Stars, Suspicion, and Stability: Defining Success in GitHub Repositories

Hashem Al Sailani
Computer Science and Engineering
University of South Florida
Tampa, USA
halsailani@usf.edu

Fatemah Elsewaky
Computer Science and Engineering
University of South Florida
Tampa, USA
fatemahelsewaky@usf.edu

Robin Emmanuel Sardja
Computer Science and Engineering
University of South Florida
Tampa, USA
resardja@usf.edu

Kathryn Bodden
Computer Science and Engineering
University of South Florida
Tampa, USA
boddenk@usf.edu

Nathan Allen
Computer Science and Engineering
University of South Florida
Tampa, USA
allen243@usf.edu

Raiyan Abdul Baten
Computer Science and Engineering
University of South Florida
Tampa, USA
rbaten@usf.edu

***Abstract*—In the dynamic landscape of open source software development on GitHub, understanding the nuances between repository popularity and success becomes essential. This study examines various GitHub repositories and user behaviors to discern the dynamics of repository popularity, the markers of success, and the detection of false actors within the platform. Initially, we explore the characteristics of popular and successful repositories, finding that the factors contributing to success and popularity are not identical. Popularity is quantified by the number of stars, commonly used in previous studies, while success is defined in terms of a project's state-of-the-art claims. Our investigation reveals only a low positive correlation between development consistency, measured through commit activities, and repository stars, suggesting that regular updates alone do not guarantee popularity. Furthermore, we delve into the detection of false actors within the community, analyzing their characteristics and the potential impact on repository metrics. Our results align with prior findings in identifying suspicious behaviors, enhancing understanding of how authenticity affects repository evaluations. However, our research faced limitations, including the inability to apply methods across the entire dataset and difficulties in discerning connections within the network of interactions. Despite these challenges, our findings provide valuable insights into the factors that can enhance repository success, popularity, and credibility on GitHub.**

***Index Terms*—Social Media, GitHub, Popularity, Success, Consistency**

I. Motivation

In the GitHub community, popularity is often equated with the number of stars a repository receives, serving as a social endorsement that suggests a project's perceived usefulness and quality. These stars operate similarly to "likes" on social media platforms and are a primary indicator for users to gauge which projects warrant attention. However, this study aims to critically assess whether this measure of popularity correlates with 'success' in open source projects. Success in these projects can extend to include factors like the impact on the software community, or the achievement of specific, innovative goals. This raises a crucial question about the divergence between popularity and actual success, and whether popular repositories consistently deliver on these broader success metrics. By exploring the nuanced dynamics between what is considered successful and actual project outcomes, we aim to provide insights into effective project development and community engagement on GitHub. As such, the following research questions will tackle key aspects of repository success, user authenticity, and development consistency:

1) *What are the characteristics of popular & successful repositories?*
2) *How can we determine false actors? What makes a GitHub user suspicious?*
3) *How can development consistency affect repository popularity?*

II. Literature Review

A. Stars and Popularity

GitHub, as the largest platform for open source collaboration, features several social coding attributes that significantly influence project popularity, particularly the ability for users to "star" repositories to express approval or interest [1][2][3]. Borges et al. have extensively studied the factors influencing the accumulation of these stars, identifying key influences such as the programming language used, application domain, and the introduction of new features [1]. Their analysis of 2,279 repositories delineates four patterns of popularity growth, providing strategic insights for developers [2].

Expanding on this, Borges et al. used multiple linear regressions to forecast the number of stars a repository might receive. They demonstrated that these models perform optimally with data from the past six months, emphasizing the competitive nature of software development [2]. They further investigated the motivations behind starring projects and found that a significant majority of developers consider star counts before engaging with a project, underscoring the practical and psychological impacts of this feature [3].

Additional measures such as forks and open issues are crucial to comprehensively measure repository popularity. Forks represent an essential dimension of collaboration, often indicating a repository's influence and community engagement. Brisson et al. studied communication within "software families" on GitHub, finding that the depth of forks, the cross-collaboration among repositories, and familial pull requests significantly correlate with higher star counts [4]. This study highlights the intricate communication networks that evolve within forked repositories and their impact on a repository's visibility and popularity.

Open issues also play a critical role in determining a repository's health and activity. Bissyandé et al. conducted a large-scale investigation of issue trackers on GitHub and identified a correlation between the activity in issue trackers and the success of software projects [5]. They demonstrated that active participation in issue reporting and resolution can significantly impact a repository's popularity and perceived reliability.

B. Suspicious Accounts

Previous work has utilized various methods to identify unusual GitHub user behaviors. One study employed the DBSCAN clustering method to analyze commit timing and locations [6], while another analyzed comments in pull requests, focusing on the presence of empty and non-empty comments and their patterns to detect bots [7]. Building on these approaches, our research extends beyond these analyses to consider additional metrics such as follower-to-following ratios, account creation dates, activity durations, and profile completeness. Our findings suggest that false actors display markedly different activity and account behavior compared to genuine users, thus confirming our hypothesis and broadening the understanding of how to identify suspicious actors on GitHub.

C. Development Consistency

Previous studies have investigated the relationship between commit frequency and GitHub repository popularity, but found no compelling correlation [9]. We argue that commit frequency overemphasizes the significance of isolated peaks of "crunch-time development" relative to the broader development timeline. By focusing on development consistency as a correlate of repository popularity, we aim to mitigate the influence of these intensive bursts while appropriately accounting for periods of inactivity (development hiatuses).

III. Data

This research utilizes two comprehensive datasets to explore the multifaceted factors influencing GitHub repository popularity. The first dataset, "A Representative User-centric Dataset of 10 Million GitHub Developers" by Gong et al., is a user-centric JSON file approximately 50 GB in size, detailing the GitHub activities of 10 million developers. It includes nested objects such as `repo_list` and `commit_list`, allowing for an in-depth analysis of user behaviors and their contributions to repository popularity [10].

The second dataset, "GitHub Public Repository Metadata," sourced from Kaggle and compiled by Pelmers, contains metadata for about 3.1 million GitHub repositories. This repository-centric dataset provides broad information about repository attributes, facilitating an understanding of how repository characteristics like the main language, update frequency, and documentation presence influence their popularity within the GitHub community [11].

Both datasets are preprocessed for consistency and to enable combined analysis. From the user-centric dataset, relevant fields are extracted from nested JSON structures to focus on user engagement metrics hypothesized to impact repository popularity. For the repository-centric dataset, data cleaning is performed to remove incomplete records and standardize categorical variables for easier comparison and analysis.

By analyzing these datasets, we aim to identify and quantify the influences on GitHub repository popularity and success, providing comprehensive and actionable insights for developers and project managers in the open source community.

IV. Methodology

Prediction Technique

Our study utilizes multiple linear regression to predict three primary indicators of GitHub repository popularity: annual stars, annual forks, and open issues per year. These indicators are treated as separate dependent variables in our models. The independent variables are derived from a comprehensive set of features that characterize repository attributes, with the inclusion of a manually created 'age' feature representing the maturity of the repository. The regression model for each dependent variable Y_k is formulated as follows:

$$Y_k = b_0 + b_1 X_1 + b_2 X_2 + \ldots + b_r X_r$$

Here, Y_k signifies the k-th dependent variable (stars, forks, or issues per year), X_i are the independent variables from our assembled feature set, and b_j are the regression coefficients. This structured approach allows us to quantitatively assess the influence of various repository characteristics on its popularity metrics.

