```
#Import packages
import numpy as np
import pandas as pd
from sklearn.tree import export_graphviz
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
from matplotlib.patches import Rectangle
```

Data Cleaning

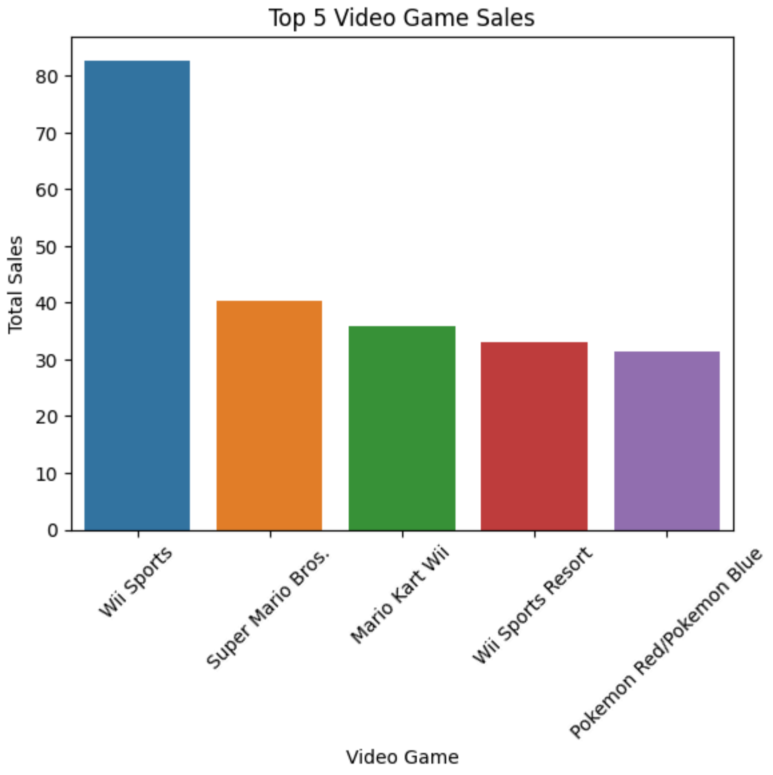
```
#Import the dataset
data = pd.read_csv("vgsales.csv")
#Check the NA values
data.isnull().sum()
#Drop NA values and rename the dataset without NA values
data = data.dropna()
```

Question 1: What are the top 5 video games with the most sales and in which region did they sell the most?

```
# Verifying Top 5 Video Games
top_five = data[:5]
print(top_five)
```

```
   Rank                      Name Platform    Year         Genre Publisher  \
0     1                Wii Sports      Wii  2006.0        Sports  Nintendo
1     2         Super Mario Bros.      NES  1985.0      Platform  Nintendo
2     3            Mario Kart Wii      Wii  2008.0        Racing  Nintendo
3     4         Wii Sports Resort      Wii  2009.0        Sports  Nintendo
4     5  Pokemon Red/Pokemon Blue       GB  1996.0  Role-Playing  Nintendo

   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales
0     41.49     29.02      3.77         8.46         82.74
1     29.08      3.58      6.81         0.77         40.24
2     15.85     12.88      3.79         3.31         35.82
3     15.75     11.01      3.28         2.96         33.00
4     11.27      8.89     10.22         1.00         31.37
```

```
# Top 5 Video Games
top_five = data[:5]
plot = sns.barplot(data=top_five, x='Name', y='Global_Sales', hue='Name')
plot.tick_params(axis="x", rotation=45)
plot.set(title='Top 5 Video Game Sales', xlabel='Video Game', ylabel='Total Sales')
plt.show()
```

