4. DBSCAN Descriptive Results.

The **Mean Shifting Clustering** probably performed the worst on our data as it identified too many clusters from which it was impossible to derive any information or find any underlying pattern that contributes to any of those clusters.

The Gaussian Mixture Models Clustering gave interesting results as it identified 5 clusters but unlike DBSCAN the results of this algorithm made more sense in terms of the stargazer mean values. For the first cluster, the mean was **3.2**, for the second one **0.28**, third **5923.7**, fourth **1215.95**, fifth **0**.



Cluster_GMM	count	mean	std	min	25%	50%	75%	max
0	112653.0	3.196630	9.400842	0.0	0.0	0.0	2.0	1548.0
1	85297.0	0.276129	0.732478	0.0	0.0	0.0	0.0	9.0
2	3.0	5923.666667	10116.671900	0.0	83.0	166.0	8885.5	17605.0
3	24861.0	1215.948433	7015.534298	0.0	5.0	59.0	277.0	320325.0
4	1.0	0.000000	NaN	0.0	0.0	0.0	0.0	0.0

Table 5. GMM Descriptive Results.

Eventually, we decided to move on with the K-Means clustering with k = 3.

Dashboard

To make the obtained results more user-friendly and practical for the software development community the last stage of the project was creating a dashboard. It consists of 6 pages (here excluding the HOME page):

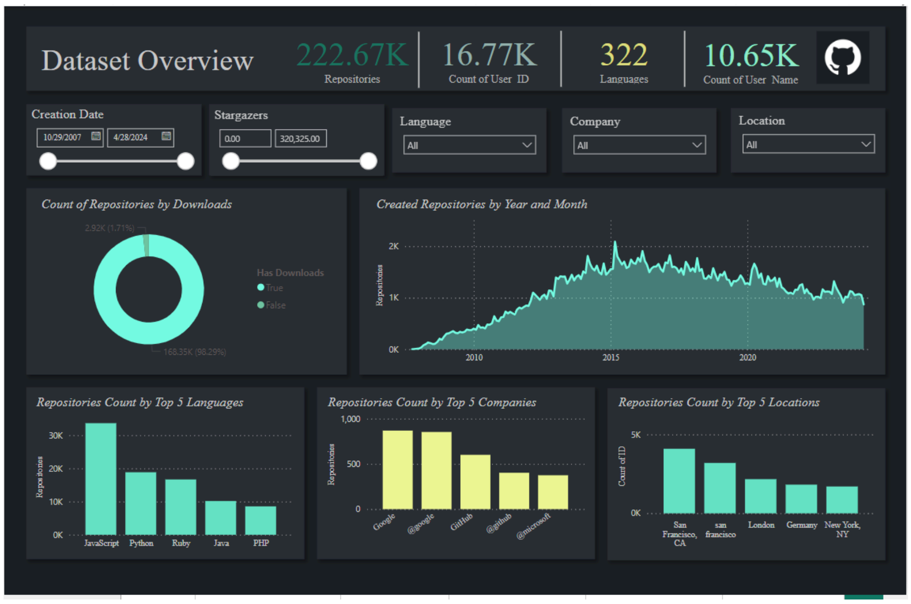


Fig. 11. Repository Analysis Dashboard Section.

