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_score
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)

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

```
Dataset downloaded successfully.
```

Load the Dataset

```
# Unzip the dataset
import zipfile

try:
    with zipfile.ZipFile('archive (2).zip', 'r') as zip_ref:
        zip_ref.extractall('.')
    print("Dataset unzipped successfully.")

    # Load the dataset
    df = pd.read_csv('vgsales.csv')

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

except FileNotFoundError:
    print("Error: archive (2).zip not found. Please upload the zip file.")
except zipfile.BadZipFile:
    print("Error: archive (2).zip is not a valid zip file. Please ensure you have uploaded the correct fi
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Dataset unzipped successfully.
First 5 rows of the dataset:
   Rank                      Name Platform    Year         Genre Publisher  NA_Sales  EU_Sales  JP_Sales
0     1                Wii Sports      Wii  2006.0        Sports  Nintendo     41.49     29.02      3.77
1     2         Super Mario Bros.      NES  1985.0      Platform  Nintendo     29.08      3.58      6.81
2     3            Mario Kart Wii      Wii  2008.0        Racing  Nintendo     15.85     12.88      3.79
3     4         Wii Sports Resort      Wii  2009.0        Sports  Nintendo     15.75     11.01      3.28
4     5  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo     11.27      8.89     10.22
```

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_Sales  Global_Sales
count  16598.000000  16327.000000  16598.000000  16598.000000  16598.000000  16598.000000  16598.000000
mean    8300.605254   2006.406443      0.264667      0.146652      0.077782      0.048063      0.537441
std     4791.853933      5.828981      0.816683      0.505351      0.309291      0.188588      1.555028
min        1.000000   1980.000000      0.000000      0.000000      0.000000      0.000000      0.010000
25%     4151.250000   2003.000000      0.000000      0.000000      0.000000      0.000000      0.060000
50%     8300.500000   2007.000000      0.080000      0.020000      0.000000      0.010000      0.170000
75%    12449.750000   2010.000000      0.240000      0.110000      0.040000      0.040000      0.470000
max    16600.000000   2020.000000     41.490000     29.020000     10.220000     10.570000     82.740000
```

```
# 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()
```

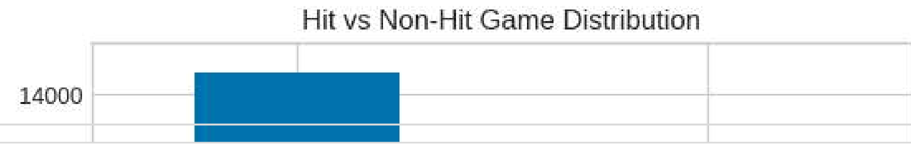

```
# Cell 7: Create Target Variable
# ============================
# TASK: Create a binary target variable for "hit" games
# A game is considered a hit if it sold more than 1 million units (Global_Sales > 1)
# YOUR CODE HERE
df['Hit'] = (df['Global_Sales'] > 1).astype(int)
```

```
# Cell 8: Analyze Target Distribution
# =================================
# Let's see the proportion of hits in our dataset
# YOUR CODE HERE
hit_distribution = df['Hit'].value_counts(normalize=True)

# Display as percentages
print("Proportion of hit vs. non-hit games:")
print(hit_distribution)

# visualize it
df['Hit'].value_counts().plot(kind='bar', title='Hit vs Non-Hit Game Distribution')
```

```
Proportion of hit vs. non-hit games:
Hit
0    0.87625
1    0.12375
Name: proportion, dtype: float64
<Axes: title={'center': 'Hit vs Non-Hit Game Distribution'}, xlabel='Hit'>
```




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

# check remaining columns
df.head()
```

	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Hit
0	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74	1
1	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24	1
2	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82	1
3	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00	1
4	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37	1

Next steps: Generate code with df New interactive sheet

```
# Cell 10: Missing Value Analysis
# ============================
# Examine the 'Year' column which might have missing values
print("Missing values in 'Year':", df['Year'].isna().sum())

# Display rows with missing 'Year' values
df[df['Year'].isna()].head()
```

```
Missing values in 'Year': 271
     Platform  Year   Genre    Publisher  NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales  Hit
```

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

# Verify that there are no more missing values
print("Remaining missing values after dropping:")
print(df.isna().sum())
```

```
Remaining missing values after dropping:
Platform        0
Year            0
Genre           0
Publisher       0
NA_Sales        0
EU_Sales        0
JP_Sales        0
Other_Sales     0
Global_Sales    0
Hit             0
dtype: int64
```

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

```
Categorical columns: ['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
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
le = LabelEncoder()

# Encode categorical columns
categorical_cols = ['Platform', 'Genre', 'Publisher']  # adjust if your dataset has different ones

for col in categorical_cols:
    df[col] = le.fit_transform(df[col].astype(str))

# Verify encoding
df[categorical_cols].head()
```

	Platform	Genre	Publisher
0	26	10	359
1	11	4	359
2	26	6	359

```
# 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
```

```
# Cell 15: Explore Processed Dataset
# ================================
# Let's look at the processed dataset
# YOUR CODE HERE

print("Dataset shape:", df.shape)
print("\nColumns in dataset:\n", df.columns.tolist())

