Yuling He

Jacob Koehler

Data Bootcamp

08 May 2025

Predicting App Quality in the Google Play Store: A Classification Approach

In today's digital economy, mobile applications serve as essential platforms for entertainment, productivity, commerce, and social connection. With millions of apps available on marketplaces like the Google Play Store, users rely heavily on app ratings and reviews to inform download decisions. These ratings, in turn, affect app visibility, monetization potential, and long-term success. Understanding what drives high app ratings is therefore crucial for developers, product managers, and marketers navigating a saturated app market.

This project centers on a predictive classification task: identifying whether an app is highly rated—defined as having a rating of 4.5 or higher—based solely on structured app metadata. Unlike user reviews or behavioral data, this analysis uses static and accessible features such as category, install count, pricing, in-app purchases, and advertising support. The goal is to pinpoint which features best predict user satisfaction and to build a machine learning model that generalizes these insights across apps.

Such a predictive model can support better decision-making. For developers, it offers guidance on optimizing app design and platform compatibility. For marketers, it provides evidence-based indicators of features associated with user approval and download momentum.

The Random Forest model reveals that user engagement signals, especially the number of user ratings and various install count metrics, are the strongest predictors of high ratings. These features likely reflect the credibility and exposure of an app. In contrast, monetization features such as ad support or app category show only minor influence. While developers can't directly control engagement volume, they can influence app design, pricing, and update strategies—suggesting that both visibility and thoughtful configuration play complementary roles in app success.

DATA DESCRIPTION

This project draws on a cleaned version of the Google Play Store dataset, which includes structured metadata for over 525,000 mobile applications. Each record provides details such as app name, category, user rating, number of reviews, install counts, pricing structure, presence of in-app purchases or ads, and technical specifications like app size and minimum Android version. These features offer a robust foundation for exploring patterns in app quality and building predictive models. The primary task is to classify whether an app is highly rated, defined as having a user rating of 4.5 or above.

In the exploratory data analysis (EDA) phase, initial cleaning steps were performed to address data quality issues. Missing or null values in key fields such as Rating, Reviews, and Installs were removed. String-formatted numeric columns like Installs and Price were standardized and converted to integers or floats for analysis. Rare or inconsistent categorical values were consolidated, and implausible records—such as apps with extremely high ratings but very low install counts—were filtered out. This produced a consistent, structured dataset ready for modeling.

Building on this foundation, the modeling phase introduced further preprocessing tailored to the needs of classification algorithms. A new binary target variable, is_high_rating, was created, where apps with ratings ≥ 4.5 were labeled as 1 and all others as 0. To ensure that only predictive features remained, non-informative columns such as App Name, App Id, and Developer Id were excluded. Boolean columns were converted to integers (0 or 1), and categorical variables such as Category and Content Rating were one-hot encoded to support machine learning algorithms that require numerical input.

Numerical features, including Rating Count, Price, and Size, were standardized using StandardScaler to ensure they were on a comparable scale—especially important for models sensitive to feature magnitudes. To optimize memory efficiency, large numeric columns such as Installs and Rating Count were downcast to more compact data types. The dataset was then split into training and testing subsets using an 80/20 ratio, with stratified sampling to preserve the class distribution of the target variable.

Given the size of the dataset, a 10% random sample was taken from both training and test sets to reduce computational load during model training and hyperparameter tuning. This sample still provided approximately 52,500 rows—more than sufficient to support reliable model development while improving runtime efficiency.

Together, these steps ensured that the feature matrix X and target vector y were appropriately cleaned, encoded, and scaled for input into multiple classification models, enabling both accuracy and scalability in predictive performance.

EXPLORATORY DATA ANALYSIS

The EDA revealed strong skews in app installs and reviews, with a small proportion of apps attracting the majority of user engagement. Highly rated apps (≥4.5) tended to cluster in categories like Productivity and Education, and were more common among free apps with in-app purchases. Scatterplots showed a positive association between the number of reviews and average rating, suggesting that engagement volume enhances perceived quality. These patterns supported the modeling decision to emphasize features like rating count and installs, and provided early strategic insight: boosting visibility and user interaction may be key levers for improving app ratings.

