```
#Import some libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
```

```
#Read Excel File
```

```
data=pd.read_excel('/content/Strikers_performance.xlsx')
```

```
data.head()
```

	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	
0	1	Spain	Left-footed	No	17.483571	10.778533	34.795488	0.677836	0.166241	0.757061	50.921924	71.806409	15
1	2	France	Left-footed	Yes	14.308678	13.728250	31.472436	0.544881	0.192774	0.796818	61.396150	53.726866	19
2	3	Germany	Left-footed	No	18.238443	3.804297	25.417413	0.518180	0.160379	0.666869	65.863945	60.452227	20
3	4	France	Right-footed	No	22.615149	9.688908	20.471443	0.599663	0.184602	0.638776	88.876877	60.511979	22
4	5	France	Left-footed	Yes	13.829233	6.048072	29.887563	0.582982	0.105319	0.591485	75.565531	54.982158	13

```
# View data type
```

```
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
```

```
# Describe Data
```

```
data.describe()
```

	Striker_ID	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate	Dribbling Success	Movement off the Ball	Hold-up Play	Aerial Duels Won	Defensive Contribution
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	494.000000	500.000000	500.000000	500.000000
mean	250.500000	15.036603	8.095660	25.759392	0.603319	0.199424	0.702133	69.790151	59.810879	19.529831	39.954865
std	144.481833	4.898554	2.933487	7.071724	0.098407	0.047978	0.100108	10.400347	10.167912	4.958031	9.921254
min	1.000000	0.000000	0.000000	4.726212	0.305961	0.049024	0.400886	40.705513	35.067292	4.961838	8.232962
25%	125.750000	11.498463	6.214125	20.782993	0.538806	0.166963	0.637285	62.668746	52.425012	16.395836	33.458187
50%	250.500000	15.063986	8.085595	25.838641	0.599109	0.199842	0.700354	69.617049	60.075293	19.741772	39.978322
75%	375.250000	18.183916	9.953727	30.283169	0.669977	0.233459	0.763227	76.953526	66.033512	22.856931	46.693579
max	500.000000	34.263657	15.897146	43.211782	0.919311	0.355496	1.000000	98.684031	92.430930	34.073272	71.129102

```
# Select data which is non numeric

non_numeric_data = data.select_dtypes(exclude=['number']).columns
non_numeric_data
```

```
Index(['Nationality', 'Footedness', 'Marital Status'], dtype='object')
```

```
# Select Data which is Numeric

numeric_data = data.select_dtypes(include=['number']).columns
numeric_data
```

```
Index(['Striker_ID', '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'],
      dtype='object')
```

```
# Fill the numeric missing values with strategy of median

imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')
nw_imputer=imputer.fit(data[numeric_data])
numeric_data_imputed=nw_imputer.transform(data[numeric_data])
```

```
numeric_data_imputed
```

```
array([[  1.        ,  17.48357077,  10.77853264, ...,   0.92272738,
          8.57037016,  11.45138759],
       [  2.        ,  14.30867849,  13.72824992, ...,   0.67898395,
          3.44463808,   8.24368893],
       [  3.        ,  18.23844269,   3.80429728, ...,   0.84385793,
          8.4294913 ,   9.50683495],
       ...,
       [498.        ,  14.04830661,   9.92252858, ...,   0.74726109,
         11.24911246,   6.32975099],
       [499.        ,  10.62190873,   6.28646303, ...,   0.79948934,
          1.45237012,  11.3058261 ],
       [500.        ,   8.08600135,   9.71774834, ...,   0.83987638,
          6.64076056,  12.15555517]])
```

```
# Fill categorical missing value with strategy of most frequent

imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
nw_imputer=imputer.fit(data[non_numeric_data])
non_numeric_data_imputed=nw_imputer.transform(data[non_numeric_data])
non_numeric_data_imputed
```

```
array([['Spain', 'Left-footed', 'No'],
       ['France', 'Left-footed', 'Yes'],
       ['Germany', 'Left-footed', 'No'],
       ...,
       ['England', 'Left-footed', 'Yes'],
       ['England', 'Right-footed', 'Yes'],
       ['England', 'Left-footed', 'No']], dtype=object)
```

```
#Check the null values
data.isnull().sum()
```

```
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
```

```
# Removing null value
```

```
data['Big Game Performance'].fillna('Unknown',inplace=True)
```

```
data['Movement off the Ball'].fillna('Unknown',inplace=True)
```

```
data['Penalty Success Rate'].fillna('Unknown',inplace=True)
```

```
data.isnull().sum()
```

```
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
```

```
# Replace Unknown with -1 and convert it into given column into integer
```

```
data.replace('Unknown', -1, inplace=True)
```

```
data[['Goals Scored', 'Assists', 'Shots on Target', 'Movement off the Ball',
      'Hold-up Play', 'Aerial Duels Won', 'Defensive Contribution',
      'Big Game Performance', 'Impact on Team Performance',
      'Off-field Conduct']].astype(int)
```

