# Video Game Sales Prediction - Machine Learning Lab

## Introduction and Setup

Welcome to this machine learning lab where we'll build a model to predict whether a video game will be a "hit" based on its characteristics and sales data. This notebook will guide you through the entire process, from data loading to model evaluation and optimization.

Learning objectives:

1. Learn to preprocess and explore a real-world dataset
2. Build and evaluate a decision tree classifier
3. Optimize a model through hyperparameter tuning
4. Interpret model results and feature importance

```
#install libraries if necessary

# Import the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning libraries
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_scor
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, RocCurveDisplay
```

```
# Set random seed for reproducibility
np.random.seed(42)
```

```
# Configure visualizations
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("colorblind")
```

```
# Display settings for better output
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
```

# Download the Dataset

```
# You can run this cell to download the dataset directly, or upload it manually!
import requests

url = 'https://www.kaggle.com/datasets/gregorut/videogamesales/download'
response = requests.get(url)

with open('videogamesales.zip', 'wb') as f:
        f.write(response.content)
from google.colab import files
uploaded = files.upload()  # choose vgsales-video-games.csv from your PC


print("Dataset downloaded successfully.")
```

```
Choose Files vgsales.csv
vgsales.csv(text/csv) - 1372380 bytes, last modified: 10/21/2025 - 100% done
Saving vgsales.csv to vgsales.csv
Dataset downloaded successfully.
```

# Load the Dataset

```
# Load the dataset
df = pd.read_csv('vgsales-video-games.csv')

# Let's take a look at the first few rows of the dataset
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
   Rank                      Name Platform    Year         Genre Publisher  NA_Sales
0     1                Wii Sports      Wii  2006.0        Sports  Nintendo     41.49
1     2         Super Mario Bros.      NES  1985.0      Platform  Nintendo     29.08
2     3            Mario Kart Wii      Wii  2008.0        Racing  Nintendo     15.85
3     4         Wii Sports Resort      Wii  2009.0        Sports  Nintendo     15.75
4     5  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo     11.27
```

## Importing My Kaggle datasets

```
!pip install -q kaggle
!mkdir -p ~/.kaggle
!echo '{"username":"YOUR_KAGGLE_USERNAME","key":"YOUR_KAGGLE_API_KEY"}' > ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d gregorut/videogamesales -f vgsales.csv -p . --force
```

```
Dataset URL: https://www.kaggle.com/datasets/gregorut/videogamesales
License(s): unknown
Downloading vgsales.csv to .
  0% 0.00/381k [00:00<?, ?B/s]
100% 381k/381k [00:00<00:00, 263MB/s]
```

```
# Unzip the dataset
import zipfile

with zipfile.ZipFile('videogamesales.zip', 'r') as zip_ref:
    zip_ref.extractall('.')

print("Dataset unzipped successfully.")
```

```
---------------------------------------------------------------------------
BadZipFile                                Traceback (most recent call last)
/tmp/ipython-input-3411850770.py in <cell line: 0>()
      2 import zipfile
      3
----> 4 with zipfile.ZipFile('videogamesales.zip', 'r') as zip_ref:
      5     zip_ref.extractall('.')
      6

                                 1 frames
/usr/lib/python3.12/zipfile/__init__.py in _RealGetContents(self)
   1435             raise BadZipFile("File is not a zip file")
   1436         if not endrec:
-> 1437             raise BadZipFile("File is not a zip file")
   1438         if self.debug > 1:
   1439             print(endrec)

BadZipFile: File is not a zip file
```

Next steps: Explain error

# Dataset Information

```
# Get basic information about the dataset
print("\nDataset basic information:")
print(df.info())

# Get descriptive statistics
print("\nDescriptive statistics:")
print(df.describe())
```

```
Dataset basic information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
```

