

Video Game Sales

Exploring and Predicting Video Game Sales Using Data Visualization and Machine Learning

Group 13:

- Haitham Assaf
- Shruthi Raghavachary
- Kaulan Serzhanuly

Date: 11/30/2025

Table of Contents

- **Abstract** 3
- **1. Background and Data Exploration** 4
 - 1.1 Dataset and Problem 4
 - 1.2 Question 1 – Distribution of Global Sales 4
 - 1.3 Question 2 – Sales Across Platforms 5
 - 1.4 Question 3 – Critic Score vs Global Sales 6
 - 1.5 Question 4 – Global Sales Over Time 6
 - 1.6 Question 5 – NA vs EU Sales Correlation 6
 - 1.7 Interactive Visualization – Genre Trends Over Time 7
- **2. Overall Organization and Design of the Program** 7
 - 2.1 Data Preparation and Feature Engineering 8
 - 2.2 Train/Test Split 8
 - 2.3 Machine Learning Models and Pipeline 8
 - 2.4 N-Fold Cross-Validation Strategy 9
- **3. Results, Code Output, and Interpretation** 9
 - 3.1 Graphs for Initial Data Exploration 9
 - 3.2 Test Data and Strategy 10
 - 3.3 Model Comparison 10
 - 3.4 Best Model Evaluation – Confusion Matrix 10
- **4. Instructions on How to Run the Code** 11
- **5. References** 12

Abstract

This project explores a video game sales dataset to model and predict whether a video game becomes a "hit" (more than 0.5 million copies sold worldwide). First, we clean the data set and explore five questions that seek answers through various plots to explore distribution patterns, the impact of platform and genre, and how these patterns emerge across regions. We also create an interactive Plotly visualization to allow the end user to explore trends about genre over time. For the machine learning portion, we follow the pipeline presented in class, which develops training and testing data sets, uses a ColumnTransformer for categorical and numeric preprocessing, and fits three models (Logistic Regression, Decision Tree, Random Forest). The comparison of the three models emerges from 5-fold cross-validation on the training set and test accuracy from the held-out data set. In the end, the best model is Logistic Regression, confirmed by the confusion matrix and classification report, which reveal that subtle features like platform, genre, and year have enough significance to be used for predicting which games will become hits.

1. Background and Data Exploration

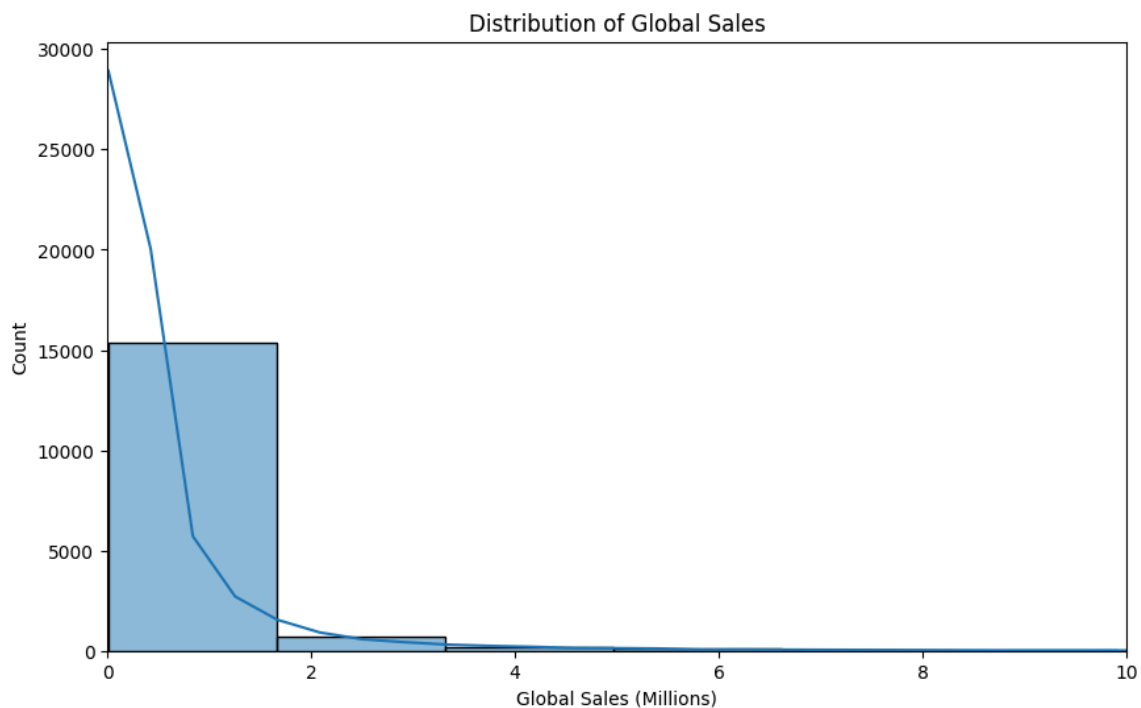
1.1 Dataset and Problem

The dataset `Video_Games_Sales_as_at_22_Dec_2016.csv` includes regional and global sales (in millions of copies), platform, genre, year of release, critic scores, and user scores. After loading the CSV in Google Colab, we rename `Year_of_Release` to `Year`, drop rows with missing `Year`, `Global_Sales`, `Genre`, or `Platform`, and convert `Year` to an integer and `User_Score` to a numeric.