	Goals Scored	Assists	Shots on Target	Movement off the Ball	Hold-up Play	Aerial Duels Won	Defensive Contribution	Big Game Performance	Impa Perfor
0	17	10	34	50	71	15	30	6	
1	14	13	31	61	53	19	26	6	
2	18	3	25	65	60	20	24	3	
3	22	9	20	88	60	22	44	6	
4	13	6	29	75	54	13	37	8	
...	...	...	...	...	...	...	...	...	
495	17	7	39	89	60	28	39	4	
496	9	13	39	78	39	15	47	6	
497	14	9	33	69	56	25	71	5	
498	10	6	32	68	76	9	48	2	
499	8	9	29	66	63	14	31	10	

```
data.dtypes
```

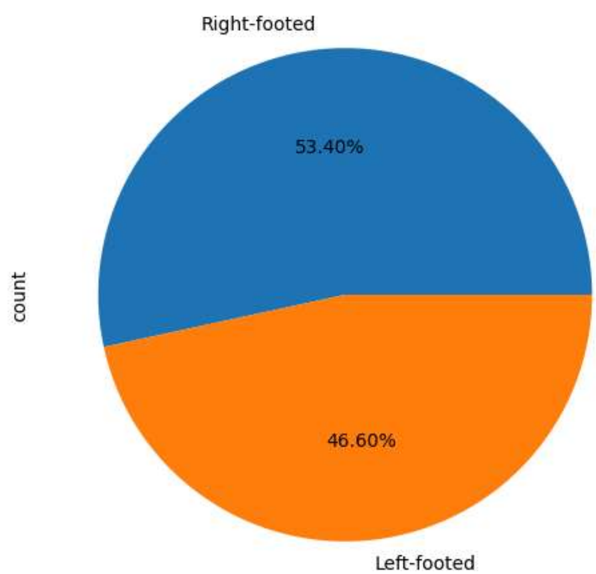
```
Striker_ID                   int64
Nationality                 object
Footedness                  object
Marital Status              object
Goals Scored               float64
Assists                    float64
Shots on Target            float64
Shot Accuracy              float64
```

```
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
```

