Add the imports, Define the folder to your desktop, Create a combined DataFrame, Display the combined DataFrame, Save the combined DataFrame

import os

import pandas as pd

# Define the path to the folder on your desktop

folder\_path = os.path.expanduser(r'C:\Users\fabio\Downloads\INF791')

# List to store individual DataFrames

dfs = []

# Loop through all files in the folder

for filename in os.listdir(folder\_path):

    if filename.endswith('.csv'):

        file\_path = os.path.join(folder\_path, filename)

        # Read the CSV file and append it to the list

        df = pd.read\_csv(file\_path)

        dfs.append(df)

# Combine all DataFrames into a single DataFrame

combined\_df = pd.concat(dfs, ignore\_index=True)

# Display the combined DataFrame

combined\_df.head()

# Optionally, save the combined DataFrame to a new CSV file

combined\_df.to\_csv(os.path.join(folder\_path, 'combined\_csv\_output.csv'), index=False)

print(f"Combined CSV saved to {os.path.join(folder\_path, 'combined\_csv\_output.csv')}")

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from sklearn.preprocessing import StandardScaler

# Phase 1: Data Collection

pd.set\_option("expand\_frame\_repr", False)

df= pd.read\_csv("Game\_Analytics\_Dataset.csv", delimiter=';')

**Data Preparation**

# Drop all the duplicates

df2 = df.drop\_duplicates()

# --- Remove negative values from time feature --- #

df2['Time\_in\_seconds'] = pd.to\_numeric(df2['Time\_in\_seconds'], errors='coerce')

df2 = df2.dropna(subset=['Time\_in\_seconds'])

df2['Time\_in\_seconds'] = df2['Time\_in\_seconds']

print(df2.head())

**A screen shot of a computer

Description automatically generated**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# --- Summary Statistics --- #

# Replace these column names with the relevant numeric columns in your dataset

summary\_stats = df[['Time\_in\_seconds', 'Defender\_Score', 'Attacker\_Score']].describe()

print("Summary Statistics:\n", summary\_stats)

# Display first few rows

print("\nFirst few rows:\n", df.head())

# Display last few rows

print("\nLast few rows:\n", df.tail())

# Calculate the median of the columns

median\_values = df[['Time\_in\_seconds', 'Defender\_Score', 'Attacker\_Score']].median()

print("\nMedian Values:\n", median\_values)

# --- Correlations --- #

# Calculate the correlation matrix

c\_matrix = df[['Time\_in\_seconds', 'Defender\_Score', 'Attacker\_Score']].corr()

# Plot the correlation heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(c\_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

plt.title('Correlation Heatmap')

plt.show()

# (Optional) Save the cleaned dataset as a new CSV file

output\_path = 'Cleaned\_Game\_Analytics\_Dataset\_Processed.csv'

df.to\_csv(output\_path, index=False)

print(f"\nCleaned dataset saved to {output\_path}")

****import pandas as pd

from pandas\_profiling import ProfileReport

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Generate the EDA report

profile = ProfileReport(df, title="Pandas Profiling Report", explorative=True)

# Display the report in the notebook (or open it in a browser)

profile.to\_file("EDA\_Report.html")  # Saves the report as an HTML file

# Optionally, if you want to directly open in a browser (Jupyter Notebook)

# profile.to\_notebook\_iframe()  # Displays the report directly in a Jupyter Notebook

****

fig, ax = plt.subplots(figsize=(10, 6))

ax.hist(df2['Defender\_Score'], bins=40, alpha=0.5, color='blue', label='Defender Score')

ax.hist(df2['Attacker\_Score'], bins=40, alpha=0.5, color='yellow', label='Attacker Score')

# Add labels and a legend

ax.set\_xlabel('Attacker vs Defender Distribution')

ax.set\_ylabel('Frequency')

ax.set\_title('Distribution of scores obtained by the Defender and Attacker')

ax.legend()

# Set x-axis ticks to increment by 2

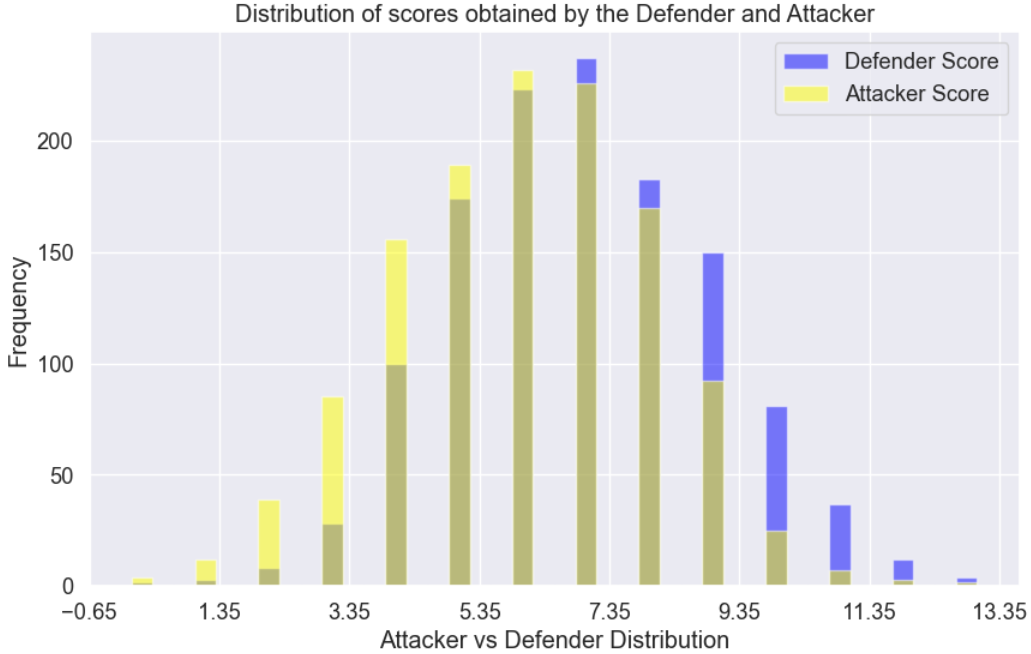
start, end = ax.get\_xlim()

ax.set\_xticks(np.arange(start, end, 2))

ax.legend()

# Show the plot

plt.show()

****

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('Cleaned\_Game\_Analytics\_Dataset.csv')

# Count the occurrences of each level

level\_counts = df['Level'].value\_counts()

# Define colors for each level (Beginner = Green, Intermediate = Yellow, Expert = Red)

colors = {

    'Beginner': 'green',

    'Intermediate': 'yellow',

    'Expert': 'red'

}

# Assign colors based on the levels

level\_colors = [colors[level] for level in level\_counts.index]