Estimating the Errors

To determine the accuracy of our models, we calculate the Relative Squared Error (RSE) for each prediction. For a repository r, let $N_k(r)$ be the actual observed value of the k-th dependent variable, and $Nb_k(r)$ the corresponding predicted value derived from our regression model. The RSE for each dependent variable is computed as follows:

$$RSE_k = \left(\frac{Nb_k(r)}{N_k(r)} - 1 \right)^2$$

To evaluate the overall performance of our models across the dataset, we compute the mean Relative Squared Error (mRSE) for each dependent metric, defined as the arithmetic mean of the RSE values across all evaluated repositories:

$$mRSE_k = \frac{1}{|R|} \sum_{r \in R} \left(\frac{Nb_k(r)}{N_k(r)} - 1 \right)^2$$

Feature Selection

To optimize the input feature set for predicting the annual metrics of stars, forks, and open issues on GitHub repositories, LASSO Regression (Least Absolute Shrinkage and Selection Operator) was utilized. LASSO Regression serves as both a regularization and variable selection technique, which helps in enhancing the predictiveness of the regression model while simultaneously reducing the complexity.

The mathematical formulation for LASSO Regression is represented as:

$$\min_{\beta} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

In this formula, y_i denotes the dependent variable vector, X_i represents the matrix of input features, β are the coefficients, and λ is the regularization parameter that controls the extent of sparsity in the coefficient vector β. The objective is to minimize the residual sum of squares subject to a penalty on the absolute size of the coefficients. This penalty term encourages the solution to have fewer non-zero coefficients, effectively reducing the number of features in the model.

SoTA Classification

The classification of state-of-the-art (SoTA) claiming GitHub repositories involves several computational steps to handle and analyze textual data effectively. Below we outline the methodological process employed in our study, detailing each step under sub-headings.

I. Data Preparation: The initial step in our analysis involves creating a standardized list of common machine learning (ML) phrases. This list is utilized to filter and focus the textual content of repository descriptions. These descriptions are then subjected to tokenization, a process where text is broken down into its constituent tokens (words or phrases). We employed the model *"sentence-transformers/all-mpnet-base-v2"* to remove non-essential words, commonly referred to as fluff words, enhancing the relevance of the textual data for further processing.

$$T_d = \text{Tokenize}(D, \text{model})$$

Where T_d represents the tokenized description, and D is the original repository description. The tokenization model aids in refining the content to features more pertinent to our analytical goals.

II. Embedding Conversion + Similarity Score: Following pre-processing, the tokenized repository descriptions T_d and the ML phrases are transformed into embeddings. Embeddings are high-dimensional vectors that capture semantic meanings of the text, allowing computational models to process text-based data effectively. We utilized techniques that convert text into vector space models, where the proximity of vectors indicates semantic similarities:

$$E_r = \text{Embed}(T_d)$$
$$E_m = \text{Embed}(M)$$

where E_r is the embedding of the repository description and E_m is the embedding of ML phrases. The embeddings are then compared using cosine similarity, a metric used to measure how similar the documents are irrespective of their size.

$$\text{Cosine Similarity} = \frac{E_r \cdot E_m}{|E_r||E_m|}$$

This measure helps in identifying repositories whose descriptions are semantically close to common ML phrases, suggesting potential SoTA claims.

III. Contextual Tagging using AI: To classify repositories as claiming state-of-the-art status, we leveraged the gpt3.5 turbo API. This advanced AI tool assists in tagging by analyzing the context of repository descriptions and aligning them with our predefined criteria for SoTA claims.

$$\text{Tags} = \text{AI}_{\text{tag}}(E_r, \text{GPT-3.5 Turbo})$$

Where Tags are the labels assigned by the AI based on the analysis of the embeddings E_r under the guidance of GPT-3.5 Turbo's contextual understanding capabilities.

IV. Handling Class Imbalance: The datasets used for classifying SoTA and non-SoTA repositories exhibited a pronounced class imbalance, with SoTA-claiming repositories making up 30% of Dataset 2 and 20% of Dataset 1. To mitigate this imbalance and enhance model performance, we have now adopted the method of applying class weights directly in the model training process. This approach adjusts the importance assigned to each class during the training of the machine learning model, compensating for imbalances by assigning higher weights to the minority class and lower weights to the majority class [12].

The mathematical basis for class weights can be formulated as follows:

$$w_j = \frac{N}{k \times n_j}$$

where w_j is the weight for class j, N is the total number of samples in the dataset, k is the number of classes, and n_j is the number of samples in class j. By applying these weights, the loss function during training is adjusted so that errors in predicting the minority class have a larger impact compared to the majority class. This effectively increases the cost of misclassifying the minority class, allowing the model to pay more attention to it.

Model Interpretation

In the analysis of predictive models, understanding the influence of each feature on the predicted outcomes is crucial. Our approach to model interpretation encompasses both the visualization of input feature weights and the application of Shapley Additive exPlanations (SHAP) values to provide a more nuanced understanding of model behavior.

Visualizing Feature Weights: Feature weights are fundamental to interpreting the traditional linear models where each weight represents the strength and direction of the relationship between a feature and the outcome. By analyzing these weights, one can determine which features are most influential, and how changes in those features are expected to impact the predicted results. The positive or negative sign of each weight indicates whether the feature contributes positively or negatively to the outcome, respectively.

$$\text{Influence} = \beta_i \times \text{Feature}_i$$

Where β_i is the weight associated with the i-th feature. This method is straightforward and provides a direct interpretation but is limited to linear associations.

Shapley Additive exPlanations (SHAP) Values: To overcome the limitations of basic feature weight interpretation, we employed SHAP values, which offer a more comprehensive insight into the contribution of each feature across a complex model's predictions. SHAP values are derived from game theory, specifically from the concept of Shapley values, which distribute the "payout" (prediction effect) among the "players" (features) [13].

The key difference between SHAP and traditional feature importance is that SHAP considers the interaction effects between features. It provides a value for each feature for each prediction, indicating how much each feature contributed, positively or negatively, to that specific prediction. This is particularly useful in models where interactions or non-linearities obscure the effects of individual features.

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S)]$$

Where ϕ_i is the SHAP value for feature i, S is a subset of features excluding i, N is the total set of features, and v is the model's prediction function.

Suspicious User Analysis

To systematically analyze the dataset comprising approximately 500,000 users across 20 CSV files, we implemented a chunking technique to efficiently process 50,000 rows from each file sequentially. Subsequently, we performed random sampling to select 5,000 users from each file, which we then aggregated into a structured DataFrame for in-depth analysis. Utilizing Python libraries, specifically Pandas [14] for data manipulation and both Matplotlib [15] and Seaborn [16] for data visualization, we conducted comparative analyses of behavioral patterns between suspicious and non-suspicious accounts. Key metrics examined included follower-to-following ratios and commit frequencies. Moreover, we investigated temporal aspects of the accounts, comparing account creation times with the dates of last activity to gauge the duration of account activity. We also assessed the completeness of user profiles to identify any characteristic patterns indicative of suspicious activities. These methodologies enabled us to discern distinct behavioral signatures associated with suspicious accounts, thereby validating our analytical framework against established benchmarks in the domain.

