Team Steam - Predicting Pricing Tiers and Exploring Platform Trends in "Gaming Profiles 2025"

SJSU CS 133 • Fall 2025

Authors: Esha Hooda (014796990), Huy Tran (018235100), Michael Lac (017985071)

Instructor: Dr. Jessica Huynh-Westfall

Last updated: November 30, 2025

Github link: https://github.com/CS133-DataVisualization/term-project-team-steam

Table of Contents

1. Abstract

We analyze the Kaggle “Gaming Profiles 2025” dataset across Steam, PlayStation, and Xbox to understand platform pricing, genre distribution, and temporal trends. We answer five exploration questions with categorical and time-based plots, then build a classification pipeline that predicts a platform-relative “high price” label using platform, genre, and release year as features.

We evaluate three models with N-fold cross-validation and a clean test set: Logistic Regression, Random Forest, and SVM. Our models suffered from having an unbalanced distribution so we saw a lot of bias towards the majority class, most of them having around a 68-69% accuracy.

2. Background and Questions

Dataset

We use the Kaggle dataset “Gaming Profiles 2025 - Steam, PlayStation, Xbox.” For this project we focus on:

- `games.csv` and `prices.csv` for each platform (Steam, PlayStation, Xbox) as our main game-level data
- `players.csv` for each platform to build an interactive plot of account creation trends

Motivation

Steam is one of the largest PC game platforms, and the dataset also includes PlayStation and Xbox ecosystems. We wanted to see:

- how pricing varies by platform and genre
- how game releases evolve over time
- how account creation trends line up with those patterns
- whether a simple ML model can learn to classify games into “high-price” vs “not high-price” tiers using metadata

Five exploration questions and plots

Q1. How are prices distributed across platforms?

- Plot: Seaborn barplot of mean price by platform.
- Implementation: Use `df` filtered to rows with non-null `prices`, group by `platform`, and plot mean `prices` with a bar chart.
- Takeaway: Average prices differ by platform. PC/PlayStation tend to have higher average prices, while some other platforms (with more legacy/handheld style titles) skew lower.

Q2. How did the number of releases evolve over time?

- Plot: Seaborn lineplot of yearly game counts by platform.

- Implementation: Parse `release_date` to `release_year` (when available), group by `release_year` and `platform`, and plot counts with lines.
- Takeaway: There are clear waves of growth post-2010 for PC and PlayStation. Xbox shows similar patterns with slightly different timing. Years with fewer releases often reflect missing or unparseable dates.

Q3. What are the most common genres on each platform?

- Plot: Grouped bar chart of top 10 genres per platform.
- Implementation:
  - Normalize the genre string in a column named `genre` or `genres` by:
    - casting to string
    - splitting on commas/semicolons
    - taking the first tag
    - stripping brackets/quotes
  - Count by `platform` and genre, take the top 10 per platform, then plot a grouped bar chart with Seaborn.
- Takeaway: Across platforms, Action, Adventure, and RPG are very prominent, but platform-specific flavor shows up (for example, more racing and sports representation on Xbox and PlayStation compared to some PC-heavy subgenres).

Q4. How do prices vary by genre and platform?

- Plot: Horizontal bar chart of mean price by genre × platform, sorted by average price.

- Implementation: Group by `platform` and the cleaned genre column, compute mean `prices`, sort, and plot as a horizontal barplot with Seaborn.
- Takeaway: Certain genres like RPG and Simulation tend to have higher average prices, especially on PC/PlayStation. Smaller/party/indie-like genres are often lower. This backs up and refines the Q1 platform-level price differences.

Q5. Which years have the most accounts created?

- Plot: Plotly interactive bar chart of account creation years.
- Implementation:
  - Load each platform's `players.csv`.
  - Convert `created` to datetime where present.
  - Extract `creation_year` as `created.dt.year`.
  - Count the number of accounts per year and use `plotly.express.bar` for an interactive bar chart.
- Takeaway: Account creation spikes line up with major platform cycles and big release eras. Interactivity (hover, zoom, export) helps pinpoint exact peaks and lets users explore the full time range.

