A Comparative Machine Learning Approach to Forecasting "Hit" Video Games

Team 4B:

Srinidhi Chevvuri, Preme Chinpattanakul, Edward Dai, Alex Huang, & Megnha Prakash

Paul Merage School of Business, University of California, Irvine

BANA 273 Machine Learning Analytics

Professor Mingdi Xin

December 5, 2025

Executive Summary

This project investigates whether video games can be correctly classified as “hit” or "non-hit" based solely on early attributes such as platform, genre, publisher, and release year. We deployed multiple machine learning models to explore any patterns that may arise in a dataset of 1,907 video game titles. After preparing the data and addressing class imbalance, we trained machine learning models such as Logistic Regression, Naive Bayes, Decision Tree, and Random Forest. Each model was evaluated using recall, F1-score, and AUC. Additional methods, such as class weighting, SMOTE, regularization, and hyperparameter tuning, were added to improve performance. Post-analysis, we uncovered that the tuned Random Forest classifier performed the best across all models. Our study confirms that machine learning is able to predict hit games using structured pre-release metadata. These insights can support early-stage decision-making in game development, such as forecasting performance, prioritizing projects, and planning marketing strategies.

Table of Contents

I. Introduction

a. Business Implications

The video game industry is huge, where studios face high development costs and uncertainty around consumer and critic response. Identifying the factors that make a game successful before its release can provide developers with a strong competitive advantage. By predicting whether a game is likely to become a “hit,” developers and publishers can make more informed decisions about game design, platform selection, marketing investment, release timing, and launch planning.

Machine learning offers a way to capture patterns using data such as genre, platform, publisher reputation, release year, and early sales indicators. Since this can be done long before detailed review or player feedback becomes available, developers and studios can benefit from these insights. In this context, the business implication of our study is clear: a data-driven model that predicts hit status enables studios to optimize strategy, prioritize high-potential projects, and better understand the attributes associated with strong critical reception.

II. Data

a. Acquisition:

The dataset *“Discovering Hidden Trends in Global Video Games”* used in this project was obtained from Kaggle, which provides sales data and review information for commercial video games. The original dataset contained 1,907 records and included variables such as platform, genre, publisher, release year, regional/global sales, and a review score. These review scores helped us differentiate between hit and non-hit games.

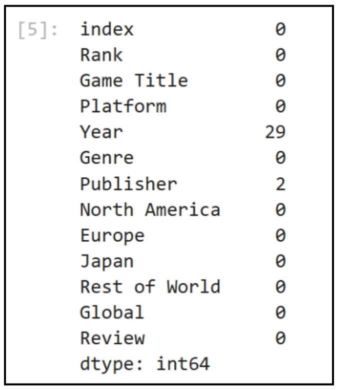
The dataset was downloaded as a CSV file and imported into Jupyter Notebook. Using Python, rows containing missing review scores or other null values were removed, resulting in a final dataset of 1,878 complete observations. No additional external datasets were merged. The Kaggle dataset’s structured format and the inclusion of Metacritic ratings made it a strong foundation for our machine learning analysis.

b. Defining “Hit” Games

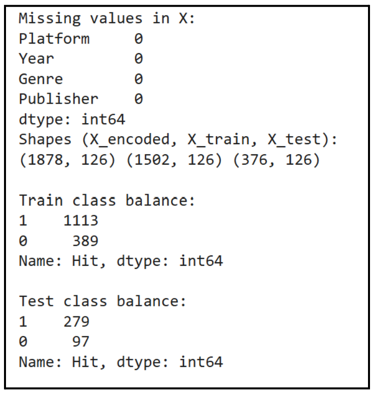
In this project, we defined a “hit” video game using industry standards. Our dataset included a Review column with critic scores for each game. To convert this numeric score into a target for prediction, we used Metacritic’s scoring system as a rubric. Metacritic classifies games scoring 75 or higher as “hit” titles, a threshold commonly associated with strong critical reception and higher commercial potential. (Metacritic). Using this standard, we labeled games with a Review score of 75 or higher as "hit" (1) and all others as "non-hit" (0). This binary label became our target variable (Y) for all machine learning models. By establishing our definition of “hit” in a widely used industry benchmark, we ensured that our target variable was both interpretable and credible throughout the analysis.

c. Preparation

Cleaning was a vital part of preparing our data; therefore, identifying the null values and removing outliers were a priority. We identified the null values in our dataset (29 missing values from the “year” feature and 2 missing values from the “Publisher” feature), removed the null values from each feature column, and then checked again to ensure there were no null values in our dataset. The pandas library was used in order to run these functions.



```
[5]:  index            0
      Rank             0
      Game Title       0
      Platform         0
      Year            29
      Genre            0
      Publisher        2
      North America    0
      Europe           0
      Japan            0
      Rest of World    0
      Global           0
      Review           0
      dtype: int64
```



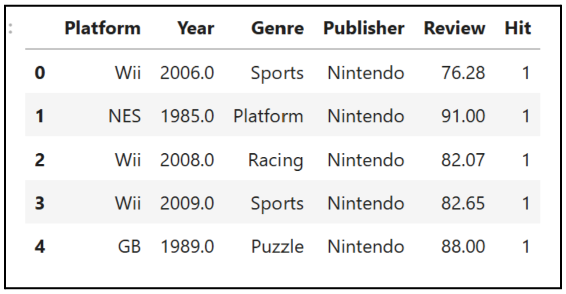
```
Missing values in X:
Platform     0
Year         0
Genre        0
Publisher    0
dtype: int64
Shapes (X_encoded, X_train, X_test):
(1878, 126) (1502, 126) (376, 126)

Train class balance:
1    1113
0     389
Name: Hit, dtype: int64

Test class balance:
1    279
0     97
Name: Hit, dtype: int64
```

The binary target variable "is_hit" was also created using the "Review" variable. Utilizing Metacritic, a widely used industry benchmark that aggregates professional critic reviews, it was determined that a review that is equal to 75 or higher would be considered a "hit" while a review below 75 would be considered a non-hit.

For the model, it was important to remove features that leak information to our model, leading to unreliable results. In the context of the problem statement, regional sales, rank, and review provide information about a video game post-launch, which will cause the model to cheat. Utilizing only pre-launch features (Platform, Year, Genre, and Publisher) to predict hit status was a crucial decision in generating a fair model. Therefore, the columns "Index", "Rank", "Game Title", "North America", "Europe", "Japan", "Rest of World", "Global", and "Review" were dropped from the dataset.



	Platform	Year	Genre	Publisher	Review	Hit
0	Wii	2006.0	Sports	Nintendo	76.28	1
1	NES	1985.0	Platform	Nintendo	91.00	1
2	Wii	2008.0	Racing	Nintendo	82.07	1
3	Wii	2009.0	Sports	Nintendo	82.65	1
4	GB	1989.0	Puzzle	Nintendo	88.00	1

In the finalized dataset ready for modeling, the four features (Platform, Year, Genre, Publisher) containing no missing values were assigned as predictors, while "hit" was assigned as the target. For categorical variables such as Genre, Platform, and Publisher, the One Hot

Encoding method was deployed to convert them into binary variables that could be interpreted by the model. Modeling was approached with a test size of 0.2, a train size of 0.8, and stratifying the distribution of hit/non-hit so that the distribution would stay consistent. The ratio of hit/non-hit in training and testing was close to 74:26.

