In [347…

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, ro
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import eli5
from eli5.sklearn import PermutationImportance
from collections import Counter
```

In [348…

```
Fifa_22_database = pd.read_csv("C:\\Users\\welcome\\Desktop\\Msc Data Science Cov Uni\
```

In [349…

```
pd.set_option('display.max_rows', 500)
```

In [350…

```
Fifa_22_database.head()
```

Out[350]:

	sofifa_id	player_url	short_name	long_name	player_positions	overa
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	9
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	9
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	9
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	9
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	9

5 rows × 110 columns

In [351…

```
print(Fifa_22_database.iloc[0:13, 0:50])
```

```
    sofifa_id                                         player_url  \
0      158023  https://sofifa.com/player/158023/lionel-messi/...
1      188545  https://sofifa.com/player/188545/robert-lewand...
2       20801  https://sofifa.com/player/20801/c-ronaldo-dos-...
3      190871  https://sofifa.com/player/190871/neymar-da-sil...
4      192985  https://sofifa.com/player/192985/kevin-de-bruy...
5      200389  https://sofifa.com/player/200389/jan-oblak/220002
6      231747  https://sofifa.com/player/231747/kylian-mbappe...
7      167495  https://sofifa.com/player/167495/manuel-neuer/...
8      192448  https://sofifa.com/player/192448/marc-andre-te...
9      202126  https://sofifa.com/player/202126/harry-kane/22...
10     215914  https://sofifa.com/player/215914/ngolo-kante/2...
11     165153  https://sofifa.com/player/165153/karim-benzema...
12     192119  https://sofifa.com/player/192119/thibaut-court...

           short_name                            long_name player_positions  \
0            L. Messi       Lionel Andrés Messi Cuccittini       RW, ST, CF
1      R. Lewandowski                   Robert Lewandowski               ST
2   Cristiano Ronaldo  Cristiano Ronaldo dos Santos Aveiro           ST, LW
3           Neymar Jr       Neymar da Silva Santos Júnior          LW, CAM
4         K. De Bruyne                      Kevin De Bruyne          CM, CAM
5            J. Oblak                            Jan Oblak               GK
6           K. Mbappé                 Kylian Mbappé Lottin           ST, LW
7            M. Neuer                   Manuel Peter Neuer               GK
8       M. ter Stegen               Marc-André ter Stegen               GK
9             H. Kane                           Harry Kane               ST
10           N. Kanté                         N'Golo Kanté          CDM, CM
11         K. Benzema                        Karim Benzema           CF, ST
12        T. Courtois                     Thibaut Courtois               GK

    overall  potential    value_eur  wage_eur  age  ... dribbling  defending  \
0        93         93   78000000.0  320000.0   34  ...      95.0       34.0
1        92         92  119500000.0  270000.0   32  ...      86.0       44.0
2        91         91   45000000.0  270000.0   36  ...      88.0       34.0
3        91         91  129000000.0  270000.0   29  ...      94.0       37.0
4        91         91  125500000.0  350000.0   30  ...      88.0       64.0
5        91         93  112000000.0  130000.0   28  ...       NaN        NaN
6        91         95  194000000.0  230000.0   22  ...      92.0       36.0
7        90         90   13500000.0   86000.0   35  ...       NaN        NaN
8        90         92   99000000.0  250000.0   29  ...       NaN        NaN
9        90         90  129500000.0  240000.0   27  ...      83.0       47.0
10       90         90  100000000.0  230000.0   30  ...      82.0       87.0
11       89         89   66000000.0  350000.0   33  ...      87.0       39.0
12       89         91   85500000.0  250000.0   29  ...       NaN        NaN

    physic  attacking_crossing attacking_finishing attacking_heading_accuracy  \
0     65.0                  85                  95                         70
1     82.0                  71                  95                         90
2     75.0                  87                  95                         90
3     63.0                  85                  83                         63
4     78.0                  94                  82                         55
5      NaN                  13                  11                         15
6     77.0                  78                  93                         72
7      NaN                  15                  13                         25
8      NaN                  18                  14                         11
9     83.0                  80                  94                         86
10    83.0                  68                  65                         54
11    77.0                  75                  90                         89
12     NaN                  14                  14                         13
```

```
    attacking_short_passing attacking_volleys  skill_dribbling skill_curve
0                        91                88               96          93
1                        85                89               85          79
2                        80                86               88          81
3                        86                86               95          88
4                        94                82               88          85
5                        43                13               12          13
6                        85                83               93          80
7                        60                11               30          14
8                        61                14               21          18
9                        85                88               83          83
10                       82                56               79          49
11                       86                86               87          81
12                       33                12               13          19

[13 rows x 50 columns]
```

In [352…

```
Fifa_22_database.columns
```

Out[352]:

```
Index(['sofifa_id', 'player_url', 'short_name', 'long_name',
       'player_positions', 'overall', 'potential', 'value_eur', 'wage_eur',
       'age',
       ...
       'lcb', 'cb', 'rcb', 'rb', 'gk', 'player_face_url', 'club_logo_url',
       'club_flag_url', 'nation_logo_url', 'nation_flag_url'],
      dtype='object', length=110)
```

In [353…

```
Fifa_22_database.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19239 entries, 0 to 19238
Columns: 110 entries, sofifa_id to nation_flag_url
dtypes: float64(16), int64(44), object(50)
memory usage: 16.1+ MB
```

In [354…

```
Fifa_22_database.isnull().sum()
```

Out[354]:

```
sofifa_id                        0
player_url                       0
short_name                       0
long_name                        0
player_positions                 0
overall                          0
potential                        0
value_eur                       74
wage_eur                        61
age                              0
dob                              0
height_cm                        0
weight_kg                        0
club_team_id                    61
club_name                       61
league_name                     61
league_level                    61
club_position                   61
club_jersey_number              61
club_loaned_from             18137
club_joined                   1163
club_contract_valid_until       61
nationality_id                   0
nationality_name                 0
nation_team_id               18480
nation_position              18480
nation_jersey_number         18480
preferred_foot                   0
weak_foot                        0
skill_moves                      0
international_reputation         0
work_rate                        0
body_type                        0
real_face                        0
release_clause_eur            1176
player_tags                  17798
player_traits                 9841
pace                          2132
shooting                      2132
passing                       2132
dribbling                     2132
defending                     2132
physic                        2132
attacking_crossing               0
attacking_finishing              0
attacking_heading_accuracy       0
attacking_short_passing          0
attacking_volleys                0
skill_dribbling                  0
skill_curve                      0
skill_fk_accuracy                0
skill_long_passing               0
skill_ball_control               0
movement_acceleration            0
movement_sprint_speed            0
movement_agility                 0
movement_reactions               0
movement_balance                 0
power_shot_power                 0
power_jumping                    0
```

```
power_stamina                  0
power_strength                 0
power_long_shots               0
mentality_aggression           0
mentality_interceptions        0
mentality_positioning          0
mentality_vision               0
mentality_penalties            0
mentality_composure            0
defending_marking_awareness    0
defending_standing_tackle      0
defending_sliding_tackle       0
goalkeeping_diving             0
goalkeeping_handling           0
goalkeeping_kicking            0
goalkeeping_positioning        0
goalkeeping_reflexes           0
goalkeeping_speed          17107
ls                             0
st                             0
rs                             0
lw                             0
lf                             0
cf                             0
rf                             0
rw                             0
lam                            0
cam                            0
ram                            0
lm                             0
lcm                            0
cm                             0
rcm                            0
rm                             0
lwb                            0
ldm                            0
cdm                            0
rdm                            0
rwb                            0
lb                             0
lcb                            0
cb                             0
rcb                            0
rb                             0
gk                             0
player_face_url                0
club_logo_url                 61
club_flag_url                 61
nation_logo_url            18480
nation_flag_url                0
dtype: int64
```

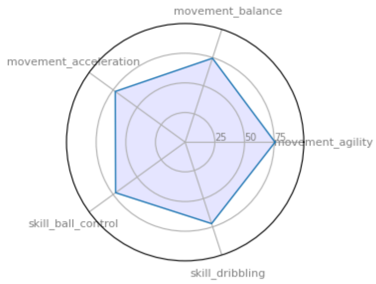
In [355…

```
player_features = (
    'movement_acceleration', 'movement_agility', 'mentality_aggression',
    'movement_balance', 'skill_ball_control', 'mentality_composure',
    'attacking_crossing', 'skill_dribbling', 'skill_fk_accuracy',
    'attacking_finishing', 'goalkeeping_diving', 'goalkeeping_handling',
    'goalkeeping_kicking', 'goalkeeping_positioning', 'goalkeeping_reflexes',
    'attacking_heading_accuracy', 'mentality_interceptions', 'power_jumping',
    'skill_long_passing', 'power_long_shots', 'defending_marking_awareness', 'mentalit
```

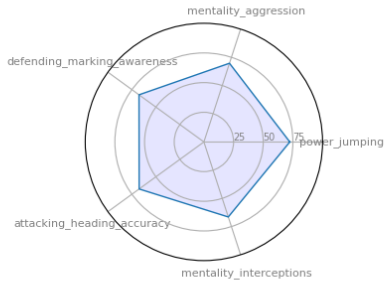
```
)

from math import pi
idx = 1
plt.figure(figsize=(15,45))
for position_name, features in Fifa_22_database.groupby(Fifa_22_database['club_positio
    top_features = dict(features.nlargest(5))

    # number of variable
    categories=top_features.keys()
    N = len(categories)

    # We are going to plot the first line of the data frame.
    # But we need to repeat the first value to close the circular graph:
    values = list(top_features.values())
    values += values[:1]

    # What will be the angle of each axis in the plot? (we divide the plot / number of
    angles = [n / float(N) * 2 * pi for n in range(N)]
    angles += angles[:1]

    # Initialise the spider plot
    ax = plt.subplot(10, 3, idx, polar=True)

    # Draw one axe per variable + add labels labels yet
    plt.xticks(angles[:-1], categories, color='grey', size=8)

 # Draw ylabels
    ax.set_rlabel_position(0)
    plt.yticks([25,50,75], ["25","50","75"], color="grey", size=7)
    plt.ylim(0,100)

    plt.subplots_adjust(hspace = 0.5)