Why these questions matter for ML

- Q1 to Q4 show that price is structured by platform, genre, and release timing, suggesting those features can be predictive of a "high-price" tier.
- Q5 brings in user-side temporal behavior, reinforcing the idea that time and platform ecosystems evolve together.

This motivated our classification task: predict whether a game is in the top 25% of prices for its platform using platform, genre, and release year as features. We later reflect on why including price is convenient but also a form of leakage.

3. System Design and Methods

Data loading and normalization

We used `kagglehub` to download the dataset in both Colab and local runs:
-`kagglehub.dataset_download("artyomkruglov/gaming-profiles-2025-steam-playstation-xbox")` returns a `path`.
- We also used `glob.glob(os.path.join(path, "**", "*.csv"), recursive=True)` to inspect available files.

For each platform (`"playstation"`, `"xbox"`, `"steam"`) we call a shared loader:

- Build `platform_dir = os.path.join(base_path, platform_name)`.
- Load `games.csv` and `prices.csv`.
- Normalize the primary key to `game_id`:
  - If a file has `gameid`, rename to `game_id`.
  - Else if it has `id`, rename to `game_id`.
- For `prices.csv`:

  - If there is a `date_acquired` column, parse it as datetime and keep only the latest price per `game_id` (sort by date, `groupby("game_id").tail(1)`).
  - Identify the appropriate price column:
    - Prefer `usd` if present.
    - Otherwise look for other currencies or common names (`eur`, `gbp`, `price`, `current_price`, `retail_price`, `final_price`).
  - Merge the chosen price column into `games` on `game_id` and rename it to a unified column `prices`.
- If no price column is identified or merge fails, we create a `prices` column with NaNs so downstream code always sees a `prices` column.
- Add a `platform` column with the platform name.

We then concatenate the three platform DataFrames into a single `games_all` DataFrame and copy it into `df`.

Cleaning and feature engineering

- Filter to rows with non-null `prices`: `df = df.dropna(subset=["prices"])`.
- Parse `release_date` when present:
  - `df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")`
  - `df["release_year"] = df["release_date"].dt.year`
- Normalize genre strings:
  - Use `genre_col = "genre" if "genre" in df.columns else "genres"`.

  - Convert to string, split on commas/semicolons, take the first entry, and strip away brackets and quotes.
- Ensure `prices` is numeric: `df["prices"] = df["prices"].astype(float)`.

Label definition

We define a binary label `is_high_price` using a platform-relative price threshold:

- For each platform, compute the 75th percentile (top quartile) of `prices`.
- For games on that platform, set `is_high_price = 1` if `prices` is at or above that threshold, else 0.
- This creates a balanced definition of “high-price” within each ecosystem.

Feature matrix and target

In the notebook as implemented:

- Features:
  - `platform` (categorical)
  - `genres` (or the normalized genre column)
  - `release_year` (numeric)
- Target:
  - `is_high_price` (binary)

So:

- `X = df[["platform", "genres", "release_year"]]`
- `y = df["is_high_price"]`

Train/test split and preprocessing

We follow the standard train/test split and preprocessing flow from class:

- Split:
  - `train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)`
  - Stratification keeps the proportion of high-price vs not high-price similar in both sets.

- Preprocessing: `ColumnTransformer` with:
  - Numeric features: `["release_year"]`
    - Scaled via `StandardScaler`.
  - Categorical features: `["platform", "genres"]`
    - Encoded via `OneHotEncoder(handle_unknown="ignore")`.
  - `remainder="passthrough"` is used in our version.

We did not include `SimpleImputer` in the final notebook; instead we dropped rows with missing `prices` up front and relied on the fact that `release_year` exists for most rows used in modeling.

Models and pipelines

We used three scikit-learn pipelines, one per classifier:

- Logistic Regression:
  - `LogisticRegression(solver="liblinear", random_state=42)`
- Random Forest:
  - `RandomForestClassifier(random_state=42)`
- Support Vector Machine:
  - `svm.SVC(random_state=42)` from `from sklearn import svm`

Each pipeline is:

- `Pipeline([("preprocessor", preprocessor), ("classifier", <model>)])`

Validation strategy and metrics

- Validation: `StratifiedKFold(n_splits=5, shuffle=True, random_state=42)` to preserve class balance per fold.