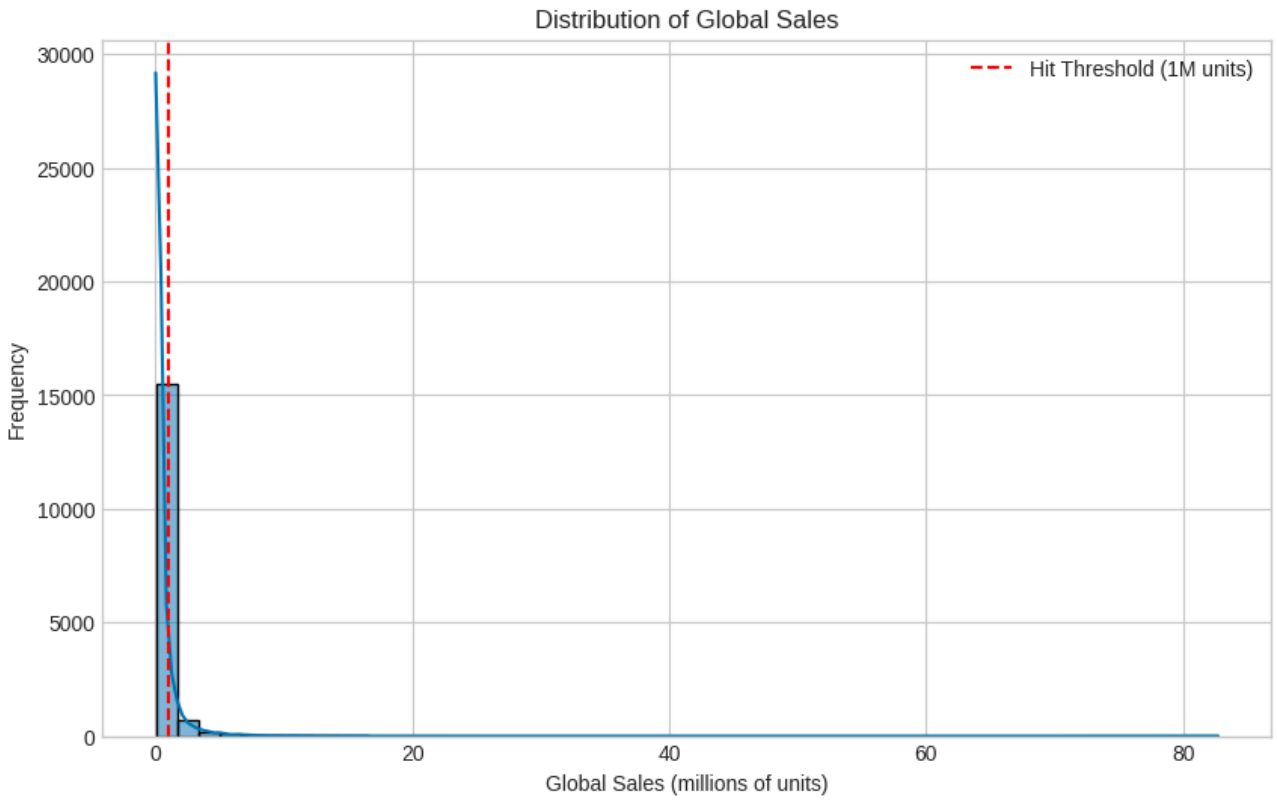
```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Rank          16598 non-null  int64
 1   Name          16598 non-null  object
 2   Platform      16598 non-null  object
 3   Year          16327 non-null  float64
 4   Genre         16598 non-null  object
 5   Publisher     16540 non-null  object
 6   NA_Sales      16598 non-null  float64
 7   EU_Sales      16598 non-null  float64
 8   JP_Sales      16598 non-null  float64
 9   Other_Sales   16598 non-null  float64
 10  Global_Sales  16598 non-null  float64
dtypes: float64(6), int64(1), object(4)
memory usage: 1.4+ MB
None

Descriptive statistics:
               Rank          Year      NA_Sales      EU_Sales      JP_Sales   Other_
count  16598.000000  16327.000000  16598.000000  16598.000000  16598.000000  16598.0
mean    8300.605254   2006.406443      0.264667      0.146652      0.077782      0.0
std     4791.853933      5.828981      0.816683      0.505351      0.309291      0.1
min        1.000000   1980.000000      0.000000      0.000000      0.000000      0.0
25%     4151.250000   2003.000000      0.000000      0.000000      0.000000      0.0
50%     8300.500000   2007.000000      0.080000      0.020000      0.000000      0.0
75%    12449.750000   2010.000000      0.240000      0.110000      0.040000      0.0
max    16600.000000   2020.000000     41.490000     29.020000     10.220000     10.5
```

```
# Cell 5:Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
Rank              0
Name              0
Platform          0
Year            271
Genre             0
Publisher        58
NA_Sales          0
EU_Sales          0
JP_Sales          0
Other_Sales       0
Global_Sales      0
dtype: int64
```

```
# Cell 6: Data Visualization - Global Sales Distribution
# ======================================================
# Visualize the distribution of global sales
plt.figure(figsize=(10, 6))
sns.histplot(df['Global_Sales'], bins=50, kde=True)
plt.title('Distribution of Global Sales')
plt.xlabel('Global Sales (millions of units)')
```

```
plt.ylabel('Frequency')
plt.axvline(x=1, color='red', linestyle='--', label='Hit Threshold (1M units)')
plt.legend()
plt.show()
```




```
# Cell 7: Create Target Variable
# ============================
# TASK: Create a binary target variable for "hit" games
# A game is considered a hit if it sold more than 1 million units (Global_Sales > 1)
# YOUR CODE HERE
df['Hit'] = (df['Global_Sales'] > 1).astype(int)
```

```
# Cell 8: Analyze Target Distribution
# ================================
# Let's see the proportion of hits in our dataset
# YOUR CODE HERE
print("\nProportion of hits in the dataset:")