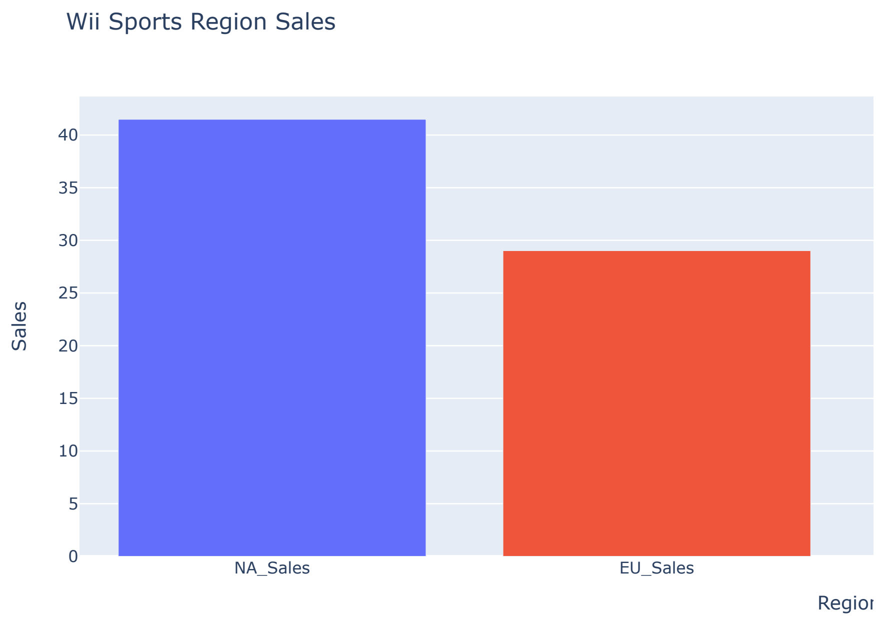


```
# Verifying Region Sale Values
regions = data[['Name','NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
game_region = regions[:5]
print(game_region)
```

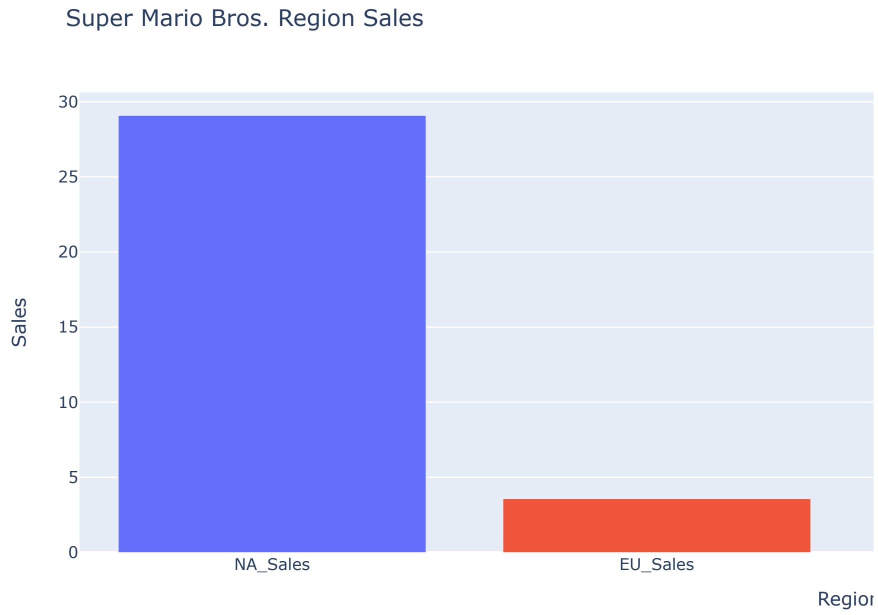
```
                       Name  NA_Sales  EU_Sales  JP_Sales  Other_Sales
0                Wii Sports     41.49     29.02      3.77         8.46
1         Super Mario Bros.     29.08      3.58      6.81         0.77
2            Mario Kart Wii     15.85     12.88      3.79         3.31
```

```
3         Wii Sports Resort     15.75     11.01      3.28         2.96
4  Pokemon Red/Pokemon Blue     11.27      8.89     10.22         1.00
```