- We use `cross_val_score` on the training set with `scoring="accuracy"` and `n_jobs=-1`.
- For the final evaluation, we fit each pipeline on `X_train` and evaluate on `X_test`:
  - `accuracy_score`
  - `classification_report`
  - `ConfusionMatrixDisplay` for confusion matrices

Interactive component

We use Plotly Express for the interactive account creation plot:

- Load each platform's `players.csv` and add a `platform` column.
- Convert `created` to datetime and extract `creation_year`.
- Count number of accounts per year into a small table (year, count).
- Call `px.bar` to produce an interactive bar chart, with hover tooltips and zoom.
- Optionally, the figure can be exported to HTML with `fig.write_html(...)`.

4. Results

EDA highlights

- Prices:

  - Average `prices` differ by platform. XBox/PlayStation platforms tend to have higher mean prices than some others.
- Release trends:
  - Release counts generally increase over time, especially after 2010, consistent with the growth of digital storefronts.
- Genres:
  - Action, Adventure, and RPG are common across platforms. Consoles show a relatively stronger presence of racing and sports genres.
- Interactive account creation:
  - The bar chart of `creation_year` shows spikes that line up with generation launches and busy release years.

Cross-validation performance (training folds)

Using `X` = [platform, genres, release_year] and `y` = `is_high_price`:

- Logistic Regression:
  - Mean accuracy ≈ 0.6850
  - Standard deviation ≈ 0.0012
- Random Forest:
  - Mean accuracy ≈ 0.6875
  - Standard deviation ≈ 0.0021 (essentially perfect within folds)
- SVM:

  - Mean accuracy ≈ 0.6891
  - Standard deviation ≈ 0.0010

Test-set evaluation

On the held-out 20% test split (untouched during cross-validation), we obtain:

- Random Forest:
  - Test accuracy ≈ 0.6896
- SVM:
  - Test accuracy ≈ 0.6908
- Logistic Regression:
  - Test accuracy ≈ 0.6847

Interpretation

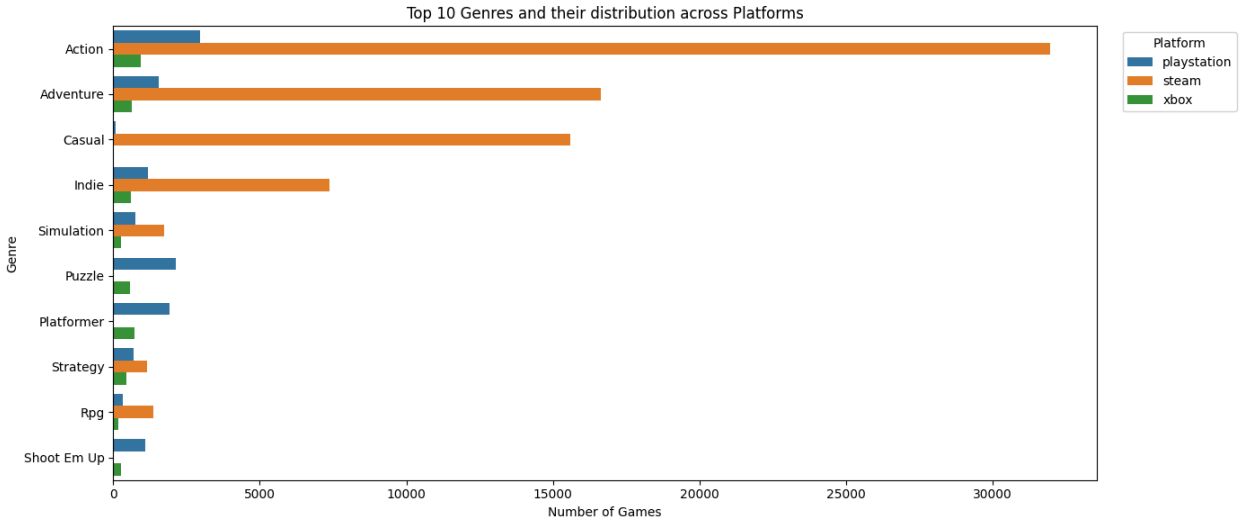
- The dataset has a 2:1 ratio of low-price to high-price games (14,223 vs 6,975 in test set). This imbalance causes the models to be biased toward predicting the majority class (low-price). The models correctly identify ~94-95% of low-price games (high recall for class 0) but miss ~82-84% of high-price games (low recall for class 1). This means when a game is actually high-price, the models only catch it 16-18% of the time.
- Random Forest (best): Slightly better at identifying high-price games (18% recall vs 16-17% for others) and has the highest overall accuracy