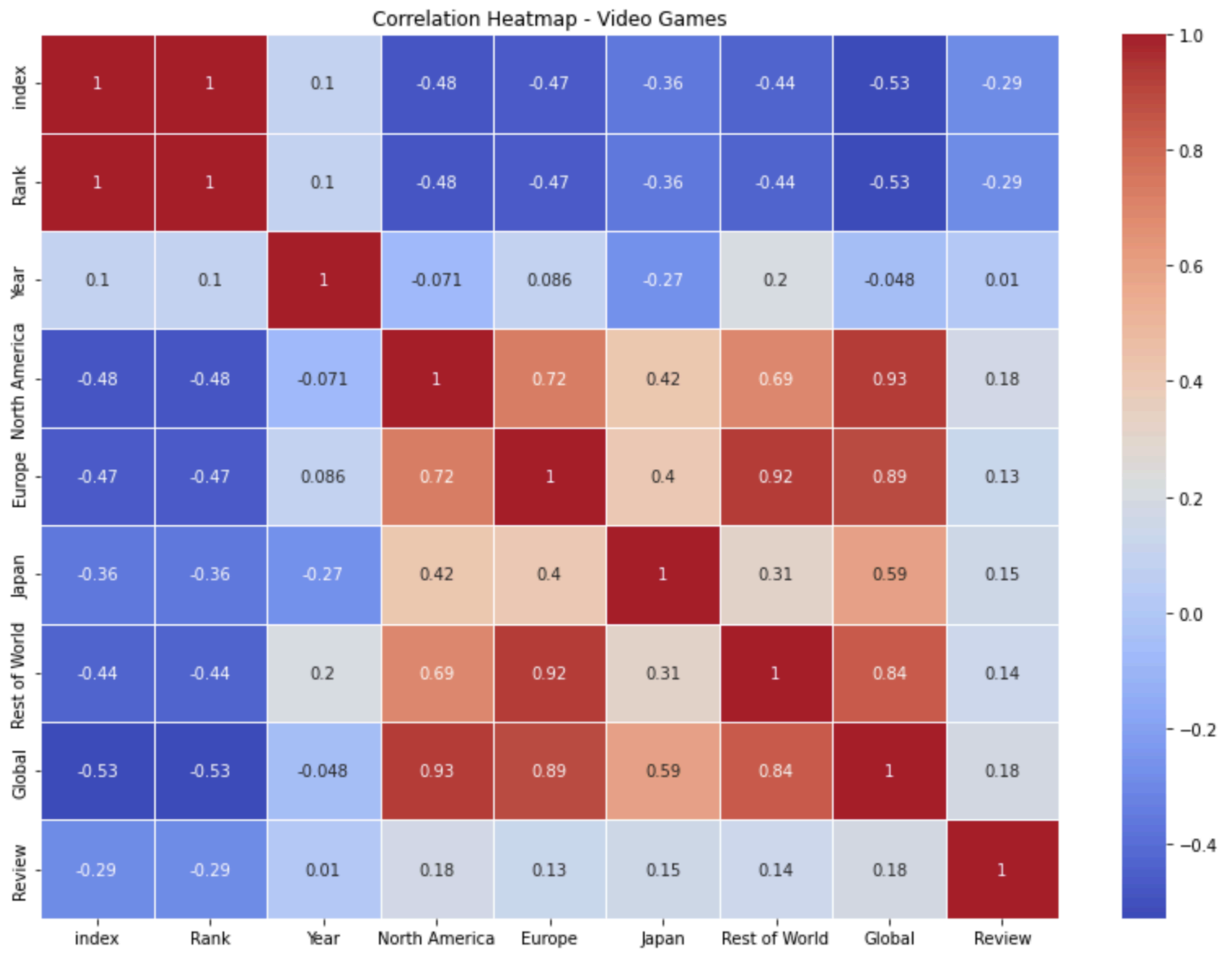
d. Feature Description

In our initial dataset, all the features available are displayed in the table below along with a description. Platform, Year, Genre, Publisher, and Hit were the only variables incorporated into our machine learning modeling.

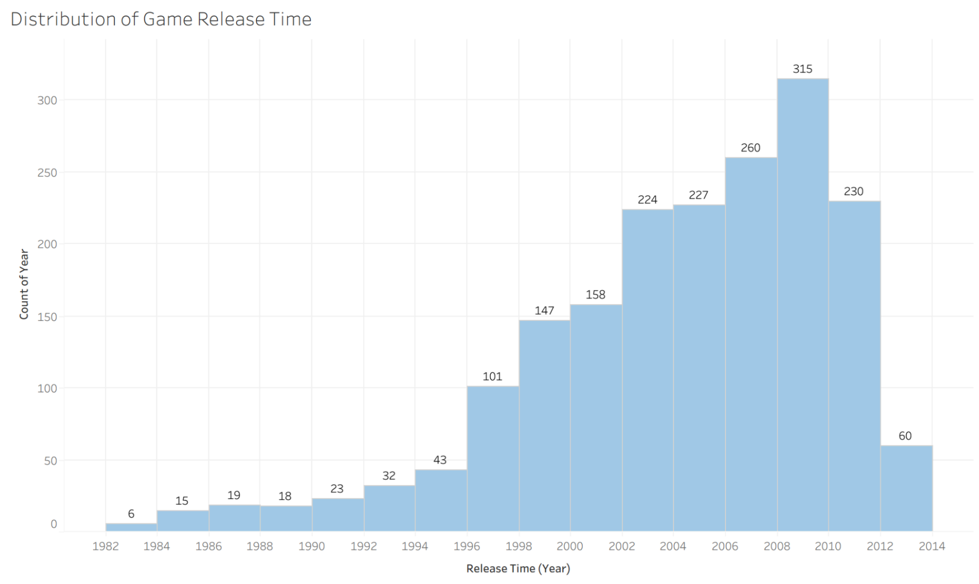
Feature Name	**Description**
Rank	Based on global sales, a number is assigned to a video game where 1 indicates the highest-ranked game.
Year	Release timing of the video game.
North America	Total video game sales specific to North America.
Europe	Total video game sales specific to Europe.
Japan	Total video game sales specific to Japan
Rest of World	Total video game sales everywhere other than North America, Europe, or Japan.
Global	Total video game sales all over the planet.
Game Title	The video game's name/title.
Platform	The console that the video game was released to.
Genre	The category of the video game.
Publisher	The video game's publisher.
Review	The assigned score of the video game in the dataset.
Hit	An indication of whether the video game was a hit or not. If a video game was given a review of 75 or higher, it was assigned to be a hit (1), but if the video game was given a score of below 75, it was assigned to be a non-hit (0).

e. Visualization (Exploratory Data Analysis using Tableau and Python)

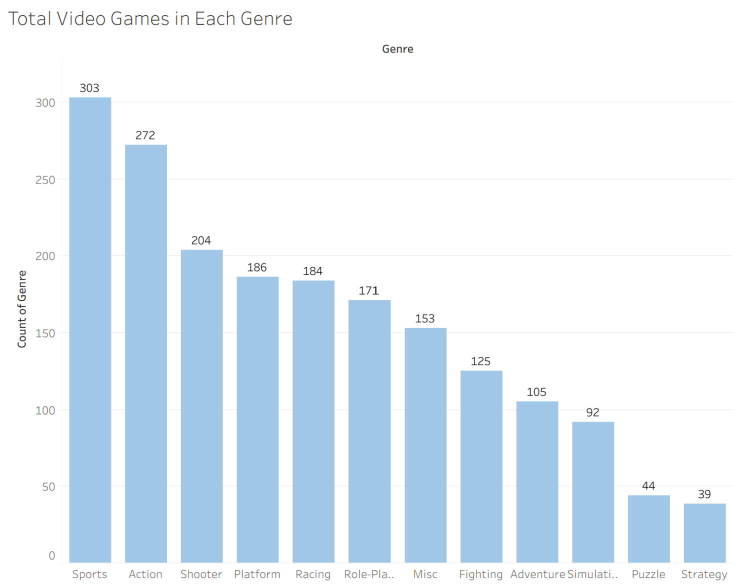
This heatmap shows how the numeric variables in the dataset relate to each other. As expected, the strongest correlations appear among the different regional sales columns, since games that sell well in one region tend to perform well globally. For example, Europe and North America sales have a strong positive correlation, and both correlate highly with Global sales, which is simply the sum of all regions.



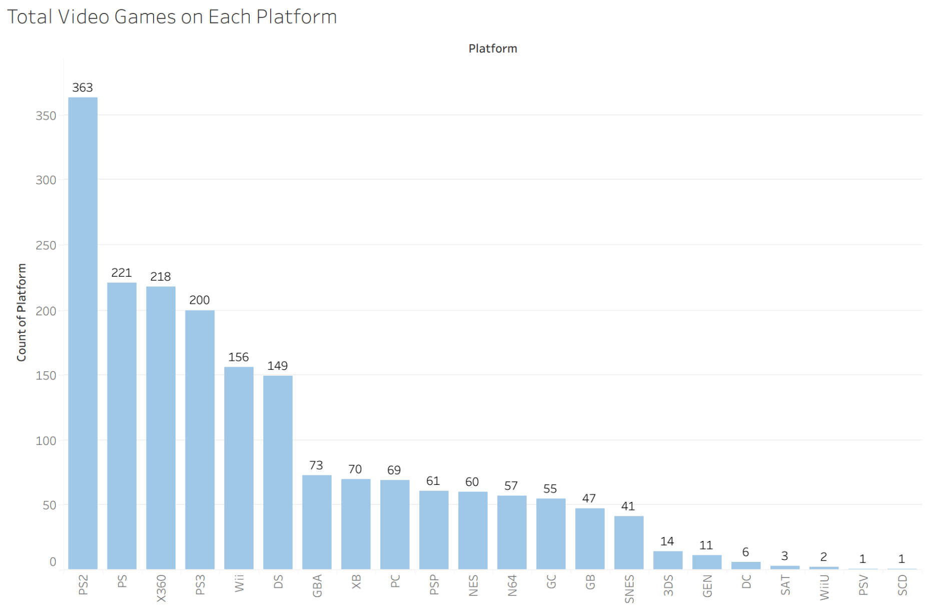
The visual below demonstrates the distribution of video game release timing. This dataset contains games released between 1982 and 2014. Most of the video games were released after 2000, with the highest number of games (315) being released between 2008 to 2010.