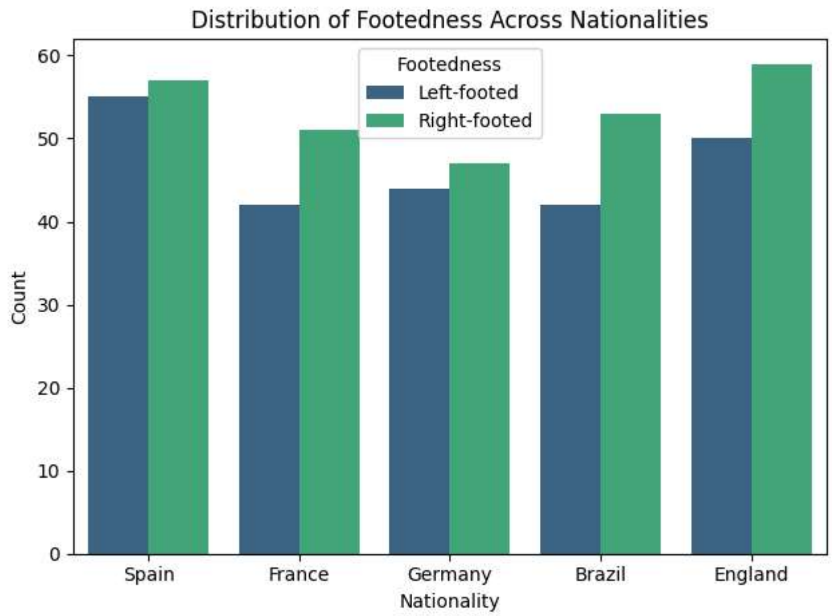
```
round(data.describe(),2)
```

	Striker_ID	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate	Dribbling Success	Movement off the Ball	H
count	500.00	500.00	500.00	500.00	500.00	500.00	500.00	500.00	5(
mean	250.50	15.04	8.10	25.76	0.60	0.20	0.70	68.94	!
std	144.48	4.90	2.93	7.07	0.10	0.05	0.10	12.90	
min	1.00	0.00	0.00	4.73	0.31	0.05	0.40	-1.00	:
25%	125.75	11.50	6.21	20.78	0.54	0.17	0.64	62.25	!
50%	250.50	15.06	8.09	25.84	0.60	0.20	0.70	69.45	(
75%	375.25	18.18	9.95	30.28	0.67	0.23	0.76	76.93	(
max	500.00	34.26	15.90	43.21	0.92	0.36	1.00	98.68	!

```
# Import Libraries

import matplotlib.pyplot as plt
import seaborn as sns


# Draw the pie chart of Footedness

Footedness= data['Footedness'].value_counts()
plt.figure(figsize=(10,6))
Footedness.plot(kind='pie',autopct='%1.2f%%')
plt.show()
```



```
# Draw the pie Countplot for Distribution of Footedness Across Nationalities

sns.countplot(x='Nationality', hue='Footedness',data=data, palette='viridis')
plt.title('Distribution of Footedness Across Nationalities')
plt.xlabel('Nationality')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```




```
# Determine which nationality strikers have the highest average number of goals scored.

avg_hg_goal = data.groupby('Nationality') ['Goals Scored'].mean()
avg_hg_goal.sort_values(ascending=False)
```

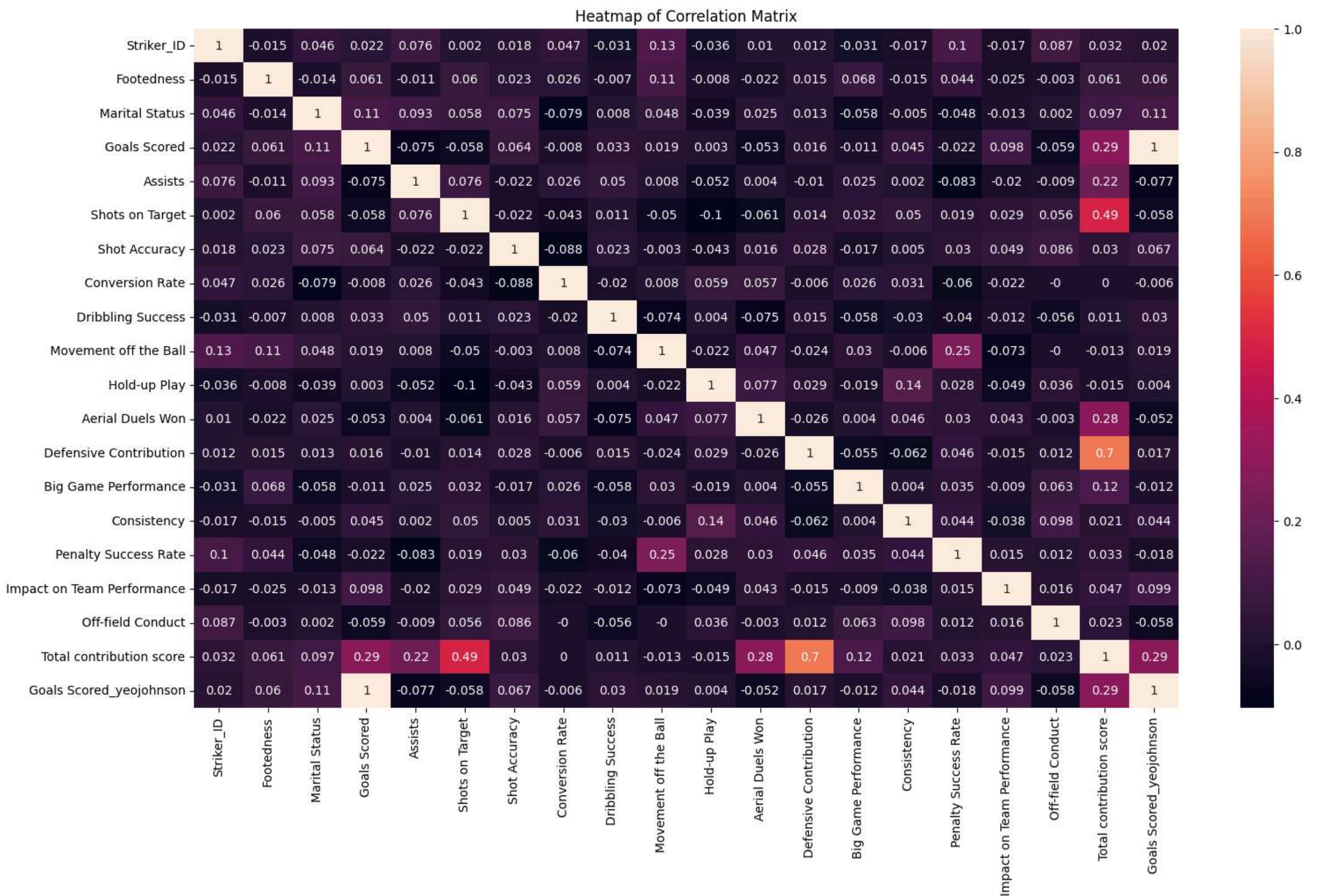
```
Nationality
Brazil     15.804927
Spain      15.196491
France     14.900827
Germany    14.860242
England    14.465756
Name: Goals Scored, dtype: float64
```