# Display basic info
print("\nDataset Info:")
print(df.info())

# Preview the first few rows
print("\nFirst 5 rows of the processed dataset:")
display(df.head())

# Check summary statistics
print("\nSummary statistics:")
display(df.describe())
```

```
Dataset shape: (16291, 10)

Columns in dataset:
 ['Platform', 'Year', 'Genre', 'Publisher', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sa

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 16291 entries, 0 to 16597
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Platform      16291 non-null  int64
 1   Year          16291 non-null  float64
 2   Genre         16291 non-null  int64
 3   Publisher     16291 non-null  int64
 4   NA_Sales      16291 non-null  float64
 5   EU_Sales      16291 non-null  float64
 6   JP_Sales      16291 non-null  float64
 7   Other_Sales   16291 non-null  float64
 8   Global_Sales  16291 non-null  float64
 9   Hit           16291 non-null  int64
dtypes: float64(6), int64(4)
memory usage: 1.4 MB
None

First 5 rows of the processed dataset:
```

	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Hit
0	26	2006.0	10	359	41.49	29.02	3.77	8.46	82.74	1
1	11	1985.0	4	359	29.08	3.58	6.81	0.77	40.24	1
2	26	2008.0	6	359	15.85	12.88	3.79	3.31	35.82	1
3	26	2009.0	10	359	15.75	11.01	3.28	2.96	33.00	1
4	5	1996.0	7	359	11.27	8.89	10.22	1.00	31.37	1

Summary statistics:

	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Ot
count	16291.000000	16291.000000	16291.000000	16291.000000	16291.000000	16291.000000	16291.000000	162
mean	15.812841	2006.405561	4.928611	291.983365	0.265647	0.147731	0.078833	
std	8.369998	5.832412	3.762844	176.642066	0.822432	0.509303	0.311879	
min	0.000000	1980.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	7.000000	2003.000000	1.000000	137.000000	0.000000	0.000000	0.000000	
50%	16.000000	2007.000000	5.000000	323.000000	0.080000	0.020000	0.000000	
75%	21.000000	2010.000000	8.000000	455.000000	0.240000	0.110000	0.040000	
max	30.000000	2020.000000	11.000000	575.000000	41.490000	29.020000	10.220000	

```
# Cell 16: Split Features and Target
# ================================
# TASK: Split the data into features (X) and target (y)
# YOUR CODE HERE

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

# Confirm the split
```

```
print("Features shape:", X.shape)
print("Target shape:", y.shape)

# preview
X.head(), y.head()
```

```
Features shape: (16291, 9)
Target shape: (16291,)
(   Platform    Year  Genre  Publisher  NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales
 0        26  2006.0     10        359     41.49     29.02      3.77         8.46         82.74
 1        11  1985.0      4        359     29.08      3.58      6.81         0.77         40.24
 2        26  2008.0      6        359     15.85     12.88      3.79         3.31         35.82
 3        26  2009.0     10        359     15.75     11.01      3.28         2.96         33.00
 4         5  1996.0      7        359     11.27      8.89     10.22         1.00         31.37,
 0    1
 1    1
 2    1
 3    1
 4    1
 Name: Hit, dtype: int64)
```

```
# 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

from sklearn.model_selection import train_test_split

# 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, stratify=y
)

# Confirm the shapes
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (13032, 9)
X_test shape: (3259, 9)
y_train shape: (13032,)
y_test shape: (3259,)
```

```
# Cell 18: Train Initial Model
# =========================
# TASK: Train a Decision Tree classifier with default parameters
# YOUR CODE HERE
from sklearn.tree import DecisionTreeClassifier

# Initialize the classifier
clf = DecisionTreeClassifier(random_state=42) # Added random_state for reproducibility

# Train the classifier
clf.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Cell 19: Make Predictions
# ======================
# TASK: Make predictions on the test set
# YOUR CODE HERE

y_pred = clf.predict(X_test)

y_prob = clf.predict_proba(X_test)[:, 1]  # probability for class 1 (hit)

# Preview predictions
print("Predicted classes:\n", y_pred[:10])
print("Predicted probabilities of being a hit:\n", y_prob[:10])
```

```
Predicted classes:
 [0 0 0 1 0 0 0 0 0 0]
Predicted probabilities of being a hit:
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

```
# Cell 20: Calculate Evaluation Metrics
# ==================================
# TASK: Calculate evaluation metrics
# YOUR CODE HERE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Calculate metrics
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)
roc_auc = roc_auc_score(y_test, y_prob)

# Print metrics
print(f"Accuracy:  {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall:    {recall:.3f}")
print(f"F1 Score:  {f1:.3f}")
print(f"ROC AUC:   {roc_auc:.3f}")
```

```
Accuracy:  1.000
Precision: 1.000
Recall:    1.000
F1 Score:  1.000
ROC AUC:   1.000
```

```
# Cell 21: Confusion Matrix Visualization
# ====================================
# TASK: Visualize the confusion matrix
# YOUR CODE HERE
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

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

```
# Plot confusion matrix
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Hit', 'Hit'], yticklabels=['Not Hit
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```