steam copies sold prediction (CS)

| **Name** | **University ID** |
| --- | --- |
| عبدالرحمن تامر محمد عبدالفتاح نجم | 2022170230 |
| محمد طارق | 2022170617 |
| حسام عبدالرحمن محمد عبدالتواب | 2022170557 |
| مصطفى محمد مصطفى ابراهيم الشرقاوي | 2022170426 |
| هادي أحمد عثمان عطية الكفراوي | 2022170480 |
| يوسف احمد محمد على | 2022170506 |

Team: 23

**📄**

**Table of Contents**

1. **Introduction to Our Approach …………………………1**
2. **Data Preprocessing Techniques .................................. 2  
     2.1 Cleaning and Correcting Data Types  
     2.2 Merging Datasets  
     2.3 Handling Missing Values  
     2.4 Dealing with Inconsistencies  
     2.5 Handling Duplicates  
     2.6 Outlier Detection and Treatment**
3. **Exploratory Data Analysis (EDA)** ............................... 4  
     3.1 Feature Correlations  
     3.2 Data Distribution
4. **Regression Techniques Used** .................................... 6
5. **Model Comparison and Evaluation** ............................ 8  
     5.1 Accuracy and Error Metrics  
     5.2 Discussion of Differences
6. **Feature Selection** .................................................... 10  
     6.1 Used Features  
     6.2 Discarded Features and Justification
7. **Dataset Splitting** .................................................... 11  
     7.1 Training, Testing, and Validation Sets
8. **Performance Optimization Techniques** .................... 12  
     8.1 Feature Engineering
9. **Visualization of Regression Results** .......................... 13  
     9.1 Regression Line Plots
10. **Conclusion and Intuition Reflection** ........................ 14
11. **Classification Phase Analysis** ................................................... 15  
      11.1 Classification Accuracy, Training Time, and Test Time (Bar Graphs)  
      11.2 Feature Selection in Classification  
      11.3 Impact of Hyperparameter Tuning  
      11.4 Phase Conclusion and Insights

**Introduction to Our Approach**

In our approach, we avoided fetching data from external APIs. Instead, we focused on applying statistical techniques to extract meaningful insights from the dataset itself. By using random sampling, we aimed to design our analysis based on real-world data behavior rather than relying on external sources. We used this approach to simulate real-world AI modeling scenarios, where access to the Steam API is not always available.

**Data Preprocessing Techniques**

**2.1 Cleaning and correcting data type**

We began by exploring the dataset. During this process, we found that the appId field was stored as a string. Additionally, we noticed a duplicate row containing column headers, which we promptly removed.

Next, we ensured that all columns were correctly formatted according to their expected data types:

* **info\_base\_steam\_games table:**
  + Converted the release\_date column to **Date** format.
  + Any date written in terms of a **quarter** or **month** was reformatted properly. For quarterly entries, we selected the **middle month** of the quarter and used the **first day** of that month as the release date.
  + We also found games with release dates listed as **"Coming Soon"** or **"To Be Announced"**, even though they had reported copies sold. After manually checking these entries on Steam, we confirmed that most had already been released—many in **March 2025**. Therefore, we standardized these entries by setting their release date to **March 1, 2025**.
  + Converted the appid column to **Integer**.
* **gamalytic\_steam\_games table:**
  + Converted the steamId column to **Integer**.
* **dlcs table:**
  + Converted the base\_appid column to **Integer**.
* **demos table:**
  + Converted the full\_game\_appid column to **Integer**.

**2.2 Merging**

After cleaning and formatting the data, we proceeded with merging the tables. Before merging, we ensured that the ID fields in all relevant tables were correctly formatted to avoid unnecessary complications or misinterpretation of statistics.

Our merging strategy was as follows:

* We first performed an **inner join** between the info\_base\_steam\_games table and the gamalytic\_steam\_games table.
  + This was done because:
    - gamalytic\_steam\_games contains our **target variable (Y)**.
    - info\_base\_steam\_games includes our **main features**.
  + An inner join ensures we only keep the records that exist in both tables, which are essential for building our model.
* Next, we performed **left joins** with the dlcs and demos tables.
  + The reason for using a **left join** is that **not all games have DLCs or demos**, and we didn’t want to lose those games from the dataset.
  + Instead, we added two new features:
    - has\_dlc
    - has\_demo
  + These features were initially set to null for games that don’t have corresponding DLC or demo entries. Later, during the feature engineering phase, we converted these nulls into meaningful binary indicators (e.g., 0 = no DLC, 1 = has DLC).