```
# Calculate the average conversion rate for players based on their footedness.

avg_conversion_rate = data.groupby('Footedness')['Conversion Rate'].mean()
print(avg_conversion_rate)

footedness_by_nationality = pd.crosstab(data['Nationality'], data['Footedness'])
footedness_by_nationality
```

```
Footedness
0    0.198086
1    0.200592
Name: Conversion Rate, dtype: float64
```

Footedness	0	1
Nationality		
Brazil	42	53
England	50	59
France	42	51
Germany	44	47
Spain	55	57

```
'''Find whether there is any significant difference in consistency rates among
strikers from various nationalities. Before doing the appropriate test, must check
for the assumptions'''

# Correlation Matrix
num_variables = data.select_dtypes(include = ['number']).columns

correl_matrix = round(data[num_variables].corr(), 3)
correl_matrix

plt.figure(figsize=(18, 10))
sns.heatmap(correl_matrix, annot=True)
plt.title('Heatmap of Correlation Matrix')
plt.show()



# Shapiro Normality test
from scipy.stats import shapiro
import pandas as pd
df = data[['Nationality', 'Consistency']]
for nationality in df['Nationality'].unique():
    stat, p = shapiro(df[df['Nationality'] == nationality]['Consistency'])
    print(f'Nationality: {nationality}, p-value: {p}')
```




```
Nationality: Spain, p-value: 0.6272388696670532
Nationality: France, p-value: 0.4209076464176178
Nationality: Germany, p-value: 0.9636307954788208
Nationality: Brazil, p-value: 0.7303183674812317
Nationality: England, p-value: 0.9730228185653687
```

```
#Levene’s test p-value

from scipy.stats import levene

grouped_data = [df[df['Nationality'] == nationality]['Consistency'] for
                nationality in df['Nationality'].unique()]
stat, p = levene(*grouped_data)
print(f'Levene’s test p-value: {p}')
```

```
Levene’s test p-value: 0.8083990350934653
```

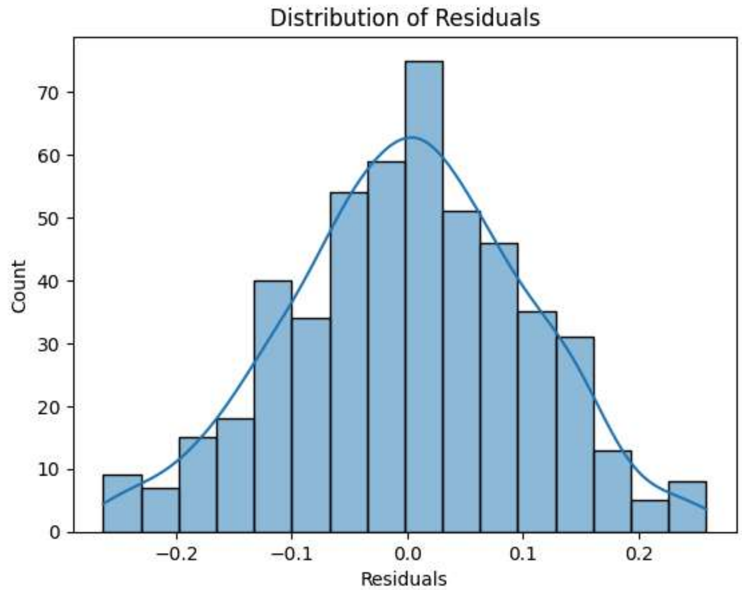
```
# One way Anova test