    # Plot data
    ax.plot(angles, values, linewidth=1, linestyle='solid')
    # Fill area
    ax.fill(angles, values, 'b', alpha=0.1)

    plt.title(position_name, size=11, y=1.1)

    idx += 1
```

```
C:\Users\welcome\AppData\Local\Temp\ipykernel_22484\151606241.py:14: FutureWarning: I
ndexing with multiple keys (implicitly converted to a tuple of keys) will be deprecat
ed, use a list instead.
  for position_name, features in Fifa_22_database.groupby(Fifa_22_database['club_posi
tion'])[player_features].mean().iterrows():
```

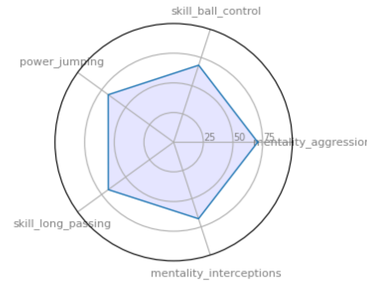
CAM



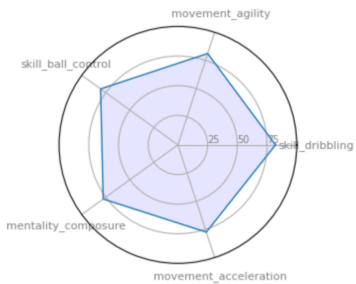
CB



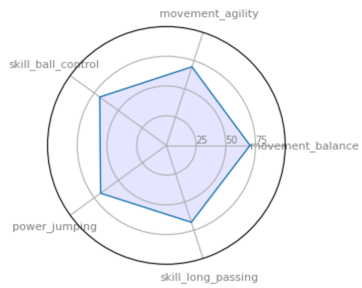
CDM



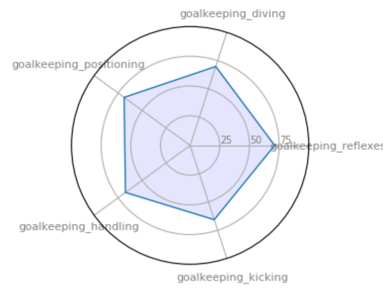
CF



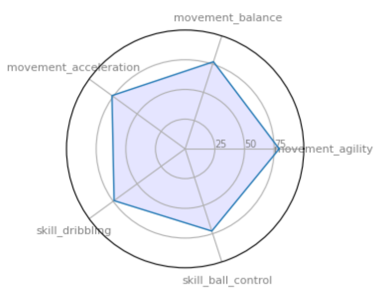
CM



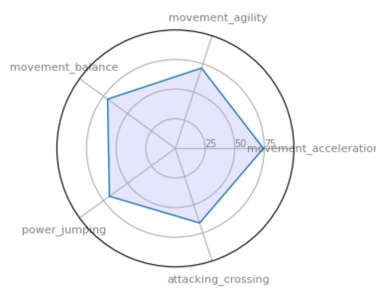
GK



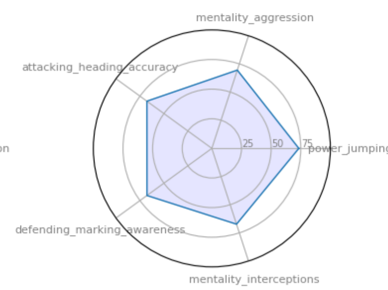
LAM



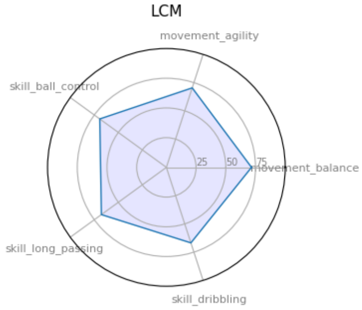
LB



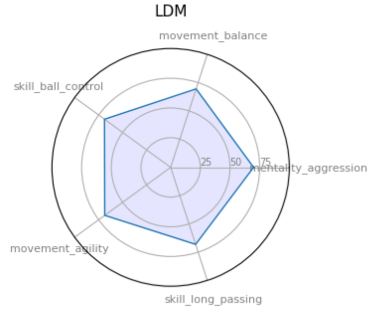
LCB



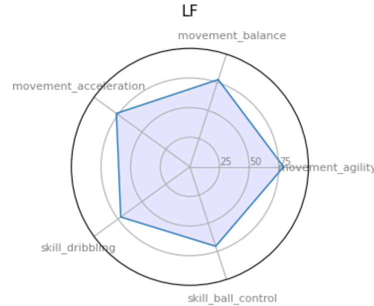
LCM



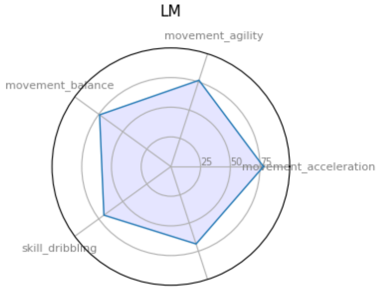
LDM



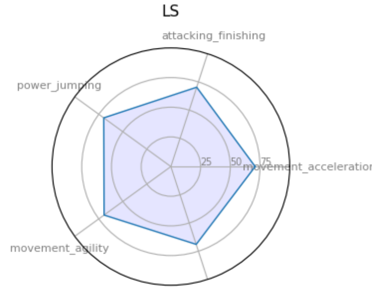
LF



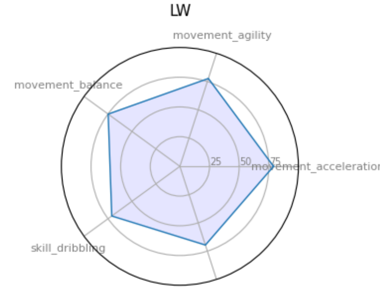
LM



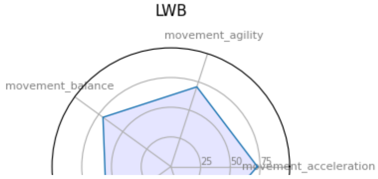
LS



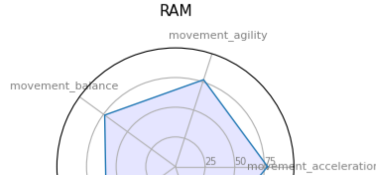
LW



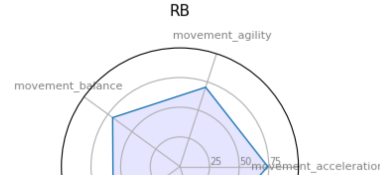
LWB

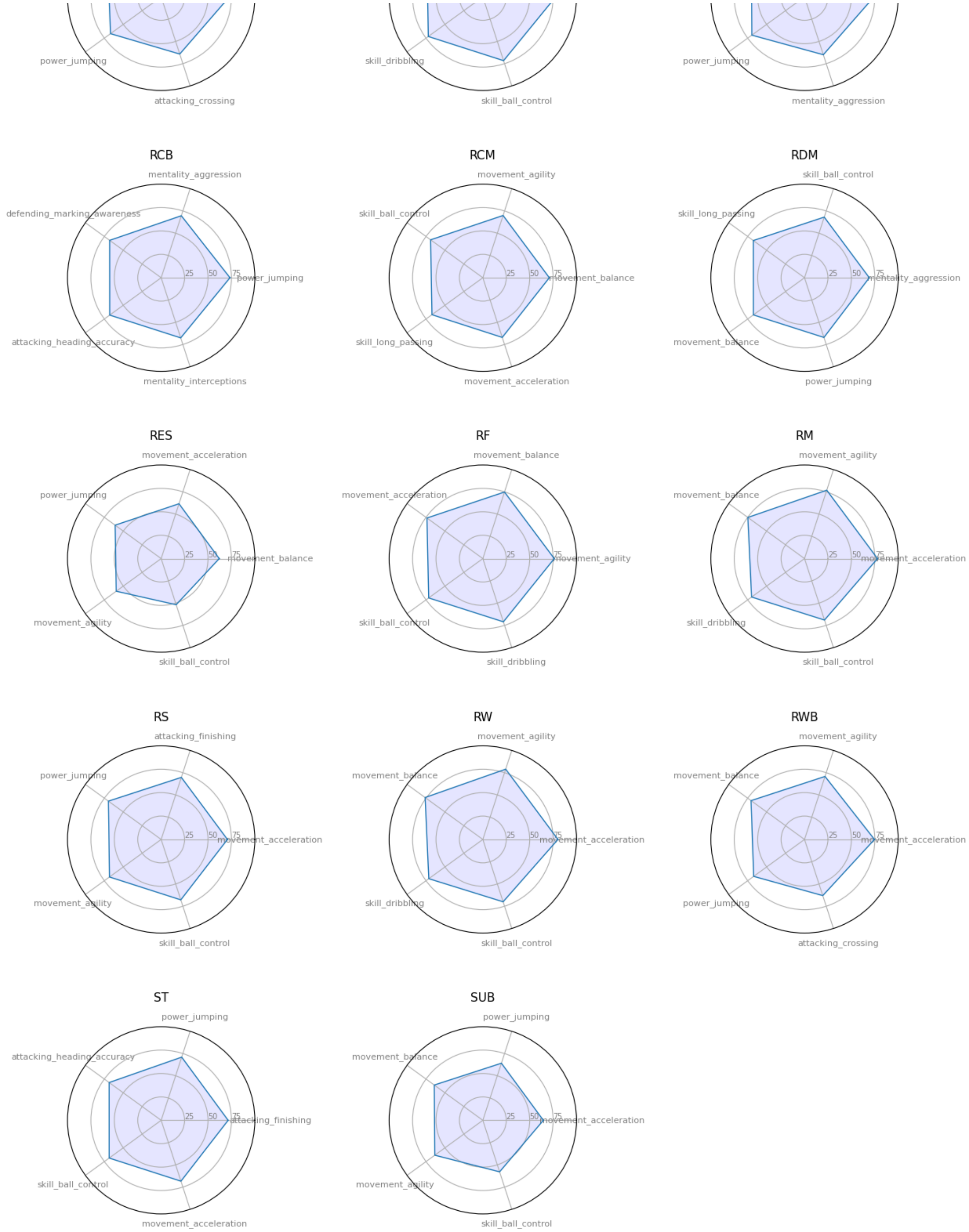


RAM



RB





In [356…