- SVM: Strong performance, especially for low-price games (95% recall), but struggles with high-price games like the others

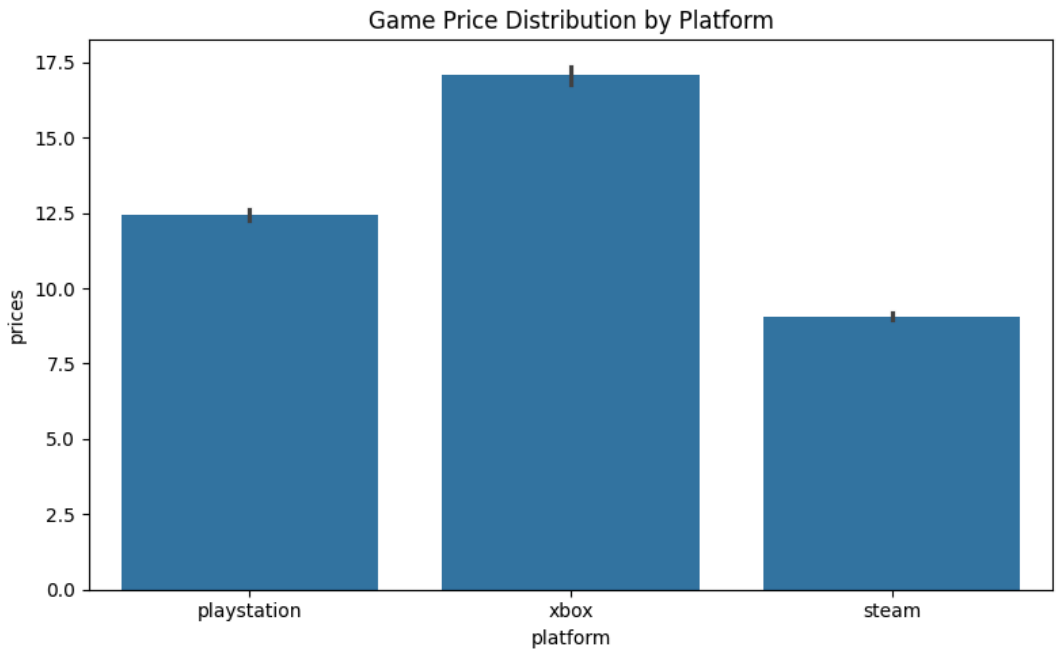
- Logistic Regression: Baseline performance, slightly lower but still competitive