Commit Analysis

In this study, we investigate the correlation between development consistency and popularity in GitHub repositories. We quantify development consistency through the standard deviation of the intervals between consecutive commits, with lower values indicating more regular update patterns and higher values suggesting irregular, sporadic activity. This metric helps in identifying patterns of "crunch-time development," characterized by bursts of intense activity, and "development hiatuses," marked by prolonged inactivity. Popularity is assessed by the number of stars a repository receives, which reflects user interest and engagement. Stars not only indicate a user's interest in following repository updates but also influence GitHub's content recommendation algorithms, enhancing visibility. The analysis involved preprocessing a user-centric dataset, where repositories were extracted from individual user profiles into a unified list annotated with star counts. We derived the standard deviation of commit intervals for each repository from commit timestamps, standardized to UTC. These intervals were sorted chronologically to calculate the time differences between consecutive commits. Due to computational constraints, the dataset was divided into twenty uniform subsets and one smaller subset for manageable processing. These subsets were stored as JSON files and designed to be recombined flexibly, allowing for scalable analysis depending on available computational resources. Correlation analysis was conducted by plotting repositories with their star counts on the x-axis against the standard deviation of commit intervals on the y-axis. Initial observations suggested a power-law distribution, prompting a replotting on logarithmic scales to better discern potential correlations. We computed Pearson, Spearman, and Kendall correlation coefficients to assess the strength and significance of the observed relationships.

V. Results

Q1: What are the characteristics of popular & successful Repositories?

This section delves into the analysis of GitHub repositories to discern the underlying characteristics that define their popularity and success. We assess this through predictive modeling, examining both popularity metrics (stars, forks, issue counts) and success classifications.

1) ***Popularity****:* For the popularity assessment, our primary objective is to predict the number of stars, forks, and open issues. These metrics serve as indicators of a repository's popularity. Our modeling approach includes XGBoost, Linear Regression, and Random Forest. These models were chosen for their varying abilities to handle linear and non-linear relationships within the data.

In our analysis, we established a baseline for comparison by using a simple model that always predicts the median value of the target feature from the training dataset. This baseline model does not involve linear regression but is rather a method of measuring central tendency. We evaluated the performance of this model using the Root Mean Square Error (RMSE), which quantifies the average magnitude of the prediction error. The R^2 score, which measures the proportion of variance in the target variable that is predictable from the features, was then calculated to assess how effectively this baseline model captures the variability in the observed outcomes.

The results of this comparison are captured in Table I - IV, which detail the models' performances using R^2 score and confidence intervals for Dataset 1 and Dataset 2, respectively. Figures and tables representing the overlapping input features between both datasets will be indicated by an asterisk symbol (*). They form the basis of our evaluations, allowing for a direct and consistent comparison of the models under standardized conditions. This approach underscores the robustness and generalizability of the models. Table V - VI complements this by showcasing the performance of all models utilizing all available input features in Dataset 2.

TABLE I
MODEL'S REGRESSION PERFORMANCE ON DATASET 1 FOR STARS AND FORKS, BY R^2 SCORE*

Model	Stars (%)	Forks (%)
Baseline Model	-3.75% ± 0.18%	-8.18% ± 0.40%
Linear Regression	19.42% ± 0.51%	17.87% ± 0.38%
XGBoost	**21.20% ± 0.36%**	**19.69% ± 0.50%**

TABLE II
MODEL'S REGRESSION PERFORMANCE ON DATASET 1 FOR ISSUES, BY R^2 SCORE*

Model	Issues (%)
Baseline Model	-15.71% ± 0.45%
Linear Regression	10.42% ± 0.31%
XGBoost	**13.03% ± 0.39%**

TABLE III
MODEL'S REGRESSION PERFORMANCE ON DATASET 2 FOR STARS AND FORKS, BY R^2 SCORE*

Model	Stars (%)	Forks (%)
Baseline Model	-10.94% ± 0.09%	-1.08% ± 0.02%
Linear Regression	14.52% ± 0.15%	18.36% ± 0.21%
XGBoost	**16.34% ± 0.10%**	**19.85% ± 0.12%**

TABLE IV
MODEL'S REGRESSION PERFORMANCE ON DATASET 2 FOR ISSUES, BY R^2 SCORE*

Model	Issues (%)
Baseline Model	-9.53% ± 0.06%
Linear Regression	24.18% ± 0.12%
XGBoost	**27.65% ± 0.13%**

TABLE V
MODEL'S REGRESSION PERFORMANCE ON DATASET 2 FOR STARS AND FORKS, BY R^2 SCORE

Model	Stars (%)	Forks (%)
Baseline Model	-10.95% ± 0.09%	-1.09% ± 0.03%
Linear Regression	44.76% ± 0.12%	45.76% ± 0.18%
XGBoost	**49.63% ± 0.15%**	**54.72% ± 0.14%**

TABLE VI
MODEL'S REGRESSION PERFORMANCE ON DATASET 2 FOR ISSUES, BY R^2 SCORE

Model	Issues (%)
Baseline Model	-9.51% ± 0.09%
Linear Regression	**50.70% ± 0.13%**
XGBoost	49.78% ± 0.23%

To verify the robustness of our results, we used ten different random seeds, ranging from 20 to 32. For each seed, the dataset was partitioned into training and testing sets using the `random_state` parameter set to the corresponding seed value. This ensured that each loop's train-test split was consistent with the seed, allowing for reliable training and evaluation of the model across varied data splits.