MODELS AND METHODS

To predict whether an app is highly rated based on its metadata, we implemented a supervised binary classification framework. Several models of increasing complexity were tested to compare performance and generalization. All models used the same feature matrix X and target variable y, and preprocessing steps—such as encoding and scaling—were embedded into scikit-learn pipelines for consistency. Model evaluation relied on a held-out 20% test set, with key metrics including accuracy, precision, recall, and F1 score. Emphasis was placed on F1 and recall, as correctly identifying high-rated apps is more valuable than simply maximizing overall accuracy in an imbalanced context.

To establish a baseline for comparison, we first implemented a Logistic Regression model embedded within a preprocessing pipeline. This model applies a linear decision boundary to predict whether an app is highly rated based on its metadata. The pipeline ensured that categorical variables were one-hot encoded and numeric features were standardized, preserving consistency with subsequent models.

(1) Logistic Regression

Logistic Regression achieved an overall accuracy of 81.6%, which might initially seem adequate. However, a closer look at the classification metrics reveals a major limitation: the model's recall for high-rated apps was 0.00, resulting in an F1 score near zero. Although precision was technically 1.00, this is misleading—nearly all high-rated apps were missed entirely, while low-rated apps were classified correctly.

The confusion matrix underscores this imbalance, showing that the model classified almost all instances as the majority class (low-rated). This result indicates that Logistic Regression, despite its interpretability and speed, is insufficient for this task. The linear nature of the model fails to capture the nonlinear patterns and feature interactions that likely drive user ratings. While it offers a valuable benchmark, more flexible models are clearly needed to identify the nuanced relationships within app success data.

(2) K-Nearest Neighbors (Tuned)

The second model implemented was K-Nearest Neighbors (KNN), a non-parametric algorithm that classifies data points based on the majority label of their nearest neighbors in feature space. Because KNN relies heavily on distance metrics, all numeric features were standardized using StandardScaler within the preprocessing pipeline. Categorical variables were one-hot encoded, as with the logistic regression model.

To optimize performance, a grid search was used to tune the n_neighbors hyperparameter. The best-performing model used k = 15, balancing local decision boundaries with generalizability. On the test set, the tuned KNN model achieved higher recall and F1 score than the baseline logistic regression. While precision decreased due to a slight increase in false

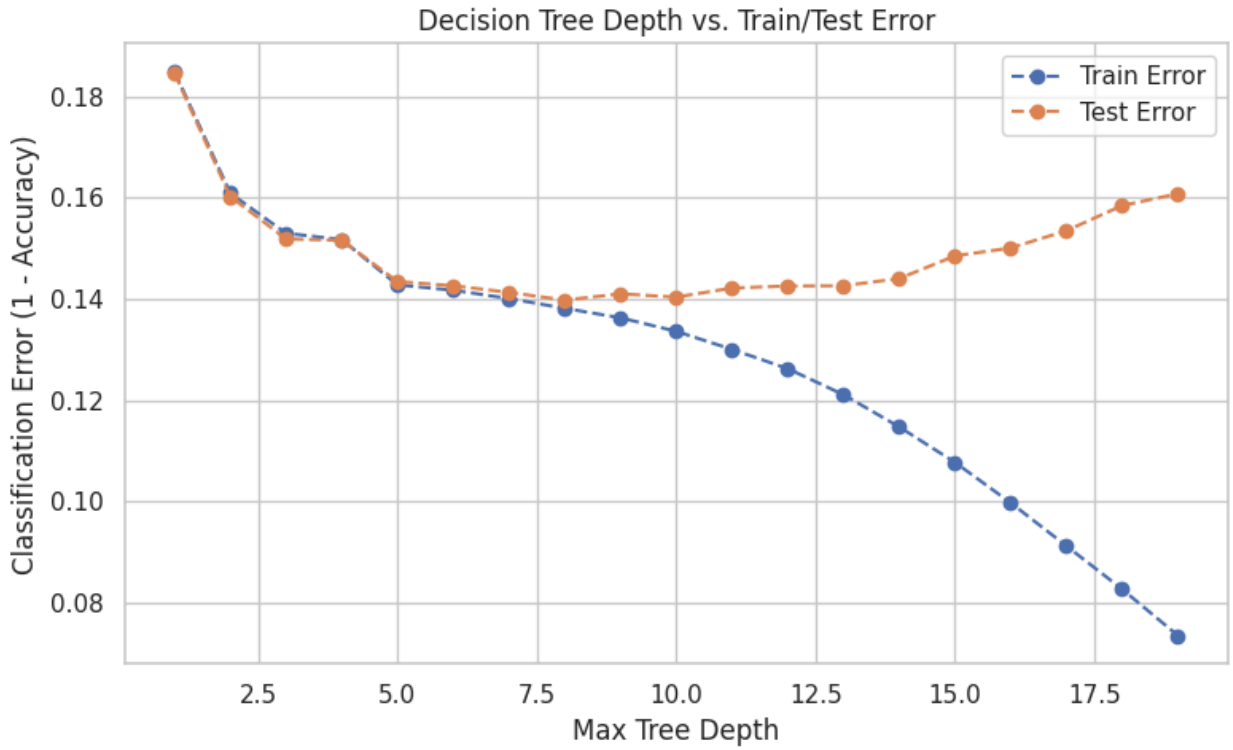
positives, the model demonstrated a more balanced classification ability, correctly identifying some high-rated apps.