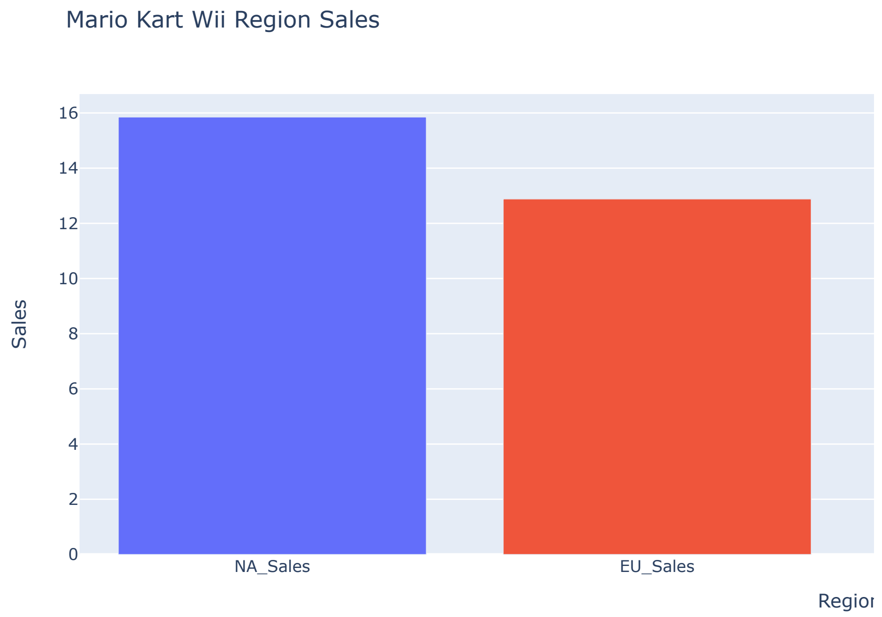
```
# Wii Sports Region with most sales
regions = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
wii_region = regions[:1]

# making region sales into rows
melted_data_1 = wii_region.melt(
        var_name='Region',
        value_name='Sales',
        ignore_index=False)

plot = px.bar(melted_data_1, x='Region', y='Sales', color='Region', title='Wii Sports Region Sales')
plot
```



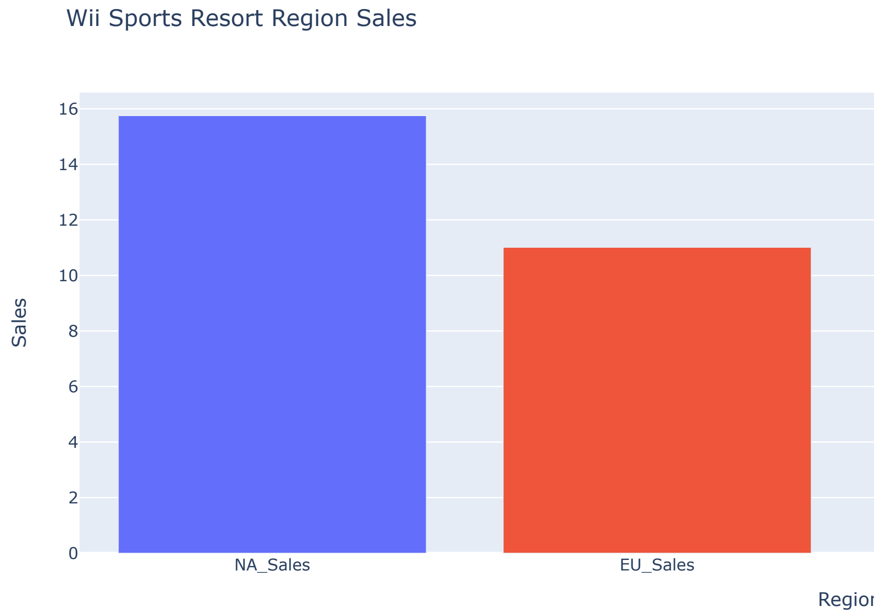
```
# Super Mario Bros. Region with most sales
regions = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
mario_region = regions[1:2]

# making region sales into rows
melted_data_2 = mario_region.melt(
        var_name='Region',
        value_name='Sales',
        ignore_index=False)

plot = px.bar(melted_data_2, x='Region', y='Sales', color='Region', title='Super Mario Bros. Region Sales')
plot
```