A. SHAP Value Analysis

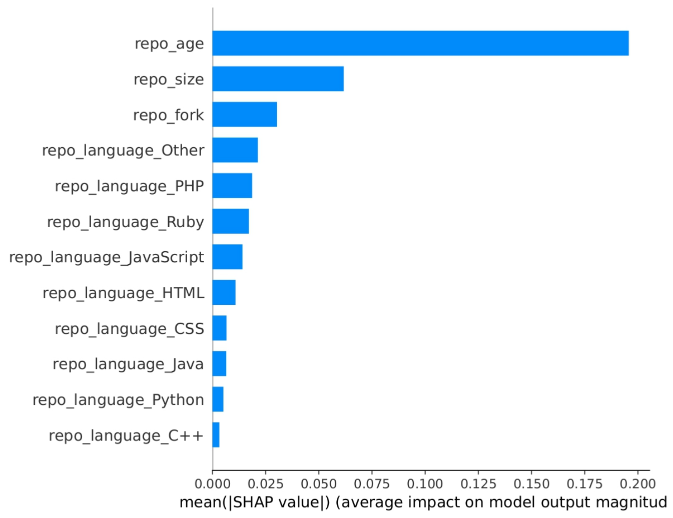


Fig. 1. XGBoost SHAP values on Dataset 1*

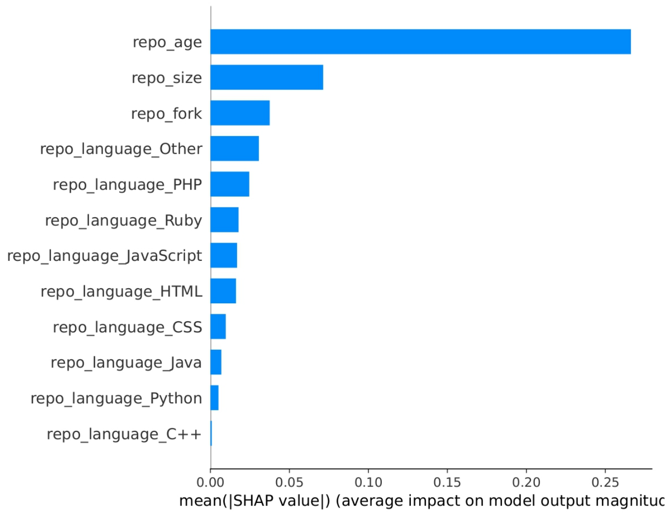


Fig. 2. Linear Regression SHAP values on Dataset 1*

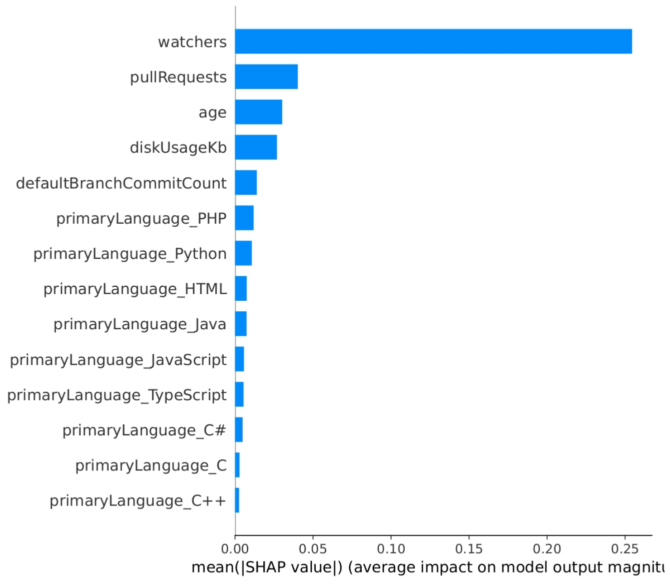


Fig. 5. XGBoost SHAP values for extended features

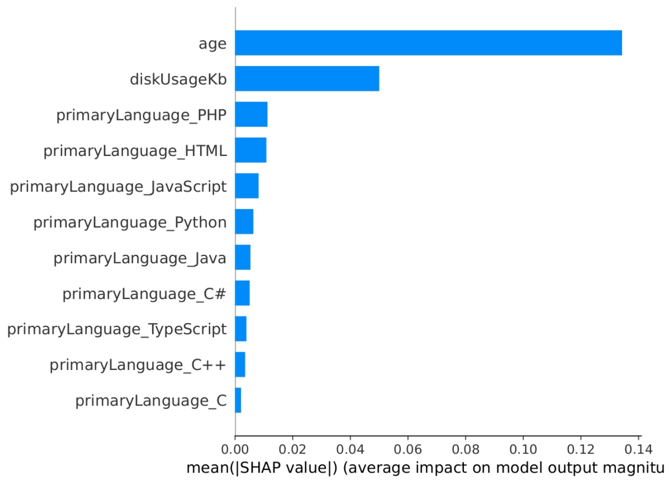


Fig. 3. XGBoost SHAP values on Dataset 2*

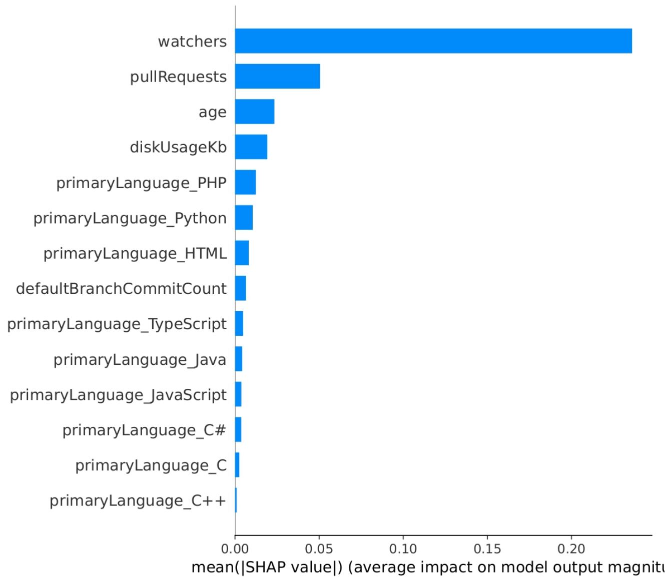


Fig. 6. Linear Regression SHAP values for extended features

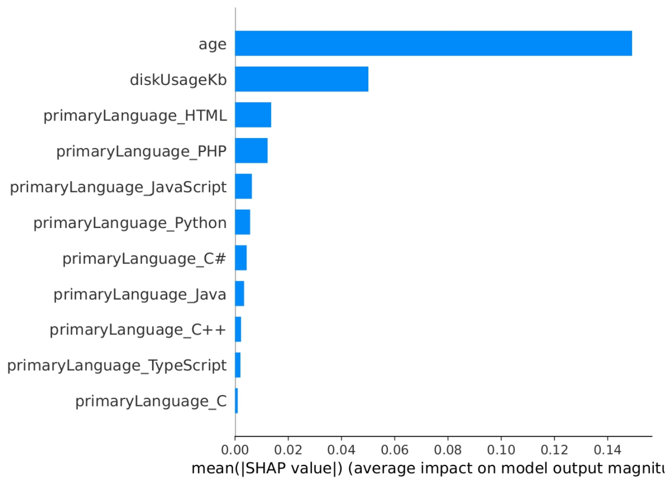


Fig. 4. Linear Regression SHAP values on Dataset 2*

Feature Influence: Through targeted analysis, we've distilled the factors influencing GitHub repository popularity. Figures 1 and 2 reveal `repo_age` and `repo_size` as key predictors, suggesting repositories accrue recognition and stars progressively, contradicting prior findings of a negligible age-popularity correlation. Notably, programming languages such as Python and JavaScript significantly affect repository appeal, a point consistent with past literature.

Further insights from Figures 3 and 4 extended feature range emphasize the relevance of these languages in predicting repository stars, reinforcing their potential to impact popularity. Additionally, the presence of `watchers`, pullRequests, and defaultBranchCommitCount correlates positively with stars, aligning with previous correlations reported in studies.

For a full examination of features, including forks and issues, please refer to our GitHub repository.

Further extending our analysis, Figures 5 and 6 showcase SHAP plots and feature weights for Dataset 2, covering a comprehensive range of predictive features. In our study, the influence of programming languages like Python and JavaScript is particularly noteworthy, reinforcing the notion that the choice of language can affect a repository's appeal and utility, thereby impacting its popularity metrics. The analysis of watchers, pullRequests, and defaultBranchCommitCount also aligns with findings from previous studies, which indicated a weak but significant positive correlation between these factors and the number of stars a repository garners.

*1) **Success**:* In the study of repository success classification, we employ several predictive models, namely XGBoost, Logistic Regression, and Random Forest. These models were selected due to their proven robustness and efficacy in handling diverse data distributions and interactions among features in classification tasks. To establish a meaningful baseline, we utilize the DummyClassifier from scikit-learn with a strategy of stratified sampling.

The benchmark for evaluating the success classification is the use of classification metrics derived from a stratified random sampling technique. This approach ensures that the proportion of classes in the dataset is consistently represented in both training and testing sets, thereby offering a realistic and relevant baseline for comparative analysis. This stratified sampling method aids in reflecting the actual class distribution within the dataset, crucial for assessing the performance of our predictive models against a baseline that mirrors real-world conditions.