from scipy.stats import f_oneway

stat, p = f_oneway(*grouped_data)
print(f'ANOVA p-value: {p}')
```

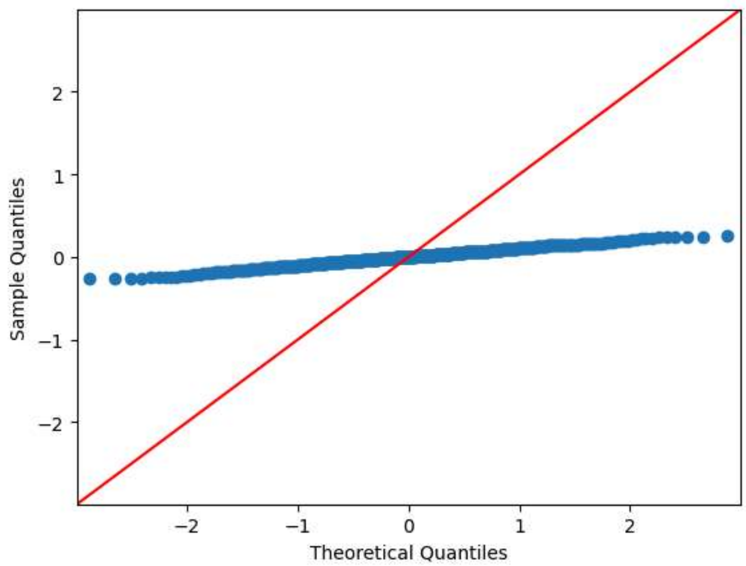
```
ANOVA p-value: 0.19278675901599154
```

```
'''Check if there is any significant correlation between strikers
 Hold-up play and consistency rate. Must check for the assumptions.'''

# Pearson correlation test

from scipy.stats import pearsonr
stat, p = pearsonr(data['Hold-up Play'], data['Consistency'])
print(f'Pearson’s correlation coefficient: {stat}, p-value: {p}')
```

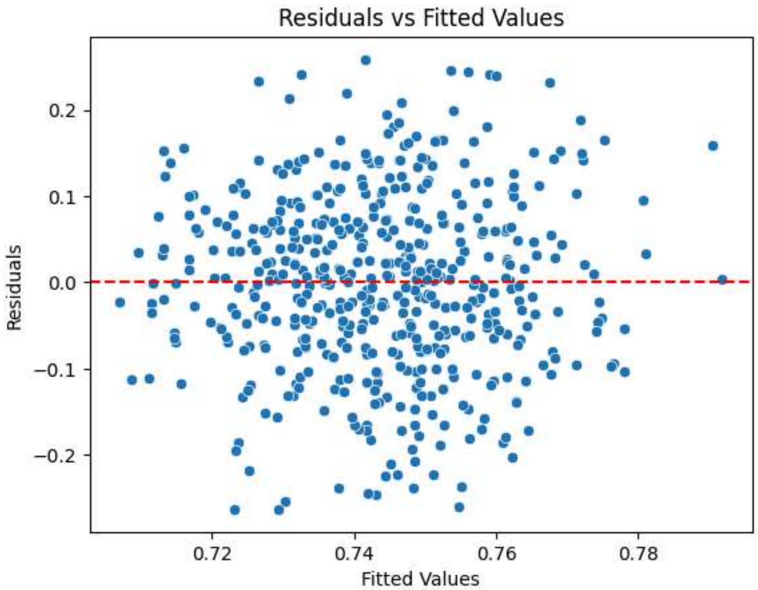
```
Pearson’s correlation coefficient: 0.14504436542869958, p-value: 0.001144397241805525
```

```
'''Check if strikers hold-up play significantly influences their consistency rate.'''

# Linear Regression
import statsmodels.api as sm

X = data['Hold-up Play']
y = data['Consistency']
x_constant = sm.add_constant(X)

model = sm.OLS(y, x_constant).fit()
residuals = model.resid


sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals')
plt.xlabel('Residuals')
plt.show()

sm.qqplot(residuals, line='45')
plt.show()
```






```
# Scatter Plot

fitted_vals = model.predict(x_constant)
sns.scatterplot(x=fitted_vals, y=residuals)
plt.axhline(0, linestyle='--', color='red')
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```




```
#Linear Regression Result
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            Consistency   R-squared:                       0.021
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     10.70
Date:                Tue, 09 Jul 2024   Prob (F-statistic):            0.00114
Time:                        06:29:25   Log-Likelihood:                 429.86
No. Observations:                 500   AIC:                            -855.7
Df Residuals:                     498   BIC:                            -847.3
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const            0.6552      0.027     23.903      0.000       0.601       0.709
Hold-up Play     0.0015      0.000      3.271      0.001       0.001       0.002
==============================================================================
Omnibus:                        1.695   Durbin-Watson:                   2.135
Prob(Omnibus):                  0.428   Jarque-Bera (JB):                1.734
Skew:                          -0.100   Prob(JB):                        0.420
Kurtosis:                       2.792   Cond. No.                         362.
==============================================================================

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