These improvements suggest that KNN can better capture localized, nonlinear patterns within the app metadata, especially in high-dimensional space. However, its performance was still constrained by sensitivity to feature scaling and computational inefficiency in large datasets. Although not the top performer overall, KNN serves as a meaningful middle ground—outperforming the linear baseline while still lagging behind more sophisticated tree-based methods.

(3) Decision Tree

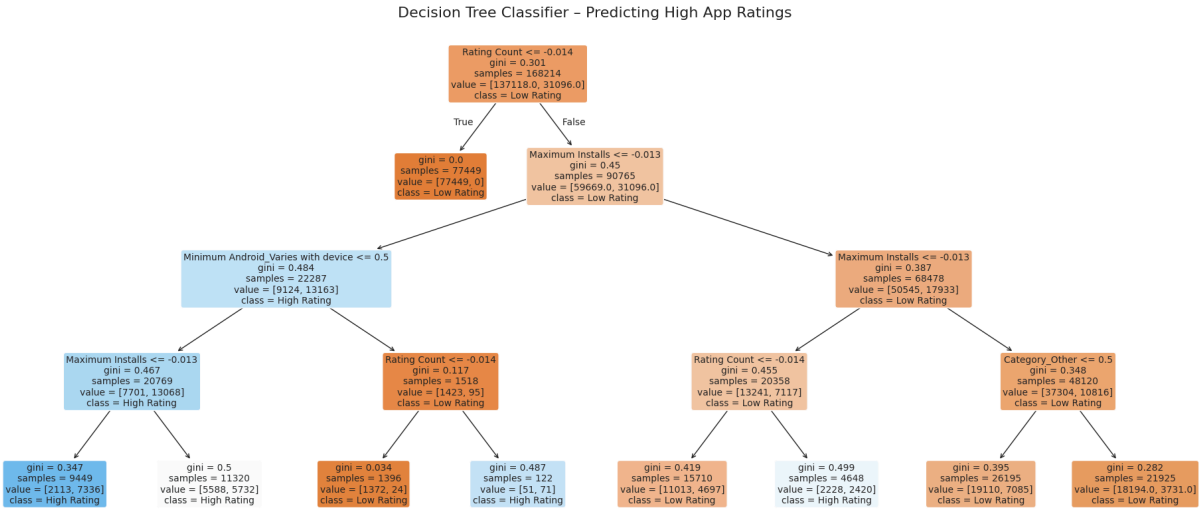
The third model implemented was a Decision Tree Classifier, which offers strong interpretability and handles nonlinear feature interactions effectively. Unlike KNN, which relies on distance metrics, decision trees recursively split the dataset into subgroups based on learned thresholds that maximize information gain. This allows them to capture complex relationships between features and the target variable.

To balance bias and variance, we conducted hyperparameter tuning on the tree's max_depth. The graph below shows the classification error on the training and test sets across different tree depths.



As the graph illustrates, training error continuously decreases with depth, while test error reaches its minimum around depth 7–9 and then begins to rise. This divergence indicates overfitting at higher depths, where the model captures noise in the training data. Based on this, we selected a depth of 9, which achieves strong generalization while maintaining manageable complexity.

In addition to performance tuning, decision trees provide visual transparency into how predictions are made. The diagram below shows a visualization of the trained decision tree, revealing which features drive the splits.



The top-level splits confirm that Rating Count and Maximum Installs are the most influential features in predicting whether an app is highly rated. These features dominate early decision paths, aligning with insights from the Random Forest model and EDA. Deeper nodes incorporate variables such as Minimum Android Version and App Category, which refine predictions in more nuanced subgroups.