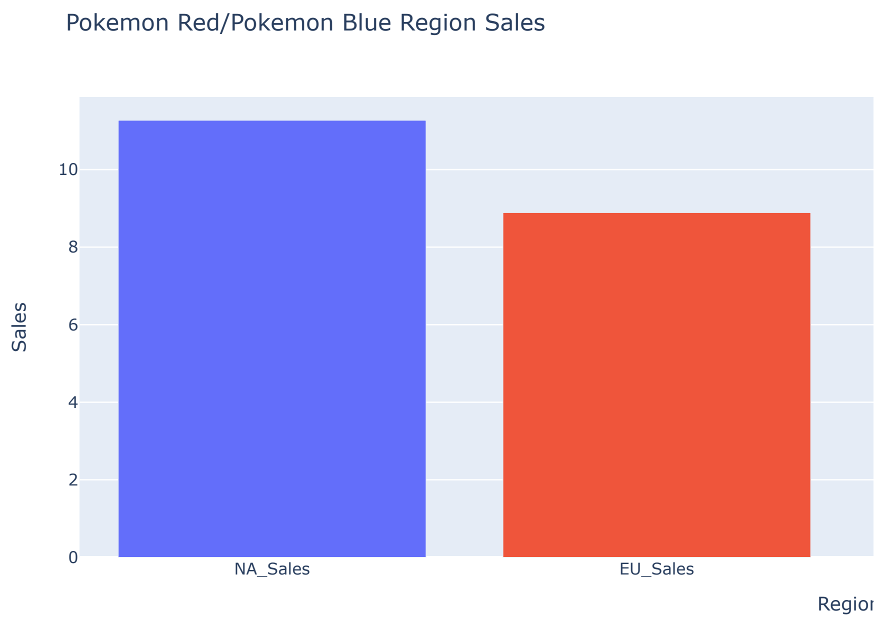
```
# Mario Kart Wii Region with the most sales
regions = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
kart_region = regions[2:3]

# making region sales into rows
melted_data_3 = kart_region.melt(
        var_name='Region',
        value_name='Sales',
        ignore_index=False)

plot = px.bar(melted_data_3, x='Region', y='Sales', color='Region', title='Mario Kart Wii Region Sales')
plot
```



```
# Wii Sports Resort Region with the most sales
regions = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
resort_region = regions[3:4]

# making region sales into rows
melted_data_4 = resort_region.melt(
        var_name='Region',
        value_name='Sales',
        ignore_index=False)

plot = px.bar(melted_data_4, x='Region', y='Sales', color='Region', title='Wii Sports Resort Region Sales')
plot
```



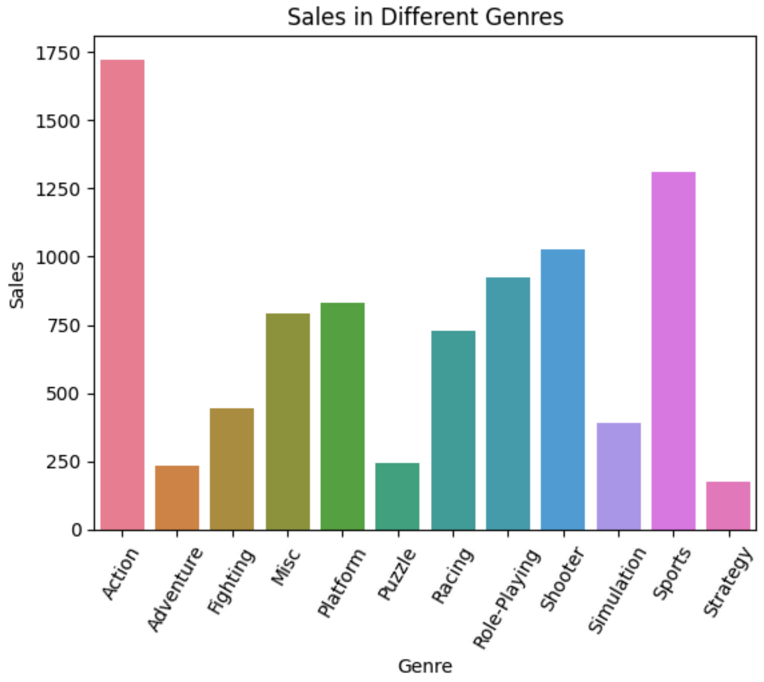
```
# Pokemon Red/Pokemon Blue Region with the most sales
regions = data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
pokemon_region = regions[4:5]

# making region sales into rows
melted_data_5 = pokemon_region.melt(
        var_name='Region',
        value_name='Sales',
        ignore_index=False)

plot = px.bar(melted_data_5, x='Region', y='Sales', color='Region', title='Pokemon Red/Pokemon Blue Region Sales')
plot
```



Question 2: Which genre of video game has the highest sales figures and what potential factors contribute to its appeal?

```
#sales of each genre
genre_data = data.groupby('Genre')
sales = genre_data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales']]
total_sales = sales.sum()
total_sales['Sales'] = genre_data['Global_Sales'].sum()


bar = sns.barplot(total_sales, x='Genre', y = 'Sales', hue='Genre')
bar.tick_params(axis="x", rotation=60)
bar.set(title='Sales in Different Genres', xlabel='Genre', ylabel= 'Sales')
```

```
[Text(0.5, 1.0, 'Sales in Different Genres'),
 Text(0.5, 0, 'Genre'),
 Text(0, 0.5, 'Sales')]
```



Question 3: Have the genres with the highest sales maintained consistent popularity over time?