```
data['Total contribution score']=(data['Goals Scored']+data['Assists']+data['Shots on Target']+data['Dribbling Success']
                                  +data['Aerial Duels Won']+data['Consistency']+data['Defensive Contribution']+data['Big Game Performance']
data.head()
```

	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	
0	1	Spain	Left-footed	No	17.483571	10.778533	34.795488	0.677836	0.166241	0.757061	50.921924	71.806409	15
1	2	France	Left-footed	Yes	14.308678	13.728250	31.472436	0.544881	0.192774	0.796818	61.396150	53.726866	19
2	3	Germany	Left-footed	No	18.238443	3.804297	25.417413	0.518180	0.160379	0.666869	65.863945	60.452227	20
3	4	France	Right-footed	No	22.615149	9.688908	20.471443	0.599663	0.184602	0.638776	88.876877	60.511979	22
4	5	France	Left-footed	Yes	13.829233	6.048072	29.887563	0.582982	0.105319	0.591485	75.565531	54.982158	13

```
# Encode the Footedness and marital status

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
data['Footedness'] = label_encoder.fit_transform(data['Footedness'])
data['Marital Status'] = label_encoder.fit_transform(data['Marital Status'])
data.head()
```

	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	
0	1	Spain	0	0	17.483571	10.778533	34.795488	0.677836	0.166241	0.757061	50.921924	71.806409	15
1	2	France	0	1	14.308678	13.728250	31.472436	0.544881	0.192774	0.796818	61.396150	53.726866	19
2	3	Germany	0	0	18.238443	3.804297	25.417413	0.518180	0.160379	0.666869	65.863945	60.452227	20
3	4	France	1	0	22.615149	9.688908	20.471443	0.599663	0.184602	0.638776	88.876877	60.511979	22
4	5	France	0	1	13.829233	6.048072	29.887563	0.582982	0.105319	0.591485	75.565531	54.982158	13

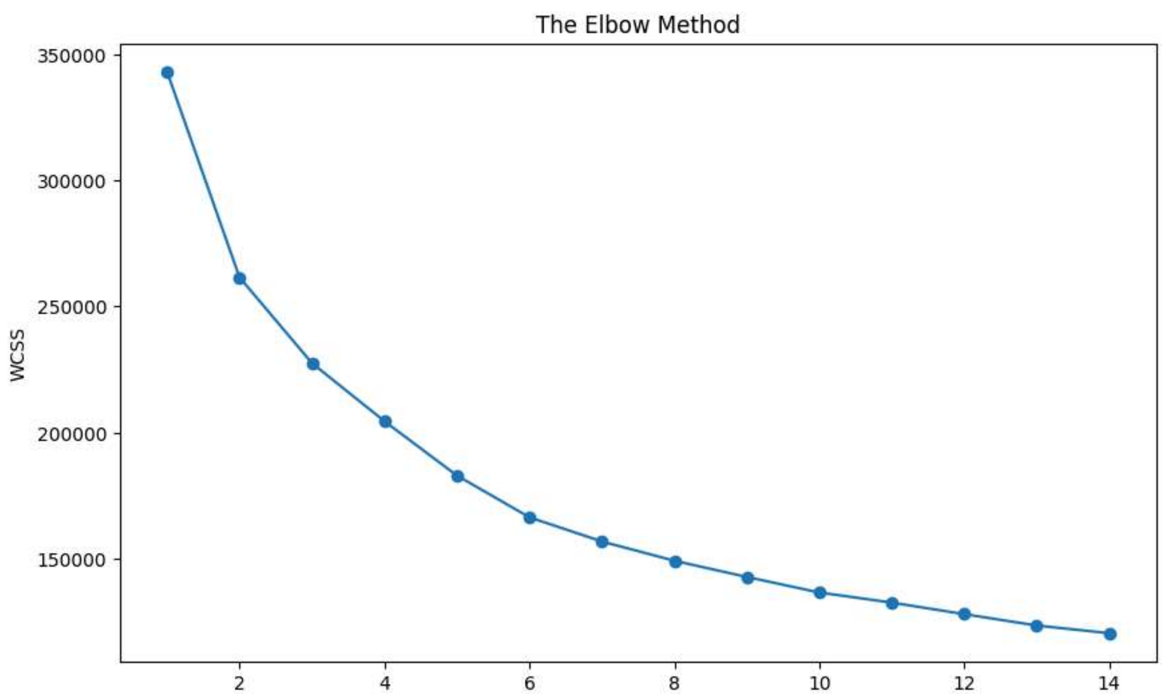
```
# Make dummies of Nationality