# Plot a pie chart for the levels with custom colors

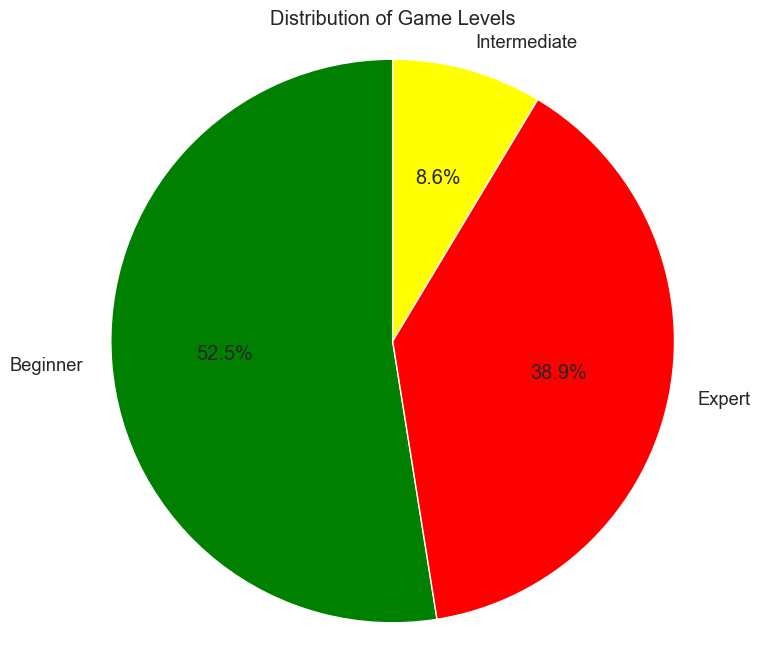
plt.figure(figsize=(8, 8))

plt.pie(level\_counts, labels=level\_counts.index, autopct='%1.1f%%', startangle=90, colors=level\_colors)

plt.title('Distribution of Game Levels')

plt.axis('equal')  # Equal aspect ratio ensures that pie chart is drawn as a circle.

plt.show()



import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

df = pd.read\_csv('Cleaned\_Game\_Analytics\_Dataset.csv')

# ------------------- Horizontal Bar Plot for Nicknames (Top 10 Players) -------------------

# Count the number of occurrences for each Nickname

nickname\_counts = df['Nickname'].value\_counts().head(10)  # Show top 10 players

# Create a horizontal bar plot for top players by nickname (using a blue color palette)

plt.figure(figsize=(10, 6))

sns.barplot(y=nickname\_counts.index, x=nickname\_counts.values, palette='Blues\_r', orient='h')

plt.title('Top 10 Players by Number of Game Rounds')

plt.xlabel('Number of Rounds Played')

plt.ylabel('Nickname')

plt.show()

# ------------------- Box Plot for Rounds Won by Defender, Attacker, and Draw -------------------

# Box plots show the distribution and highlight any outliers (Defender = blue, Attacker = orange, Draw = gray)

plt.figure(figsize=(12, 8))

sns.boxplot(x='Winner', y='Time\_in\_seconds', data=df, palette={'Defender': 'blue', 'Attacker': 'orange', 'Draw': 'gray'})

plt.title('Game Duration Distribution by Winner (Defender, Attacker, and Draw)')

plt.xlabel('Winner')

plt.ylabel('Time in Seconds')

plt.show()

# ------------------- Stacked Bar Plot for Rounds Won by Defender and Attacker at Each Level -------------------

# Create a cross-tabulation for Level and Winner

level\_winner\_counts = pd.crosstab(df['Level'], df['Winner'])

# Create a stacked bar plot for levels and winners (Defender = blue, Attacker = orange, Draw = gray)

level\_winner\_counts.plot(kind='bar', stacked=True, color=['blue', 'orange', 'gray'], figsize=(10, 6))

plt.title('Rounds Won by Defender, Attacker, and Draw at Each Level')

plt.xlabel('Game Level')

plt.ylabel('Number of Rounds Won')

plt.legend(title='Winner')

plt.show()

# ------------------- Donut Chart for Winner Distribution -------------------

# Count the occurrences for each winner (Defender, Attacker, Draw)

winner\_counts = df['Winner'].value\_counts()

# Create a donut chart (pie chart with a hole in the center) (Defender = blue, Attacker = orange, Draw = gray)

plt.figure(figsize=(8, 8))

plt.pie(winner\_counts, labels=winner\_counts.index, autopct='%1.1f%%', startangle=90, colors=['blue', 'orange', 'gray'], wedgeprops=dict(width=0.3))

plt.title('Distribution of Rounds Won by Defender, Attacker, and Draw')

plt.gca().set\_aspect('equal')  # Equal aspect ratio ensures the chart is a circle.

plt.show()

# ------------------- Heatmap for Correlation Between Numeric Variables -------------------

# Correlation matrix to show relationships between numeric values (e.g., Defender\_Score, Attacker\_Score, etc.)

numeric\_columns = df[['Defender\_Score', 'Attacker\_Score', 'Time\_in\_seconds', 'Score\_Difference']]

correlation\_matrix = numeric\_columns.corr()

# Create a heatmap to visualize correlations with a coolwarm color palette

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.2f')

plt.title('Correlation Heatmap of Numeric Variables')

plt.show()

# ------------------- Violin Plot for Level vs Time in Seconds -------------------

# Violin plots show the distribution of the data and density (Beginner = green, Intermediate = yellow, Expert = red)

plt.figure(figsize=(10, 8))

sns.violinplot(x='Level', y='Time\_in\_seconds', data=df, palette={'Beginner': 'green', 'Intermediate': 'yellow', 'Expert': 'red'})

plt.title('Distribution of Time Spent in Each Game Level')

plt.xlabel('Game Level')

plt.ylabel('Time in Seconds')

plt.show()

# ------------------- Pie Chart for Level Distribution -------------------

# Count the number of occurrences for each level

level\_counts = df['Level'].value\_counts()