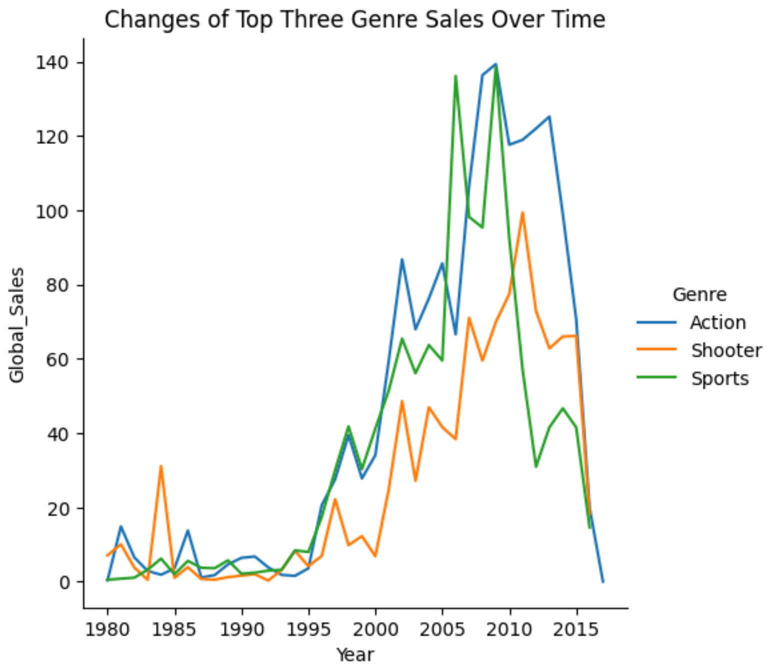
```
#Top three most popular genre
sort = total_sales.sort_values(by='Sales', ascending=False)
top_three = sort.head(3)
top_three
```

Genre	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Sales
Action	861.77	516.48	158.65	184.92	1722.84
Sports	670.09	371.34	134.76	132.65	1309.24
Shooter	575.16	310.45	38.18	101.90	1026.20

Next steps: Generate code with top_three View recommended plots

```
#changes over years
top_three_genre = data.loc[data['Genre'].isin(['Action', 'Sports', 'Shooter'])]
group = top_three_genre.groupby(['Year', 'Genre']).sum()
line = sns.relplot(group, x="Year", y="Global_Sales", hue="Genre", kind="line")
line.set(title='Changes of Top Three Genre Sales Over Time')
line
```

```
<ipython-input-13-9e11047ea8d6>:3: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a fu

<seaborn.axisgrid.FacetGrid at 0x7f3f2f494f70>
```



Sales Region Analysis: Analyze the factors (platform, genre, publisher, and year of release) in influencing global sales

Data Preparation

```
# Convert 'Year' to integer
data['Year'] = data['Year'].astype(int)
# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
# Transform the 'Platform', 'Genre', 'Publisher' columns, and convert it to a DataFrame
trasformed_data = encoder.fit_transform(data[['Platform', 'Genre', 'Publisher']])
feature_names = encoder.get_feature_names_out(['Platform', 'Genre', 'Publisher'])
encoded_df = pd.DataFrame(trasformed_data, columns=feature_names)
# Concatenate the encoded_df with 'Year' and 'Global_Sales' from the original data
new_data = pd.concat([data[['Year', 'Global_Sales']].reset_index(drop=True), encoded_df], axis=1)
# Define features X and target y using the new_data DataFrame
X = new_data.drop(['Global_Sales'], axis=1)
y = new_data['Global_Sales']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning:

`sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you lea
```

Train the model, Predict, and Evaluate

```
# Initialize and train the RandomForestRegressor
model = RandomForestRegressor(n_estimators = 20,random_state=42)
model.fit(X_train, y_train)
# Predict on the test set
y_predict = model.predict(X_test)
# Evaluate the model
MAE = mean_absolute_error(y_test, y_predict)
R2 = r2_score(y_test, y_predict)
scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')
print(f"Mean Absolute Error: {MAE}")
print(f"R^2 Score: {R2}")
print(f"Average cross-validated R^2: {np.mean(scores)}")
```