Overall, the Decision Tree model offered improved recall and F1 score compared to both KNN and Logistic Regression, showing better capacity to identify high-rated apps. Its clear structure also makes it useful for interpretability-focused tasks, though some performance trade-offs remain compared to ensemble methods like Random Forest.

(4) Random Forest

To improve predictive performance and reduce overfitting, we implemented a Random Forest Classifier, an ensemble learning method that aggregates predictions from multiple decision trees trained on random feature subsets and bootstrapped samples. This approach

introduces diversity into the model and helps control variance, making it more robust than a single decision tree.

The Random Forest model was embedded in a scikit-learn pipeline to include all preprocessing steps—standardization for numeric features and one-hot encoding for categorical variables. Hyperparameters such as the number of trees (n_estimators) and tree depth were tuned using cross-validation. This tuned pipeline achieved the highest overall accuracy (86.1%) among all models, with strong balance across precision, recall, and F1 score.

One of the key advantages of Random Forest is its ability to output feature importances, offering insights into which app characteristics drive predictions. As discussed later in the Results section, the model identified Rating Count as the most influential predictor—accounting for over 50% of the model's decision power—followed by various Install metrics. These engagement signals align strongly with EDA observations and highlight the significance of user interaction volume in driving perceived app quality.

While slightly less interpretable than a single decision tree, the Random Forest achieved more stable and generalizable results, making it an excellent option for production deployment or automated quality prediction systems. It balances performance and interpretability, offering developers both actionable insights and reliable classification.

RESULTS AND INTERPRETATION

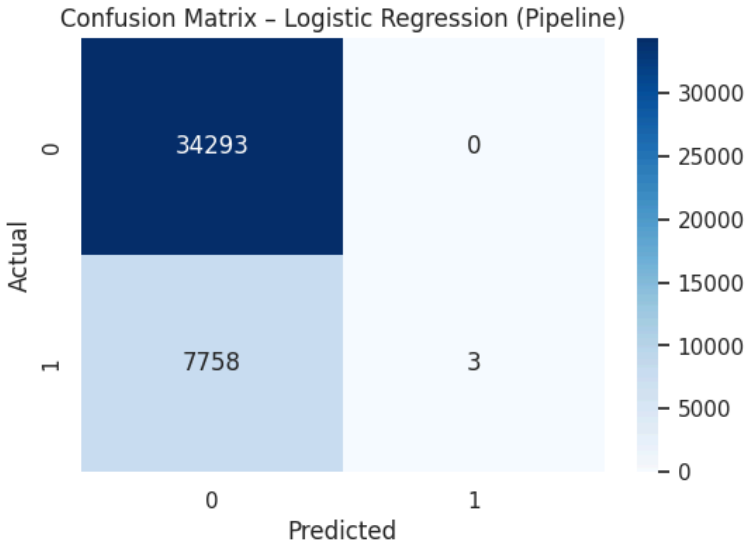
To evaluate each model's ability to predict high-rated apps, we compared performance using accuracy, precision, recall, and F1 score on a held-out test set. Given the class imbalance in the dataset—where high-rated apps form a minority—recall and F1 score were emphasized over

raw accuracy. Below, we interpret the predictive results for each model, highlighting strengths, weaknesses, and implications for identifying successful apps.

(1) Logistic Regression

The Logistic Regression model, implemented as a baseline, achieved an accuracy of 81.6% but failed to identify high-rated apps meaningfully. It produced a precision of 1.00, a recall of 0.00, and an F1 score of 0.00 for the positive class. These metrics reveal that while the model avoided false positives, it entirely missed the minority class.

The confusion matrix confirms this failure: of the 7,761 actual high-rated apps, only 3 were correctly identified (true positives), while 7,758 were misclassified as low-rated (false negatives). Conversely, all 34,293 low-rated apps were correctly predicted. This pattern reveals the model's extreme conservatism—defaulting to the dominant class in an imbalanced setting.