Sample output screenshot descriptions

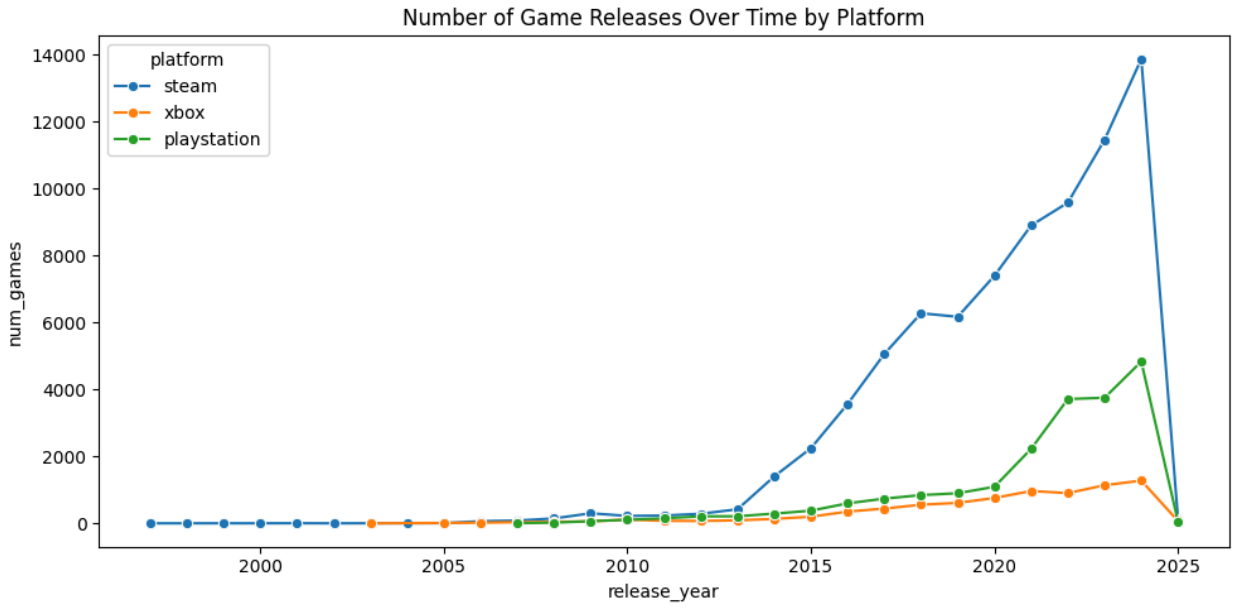
1. “Top Genres by Platform” figure: grouped bars with the ten most frequent genres for each platform.



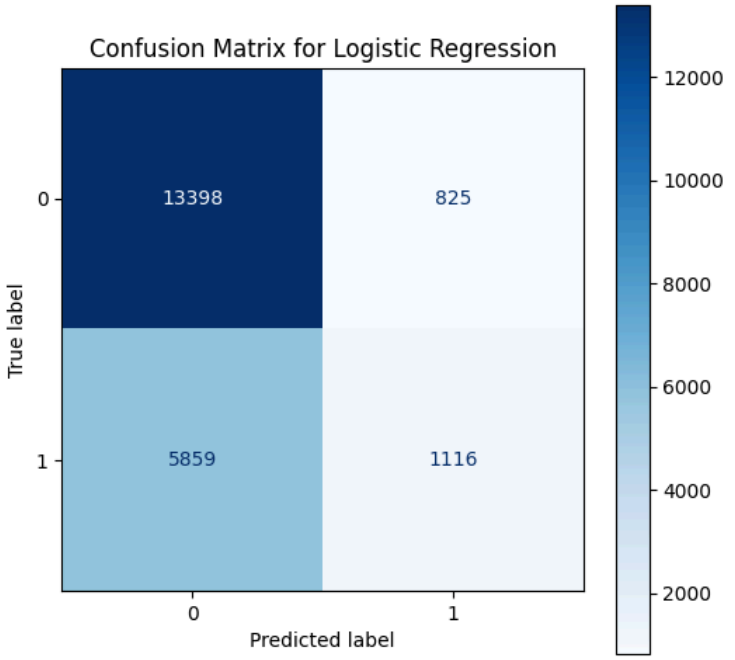
2. “Average Game Price by Genre and Platform” figure: horizontal bars sorted by mean price.