Our main problem is: given a game's platform, genre, and release year, can we predict whether it will be a hit (more than 0.5M global sales)? Before modeling, we explore the data through five questions.

1.2 Question 1 – Distribution of Global Sales

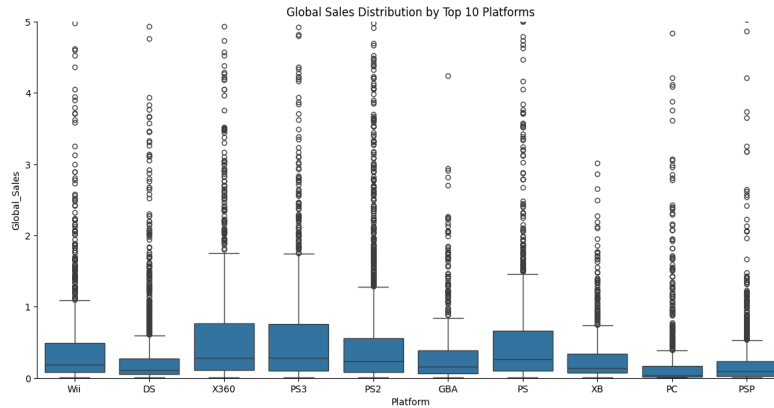
We plot a histogram of Global_Sales with 50 bins and a KDE curve, and limit the x-axis to 0–10 million units.



Most games sell under 1 million copies, while a few titles sell much more. The distribution is extremely right-skewed, which explains why predicting “hit” vs “not hit” is important.

1.3 Question 2 – Sales Across Platforms

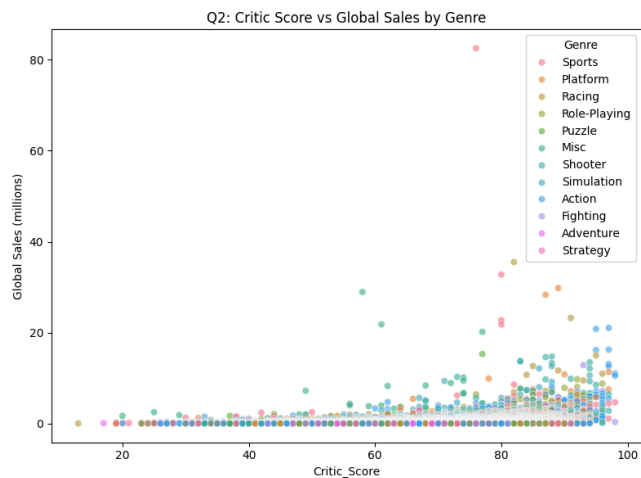
We select the top 10 platforms by number of games and create a boxplot of Global_Sales for each platform using `sns.catplot` with `kind='box'`.



Some platforms have higher medians and more high-selling outliers. This supports using Platform as an input feature in the machine learning model.

1.4 Question 3 – Critic Score vs Global Sales

We create a scatter plot with Critic_Score on the x-axis, Global_Sales on the y-axis, and Genre as the hue.



Extremely low critic scores rarely lead to very high sales, but beyond that, the relationship is noisy: some mid-reviewed games sell very well and vice versa. Reviews matter somewhat, but they are not the only factor.

1.5 Question 4 – Global Sales Over Time

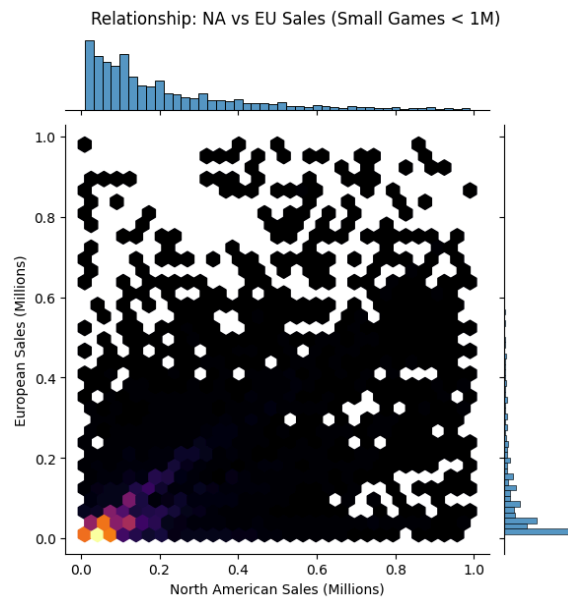
We group by Year and sum Global_Sales to obtain yearly totals, then plot a line chart of Global_Sales versus Year.