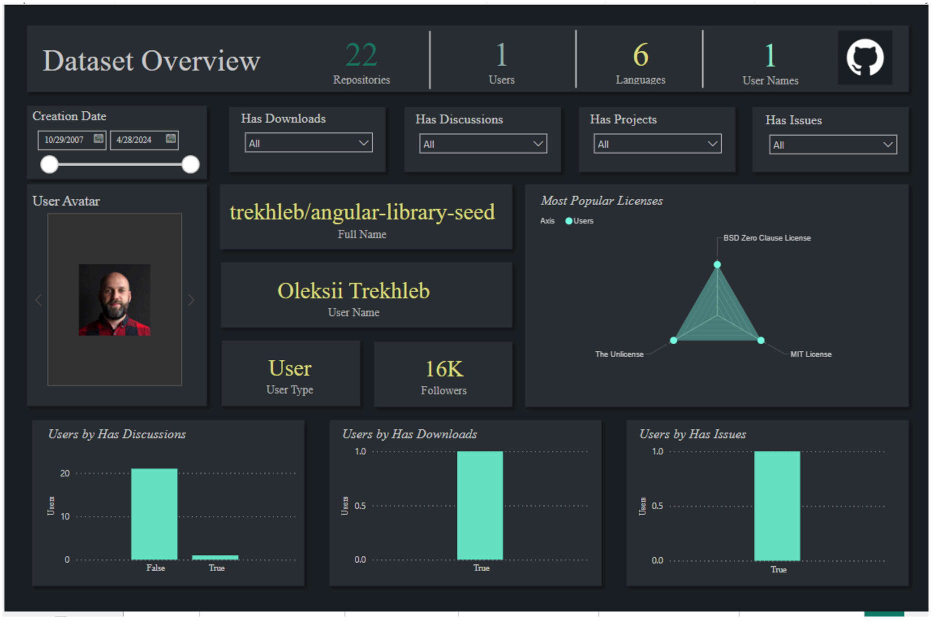


Fig. 12. User Analysis Dashboard Section.

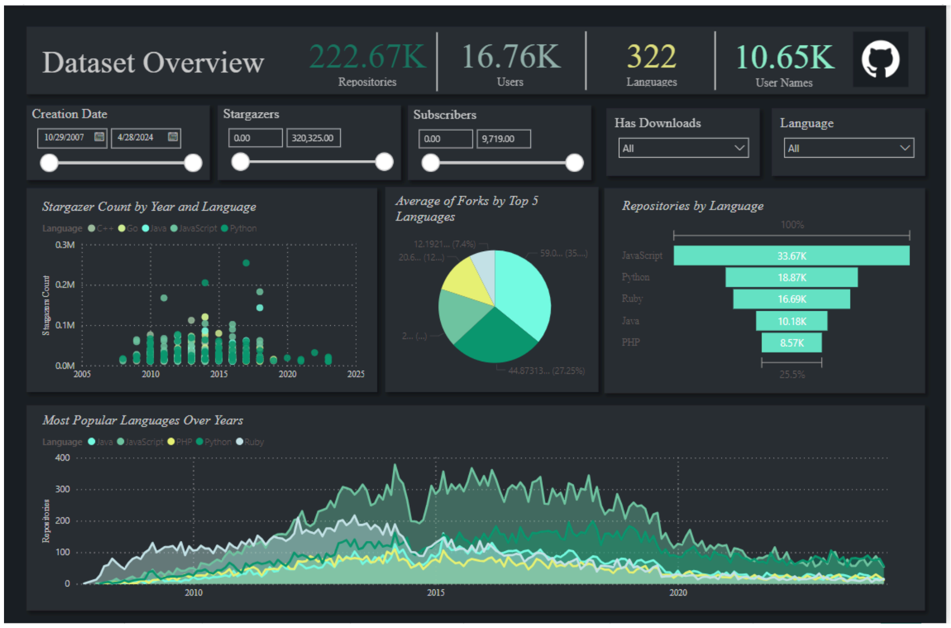
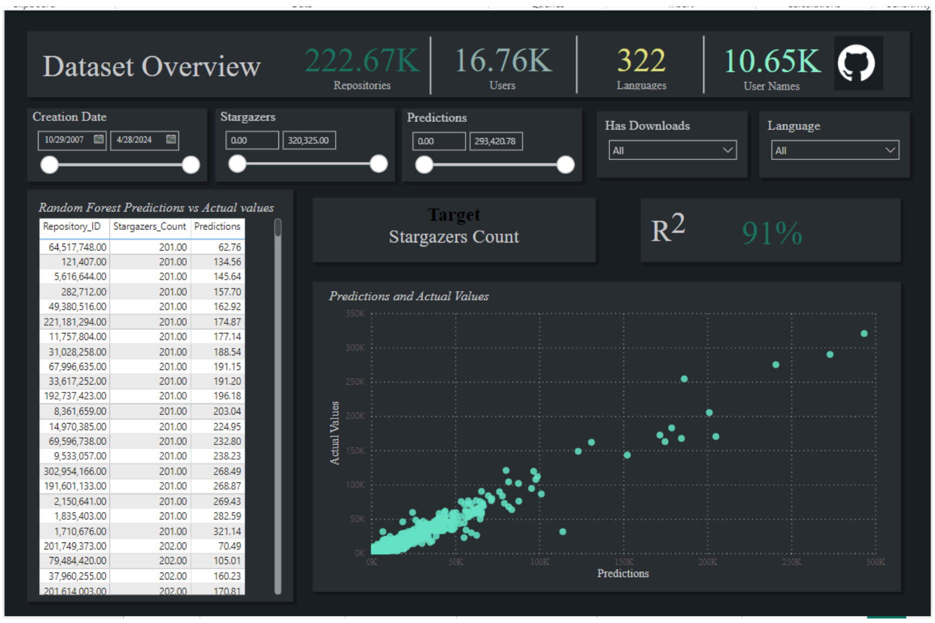


Fig. 13. Language Analysis Dashboard Section.



Random Forest Predictions vs Actual values

Repository_ID	Stargazers_Count	Predictions
64,517,748.00	201.00	62.76
121,407.00	201.00	134.56
5,616,644.00	201.00	145.64
282,712.00	201.00	157.70
49,380,516.00	201.00	162.92
221,181,294.00	201.00	174.87
11,757,804.00	201.00	177.14
31,028,258.00	201.00	188.54
67,996,635.00	201.00	191.15
33,617,252.00	201.00	191.20
192,737,423.00	201.00	196.18
8,361,659.00	201.00	203.04
14,970,385.00	201.00	224.95
69,596,738.00	201.00	232.80
9,533,057.00	201.00	238.23
302,954,166.00	201.00	268.49
191,601,133.00	201.00	268.87
2,150,641.00	201.00	269.43
1,835,403.00	201.00	282.59
1,710,676.00	201.00	321.14
201,749,373.00	202.00	70.49
79,484,420.00	202.00	105.01
37,960,255.00	202.00	160.23
201,614,003.00	202.00	170.81

Fig. 14. Predictive Analysis Dashboard Section.

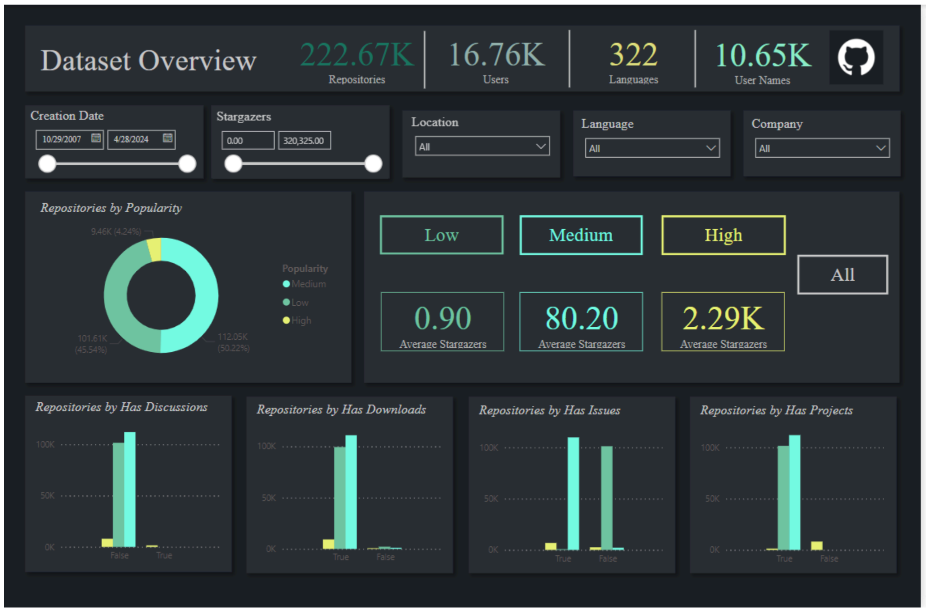


Fig. 15. Cluster Analysis Dashboard Section.

Pipelines

Following the data retrieval phase, we created an ELT (Extract, Load, Transform) pipeline to make data processing and analysis more efficient. This pipeline made the transfer of data from its source all the way to the analysis phase and later to the visualization phase easier and smoother, ensuring reliability throughout the process.

The initial step in the ELT process involved pushing the fetched data CSV file, which we gained from the GitHub API to Google Drive. Using Google Drive allowed us to store all the data in one place before moving it for processing. From Google Drive, the data was then migrated to BigQuery within the Google Cloud Platform (GCP). A designated table named Staging_Raw within BigQuery served as the primary repository for the incoming data.

To organize and structure the data for analysis, a Project and a Dataset were created within BigQuery. Subsequently, SQL creation files were developed to define the schema and structure of the data tables. These SQL creation files were executed using Python scripts, resulting in the creation of Dimension (Dim) tables such as Dim_Users, Dim_Repositories, Dim_Date and Dim_License, and a Fact table Fact_Metrics.

To ensure data accuracy and relevance, update scripts were devised to periodically refresh the data within the BigQuery tables. Each update script targeted a specific table and was executed in a predetermined order using Python scripts. This systematic approach enabled efficient and timely updates to the data, maintaining its integrity and currency.

Following the data preparation phase, a view table was created to join several tables and present the necessary columns required for subsequent data analysis. This view table served as a simplified interface for accessing and querying the data, facilitating smoother analysis processes.

Upon establishing the structured dataset within BigQuery, Python scripts were utilized to connect to the database and conduct data analysis and machine learning (ML) modeling. These analyses included exploring patterns, correlations, and trends within the dataset to derive meaningful insights.

Once the analysis was completed, the analyzed dataset was integrated back into the BigQuery Database. Finally, Microsoft Power BI was employed to connect to the BigQuery Database, enabling the creation of a dynamic dashboard to visualize and communicate the findings effectively.

This pipeline ensured smooth data flow from start to finish, making it easier to analyze and visualize the data, leading to better decisions and practical insights.

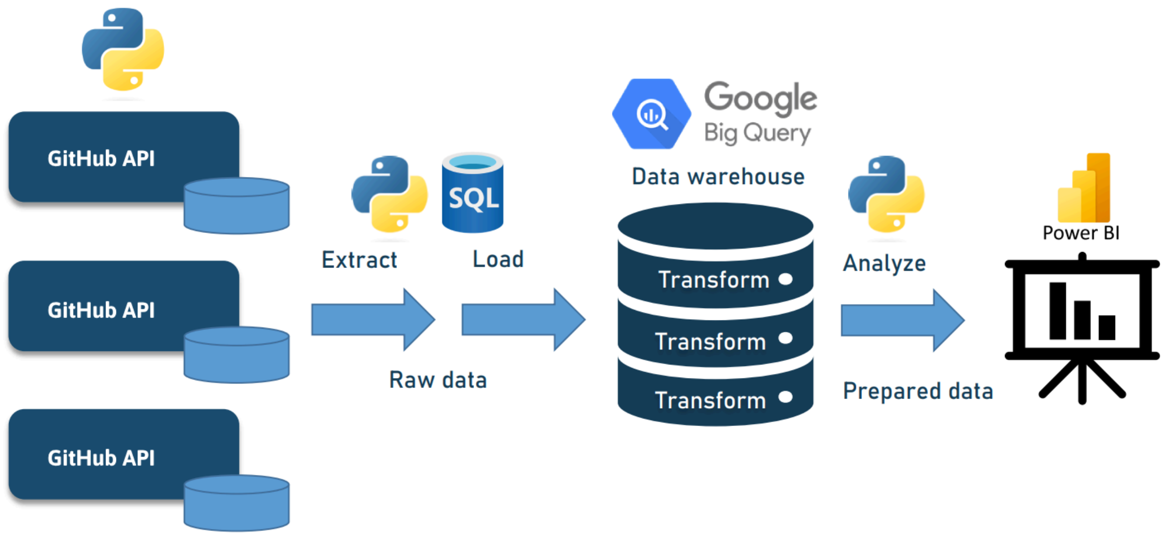


Fig. 16. Data Pipeline Architecture (ELT).

Conclusion

To conclude, we reached our goal and provided solutions to the problem statement mentioned above. The final results that we obtained that can be accessible through the dashboard were predicted Stargazers and derived clusters. The supervised learning section of our analysis provided predictions with a 90%+ accuracy which is a good indicator. That shows that our target variable that is **Stargazers Count** can be explained with the chosen features for the analysis that were Subscribers Count, Forks Count, User Followers, Size, Repository Age, Open Issues Count, Updated In Days, Has Projects_True, User Following, Has Discussions_True, Has Wiki_True, Has Pages_True, User Type_User, Archived_True, Has, Issues_True, Has Downloads_True, Fork_True. From the clustering analysis we could identify patterns that lead to the repository's success. We could identify three clusters: **High**, **Medium**, and **Low**.

The key indicators for the **High** popularity cluster are:

- The repository does not have to have discussions for it to be popular, even though compared to other clusters this one more or less had some repositories included that had discussions.
- The repository has to have downloads because 90% of the high-popularity cluster had downloads.
- The repository can have issues and yet still be successful. Most of the repositories included have issues about 60%.
- The repository does not necessarily need to have projects to be successful. About 90% do not have.
- For the repository to be considered High, it needs to be forked around 2000 times.

- The stargazers of the repository need to fluctuate around 2000 as well.
- The high-popularity users need to have around 2400 followers.

The key indicators for the **Medium** popularity cluster are:

- The repository does not have to have discussions for it to be in the medium cluster at all as 100% of the repositories in this cluster do not have discussions.
- The repository has to have downloads because more than 90% of the medium popularity cluster had downloads.
- The repositories in this cluster mostly have issues. More than 90%.
- The repositories here all have projects.
- For the repository to be considered Medium, it needs to be forked around 38 times.
- The stargazers of the repository need to fluctuate around 187 as well.
- The medium-popularity users need to have 11 followers.

The key indicators for the **Low** popularity cluster are:

- The repository does not have to have discussions at all as 100% of the repositories in this cluster do not have discussions.
- The repository can have downloads.
- The repositories in this cluster mostly if not always have issues. Very close to 100%.
- The repositories here all have projects.
- For the repository to be considered Low, it needs to be forked once..
- The stargazers of the repository need to fluctuate around 2 as well.
- The low-popularity users need to have 2 followers.

Limitations & Future Research

- **Limited data volume:** (around 230,000 data points) due to challenges with GitHub API restrictions (as the famous quote goes "more data leads to more accurate results").
- **Lack of External Validation:** The analysis might lack validation against external benchmarks or ground truth, affecting the interpretation and generalizability of findings.
- **Slow Processing in Jupyter Notebook and PowerBI:** Large data size led to slow processing times and difficulty in handling visualizations in PowerBI and running machine learning models in Jupyter Notebook.
- **Difficulty in Creating Visuals in PowerBI:** Visualizations in PowerBI were prone to memory capacity errors.
- **Algorithm Compatibility Issues:** Certain machine learning algorithms failed to execute properly. Compatibility issues with the dataset or computational environment hindered algorithm performance.
- **Possible future migration to AWS:** Utilizing Amazon Web Services (AWS) instead of Google Cloud Platform (GCP) for data processing and analysis tasks. Leveraging AWS's scalable computing resources and services to overcome performance limitations.

- **Exploring additional metrics:** Future research could explore additional metrics, such as developer activity or project documentation
- **Utilizing TF-IDF in description and bio columns:** Employing Term Frequency-Inverse Document Frequency (TF-IDF) analysis on description and bio columns in the future could reveal significant keywords or terms that contribute to repository success.

References

1. Al-Rubaye, A., & Sukthankar, G. (2020). *Scoring Popularity in GitHub*. https://doi.org/10.1109/csci51800.2020.00044
2. Blincoe, K., Sheoran, J., Goggins, S. P., Petakovic, E., & Damian, D. (2016). Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology*, *70*, 30–39. https://doi.org/10.1016/j.infsof.2015.10.002
3. Borges, H., Hora, A., & Marco Tulio Valente. (2013). Predicting the Popularity of GitHub Repositories. In *https://arxiv.org/pdf/1607.04342.pdf*.
4. Borges, H., Hora, A., & Marco Tulio Valente. (2016). Understanding the Factors that Impact the Popularity of GitHub Repositories. In *https://arxiv.org/pdf/1606.04984.pdf*.
5. Jamali, S., & Rangwala, H. (2009). Digging Digg: Comment Mining, Popularity Prediction, and Social Network Analysis. In *https://cs.gmu.edu/media/techreports/GMU-CS-TR-2009-7.pdf*.