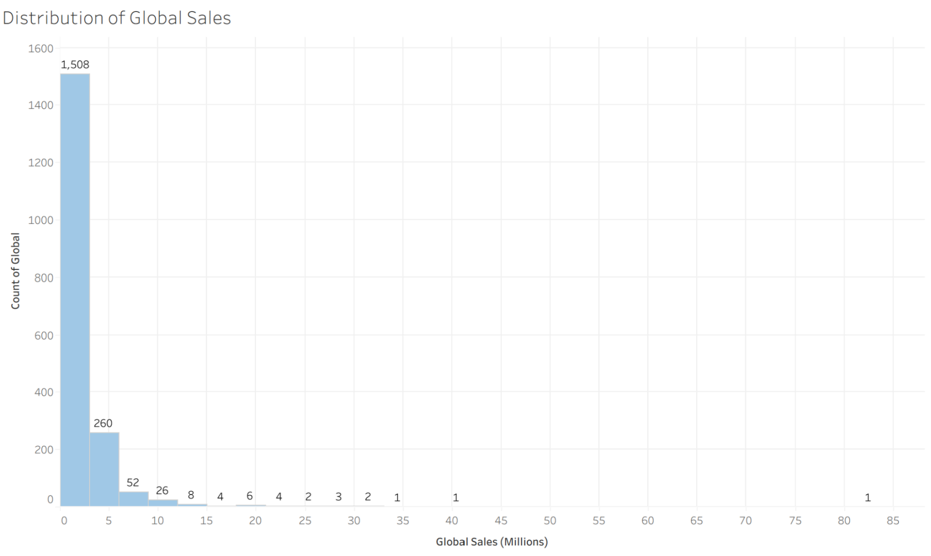
The visual below displays the total number of video games in each genre. There is a high number of video games belonging to the Sports Genre (303) and only a few strategy games (29) available in this dataset.

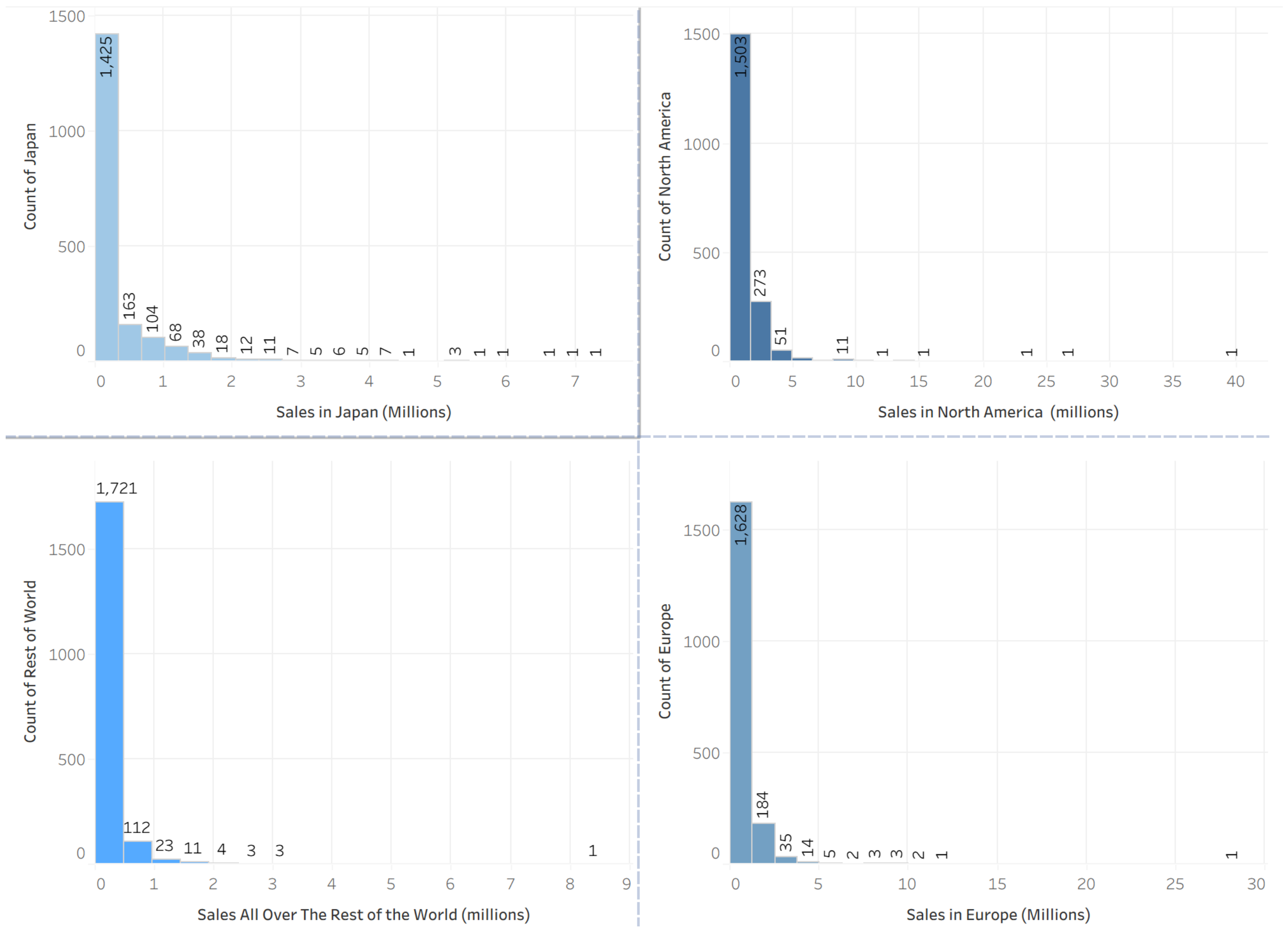


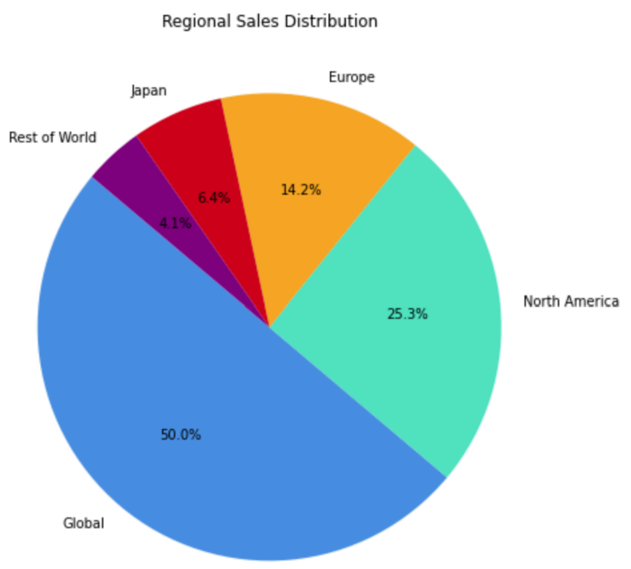
The visual below shows the total video games in each of the platform categories. The highest number of video games was released on the PS2 (363), while very few video games in the dataset were released on PSV(1) or SCD(1).



The bar charts and pie charts below show the distribution of sales in North America, Japan, Europe, and all over the Rest of the World. In all the distributions, most of the sales fall below one million. The pie chart shows that 50% of the sales were distributed globally, meanwhile, only 4.1% were distributed over the Rest of the World.




Count of Japan
Sales in Japan (Millions)
1,425
163
104
68
38
18
12
11
7
5
6
5
7
1
3
1
1
1
1
1
Count of North America
Sales in North America (millions)
1,503
273
51
11
1
1
1
1
1
Count of Rest of World
Sales All Over The Rest of the World (millions)
1,721
112
23
11
4
3
3
1
Count of Europe
Sales in Europe (Millions)
1,628
184
35
14
5
2
3
3
2
1
1


Regional Sales Distribution
Europe
14.2%
Japan
6.4%
Rest of World
4.1%
North America
25.3%
Global
50.0%

III. Analysis

Because this project focuses on predicting whether a game becomes a hit or a non-hit, we approached the problem as a binary classification and implemented four machine learning models:

a. Logistic Regression
b. Naive Bayes
c. Decision Tree
d. Random Forest

Each model was selected to represent a different family of classification approaches, allowing us to compare linear, probabilistic, rule-based, and ensemble methods. To clarify, after the preprocessing steps described in the Data Preparation section, including transforming all categorical variables (Platform, Genre, and Publisher) into binary features using one-hot encoding, the dataset was converted into a fully numeric structure suitable for all four models.

All models were developed using the scikit-learn package in Python and were trained using an 80%-20% stratified train-test split to ensure consistency with the preprocessing steps and fair comparison across models. In addition to the train-test split, we also used 10-fold cross-validation during model development to lessen the model's sensitivity to how the data is split. In 10-fold cross-validation, the training data is divided into ten equal parts. The model trains on nine parts and validates on the tenth, and this process repeats ten times so that every portion of the data is used for validation once.