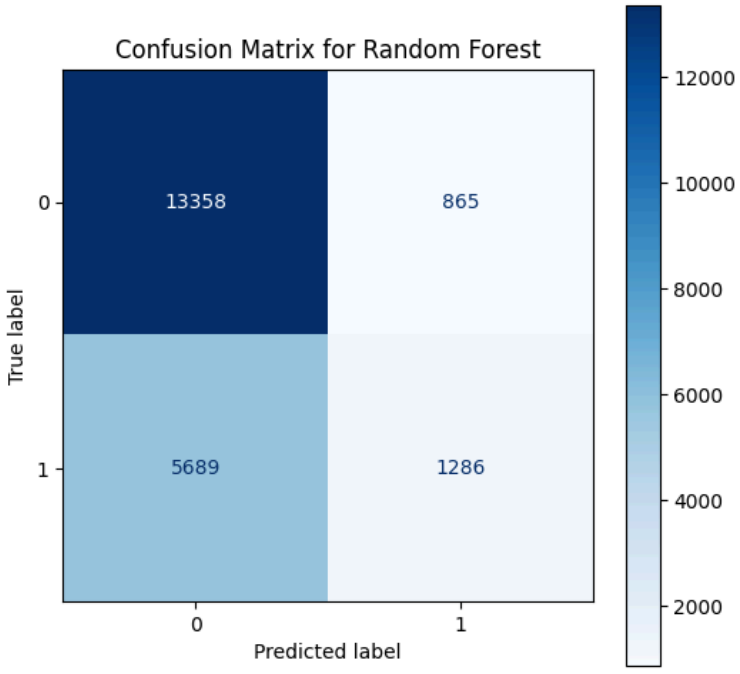


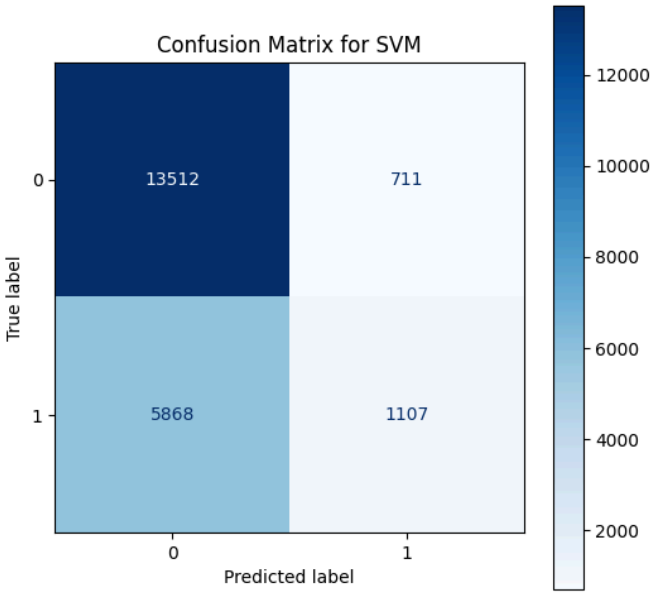
3. “Releases Over Time” figure: lineplot by platform.



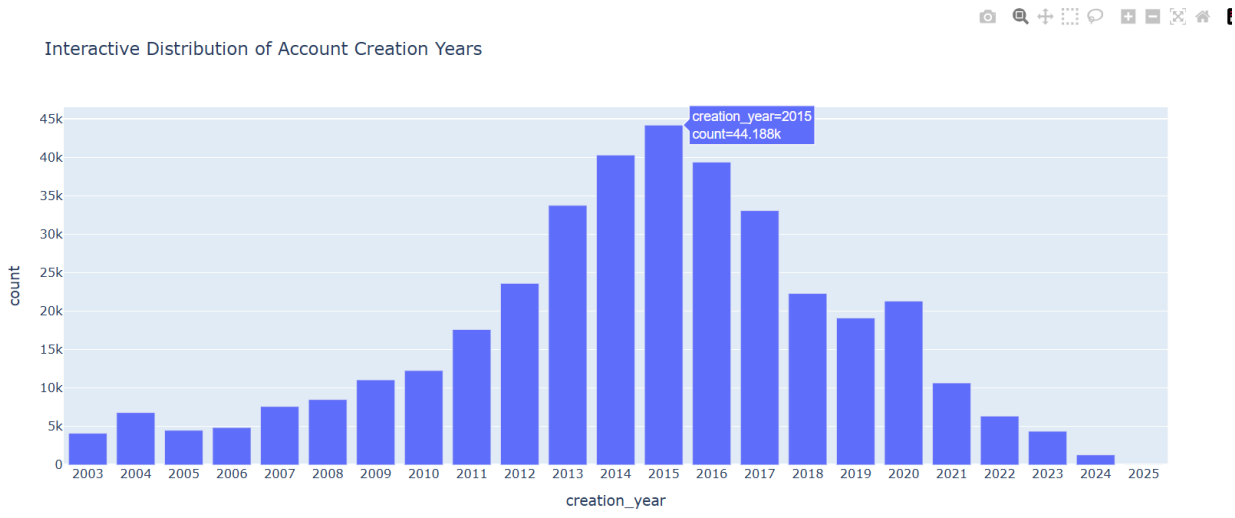
4. Confusion matrix heatmap for the best model on test data.