TABLE VII
MODEL'S PERFORMANCE ON DATASET 1, BY CLASSIFICATION METRICS

Model	Accuracy (%)	Precision (%)
Baseline Model	59.13% ± 0.70%	49.81% ± 0.98%
Logistic Regression	59.96% ± 1.29%	56.90% ± 0.98%
Random Forest	71.08% ± 1.87%	59.43% ± 2.68%
XGBoost	63.79% ± 0.97%	58.57% ± 1.11%

TABLE VIII
MODEL'S PERFORMANCE ON DATASET 1, BY CLASSIFICATION METRICS (CONTINUED)

Model	Recall (%)	F1 Score
Baseline Model	49.81% ± 0.98%	49.79% ± 0.98%
Logistic Regression	58.35% ± 1.14%	56.21% ± 1.07%
Random Forest	53.45% ± 1.22%	51.25% ± 2.25%
XGBoost	59.81% ± 1.38%	**58.65% ± 1.16%**

TABLE IX
MODEL'S PERFORMANCE ON DATASET 2, BY CLASSIFICATION METRICS (ACCURACY AND PRECISION)

Model	Accuracy (%)	Precision (%)
Baseline Model	55.10% ± 0.65%	49.97% ± 0.84%
Logistic Regression	37.91% ± 1.29%	51.68% ± 1.21%
Random Forest	67.15% ± 0.77%	61.52% ± 0.63%
XGBoost	60.16% ± 0.82%	58.82% ± 1.27%

TABLE X
MODEL'S PERFORMANCE ON DATASET 2, BY CLASSIFICATION METRICS (RECALL AND F1 SCORE)

Model	Recall (%)	F1 Score
Baseline Model	49.96% ± 0.84%	49.96% ± 0.83%
Logistic Regression	50.71% ± 0.54%	34.21% ± 1.78%
Random Forest	55.91% ± 0.48%	54.39% ± 0.90%
XGBoost	59.76% ± 1.40%	**58.40% ± 1.17%**

As previously described, we employed ten different random seeds (20 to 32) to partition the dataset into training and testing sets consistently using the random_state parameter tied to each seed. This method ensured reliable model evaluations across various data splits.

Our comprehensive analysis using XGBoost, Logistic Regression and Random Forest models across two distinct datasets reveals key insights into the relationship between repository features and their perceived success. Notably, the application of SHAP value analysis to these models provides a deeper understanding of feature contributions to predictive accuracy.

B. SHAP Value Analysis

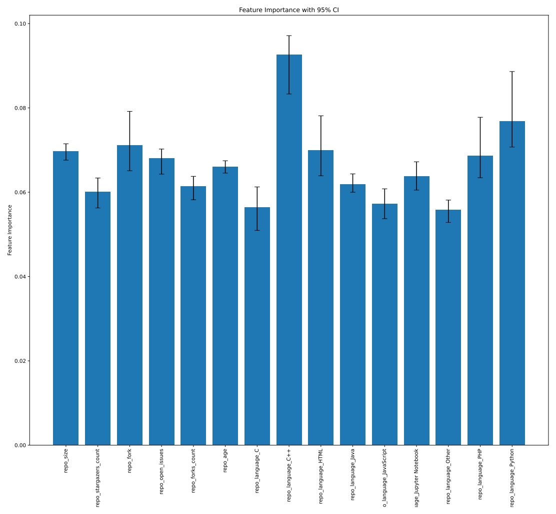


Fig. 7. XGBoost SHAP values: Dataset 1

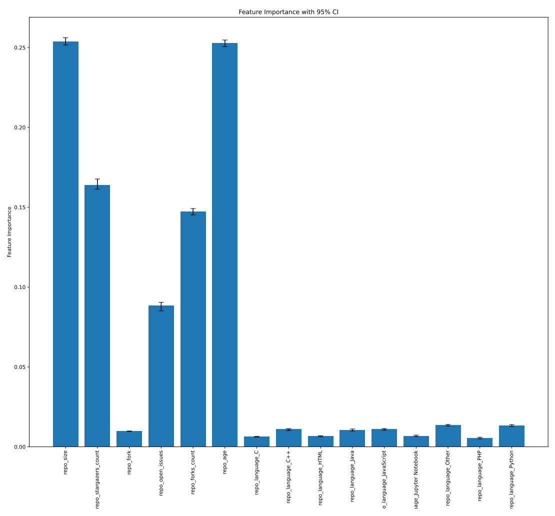


Fig. 8. Random Forest SHAP values: Dataset 1

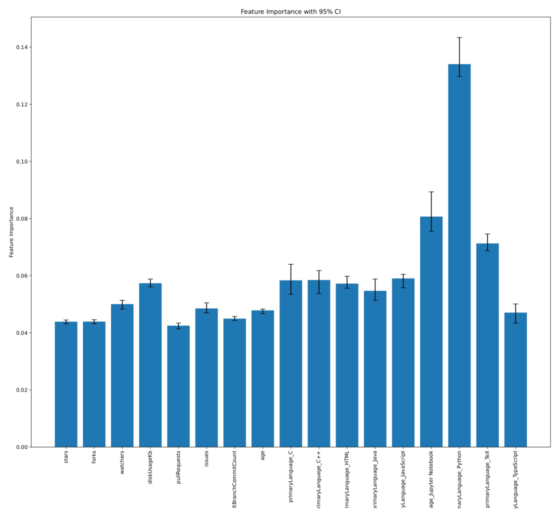


Fig. 10. XGBoost SHAP values: Dataset 2

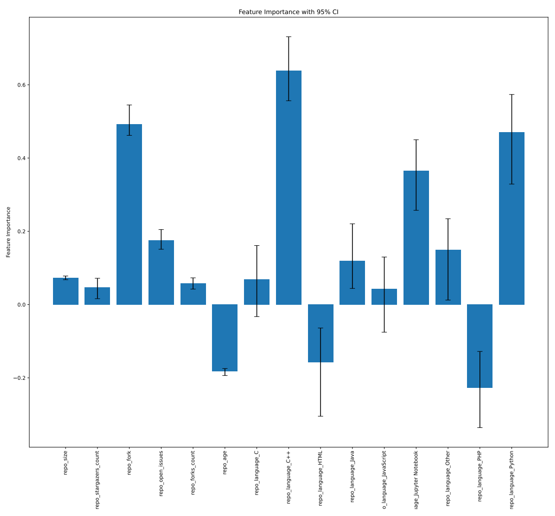


Fig. 9. Logistic Regression SHAP values: Dataset 1

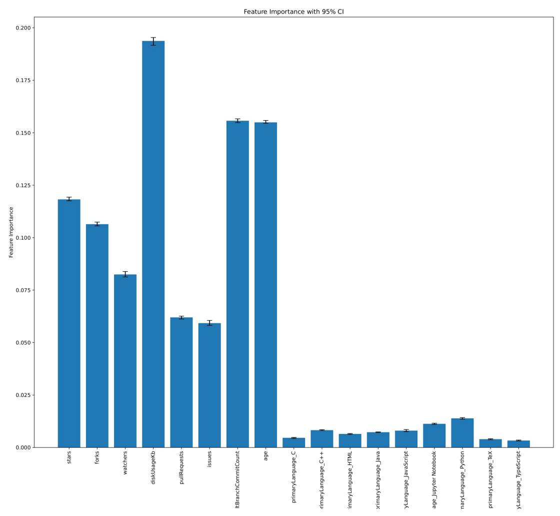


Fig. 11. Random Forest SHAP values: Dataset 2

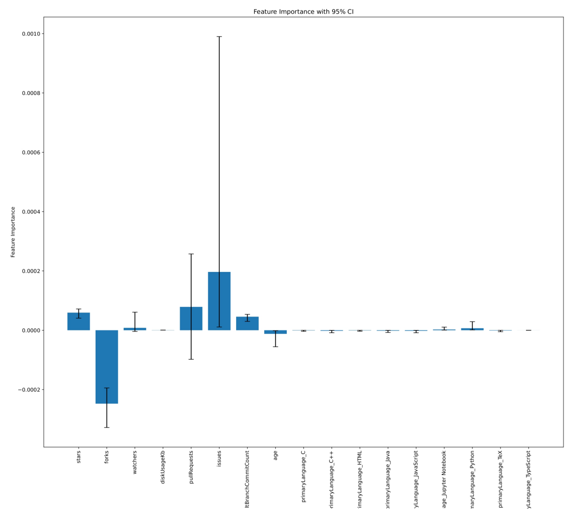


Fig. 12. Plot F: Logistic Regression SHAP values

These plots will showcase the predictive power of individual features, with a particular focus on `stars` as the primary indicator of popularity, aligning with previous research.