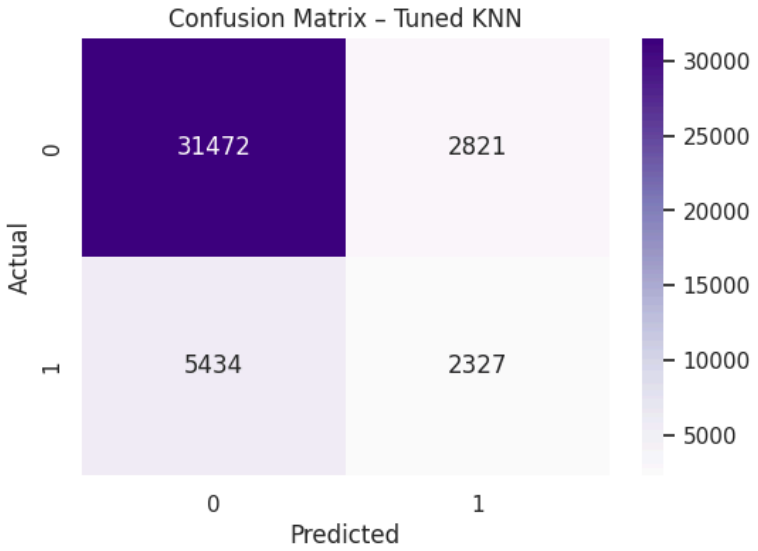
Although the overall accuracy appears high, this is misleading given the class imbalance. The model's complete inability to detect high-rated apps demonstrates the limitations of linear

classifiers in capturing complex, nonlinear relationships. Thus, Logistic Regression is valuable only as a performance benchmark and not as a viable predictive tool in this context.

(2) K-Nearest Neighbors

The tuned K-Nearest Neighbors model showed marked improvement over the baseline in identifying high-rated apps. It achieved an accuracy of 80.0%, with a precision of 0.45, recall of 0.30, and an F1 score of 0.36 for the positive class. These results indicate a better trade-off between precision and recall compared to Logistic Regression.

The confusion matrix shows that KNN correctly identified 2,327 high-rated apps, while misclassifying 5,434 as low-rated. On the negative side, it also produced 2,821 false positives, incorrectly labeling some low-rated apps as highly rated. This reflects the model's more balanced—but still imperfect—ability to distinguish between classes in an imbalanced dataset.



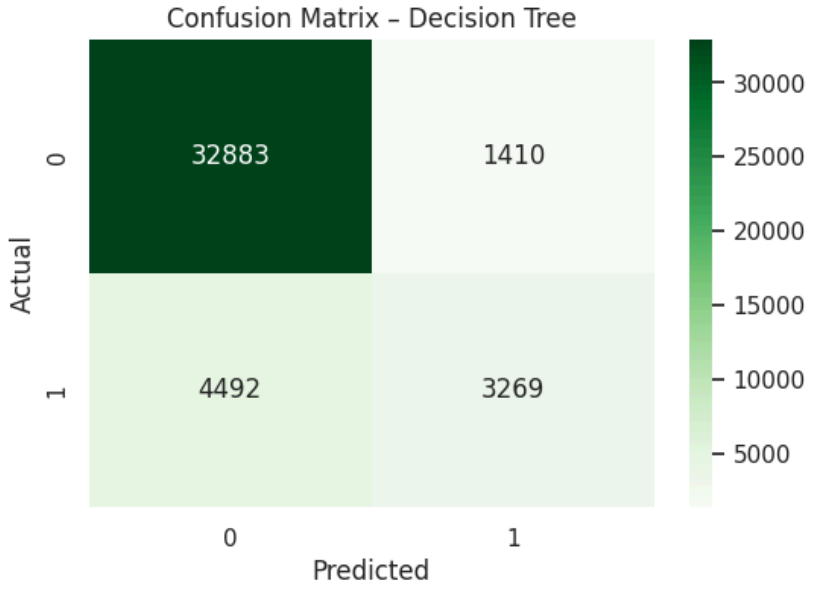
KNN's ability to detect a portion of high-rated apps validates its use as a non-linear learner, but the large number of both false positives and false negatives reveals limitations. Its performance depends heavily on feature scaling and neighborhood structure, making it sensitive

in high-dimensional space. Overall, KNN improved over the baseline but was ultimately outperformed by tree-based models in both precision and generalization.

(3) Decision Trees

The Decision Tree Classifier, tuned to a maximum depth of 9, significantly improved the model's ability to detect high-rated apps. It achieved an accuracy of 86.0%, a precision of 0.70, a recall of 0.42, and an F1 score of 0.53 for the high-rated class. These metrics demonstrate strong gains in recall and F1 over both KNN and Logistic Regression.

The confusion matrix highlights this performance shift: the model correctly identified 3,269 high-rated apps, while missing 4,492 (false negatives). Meanwhile, it misclassified 1,410 low-rated apps as high-rated—a manageable increase in false positives that reflects the model's improved inclusiveness. With over 32,000 true negatives, the model maintained solid overall accuracy while becoming more responsive to the minority class.