A key challenge in this project is the imbalance in the dataset: we have 1,415 hit games and 492 non-hit games. Because hits make up more than 70% of the data, accuracy alone becomes misleading. A model could predict "hit" for every game and still achieve high accuracy without actually learning meaningful patterns. For this reason, we prioritize metrics that better reflect the model's ability to identify hit games, which would be the recall, F1-score, and AUC score. Recall tells us how many true hit games the model successfully identifies, F1-score captures the balance between precision and recall, and AUC measures how well the model separates hits from non-hits across all thresholds. Together, these metrics allow us to assess how effectively each model detects hit games and generalizes beyond the training data, rather than how often it simply predicts the majority class.

a. Logistic Regression

Logistic Regression was the first model we explored. It is one of the most common machine learning methods utilized for binary classification problems like our project. The model helped us predict whether a game becomes a hit by modeling the relationship between the features and the log odds of being a hit. It expresses this relationship as

$$In\left(\frac{P(hit)}{1-P(hit)}\right) = \beta_0 + \beta x ,$$

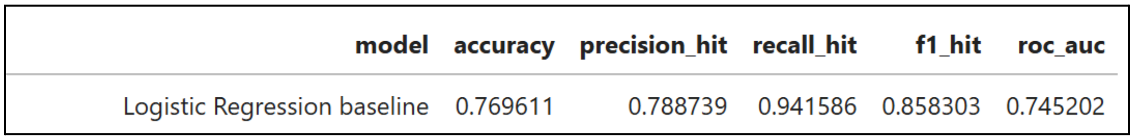
Where *x* represents the feature values (Platform, Year, Genre, and Publisher) for a game, and β represents the learned coefficients that measure how strongly each feature influences the odds of a game becoming a hit. The model then converts this log-odds value into a probability using the sigmoid function:

$$P(hit \mid X) = \frac{1}{1+e^{-(\beta_0 + \beta x)}}$$

Because Logistic Regression is simple, stable, and highly interpretable, even with many one-hot encoded features, it serves as a strong starting point for understanding which aspects of video games are most associated with hit status before moving on to more flexible or complex models. In our implementation, we used the liblinear solver from scikit-learn with different combinations of regularization, class weights, and SMOTE ("Logistic Regression").

a.i. Logistic Regression Baseline (L2, No Class Weight)

To begin, we initialized a baseline Logistic Regression model without class weights or resampling using the default L2 (Ridge) regularization from scikit-learn, which helps prevent overfitting by shrinking coefficient values. Below are the results of the baseline model:

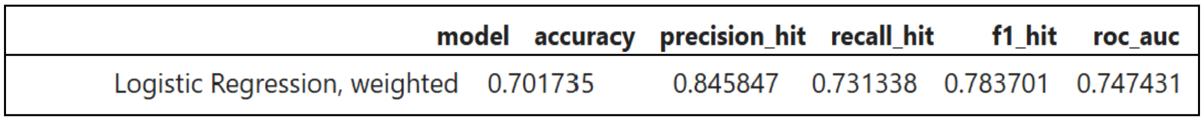


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression baseline	0.769611	0.788739	0.941586	0.858303	0.745202

Using our priority evaluation metrics, the baseline model achieved a hit recall of 0.94, indicating that it detected 94% of the true hit video games. The F1-score was 0.86, indicating a stable balance between identifying hits and predicting them reliably. The AUC of 0.75 indicates the model has an acceptable ability to distinguish hits from non-hits. Overall, this logistic regression baseline performed significantly well at detecting hit video games and overall ranking ability.

a.ii. Logistic Regression with L2 + Class Weights

Because the dataset is imbalanced, we next added class_weight = "balanced" to increase the penalty for misclassifying non-hit games. The results of the model are shown below:

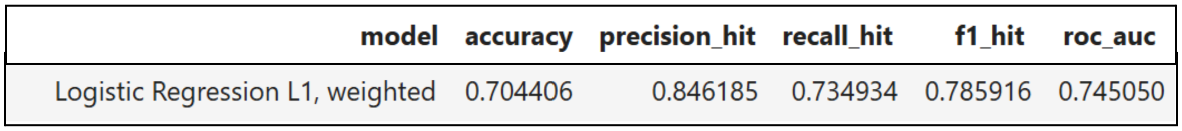


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression, weighted	0.701735	0.845847	0.731338	0.783701	0.747431

With L2 (Ridge) regularization and class weight, a hit recall dropped to 0.73, which was as expected since weights were used more for improving the prediction of the minority class. The F1-score also decreased to a mere 0.78, suggesting that the model's balance in identifying hits and predicting them reliably got worse. The AUC is mostly consistent with a score of 0.75, suggesting that with significantly lower recall and F1-score, the model still failed to improve the AUC. Overall, this model performed the worst compared to the baseline.

a.iii. Logistic Regression with L1 + Class Weights

To encourage sparsity and test whether a simpler set of features could improve generalization, we then applied L1 (LASSO) Regularization with class weights. This method approaches logistic regression with a procedure to prioritize important features, reducing some of the coefficients to 0 through the implementation of a penalty.

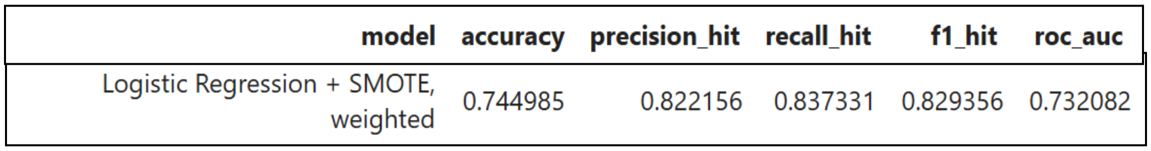


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression L1, weighted	0.704406	0.846185	0.734934	0.785916	0.745050

The chart above shows there was very little improvement in recall (0.73), f1-score (0.79), and AUC (0.75) compared to the L2 + Class Weights model. This indicates that switching from L2 to L1 did not materially change how well Logistic Regression distinguished hit games from non-hit games, even though it may have simplified the underlying coefficient structure.

a.iv. Logistic Regression with SMOTE + L2 + Class Weights

Since class weighting alone did not improve the model, we next combined Logistic Regression with SMOTE to oversample the minority (non-hit) class before training. The chart below reveals the result of the model:

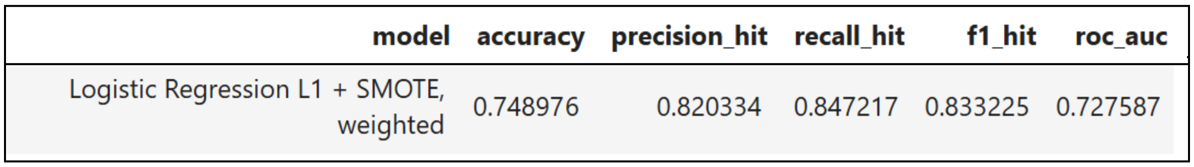


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression + SMOTE, weighted	0.744985	0.822156	0.837331	0.829356	0.732082

Using L2 regularization, class weights, and SMOTE, the model shows significant improvement. It was able to detect 84% of true hits and had an 83% balance in predicting hits and predicting them reliably. The AUC decreased a little, still indicating an acceptable ability in distinguishing hits from non-hits. Compared to the weighted models without SMOTE, this version recovered some recall while maintaining a strong F1-score, although its AUC was slightly lower than the baseline.

a.v. Logistic Regression with SMOTE + L1 + Class Weights

Finally, we combined SMOTE with L1-regularized Logistic Regression and class weights:

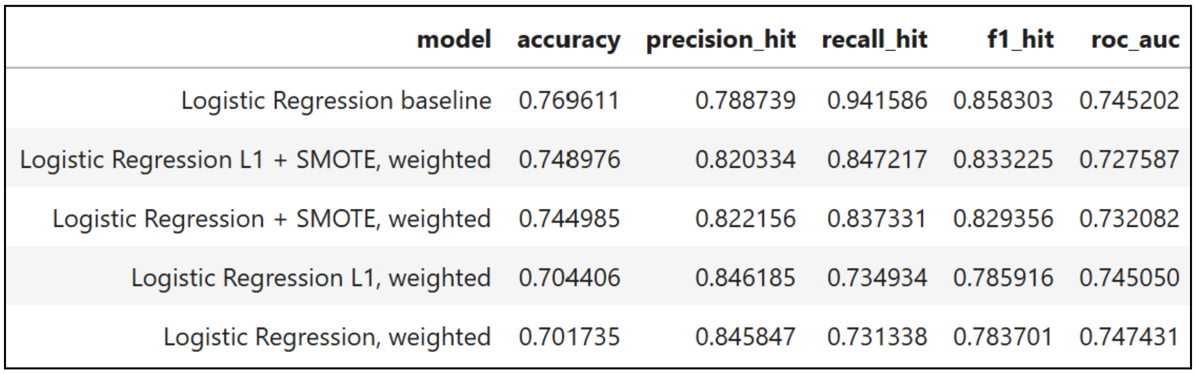


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression L1 + SMOTE, weighted	0.748976	0.820334	0.847217	0.833225	0.727587