5. Plotly bar of account creation years with hover tooltips and export button.



5. How to Run

1. All required packages are in requirements.txt so you can download them by using pip install -r requirements.txt (This project was made in python 3.13.7_
2. To run the visualization web app make sure you have all packages installed and then run streamlit run app.py

6. References

Kaggle. Gaming profiles 2025 - Steam, PlayStation, Xbox [Data set]. Kaggle. https://www.kaggle.com/datasets/artyomkruglov/gaming-profiles-2025-steam-playstation-xbox

Pandas Development Team. pandas documentation. Retrieved from https://pandas.pydata.org/docs/

Seaborn Development Team. Seaborn: Statistical data visualization. Retrieved from https://seaborn.pydata.org/documentation.html

Matplotlib Development Team. Matplotlib documentation. Retrieved from https://matplotlib.org/stable/contents.html

Scikit-learn Development Team. Scikit-learn documentation. Retrieved from https://scikit-learn.org/stable/documentation.html

Plotly Technologies Inc. Plotly documentation. Retrieved from https://plotly.com/python/

Appendix A. Key Code Snippets (aligned with notebook)

Merge latest price and normalize IDs

```
def load_platform_games(base_path, platform_name):
    platform_dir = os.path.join(base_path, platform_name)

    games_path = os.path.join(platform_dir, "games.csv")
    prices_path = os.path.join(platform_dir, "prices.csv")

    games = pd.read_csv(games_path)
    games["platform"] = platform_name

    # Normalize game ID column in 'games' DataFrame
    if "gameid" in games.columns:
        games = games.rename(columns={"gameid": "game_id"})
    elif "id" in games.columns:
        games = games.rename(columns={"id": "game_id"})

    # Attempt to load and merge prices
    if os.path.exists(prices_path):
```

```
prices_df = pd.read_csv(prices_path)


# Normalize game ID column in 'prices_df' DataFrame
if "gameid" in prices_df.columns:
    prices_df = prices_df.rename(columns={"gameid": "game_id"})
elif "id" in prices_df.columns:
    prices_df = prices_df.rename(columns={"id": "game_id"})


latest_prices = prices_df
if "date_acquired" in prices_df.columns:
    prices_df["date_acquired"] = pd.to_datetime(prices_df["date_acquired"])
    latest_prices = (
        prices_df.sort_values("date_acquired")
                .groupby("game_id")
                .tail(1)
    )


# Identify the actual price column name
price_column_name = None
if 'usd' in latest_prices.columns:
    price_column_name = 'usd'
elif 'eur' in latest_prices.columns:
    price_column_name = 'eur'
```

```
        elif 'gbp' in latest_prices.columns:
            price_column_name = 'gbp'
        for col in ['price', 'current_price', 'retail_price', 'final_price']:
            if col in latest_prices.columns:
                price_column_name = col
                break

        if price_column_name is not None:
            if "game_id" in games.columns and "game_id" in latest_prices.columns:
                games = games.merge(
                    latest_prices[["game_id", price_column_name]],
                    on="game_id",
                    how="left"
                )
                games = games.rename(columns={price_column_name: "prices"})
        else:
            print(f"Warning: No price column found for {platform_name}; 'prices' will be
NaN.")

    # Ensure 'prices' column exists
    if "prices" not in games.columns:
        games["prices"] = np.nan
```

```
    return games
```