print(df['Hit'].value_counts(normalize=True).rename({0: 'Not Hit', 1: 'Hit'}).mul(100).round
```

```
Proportion of hits in the dataset:
Hit
Not Hit    87.63
Hit        12.37
Name: proportion, dtype: float64
```

```
# Cell 9: Drop Non-Informative Columns
```

```
# Cell 9: Drop Non-Informative Columns
# ==================================
# TASK: Drop non-informative columns
# Think about which columns won't help with prediction
# YOUR CODE HERE
df_clean = df.drop(['Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher'], axis=1)
```

```
import pandas as pd

# work on a copy so df stays unchanged for later steps
df_clean = df.copy()

# quick look at missing values
print("Before:", df_clean.shape)
print(df_clean.isna().sum().sort_values(ascending=False).head(10))


# Cell 10: Missing Value Analysis
# ==============================
# Examine the 'Year' column which might have missing values
```

```
Before: (16598, 12)
Year           271
Publisher       58
Name             0
Rank             0
Platform         0
Genre            0
NA_Sales         0
EU_Sales         0
JP_Sales         0
Other_Sales      0
dtype: int64
```

```
# Cell 11: Handle Missing Values
# ============================
# TASK: Handle missing values
# Option 1: Drop rows with missing values
# YOUR CODE HERE

# Option 2: Fill missing values with median or mean
# YOUR CODE HERE

# === Cell 11 — Option 2: Fill missing values (Median + Mean) ===
import pandas as pd
df_clean = df.copy()

# 0) Make sure a few columns are numeric so they can be imputed
for col in ['Year', 'User_Score', 'Critic_Score']:
    if col in df_clean.columns:
        df_clean[col] = pd.to_numeric(df_clean[col], errors='coerce')

# 1) Filling MEAN (typically score-like columns)
mean_cols = [c for c in ['Critic_Score', 'User_Score'] if c in df_clean.columns]
```

```
# 2) All other numeric columns get MEDIAN
numeric_cols = df_clean.select_dtypes(include='number').columns.tolist()
median_cols = [c for c in numeric_cols if c not in mean_cols]

# 3) Impute
if median_cols:
    df_clean[median_cols] = df_clean[median_cols].fillna(df_clean[median_cols].median())
if mean_cols:
    df_clean[mean_cols] = df_clean[mean_cols].fillna(df_clean[mean_cols].mean())

# 4) Categorical columns -> simple label
categorical_cols = df_clean.select_dtypes(exclude='number').columns
if len(categorical_cols):
    df_clean[categorical_cols] = df_clean[categorical_cols].fillna('Unknown')

# 5) Checks
print("Imputation done.")
print("Remaining missing values:", int(df_clean.isna().sum().sum()))
print("Shape:", df_clean.shape)
df_clean.head()

# df_clean['Year'] = df_clean['Year'].fillna(df_clean['Year'].median())
```

```
Imputation done.
Remaining missing values: 0
Shape: (16598, 12)
```