We decided to retrieve additional game information for entries that exist in the **gamalytic\_steam\_games** table but are missing in the **info\_base\_steam\_games** table. To accomplish this, we used the **Steam API** to fetch the missing data. As a result, we successfully retrieved and added approximately **26,000** new rows of game information.

**2.3 Handling Missing Values**

Below is a summary of the missing (null) values in our dataset:

| **Column** | **Number of Nulls** |
| --- | --- |
| metacritic | 92,501 |
| genres | 132 |
| achievements\_total | 43,998 |
| release\_date | 3 |
| dlc\_appid | 91,365 |
| dlc\_game\_name | 91,365 |
| demo\_appid | 87,031 |
| demo\_game\_name | 87,032 |

For the **DLC and demo columns**, we created two new boolean columns (FE) :

* + has\_dlc: 1 if dlc\_appid is not null, otherwise 0
  + has\_demo: 1 if demo\_appid is not null, otherwise 0  
    After creating these features, we dropped the original dlc\_appid, dlc\_game\_name, demo\_appid, and demo\_game\_name columns.
* The **ai\_content** column was entirely null, so we dropped it from the dataset.
* For the **metacritic** column:
  + Initially, we attempted to fill null values with -1, but this negatively impacted the model’s accuracy.
  + We then tried filling with 0, which yielded better results.
  + (Note: Filling with the mean was considered, but not used.)
* In the **achievements\_total** column, we discovered that a null value actually indicated that the game had no achievements. Therefore, we filled all nulls with 0.
* For the **release\_date** column, only three values were missing. We manually retrieved and filled them using online sources.
* In the **genres** column:
  + We grouped the data by publisher\_class, identified the **mode genre** within each group, and used any mode greater than 40% to fill in the missing genres , if not found we use the top two modes to fill .

**2.4 Dealing With Inconsistencies**

During data exploration, we identified several inconsistencies within the dataset that required attention:

* **Steam Achievements:**
  + We noticed inconsistencies between the steam\_achievements flag and the achievements\_total column.
    - Some games had steam\_achievements = false but still had a non-zero total count.
    - Others had steam\_achievements = true but a null value for the total.
  + To resolve this, we decided **not to rely on the steam\_achievements flag**.
  + Instead, we recreated it , based on actual values found through the Steam API and verified that these were more reliable.
* **Workshop Support & Steam Trading Cards:**
  + While validating the truthfulness of the dataset, we discovered that **some games had incorrect values** for the workshop\_support and steam\_trading\_cards fields.
  + Upon further investigation, we noticed a **pattern among games that had sold fewer than 20,000 copies**:
    - Many of these games were incorrectly marked as having workshop\_support = true and steam\_trading\_cards = true.
  + We manually checked over 200 such games on Steam and found that in the majority of cases, both values should actually be false.

**2.5 Dealing with duplicates**

We identified several duplicate entries based on the steamId, where values for workshop\_support and steam\_trading\_cards differed. To resolve this, we manually verified the data on Steam and found that most entries with both values set to true were incorrect. These incorrect entries were subsequently removed.

After this cleanup, only two duplicate entries remained. We manually inspected them and removed the inaccurate ones.

It is worth noting that while there are duplicate values in the **game names**, this is expected behavior on Steam, as multiple games can share the same name but have different steamId

**2.6 Dealing With Outliers**

We applied the **Interquartile Range (IQR)** technique to identify outliers in the dataset. The following number of outliers were detected in each feature:

* **price**: 4,796 outliers
* **copiesSold**: 15,861 outliers
* **reviewScore**: 7,379 outliers
* **metacritic**: 3,798 outliers
* **achievements\_total**: 5,686 outliers

We tried to handle outliers, but when we removed them, the model'*s* accuracy decreased. Therefore, we conclude that removing outliers is not always beneficial; it might actually be better to keep them when working with real-world data**.**

**Exploratory Data Analysis (EDA)**

**3.1 Feature Correlations**

**We used two techniques to measure correlation: Pearson correlation and ANOVA**

* **Pearson correlation** was used to measure the linear relationship between continuous numerical features and the target variable.
* **ANOVA** was applied to assess the relationship between categorical features and the target variable

| **Feature** | **F-Score** | **P-Value** |
| --- | --- | --- |
| has\_metacritic | 18.4638 | 0.0000000000 |
| encoded\_publisherClass | 4.9073 | 0.0000000000 |
| is\_free | 3.4651 | 0.0000000000 |
| has\_dlc | 2.9135 | 0.0000000000 |
| mac | 1.3337 | 0.0000000000 |
| linux | 1.2169 | 0.0000000000 |
| encoded\_genres | 1.1093 | 0.0000000000 |
| encoded\_supported\_platforms | 1.1045 | 0.0000000000 |
| encoded\_sorted\_genres | 1.0850 | 0.0000000000 |
| has\_demo | 1.0056 | 0.3041007760 |
| windows | 0.1859 | 1.0000000000 |