The model appears to be slightly enhanced compared to the previous version. It was able to detect 85% of true hits and had an 83% balance in predicting hits and predicting them reliably. The AUC was mostly consistent, still indicating an acceptable ability in distinguishing hits from non-hits (0.73). Out of all the improvement methods, the combination of L1 Regularization and SMOTE with class weights refined the model.

a.v. Summary

The table below displays the best model performance for the Logistic Regression:



model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Logistic Regression baseline	0.769611	0.788739	0.941586	0.858303	0.745202
Logistic Regression L1 + SMOTE, weighted	0.748976	0.820334	0.847217	0.833225	0.727587
Logistic Regression + SMOTE, weighted	0.744985	0.822156	0.837331	0.829356	0.732082
Logistic Regression L1, weighted	0.704406	0.846185	0.734934	0.785916	0.745050
Logistic Regression, weighted	0.701735	0.845847	0.731338	0.783701	0.747431

Across all Logistic Regression variations, the baseline model (L2, No Class Weight) produced the highest hit recall. It was likely performed the best because of the unbalanced data that skewed it more towards predicting hits while underperforming in predicting non-hits. Thus, the result should be interpreted cautiously. The weighted and SMOTE-enhanced models provide fairer evaluations because they reduce this bias, even if their recall appears lower. Therefore, the model selection should not rely solely on the baseline's performance but should consider how well each model handles imbalance and generalizes beyond majority-class behavior. All in all, the SMOTE + L1 + Class Weights model is the best Logistic Regression model and will be used to compare its effectiveness in predicting hit video games with the other machine learning methods.

b. Gaussian Naive Bayes

The second method we explored was Gaussian Naive Bayes, a simple probabilistic classifier that predicts a class membership by estimating the likelihood of each feature under the assumption that all features are independent. Even though the independence assumption is unlikely to hold in our data, since many of the study's features tend to correlate (i.e., certain publishers release specific genres on specific platforms), we still explored this model because it provides a simple and interpretable baseline that allows us to use as a reference point for evaluating whether more flexible models meaningfully improve performance on predicting hit video games.

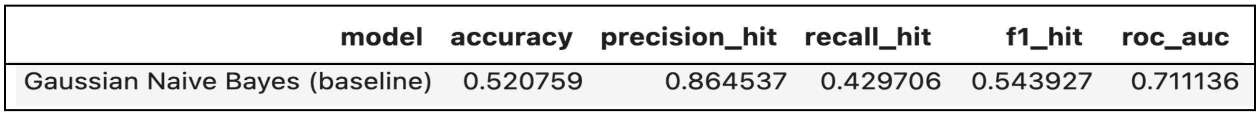
Mathematically, the model estimates the probability that a game is a hit given its feature values,

$$P(\text{hit} = 1 \mid x_i),$$

where x_i represents the features (Platform, Year, Genre, and Publisher) for a particular game. To compute this probability, the model assumes that each feature follows a normal distribution within each class. For the hit class and the non-hit class, the model estimates a mean (μ_k) and a variance (σ_k^2) for every feature. The likelihood of observing a feature value x_i under class k is then computed by substituting x_i into the Gaussian density function parameterized by μ_k and σ_k^2. These likelihoods are then combined to determine whether a game should be classified as a hit.

b.i. Gaussian Naive Bayes Baseline

To start with, here are the results for our Gaussian Naive Bayes baseline model:

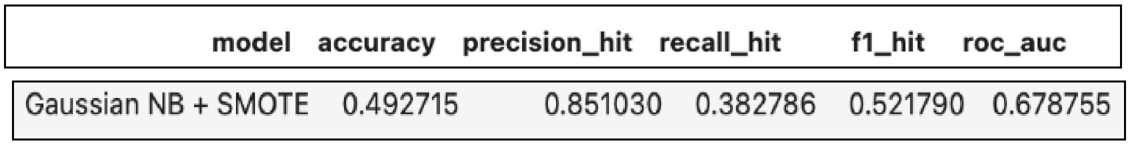


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Gaussian Naive Bayes (baseline)	0.520759	0.864537	0.429706	0.543927	0.711136

Using our priority evaluation metrics, the baseline model achieved a hit recall of 0.43, indicating that it detected less than half of the true hit games. Its hit F1-score was 0.54, reflecting a limited balance between identifying hits and predicting them reliably. The AUC of 0.71 shows that the model had moderate ability to distinguish hits from non-hits, even though this ability did not translate into strong hit detection.

b.ii. Gaussian Naive Bayes with SMOTE

Because the training data contains far fewer non-hit games than hit games, we applied SMOTE, which generated synthetic samples of the minority class, to balance the classes and give the model more opportunities to learn patterns associated with non-hits. Below are the results for our Gaussian Naive Bayes SMOTE model:

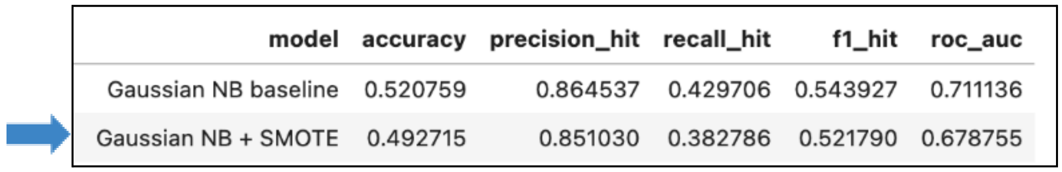


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Gaussian NB + SMOTE	0.492715	0.851030	0.382786	0.521790	0.678755

The SMOTE-enhanced Naive Bayes model achieved a hit recall of 0.38, slightly lower than the baseline, indicating reduced sensitivity to identifying actual hit games. The F1-score shifted to 0.52, showing only minor changes between detecting hits and predicting them accurately. The AUC decreased to 0.68, suggesting reduced ability to separate hits from non-hits.

b.iii. Summary

The chart below shows the comparison between the baseline and the SMOTE model:



model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Gaussian NB baseline	0.520759	0.864537	0.429706	0.543927	0.711136
Gaussian NB + SMOTE	0.492715	0.851030	0.382786	0.521790	0.678755

Overall, Naive Bayes did not gain predictive strength from resampling, and the baseline model produced the more reliable results. This pattern shows that synthetic samples did not support the assumptions of the model and may have added noise rather than useful information. However, because the baseline Naive Bayes model is trained on an imbalanced dataset, its performance cannot be directly interpreted as superior when comparing across all models in this project. Thus, the Gaussian Naive Bayes with SMOTE will be used when making the comparison. But given its low score, it may be more suitable to use other modeling approaches as the final method to predict hit video games.

c. Decision Tree

The next method we explored was Decision Trees, a classifier that predicts class membership by recursively splitting the dataset into smaller groups based on feature values, forming a tree-like structure of decision rules. Unlike Naive Bayes, Decision Trees do not assume that features are independent. Instead, they can naturally capture nonlinear patterns and interactions between variables. This flexibility is valuable for our project, where features such as

Platform, Year, Genre, and Publisher may work together in complex ways to influence whether a game becomes a hit.

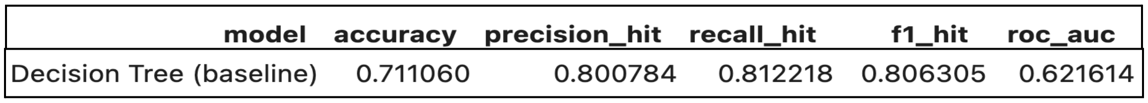
Mathematically, a Decision Tree builds a series of splits that break the dataset into increasingly pure groups, guiding the model toward distinguishing hit from non-hit games. In this project, we chose Gini impurity as the splitting criterion because it is fast to compute and provides a clear measure of how mixed the classes are within a node. Gini impurity is defined as:

$$G = \sum_{k=1}^{K} 2p_k(1 - p_k)$$

where p_k represents the proportion of samples belonging to class *k.* Lower Gini values indicate purer groups, so the tree selects splits that reduce impurity and create more homogeneous hit or non-hit sections of the dataset.

c.i. Decision Tree Baseline

Below are the results of our Decision Tree baseline model:

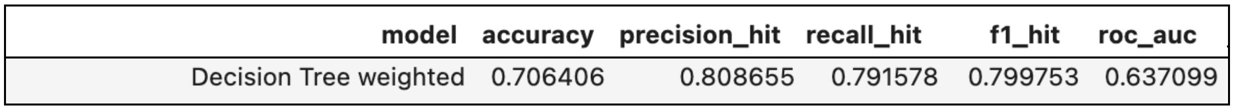


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Decision Tree (baseline)	0.711060	0.800784	0.812218	0.806305	0.621614

Using our priority metrics, the Decision Tree baseline achieved a hit recall of 0.81, successfully identifying most true hit games. Its hit F1-score was also 0.81, showing a strong balance between identifying hits and predicting them accurately. However, the model's AUC of 0.62 indicates weaker overall separation between hits and non-hits across thresholds. Overall, the baseline model performed well in terms of detecting hit games but showed limited ability to generalize its decision.