```
# Most valued player
most_valued_player = Fifa_22_database.loc[Fifa_22_database['value_eur'].idxmax()]
print('Most valued player:')
print(most_valued_player['short_name'])  # Assuming the player's name is stored in the

# Highest earner
highest_earner = Fifa_22_database.loc[Fifa_22_database['wage_eur'].idxmax()]
print('\nHighest earner:')
print(highest_earner['short_name'])  # Assuming the player's name is stored in the 'sh
```

```
print("--" * 40)
print("\nTop Earners:")
# Sorting the DataFrame by 'wage_eur' in descending order to get the top earners
top_earners = Fifa_22_database.sort_values(by='wage_eur', ascending=False).head(10)
print(top_earners[['short_name', 'wage_eur']])
```

```
Most valued player:
K. Mbappé

Highest earner:
K. De Bruyne
--------------------------------------------------------------------------------

Top Earners:
           short_name  wage_eur
4        K. De Bruyne  350000.0
11         K. Benzema  350000.0
0            L. Messi  320000.0
14           Casemiro  310000.0
24           T. Kroos  310000.0
27        R. Sterling  290000.0
2   Cristiano Ronaldo  270000.0
3           Neymar Jr  270000.0
1      R. Lewandowski  270000.0
17           M. Salah  270000.0
```

In [357…

```
#Data Cleaning and removing NaN values to null values(0's)

columns_required_only = ['short_name', 'overall', 'potential', 'age', 'weak_foot', 'sk
Fifa_22_database = Fifa_22_database[columns_required_only].fillna(0) #fillna() is the
```

In [358…