**Pearson correlation ANOVA**

| **Feature** | **Pearson Correlation** |
| --- | --- |
| price | 0.0227 |
| reviewScore | 0.0218 |
| metacritic | 0.1163 |
| steam\_achievements | 0.0253 |
| steam\_trading\_cards | 0.0725 |
| workshop\_support | 0.0841 |
| achievements\_total | 0.0145 |
| months\_passed | 0.0581 |
| days\_passed | 0.0581 |
| years\_passed | 0.0584 |
| month | 0.0072 |
| day | 0.0002 |
| year | -0.0584 |

**3.2 Data Distribution**

**A graph with numbers and lines

Description automatically generatedA graph with numbers and a number

Description automatically generated with medium confidence**

**A graph with a purple square

Description automatically generated with medium confidenceA graph with a blue square

Description automatically generatedA graph with a bar graph

Description automatically generatedA graph with numbers and letters

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generatedA screenshot of a graph

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph with numbers and letters

Description automatically generated with medium confidenceA graph with a bar and a number of text

Description automatically generated with medium confidenceA graph with a purple rectangle

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph with a purple bar

Description automatically generated with medium confidenceA screenshot of a graph

Description automatically generatedA purple rectangle with white text

Description automatically generatedA purple rectangular object with black text

Description automatically generatedA graph with a bar

Description automatically generatedA purple rectangle with white text

Description automatically generatedA graph with a bar and numbers

Description automatically generated with medium confidenceA graph with a number of bars

Description automatically generatedA graph with a bar graph

Description automatically generated with medium confidenceA graph with a red line

Description automatically generatedA graph of a number of columns

Description automatically generated with medium confidenceA graph with a bar and a number of numbers

Description automatically generated with medium confidenceA graph with a bar graph

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph with a bar graph

Description automatically generated with medium confidence**

**Regression Techniques Used**

**We used liner regression and polynomial regression with ridge (L2) and lasso (L1) we use catboost , design tree , Random forest and ElasticNet**

| **Model** | **Train MSE** | **Train R²** | **Log Test MSE** | **Log Test R²** | **Exp Test MSE** | **Exp Test R²** | **Bias Correction** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Linear Regression (Base) | 4.26 | 0.55 | 4.20 | 0.56 | 473,026,948,983.47 | 0.54 | 0.92 |
| Ridge Regression (Base) | 4.26 | 0.55 | 4.20 | 0.56 | 472,546,816,461.72 | 0.54 | 0.92 |
| CatBoost Regression (Base) | 2.25 | 0.76 | 2.43 | 0.75 | 381,463,203,332.73 | 0.63 | 0.87 |
| Lasso Regression (Base) | 6.61 | 0.31 | 6.59 | 0.31 | 964,749,466,654.87 | 0.06 | 0.87 |
| ElasticNet Regression (Base) | 6.13 | 0.36 | 6.09 | 0.36 | 938,458,506,398.44 | 0.08 | 0.85 |
| DecisionTree Regression (Base) | 1.96 | 0.79 | 3.07 | 0.68 | 860,418,624,980.43 | 0.16 | 0.94 |
| RandomForest Regression (Base) | 0.54 | 0.94 | 2.91 | 0.69 | 827,786,070,470.21 | 0.19 | 1.00 |
| Linear Regression (Poly) | 3.96 | 0.59 | 3.90 | 0.59 | 15,933,781,302,660.40 | -14.54 | 0.84 |
| Ridge Regression (Poly) | 3.96 | 0.59 | 3.90 | 0.59 | 15,519,413,475,094.73 | -14.14 | 0.84 |
| CatBoost Regression (Poly) | 2.21 | 0.77 | 2.45 | 0.74 | 428,730,120,481.39 | 0.58 | 0.86 |
| Lasso Regression (Poly) | 4.64 | 0.51 | 4.58 | 0.52 | 533,861,877,724.61 | 0.48 | 0.90 |
| ElasticNet Regression (Poly) | 4.54 | 0.53 | 4.48 | 0.53 | 393,773,762,674.66 | 0.62 | 0.91 |