# Create a pie chart for level distribution (Beginner = green, Intermediate = yellow, Expert = red)

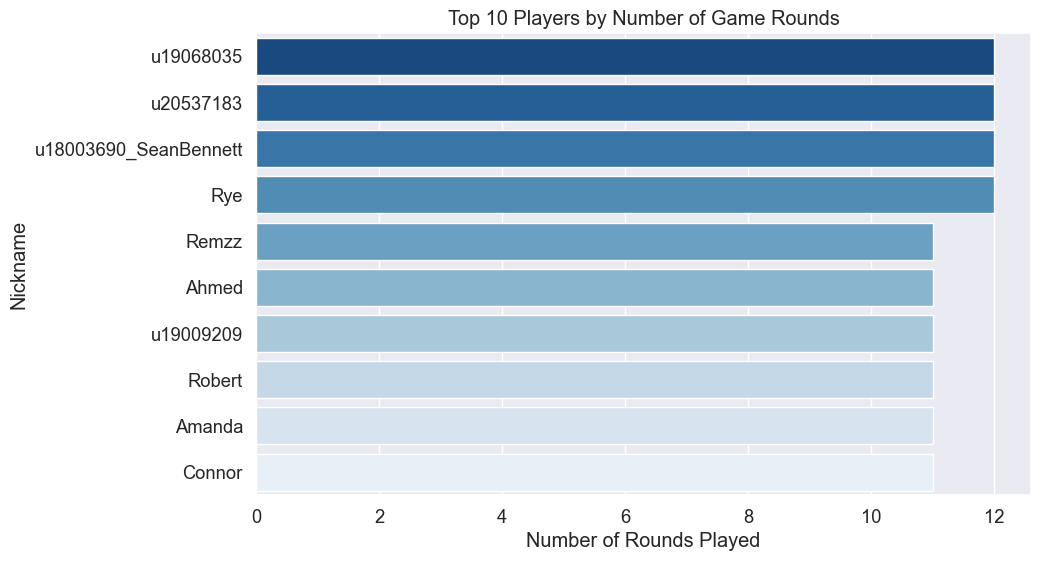
plt.figure(figsize=(8, 8))

plt.pie(level\_counts, labels=level\_counts.index, autopct='%1.1f%%', startangle=90, colors=['green', 'yellow', 'red'])

plt.title('Game Rounds Distribution by Level')

plt.axis('equal')  # Equal aspect ratio ensures that pie chart is drawn as a circle.

plt.show()



A graph of a game

Description automatically generated with medium confidence

A graph of a bar chart

Description automatically generated with medium confidence

A blue and orange circle with white text

Description automatically generated

A red and blue squares with white text

Description automatically generated

A graph showing different levels of a game level

Description automatically generated

A pie chart with numbers and a red green and yellow circle

Description automatically generated

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# print(df.columns)

# Specify the categorical features in your dataset

categorical\_features = ['Defender\_Score', 'Attacker\_Score', 'Winner', 'Level', 'Score\_Difference']

# Determine the number of rows needed for the subplots

num\_rows = int(len(categorical\_features) / 2) + len(categorical\_features) % 2

# Create subplots

fig, axs = plt.subplots(num\_rows, 2, figsize=(15, 5 \* num\_rows))

# Create the bar graphs

axs = axs.flatten()

for i, feature in enumerate(categorical\_features):

    sns.countplot(data=df, y=feature, ax=axs[i])

    axs[i].set\_title(f'Score Count Plot for {feature}')

    axs[i].set\_ylabel(f'{feature}')

    axs[i].set\_xlabel('Score Count')

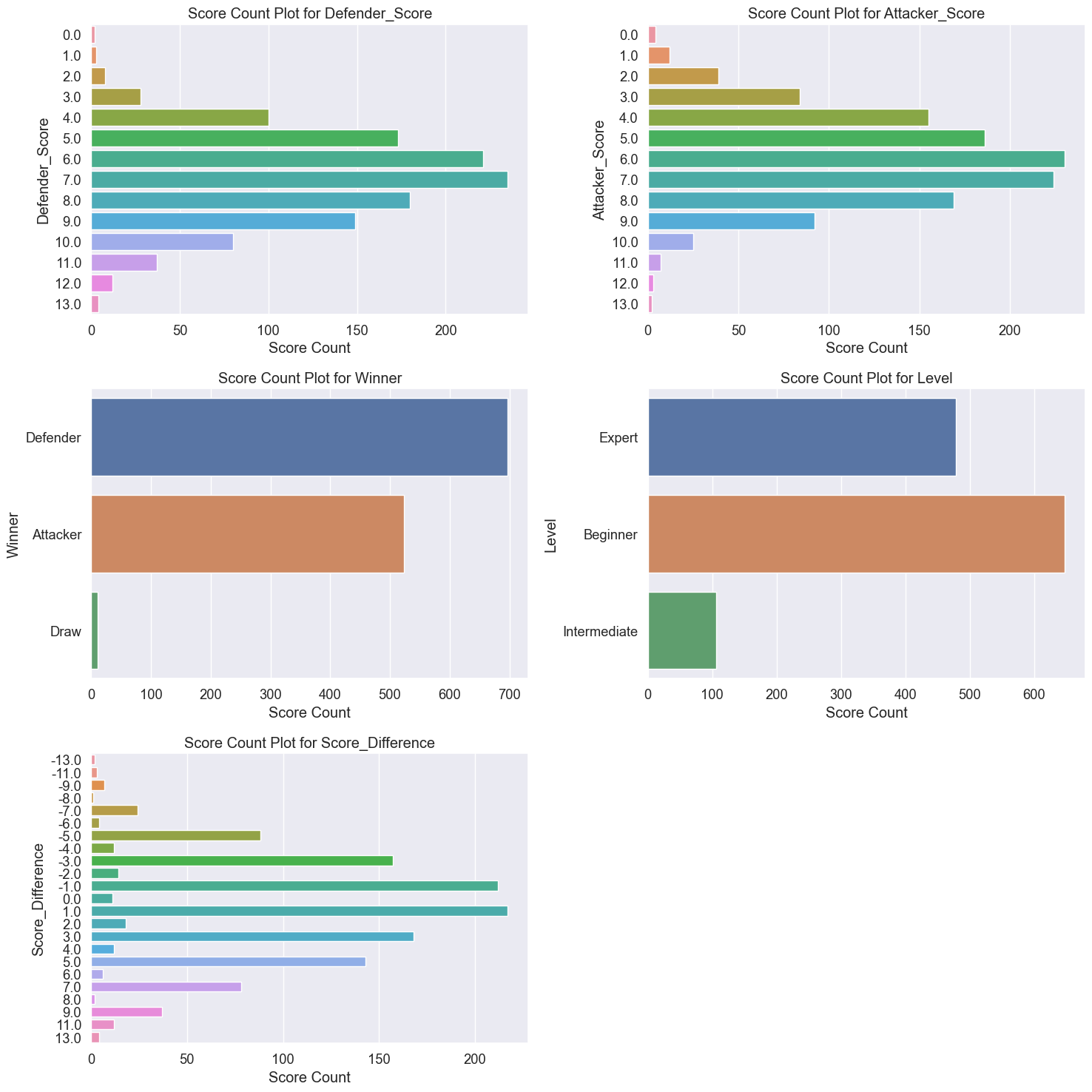
# Remove any unused subplots

for i in range(len(categorical\_features), num\_rows \* 2):

    fig.delaxes(axs[i])

plt.tight\_layout()

plt.show()



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from scipy import stats

# Load the cleaned data

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# --- Drop all duplicate rows --- #

df = df.drop\_duplicates()

# --- Handling and transforming 'Time\_in\_seconds' --- #

# Ensure 'Time\_in\_seconds' has no negative values and apply a shift

df['Time\_in\_seconds'] = df['Time\_in\_seconds'].apply(lambda x: max(x, 0))