c.ii. Decision Tree with Class Weights

To address class imbalance, we improved and trained a version of the model using class_weight = "balanced" so that misclassifying non-hit games carried more weight. Below are the results for our Decision Tree with Class Weights ("balanced") model:

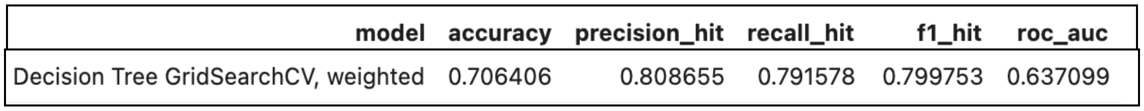


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Decision Tree weighted	0.706406	0.808655	0.791578	0.799753	0.637099

This model variation performed very similarly to the baseline. Hit recall decreased slightly to 0.79, hit F1-score also slightly lowered to 0.80, while AUC increased slightly to 0.64. The small differences suggest that reweighting the classes this way did not meaningfully change how the model learned its splits.

c.iii. Decision Tree with GridSearchCV + Class Weights

Because class weighting alone did not address the tree's limitations, we proceeded to tune key hyperparameters to test whether a more optimal tree structure (i.e., refining the tree's depth and split rules) could improve recall, F1-score, or AUC. (GeeksforGeeks, 2024)

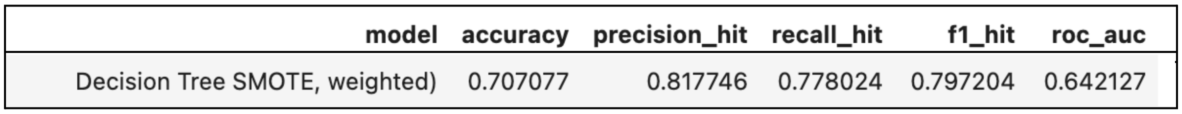


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Decision Tree GridSearchCV, weighted	0.706406	0.808655	0.791578	0.799753	0.637099

Based on the results above, the tuned model again showed the same performance: hit recall of 0.79, F1-score of 0.80, and AUC of 0.64. This shows that parameter adjustments alone were not enough to meaningfully improve the tree's ability to separate classes.

c.iv. Decision Tree with SMOTE + Class Weights

As the final attempt to improve the model, we applied SMOTE to oversample the minority class before training the tree, giving the model more exposure to patterns associated with non-hits. Here are the results of the SMOTE model:

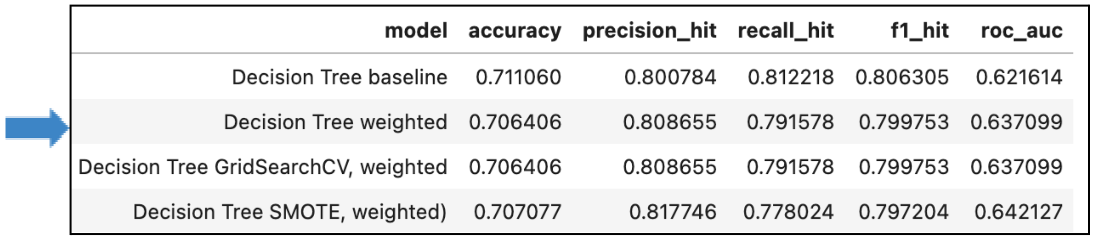


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Decision Tree SMOTE, weighted)	0.707077	0.817746	0.778024	0.797204	0.642127

The best SMOTE-enhanced version achieved a hit recall of 0.77, F1-score of 0.80, and AUC of 0.64, nearly identical to the weighted and GridSearchCV model.

c.v. Summary

The chart below shows the comparison between the baseline and other developed variation models:



model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Decision Tree baseline	0.711060	0.800784	0.812218	0.806305	0.621614
Decision Tree weighted	0.706406	0.808655	0.791578	0.799753	0.637099
Decision Tree GridSearchCV, weighted	0.706406	0.808655	0.791578	0.799753	0.637099
Decision Tree SMOTE, weighted)	0.707077	0.817746	0.778024	0.797204	0.642127

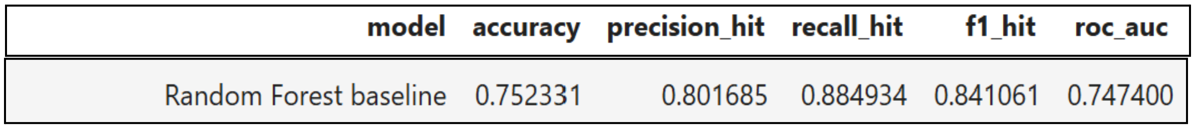
Each Decision Tree variation produced almost identical results, indicating that a single tree has limited capacity for this dataset. The consistently low AUC is expected because single trees overfit easily and are sensitive to class imbalance, making it difficult for them to form smooth, generalizable boundaries between hit and non-hit games. Although class weighting, GridSearchCV, and SMOTE offered minor adjustments, they did not overcome the structural limitations of a single tree. While the baseline variation performed slightly better within the Decision Tree Family, this should not be interpreted as the “best” model overall, as it also benefits from the underlying class imbalance and may not reflect true generalization. Thus, we used the next best model, Decision Tree with Class Weights, to compare with other methods. Still, these results suggest that more robust ensemble methods may be better suited for capturing the complex patterns involved in predicting video game hits.

d. Random Forest

The final method we explored was Random Forest, an ensemble method that builds multiple decision trees and aggregates their predictions. Unlike a single tree, which can also overfit and struggle with complex boundaries, Random Forest reduces variance by averaging across many trees trained on different bootstrap samples and randomized feature subsets. This makes it a strong candidate for our project because hit games may be influenced by nonlinear interactions among Platform, Year, Genre, and Publisher, patterns that a single tree might miss but an ensemble can detect through variation across many trees.

d.i. Random Forest Baseline

We first implemented a baseline Random Forest with 100 trees and no class weights. Below shows the results of the model:

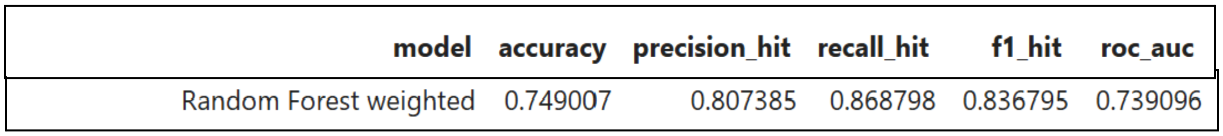


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Random Forest baseline	0.752331	0.801685	0.884934	0.841061	0.747400

At the initial stage, the Random Forest Baseline model has already outperformed all other models' baselines. The model achieved a hit recall of 0.88, meaning it was able to capture 88% of true video game hits. F1-score of 0.84, indicating the powerful ability to balance identifying hits and predicting them accurately. The AUC value of 0.75 explains that this model has a decent ability to distinguish hits from non-hits. Although promising, the baseline still risked being biased toward the majority class (hit games).

d.ii. Random Forest with Class Weights

As explained in earlier sections, the addition of "balanced" class weights to the model was an accommodation for the uneven distribution of hits/non-hits classes. The results below indicate the model's performance:



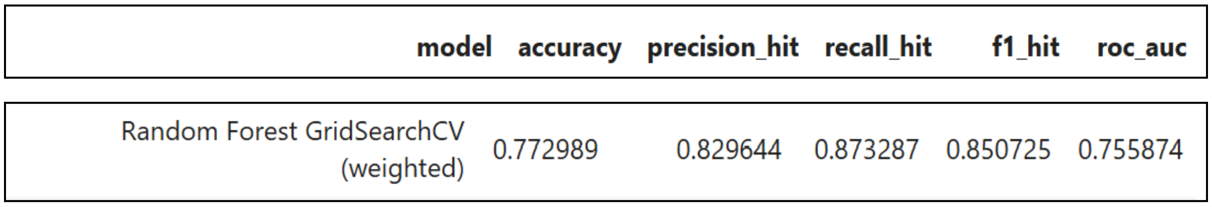
model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Random Forest weighted	0.749007	0.807385	0.868798	0.836795	0.739096

This variation presents similar results to the baseline, with a recall of 0.87, an F1-score of 0.84, and an AUC of 0.74. The similarity of the results of the baseline model and the Class Weights suggests that Random Forest was already handling the imbalance relatively well through bootstrapping and random feature sampling. Still, the similarity across metrics indicated that additional improvement would likely be possible.

d.iii. Random Forest with GridSearchCV + Class Weights

To further refine the model, GridSearchCV was applied to perform hyperparameter tuning on Random Forest for similar reasons mentioned earlier when applying this method to the

Decision Tree model. The tuned model consisted of 200 trees, a maximum depth of 20, square-root feature sampling, and balanced class weighting. The results of the model are shown below:

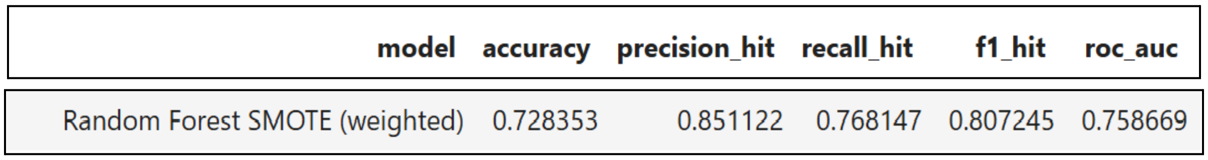


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Random Forest GridSearchCV (weighted)	0.772989	0.829644	0.873287	0.850725	0.755874

This improvement significantly boosted the performance of the Random Forest model. It was able to capture 87% of true hits, and improved its ability to detect and predict true hits as signified by its F1-Score (85%). Even with the addition of the class weights, the AUC value improved as well, reaching 0.76, still demonstrating an acceptable ability to distinguish hits from non-hits. These results show that hyperparameter tuning helped increase the model's ability to generalize while maintaining strong sensitivity to identifying hit games.

d.iv. Random Forest with SMOTE + Class Weights

Finally, SMOTE was applied to this model as well to oversample non-hits with synthetic data before training to give it more experience distinguishing the minority class. In this variation, we used a pipeline that first oversampled the non-hit class and then trained a weighted Ransom Forest with 200 trees and a maximum depth of 20. Here are the results of the SMOTE model:

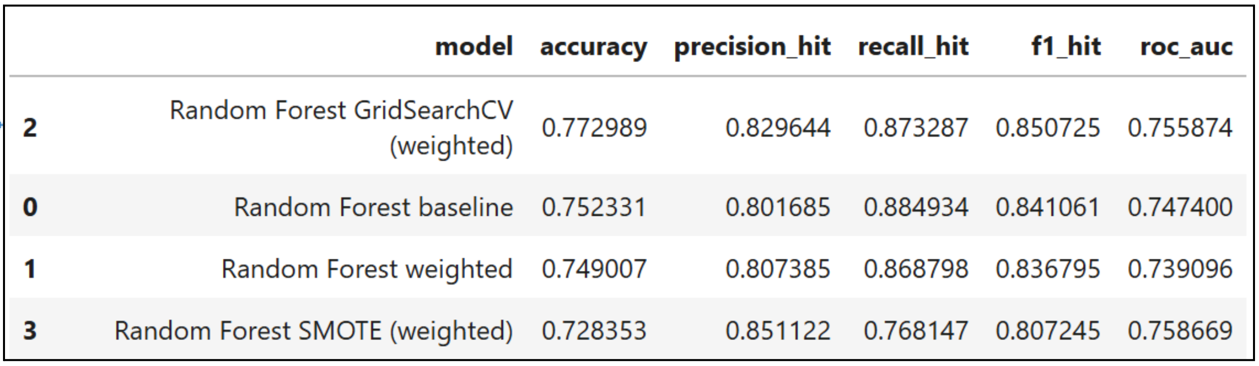


model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
Random Forest SMOTE (weighted)	0.728353	0.851122	0.768147	0.807245	0.758669

Unlike the GridSearchCV model, the SMOTE-enhanced version did not improve performance. It performed worse than all of the variations and the baseline, with a hit recall of 0.77 and an F1-score of 0.81, while AUC remained around 0.76. Because Random Forest already exposes each tree to different variations of the data through bootstrapping, SMOTH may have added synthetic noise rather than meaningful structure, causing the model to lose some sensitivity to true hit patterns.

d.v. Summary

The table below displays all the Random Forest model variations in order from best to worst performance:



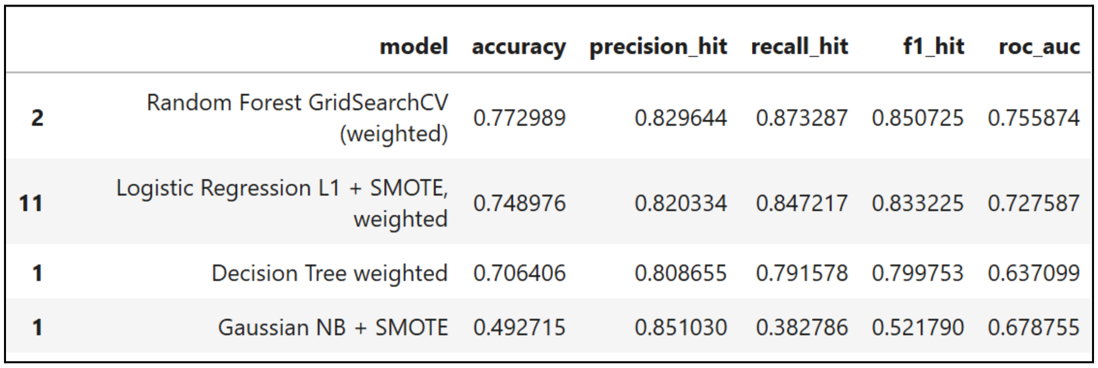
	model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
2	Random Forest GridSearchCV (weighted)	0.772989	0.829644	0.873287	0.850725	0.755874
0	Random Forest baseline	0.752331	0.801685	0.884934	0.841061	0.747400
1	Random Forest weighted	0.749007	0.807385	0.868798	0.836795	0.739096
3	Random Forest SMOTE (weighted)	0.728353	0.851122	0.768147	0.807245	0.758669

The Random Forest GridSearchCV performed the best, while the Random Forest SMOTE (weighted) performed the worst. GridSearchCV helps the Random Forest model become more stable. By controlling complexity through tuned depth and leaf size parameters, this model reduced unnecessary variance while still capturing meaningful patterns across all features. This is likely what caused the model to have the best performance in model evaluations.

e. Overall Model Evaluation, Significance, and Performance

e.i. Determining the Model with Best Performance

The table below indicates the top performance of all the machine learning methods used (not including baseline).

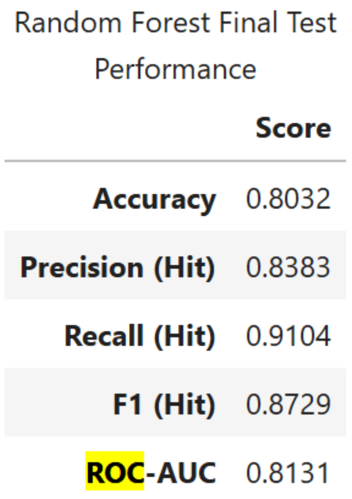


	model	accuracy	precision_hit	recall_hit	f1_hit	roc_auc
2	Random Forest GridSearchCV (weighted)	0.772989	0.829644	0.873287	0.850725	0.755874
11	Logistic Regression L1 + SMOTE, weighted	0.748976	0.820334	0.847217	0.833225	0.727587
1	Decision Tree weighted	0.706406	0.808655	0.791578	0.799753	0.637099
1	Gaussian NB + SMOTE	0.492715	0.851030	0.382786	0.521790	0.678755

Overall, the Random Forest GridSearchCV with Class Weights model scored the highest in its evaluation. After concluding that Random Forest GridSearchCV with Class Weights was the best model, it was then trained on the entire training dataset to ensure catching any missing patterns that may have been left out during cross-validation. The model was then implemented on the unused testing dataset from the 80 (training): 20 (testing) split.

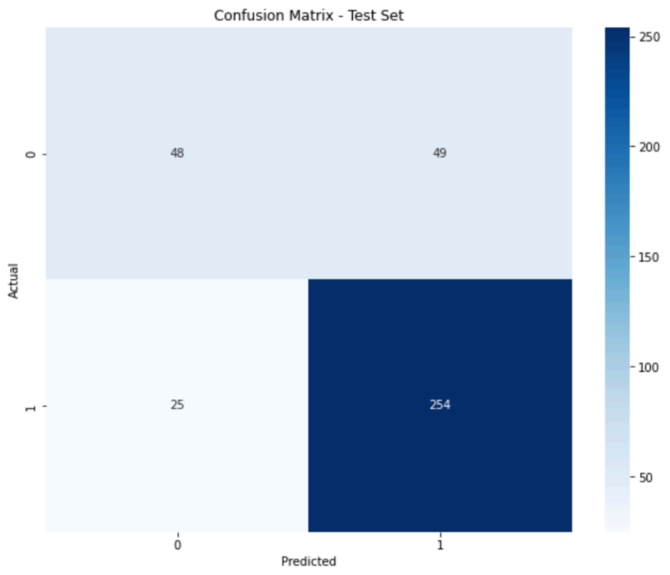
e.ii. Further Evaluation

Random Forest GridSearchCV with Class Weights' overall performance on the testing data was further evaluated with precision, recall, F1-score, AUC, and the confusion matrix. The model achieved a high recall (0.91), meaning it correctly identified most hit games. Its precision (0.84) shows that most of the games it predicted as hits were truly hits, indicating it was not an over-predicting success. The F1 score (0.87) balances these two strengths and confirms that the model handles the imbalanced dataset effectively. Finally, an AUC of 0.81 shows that the model has strong overall ranking ability and can reliably separate hit vs. non-hit games as seen below.