| **Model** | **Performance** | **Explanation** |
| --- | --- | --- |
| **CatBoost** | Very High | Handles categorical features well, prevents overfitting, and is highly optimized for gradient boosting. |
| **RandomForest** | Very High | Uses ensemble of decision trees, reduces variance and overfitting, good for complex data patterns. |
| **XGBRegressor** | High | Efficient and powerful boosting algorithm, performs well with structured data, slightly sensitive to tuning. |
| **DecisionTree** | High | Simple and interpretable, but can overfit without pruning or ensemble techniques. |
| **LinearRegression** | Moderate | Assumes linear relationships, may underfit complex data. |
| **Ridge** | Moderate | Adds L2 regularization, helps with multicollinearity, but still assumes linearity. |
| **Lasso** | Low | L1 regularization can zero out features, not ideal for complex data. |
| **ElasticNet** | Low | Combines L1 & L2, but may not add much value if data isn't sparse or highly correlated. |

**Feature Selection**

| Feature Name | Justification |
| --- | --- |
| price\_log | Log-transformed game price provides better distribution and captures economic signals influencing purchases. |
| reviewScore\_log | Represents player satisfaction and engagement; log transformation reduces skew. |
| metacritic\_log | External critic ratings impact user perception and trust in a game. |
| premium\_score\_review\_interaction | Captures interaction between game pricing and user reviews, reflecting perceived value. |
| encoded\_publisherClass | Publisher reputation and scale often correlate with game visibility and quality. |
| encoded\_sorted\_genres | Genre preferences are strong indicators of user interest and game type. |
| metacritic\_review\_interaction | Combines user and critic feedback for a holistic quality metric. |
| year | Encodes temporal trends and market evolution, affecting game popularity. |
| steam\_trading\_cards | May influence purchase for collectors or reward-seeking players. |
| workshop\_support | Community modding support can increase user retention and value. |
| achievements\_total\_log | Indicates game complexity and engagement level. |
| has\_dlc | Presence of downloadable content can affect user interest and game longevity. |
| has\_demo | Affects user trust and willingness to try a game. |
| is\_free | Price category directly affects accessibility and user base size. |
| encoded\_supported\_platforms | Platform availability broadens market reach and accessibility. |
| encoded\_sorted\_genres | Genre preferences are strong indicators of user interest and game type. |
| metacritic\_review\_interaction | Combines user and critic feedback for a holistic quality metric. |

**Discarding**

**Features Discarded via Pearson Correlation (Weak Linear Relationship)**

**|Pearson correlation| < 0.1 and ANOVA we chose the best from them and make meaning**

price\_log, reviewScore\_log, metacritic\_log, steam\_trading\_cards, encoded\_sorted\_genres, encoded\_publisherClass, premium\_score\_review\_interaction, workshop\_support, achievements\_total\_log, has\_dlc, has\_demo, is\_free, encoded\_supported\_platforms, year, metacritic\_review\_interaction

**Dataset Splitting**

We split the data into 80% for training and 20% for testing.

**Visualization of Regression Results**

**A graph with a line and a line

Description automatically generated with medium confidenceA graph with a red line and green line

Description automatically generated**

**A graph with a line and a red line

Description automatically generatedA graph with a line and a line

Description automatically generated with medium confidenceA graph with a red line and green line

Description automatically generatedA graph with a red line and green line

Description automatically generatedA graph with red line and green line

Description automatically generatedA graph with a red line and green line

Description automatically generatedA graph with a red line and green line

Description automatically generated**

**Performance Optimization Techniques**

Feature Engineering

As part of our data preprocessing and enhancement process, we performed several feature engineering steps to improve model performance:

#### 1. **Boolean Feature Creation**

We created several binary (Boolean) features based on categorical information, such as:

* is\_free: Indicates whether the game is free.
* has\_dlc: Indicates the presence of downloadable content.
* has\_demo: Indicates whether a demo version is available.
* has\_metacritic: Indicates the availability of a Metacritic score.