# Yeo-Johnson transformation applied to 'Time\_in\_seconds'

df['YJ\_Time\_in\_seconds'], \_ = stats.yeojohnson(df['Time\_in\_seconds'])

# --- Math transformations to reduce skewness on other columns --- #

# Log transformation applied to 'Defender\_Score'

df['Log\_Defender\_Score'] = np.log(df['Defender\_Score'] + 1)

# Square root transformation applied to 'Attacker\_Score'

df['Sqrt\_Attacker\_Score'] = np.sqrt(df['Attacker\_Score'])

# -- PLOTTING TRANSFORMED DATA -- #

fig, ax = plt.subplots(figsize=(10, 6))

# Plot the transformed 'Defender\_Score' column

ax.hist(df['Log\_Defender\_Score'], bins=50, alpha=0.5, color='blue', label='Defender Score (Log)')

# Plot the transformed 'Attacker\_Score' column

ax.hist(df['Sqrt\_Attacker\_Score'], bins=50, alpha=0.5, color='#8A0707', label='Attacker Score (Square Root)')

# Plot the transformed 'YJ\_Time\_in\_seconds' column

ax.hist(df['YJ\_Time\_in\_seconds'], bins=50, alpha=0.5, color='green', label='Time in seconds (Yeo-Johnson)')

# Add labels and a legend

ax.set\_xlabel('Transformed Values')

ax.set\_ylabel('Frequency')

ax.set\_title('Distribution of Transformed Columns')

ax.legend()

# Show the plot

plt.show()

# Create a figure and axis for the density plot

fig, ax = plt.subplots(figsize=(10, 6))

# Create a StandardScaler instance

scaler = StandardScaler()

# Normalize each column's features

df\_normalized = df.copy()

df\_normalized[['Log\_Defender\_Score', 'Sqrt\_Attacker\_Score', 'YJ\_Time\_in\_seconds']] = scaler.fit\_transform(

    df[['Log\_Defender\_Score', 'Sqrt\_Attacker\_Score', 'YJ\_Time\_in\_seconds']]

)

# Plot the density of the normalized 'Log\_Defender\_Score' column

sns.kdeplot(df\_normalized['Log\_Defender\_Score'], color='blue', label='Defender Score (Log)', ax=ax)

# Plot the density of the normalized 'Sqrt\_Attacker\_Score' column

sns.kdeplot(df\_normalized['Sqrt\_Attacker\_Score'], color='#8A0707', label='Attacker Score (Square Root)', ax=ax)

# Plot the density of the normalized 'YJ\_Time\_in\_seconds' column

sns.kdeplot(df\_normalized['YJ\_Time\_in\_seconds'], color='green', label='Time in seconds (Yeo-Johnson)', ax=ax)

# Add labels and a legend

ax.set\_xlabel('Normalized Values')

ax.set\_ylabel('Density')

ax.set\_title('Density Plot of Normalized Columns')

ax.legend()

# Show the plot

plt.show()

A graph of different colored columns

Description automatically generated

A graph of a normalized column

Description automatically generated

import pandas as pd

# Load the data

file\_path = 'Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path, delimiter=';')

# Handle missing values

# Drop rows where the 'Nickname' is missing

df = df.dropna(subset=['Nickname'])

# Convert data types

df['Defender\_Score'] = pd.to\_numeric(df['Defender\_Score'], errors='coerce')

df['Attacker\_Score'] = pd.to\_numeric(df['Attacker\_Score'], errors='coerce')

df['Time\_in\_seconds'] = pd.to\_numeric(df['Time\_in\_seconds'], errors='coerce')

# Handle any remaining missing values (e.g., by filling with a default value or removing)

# df = df.fillna({'Defender\_Score': 0, 'Attacker\_Score': 0, 'Time\_in\_seconds': df['Time\_in\_seconds'].mean()})

# Remove duplicates

df = df.drop\_duplicates()

# Additional processing

# Example: Creating a new column for the score difference

df['Score\_Difference'] = df['Defender\_Score'] - df['Attacker\_Score']

# Save the cleaned data

output\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your desired output path

df.to\_csv(output\_path, index=False)

print("Data cleaned and saved to", output\_path)

fig, ax = plt.subplots(figsize=(10, 4))

ax.hist(df['Defender\_Score'], bins=30, alpha=0.5, color='blue', label='Defender Score')

ax.hist(df['Attacker\_Score'], bins=30, alpha=0.7, color='#8A0707', label='Attacker Score')

# Add labels and a legend

ax.set\_xlabel('Score Obtained')

ax.set\_ylabel('Frequency')

ax.set\_title('Distribution of scores obtained by the Defender and Attacker')

ax.legend()

# Set x-axis ticks to increment by 2

# start, end = ax.get\_xlim()

# ax.set\_xticks(np.arange(start, end, 2))

ax.legend()

# Show the plot

plt.show()

A graph of a bar graph

Description automatically generated with medium confidence

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned data

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Function to normalize the "Time\_in\_seconds" column using Z-score normalization

def normalize\_column\_z\_score(df, column):

    normalized\_column = (df[column] - df[column].mean()) / df[column].std()

    return normalized\_column

# Apply z-score normalization to the "Time\_in\_seconds" column

df['Normalized\_Time\_in\_seconds'] = normalize\_column\_z\_score(df, 'Time\_in\_seconds')

# Function to plot histogram with mean and standard deviation lines

def plot\_histogram\_with\_stats(df, feature, bins=30):

    data = df[feature]

    mean = np.mean(data)

    std\_dev = np.std(data)

    plt.figure(figsize=(10, 6))

    ax = sns.histplot(data, bins=bins, kde=True, color='skyblue', edgecolor='black', alpha=0.7)

    ax.lines[0].set\_color('black')

    plt.axvline(mean, color='red', linestyle='dashed', linewidth=1, label=f'Mean: {mean:.2f}')

    plt.axvline(mean - std\_dev, color='green', linestyle='dashed', linewidth=1, label=f'Mean - Std Dev: {mean - std\_dev:.2f}')

    plt.axvline(mean + std\_dev, color='orange', linestyle='dashed', linewidth=1, label=f'Mean + Std Dev: {mean + std\_dev:.2f}')

    plt.axvline(std\_dev, color='blue', linestyle='dotted', linewidth=1, label=f'Std Dev: {std\_dev:.2f}')

    plt.legend(loc='upper right')

    plt.title(f'Histogram of {feature}')

    plt.xlabel(feature)

    plt.ylabel('Frequency')

    plt.show()

# Visualizations for relevant numeric columns in your dataset

# Z-score normalized Time\_in\_seconds

plot\_histogram\_with\_stats(df, 'Normalized\_Time\_in\_seconds')