```
Fifa_22_database.head(50)
```

Out[358]:

	short_name	overall	potential	age	weak_foot	skill_moves	pace	shooting	passing	dribbling
0	L. Messi	93	93	34	4	4	85.0	92.0	91.0	95.0
1	R. Lewandowski	92	92	32	4	4	78.0	92.0	79.0	86.0
2	Cristiano Ronaldo	91	91	36	4	5	87.0	94.0	80.0	88.0
3	Neymar Jr	91	91	29	5	5	91.0	83.0	86.0	94.0
4	K. De Bruyne	91	91	30	5	4	76.0	86.0	93.0	88.0
5	J. Oblak	91	93	28	3	1	0.0	0.0	0.0	0.0
6	K. Mbappé	91	95	22	4	5	97.0	88.0	80.0	92.0
7	M. Neuer	90	90	35	4	1	0.0	0.0	0.0	0.0
8	M. ter Stegen	90	92	29	4	1	0.0	0.0	0.0	0.0
9	H. Kane	90	90	27	5	3	70.0	91.0	83.0	83.0
10	N. Kanté	90	90	30	3	2	78.0	66.0	75.0	82.0
11	K. Benzema	89	89	33	4	4	76.0	86.0	81.0	87.0
12	T. Courtois	89	91	29	3	1	0.0	0.0	0.0	0.0
13	H. Son	89	89	28	5	4	88.0	87.0	82.0	86.0
14	Casemiro	89	89	29	3	2	65.0	73.0	76.0	73.0
15	V. van Dijk	89	89	29	3	2	78.0	60.0	71.0	72.0
16	S. Mané	89	89	29	4	4	91.0	83.0	80.0	89.0
17	M. Salah	89	89	29	3	4	90.0	87.0	81.0	90.0
18	Ederson	89	91	27	3	1	0.0	0.0	0.0	0.0
19	J. Kimmich	89	90	26	4	3	70.0	73.0	86.0	84.0
20	Alisson	89	90	28	3	1	0.0	0.0	0.0	0.0
21	G. Donnarumma	89	93	22	3	1	0.0	0.0	0.0	0.0
22	Sergio Ramos	88	88	35	3	3	70.0	70.0	76.0	74.0
23	L. Suárez	88	88	34	4	3	72.0	90.0	82.0	84.0
24	T. Kroos	88	88	31	5	3	53.0	81.0	91.0	81.0
25	R. Lukaku	88	88	28	4	3	84.0	87.0	74.0	78.0
26	K. Navas	88	88	34	3	1	0.0	0.0	0.0	0.0
27	R. Sterling	88	89	26	3	4	91.0	82.0	79.0	87.0
28	Bruno Fernandes	88	89	26	3	4	75.0	86.0	89.0	84.0
29	E. Haaland	88	93	20	3	3	89.0	91.0	65.0	80.0
30	S. Agüero	87	87	33	4	4	71.0	89.0	75.0	87.0

	short_name	overall	potential	age	weak_foot	skill_moves	pace	shooting	passing	dribbling
31	H. Lloris	87	87	34	1	1	0.0	0.0	0.0	0.0
32	L. Modrić	87	87	35	4	4	73.0	76.0	89.0	88.0
33	Á. Di María	87	87	33	2	5	83.0	81.0	86.0	87.0
34	W. Szczęsny	87	87	31	3	1	0.0	0.0	0.0	0.0
35	T. Müller	87	87	31	4	3	67.0	84.0	83.0	80.0
36	C. Immobile	87	87	31	4	3	86.0	87.0	67.0	81.0
37	P. Pogba	87	87	28	4	5	71.0	81.0	86.0	86.0
38	M. Verratti	87	87	28	4	4	64.0	61.0	87.0	91.0
39	Marquinhos	87	90	27	3	3	81.0	53.0	75.0	74.0
40	L. Goretzka	87	88	26	4	3	81.0	82.0	82.0	84.0
41	P. Dybala	87	88	27	3	4	84.0	86.0	86.0	90.0
42	A. Robertson	87	88	27	2	3	84.0	61.0	81.0	81.0
43	F. de Jong	87	92	24	3	4	81.0	69.0	85.0	88.0
44	T. Alexander-Arnold	87	92	22	4	3	79.0	68.0	88.0	80.0
45	J. Sancho	87	91	21	3	5	81.0	76.0	82.0	91.0
46	Rúben Dias	87	91	24	4	2	61.0	38.0	65.0	68.0
47	G. Chiellini	86	86	36	3	2	68.0	46.0	60.0	59.0
48	S. Handanovič	86	86	36	3	1	0.0	0.0	0.0	0.0
49	M. Hummels	86	86	32	3	2	56.0	59.0	77.0	73.0

In [359…

```
Fifa_22_database.isnull().sum()
```

Out[359]:

```
short_name     0
overall        0
potential      0
age            0
weak_foot      0
skill_moves    0
pace           0
shooting       0
passing        0
dribbling      0
defending      0
physic         0
dtype: int64
```

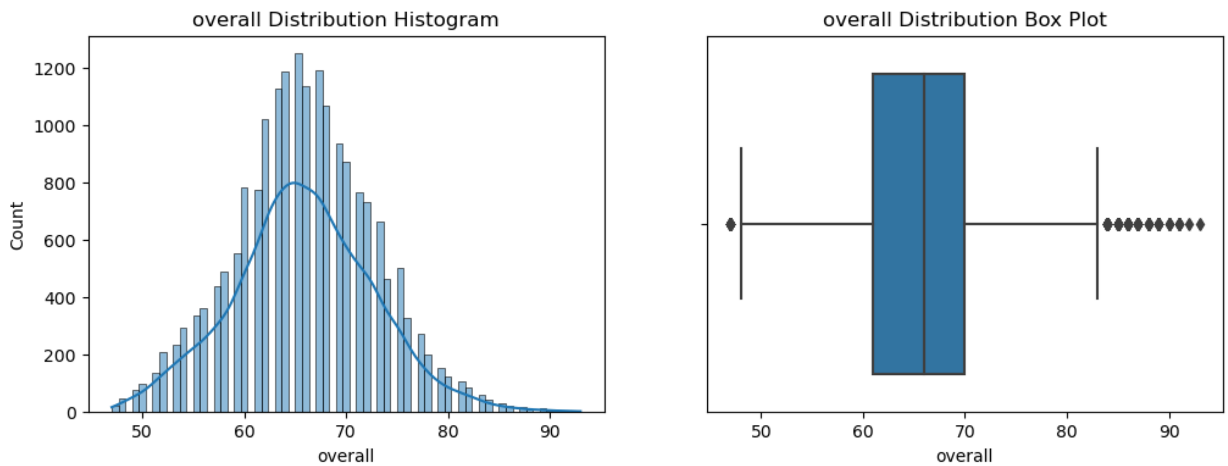
In [360…

```
#Graphical Representation of each entity/column

def plt_df(data, col):
    fig, ax = plt.subplots(1, 2, figsize=(12, 4))

    sns.histplot(data=data, x=col, kde=True, ax=ax[0])
```

```
    sns.boxplot(data=data, x=col, ax=ax[1])

    ax[0].set_title(f"{col} Distribution Histogram")
    ax[1].set_title(f"{col} Distribution Box Plot")

    plt.show()
```

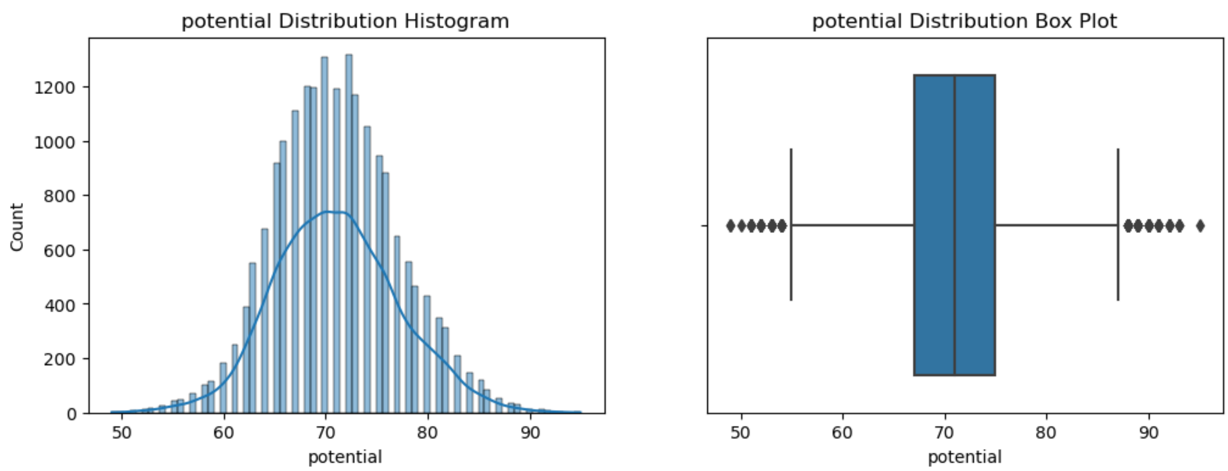
In [361…

```
plt_df(Fifa_22_database,'overall')
```



In [362…

```
plt_df(Fifa_22_database,'potential')
```



In [363…

```
#Create a new column and add all skill attributes to one column
import os
Fifa_22_database['Players Skill Set (In Total)'] = 0

# Save the updated dataset
Fifa_22_database.to_csv("Fifa22updates.csv", index=False)
current_directory = os.getcwd()

# Print the location of the saved CSV file
print("The CSV file is saved in the following location:")
print(current_directory + "\\Fifa22updates.csv")  # On Windows

Fifa_22_database
```

```
The CSV file is saved in the following location:
C:\Users\welcome\Fifa22updates.csv
```

Out[363]:

	short_name	overall	potential	age	weak_foot	skill_moves	pace	shooting	passing	dri
0	L. Messi	93	93	34	4	4	85.0	92.0	91.0	
1	R. Lewandowski	92	92	32	4	4	78.0	92.0	79.0	
2	Cristiano Ronaldo	91	91	36	4	5	87.0	94.0	80.0	
3	Neymar Jr	91	91	29	5	5	91.0	83.0	86.0	
4	K. De Bruyne	91	91	30	5	4	76.0	86.0	93.0	
...	...	...	...	...	...	...	...	...	...	
19234	Song Defu	47	52	22	3	2	58.0	35.0	46.0	
19235	C. Porter	47	59	19	3	2	59.0	39.0	50.0	
19236	N. Logue	47	55	21	3	2	60.0	37.0	45.0	
19237	L. Rudden	47	60	19	3	2	68.0	46.0	36.0	
19238	E. Lalchhanchhuaha	47	60	19	3	2	68.0	38.0	45.0	

19239 rows × 13 columns

In [364...

