G.v.s Kowshik AM.EN.U4CSE21271

```
pip install Faker;
```

```
Collecting Faker
  Downloading Faker-22.0.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 13.5 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/dist-packages (from Faker) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.4->Faker) (1.16.0)
Installing collected packages: Faker
Successfully installed Faker-22.0.0
```

```
from faker import Faker
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.ensemble import IsolationForest
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
```

Data Set Creation using Faker(Python)

```


fake = Faker()
match_data_list = []

num_matches = 1000
K_weight = 2  # Weight for kills
W_weight = 5  # Weight for wins

for match_id in range(num_matches):
    # Generate random data
    wins = np.random.randint(1, 10)
    loses = 10 - wins
    wl_ratio = wins / loses
    avg_kills = np.random.randint(0, 999)
    avg_deaths = np.random.randint(0, 999)
    avg_assists = np.random.randint(0, 555)
    avg_kdr = avg_kills / (avg_deaths + 1)  # Adding 1 to avoid division by zero

    # Introduce null values randomly in some columns
    if np.random.rand() < 0.1:  # 10% chance of having null values
        wins = None
    if np.random.rand() < 0.1:
        avg_kills = None
    if np.random.rand() < 0.05:  # 5% chance for additional columns
        avg_deaths = None
        avg_assists = None

    # Generate a random device type (e.g., 'PC', 'Console', 'Mobile')
    device_played = fake.random_element(elements=('PC', 'Console', 'Mobile'))

    # Calculate the score using Method 1, handle None values
    if avg_kills is not None and wins is not None:
        score = (avg_kills * K_weight) + (wins * W_weight)
    else:
        score = None

    match_data = {
        'Match_id': fake.building_number(),
        'Wins': wins,
        'Loses': loses,
        'WL_Ratio': wl_ratio,
        'Avg_Kills': avg_kills,
        'Avg_Deaths': avg_deaths,
        'Avg_Assists': avg_assists,
        'Avg_KDR': avg_kdr,
        'Device_Played': device_played,  # Include the new categorical feature
        'Score': score
    }

    match_data_list.append(match_data)

# Convert data to DataFrame
```

```
df = pd.DataFrame(match_data_list)

# Save to CSV
df.to_csv('Data_set.csv', index=False)
```

```
print("Head of the dataset:")
print(df.head())
```

```
Head of the dataset:
  Match_id  Wins  Loses  WL_Ratio  Avg_Kills  Avg_Deaths  Avg_Assists  \
0     2358   4.0      6  0.666667      245.0       440.0        298.0   
1    71345   NaN      7  0.428571      877.0       606.0        172.0   
2     3763   9.0      1  9.000000      368.0       382.0         21.0   
3     3173   1.0      9  0.111111      989.0       994.0        490.0   
4      084   8.0      2  4.000000      753.0       792.0         83.0   

    Avg_KDR Device_Played   Score  
0  0.555556        Mobile   510.0  
1  1.444811       Console     NaN  
2  0.960836       Console   781.0  
3  0.993970       Console  1983.0  
4  0.949559            PC  1546.0  
```

General Information About the Data Set

```
# Display general information about the dataset
print("\nGeneral information about the dataset:")
print(df.info())
```

```
General information about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Match_id       1000 non-null   object 
 1   Wins           896 non-null    float64
 2   Loses          1000 non-null   int64  
 3   WL_Ratio       1000 non-null   float64
 4   Avg_Kills      908 non-null    float64
 5   Avg_Deaths     948 non-null    float64
 6   Avg_Assists    948 non-null    float64
 7   Avg_KDR        1000 non-null   float64
 8   Device_Played  1000 non-null   object 
 9   Score          815 non-null    float64
dtypes: float64(7), int64(1), object(2)
memory usage: 78.2+ KB
None
```

Size of the Dataset

```
# Number of rows and columns
num_rows, num_columns = df.shape
print("\nNumber of rows:", num_rows)
print("Number of columns:", num_columns)
```

```
Number of rows: 1000
Number of columns: 10
```

Types of Data Present in the Data set

```
# Data types of each column
print("\nData types of each column:")
print(df.dtypes)
```

```
Data types of each column:
Match_id          object
Wins             float64
Loses              int64
WL_Ratio         float64
Avg_Kills        float64
Avg_Deaths       float64
Avg_Assists      float64
Avg_KDR          float64
Device_Played     object
```

```
Score            float64
dtype: object
```

Stats of Data

```
df.describe()
```

	Wins	Loses	WL_Ratio	Avg_Kills	Avg_Deaths	Avg_Assists	Avg_KDR	Score
count	896.000000	1000.000000	1000.000000	908.000000	948.000000	948.000000	1000.000000	815.000000
mean	4.972098	5.006000	2.148623	495.511013	489.602321	275.599156	4.142303	1012.224540
std	2.608603	2.602833	2.697721	280.366874	292.670281	160.162687	28.094053	563.729872
min	1.000000	1.000000	0.111111	1.000000	0.000000	0.000000	0.000000	17.000000
25%	3.000000	3.000000	0.428571	257.750000	221.750000	137.250000	0.495351	537.000000
50%	5.000000	5.000000	1.000000	494.500000	498.000000	273.500000	0.977651	1016.000000
75%	7.000000	7.000000	2.333333	725.250000	743.000000	412.250000	2.069180	1469.000000
max	9.000000	9.000000	9.000000	998.000000	998.000000	553.000000	831.000000	2036.000000

Checking whether the data set has null values or not

```
# Missing values
missing_values = df.isnull().sum()
print("\nMissing values per column:")
print(missing_values)
```

```
Missing values per column:
Match_id           0
Wins             104
Loses              0
WL_Ratio           0
Avg_Kills         92
Avg_Deaths        52
Avg_Assists       52
Avg_KDR            0
Device_Played      0
Score            185
dtype: int64
```

```
#checking the data set whether it have any null values or null fields or not
df.isnull()
```

	Match_id	Wins	Loses	WL_Ratio	Avg_Kills	Avg_Deaths	Avg_Assists	Avg_KDR	Device_Played	Score
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	False	False	False	False
996	False	True	False	False	False	False	False	False	False	True
997	False	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False	False	False
999	False	False	False	False	False	False	False	False	False	False

1000 rows × 10 columns

Finding Mean, Median & Mode of Data set along with Null values

```
import matplotlib.pyplot as plt

# Assuming you have a DataFrame 'df' defined somewhere in your code

mean_values = df.mean()
median_values = df.median()
mode_values = df.mode().iloc[0]  # Mode may return multiple values, so we take the first one

print("Mean Values of each Column:")
print(mean_values)
print("\n")

print("Median Values of each Column:")
print(median_values)
print("\n")

print("Mode Values of each Column:")
print(mode_values)
```

```
Mean Values of each Column:
Match_id               inf
Wins              4.972098
Loses             5.006000
WL_Ratio          2.148623
Avg_Kills       495.511013
Avg_Deaths      489.602321
Avg_Assists     275.599156
Avg_KDR           4.142303
Score          1012.224540
dtype: float64


Median Values of each Column:
Match_id       4332.500000
Wins              5.000000
Loses             5.000000
WL_Ratio          1.000000
Avg_Kills       494.500000
Avg_Deaths      498.000000
Avg_Assists     273.500000
Avg_KDR           0.977651
Score          1016.000000
dtype: float64


Mode Values of each Column:
Match_id              024
Wins                  1.0
Loses                 5.0
WL_Ratio              1.0
Avg_Kills           170.0
Avg_Deaths          412.0
Avg_Assists          85.0
Avg_KDR          0.102941
Device_Played      Mobile
Score              1816.0
Name: 0, dtype: object
<ipython-input-12-e0c3425f383f>:5: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ver
  mean_values = df.mean()
<ipython-input-12-e0c3425f383f>:6: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future v
  median_values = df.median()
```

Null Values Imputation

- Using KNN
- Using Mean

```
import pandas as pd
from sklearn.impute import KNNImputer  # Import the KNNImputer class

# Assuming 'df' is your DataFrame
# Select the columns with missing values
columns_to_impute = ['Wins', 'Avg_Kills', 'Avg_Deaths']

# Create a subset DataFrame with the columns to impute and other relevant features
subset_df = df[['Loses', 'WL_Ratio', 'Avg_Assists', 'Avg_KDR', 'Score'] + columns_to_impute]

# Initialize the KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Use the fit_transform method to impute missing values
df_imputed = pd.DataFrame(imputer.fit_transform(subset_df), columns=subset_df.columns)

# Update the original DataFrame with imputed values for the specified columns
df[columns_to_impute] = df_imputed[columns_to_impute]

# Display the DataFrame with imputed values
print(df)
```

```
    Match_id  Wins  Loses  WL_Ratio  Avg_Kills  Avg_Deaths  Avg_Assists  \
0       8800   7.0      3  2.333333      826.0       681.0        228.0
1      94275   1.0      9  0.111111      133.0       485.0        422.0
2        856   8.0      2  4.000000      167.0       198.0        158.0
3        362   1.0      9  0.111111      362.0       931.0        125.0
4       0232   3.0      7  0.428571        1.0       905.0        200.0
..       ...   ...    ...       ...        ...         ...          ...
995     7881   8.0      2  4.000000      430.0       579.0         75.0
996      895   3.8      6  0.666667      289.0       311.0        164.0
997      400   7.0      3  2.333333      457.0       194.0        105.0
998      261   4.0      6  0.666667       63.0       437.0        175.0
999      578   3.0      7  0.428571      510.0       524.0        206.0

      Avg_KDR Device_Played   Score
0    1.211144            PC  1687.0
1    0.273663       Console   271.0
2    0.839196       Console   374.0
3    0.388412        Mobile   729.0
4    0.001104       Console    17.0
..        ...           ...     ...
995  0.741379       Console   900.0
996  0.926282       Console     NaN
997  2.343590        Mobile   949.0
998  0.143836       Console   146.0
999  0.971429       Console  1035.0

[1000 rows x 10 columns]
```

> Imputing Null Values using mentioned methods

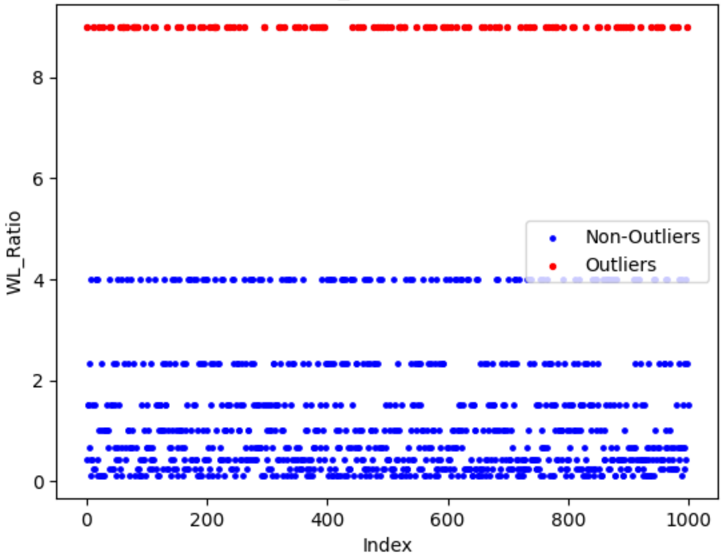
```
#Imputing Avg_assists and Score features using Hot deck implementation

# Select the columns with missing values
columns_to_impute = ['Avg_Assists', 'Score']

# Create a subset DataFrame with the columns to impute
subset_df = df[columns_to_impute]

# Initialize SimpleImputer with a strategy (e.g., 'mean', 'median', 'most_frequent')
imputer = SimpleImputer(strategy='mean')

# Impute missing values in the selected columns
df[columns_to_impute] = imputer.fit_transform(subset_df)
```

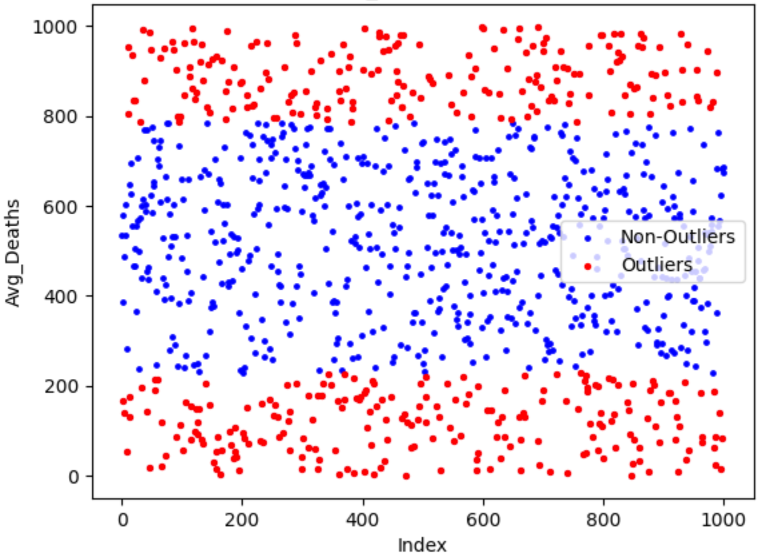
```
df.isnull().sum()
```

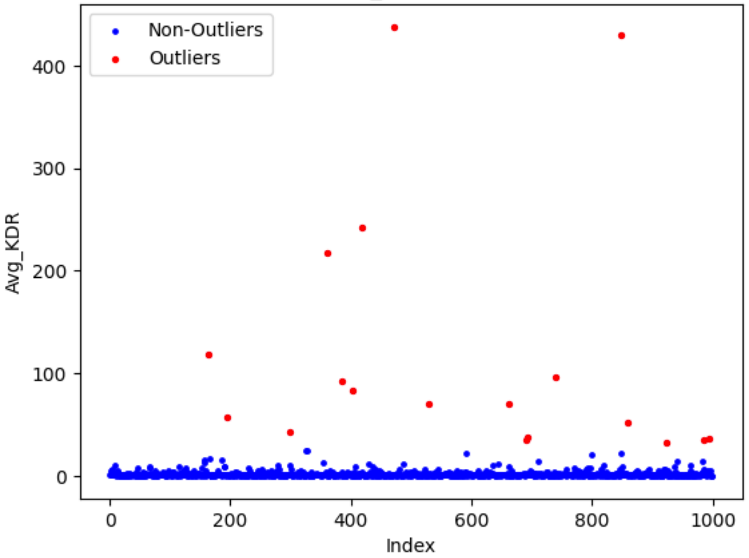
```
Match_id         0
Wins             0
Loses            0
WL_Ratio         0
Avg_Kills        0
Avg_Deaths       0
Avg_Assists      0
Avg_KDR          0
Device_Played    0
Score            0
dtype: int64
```

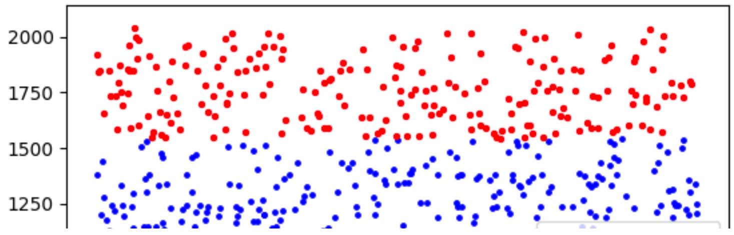
Outlier Detection Using Z-Score Method

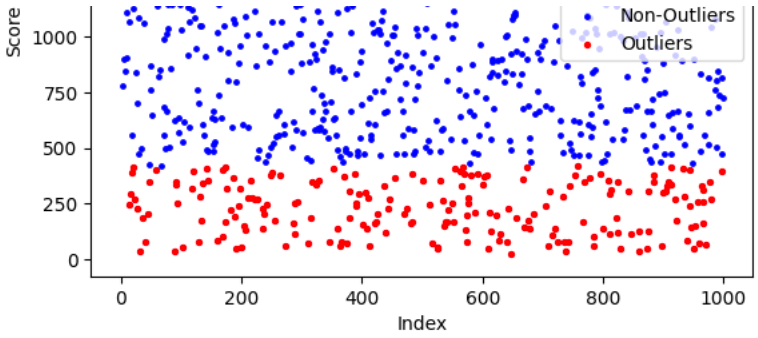
```
import pandas as pd
from scipy.stats import zscore
import matplotlib.pyplot as plt
z_score_columns = ['WL_Ratio', 'Avg_Deaths', 'Avg_KDR', 'Score']

# Define a threshold for considering a data point as an outlier
threshold = 1
# Identify outliers using the Z-score for each specified column
outliers_wl_ratio = df[abs(zscore(df['WL_Ratio'])) > threshold]
outliers_avg_deaths = df[abs(zscore(df['Avg_Deaths'])) > threshold]
outliers_avg_kdr = df[abs(zscore(df['Avg_KDR'])) > threshold]
outliers_score = df[abs(zscore(df['Score'])) > threshold]
# Function to create scatter plots
def create_scatter_plot(x, y, outliers, xlabel, ylabel, title):
    plt.scatter(df.index, df[y], c='blue', label='Non-Outliers', s=6)
    plt.scatter(outliers.index, outliers[y], c='red', label='Outliers', s=8)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.legend()
    plt.show()
# Scatter Plots with reduced sizes
create_scatter_plot('Index', 'WL_Ratio', outliers_wl_ratio, 'Index', 'WL_Ratio', 'Scatter Plot for WL_Ratio with Z-score Outliers')
create_scatter_plot('Index', 'Avg_Deaths', outliers_avg_deaths, 'Index', 'Avg_Deaths', 'Scatter Plot for Avg_Deaths with Z-score Outlier
create_scatter_plot('Index', 'Avg_KDR', outliers_avg_kdr, 'Index', 'Avg_KDR', 'Scatter Plot for Avg_KDR with Z-score Outliers')
create_scatter_plot('Index', 'Score', outliers_score, 'Index', 'Score', 'Scatter Plot for Score with Z-score Outliers')
```


Scatter Plot for WL_Ratio with Z-score Outliers


Scatter Plot for Avg_Deaths with Z-score Outliers

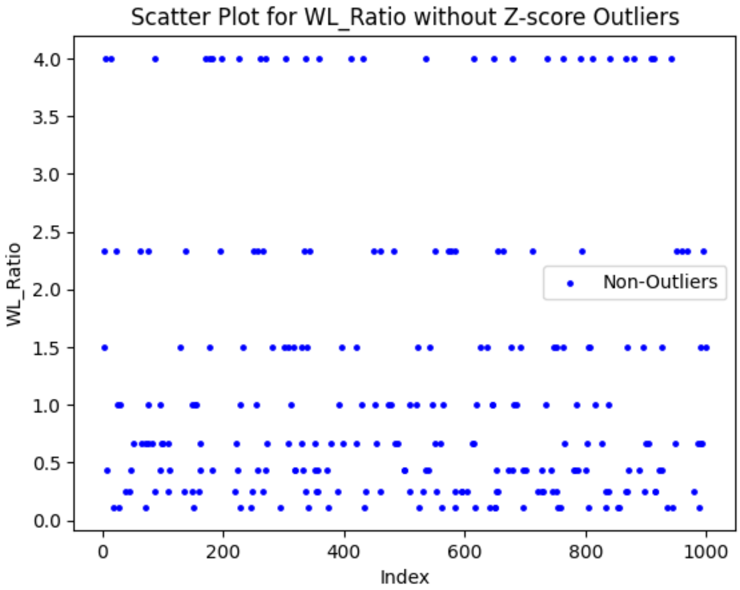

Scatter Plot for Avg_KDR with Z-score Outliers

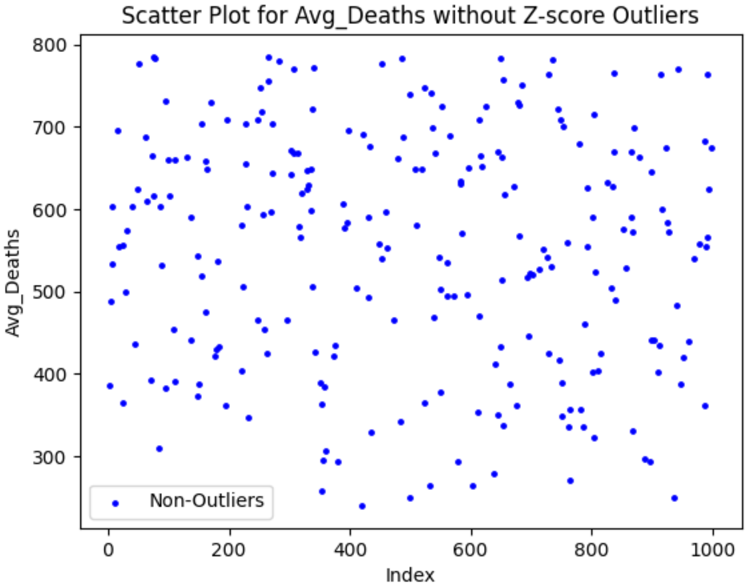

Scatter Plot for Score with Z-score Outliers

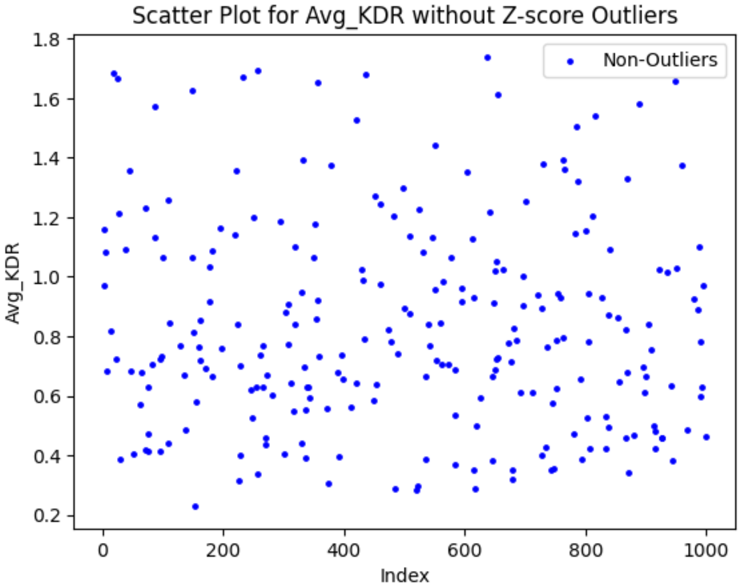


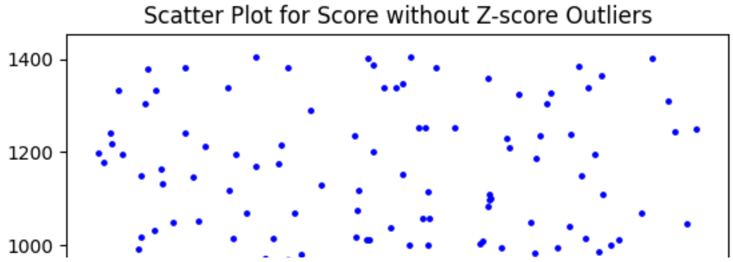
> After Removing Outliers from the data set

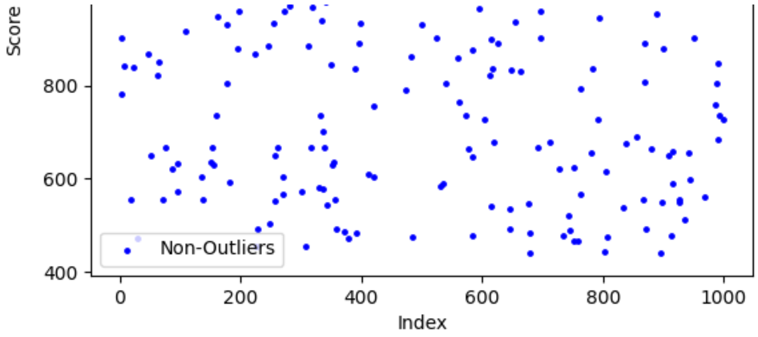
```
import pandas as pd
from scipy.stats import zscore
import matplotlib.pyplot as plt

# Assuming you have the DataFrame 'df' from your code

# Columns for Z-score Outlier Detection
z_score_columns = ['WL_Ratio', 'Avg_Deaths', 'Avg_KDR', 'Score']

# Define a threshold for considering a data point as an outlier
threshold = 1

# Remove outliers for each specified column separately
df_no_outliers = df.copy()
for column in z_score_columns:
    outliers = df_no_outliers[abs(zscore(df_no_outliers[column])) > threshold]
    df_no_outliers = df_no_outliers[abs(zscore(df_no_outliers[column])) <= threshold]

   # # Visualize the scatter plot for each column with outliers removed
   # plt.scatter(df_no_outliers.index, df_no_outliers[column], c='blue', label='Non-Outliers', s=6)
   # plt.scatter(outliers.index, outliers[column], c='red', label='Outliers', s=8)
   # plt.title(f'Scatter Plot for {column} with Z-score Outliers Removed')
   # plt.xlabel('Index')
   # plt.ylabel(column)
   # plt.legend()
   # plt.show()

# Visualize the final DataFrame without outliers
for column in z_score_columns:
    plt.scatter(df_no_outliers.index, df_no_outliers[column], c='blue', label='Non-Outliers', s=6)
    plt.title(f'Scatter Plot for {column} without Z-score Outliers')
    plt.xlabel('Index')
    plt.ylabel(column)
    plt.legend()
    plt.show()
```


Scatter Plot for WL_Ratio without Z-score Outliers
WL_Ratio
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Index
Non-Outliers


Scatter Plot for Avg_Deaths without Z-score Outliers
Avg_Deaths
800
700
600
500
400
300
0
200
400
600
800
1000
Index
Non-Outliers


Scatter Plot for Avg_KDR without Z-score Outliers
Avg_KDR
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
Index
Non-Outliers


Scatter Plot for Score without Z-score Outliers
1400
1200
1000



Normalization

```
# Select columns to normalize
columns_to_normalize = ['Wins', 'Loses', 'WL_Ratio', 'Avg_KDR', 'Score']
scaler = MinMaxScaler()
# Normalize the selected columns in the new DataFrame
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
print(df.head)
```

```
<bound method NDFrame.head of     Match_id   Wins  Loses  WL_Ratio  Avg_Kills  Avg_Deaths  Avg_Assists  \
0       8800  0.750  0.250  0.250000      826.0       681.0        228.0
1      94275  0.000  1.000  0.000000      133.0       485.0        422.0
2        856  0.875  0.125  0.437500      167.0       198.0        158.0
3        362  0.000  1.000  0.000000      362.0       931.0        125.0
4       0232  0.250  0.750  0.035714        1.0       905.0        200.0
..       ...    ...    ...       ...        ...         ...          ...
995     7881  0.875  0.125  0.437500      430.0       579.0         75.0
996      895  0.350  0.625  0.062500      289.0       311.0        164.0
997      400  0.750  0.250  0.250000      457.0       194.0        105.0
998      261  0.375  0.625  0.062500       63.0       437.0        175.0
999      578  0.250  0.750  0.035714      510.0       524.0        206.0

      Avg_KDR Device_Played     Score
0    0.001457            PC  0.827142
1    0.000329       Console  0.125805
2    0.001010       Console  0.176820
3    0.000467        Mobile  0.352650
4    0.000001       Console  0.000000
..        ...           ...       ...
995  0.000892       Console  0.437345
996  0.001115       Console  0.492929
997  0.002820        Mobile  0.461615
998  0.000173       Console  0.063893
999  0.001169       Console  0.504210

[1000 rows x 10 columns]>
```

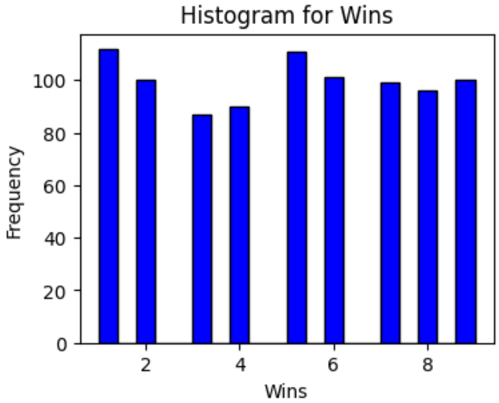
```
# Create a LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'Device_Played' column
df['Device_Played_encoded'] = label_encoder.fit_transform(df['Device_Played'])

# Print the DataFrame with the encoded column
print(df[['Device_Played', 'Device_Played_encoded']])
```

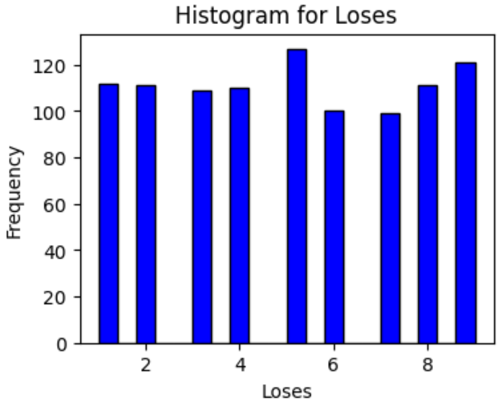
```
    Device_Played  Device_Played_encoded
0              PC                      2
1         Console                      0
2         Console                      0
3          Mobile                      1
4         Console                      0
..            ...                    ...
995       Console                      0
996       Console                      0
997        Mobile                      1
998       Console                      0
999       Console                      0

[1000 rows x 2 columns]
```

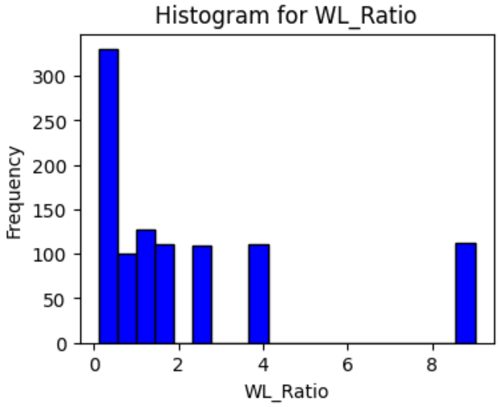
Data Visualization

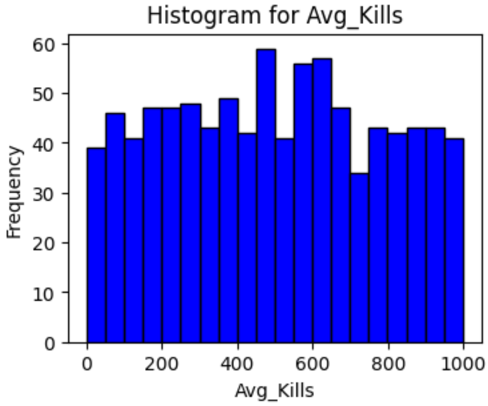
> Histogram

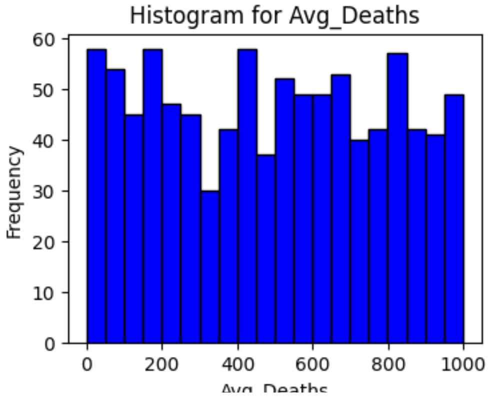
```
# Function to create histograms for each feature
def create_histograms(dataframe):
    features_to_visualize = ['Wins', 'Loses', 'WL_Ratio', 'Avg_Kills', 'Avg_Deaths', 'Avg_Assists', 'Avg_KDR', 'Score']

    for feature in features_to_visualize:
        plt.figure(figsize=(4, 3))
        plt.hist(dataframe[feature].dropna(), bins=20, color='blue', edgecolor='black')  # Drop NA values
        plt.title(f'Histogram for {feature}')
        plt.xlabel(feature)
        plt.ylabel('Frequency')
        plt.show()
    match_data = {
        'Match_id': fake.building_number(),
        'Wins': wins,
        'Loses': loses,
        'WL_Ratio': wl_ratio,
        'Avg_Kills': avg_kills,
        'Avg_Deaths': avg_deaths,
        'Avg_Assists': avg_assists,
        'Avg_KDR': avg_kdr,
        'Device_Played': device_played,  # Include the new categorical feature
        'Score': score
    }

    match_data_list.append(match_data)

# Convert data to DataFrame
df = pd.DataFrame(match_data_list)

# Save to CSV
df.to_csv('Data_set.csv', index=False)

# Create histograms for the features
create_histograms(df)
```

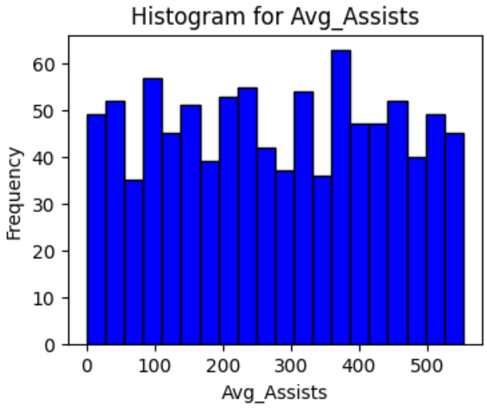

Histogram for Wins
Frequency
0
20
40
60
80
100
2
4
6
8
Wins

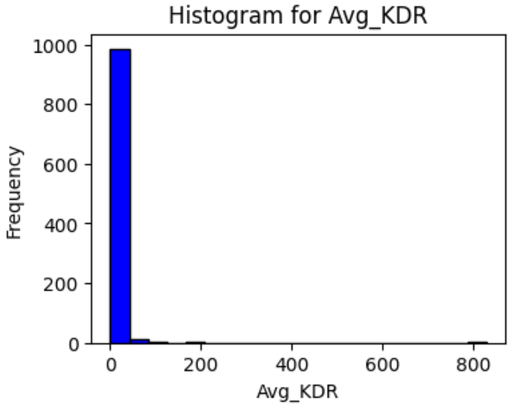

Histogram for Loses
Frequency
0
20
40
60
80
100
120
2
4
6
8
Loses

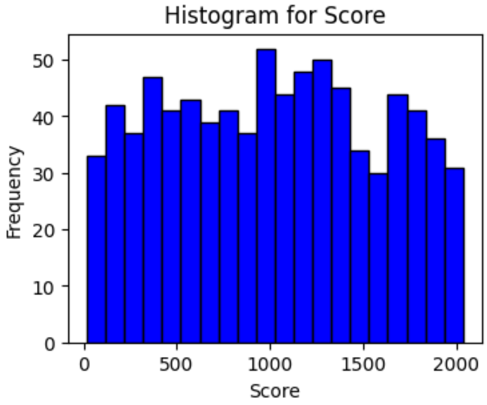

Histogram for WL_Ratio
Frequency
0
50
100
150
200
250
300
0
2
4
6
8
WL_Ratio


Histogram for Avg_Kills
Frequency
0
10
20
30
40
50
60
0
200
400
600
800
1000
Avg_Kills


Histogram for Avg_Deaths
Frequency
0
10
20
30
40
50
60
0
200
400
600
800
1000
Avg_Deaths

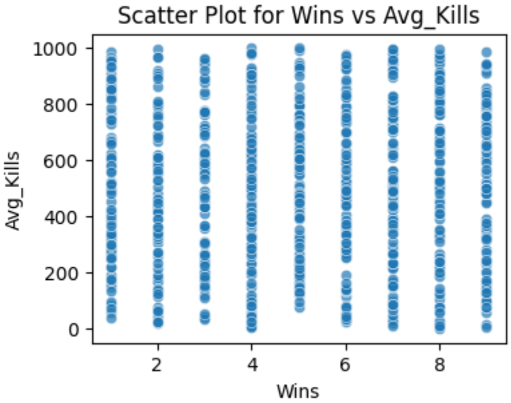


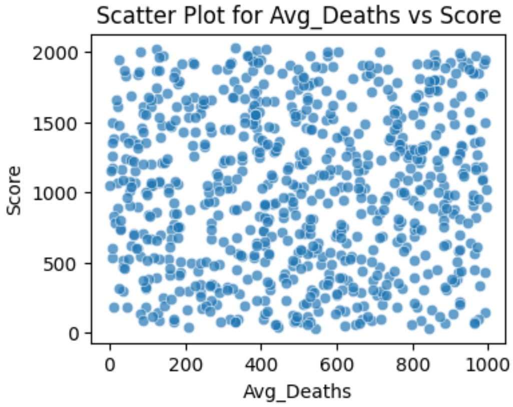


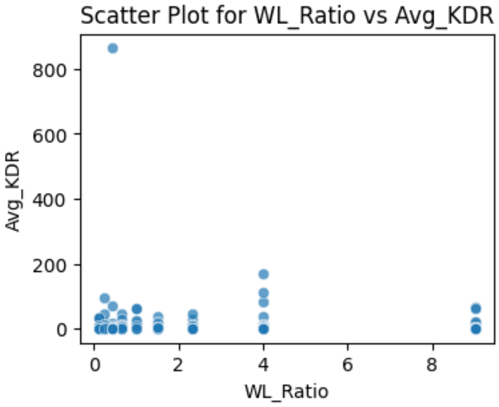


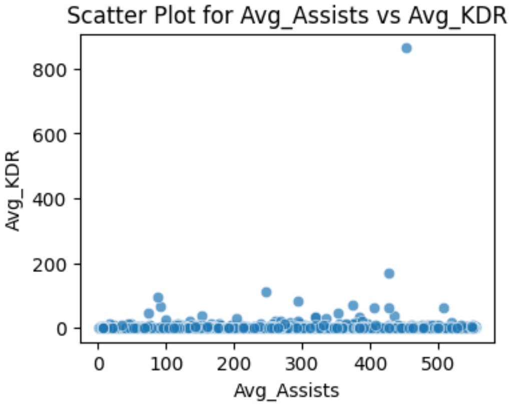
> Scatter Plot

```
import seaborn as sns
import matplotlib.pyplot as plt

# List of numerical features
numerical_features = ['Wins', 'Loses', 'WL_Ratio', 'Avg_Kills', 'Avg_Deaths', 'Avg_Assists', 'Avg_KDR', 'Score']

# Scatter plot for selected numerical feature pairs
selected_scatter_pairs = [('Wins', 'Avg_Kills'), ('Avg_Deaths', 'Score'), ('WL_Ratio', 'Avg_KDR'), ('Avg_Assists', 'Avg_KDR')]

# Function to create scatter plots for selected feature pairs
def create_scatter_plots(dataframe, feature_pairs):
    for pair in feature_pairs:
        plt.figure(figsize=(4, 3))
        sns.scatterplot(x=pair[0], y=pair[1], data=dataframe, alpha=0.7)
        plt.title(f'Scatter Plot for {pair[0]} vs {pair[1]}')
        plt.xlabel(pair[0])
        plt.ylabel(pair[1])
        plt.show()

# Create scatter plots for selected feature pairs
create_scatter_plots(df, selected_scatter_pairs)
```


Scatter Plot for Wins vs Avg_Kills
Avg_Kills
Wins
0
200
400
600
800
1000
2
4
6
8

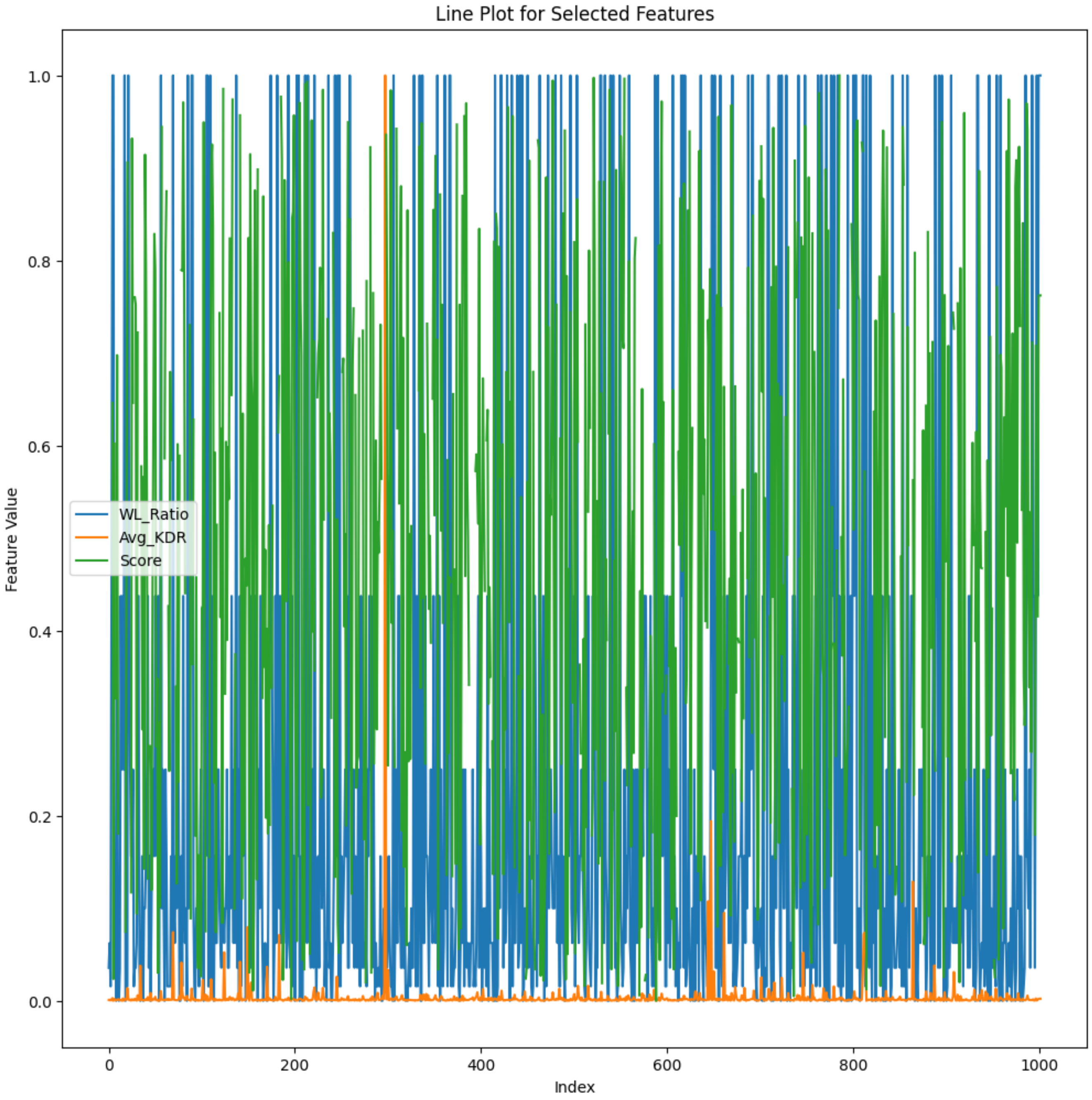

Scatter Plot for Avg_Deaths vs Score
Score
Avg_Deaths
0
500
1000
1500
2000
0
200
400
600
800
1000


Scatter Plot for WL_Ratio vs Avg_KDR
Avg_KDR
WL_Ratio
0
200
400
600
800
0
2
4
6
8


Scatter Plot for Avg_Assists vs Avg_KDR
Avg_KDR
Avg_Assists
0
200
400
600
800
0
100
200
300
400
500

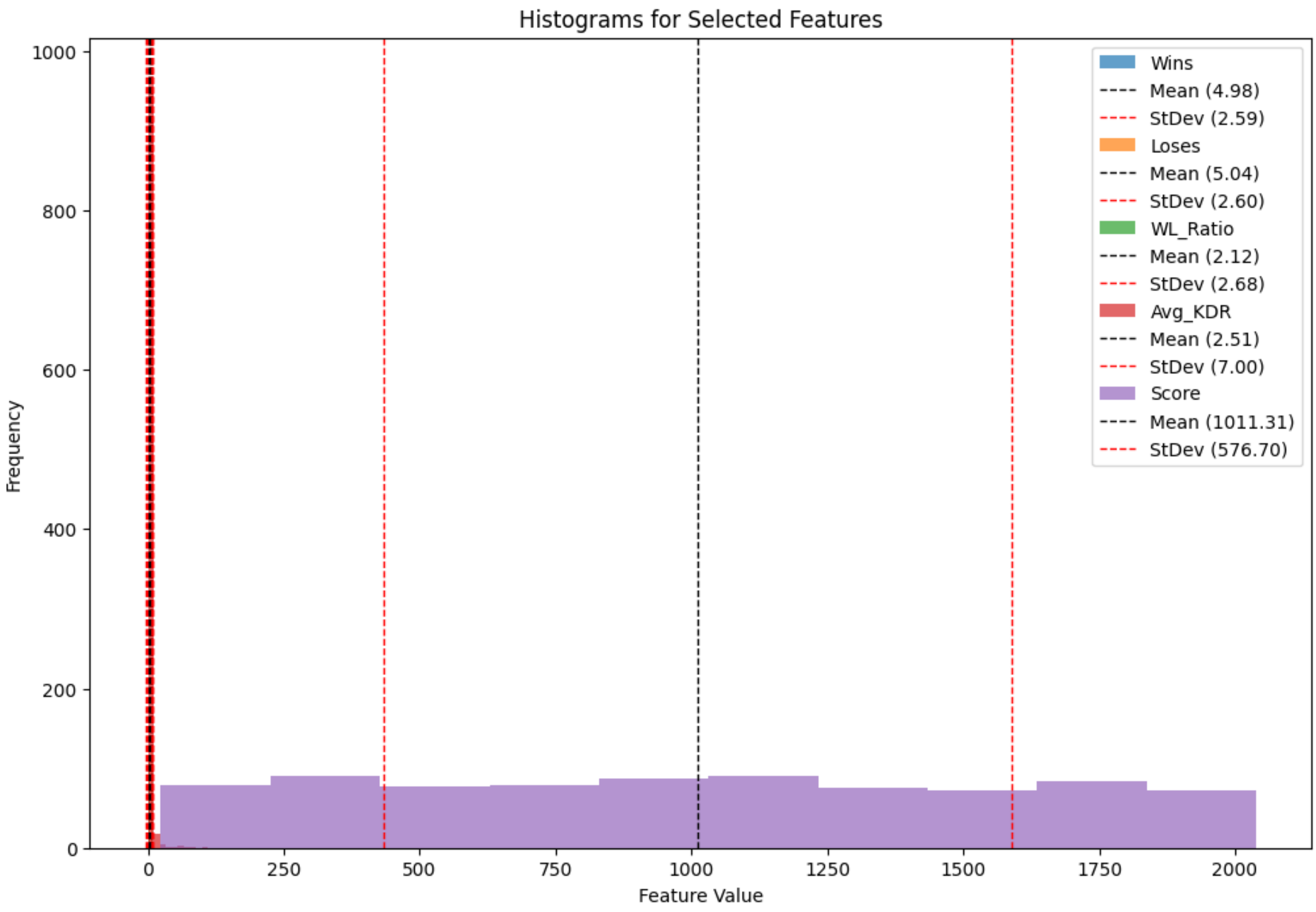
Line Plot

```
# List of features for line plot
features_for_line_plot = ['WL_Ratio',  'Avg_KDR', 'Score']

# Create a line plot for each feature
plt.figure(figsize=(12, 12))
for feature in features_for_line_plot:
    plt.plot(df.index, df[feature], label=feature)

plt.title('Line Plot for Selected Features')
plt.xlabel('Index')
plt.ylabel('Feature Value')
plt.legend()
plt.show()
```



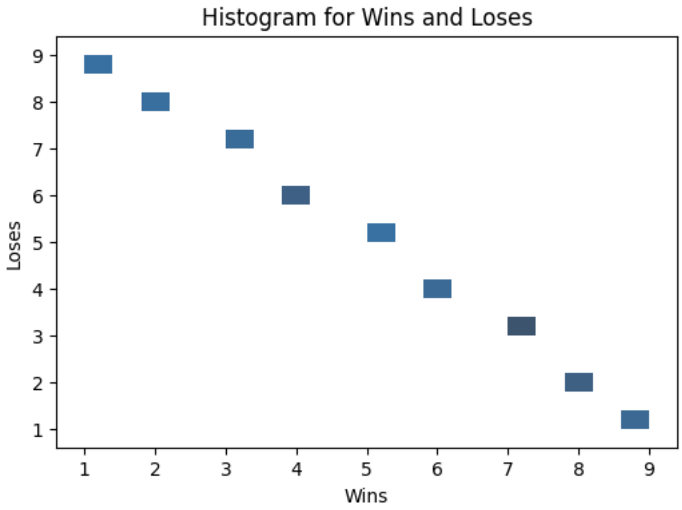
Histogram

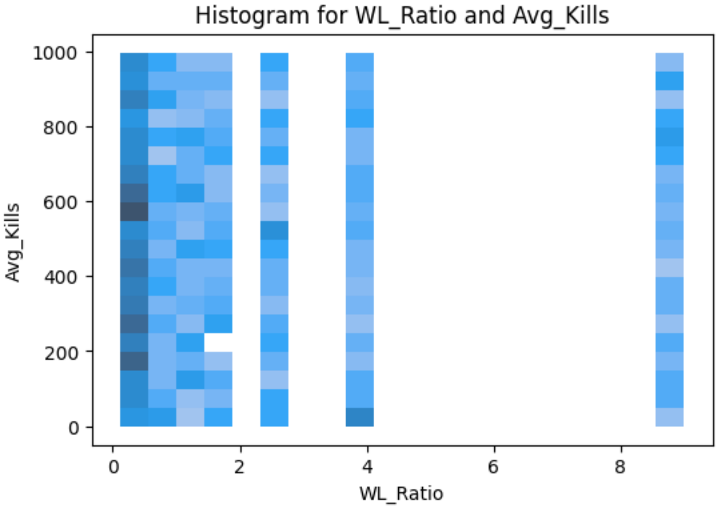
```
import matplotlib.pyplot as plt

# List of features for histograms
features_for_histogram = ['Wins', 'Loses', 'WL_Ratio', 'Avg_KDR', 'Score']

# Create histograms for each feature
plt.figure(figsize=(12, 8))
for feature in features_for_histogram:
    plt.hist(df[feature], bins=10, alpha=0.7, label=feature)

    # Add mean and standard deviation to the plot
    mean_value = df[feature].mean()
    std_dev = df[feature].std()
    plt.axvline(mean_value, color='k', linestyle='dashed', linewidth=1, label=f'Mean ({mean_value:.2f})')
    plt.axvline(mean_value + std_dev, color='r', linestyle='dashed', linewidth=1, label=f'StDev ({std_dev:.2f})')
    plt.axvline(mean_value - std_dev, color='r', linestyle='dashed', linewidth=1)

plt.title('Histograms for Selected Features')
plt.xlabel('Feature Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

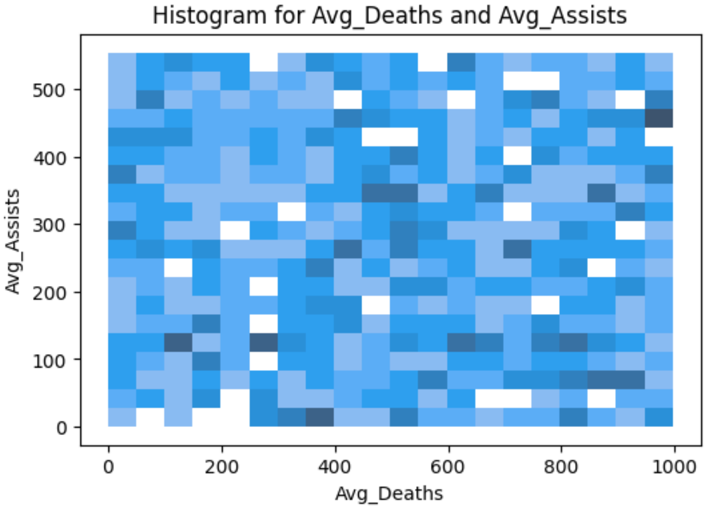


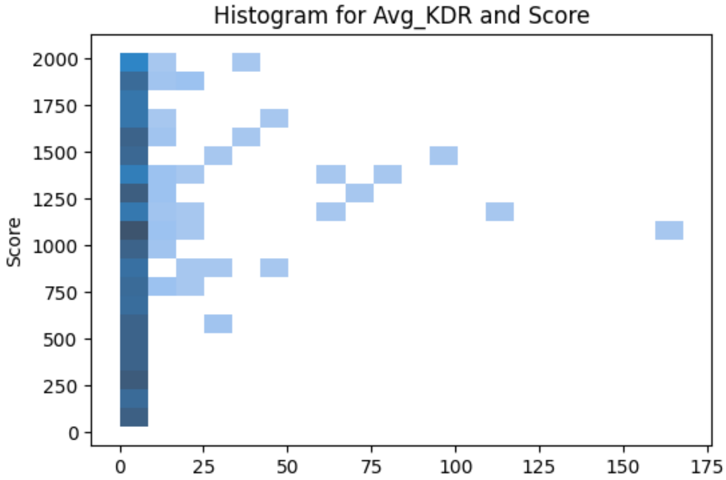
```

# List of feature pairs for histograms
feature_pairs_for_histogram = [('Wins', 'Loses'), ('WL_Ratio', 'Avg_Kills'), ('Avg_Deaths', 'Avg_Assists'), ('Avg_KDR', 'Score')]

# Create separate histograms for each feature pair
plt.figure(figsize=(12, 8))
for pair in feature_pairs_for_histogram:
    plt.figure(figsize=(6, 4))
    sns.histplot(data=df, x=pair[0], y=pair[1], bins=20)
    plt.title(f'Histogram for {pair[0]} and {pair[1]}')
    plt.xlabel(pair[0])
    plt.ylabel(pair[1])
    plt.show()
```

```
<Figure size 1200x800 with 0 Axes>
```


Histogram for Wins and Loses
Loses
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
Wins


Histogram for WL_Ratio and Avg_Kills
Avg_Kills
1000
800
600
400
200
0
0
2
4
6
8
WL_Ratio


Histogram for Avg_Deaths and Avg_Assists
Avg_Assists
500
400
300
200
100
0
0
200
400
600
800
1000
Avg_Deaths

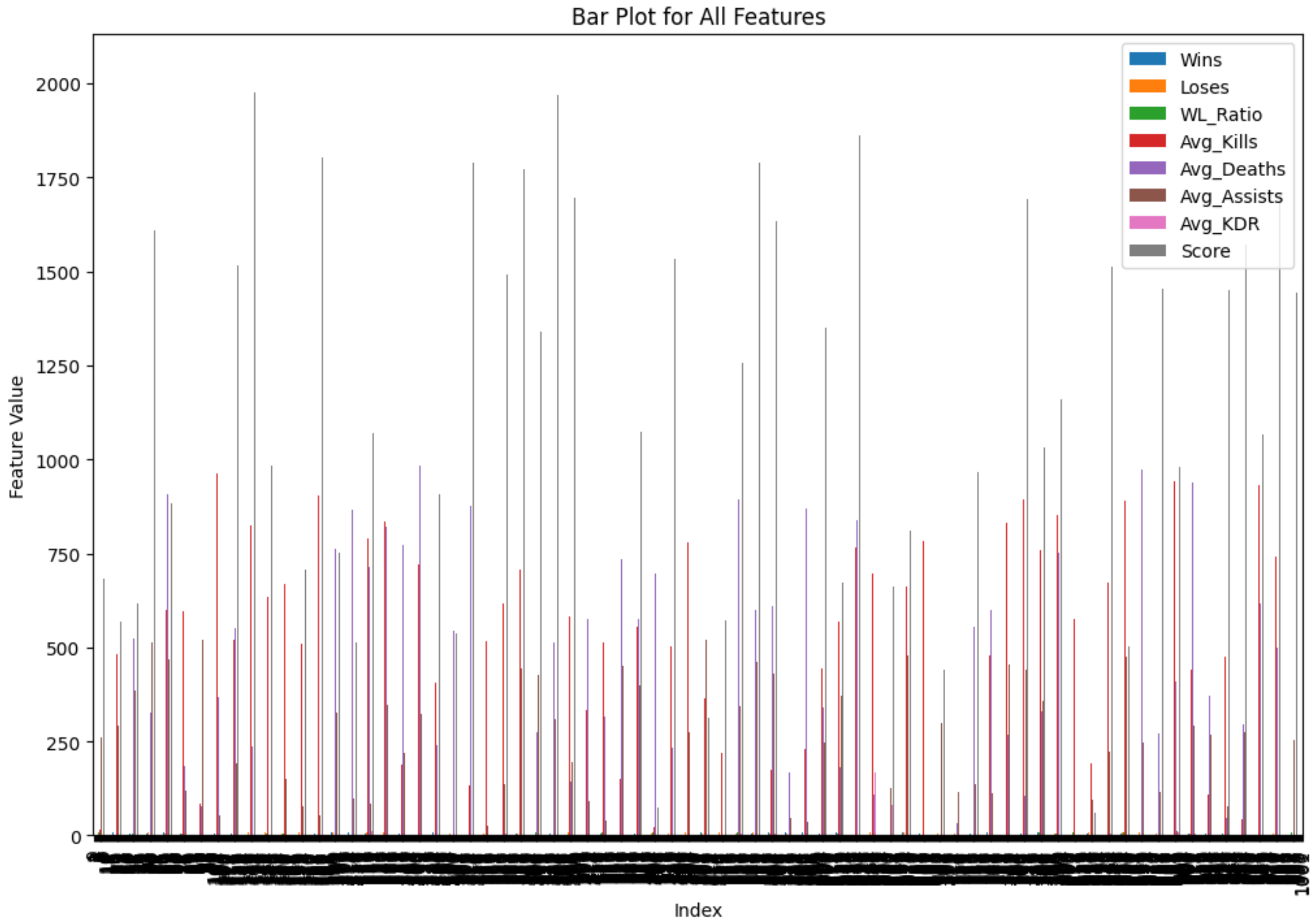

Histogram for Avg_KDR and Score
Score
2000
1750
1500
1250
1000
750
500
250
0
0
25
50
75
100
125
150
175

Avg_KDR

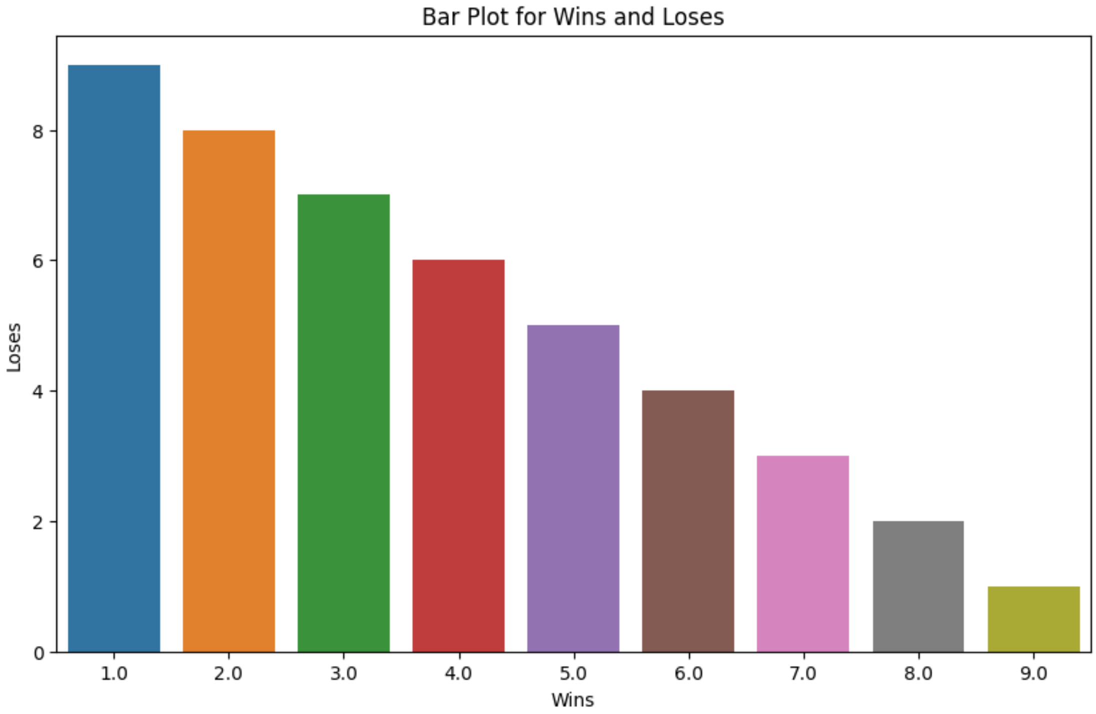
Bar Plot

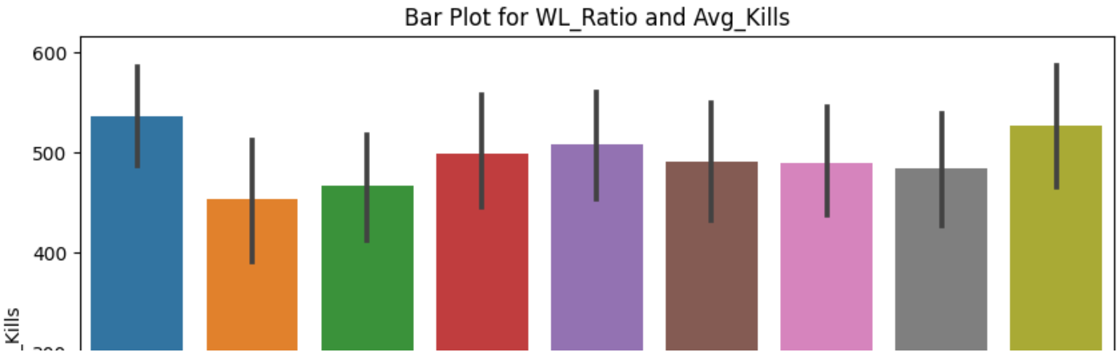
```
# List of features for bar plots
features_for_bar_plot = ['Wins', 'Loses', 'WL_Ratio', 'Avg_Kills', 'Avg_Deaths', 'Avg_Assists', 'Avg_KDR', 'Score']

# Bar plot for all features at once
plt.figure(figsize=(12, 8))
df[features_for_bar_plot].plot(kind='bar', figsize=(12, 8))
plt.title('Bar Plot for All Features')
plt.xlabel('Index')
plt.ylabel('Feature Value')
plt.legend(loc='upper right')
plt.show()

# Bar plots for two comparable features at a time
feature_pairs_for_bar_plot = [('Wins', 'Loses'), ('WL_Ratio', 'Avg_Kills'), ('Avg_Deaths', 'Avg_Assists'), ('Avg_KDR', 'Score')]

plt.figure(figsize=(12, 8))
for pair in feature_pairs_for_bar_plot:
    plt.figure(figsize=(10, 6))
    sns.barplot(data=df, x=pair[0], y=pair[1])
    plt.title(f'Bar Plot for {pair[0]} and {pair[1]}')
    plt.xlabel(pair[0])
    plt.ylabel(pair[1])
    plt.show()
```

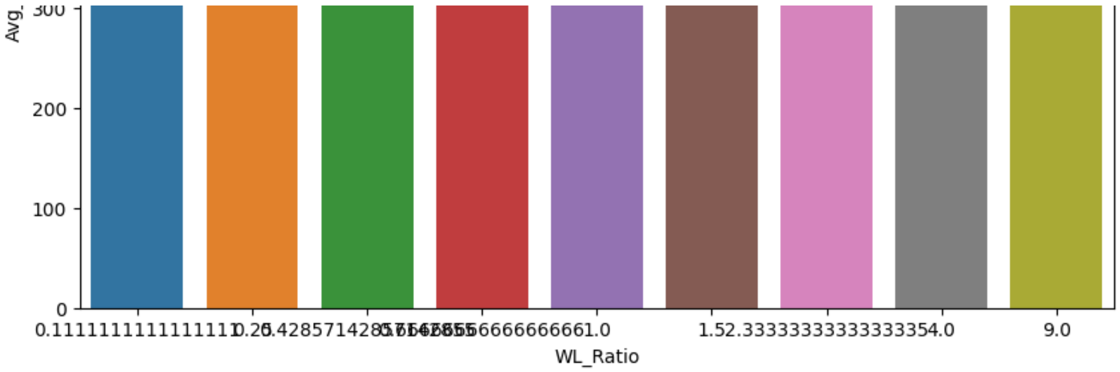
```
<Figure size 1200x800 with 0 Axes>
```

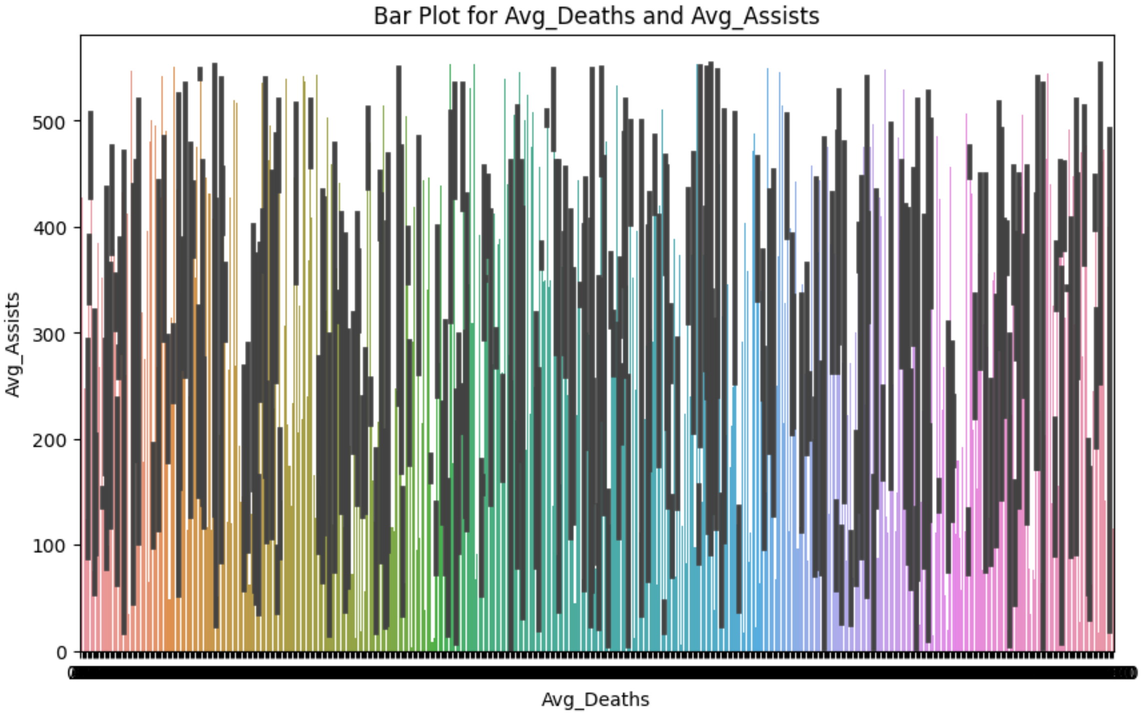

Bar Plot for All Features
Wins
Loses
WL_Ratio
Avg_Kills
Avg_Deaths
Avg_Assists
Avg_KDR
Score
Feature Value
2000
1750
1500
1250
1000
750
500
250
0
Index

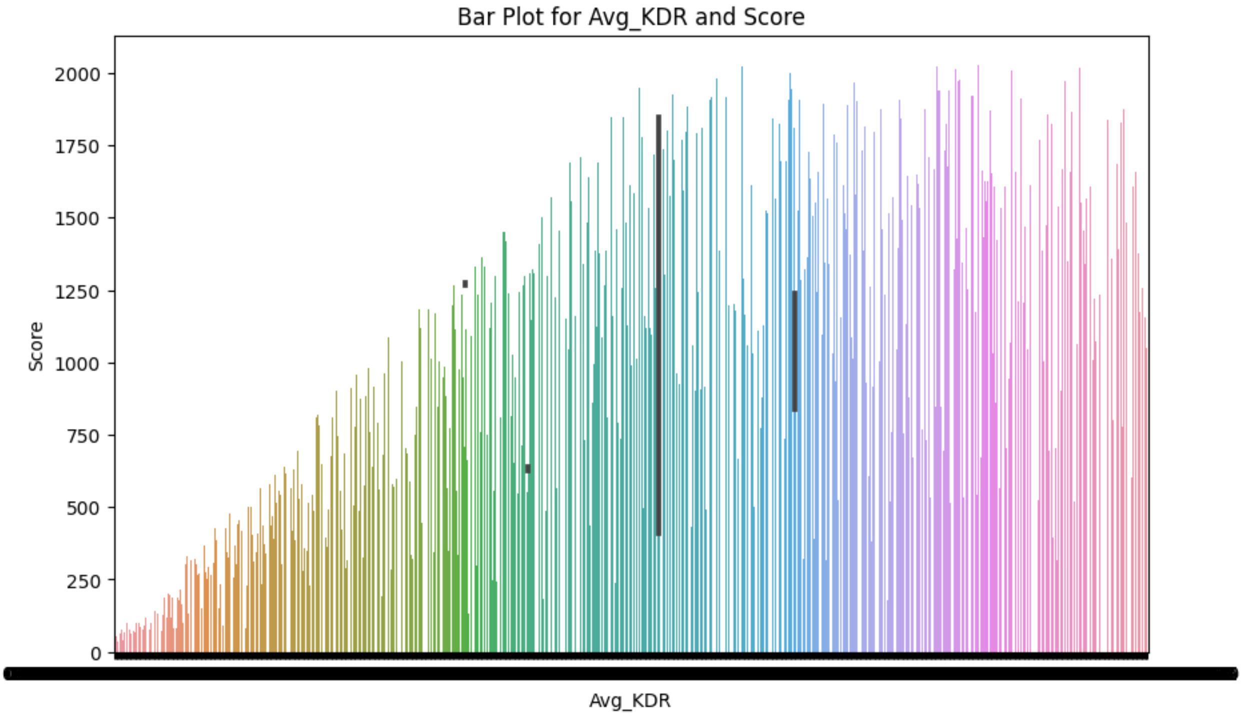
```
<Figure size 1200x800 with 0 Axes>
```


Bar Plot for Wins and Loses
Loses
8
6
4
2
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
Wins


Bar Plot for WL_Ratio and Avg_Kills
600
500
400
Kills


Avg_
300
200
100
0
0.1111111111111111
0.25
0.42857142857142855
0.5714285714285714
0.6666666666666666
1.0
1.5
2.3333333333333335
4.0
9.0
WL_Ratio

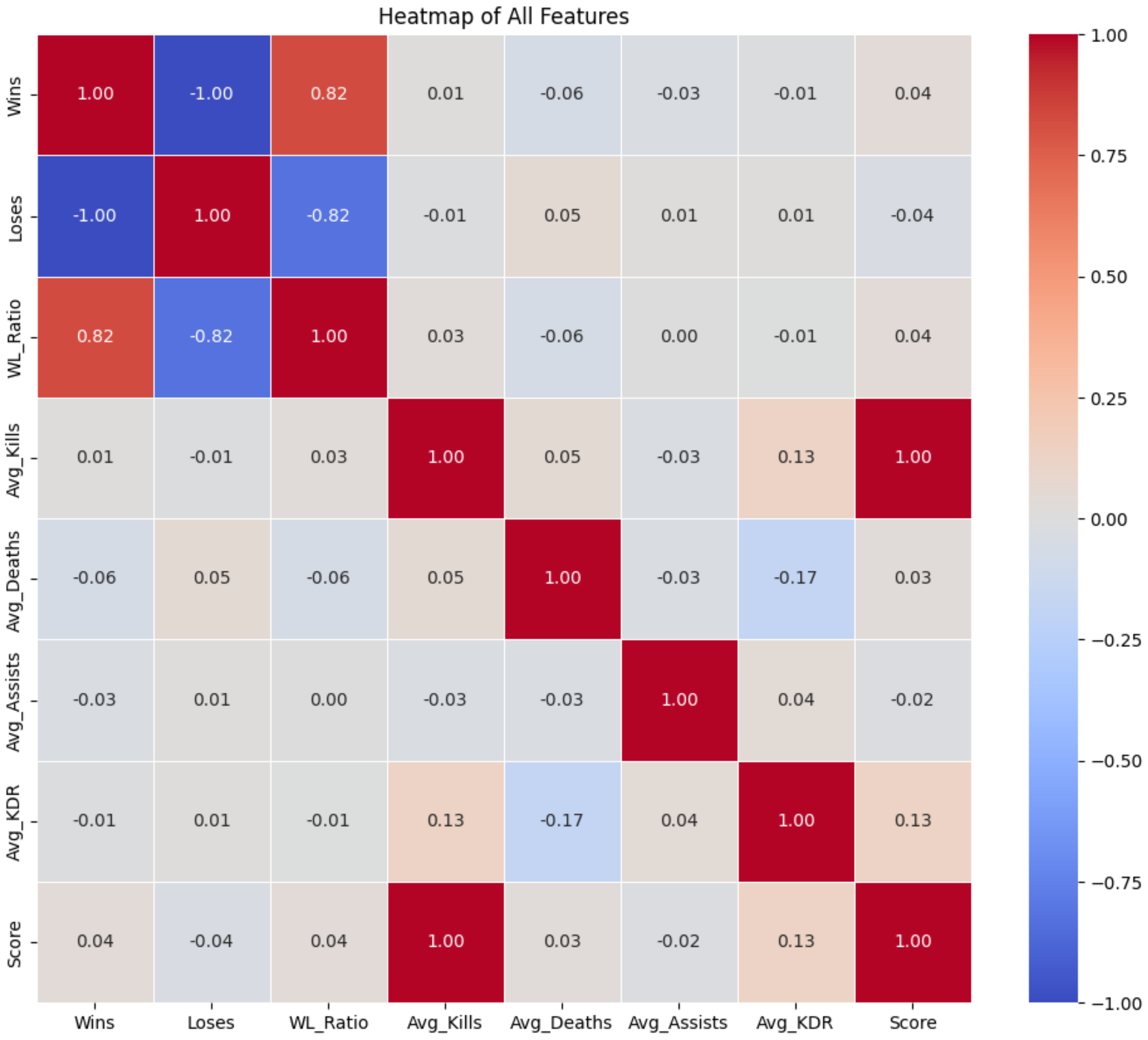

Bar Plot for Avg_Deaths and Avg_Assists
Avg_Assists
500
400
300
200
100
0
Avg_Deaths


Bar Plot for Avg_KDR and Score
Score
2000
1750
1500
1250
1000
750
500
250
0
Avg_KDR

Heat Map

```
# Create a heatmap for all features
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Heatmap of All Features')
plt.show()
```

```
<ipython-input-34-a5bff71eb4e1>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
```



Pie Plot for Categorical Plotting

```
#Categorical feature for the pie chart
categorical_feature = 'Device_Played'

# Count the occurrences of each category
category_counts = df[categorical_feature].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
plt.title(f'Pie Chart for {categorical_feature}')
plt.show()
```