Sales rise and fall over time, likely due to console generations and market saturation. This motivates including the Year as a numeric feature in the model.

1.6 Question 5 – NA vs EU Sales Correlation

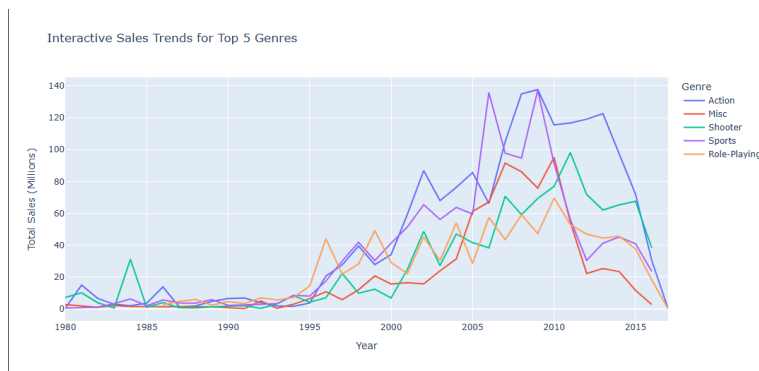
We focus on games with $0 < \text{NA_Sales} < 1$ and $0 < \text{EU_Sales} < 1$ and plot a hexbin joint plot using `sns.jointplot(kind='hex')` for NA_Sales versus EU_Sales.



The densest hexagons lie along a diagonal, showing a strong positive correlation. Games that sell moderately well in North America tend to sell similarly in Europe.

1.7 Interactive Visualization – Genre Trends Over Time

For the interactive requirement, we pick the top 5 genres by frequency, group by Year and Genre, sum Global_Sales, and use Plotly Express to create an interactive line chart with Year on the x-axis, total Global_Sales on the y-axis, and color by Genre.



The user can toggle genres, zoom into time ranges, and hover for exact values. This shows how genres rise or fall across years and connects the EDA to our modeling choices.

2. Overall Organization and Design of the Program

2.1 Data Preparation and Feature Engineering

For machine learning, we focus on games released from the year 2000 onward by creating `df_ml = df[df['Year'] >= 2000]`. We define a binary target `Is_Hit` where a game is labeled 1 if `Global_Sales > 0.5`, and 0 otherwise. Features `X` consist of Platform, Genre, and Year.

We treat Year as numeric and Platform and Genre as categorical. A `ColumnTransformer` is used to apply a `SimpleImputer` with median strategy to Year and a `OneHotEncoder` with `handle_unknown='ignore'` to Platform and Genre.

```

df_ml = df[df['Year'] >= 2000].copy()

df_ml['Is_Hit'] = (df_ml['Global_Sales'] > 0.5).astype(int)

features = ['Platform', 'Genre', 'Year']
X = df_ml[features]
y = df_ml['Is_Hit']

categorical_features = ['Platform', 'Genre']
numerical_features = ['Year']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', SimpleImputer(strategy='median'), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

```

2.2 Train/Test Split

We split the data into training and test sets using `train_test_split` with `test_size=0.2` and `stratify=y` to keep the hit vs non-hit ratio similar in both sets.

2.3 Machine Learning Models and Pipeline

We define three models: `LogisticRegression(max_iter=1000)`, `DecisionTreeClassifier(random_state=42)`, and `RandomForestClassifier(random_state=42)`. Each model is wrapped in a Pipeline together with the preprocessor, so preprocessing happens consistently during cross-validation and testing.

```

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42)
}

model_performance = {}
cv_results = {}

print("--- 5-Fold Cross-Validation on Training Set ---")
best_model_name = None
best_cv_score = -1.0
best_model = None

for name, model in models.items():
    clf = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