```
# Calculate the average skill set score and round it
Fifa_22_database['Players Skill Set (In Total)'] = (Fifa_22_database["pace"] + Fifa_22
                                                    Fifa_22_database["passing"] + Fifa_2
                                                    Fifa_22_database["defending"]) / 5

Fifa_22_database['Players Skill Set (In Total)'] = Fifa_22_database['Players Skill Set

# Save the updated dataset to a CSV file named "Fifa22updates.csv"
Fifa_22_database.to_csv("Fifa22updates.csv", index=False)

# Apply a mapping to convert skill set scores into 0's and 1's based on the condition
Fifa_22_database['Players Skill Set (In Total)'] = Fifa_22_database['Players Skill Set

Fifa_22_database
```

Out[364]:

	short_name	overall	potential	age	weak_foot	skill_moves	pace	shooting	passing	dri
0	L. Messi	93	93	34	4	4	85.0	92.0	91.0	
1	R. Lewandowski	92	92	32	4	4	78.0	92.0	79.0	
2	Cristiano Ronaldo	91	91	36	4	5	87.0	94.0	80.0	
3	Neymar Jr	91	91	29	5	5	91.0	83.0	86.0	
4	K. De Bruyne	91	91	30	5	4	76.0	86.0	93.0	
...	...	...	...	...	...	...	...	...	...	
19234	Song Defu	47	52	22	3	2	58.0	35.0	46.0	
19235	C. Porter	47	59	19	3	2	59.0	39.0	50.0	
19236	N. Logue	47	55	21	3	2	60.0	37.0	45.0	
19237	L. Rudden	47	60	19	3	2	68.0	46.0	36.0	
19238	E. Lalchhanchhuaha	47	60	19	3	2	68.0	38.0	45.0	

19239 rows × 13 columns

In [365…

```
#Split Data into features and targets

X = Fifa_22_database[['overall','potential','weak_foot','skill_moves']]
Y = Fifa_22_database.iloc[:,-1]

# Check the class distribution before oversampling
print("Class distribution before oversampling:")
print(Y.value_counts())

from imblearn.over_sampling import RandomOverSampler
# Define the oversampler
oversampler = RandomOverSampler(sampling_strategy='minority')

# Resample the data
X_resampled, y_resampled = oversampler.fit_resample(X, Y)

# Check the class distribution after oversampling
print("\nClass distribution after oversampling:")
print(pd.Series(y_resampled).value_counts())

# Convert the resampled data back to a Pandas DataFrame
data_resampled = pd.DataFrame(np.concatenate([X_resampled, y_resampled.values.reshape(

# Save the resampled data to a new file
data_resampled.to_csv(".csv", index=False)
```

```
Class distribution before oversampling:
0    11487
1     7752
Name: Players Skill Set (In Total), dtype: int64

Class distribution after oversampling:
1    11487
0    11487
Name: Players Skill Set (In Total), dtype: int64
```

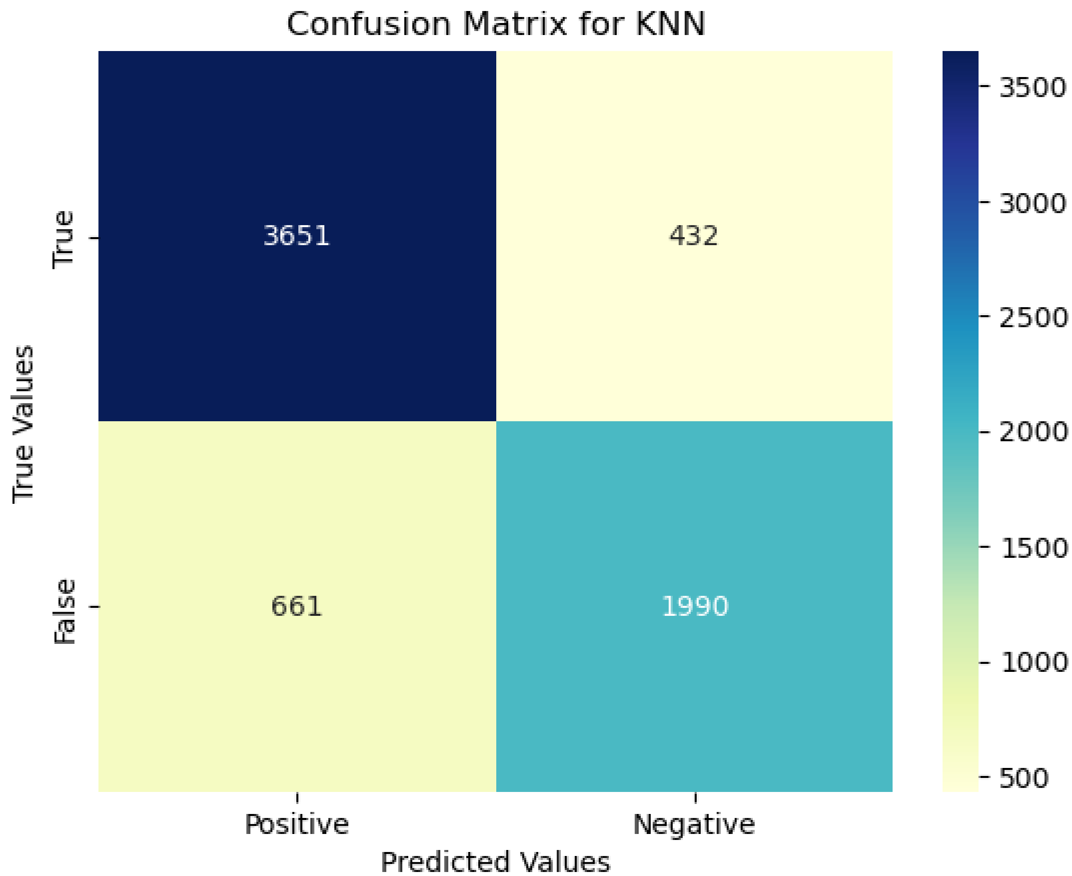
In [367…