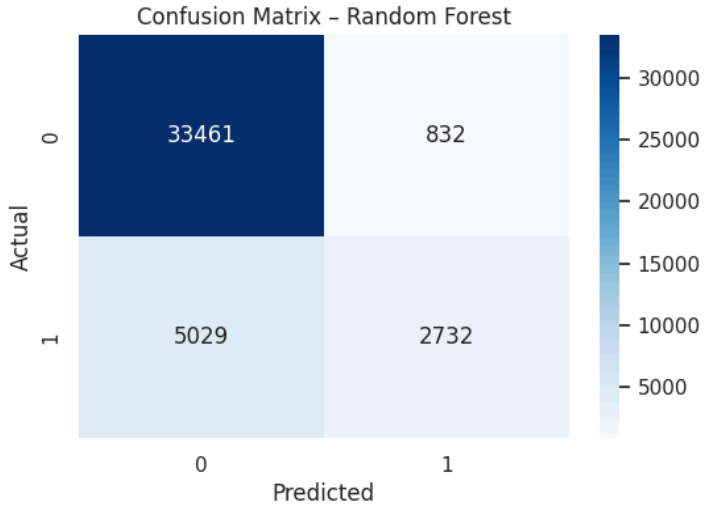
This balance makes the Decision Tree particularly effective when recall is prioritized, such as in developer tools that aim to flag promising apps for review. Moreover, the model's

transparent structure—based on intuitive splits like Rating Count and Max Installs—offers valuable interpretability for stakeholders. While not the top performer overall, it was the first model to offer meaningful, actionable detection of high-quality apps.

(4) Random Forest

The Random Forest Classifier, implemented with a tuned preprocessing pipeline, achieved the best overall performance. It recorded an accuracy of 86.0%, a precision of 0.77, a recall of 0.35, and an F1 score of 0.48 for high-rated apps. These results indicate a balanced model that improves recall without heavily sacrificing precision.

The confusion matrix confirms this: the model correctly identified 2,732 high-rated apps (true positives) and missed 5,029 (false negatives). It also achieved 33,461 true negatives while misclassifying just 832 low-rated apps as high-rated. Compared to the Decision Tree, Random Forest reduced false positives and maintained similar gains in recall—highlighting its robustness and ability to generalize across the dataset.



Beyond predictive power, Random Forest also produced meaningful feature importances. Rating Count emerged as the dominant variable, followed by install-related metrics. These

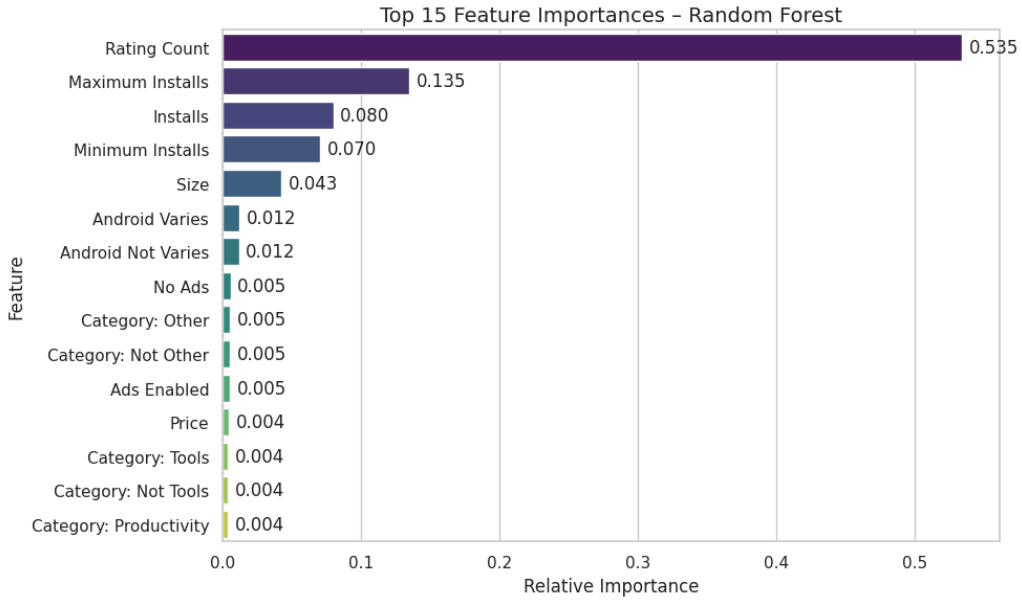
findings align with EDA and tree-based splits, confirming that user engagement volume is the most consistent predictor of app quality. The ensemble method's stability, interpretability, and overall performance make it the most suitable choice for real-world application in developer tools or market analysis dashboards.