# Defender\_Score

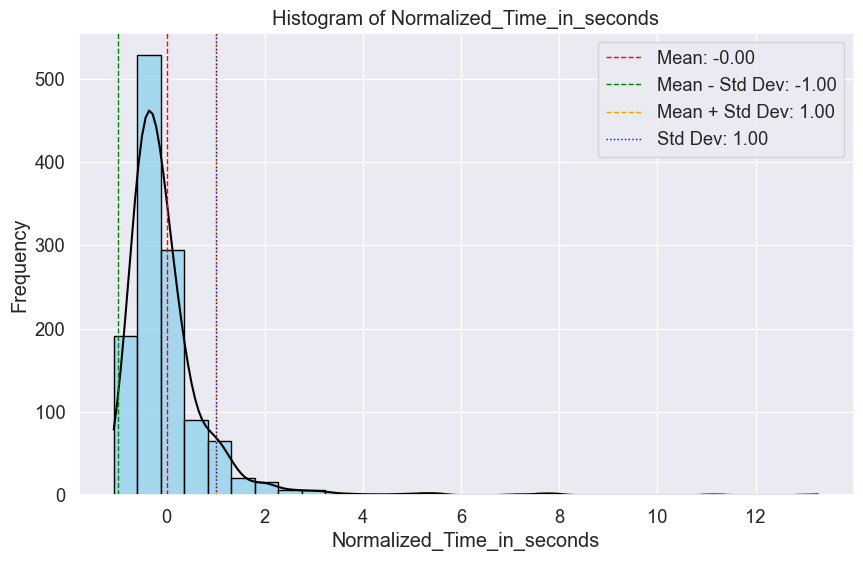
plot\_histogram\_with\_stats(df, 'Defender\_Score')

# Attacker\_Score

plot\_histogram\_with\_stats(df, 'Attacker\_Score')

# Example: If you have any other numeric column to visualize, add it here

# plot\_histogram\_with\_stats(df, 'Other\_Numeric\_Column')



A graph of a person with a line

Description automatically generated with medium confidence

A graph showing a graph of a person

Description automatically generated with medium confidence

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned data

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'

df = pd.read\_csv(file\_path)

# Function to add annotations to the bars

def annotate\_barplot(ax):

    for p in ax.patches:

        ax.annotate(f'{int(p.get\_height())}', (p.get\_x() + p.get\_width() / 2., p.get\_height()),

                    ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),

                    textcoords='offset points')

# --- Count visualizations --- #

# Winner count

plt.figure(figsize=(8, 6))

ax = sns.countplot(x='Winner', data=df, palette='viridis')

plt.title('Distribution of Winners (Defender vs Attacker)', fontsize=14)

annotate\_barplot(ax)

plt.show()

# Level count

plt.figure(figsize=(10, 6))

ax = sns.countplot(x='Level', data=df, palette='muted')

plt.title('Distribution of Game Levels', fontsize=14)

annotate\_barplot(ax)

plt.show()

# If Nickname has too many unique values, consider grouping or displaying the top N

top\_n = 10  # Adjust the top N value as necessary

top\_nicknames = df['Nickname'].value\_counts().nlargest(top\_n).index

filtered\_df = df[df['Nickname'].isin(top\_nicknames)]

plt.figure(figsize=(12, 6))

ax = sns.countplot(x='Nickname', data=filtered\_df, palette='Set2', order=top\_nicknames)

plt.title(f'Top {top\_n} Nicknames by Number of Games Played', fontsize=14)

plt.xticks(rotation=45, ha='right')

annotate\_barplot(ax)

plt.tight\_layout()

plt.show()

# Combine the analysis of Level and Winner using a grouped bar plot

plt.figure(figsize=(12, 6))

ax = sns.countplot(x='Level', hue='Winner', data=df, palette='coolwarm')

plt.title('Distribution of Winners Across Game Levels', fontsize=14)

annotate\_barplot(ax)

plt.legend(title='Winner')

plt.show()

# Additional categorical columns if needed

# Example: Distribution of Nicknames across levels

plt.figure(figsize=(12, 6))

ax = sns.countplot(x='Level', hue='Nickname', data=filtered\_df, palette='Set3')

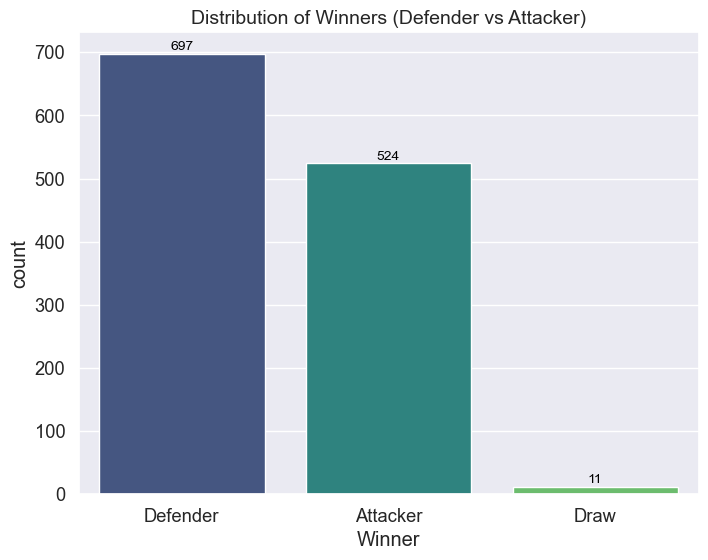
plt.title(f'Distribution of Top {top\_n} Nicknames Across Game Levels', fontsize=14)

plt.xticks(rotation=45, ha='right')

plt.legend(title='Nickname', bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.tight\_layout()

plt.show()



A graph of a distribution of game levels

Description automatically generated

A graph showing different colored bars

Description automatically generated

A graph with numbers and a bar

Description automatically generated

A graph of different colored bars

Description automatically generated

**Training the dataset**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# If `Score\_Difference` was the last column, remove it before splitting

df = df.drop(columns=['Score\_Difference'])

# Separate features (X) and the target variable (y)

# Assuming the target variable is now the last column after removing `Score\_Difference`

X = df.iloc[:, :-1]  # All rows, all columns except the last one (features)

y = df.iloc[:, -1]   # All rows, only the last column (target variable)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

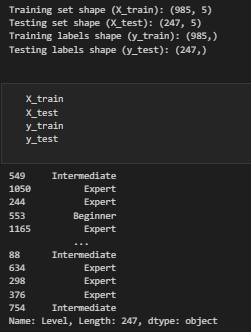
# Display the shapes of the resulting datasets

print("Training set shape (X\_train):", X\_train.shape)

print("Testing set shape (X\_test):", X\_test.shape)

print("Training labels shape (y\_train):", y\_train.shape)

print("Testing labels shape (y\_test):", y\_test.shape)

****

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier, StackingClassifier

from sklearn.svm import LinearSVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import LabelEncoder

from sklearn.impute import SimpleImputer

import warnings

# Suppress warnings for cleaner output

warnings.filterwarnings('ignore')