dummies = pd.get_dummies(data['Nationality'])
processed_data = pd.concat([data, dummies], axis = 1)
processed_data = processed_data.drop('Nationality', axis = 1)
processed_data.head()
```

	Striker_ID	Footedness	Marital Status	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate
0	1	0	0	17.483571	10.778533	34.795488	0.677836	0.166241
1	2	0	1	14.308678	13.728250	31.472436	0.544881	0.192774
2	3	0	0	18.238443	3.804297	25.417413	0.518180	0.160379
3	4	1	0	22.615149	9.688908	20.471443	0.599663	0.184602
4	5	0	1	13.829233	6.048072	29.887563	0.582982	0.105319

5 rows × 29 columns

```
# Kmean CLustering ,find no of Clusters were made

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

x = processed_data.select_dtypes(include=['number']).drop('Striker_ID', 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)

plt.figure(figsize=(10,6))
plt.plot(range(1,15),wcss,marker='o')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
```




```
from sklearn.cluster import KMeans


final_kmeans = KMeans(n_clusters=2, init='k-means++')
final_kmeans.fit(x)


labels = final_kmeans.labels_

print(labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
[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 1 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]
```

```
processed_data['clusters']= labels
processed_data.head()
```

	Striker_ID	Footedness	Marital Status	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate
0	1	0	0	17.483571	10.778533	34.795488	0.677836	0.166241
1	2	0	1	14.308678	13.728250	31.472436	0.544881	0.192774
2	3	0	0	18.238443	3.804297	25.417413	0.518180	0.160379
3	4	1	0	22.615149	9.688908	20.471443	0.599663	0.184602
4	5	0	1	13.829233	6.048072	29.887563	0.582982	0.105319

5 rows × 30 columns

```
round(processed_data.groupby('clusters')['Total contribution score'].mean(),2)
```

```
clusters
0    104.92
1    126.53
Name: Total contribution score, dtype: float64
```

```
# Mapping Clusters with Strikers type , assign 0:'Best strikers', 1:'Regular strikers'

mapping = {0:'Best strikers', 1:'Regular strikers'}
processed_data['Strikers types'] = processed_data['clusters'].map(mapping)
processed_data.head()
```

	Striker_ID	Footedness	Marital Status	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate
0	1	0	0	17.483571	10.778533	34.795488	0.677836	0.166241
1	2	0	1	14.308678	13.728250	31.472436	0.544881	0.192774
2	3	0	0	18.238443	3.804297	25.417413	0.518180	0.160379
3	4	1	0	22.615149	9.688908	20.471443	0.599663	0.184602
4	5	0	1	13.829233	6.048072	29.887563	0.582982	0.105319

5 rows × 31 columns

```
# Delete cluster column

processed_data=processed_data.drop('clusters',axis=1)
processed_data.head()
```

	Striker_ID	Footedness	Marital Status	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate
0	1	0	0	17.483571	10.778533	34.795488	0.677836	0.166241
1	2	0	1	14.308678	13.728250	31.472436	0.544881	0.192774
2	3	0	0	18.238443	3.804297	25.417413	0.518180	0.160379
3	4	1	0	22.615149	9.688908	20.471443	0.599663	0.184602
4	5	0	1	13.829233	6.048072	29.887563	0.582982	0.105319

5 rows × 30 columns