Random Forest Final Test Performance

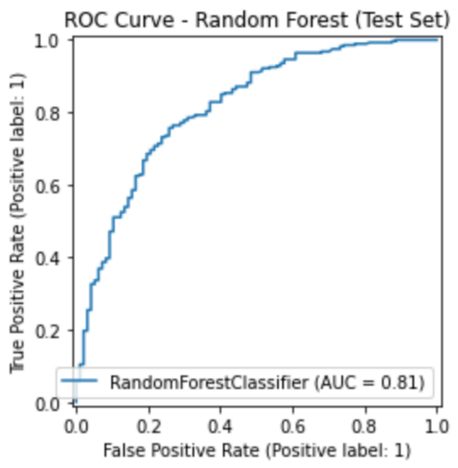
	Score
Accuracy	0.8032
Precision (Hit)	0.8383
Recall (Hit)	0.9104
F1 (Hit)	0.8729
ROC-AUC	0.8131

The confusion matrix is an additional evaluation that displays the total number of true positives, true negatives, false positives, and false negatives in our model. In context of our data, a false positive indicates the model accurately predicted a video game was a hit, a true negative indicates the model accurately predicted a video game as a non-hit, a false positive would describe an instance where the model incorrectly predicted a video game as a hit, and a false negative stipulates that the model inaccurately predicted a video game as a non-hit. The figure below displays the evaluation results of the confusion matrix.



The confusion matrix displays 254 hit games (true positives) and 48 non-hits (true negatives) accurately; however, it misclassified 25 hits as non-hits (false negatives) and 49 non-hits as hits (false positives). The high number of true positives and false positives is expected given the 74:26 class imbalance in hits/non-hits. Because our model focuses on predicting hit class, it was not a significant concern that the model performed poorly in evaluating non-hits.

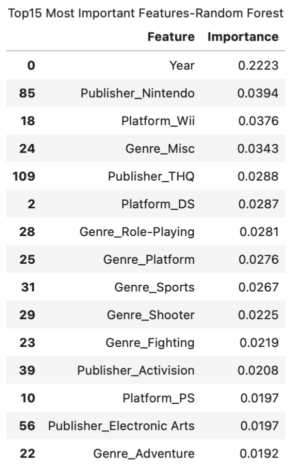
The ROC curve visualizes how well the model can separate hit and non-hit games across all possible thresholds.



The curve is raised towards the top left area of the graph, indicating an AUC value closer to 1. Specifically, the AUC value is 0.81, providing information that the model performs well in predicting actual hits, which aligns with the evaluation of the confusion matrix. The reason this value varies from the ROC-AUC value of the Random Forest GridSearchCV with Class Weights model is caused by the discrepancies in the calculations. Specifically, the graph uses RocCurveDisplay imported from sklearn.metrics, while the previous calculations depend on roc_auc_score and roc_curve imported from sklearn.metrics. (GeeksforGeeks, 2024)

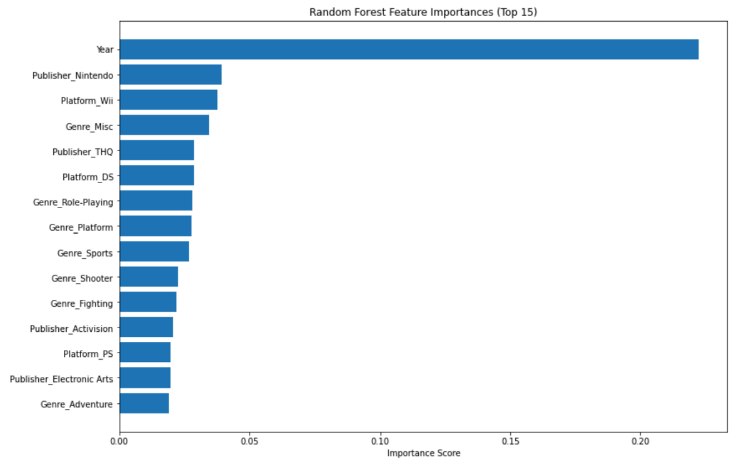
e.iii. Feature Importance

The top 15 features were studied using the model's importance scores. The figures below display these results in a table and bar chart format.



Top15 Most Important Features-Random Forest

	Feature	Importance
0	Year	0.2223
85	Publisher_Nintendo	0.0394
18	Platform_Wii	0.0376
24	Genre_Misc	0.0343
109	Publisher_THQ	0.0288
2	Platform_DS	0.0287
28	Genre_Role-Playing	0.0281
25	Genre_Platform	0.0276
31	Genre_Sports	0.0267
29	Genre_Shooter	0.0225
23	Genre_Fighting	0.0219
39	Publisher_Activision	0.0208
10	Platform_PS	0.0197
56	Publisher_Electronic Arts	0.0197
22	Genre_Adventure	0.0192



Year emerged as the most influential feature with a value of 0.22. Publishers such as Nintendo, THQ, Activision, and Electronic Arts also held positive coefficients, determining them as likely predictors for a video game being a hit. The platforms indicated as likely predictors for a video game hit include Wii, DS, and PS. Miscellaneous, Role-Playing, Platform, Sports, Shooter, Fighting, and Adventure are all genres that seem to be predictors for a video game being a hit.

.

IV. Takeaways

Our analysis provided several important insights about predicting hit video games using early, pre-release information. These insights fall into two categories: what we learned from the modeling process and what these findings imply for business strategy.

a. Insights Gained Through Data Mining

Early game features can meaningfully predict hit status. Even without sales or review data, our study found that features such as publisher, platform, genre, and release year consistently influenced whether a game became a hit. This shows that studios can evaluate a game's potential well before launch, long before expensive marketing campaigns or detailed playtesting occur.

Because the dataset was imbalanced, handling that was crucial. Since there were many more hit games than non-hits, accuracy alone was not a trustworthy measure of success. Techniques such as class weighting, SMOTE, and prioritizing metrics like recall, F1-score, and AUC were essential for building fair and interpretable models.

Models varied significantly in how they responded to improvements. Logistic Regression and Random Forest benefited the most from tuning and balancing methods, with stronger generalization and more stable separation between classes. In contrast, Decision Trees and Naive Bayes showed limited improvement across variations, indicating that simpler or assumption-heavy models may not fully capture the complexity of this dataset. to struggle.

b. Business Recommendations

Some publishers, platforms, and genres consistently correlated with higher hit rates. Random Forest feature importance and model trends suggest that certain publishers release disproportionately successful games, some platforms tend to host stronger performing titles, and particular genres outperform others. These insights can guide studios in aligning projects with historically successful patterns or identifying gaps in the market.

It is important to use early metadata to guide development decisions. Before investing heavily, studios can run predictive models to estimate whether a game has "hit" potential based on its planned attributes. This helps teams assess whether a game aligns with market trends before committing significant development and marketing resources.

Random Forest was the best overall model. After tuning with GridSearchCV, Random Forest had the strongest performance and identified the most important drivers of success. It captures complex patterns better than the other models, making it the most reliable choice. From a business perspective, its stability and interpretability reduce uncertainty and help minimize financial risk when selecting and prioritizing game projects.

V. Conclusion

Our final results reveal several patterns in what makes a video game more likely to be a hit. Even after removing all sales and review information, early attributes alone, especially platform, publisher, and genre, were highly predictive. Major game publishers appeared as strong positive indicators in Random Forests' most important features. This supported the idea that brand and publisher reputation greatly influence a game’s success. Platform and genre also played important roles, where specific consoles and game types produced significantly more hit titles compared to others.

These findings are crucial because they show that a game's success is not by random chance. Early-stage decisions, such as which platform to launch on, which publisher to partner with, or which genre to pursue, may truly benefit from such insights. For studios and developers, this means machine learning can serve as a great tool for project screening, strategic planning, and risk reduction. By identifying patterns that consistently lead to hit games, this analysis offers a data-driven insight into what developers, publishers, and investors can use to make more informed decisions in a highly competitive industry.

VI. References

- Logistic Regression | Stata Data Analysis Examples. https://stats.oarc.ucla.edu/stata/dae/logistic-regression/. Accessed 5 Dec. 2025.
- "Random Forest Hyperparameter Tuning in Python." GeeksforGeeks, 28 Dec. 2022, https://www.geeksforgeeks.org/machine-learning/random-forest-hyperparameter-tuning-in-python/.
- "Understanding the Confusion Matrix in Machine Learning." GeeksforGeeks, 15 Oct. 2017, https://www.geeksforgeeks.org/machine-learning/confusion-matrix-machine-learning/.
- "Why We Get Different Results with Roc_auc_score() and Auc()?" GeeksforGeeks, 8 July 2024, https://www.geeksforgeeks.org/machine-learning/different-results-with-rocaucscore-and-auc/.
- https://www.kaggle.com/datasets/thedevastator/discovering-hidden-trends-in-global-video-games. Accessed 1 Dec. 2025.
- About Us - Metacritic. https://neutron-cache.prod.metacritic.com/about-us/. Accessed 1 Dec. 2025.