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize the models

rf = RandomForestClassifier(random\_state=42)

svm = LinearSVC(random\_state=42)

nb = GaussianNB()

# Stacking Classifier

stacking\_model = StackingClassifier(

    estimators=[('rf', rf), ('svm', svm), ('nb', nb)],

    final\_estimator=DecisionTreeClassifier(random\_state=42)

)

# Fit the stacking model on the training data

stacking\_model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = stacking\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

# Display the evaluation metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

# Confusion Matrix and Classification Report

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

A screenshot of a computer

Description automatically generated

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize and train the Random Forest model

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

# Make predictions on the test data

rf\_pred = rf.predict(X\_test)

# Evaluate the model

rf\_accuracy = accuracy\_score(rf\_pred, y\_test)

rf\_report = classification\_report(rf\_pred, y\_test)

rf\_matrix = confusion\_matrix(rf\_pred, y\_test)

# Print evaluation metrics

print('Accuracy of Random Forest : ', round(rf\_accuracy, 3))

print('Classification report of Random Forest : \n', rf\_report)

print('Confusion Matrix of Random Forest : \n', rf\_matrix)

A screenshot of a computer

Description automatically generated

import pandas as pd

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_curve, roc\_auc\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize and train the LinearSVC model

svr = LinearSVC(random\_state=42)

svr.fit(X\_train, y\_train)

# Make predictions on the test data

svr\_pred = svr.predict(X\_test)

# Evaluate the model

svr\_accuracy = accuracy\_score(svr\_pred, y\_test)

svr\_report = classification\_report(svr\_pred, y\_test)

svr\_matrix = confusion\_matrix(svr\_pred, y\_test)

# Print evaluation metrics

print('Accuracy of SVM : ', round(svr\_accuracy, 3))

print('Classification report of SVM : \n', svr\_report)

print('Confusion Matrix of SVM :\n', svr\_matrix)

# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(svr\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label\_encoders['Level'].classes\_, yticklabels=label\_encoders['Level'].classes\_)

plt.title('Confusion Matrix Heatmap')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# ROC Curve (for binary classification)

if len(label\_encoders['Level'].classes\_) == 2:

    y\_score = svr.decision\_function(X\_test)

    fpr, tpr, \_ = roc\_curve(y\_test, y\_score)

    auc\_score = roc\_auc\_score(y\_test, y\_score)

    # Plot the ROC Curve

    plt.figure(figsize=(8, 6))

    plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc\_score:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for a random classifier

    plt.title('Receiver Operating Characteristic (ROC) Curve')

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.legend(loc='lower right')

    plt.show()

else:

    print("ROC Curve is only applicable for binary classification.")

A screenshot of a computer screen

Description automatically generated

A diagram of a heatmap

Description automatically generated

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize and train the Naive Bayes model

nb = GaussianNB()

nb.fit(X\_train, y\_train)

# Make predictions on the test data

nb\_pred = nb.predict(X\_test)

# Evaluate the model

nb\_accuracy = accuracy\_score(nb\_pred, y\_test)

nb\_report = classification\_report(nb\_pred, y\_test, output\_dict=True)

nb\_matrix = confusion\_matrix(nb\_pred, y\_test)

# Extract support for all classes

labels = [str(label) for label in np.unique(np.concatenate((nb\_pred, y\_test)))]

support = [nb\_report[label]['support'] if label in nb\_report else 0 for label in labels]

# Print evaluation metrics

print('Accuracy of Naive Bayes : ', round(nb\_accuracy, 3))

print('Classification report of Naive Bayes : \n', classification\_report(nb\_pred, y\_test))

print('Confusion Matrix of Naive Bayes :\n', nb\_matrix)

# Plot support for each class

plt.figure(figsize=(10, 6))

plt.bar(labels, support, width=0.4, color='skyblue', label='Support', align='center')

plt.xlabel('Class')

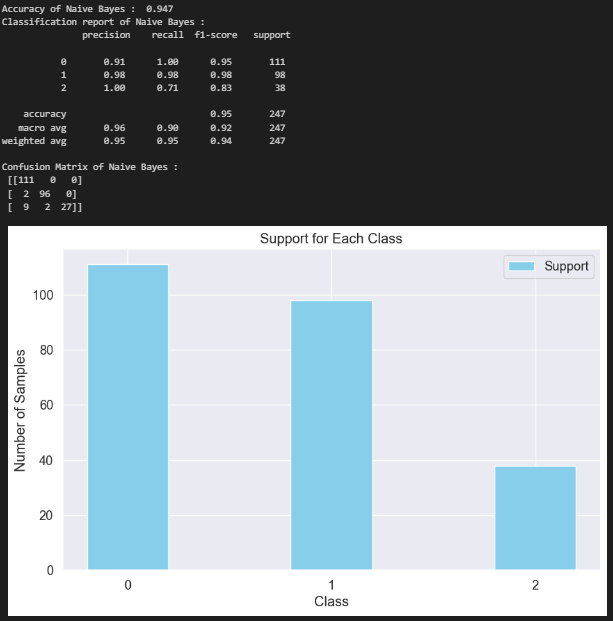
plt.ylabel('Number of Samples')

plt.xticks(labels)

plt.legend()

plt.title('Support for Each Class')

plt.show()



import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize and train the Naive Bayes model

nb = GaussianNB()

nb.fit(X\_train, y\_train)

# Make predictions on the test data

nb\_pred = nb.predict(X\_test)

# Evaluate the model

nb\_accuracy = accuracy\_score(nb\_pred, y\_test)

nb\_report = classification\_report(nb\_pred, y\_test, output\_dict=True)

nb\_matrix = confusion\_matrix(nb\_pred, y\_test)

# Extract support for all classes

labels = [str(label) for label in np.unique(np.concatenate((nb\_pred, y\_test)))]

support = [nb\_report[label]['support'] if label in nb\_report else 0 for label in labels]

# Print evaluation metrics

print('Accuracy of Naive Bayes : ', round(nb\_accuracy, 3))

print('Classification report of Naive Bayes : \n', classification\_report(nb\_pred, y\_test))

print('Confusion Matrix of Naive Bayes :\n', nb\_matrix)

# Plot support for each class

plt.figure(figsize=(10, 6))

# Use color palette for better visualization

colors = plt.cm.Paired(np.linspace(0, 1, len(labels)))

# Bar plot with improved styling

plt.bar(labels, support, width=0.5, color=colors, edgecolor='black', label='Support')

# Add data labels on top of the bars

for i, val in enumerate(support):

    plt.text(i, val + 1, str(val), ha='center', va='bottom', fontsize=10, fontweight='bold')

# Add clear axis labels and title

plt.xlabel('Class Label (Level)', fontsize=12, fontweight='bold')

plt.ylabel('Number of Instances (Support)', fontsize=12, fontweight='bold')

plt.xticks(labels, fontsize=10, fontweight='bold')

plt.yticks(fontsize=10)

plt.title('Distribution of Instances Across Class Levels', fontsize=14, fontweight='bold')