In addition, the supported\_platforms feature was expanded into separate Boolean columns for each platform:

* windows
* mac
* linux

#### 2. **Date-Based Feature Extraction**

From the release date, we extracted multiple temporal features to provide more meaningful insights into how time affects game popularity and sales:

* days\_passed
* months\_passed
* years\_passed

#### 3. **Derived Features**

We engineered a new feature called premium\_score, which combines indicators of premium game content:

premium\_score = steam\_trading\_cards + workshop\_support + has\_dlc + has\_metacritic

#### **Feature Transformation**

We applied logarithmic or normalization transformations to reduce skewness in the following numerical features:

* metacritic
* price
* reviewScore
* achievements\_total

#### **Label Encoding for Categorical Features**

To prepare categorical data for machine learning models, we applied label encoding to the following features:

* supported\_platforms
* publisherClass
* genres
* sorted\_genres
* copiesSold\_class

#### 6. **Feature Scaling we (cancelled it) effect data by revers**

Initially, we applied feature scaling to the X\_train dataset to standardize numerical features with the goal of improving model convergence and overall performance. This is particularly important for models sensitive to feature magnitude, especially when dealing with features that are not normally distributed.

However, after evaluation, we found that scaling negatively impacted model performance, likely due to the disruption of the original data distribution or relationships between features. As a result, we decided to **cancel the scaling step** in order to preserve the natural characteristics of the data, which led to more reliable and accurate model outcomes.

**Conclusion and Intuition Reflection**

In the preprocessing stage, we observed that while real-world data often contains outliers, handling or removing them did not always lead to improved model performance. In several cases, model accuracy decreased, suggesting that some outliers may carry meaningful patterns, and their removal might distort the true distribution of the data. Similarly, applying feature scaling techniques such as normalization or standardization did not consistently enhance results. Although scaling is essential for certain algorithms ,in our context, it may have disrupted the natural relationships between features and target variables, adversely impacting predictions. From the modeling perspective, introducing polynomial features to capture non-linear patterns occasionally led to underfitting, particularly when the underlying relationship was inherently linear. This underscores that increasing model complexity does not always yield better outcomes and can sometimes obscure valuable patterns. Conversely, simple linear regression frequently delivered the best performance, reinforcing the idea that when the relationship between input and output variables is approximately linear, simpler models are not only more effective but also easier to interpret and more robust.

**MS 2**

**Classification Accuracy, Training Time, and Test Time (Bar Graphs)**

**A graph with different colored bars

Description automatically generated with medium confidence**

**Feature Selection**

In this classification phase, our feature selection approach differed from the earlier regression phase to better suit the nature of classification tasks. We evaluated potential features including *price*, *reviewScore*, *steam\_achievements*, *achievements\_total*, *month*, *day*, *year*, *steam\_trading\_cards*, *workshop\_support*, *months\_passed*, *days\_passed*, and *years\_passed*.

To identify the most relevant features, we applied statistical methods such as the *ANOVA* and the *Chi-square* test. These methods helped us assess the strength of association between each feature and the target class.

**Then we get this features :**

price\_log, reviewScore\_log, metacritic\_log, steam\_trading\_cards, encoded\_sorted\_genres, encoded\_publisherClass, premium\_score\_review\_interaction, workshop\_support, achievements\_total\_log, has\_dlc, has\_demo, is\_free, encoded\_supported\_platforms, year, metacritic\_review\_interaction

**Impact of Hyperparameter Tuning (GridSearch)**

### **The model takes too long to train without significant improvement**

### **XGBoost Classifier**

**Hyperparameters Tuned:**

* **n\_estimators**: [50, 100, 500]  
  Number of boosting rounds. More estimators can improve performance but increase training time.
* **max\_features**: ['log2']  
  Determines the number of features to consider when looking for the best split. Using 'log2' helps reduce overfitting.
* **learning\_rate**: [0.01, 0.05, 0.1]  
  Controls the contribution of each tree. Lower values improve generalization but require more trees.

**Histogram-Based Gradient Boosting Classifier**

**Hyperparameters Tuned:**

* **max\_iter**: [50, 100, 500]  
  Equivalent to the number of boosting iterations or trees.
* **min\_samples\_leaf**: [2, 5, 10]  
  Minimum number of samples required to be at a leaf node. Helps prevent overfitting by avoiding overly specific splits.
* **learning\_rate**: [0.01, 0.05, 0.1]  
  Shrinks the contribution of each tree. A lower rate typically requires more iterations but generalizes better.

### **Phase Conclusion and Insights**

Hyperparameter tuning had minimal impact, and the stacking ensemble emerged as our best model—its accuracy was only about 0.001 higher than the XGboast . Overall, our classification models outperformed the regression approach. Despite extensive grid searches, hyperparameter adjustments did not yield the expected gains. We also tried to address class imbalance by applying class weights, but this had negligible effect on positive‐class performance. Finally, although we experimented repeatedly with an SVM (trying different kernels and regularization values), its training time proved prohibitively long, ruling it out for our use case.