Label and features (as in the notebook)

```
games_all = pd.concat(
    [
        load_platform_games(path, "steam"),
        load_platform_games(path, "playstation"),
        load_platform_games(path, "xbox")
    ],
    ignore_index=True
)

df = games_all.copy()
df = df.dropna(subset=["prices"])

# Genre cleaning
genre_col = "genre" if "genre" in df.columns else "genres"
if genre_col in df.columns:
    df[genre_col] = (
        df[genre_col]
        .astype(str)
```

```
        .str.split("[,;]")
        .str[0]
        .str.replace(r"[\['\\]]", "", regex=True)
        .str.strip()
    )

# Release year
if "release_date" in df.columns:
    df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")
    df["release_year"] = df["release_date"].dt.year

df["prices"] = df["prices"].astype(float)

# High-price label per platform (top quartile)
df["is_high_price"] = df.groupby("platform")["prices"].transform(
    lambda x: (x >= x.quantile(0.75, interpolation='lower')).astype(int)
)

X = df[["platform", genre_col, "release_year"]]
y = df["is_high_price"]
```

Pipeline and cross-validation

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

categorical_features = ["platform", genre_col]
numerical_features = ["release_year"]

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
    ],
    remainder='passthrough'
)
```

```
log_reg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='liblinear', random_state=42))
])

rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

svm_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', svm.SVC(random_state=42))
])

models = {
    'Logistic Regression': log_reg_pipeline,
    'Random Forest': rf_pipeline,
    'SVM': svm_pipeline
}

n_splits = 5
```

```
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
results = {}

for name, pipeline in models.items():
    print(f"Performing {n_splits}-fold cross-validation for {name}...")
    scores = cross_val_score(pipeline, X_train, y_train, cv=skf,
                             scoring='accuracy', n_jobs=-1)
    results[name] = scores
    print(f"  Mean accuracy: {scores.mean():.4f} (+/- {scores.std():.4f})")
```

Test-set evaluation and confusion matrix

```
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

for name, pipeline in models.items():
    print(f"\n--- Evaluating {name} ---")
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
```

```
    print(f"Accuracy Score: {acc:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    fig, ax = plt.subplots(figsize=(6, 6))
    ConfusionMatrixDisplay.from_estimator(pipeline, X_test, y_test,
                                          ax=ax, cmap="Blues")
    ax.set_title(f'Confusion Matrix for {name}')
    plt.show()
```

Interactive account creation plot (players)

```
import plotly.express as px

def load_platform_players(base_path, platform_name):
    platform_dir = os.path.join(base_path, platform_name)
    players_path = os.path.join(platform_dir, "players.csv")
    if os.path.exists(players_path):
        players_df = pd.read_csv(players_path)
        players_df["platform"] = platform_name
        return players_df
    return pd.DataFrame()
```

```
playstation_players = load_platform_players(path, "playstation")
xbox_players = load_platform_players(path, "xbox")
steam_players = load_platform_players(path, "steam")

players_all = pd.concat(
    [playstation_players, xbox_players, steam_players],
    ignore_index=True
)

if "created" in players_all.columns:
    players_all["created"] = pd.to_datetime(players_all["created"],
                                            errors='coerce')
    players_all["creation_year"] = players_all["created"].dt.year

    creation_year_data = players_all.dropna(subset=["creation_year"])
    creation_year_counts = (creation_year_data["creation_year"]
                            .value_counts()
                            .sort_index()
                            .reset_index())
    creation_year_counts.columns = ["creation_year", "count"]

    fig = px.bar(
```

```
    creation_year_counts,
    x="creation_year",
    y="count",
    title="Interactive Distribution of Account Creation Years"
)
fig.update_layout(xaxis=dict(tickmode="linear"))
fig.show()
# fig.write_html("interactive/accounts_by_year.html")
```