# Add grid lines to help visualize the support levels

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

A chart with a number of bars

Description automatically generated with medium confidence

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Initialize and train the Naive Bayes model

nb = GaussianNB()

nb.fit(X\_train, y\_train)

# Make predictions on the test data

nb\_pred = nb.predict(X\_test)

# Evaluate the model

nb\_accuracy = accuracy\_score(nb\_pred, y\_test)

nb\_report = classification\_report(nb\_pred, y\_test, output\_dict=True)

nb\_matrix = confusion\_matrix(nb\_pred, y\_test)

# Extract precision and recall for all classes

labels = [str(label) for label in np.unique(np.concatenate((nb\_pred, y\_test)))]  # Get all unique labels

precision = [nb\_report[label]['precision'] if label in nb\_report else 0.0 for label in labels]

recall = [nb\_report[label]['recall'] if label in nb\_report else 0.0 for label in labels]

# Print evaluation metrics

print('Accuracy of Naive Bayes : ', round(nb\_accuracy, 3))

print('Classification report of Naive Bayes : \n', classification\_report(nb\_pred, y\_test))

# Plot precision and recall

plt.figure(figsize=(10, 6))

# Bar plot with improved styling

bar\_width = 0.35

index = np.arange(len(labels))

plt.bar(index, precision, bar\_width, label='Precision', color='skyblue', edgecolor='black')

plt.bar(index + bar\_width, recall, bar\_width, label='Recall', color='orange', edgecolor='black')

# Add data labels on top of the bars

for i in range(len(labels)):

    plt.text(i, precision[i] + 0.02, f'{precision[i]:.2f}', ha='center', va='bottom', fontsize=10, fontweight='bold')

    plt.text(i + bar\_width, recall[i] + 0.02, f'{recall[i]:.2f}', ha='center', va='bottom', fontsize=10, fontweight='bold')

# Add clear axis labels and title

plt.xlabel('Class Levels Predicted by Naive Bayes Model', fontsize=12, fontweight='bold')

plt.ylabel('Scores for Precision and Recall', fontsize=12, fontweight='bold')

plt.xticks(index + bar\_width / 2, labels, fontsize=10, fontweight='bold')

plt.yticks(fontsize=10)

plt.title('Comparison of Precision and Recall for Predicted Class Levels', fontsize=14, fontweight='bold')

# Add a legend and grid

plt.legend()

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

A screenshot of a graph

Description automatically generated

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier, StackingClassifier

from sklearn.svm import LinearSVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

file\_path = 'Cleaned\_Game\_Analytics\_Dataset.csv'  # Replace with your actual file path

df = pd.read\_csv(file\_path)

# Identify categorical columns in X

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Apply Label Encoding to categorical columns

label\_encoders = {}

for col in categorical\_cols:

    label\_encoders[col] = LabelEncoder()

    df[col] = label\_encoders[col].fit\_transform(df[col])

# Separate features (X) and the target variable (y)

X = df.drop(columns=['Level'])  # All columns except 'Level'

y = df['Level']  # Target variable

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

# Split the dataset into training (80%) and testing (20%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.8, random\_state=42)

# Define the estimators for the stacking classifier

estimators = [

    ('rf', RandomForestClassifier(n\_estimators=1000, random\_state=42)),

    ('svr', LinearSVC(random\_state=42))

]

# Create the stacking classifier

clf = StackingClassifier(

    estimators=estimators,

    final\_estimator=GaussianNB()

)

# Train the stacking classifier

clf.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = clf.predict(X\_test)

# Evaluate the model

clf\_accuracy = accuracy\_score(y\_pred, y\_test)

clf\_report = classification\_report(y\_pred, y\_test)

clf\_matrix = confusion\_matrix(y\_pred, y\_test)

# Print evaluation metrics

print('Accuracy of Stacking Classifier : ', round(clf\_accuracy, 3))

print('Classification report of Stacking Classifier : \n', clf\_report)

print('Confusion Matrix of Stacking Classifier : \n', clf\_matrix)

# Plot the confusion matrix as a heatmap with more descriptive axes

plt.figure(figsize=(10, 8))

sns.set(font\_scale=1.2)

# You can customize class names based on your dataset

class\_names = ['Beginner', 'Intermediate', 'Expert']

sns.heatmap(clf\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,

            xticklabels=class\_names, yticklabels=class\_names)

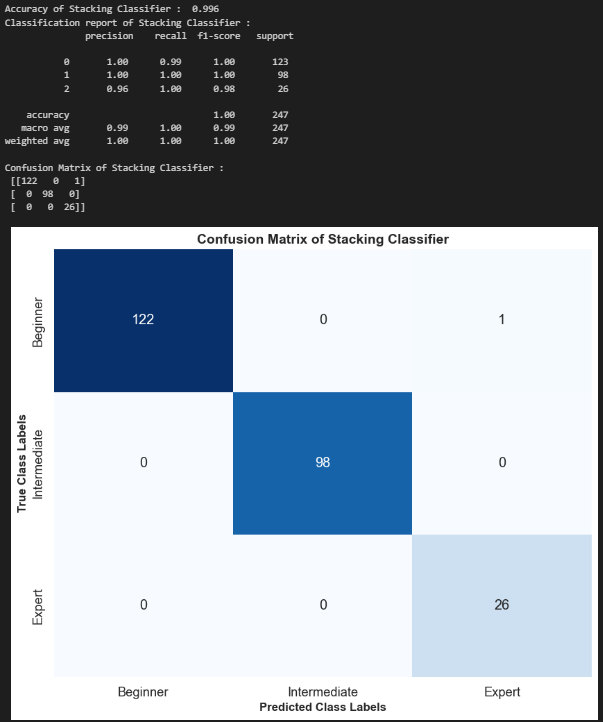
# Provide more descriptive axis labels and title

plt.xlabel("Predicted Class Labels", fontsize=12, fontweight='bold')

plt.ylabel("True Class Labels", fontsize=12, fontweight='bold')

plt.title("Confusion Matrix of Stacking Classifier", fontsize=14, fontweight='bold')

plt.show()



import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Assuming you already have pred and y\_test defined for your Ensemble Model

# Calculate accuracy, confusion matrix, and classification report

eb\_accuracy = accuracy\_score(pred, y\_test)

eb\_matrix = confusion\_matrix(pred, y\_test)

eb\_report = classification\_report(pred, y\_test)

# Print the results

print('Accuracy of Ensemble Model: ', round(eb\_accuracy, 3))

print('Confusion Matrix of Ensemble Model:\n', eb\_matrix)

print('Classification Report of Ensemble Model:\n', eb\_report)