```
mapping = {'Best strikers':0,'Regular strikers':1}
processed_data['Strikers types'] = processed_data['Strikers types'].map(mapping)
processed_data.head()
```

	Striker_ID	Footedness	Marital Status	Goals Scored	Assists	Shots on Target	Shot Accuracy	Conversion Rate
0	1	0	0	17.483571	10.778533	34.795488	0.677836	0.166241
1	2	0	1	14.308678	13.728250	31.472436	0.544881	0.192774
2	3	0	0	18.238443	3.804297	25.417413	0.518180	0.160379
3	4	1	0	22.615149	9.688908	20.471443	0.599663	0.184602
4	5	0	1	13.829233	6.048072	29.887563	0.582982	0.105319

5 rows × 30 columns

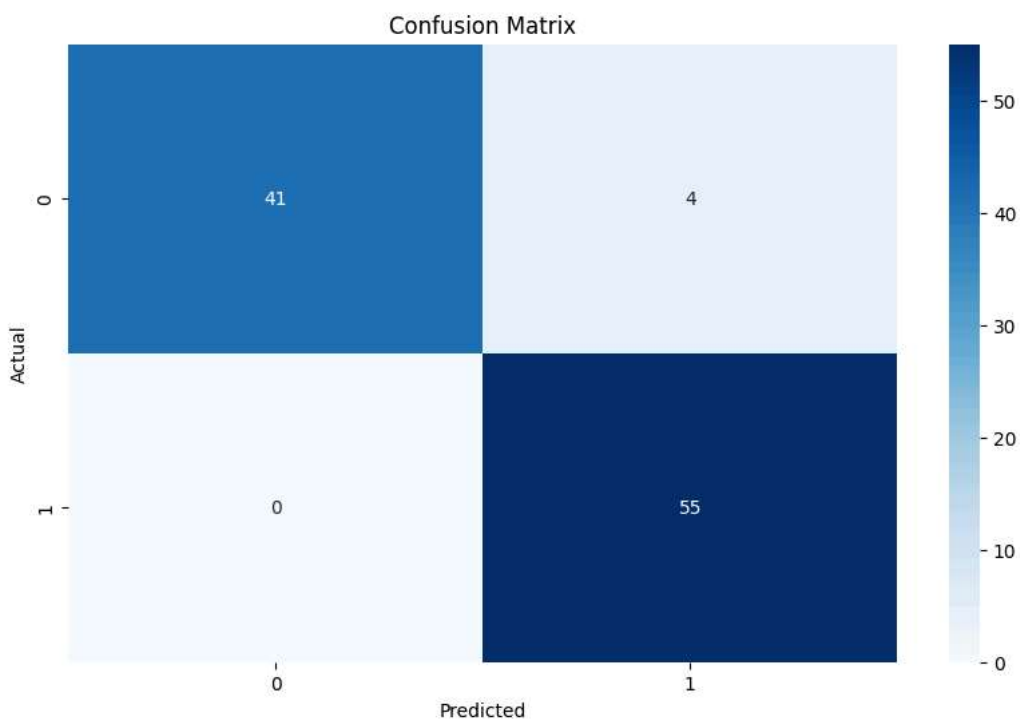
```
# Feature Selection

x = processed_data.drop(['Striker_ID', 'Strikers types'], axis = 1)
y = processed_data['Strikers types']


# Standarising Features With the help of Stanadrd Scaler

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_x=scaler.fit_transform(x)
scaled_x
```

```
array([[-1.07047781, -1.03252879,  0.50002889, ..., -0.47801802,
        -0.47169258,  1.86125917],
       [-1.07047781,  0.968496  , -0.14874861, ...,  2.09197134,
        -0.47169258, -0.53727069],
       [-1.07047781, -1.03252879,  0.65428418, ..., -0.47801802,
         2.12002488, -0.53727069],
       ...,
       [-1.07047781,  0.968496  , -0.20195464, ..., -0.47801802,
        -0.47169258, -0.53727069],
       [ 0.93416229,  0.968496  , -0.90212639, ..., -0.47801802,
        -0.47169258, -0.53727069],
       [-1.07047781, -1.03252879, -1.4203297 , ..., -0.47801802,
        -0.47169258, -0.53727069]])
```

```
# Split the data set into Train and Test

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)
```

```
# Apply ml model (logistic regression)

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix

Lr=LogisticRegression()
```

```
confusion_matrix=confusion_matrix(y_test,y_pred)
plt.figure(figsize=(10,6))
sns.heatmap(confusion_matrix,annot=True,fmt="d",cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
from scipy.stats import yeojohnson

def yeojohnson_transformation(data,col_name):
  transformed_data,_= yeojohnson(data[col_name])
  data[f'{col_name}_yeojohnson']=transformed_data
  stat,p_value=shapiro(data[f'{col_name}_yeojohnson'])
  kdeplot= sns.kdeplot(data[f'{col_name}_yeojohnson'])
  plt.title(f'yeojohnson Transformation of {col_name}')
  plt.show()
  print(f'p-value: {p_value}')


yeojohnson_transformation(data,'Goals Scored')
```