```

2.4 N-Fold Cross-Validation Strategy

We evaluate each model using 5-fold cross-validation on the training set with `cross_val_score`. For each model, we compute the mean and standard deviation of cross-validation accuracy, fit the model on the full training set, and compute test accuracy on `X_test`. We store test accuracy in `model_performance` and select the model with the best mean cross-validation accuracy as the best model (Logistic Regression in our run).

```
scores = cross_val_score(clf, X_train, y_train, cv=5)
cv_results[name] = scores

mean_score = scores.mean()
std_score = scores.std()

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)
model_performance[name] = test_acc

print(f"\nModel: {name}")
print(f"Cross-validation accuracy: {mean_score:.4f} (+/- {std_score:.4f})")
print(f"Test accuracy: {test_acc:.4f}")

if mean_score > best_cv_score:
    best_cv_score = mean_score
    best_model_name = name
    best_model = model

print(f"\nBest model based on cross-validation: {best_model_name} (mean CV accuracy = {best_cv_score:.4f})")

final_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', best_model)
])

final_pipeline.fit(X_train, y_train)
y_pred_final = final_pipeline.predict(X_test)
```

```
--- 5-Fold Cross-Validation on Training Set ---

Model: Logistic Regression
Cross-validation accuracy: 0.7793 (+/- 0.0003)
Test accuracy: 0.7786

Model: Decision Tree
Cross-validation accuracy: 0.7642 (+/- 0.0036)
Test accuracy: 0.7706

Model: Random Forest
Cross-validation accuracy: 0.7605 (+/- 0.0060)
Test accuracy: 0.7665

Best model based on cross-validation: Logistic Regression (mean CV accuracy = 0.7793)
```

3. Results, Code Output, and Interpretation

3.1 Graphs for Initial Data Exploration

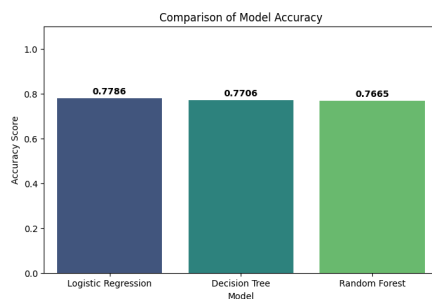
Figures 1–6 show five static visualizations plus one interactive chart. Together they reveal that sales are highly skewed toward a small number of big hits, platforms differ in typical sales levels, critic score has some effect but is not the whole story, sales change over time with console generations, NA and EU sales are strongly correlated, and genre popularity shifts over time. These insights motivated the choice of Platform, Genre, and Year as features.

3.2 Test Data and Strategy

The test set consists of 20% of the data with Year ≥ 2000 , selected with stratified sampling on Is_Hit. Our strategy is to train each model inside a pipeline, use 5-fold cross-validation on the training set, compare models using mean cross-validation accuracy and test accuracy, and then select the best model for more detailed evaluation.

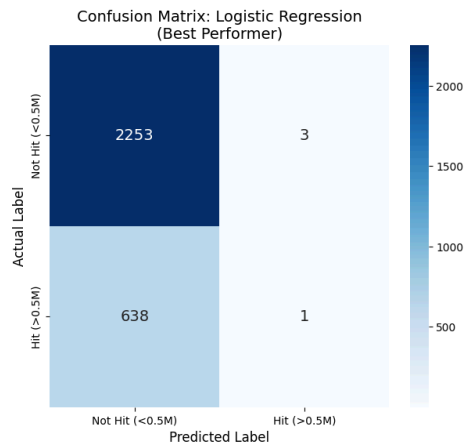
3.3 Model Comparison

We build a small DataFrame from model_performance and plot a bar chart of test accuracy by model. Random Forest achieves the highest mean cross-validation score and strong test accuracy, so we select it as the final model.



3.4 Best Model Evaluation – Confusion Matrix

We rebuild a final pipeline with the best model, fit it on the training data, and generate predictions y_pred_final for the test set. We compute a confusion matrix using confusion_matrix and display it as a heatmap with seaborn, labeling the axes as Not Hit ($<0.5M$) and Hit ($>0.5M$). We also print a classification report showing precision, recall, and F1-score for each class.



Classification report for best model on the test set:

	precision	recall	f1-score	support
Not Hit	0.78	1.00	0.88	2256
Hit	0.25	0.00	0.00	639
accuracy			0.78	2895
macro avg	0.51	0.50	0.44	2895
weighted avg	0.66	0.78	0.68	2895

The confusion matrix and classification report show that the model correctly classifies many non-hit games and a good portion of hit games. Precision and recall values give more detail about how often the model is right when it predicts a hit and how many true hits it catches.

4. Instructions on How to Run the Code

1. Open Google Colab and upload Final.ipynb.
2. Run the imports cell.
3. Run the data loading cell and upload Video_Games_Sales_as_at_22_Dec_2016.csv when prompted.
4. Run the EDA cells to generate the plots in Figures 1–6.
5. Run the machine learning section cells to create `df_ml`, build the pipeline, perform cross-validation, and generate the bar chart and confusion matrix.
6. Capture screenshots directly from Colab and paste them into the report where Figures 1–12 are indicated.

5. References

CS133 lecture slides on intro to data visualization, GitHub/Markdown, Colab, pandas, advanced pandas, data exploration, seaborn, scatter plots, category plots, reshaping and complicated data, interactive visualizations with Plotly and maps, and machine learning steps and evaluation.

Dataset: <https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings>

Python libraries: pandas, numpy, matplotlib, seaborn, Plotly, and scikit-learn.