	Model	Accuracy	Precision	Recall	F1 Score
1	Logistic	0.816	1.000	0.000	0.001
2	KNN	0.804	0.452	0.300	0.361
3	Decision Tree	0.860	0.699	0.421	0.526
4	RF	0.861	0.767	0.352	0.482

The performance comparison across four classification models—Logistic Regression, KNN, Decision Tree, and Random Forest—reveals a clear pattern: while simpler models offer baseline accuracy, only tree-based methods achieve meaningful recall and F1 scores. As shown in the table, Logistic Regression reached a high precision but failed to identify any high-rated apps (recall = 0.000), making it unsuitable for imbalanced classification. KNN improved recall to 0.300, while Decision Tree and Random Forest delivered the best overall balance. The Random Forest model achieved the highest accuracy (0.861) and strong precision (0.767), with Decision Tree slightly outperforming it in recall and F1. These results underscore the value of nonlinear, ensemble methods for modeling complex user engagement patterns in app success.

Beyond predictive accuracy, one of the key advantages of the Random Forest model is its ability to quantify feature importance, revealing which input variables most strongly influence the prediction of high app ratings. This analysis provides strategic insights into what matters

most for app success on the Google Play Store.



The model identified Rating Count as the overwhelmingly dominant feature, contributing over 50% of the total importance. This suggests that apps with a large number of user ratings are significantly more likely to be perceived as high quality—likely due to increased credibility, stability, and exposure in app rankings.

Other highly influential features included various Install metrics (e.g., Maximum Installs, Minimum Installs), reinforcing the idea that popularity and reach are key signals of app trustworthiness. Moderate contributions came from features like App Size and Minimum Android Version, which may reflect technical complexity or app recency.

Less influential features—such as whether the app is Ad Supported, contains In-App Purchases, or belongs to a particular Category—appeared much lower in the ranking. While these attributes do carry some signal, they are secondary to engagement-driven metrics in shaping user ratings.

These results suggest that developers aiming to boost perceived app quality should focus on strategies that grow user base and encourage reviews, such as streamlined onboarding, push notifications, and referral incentives. Meanwhile, thoughtful pricing, monetization, and category positioning remain important—but supplementary—levers in driving app success.

The model results underscore that user engagement metrics—particularly Rating Count and Install Volume—are the strongest predictors of high app ratings. This suggests developers should prioritize user acquisition strategies, visibility on the Play Store, and mechanisms that encourage user feedback. While secondary factors like pricing models, ad presence, and app category had less influence, they remain actionable levers. Developers can shape perceived app quality through thoughtful design choices, update frequency, and distribution strategies that support both reach and reputation.

CONCLUSION AND NEXT STEPS

This project aimed to classify whether a Google Play Store app is highly rated using structured metadata. After testing four classification models, the results demonstrated that nonlinear, tree-based methods significantly outperformed linear baselines. Random Forest delivered the best overall performance in accuracy and F1 score, while the Decision Tree offered the highest recall, making it especially effective for detecting high-rated apps. Simpler models like Logistic Regression proved too conservative, failing to capture the complexity of the data, whereas KNN offered modest improvements but remained limited by sensitivity to high-dimensional scaling.

The use of a 10% stratified sample—drawn to reduce computational costs—enabled efficient model development and hyperparameter tuning. However, applying the models to the

full dataset could further improve performance, especially for minority class detection. Future iterations may also benefit from exploring resampling techniques or cost-sensitive learning to address class imbalance more directly.

Strategically, the findings confirm that user engagement features, such as Rating Count and Install Volume, are the most influential predictors of perceived app quality. While pricing, ad presence, and app category played lesser roles, they still offer developers actionable levers for influencing user experience and ratings.

Nonetheless, the analysis is limited to structured metadata, excluding user-generated content like textual reviews or update histories. There are also trade-offs between interpretability and complexity, and some features show limited individual predictive power. Future work could enhance the model by incorporating natural language features, testing more advanced algorithms like XGBoost, or validating results across other platforms such as the Apple App Store.