| | Rank | Name | Platform | Year | Genre | Publisher | NA_Sales | EU_Sales | JP_Sa |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Wii Sports | Wii | 2006.0 | Sports | Nintendo | 41.49 | 29.02 | 3 |
| **1** | 2 | Super Mario Bros. | NES | 1985.0 | Platform | Nintendo | 29.08 | 3.58 | 6 |
| **2** | 3 | Mario Kart Wii | Wii | 2008.0 | Racing | Nintendo | 15.85 | 12.88 | 3 |
| **3** | 4 | Wii Sports Resort | Wii | 2009.0 | Sports | Nintendo | 15.75 | 11.01 | 3 |
| **4** | 5 | Pokemon Red/Pokemon Blue | GB | 1996.0 | Role-Playing | Nintendo | 11.27 | 8.89 | 10 |

```
# Cell 12: Categorical Variable Analysis
# ==================================
# Let's identify categorical columns
categorical_columns = df_clean.select_dtypes(include=['object']).columns.tolist()
print("\nCategorical columns:", categorical_columns)
```

```
Categorical columns: ['Name', 'Platform', 'Genre', 'Publisher']
```

```
# Cell 13: Encode Categorical Variables
# ==================================
# TASK: Encode categorical variables using LabelEncoder
# Label Encoder transforms categorical variables into numerical ones
# YOUR CODE HERE
label_encoder = LabelEncoder()
for col in categorical_columns:
    df_clean[col] = label_encoder.fit_transform(df_clean[col])
```

```
# Cell 14: Feature Engineering (Optional)
# ====================================
# BONUS TASK: Feature Engineering
# Creating new features might improve model performance
# Example: Total regional sales besides global
# YOUR CODE HERE
df_clean['Total_Regional_Sales'] = df_clean[['NA_Sales', 'EU_Sales', 'JP_Sales']].sum(axis=1)
```

```
# Cell 15: Explore Processed Dataset
# ================================
# Let's look at the processed dataset
# YOUR CODE HERE
print("\nFirst 5 rows of the processed dataset:")
print(df_clean.head())
```

```
First 5 rows of the processed dataset:
   Rank   Name  Platform    Year  Genre  Publisher  NA_Sales  EU_Sales  JP_Sales  Ot
0     1  11007        26  2006.0     10        359     41.49     29.02      3.77
1     2   9327        11  1985.0      4        359     29.08      3.58      6.81
2     3   5573        26  2008.0      6        359     15.85     12.88      3.79
3     4  11009        26  2009.0     10        359     15.75     11.01      3.28
4     5   7346         5  1996.0      7        359     11.27      8.89     10.22
```

```
# Cell 16: Split Features and Target
# ================================
# TASK: Split the data into features (X) and target (y)
# YOUR CODE HERE
X = df_clean.drop('Hit', axis=1)
y = df_clean['Hit']
```

```
# Cell 17: Train-Test Split
# ======================
# TASK: Split the data into training and testing sets (80/20 split)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Print the shapes to confirm the split
```

```
# Cell 18: Train Initial Model
# ==========================
# TASK: Train a Decision Tree classifier with default parameters
# YOUR CODE HERE

# Use the dataframe you ended with earlier (change df_fe -> df_enc or df_clean if that's your las
data = df_clean.copy()

# Ensure target exists
assert 'Hit' in data.columns, "Run the cell that creates 'Hit' first (Cell 7)."

# Avoid leakage: drop columns used to define the target or identifiers
drop_cols = [c for c in ['Hit', 'Global_Sales', 'Name'] if c in data.columns]

X = data.drop(columns=drop_cols)
y = data['Hit']

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

model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# Cell 19: Make Predictions
# ========================
# TASK: Make predictions on the test set
# YOUR CODE HERE
y_pred = model.predict(X_test)
                            # Probability of being a hit
```

```
# Cell 20: Calculate Evaluation Metrics
# ====================================
# TASK: Calculate evaluation metrics
# YOUR CODE HERE
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
# Cell 21: Confusion Matrix Visualization
# ======================================
# TASK: Visualize the confusion matrix
# YOUR CODE HERE

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

```
# compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# create display object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_)

# plot it
plt.figure()
disp.plot(cmap='Blues')  # you can omit cmap if your rubric says no colors
plt.title("Confusion Matrix")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```