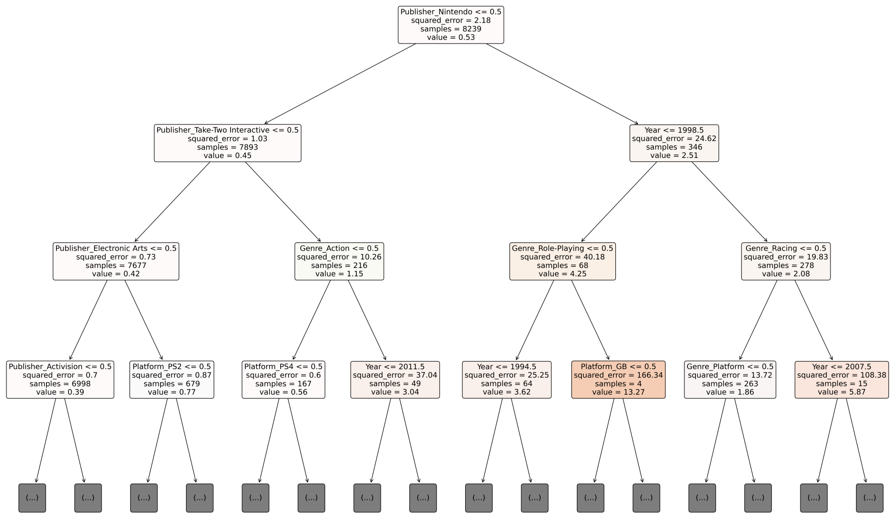
```
Mean Absolute Error: 0.5232569474055581
R^2 Score: 0.025417878866786925
Average cross-validated R^2: -0.04068050592557313
```

Feature Importance Analysis

```
# Feature Importance
feature_importances = model.feature_importances_
features = X.columns
# Combine feature names with their importance scores
feature_importance_dictionary = dict(zip(features, feature_importances))
# Sort the feature importances in descending order
sorted_feature_importances = sorted(feature_importance_dictionary.items(), key=lambda item: item[1], reverse=True)
#Total Importance
total_importance = sum(feature_importances)
# Display the top 10 most important features
print("Most Important Features in Percentage:")
for feature, importance in sorted_feature_importances[:10]:
    print(f"{feature}: {importance / total_importance * 100:.2f}%")
```

```
Most Important Features in Percentage:
Year: 42.05%
Publisher_Nintendo: 10.30%
Genre_Role-Playing: 4.89%
Genre_Shooter: 4.04%
Genre_Platform: 2.72%
Genre_Racing: 2.67%
Genre_Action: 2.65%
Genre_Misc: 2.64%
Platform_GB: 2.59%
Platform_Wii: 2.26%
```

```
# Plot the top levels of the tree with a limited max_depth
single_tree = model.estimators_[0]

plt.figure(figsize=(30, 20), dpi=480)  # Set a large figure size and high DPI for better resolution
plot_tree(single_tree,
          feature_names=X.columns,
          filled=True,
          rounded=True,
          precision=2,
          fontsize=14,
          max_depth=3)
plt.show()
```



Test

```
#test 1
def test_na_value(df):
  """
  This function called test_na_value take dataframe as an input and
  return whether there is NA value in the dataset

  >>> test_na_value(data)
  False
  """
  if df.isna().any().any():
    return True
  else:
    return False

#test 2
assert sorted((rectangle.get_height() for rectangle in bar.findobj(Rectangle)[:2]),
              reverse=True) == [1722.84, 234.59], "Data does not match expected"

#test 3
assert isinstance(line, sns.axisgrid.FacetGrid), "Failed: Plot not created successfully"
ax = line.ax
assert ax.get_title() == "Changes of Top Three Genre Sales Over Time", "title does not match expected"
assert all(line.get_xydata().size == 0 for line in ax.get_lines()[37:]), "unexpected extra data"

#test 4
expected = len(data['Platform'].unique()) + len(data['Genre'].unique()) + len(data['Publisher'].unique()) + 1
actual = encoded_df.shape[1] + 1
assert actual == expected, f"The number of columns after encoding does not match the expected value)."

#test 5
def test_importance_scores(model, features):
    """
    Test if the feature importance scores from a trained model sum up to 1 (100%).

    >>> test_importance_scores(model, X.columns)
    (True, 'The importance scores is equal to one 1 (100%).')
    """
    feature_importances = model.feature_importances_

    if np.isclose(np.sum(feature_importances), 1.0, atol=1e-6):
        return True, "The importance scores is equal to one 1 (100%)."
    else:
        return False, f"The importance scores sum up to {np.sum(feature_importances):.2f} instead of 1."
```

```
import doctest
doctest.testmod()
```

```
TestResults(failed=0, attempted=2)
```