# Plot the confusion matrix as a heatmap with more descriptive axis labels

plt.figure(figsize=(10, 8))

sns.set(font\_scale=1.2)  # Adjust the font size for better readability

# Customize class labels based on your dataset

class\_names = ['Beginner', 'Intermediate', 'Expert']

sns.heatmap(eb\_matrix, annot=True, fmt="d", cmap="Blues", cbar=False,

            xticklabels=class\_names, yticklabels=class\_names)

# Provide more descriptive axis labels and title

plt.xlabel("Predicted Class Labels", fontsize=12, fontweight='bold')

plt.ylabel("True Class Labels", fontsize=12, fontweight='bold')

plt.title("Confusion Matrix Heatmap for Ensemble Model", fontsize=14, fontweight='bold')

plt.show()

A screenshot of a computer

Description automatically generated

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# Assuming the models are already trained and you have X\_test and y\_test

# Make predictions with each model

# Random Forest

pred\_RF = rf.predict(X\_test)

# SVM

pred\_SVM = svr.predict(X\_test)

# Naive Bayes

pred\_NB = nb.predict(X\_test)

# Ensemble (Stacking Classifier)

pred\_Ensemble = clf.predict(X\_test)

# Assuming y\_test and pred\_RF, pred\_SVM, pred\_NB, and pred\_Ensemble are the predictions from your models

# Random Forest

precision\_rf = precision\_score(y\_test, pred\_RF, average='weighted')

recall\_rf = recall\_score(y\_test, pred\_RF, average='weighted')

f1\_rf = f1\_score(y\_test, pred\_RF, average='weighted')

# SVM

precision\_svm = precision\_score(y\_test, pred\_SVM, average='weighted')

recall\_svm = recall\_score(y\_test, pred\_SVM, average='weighted')

f1\_svm = f1\_score(y\_test, pred\_SVM, average='weighted')

# Naive Bayes

precision\_nb = precision\_score(y\_test, pred\_NB, average='weighted')

recall\_nb = recall\_score(y\_test, pred\_NB, average='weighted')

f1\_nb = f1\_score(y\_test, pred\_NB, average='weighted')

# Ensemble

precision\_ensemble = precision\_score(y\_test, pred\_Ensemble, average='weighted')

recall\_ensemble = recall\_score(y\_test, pred\_Ensemble, average='weighted')

f1\_ensemble = f1\_score(y\_test, pred\_Ensemble, average='weighted')

# Aggregate metrics

precision = [precision\_rf, precision\_svm, precision\_nb, precision\_ensemble]

recall = [recall\_rf, recall\_svm, recall\_nb, recall\_ensemble]

f1\_scores = [f1\_rf, f1\_svm, f1\_nb, f1\_ensemble]

import matplotlib.pyplot as plt

# Model names

models = ['Random Forest', 'SVM', 'Naive Bayes', 'Ensemble Learning']

# Convert metrics to percentages if necessary

precision = [p \* 100 for p in precision]

recall = [r \* 100 for r in recall]

f1\_scores = [f \* 100 for f in f1\_scores]

# X-axis values (models)

x = range(len(models))

# Create a figure and axis for the plot

fig, ax = plt.subplots(figsize=(14, 8))

# Plot precision scores with markers and line style for better readability

ax.plot(x, precision, marker='o', markersize=8, linestyle='-', linewidth=2, color='blue', label='Precision')

# Plot recall scores with markers and line style for better readability

ax.plot(x, recall, marker='s', markersize=8, linestyle='--', linewidth=2, color='green', label='Recall')

# Plot F1-score scores with markers and line style for better readability

ax.plot(x, f1\_scores, marker='D', markersize=8, linestyle=':', linewidth=2, color='red', label='F1-Score')

# Set x-axis ticks and labels

ax.set\_xticks(x)

ax.set\_xticklabels(models, rotation=45, fontsize=14)

ax.set\_xlabel('Machine Learning Models', fontsize=16, fontweight='bold')

# Set y-axis label

ax.set\_ylabel('Scores (%)', fontsize=16, fontweight='bold')

# Set plot title with increased font size and weight

ax.set\_title('Comparison of Precision, Recall, and F1-Score Across Models', fontsize=18, fontweight='bold')

# Add a legend with larger font size

ax.legend(fontsize=14)

# Add grid for better readability with a lighter color and style

ax.grid(True, linestyle='--', alpha=0.7, color='gray')

# Set y-axis range to zoom in on the graph, e.g., 50 to 100

ax.set\_ylim(50, 100)

# Improve layout to avoid clipping of tick labels and title

plt.tight\_layout()

# Show the plot

plt.show()

A graph showing the difference between precision and precision

Description automatically generated

import pandas as pd

from lazypredict.Supervised import LazyClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

# Load your cleaned data (replace 'Cleaned\_Game\_Analytics\_Dataset.csv' with your actual file path)

df = pd.read\_csv('Cleaned\_Game\_Analytics\_Dataset.csv')

# Preprocessing

# If 'Winner' is categorical, encode it into numeric values

le = LabelEncoder()

df['Winner'] = le.fit\_transform(df['Winner'])  # Assuming 'Defender' = 1 and 'Attacker' = 0

# Define X (features) and y (target)

X = df[['Defender\_Score', 'Attacker\_Score', 'Time\_in\_seconds']]  # Features

y = df['Winner']  # Target variable (encoded)

# Split the dataset into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize LazyClassifier

clf = LazyClassifier(verbose=0, ignore\_warnings=True, custom\_metric=None)

# Fit LazyClassifier on the data

models, predictions = clf.fit(X\_train, X\_test, y\_train, y\_test)

# Display the results of model comparison

print(models)

# Ensure metrics are numeric and handle missing data

metrics\_to\_display = models[['Accuracy', 'Balanced Accuracy', 'F1 Score', 'ROC AUC']].apply(pd.to\_numeric, errors='coerce')

# Check if there are any NaN values

print("Missing values in metrics to display:\n", metrics\_to\_display.isnull().sum())

# Fill NaN values with a placeholder (e.g., 0.0) to ensure the heatmap can be plotted

metrics\_to\_display.fillna(0.0, inplace=True)

# 1. Plot: Accuracy of models (Bar plot)

plt.figure(figsize=(10, 5))

sns.barplot(x=models.index, y='Accuracy', data=models)

plt.xticks(rotation=90)

plt.title("Accuracy of Various Models")

plt.xlabel('Models')

plt.ylabel('Accuracy')

plt.show()

# 2. Plot: Heatmap of different metrics

# Plot heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(metrics\_to\_display, annot=True, cmap='coolwarm', fmt=".3f")

plt.title('Model Performance Metrics (Accuracy, Balanced Accuracy, F1, ROC AUC)')

plt.show()

# If you want to save the model comparison to a CSV for further analysis

models.to\_csv('model\_comparison\_results.csv', index=False)

A screenshot of a computer

Description automatically generated

A chart with different colored lines

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated