```python
# Import necessary python packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Check the first 5 rows of data in the data after importing it
striker_data = pd.read_csv('Strikers_performance.csv')
striker_data.head()
```

```
   Striker_ID Nationality    Footedness Marital Status  Goals
Scored  \
0           1       Spain   Left-footed             No      17.483571

1           2      France   Left-footed            Yes      14.308678

2           3     Germany   Left-footed             No      18.238443

3           4      France  Right-footed             No      22.615149

4           5      France   Left-footed            Yes      13.829233


     Assists  Shots on Target  Shot Accuracy  Conversion Rate  \
0  10.778533        34.795488       0.677836         0.166241
1  13.728250        31.472436       0.544881         0.192774
2   3.804297        25.417413       0.518180         0.160379
3   9.688908        20.471443       0.599663         0.184602
4   6.048072        29.887563       0.582982         0.105319

   Dribbling Success  Movement off the Ball  Hold-up Play  Aerial
Duels Won  \
0           0.757061              50.921924     71.806409
15.682532
1           0.796818              61.396150     53.726866
19.843983
2           0.666869              65.863945     60.452227
20.090084
3           0.638776              88.876877     60.511979
22.363152
4           0.591485              75.565531     54.982158
13.165708

   Defensive Contribution  Big Game Performance  Consistency  \
0               30.412215              6.152481     0.820314
1               26.474913              6.093172     0.803321
2               24.164116              3.408714     0.766540
3               44.129989              6.339820     0.611798
4               37.859323              8.465658     0.701638

   Penalty Success Rate  Impact on Team Performance  Off-field Conduct
```

```
0            0.922727                  8.570370          11.451388

1            0.678984                  3.444638           8.243689

2            0.843858                  8.429491           9.506835

3            0.662997                  6.532552           8.199653

4            0.906538                  8.414915           6.665333
```

```
# Check for missing values
missing_values = striker_data.isnull().sum()
print("Missing values:")
missing_values
```

```
Missing values:

Striker_ID                    0
Nationality                   0
Footedness                    0
Marital Status                0
Goals Scored                  0
Assists                       0
Shots on Target               0
Shot Accuracy                 0
Conversion Rate               0
Dribbling Success             0
Movement off the Ball         6
Hold-up Play                  0
Aerial Duels Won              0
Defensive Contribution        0
Big Game Performance          2
Consistency                   0
Penalty Success Rate          5
Impact on Team Performance    0
Off-field Conduct             0
dtype: int64
```

```
# Look at the different data types in the dataset to fill values
properly
striker_data.dtypes
```

```
Striker_ID                      int64
Nationality                    object
Footedness                     object
Marital Status                 object
Goals Scored                  float64
Assists                       float64
Shots on Target               float64
```

```
Shot Accuracy                 float64
Conversion Rate               float64
Dribbling Success             float64
Movement off the Ball         float64
Hold-up Play                  float64
Aerial Duels Won              float64
Defensive Contribution        float64
Big Game Performance          float64
Consistency                   float64
Penalty Success Rate          float64
Impact on Team Performance    float64
Off-field Conduct             float64
dtype: object
```

```
# Use SimpleImputer to fill missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'median')
imputer.fit(striker_data[['Movement off the Ball']])
striker_data[['Movement off the Ball']] =
imputer.transform(striker_data[['Movement off the Ball']])


imputer_2 = SimpleImputer(strategy = 'median')
imputer.fit(striker_data[['Big Game Performance']])
striker_data[['Big Game Performance']] =
imputer.transform(striker_data[['Big Game Performance']])


imputer_3 = SimpleImputer(strategy = 'median')
imputer.fit(striker_data[['Penalty Success Rate']])
striker_data[['Penalty Success Rate']] =
imputer.transform(striker_data[['Penalty Success Rate']])


# Check for missing values again
missing_values = striker_data.isnull().sum()
print("Missing values:")
missing_values
```

```
Missing values:
```

```
Striker_ID                    0
Nationality                   0
Footedness                    0
Marital Status                0
Goals Scored                  0
Assists                       0
Shots on Target               0
Shot Accuracy                 0
Conversion Rate               0
Dribbling Success             0
```

```
Movement off the Ball         0
Hold-up Play                  0
Aerial Duels Won              0
Defensive Contribution        0
Big Game Performance          0
Consistency                   0
Penalty Success Rate          0
Impact on Team Performance    0
Off-field Conduct             0
dtype: int64
```

```
# Check for duplicates
duplicates = striker_data.duplicated()
striker_data[duplicates]
```

```
Empty DataFrame
Columns: [Striker_ID, Nationality, Footedness, Marital Status, Goals
Scored, Assists, Shots on Target, Shot Accuracy, Conversion Rate,
Dribbling Success, Movement off the Ball, Hold-up Play, Aerial Duels
Won, Defensive Contribution, Big Game Performance, Consistency,
Penalty Success Rate, Impact on Team Performance, Off-field Conduct]
Index: []
```

```
# Drop duplicates
striker_data.drop_duplicates(inplace = True)
```

```
# Check for duplicates again
duplicates = striker_data.duplicated()
striker_data[duplicates]
```

```
Empty DataFrame
Columns: [Striker_ID, Nationality, Footedness, Marital Status, Goals
Scored, Assists, Shots on Target, Shot Accuracy, Conversion Rate,
Dribbling Success, Movement off the Ball, Hold-up Play, Aerial Duels
Won, Defensive Contribution, Big Game Performance, Consistency,
Penalty Success Rate, Impact on Team Performance, Off-field Conduct]
Index: []
```

```
# Analyze the values in the columns, round to 2 decimal places
striker_data['Goals Scored'] = round(striker_data['Goals Scored'], 2)
striker_data['Goals Scored']
```

```
0      17.48
1      14.31
2      18.24
3      22.62
4      13.83
       ...
495    17.69
496     9.81
497    14.05
```

```
498    10.62
499     8.09
Name: Goals Scored, Length: 500, dtype: float64
```

```
striker_data['Assists'] = round(striker_data['Assists'], 2)
striker_data['Assists']
```

```
0      10.78
1      13.73
2       3.80
3       9.69
4       6.05
       ...
495     7.16
496    13.39
497     9.92
498     6.29
499     9.72
Name: Assists, Length: 500, dtype: float64
```

```
striker_data['Shots on Target'] = round(striker_data['Shots on
Target'], 2)
striker_data['Shots on Target']
```

```
0      34.80
1      31.47
2      25.42
3      20.47
4      29.89
       ...
495    39.04
496    39.43
497    33.46
498    32.17
499    29.15
Name: Shots on Target, Length: 500, dtype: float64
```

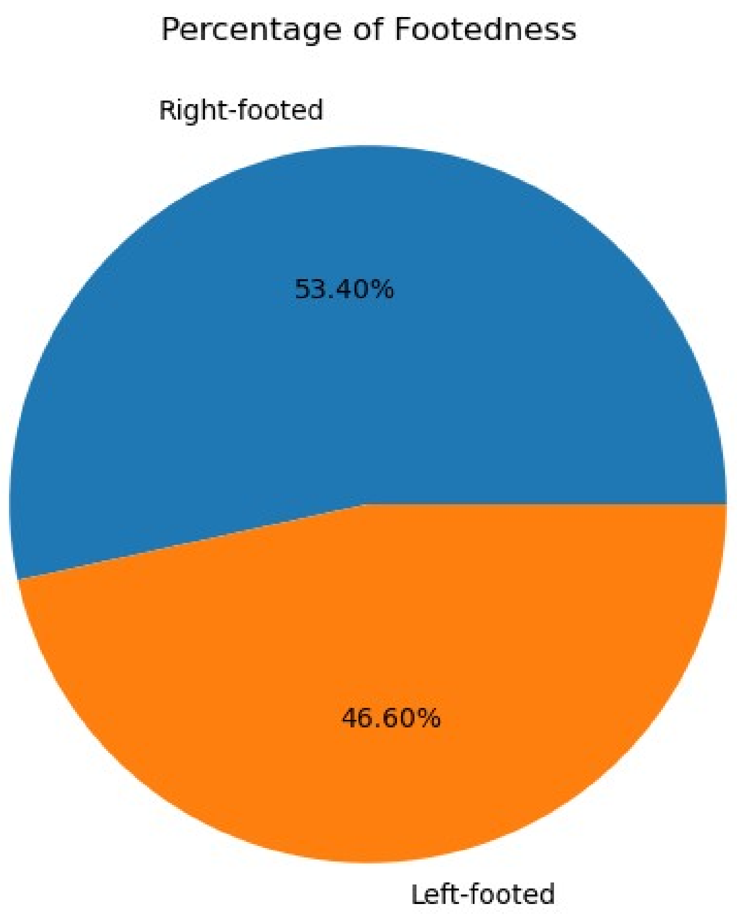
```
striker_data['Movement off the Ball'] = round(striker_data['Movement
off the Ball'], 2)
striker_data['Movement off the Ball']
```

```
0      50.92
1      61.40
2      65.86
3      88.88
4      75.57
       ...
495    89.35
496    78.16
497    69.52
498    68.17
```

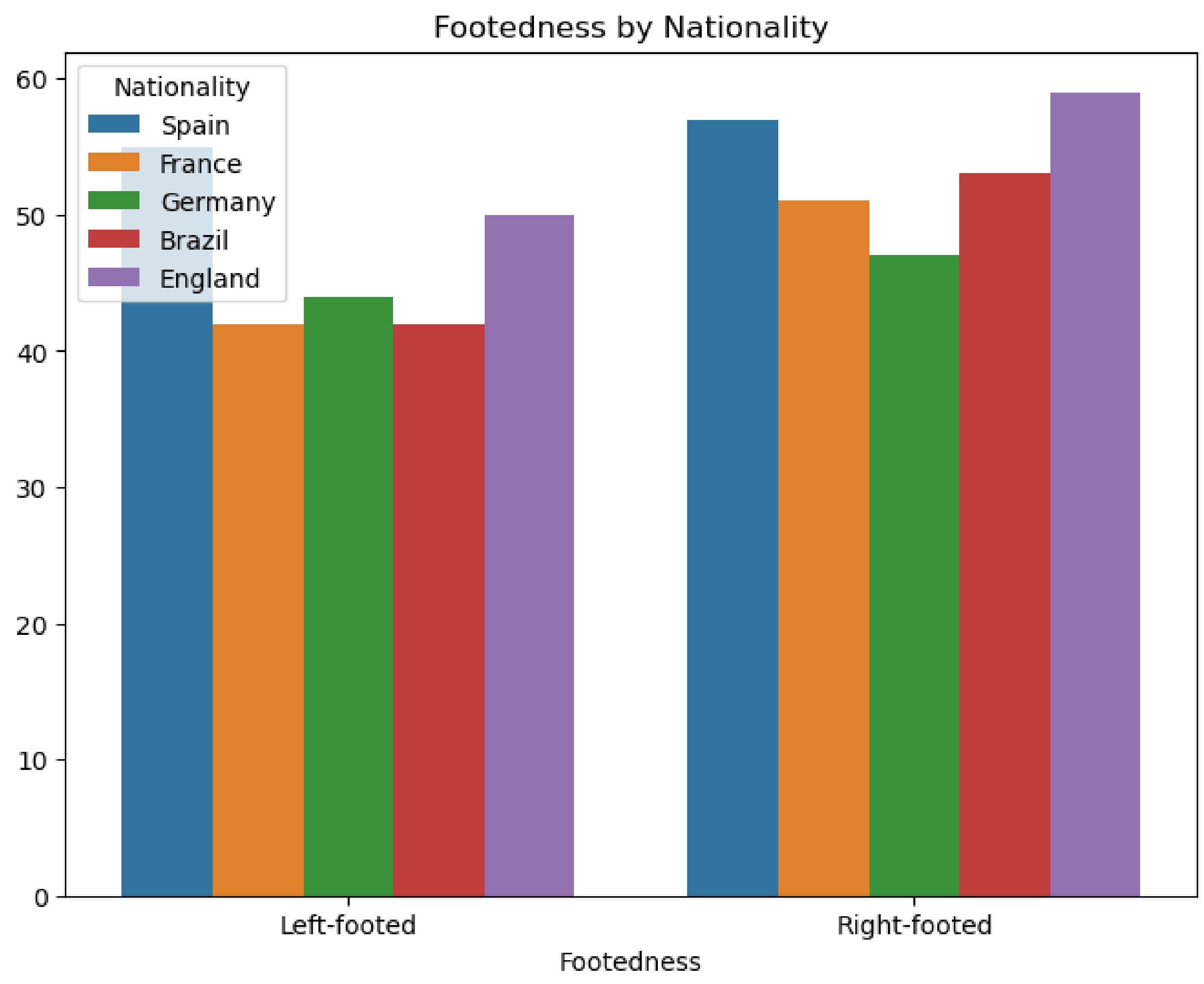
```
499    66.43
Name: Movement off the Ball, Length: 500, dtype: float64

striker_data['Hold-up Play'] = round(striker_data['Hold-up Play'], 2)
striker_data['Hold-up Play']

0      71.81
1      53.73
2      60.45
3      60.51
4      54.98
       ...
495    60.28
496    39.22
497    56.80
498    76.43
499    63.61
Name: Hold-up Play, Length: 500, dtype: float64

striker_data['Aerial Duels Won'] = round(striker_data['Aerial Duels
Won'], 2)
striker_data['Aerial Duels Won']

0      15.68
1      19.84
2      20.09
3      22.36
4      13.17
       ...
495    28.39
496    15.97
497    25.38
498     9.15
499    14.03
Name: Aerial Duels Won, Length: 500, dtype: float64

striker_data['Defensive Contribution'] = round(striker_data['Defensive
Contribution'], 2)
striker_data['Defensive Contribution']

0      30.41
1      26.47
2      24.16
3      44.13
4      37.86
       ...
495    39.51
496    47.11
497    71.13
498    48.08
```

```
499    31.52
Name: Defensive Contribution, Length: 500, dtype: float64

striker_data['Big Game Performance'] = round(striker_data['Big Game 
Performance'], 2)
striker_data['Big Game Performance']

0       6.15
1       6.09
2       3.41
3       6.34
4       8.47
       ...
495     4.45
496     6.74
497     5.70
498     2.61
499    10.20
Name: Big Game Performance, Length: 500, dtype: float64

striker_data['Impact on Team Performance'] = 
round(striker_data['Impact on Team Performance'], 2)
striker_data['Impact on Team Performance']

0       8.57
1       3.44
2       8.43
3       6.53
4       8.41
       ...
495     6.00
496     5.97
497    11.25
498     1.45
499     6.64
Name: Impact on Team Performance, Length: 500, dtype: float64

striker_data['Off-field Conduct'] = round(striker_data['Off-field 
Conduct'], 2)
striker_data['Off-field Conduct']

0      11.45
1       8.24
2       9.51
3       8.20
4       6.67
       ...
495    12.42
496     8.65
497     6.33
498    11.31
```

```
499    12.16
Name: Off-field Conduct, Length: 500, dtype: float64

# Determine which players in the dataset are left-footed or right-
footed
footedness_counts = striker_data['Footedness'].value_counts()
footedness_percentages = footedness_counts / footedness_counts.sum() *
100

footedness_counts

Footedness
Right-footed    267
Left-footed     233
Name: count, dtype: int64

footedness_percentages

Footedness
Right-footed    53.4
Left-footed     46.6
Name: count, dtype: float64

# Create a pie chart to visualize the percentage of players that are
left-footed or right-footed
plt.figure(figsize=(8, 6))
footedness_percentages.plot(kind='pie',autopct = '%1.2f%%')
plt.title('Percentage of Footedness')
plt.ylabel('')
plt.show()
```



```
plt.figure(figsize=(8, 6))
sns.countplot(x='Footedness', hue='Nationality', data = striker_data)
plt.title('Footedness by Nationality')
plt.ylabel('')
plt.show()
```



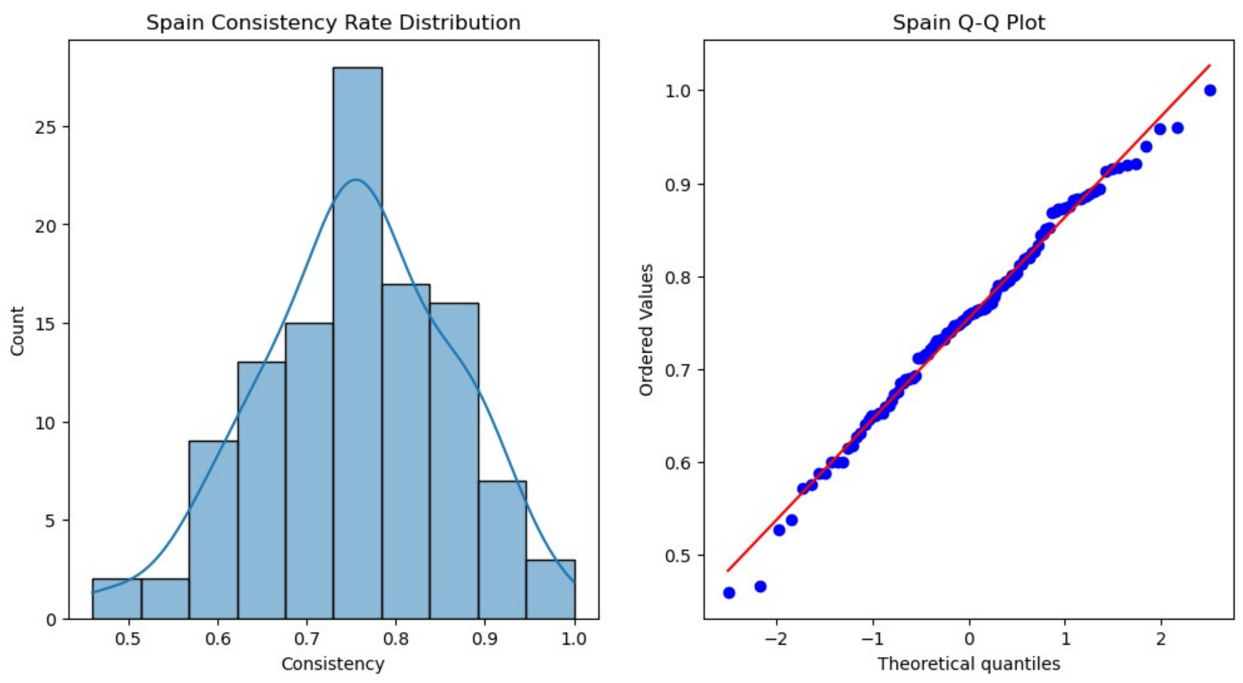
```
# look at the average goals of each nationality and determine the
highest average by nationality
average_goals = round(striker_data.groupby('Nationality')['Goals
Scored'].mean(), 2)
highest_avg_goals_nationality = average_goals.idxmax()
highest_avg_goals_value = round(average_goals.max(), 2)

average_goals

Nationality
Brazil      15.81
England     14.47
France      14.90
Germany     14.86
Spain       15.20
Name: Goals Scored, dtype: float64

highest_avg_goals_nationality

'Brazil'
```

```
highest_avg_goals_value

15.81

# Sort the goals scored by individual players in descending order
sorted_goals = striker_data.sort_values(by='Goals Scored',
ascending=False)
sorted_goals.head()

     Striker_ID Nationality    Footedness Marital Status  Goals Scored
\
209         210       Spain  Right-footed            Yes         34.26

478         479      France  Right-footed             No         30.39

179         180     England  Right-footed            Yes         28.60

113         114     Germany   Left-footed            Yes         27.32

220         221      Brazil   Left-footed            Yes         26.57


     Assists  Shots on Target  Shot Accuracy  Conversion Rate  \
209    13.09            37.25       0.502547         0.199965
478     3.25            18.79       0.522712         0.133070
179    11.81            17.27       0.556838         0.200340
113     4.62            28.47       0.592216         0.149891
220     8.92            16.69       0.651311         0.182083

     Dribbling Success  Movement off the Ball  Hold-up Play  Aerial
Duels Won  \
209           0.856341                  74.51         60.39
14.37
478           0.754348                  81.54         65.68
13.03
179           0.673655                  75.16         59.16
18.29
113           0.699348                  57.91         63.96
12.35
220           0.895416                  59.50         69.96
8.04

     Defensive Contribution  Big Game Performance  Consistency  \
209                   39.61                  7.12     0.939873
478                   43.90                  6.04     0.773986
179                   42.51                  7.03     0.700137
113                   34.88                  4.19     0.759316
220                   50.07                 10.57     0.706613

     Penalty Success Rate  Impact on Team Performance  Off-field
Conduct
```

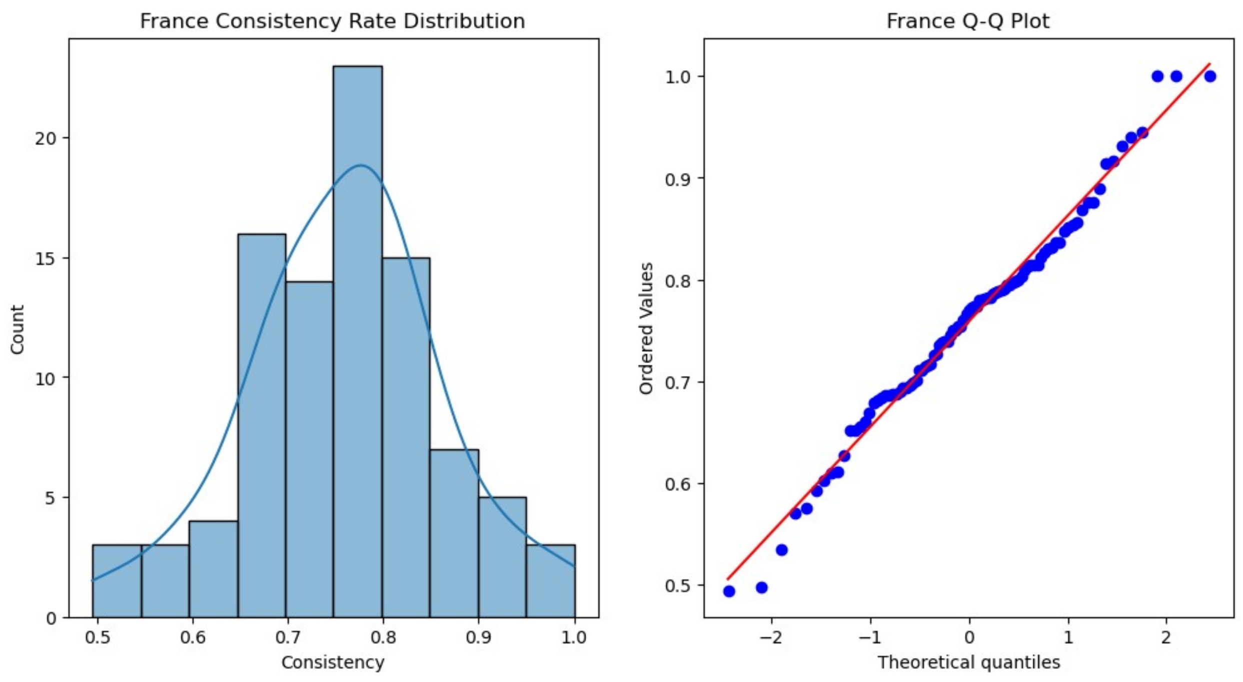
```
209                 0.770955                            9.31
6.24
478                 0.956026                           10.84
8.17
179                 0.745021                            5.55
6.25
113                 1.000000                            5.10
7.22
220                 0.856827                            9.21
7.66

# look at conversion rate grouped by footedness
average_conversion_rate_footedness =
round(striker_data.groupby('Footedness')['Conversion Rate'].mean(), 3)
average_conversion_rate_footedness

Footedness
Left-footed     0.198
Right-footed    0.201
Name: Conversion Rate, dtype: float64

import scipy.stats as stats
# List unique nationalities
nationalities = striker_data['Nationality'].unique()

# Plot histograms and Q-Q plots for each nationality
for nationality in nationalities:
    plt.figure(figsize=(12, 6))

    # Histogram
    plt.subplot(1, 2, 1)
    sns.histplot(striker_data[striker_data['Nationality'] ==
nationality]['Consistency'], kde=True)
    plt.title(f'{nationality} Consistency Rate Distribution')

    # Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(striker_data[striker_data['Nationality'] ==
nationality]['Consistency'].dropna(), dist="norm", plot=plt)
    plt.title(f'{nationality} Q-Q Plot')

    plt.show()

    # Shapiro-Wilk test for normality
    shapiro_test =
stats.shapiro(striker_data[striker_data['Nationality'] == nationality]
['Consistency'].dropna())
    print(f"Shapiro-Wilk test for {nationality}:
Statistic={shapiro_test.statistic:.3f}, p-
value={shapiro_test.pvalue:.3f}")
```

```
# Perform Levene's test for homogeneity of variances
levene_test = stats.levene(*[striker_data[striker_data['Nationality']
== nationality]['Consistency'].dropna() for nationality in
nationalities])
print(f"Levene's test: Statistic={levene_test.statistic:.3f}, p-
value={levene_test.pvalue:.3f}")

# Perform ANOVA if there are more than two nationalities
if len(nationalities) > 2:
    anova_test =
stats.f_oneway(*[striker_data[striker_data['Nationality'] ==
nationality]['Consistency'].dropna() for nationality in
nationalities])
    print(f"ANOVA test: F-statistic={anova_test.statistic:.3f}, p-
value={anova_test.pvalue:.3f}")
else:
    # Example for two nationalities
    nat1, nat2 = nationalities
    t_test = stats.ttest_ind(df[df['Nationality'] == nat1]
['Consistency'].dropna(),
                             df[df['Nationality'] == nat2]
['Consistency'].dropna())
    print(f"T-test: T-statistic={t_test.statistic:.3f}, p-
value={t_test.pvalue:.3f}")
```

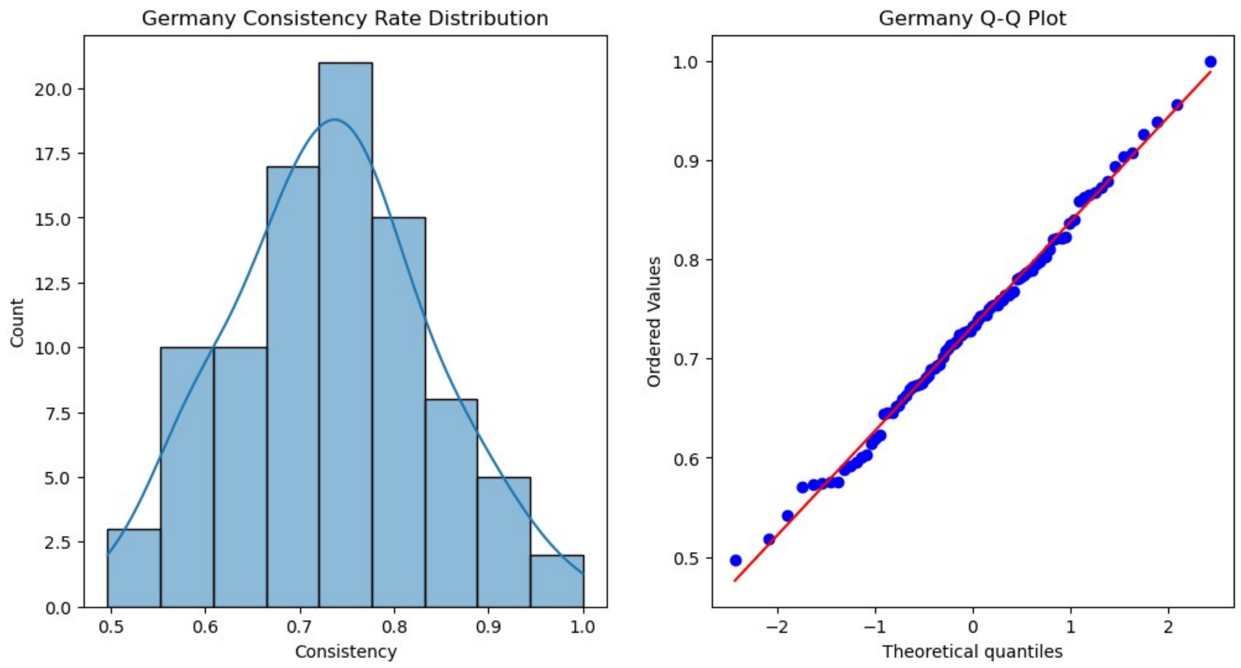
```
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



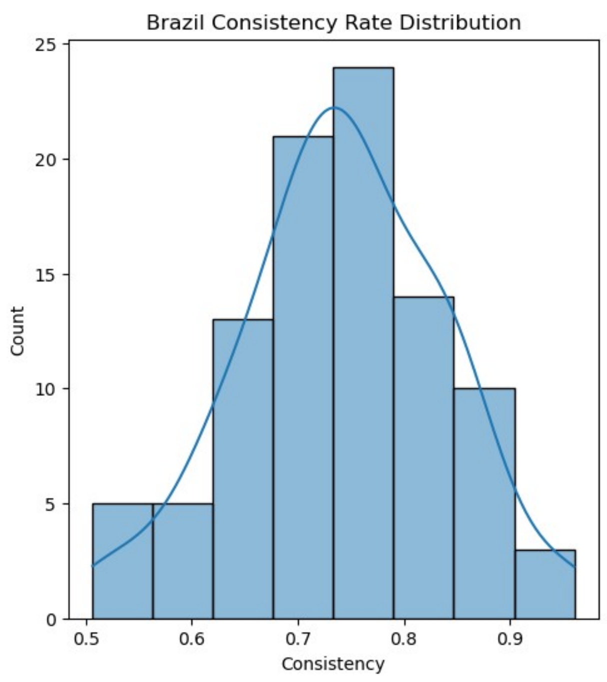
```
Shapiro-Wilk test for Spain: Statistic=0.990, p-value=0.627

C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

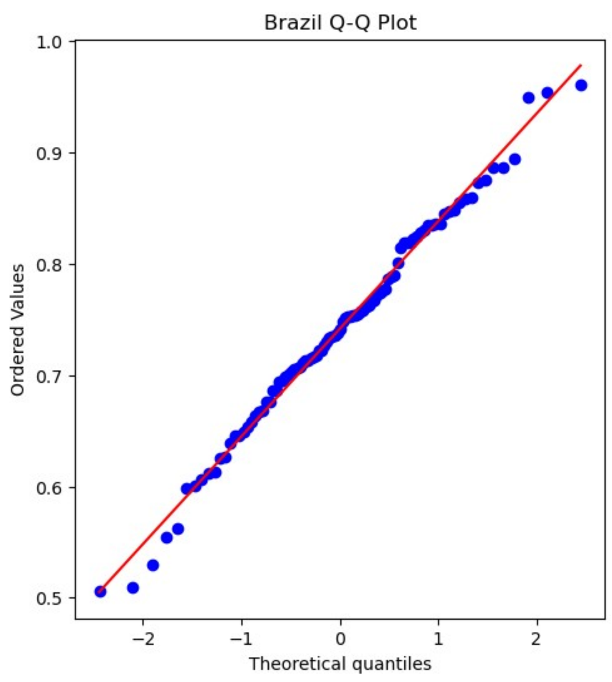


```
Shapiro-Wilk test for France: Statistic=0.986, p-value=0.421

C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

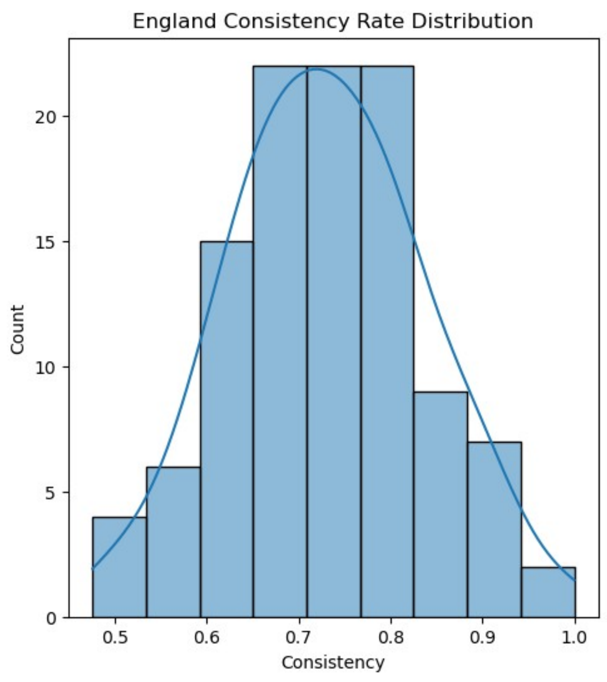


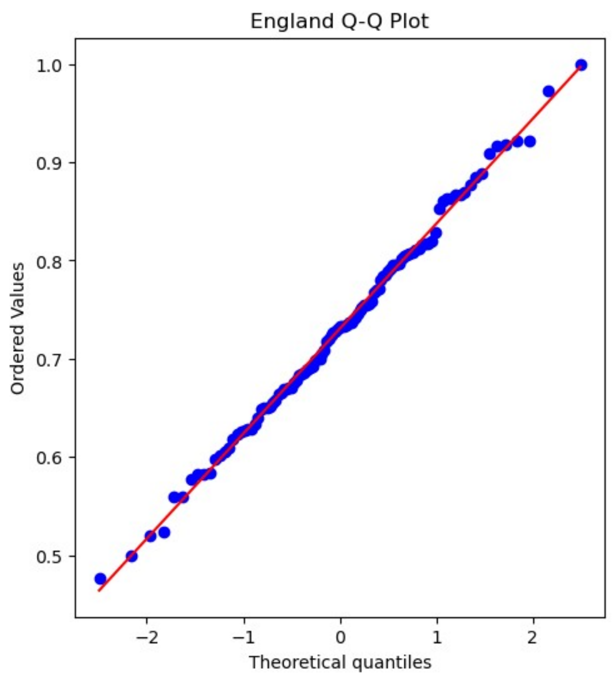
```
Shapiro-Wilk test for Germany: Statistic=0.994, p-value=0.964

C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```





```
Shapiro-Wilk test for Brazil: Statistic=0.990, p-value=0.730

C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```





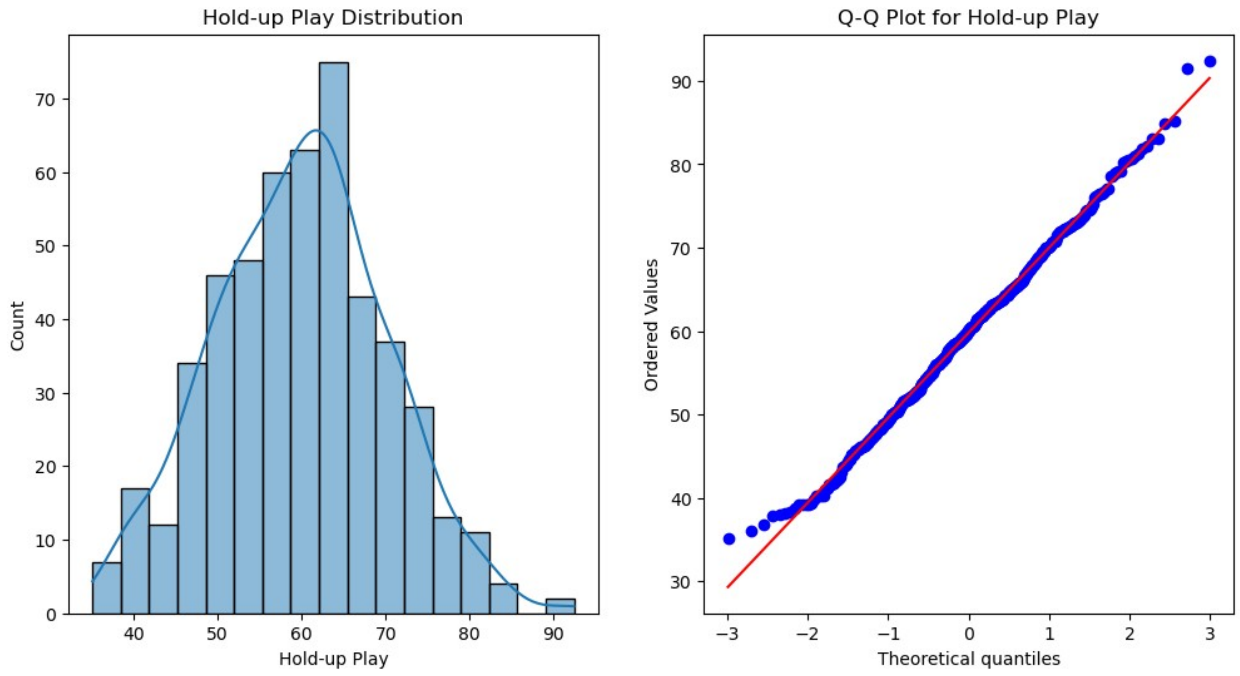
```
Shapiro-Wilk test for England: Statistic=0.995, p-value=0.973
Levene's test: Statistic=0.400, p-value=0.808
ANOVA test: F-statistic=1.528, p-value=0.193
```

```python
# Look at consistency score grouped by nationality rounded to 3
decimal places
average_consistency_nationality =
round(striker_data.groupby('Nationality')['Consistency'].mean(), 3)
average_consistency_nationality
```

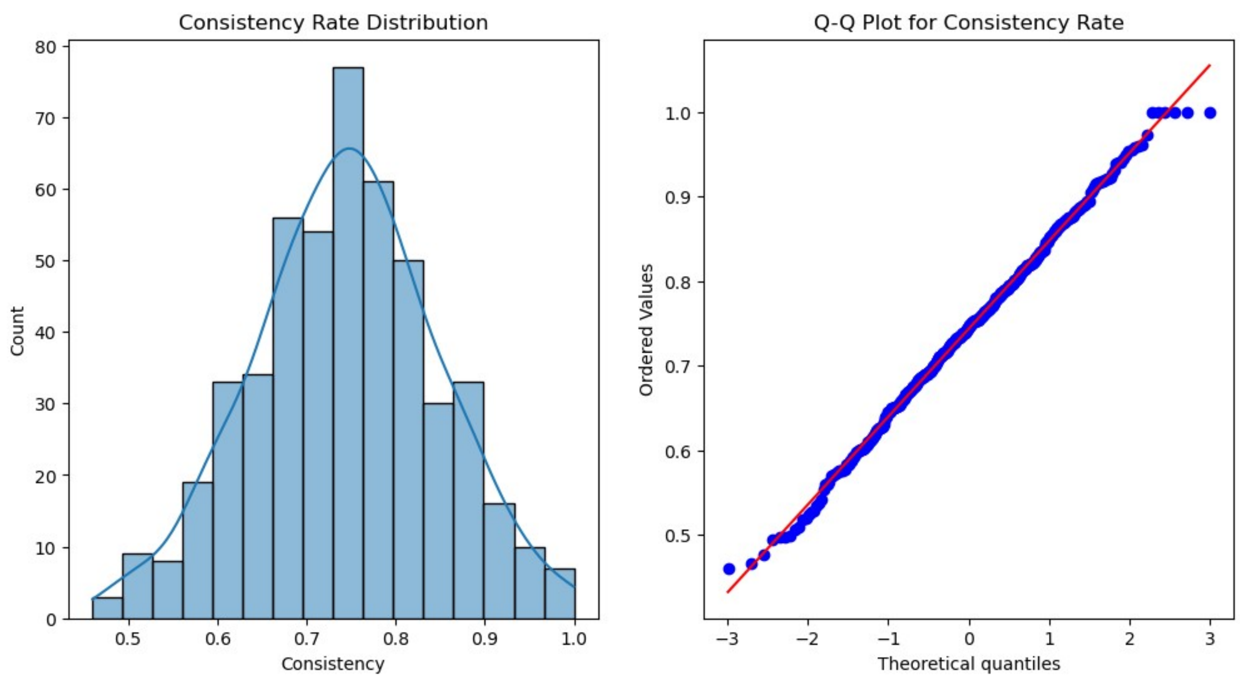
```
Nationality
Brazil     0.742
England    0.731
France     0.759
Germany    0.732
Spain      0.755
Name: Consistency, dtype: float64
```

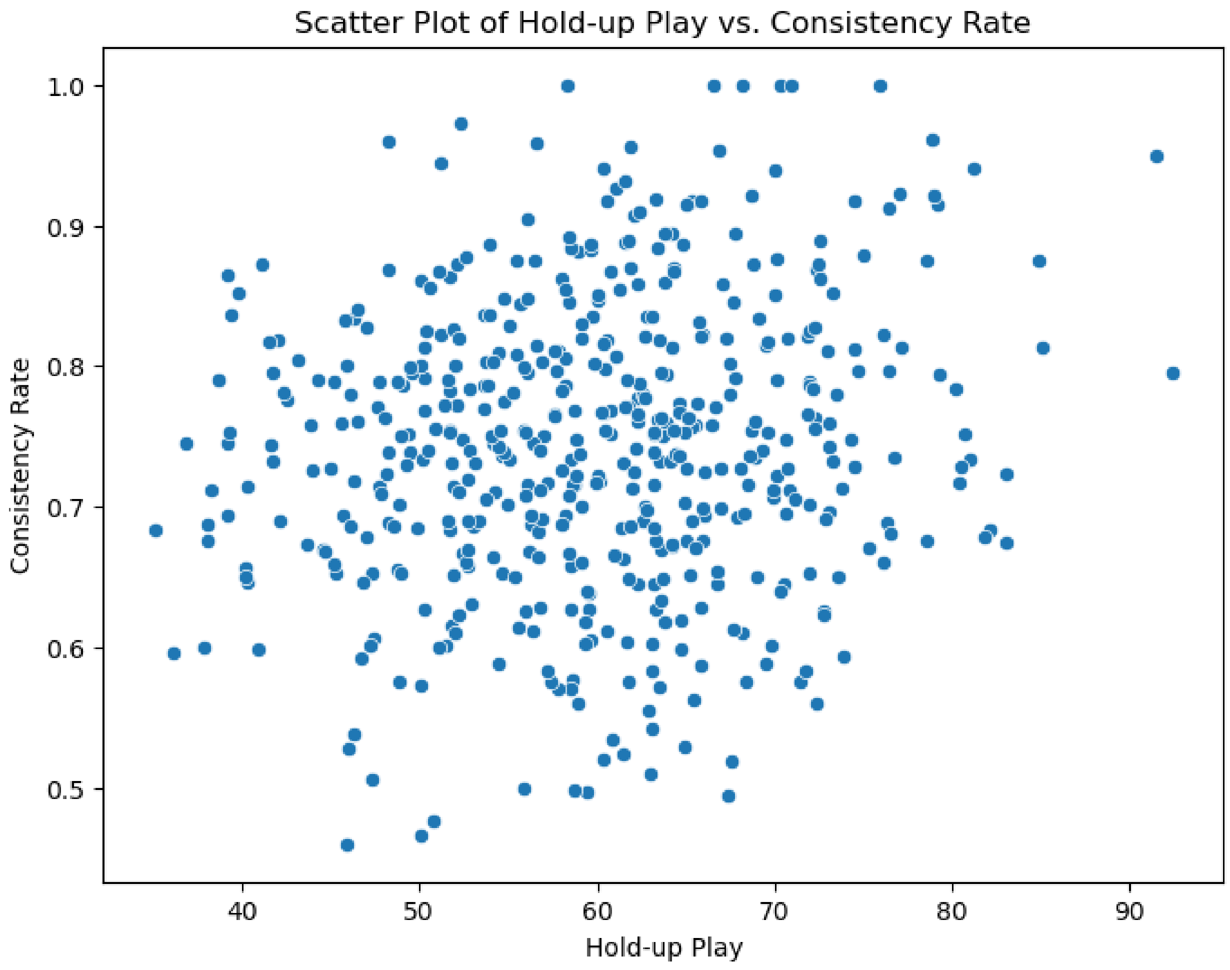
```python
# Check normality with histograms and Q-Q plots
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(striker_data['Hold-up Play'], kde=True)
plt.title('Hold-up Play Distribution')

plt.subplot(1, 2, 2)
stats.probplot(striker_data['Hold-up Play'], dist="norm", plot=plt)
plt.title('Q-Q Plot for Hold-up Play')

plt.show()

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

plt.subplot(1, 2, 1)
sns.histplot(striker_data['Consistency'], kde=True)
plt.title('Consistency Rate Distribution')

plt.subplot(1, 2, 2)
stats.probplot(striker_data['Consistency'], dist="norm", plot=plt)
plt.title('Q-Q Plot for Consistency Rate')

plt.show()

# Scatter plot to check linearity
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Hold-up Play', y='Consistency', data=striker_data)
plt.title('Scatter Plot of Hold-up Play vs. Consistency Rate')
plt.xlabel('Hold-up Play')
plt.ylabel('Consistency Rate')
plt.show()
```

```
# Calculate the Pearson correlation
pearson_corr, pearson_p_value = stats.pearsonr(striker_data['Hold-up
Play'], striker_data['Consistency'])
print(f"Pearson Correlation: {pearson_corr:.3f}, p-value:
{pearson_p_value:.3f}")


C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



```
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option
is deprecated and will be removed in a future version. Convert inf
values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Consistency Rate Distribution
Count
Consistency
Q-Q Plot for Consistency Rate
Ordered Values
Theoretical quantiles


Scatter Plot of Hold-up Play vs. Consistency Rate
Consistency Rate
Hold-up Play

```
Pearson Correlation: 0.145, p-value: 0.001

# Use the ANOVA test to determine statistical significance between
nationality and consistency score
Brazil = striker_data.query('Nationality == "Brazil"')['Consistency']
France = striker_data.query('Nationality == "France"')['Consistency']
Spain = striker_data.query('Nationality == "Spain"')['Consistency']
England = striker_data.query('Nationality == "England"')
['Consistency']
Germany = striker_data.query('Nationality == "Germany"')
['Consistency']

stats, p_value = stats.f_oneway(Brazil, France, Spain, England,
Germany)
print(round(p_value,2))

0.19

# Use the levene test to determine statistical significance
from scipy.stats import levene
stats, p_value = levene (Brazil, France, Spain, England, Germany)
print(round(p_value,2))

0.81

# Use the Shapiro-Wilk test to determine statistical significance of
Conistency and Hold-Up Play
from scipy.stats import shapiro

numeric_columns = ['Consistency', 'Hold-up Play']
shapiro_results = {}

for column in numeric_columns:
    stat, p_value = shapiro(striker_data[column])
    shapiro_results[column] = round(p_value,3)

shapiro_results

{'Consistency': 0.451, 'Hold-up Play': 0.324}

# Perform data transformation by using all the features to create a
single feature called total contribution score
striker_data['Total Contribution Score'] = round(striker_data['Goals
Scored'] + striker_data['Assists'] + striker_data['Shots on Target'] +
striker_data['Dribbling Success'] + striker_data['Aerial Duels Won']
+ striker_data['Defensive Contribution'] + striker_data['Big Game
Performance'] + striker_data['Consistency'], 2)

striker_data['Total Contribution Score']

0      116.88
1      113.51
```

```
2       96.55
3      126.86
4      110.56
        ...
495    137.71
496    134.04
497    160.84
498    110.45
499    104.03
Name: Total Contribution Score, Length: 500, dtype: float64
```

```
# Use LabelEncoder to turn footedness and marital status into numbers
so they can be used for machine learning
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
striker_data['Footedness'] =
encoder.fit_transform(striker_data['Footedness'])
```

```
striker_data['Footedness']
```

```
0      0
1      0
2      0
3      1
4      0
      ..
495    0
496    1
497    0
498    1
499    0
Name: Footedness, Length: 500, dtype: int32
```

```
striker_data.head()
```

```
   Striker_ID Nationality  Footedness Marital Status  Goals Scored
Assists  \
0           1       Spain           0             No         17.48
10.78
1           2      France           0            Yes         14.31
13.73
2           3     Germany           0             No         18.24
3.80
3           4      France           1             No         22.62
9.69
4           5      France           0            Yes         13.83
6.05

   Shots on Target  Shot Accuracy  Conversion Rate  Dribbling Success
\
0            34.80       0.677836          0.166241              0.757061

1            31.47       0.544881          0.192774              0.796818

2            25.42       0.518180          0.160379              0.666869

3            20.47       0.599663          0.184602              0.638776

4            29.89       0.582982          0.105319              0.591485


   Movement off the Ball  Hold-up Play  Aerial Duels Won  \
0                  50.92         71.81             15.68
1                  61.40         53.73             19.84
2                  65.86         60.45             20.09
3                  88.88         60.51             22.36
4                  75.57         54.98             13.17

   Defensive Contribution  Big Game Performance  Consistency  \
0                   30.41                  6.15     0.820314
1                   26.47                  6.09     0.803321
2                   24.16                  3.41     0.766540
3                   44.13                  6.34     0.611798
4                   37.86                  8.47     0.701638

   Penalty Success Rate  Impact on Team Performance  Off-field Conduct
\
0              0.922727                        8.57              11.45

1              0.678984                        3.44               8.24

2              0.843858                        8.43               9.51

3              0.662997                        6.53               8.20

4              0.906538                        8.41               6.67


   Total Contribution Score
0                    116.88
1                    113.51
2                     96.55
3                    126.86
4                    110.56

encoder_2 = LabelEncoder()
striker_data['Marital Status'] =
encoder.fit_transform(striker_data['Marital Status'])

striker_data['Marital Status']
```

```
0      0
1      1
2      0
3      0
4      1
      ..
495    1
496    1
497    1
498    1
499    0
Name: Marital Status, Length: 500, dtype: int32
```

```
# Turn the nationality column into true or false values so they can be
used for machine learning
dummies = pd.get_dummies(striker_data['Nationality'])
striker_data = pd.concat([striker_data,dummies],axis=1)
striker_data.head()
```

```
   Striker_ID Nationality  Footedness  Marital Status  Goals Scored
Assists  \
0           1       Spain           0               0         17.48
10.78
1           2      France           0               1         14.31
13.73
2           3     Germany           0               0         18.24
3.80
3           4      France           1               0         22.62
9.69
4           5      France           0               1         13.83
6.05

   Shots on Target  Shot Accuracy  Conversion Rate  Dribbling Success
...  \
0            34.80       0.677836         0.166241           0.757061
...
1            31.47       0.544881         0.192774           0.796818
...
2            25.42       0.518180         0.160379           0.666869
...
3            20.47       0.599663         0.184602           0.638776
...
4            29.89       0.582982         0.105319           0.591485
...

   Consistency  Penalty Success Rate  Impact on Team Performance  \
0     0.820314              0.922727                        8.57
1     0.803321              0.678984                        3.44
2     0.766540              0.843858                        8.43
3     0.611798              0.662997                        6.53
```

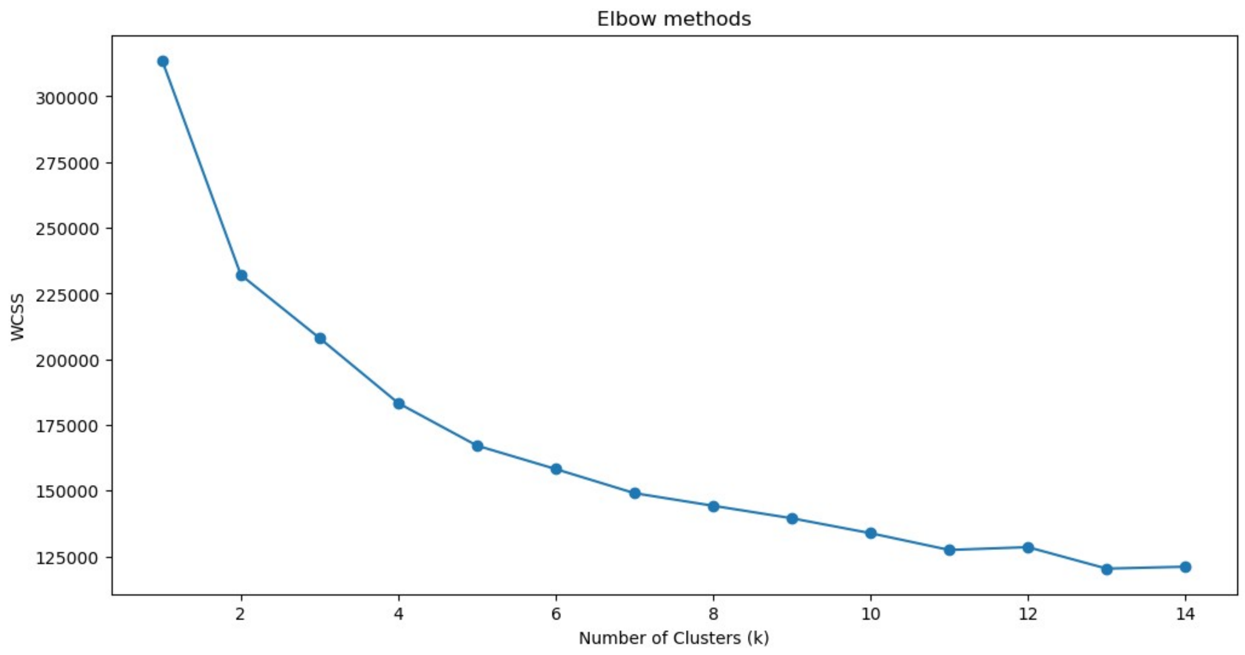
```
4     0.701638              0.906538                        8.41

   Off-field Conduct  Total Contribution Score  Brazil  England
France  \
0              11.45                    116.88   False    False
False
1               8.24                    113.51   False    False
True
2               9.51                     96.55   False    False
False
3               8.20                    126.86   False    False
True
4               6.67                    110.56   False    False
True

   Germany  Spain
0    False   True
1    False  False
2     True  False
3    False  False
4    False  False

[5 rows x 25 columns]

# Perform a model summary for hold-up play
import statsmodels.api as sm

X = sm.add_constant(striker_data['Consistency'])
Y = striker_data['Hold-up Play']

model = sm.OLS(Y, X).fit()

regression_summary = model.summary()
regression_summary

<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:           Hold-up Play   R-squared:
0.021
Model:                            OLS   Adj. R-squared:
0.019
Method:                 Least Squares   F-statistic:
10.70
Date:                Thu, 29 Aug 2024   Prob (F-statistic):
0.00114
Time:                        00:53:21   Log-Likelihood:
```

```
-1863.3
No. Observations:                 500   AIC:
3731.
Df Residuals:                     498   BIC:
3739.
Df Model:                           1

Covariance Type:            nonrobust

======================================================================
=========
                  coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
---------
const          49.2270      3.266     15.070      0.000      42.809
55.645
Consistency    14.2328      4.351      3.271      0.001       5.685
22.781
======================================================================
========
Omnibus:                        1.336   Durbin-Watson:
2.020
Prob(Omnibus):                  0.513   Jarque-Bera (JB):
1.317
Skew:                           0.041   Prob(JB):
0.518
Kurtosis:                       2.762   Cond. No.
15.0
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
"""

# Import Kmeans and look for the optimal K value between 1 and 15,
drop non-numerical columns
from sklearn.cluster import KMeans
x= striker_data.drop(['Striker_ID','Nationality'],axis=1)
wcss = []

for i in range(1,15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++')
    kmeans.fit(x)
    wcss_score = kmeans.inertia_
    wcss.append(wcss_score)
```

```
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
```

```
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(
```

```python
# Visualize the optimal K value using the elbow method(1 is not
optimal)
plt.figure(figsize = (12, 6))
plt.plot(range(1,15),wcss,marker = 'o')
plt.title('Elbow methods')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()
```



```
# Create labels using the optimal K value
final_km = KMeans(n_clusters=2)
final_km.fit(x)
labels = final_km.labels_
labels

C:\Users\jarre\OneDrive\Desktop\sample_project_1\env\Lib\site-
packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans is known
to have a memory leak on Windows with MKL, when there are less chunks
than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=2.
  warnings.warn(

array([0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
1,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0,
0,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,
0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
1,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
1,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
0,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
1,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
1,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1,
0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1,
0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0,
1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
0,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
0,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
0,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
0,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0])

# Turn the labels into the Clusters column to determine the best
strikers in the dataset
striker_data['Clusters'] = labels
striker_data.head()

   Striker_ID Nationality  Footedness  Marital Status  Goals Scored
Assists  \
0           1       Spain           0               0         17.48
10.78
1           2      France           0               1         14.31
13.73
2           3     Germany           0               0         18.24
3.80
3           4      France           1               0         22.62
9.69
4           5      France           0               1         13.83
6.05

   Shots on Target  Shot Accuracy  Conversion Rate  Dribbling Success
...  \
0            34.80       0.677836         0.166241           0.757061
```

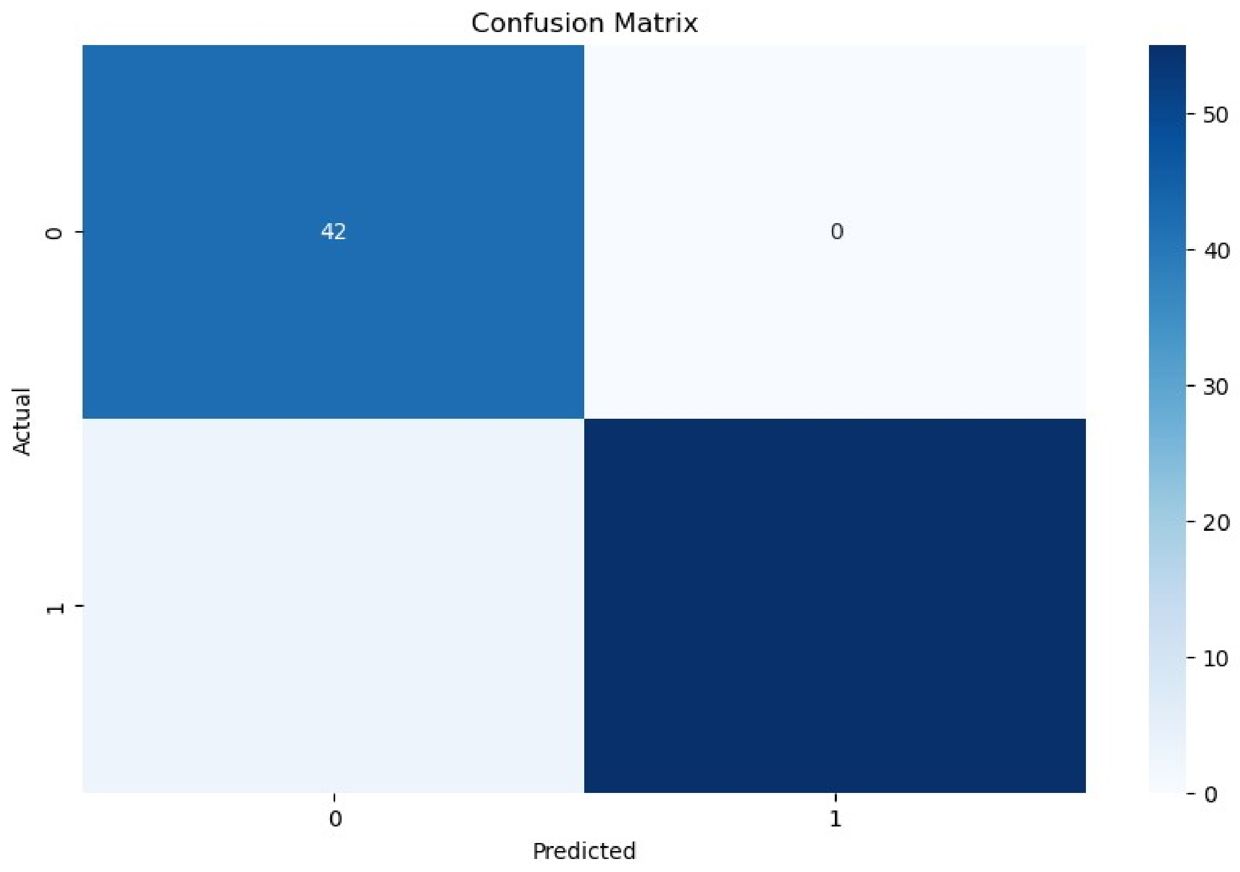
```
...
1          31.47       0.544881         0.192774              0.796818
...
2          25.42       0.518180         0.160379              0.666869
...
3          20.47       0.599663         0.184602              0.638776
...
4          29.89       0.582982         0.105319              0.591485
...

   Penalty Success Rate  Impact on Team Performance  Off-field Conduct
\
0              0.922727                        8.57              11.45

1              0.678984                        3.44               8.24

2              0.843858                        8.43               9.51

3              0.662997                        6.53               8.20

4              0.906538                        8.41               6.67


   Total Contribution Score  Brazil  England  France  Germany  Spain
Clusters
0                    116.88   False    False   False    False   True
0
1                    113.51   False    False    True    False  False
0
2                     96.55   False    False   False     True  False
0
3                    126.86   False    False    True    False  False
1
4                    110.56   False    False    True    False  False
0

[5 rows x 26 columns]

# Differentiate between the best strikers and average strikers using
the total contribution score
round(striker_data.groupby('Clusters')['Total Contribution
Score'].mean(),2)

Clusters
0    105.02
1    126.59
Name: Total Contribution Score, dtype: float64

striker_data.head()
```

```
   Striker_ID Nationality  Footedness  Marital Status  Goals Scored
Assists  \
0           1       Spain           0               0         17.48
10.78
1           2      France           0               1         14.31
13.73
2           3     Germany           0               0         18.24
3.80
3           4      France           1               0         22.62
9.69
4           5      France           0               1         13.83
6.05

   Shots on Target  Shot Accuracy  Conversion Rate  Dribbling Success
...  \
0            34.80       0.677836         0.166241           0.757061
...
1            31.47       0.544881         0.192774           0.796818
...
2            25.42       0.518180         0.160379           0.666869
...
3            20.47       0.599663         0.184602           0.638776
...
4            29.89       0.582982         0.105319           0.591485
...

   Penalty Success Rate  Impact on Team Performance  Off-field Conduct
\
0              0.922727                        8.57              11.45

1              0.678984                        3.44               8.24

2              0.843858                        8.43               9.51

3              0.662997                        6.53               8.20

4              0.906538                        8.41               6.67


   Total Contribution Score  Brazil  England  France  Germany  Spain
Clusters
0                    116.88   False    False   False    False   True
0
1                    113.51   False    False    True    False  False
0
2                     96.55   False    False   False     True  False
0
3                    126.86   False    False    True    False  False
1
4                    110.56   False    False    True    False  False
```

```
0

[5 rows x 26 columns]

# Use the mapping method to differentiate between the best strikres
and average strikers, drop the clusters column
mapping = {1:"Best Strikers",0:"Average Strikers"}
striker_data["Striker Types"] = striker_data['Clusters'].map(mapping)
striker_data.drop('Clusters',axis=1,inplace=True)
striker_data.head()

   Striker_ID Nationality  Footedness  Marital Status  Goals Scored
Assists  \
0           1       Spain           0               0         17.48
10.78
1           2      France           0               1         14.31
13.73
2           3     Germany           0               0         18.24
3.80
3           4      France           1               0         22.62
9.69
4           5      France           0               1         13.83
6.05

   Shots on Target  Shot Accuracy  Conversion Rate  Dribbling Success
...  \
0            34.80       0.677836         0.166241           0.757061
...
1            31.47       0.544881         0.192774           0.796818
...
2            25.42       0.518180         0.160379           0.666869
...
3            20.47       0.599663         0.184602           0.638776
...
4            29.89       0.582982         0.105319           0.591485
...

   Penalty Success Rate  Impact on Team Performance  Off-field Conduct
\
0              0.922727                        8.57              11.45

1              0.678984                        3.44               8.24

2              0.843858                        8.43               9.51

3              0.662997                        6.53               8.20

4              0.906538                        8.41               6.67


   Total Contribution Score  Brazil  England  France  Germany
```

```
Spain  \
0                    116.88   False    False   False    False   True

1                    113.51   False    False    True    False  False

2                     96.55   False    False   False     True  False

3                    126.86   False    False    True    False  False

4                    110.56   False    False    True    False  False


      Striker Types
0  Average Strikers
1  Average Strikers
2  Average Strikers
3     Best Strikers
4  Average Strikers

[5 rows x 26 columns]

# Drop the non-numerical features
x = striker_data.drop(['Striker_ID','Nationality','Striker
Types'],axis=1)
y = striker_data['Striker Types']

# Scale the features using StandardScalar to improve accuracy
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_x = scaler.fit_transform(x)

# Split the data into train and test splits
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(scaled_x,y,test_size=0.2,random_state=42)

# Determine the accuracy of the test set using the LogisticRegression
model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
lgr_model = LogisticRegression()
lgr_model.fit(x_train,y_train)

y_lgr_pred = lgr_model.predict(x_test)

conf_matrix_lgr = confusion_matrix(y_lgr_pred,y_test)
accuracy_lgr = accuracy_score(y_lgr_pred,y_test)
print(accuracy_lgr*100,'%')

97.0 %
```

```python
# Plot the confusion matrix to visualize the accuracy of the
LogisticRegression model
plt.figure(figsize=(10,6))
sns.heatmap(conf_matrix_lgr,annot=True,fmt="d",cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```


Confusion Matrix
Actual
Predicted