The SHAP value plots corresponding to Figure 1 elucidate the feature importance as derived from three different models applied to Dataset 1.

1) **XGBoost (Figure 7)**: The SHAP values indicate that `repo_size` and `repo_age` are the most influential features, with the size of a repository exhibiting the highest impact on model output. This suggests that larger and older repositories tend to be more successful. However, `repo_stargazers_count` which is a direct measure of popularity, shows a relatively moderate influence, supporting the notion that popularity metrics like stars may not be the sole determinants of a repository's success.
2) **Random Forest (Figure 8)**: Consistent with the XGBoost model, the Random Forest model also identifies `repo_size` and `repo_age` as critical features. It further emphasizes the role of `repo_open_issues` and `repo_forks_count`, which appear to have a substantial impact on the prediction, albeit less than the size and age.
3) **Logistic Regression (Figure 9)**: In contrast to the ensemble models, Logistic Regression shows a different pattern of feature importance, with `repo_language_Python` and `repo_language_JavaScript` among the top contributors. This underscores the relevance of the programming language used in the repository to its success, aligning with the previously identified importance of technology stacks.

The SHAP value plots for Dataset 2 reflect the feature importance across a more extensive set of predictive input features.

1) **XGBoost (Figure 10)**: This model reveals a pattern similar to that of Dataset 1, with `size` and `age` continuing to dominate as the leading indicators of success. Nonetheless, the presence of a broader feature set in Dataset 2 allows for the identification of more nuanced patterns of influence, particularly with various programming languages playing a notable role.
2) **Random Forest (Figure 11)**: The findings for the Random Forest model remain consistent with those from Dataset 1, again highlighting the significant impact of repository characteristics like size and age. However, the increased number of features in Dataset 2 reveals additional insights, particularly into the influence of developer activity, as indicated by the importance of `forks` and `issues`.
3) **Logistic Regression (Figure 12)**: With an expanded feature set in Dataset 2, the Logistic Regression model's interpretation becomes more complex. It demonstrates that while traditional metrics like `stars` are influential, the impact of other features, including `primaryLanguage_JavaScript` and `primaryLanguage_Python`, is pronounced.

Across both figures and datasets, it is evident that traditional popularity metrics such as `stars` are not the sole indicators of success. Instead, the size and age of a repository, along with the programming language, are consistently significant predictors across models, suggesting that these factors play a more critical role in the success of GitHub repositories.

Q2: How can we determine false actors? What makes a GitHub user suspicious?

According to previous research, false actors typically have a higher following count than follower count, which is also supported by this data.

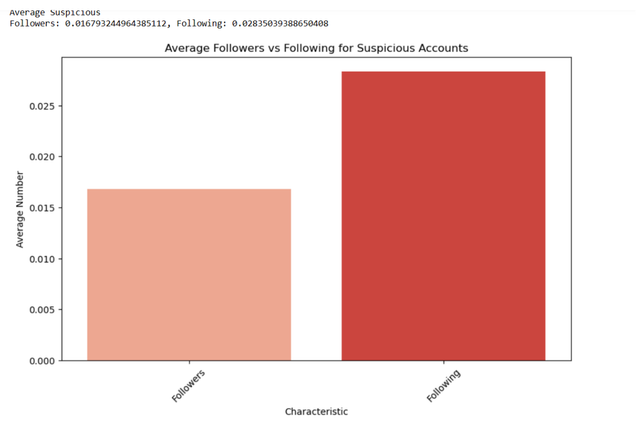

```
Average Suspicious
Followers: 0.016793244964385112, Following: 0.02835039388650408
```

Fig. 13. Bar Graph showing the Average Number of Followers vs. Following for Suspicious Accounts.

However, a notable observation is that the suspicious accounts would only have followers or a following, never both, and the number of either of them would be a maximum of 1, as evidenced by the averages both being far below 1.

Furthermore, the data shows that there is a distinct lack of commits that are made by false actors on Github.

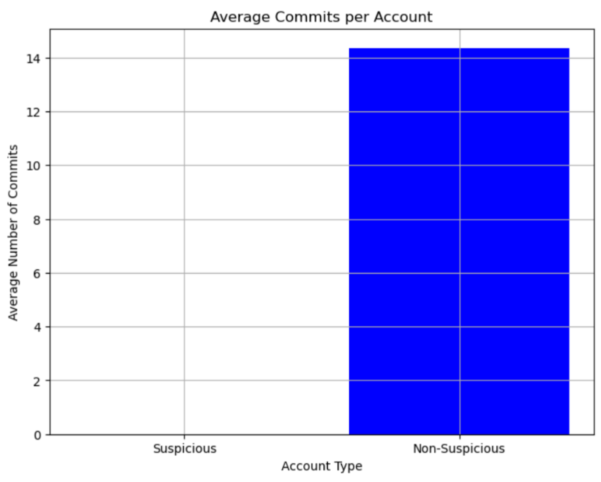


Fig. 14. Bar Graph showing the Average Number of Commits Made Per Account Type

This pattern of the suspicious account having significantly lower activity continues for various other account attributes, such as public repos and public gists as well. Seen in Fig 9 below.

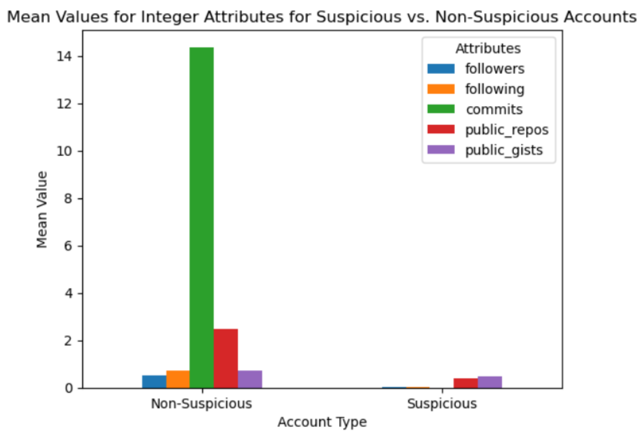


Fig. 15. Comparative Histograms of the Averages for each Integer Attribute of Suspicious and Non-Suspicious Accounts

Another notable pattern is the account creation time. This data shows that non-suspicious accounts are created more as the platform grows, while suspicious account creation fluctuates significantly.

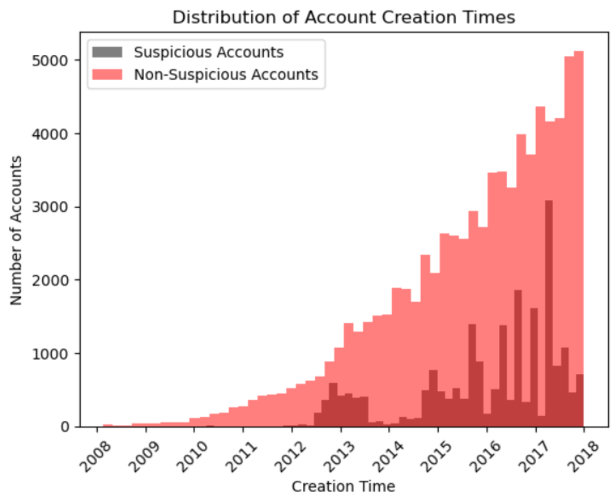
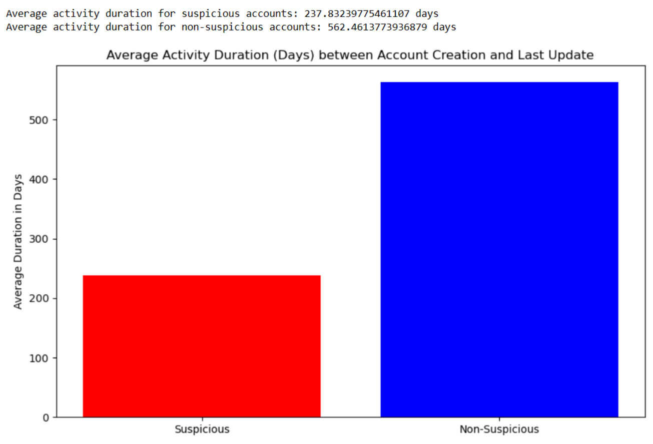


Fig. 16. Histogram showing Account Creation Over a 10 Year Period for Suspicious Accounts and Non-Suspicious Accounts

Along with this, the average length of activity on the accounts was very different for each type of account.



```
Average activity duration for suspicious accounts: 237.83239775461107 days
Average activity duration for non-suspicious accounts: 562.4613773936879 days
```

Fig. 17. Bar Chart showing the Average Length of Activity for Each Type of Account

The final pattern that was noticed in this data set is the likelihood of a suspicious account having a valid profile.

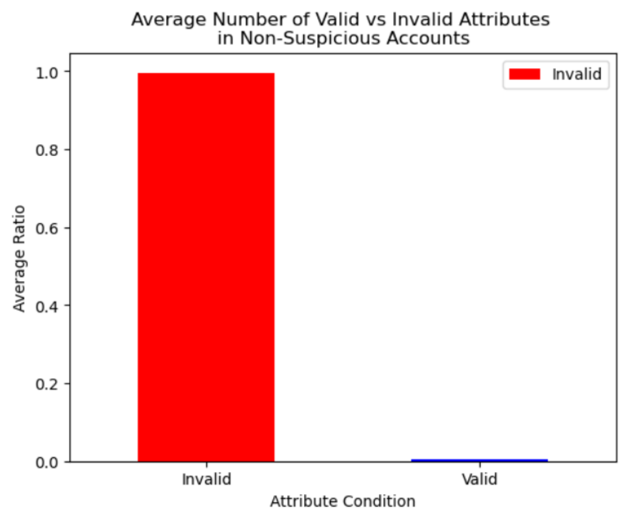


Fig. 18. Bar Chart Showing the Average Number of Valid and Invalid Non-suspicious Accounts

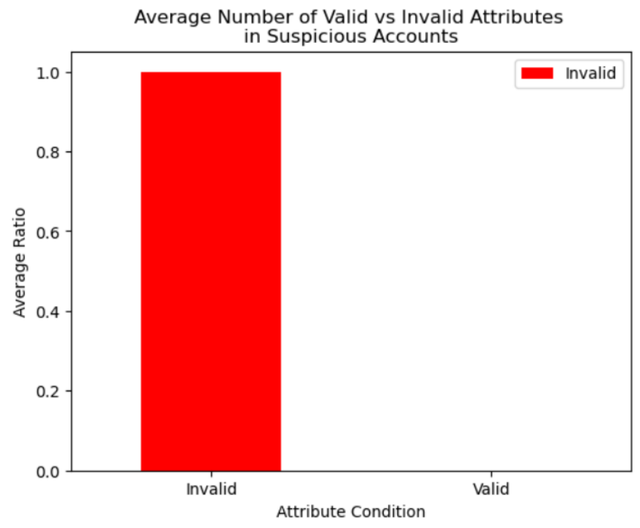
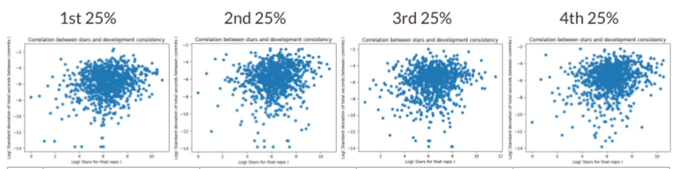


Fig. 19. Bar Chart Showing the Average Number of Valid and Invalid Suspicious Accounts

These patterns in behaviour all point in one direction when it comes to false actors on Github. While these are not definitive behavioural patterns, they are different enough from genuine accounts' behaviour that it is more than worth noting.

Q3: How can development consistency affect repository popularity?

We ran our notebook with split1.json to split5.json as the 1st 25%, split6.json to split10.json as the 2nd 25%, split11.json to split15.json as the 3rd 25%, and split16.json to split21.json as the 4th 25%. These combinations of subsets output the following plots and Pearson, Spearman, and Kendall values and their p-values in the same order.



R	0.208 P value = 2.869e-11	0.141 P value = 1.100e-5	0.145 P value = 5.683e-6	0.210 P value = 5.873e-11
P	0.166 P value = 1.303e-7	0.152 P value = 2.181e-6	0.130 P value = 4.415e-5	0.187 P value = 5.974e-9
T	0.110 P value = 1.702e-7	0.101 P value = 2.342e-6	0.087 P value = 4.688e-5	0.126 P value = 5.406e-9

Fig. 20. Correlation plots of all split combinations.

	Split 1-5	Split 6-10	Split 11-15	Split 16-21
Pearson	0.20874724272623507	0.141436327772743853	0.1452645572916012	0.21019875530419124
P value	2.8697042726794985e-11	1.100777759878337e-5	5.68379848554797e-6	5.873503664234763e-11
Spearman	0.16629598988705582	0.15223508136602185	0.13088681550439235	0.187219790816957
P value	1.3030453943034864e-7	2.1813377914782667e-6	4.415973515769236e-5	5.974947460121548e-9
Kendall	0.11066093639618675	0.10183221812626098	0.08738703239700284	0.1263627701402229
P value	1.7027811249943542e-7	2.342557298189738e-6	4.688139723439709e-5	5.406492583922721e-9

Fig. 21. Correlation coefficients of all split combinations.

From these correlation coefficients, though statistically significant from the low p-values, they suggest a low correlation between development consistency and repo popularity. Therefore, we fail to reject the null hypothesis and continue to believe that development consistency is not correlated to repo popularity.

VI. Discussion

A. *Revisiting the Concept of Success vs. Popularity*

The comparative analysis across both datasets suggests that stars, despite being a direct and quantifiable metric of popularity, do not invariably correlate with the predictive indicators of success identified by our models. In particular, the SHAP value analysis reveals that `repo_age` and `repo_size`, along with the utilization of certain programming languages—namely Python and JavaScript—exhibit a stronger predictive association.

This observation leads us to a critical distinction in our study: **success does not equate to popularity**. Our data suggests that while stars are a direct metric of popularity, they do not necessarily capture the broader, more complex aspects of a repository's impact and relevance within the developer community. For instance, a repository may have fewer stars but be highly influential due to its longevity, comprehensive documentation, or because it addresses a niche yet vital need within a specific programming community. This discrepancy is consistently observed across both datasets, despite Dataset 2 containing a more extensive set of predictive input features. It implies that our understanding of what makes a repository successful should expand beyond the conventional metrics of stars, suggesting a more holistic approach to evaluating repository significance.

B. Impact of External Events on Suspicious Account Creation

Further research was conducted to determine if there is a correlation between the peaks in the graph (Fig 4) and significant events in GitHub's history. For example, in 2013, a surge in suspicious account creation coincided with the release of the official Open Data Policy of the United States on GitHub by the White House [17]. Similarly, the acquisition of GitHub by Microsoft in 2018 is marked by a significant increase in suspicious accounts in the graph [18], in contrast to periods of heightened competition from platforms like Bitbucket and GitLab during 2014-2015, where growth slowed, reflected in lower spikes [19]. This suggests that external events can also impact the creation of suspicious accounts, underscoring the dynamic nature of this phenomenon.

Another notable pattern lies in the account profiles; while it is not typical for any account on GitHub to have a valid account, it is still more likely for non-suspicious accounts to have a valid account than a suspicious account which in this dataset had no valid accounts. Overall, the insights gained from the analysis of account behavior on GitHub provide valuable indicators for detecting fake accounts, which include notably lower activity levels, shorter average activity duration, and the absence of valid account profiles compared to genuine accounts. Additionally, patterns such as the influx of bot accounts during market fluctuations and spikes in account creation during significant events in GitHub's history offer further understanding of the dynamics behind suspicious account behavior. However, in the future, it would be useful to apply these methods to a full dataset, perhaps making use of more powerful machines, in order to get a better analysis. Furthermore, in the realm of detecting false actors, it would be useful to have some way of connecting accounts so that these connections may be analyzed as a network. Lastly, it may be useful to analyze suspicious account profiles for phishing links and other malicious activity that does not have to do with their behavior.

C. Correlation Between Repository Popularity and Development Consistency

We found a low correlation between repository popularity and development consistency. This observation ties into the paper that similarly found a low correlation between commit frequency and repository popularity. Due to hardware limitations, we were forced to split our main dataset into equally-sized, non-overlapping subsets that may have yielded different results had we been able to run the entire dataset at once. We encourage future work to utilize advanced hardware that can handle the entire dataset simultaneously and potentially propose a more mathematically sound definition for development consistency.

VII. Conclusion

Our study has provided a comprehensive analysis of the multifaceted realm of GitHub repository popularity and success. We have revealed the subtle yet distinct differences between what makes a repository popular and what constitutes its success. Our research indicates that popularity, commonly inferred from the number of stars, does not always translate into the practical success of a project, which we have measured against the yardstick of a repository's state-of-the-art contributions. Our investigation has shed light on the low to moderate correlation between consistent development activities and the garnering of stars. This revelation prompts a re-evaluation of commonly held perceptions within the open source community, suggesting that developers should focus on more than frequent updates to achieve broader recognition and impact. Moreover, our work in identifying and analyzing false actors within the GitHub ecosystem represents a significant stride towards ensuring the integrity of repository metrics. By discerning the characteristics of these actors, we have contributed to the body of knowledge that will aid in safeguarding the authenticity of repository evaluations. While we acknowledge the limitations encountered in method applicability and the intricate network of interactions, we assert that the insights gained are invaluable. Our findings serve as a stepping stone for future research to refine methodologies for analyzing GitHub data, and for open source communities to implement more robust mechanisms for assessing repository value and authenticity.

References

[1] H. Borges, A. Hora, and M. T. Valente, ”Understanding the Factors That Impact the Popularity of GitHub Repositories,” 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, USA, 2016, pp. 334-344.

[2] Borges, Hudson, Hora, Andre, and Valente, Marco Tulio, ”Predicting the Popularity of GitHub Repositories,” Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2016.

[3] Hudson Borges and Marco Tulio Valente, ”What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform,” Journal of Systems and Software, vol. 146, pp. 112-129, 2018.

[4] S. Brisson, E. Noei, and K. Lyons, ”We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks,” 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 2020, pp. 59-69.

[5] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein and Y. L. Traon, ”Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub,” 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), Pasadena, CA, USA, 2013, pp. 188-197.

[6] ”Author Risk: Bad Actors in GitHub,” Phylum Blog, 2021. [Online]. Available: https://blog.phylum.io/author-risk-bad-actors-in-github/

[7] M. Golzadeh, A. Decan, D. Legay, T. Mens, ”A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments,” Journal of Systems and Software, vol. 175, 2021, 110911, ISSN 0164-1212, DOI: https://doi.org/10.1016/j.jss.2021.110911. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100008X

[8] F. Johansson, A. Rozental, K. Edlund, P. Côté, T. Sundberg, C. Onell, A. Rudman, and E. Skillgate, ”Associations between procrastination and subsequent health outcomes among university students in Sweden,” *JAMA Network Open*, vol. 6, no. 1, e2249346, 2023. [Online]. Available: https://doi.org/10.1001/jamanetworkopen.2022.49346

[9] M. Grönlund and J. Jefford-Baker, ”Measuring correlation between commit frequency and popularity on GitHub,” 2017.

[10] Gong, Qingyuan; Zhang, Jiayun; Chen, Yang; Xiao, Yu; Fu, Xiaoming; Hui, Pan; Li, Xiang; Wang, Xin, 2018, ”A Representative User-centric Dataset of 10 Million GitHub Developers”, Harvard Dataverse, V1, DOI: 10.7910/DVN/T6ZRJT.

[11] Pelmers. ”GitHub Public Repository Metadata with 5 Stars.” Kaggle, 2021. Available: https://www.kaggle.com/datasets/pelmers/github-repository-metadata-with-5-stars

[12] B. Bakırarar and A. ELHAN, ”Class Weighting Technique to Deal with Imbalanced Class Problem in Machine Learning: Methodological Research,” *Turkiye Klinikleri Journal of Biostatistics*, vol. 15, pp. 19-29, Jan. 2023, doi: 10.5336/biostatic.2022-93961.

[13] S. M. Lundberg and S.-I. Lee, ”A unified approach to interpreting model predictions,” in *Proc. of the 31st Int. Conf. on Neural Information Processing Systems (NIPS'17)*, Long Beach, California, USA, 2017, pp. 4768–4777.

[14] W. McKinney, ”Data Structures for Statistical Computing in Python,” in *Proc. of the 9th Python in Science Conference*, 2010, pp. 445-451.

[15] J. D. Hunter, ”Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

[16] M. Waskom et al., ”Seaborn: statistical data visualization,” *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.

[17] T. Park and S. Vanroekel, “Introducing: Project Open Data,” National Archives and Records Administration, https://obamawhitehouse.archives.gov/blog/2013/05/16/introducing-project-open-data (accessed Apr. 28, 2024).

[18] “Microsoft to acquire GitHub for $7.5 billion,” Microsoft News Center, https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/ (accessed Apr. 28, 2024).

[19] H. Shah, “How github democratized coding and found a new home at Microsoft,” Nira Blog, https://nira.com/github-history/ (accessed Apr. 28, 2024).

VIII. Contributions

The research presented in this paper is a collaborative effort of the following team members, who have made substantial contributions to the study:

1) **Hashem Al Sailani**: Methodology, Abstract, Discussion, Results
2) **Fatemah Elsewaky**: Literature Review, Methodology, Discussion, Results
3) **Robin Sardja**: Literature Review, Discussion, Results, References
4) **Kathryn Bodden**: Literature Review, Methodology, Discussion, Results
5) **Nathan Allen**: Motivation, Data, Discussion, Results
6) **Professor Raiyan Abdul Baten**: guidance throughout the semester