```
#Train Test split model version
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.35, random_state
accuracy_scores = []

#K neighbors classification
kNN = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
kNN.fit(X_train, Y_train)

Y_pred_knn = kNN.predict(X_test)

# Get the accuracy score
accuracy = 100*accuracy_score(Y_test, Y_pred_knn)
print("Accuracy: {:.2f}%".format(accuracy))

# Get the classification report
cls_rpt = classification_report(Y_test, Y_pred)
print("\nClassification Report:\n", cls_rpt)

accuracy_scores.append(accuracy_score(Y_test, Y_pred_knn))
```

```
Accuracy: 81.45%

Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.89      0.87      4083
           1       0.82      0.75      0.78      2651

    accuracy                           0.84      6734
   macro avg       0.83      0.82      0.83      6734
weighted avg       0.84      0.84      0.84      6734
```

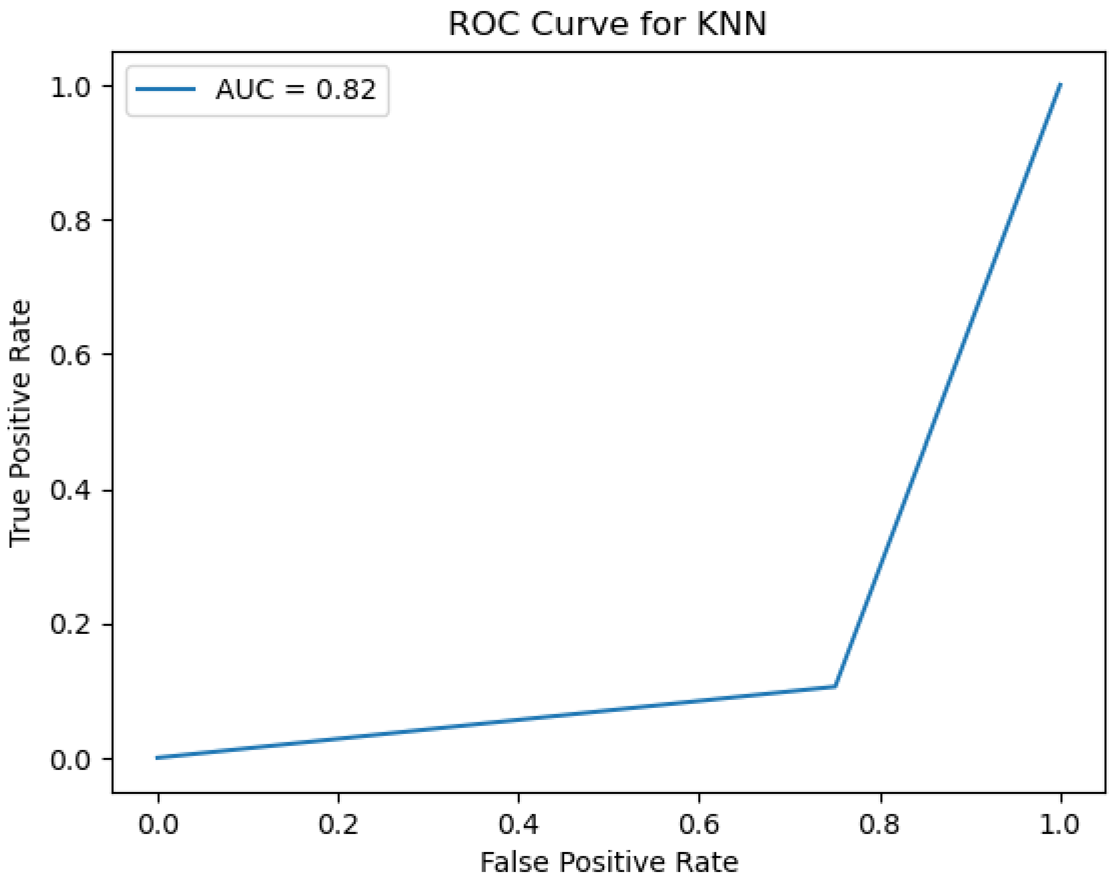
In [368…

```
confusion = confusion_matrix(Y_test, Y_pred)

# Convert the confusion matrix into a data frame
confusion_df = pd.DataFrame(confusion, index=["True", "False"], columns=["Positive", '

# Plot the confusion matrix
sns.heatmap(confusion_df, annot=True, fmt="d", cmap="YlGnBu")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.title("Confusion Matrix for KNN")
plt.show()
```



In [369…

```
y_true = [0 if y == 1.0 else 1 for y in Y_test]

# Get the ROC AUC score
auc = roc_auc_score(Y_test, Y_pred)

# Get the false positive rate and true positive rate for different thresholds
fpr, tpr, thresholds = roc_curve(y_true, Y_pred)

# Plot the ROC curve
plt.plot(fpr, tpr, label="AUC = {:.2f}".format(auc))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for KNN")
plt.legend()
plt.show()
```



In [370…

```
perm = PermutationImportance(kNN, random_state=1).fit(X_test, Y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Out[370]:

Weight	Feature
0.1707 ± 0.0042	overall
0.1037 ± 0.0066	skill_moves
0.0097 ± 0.0078	potential
0.0020 ± 0.0038	weak_foot

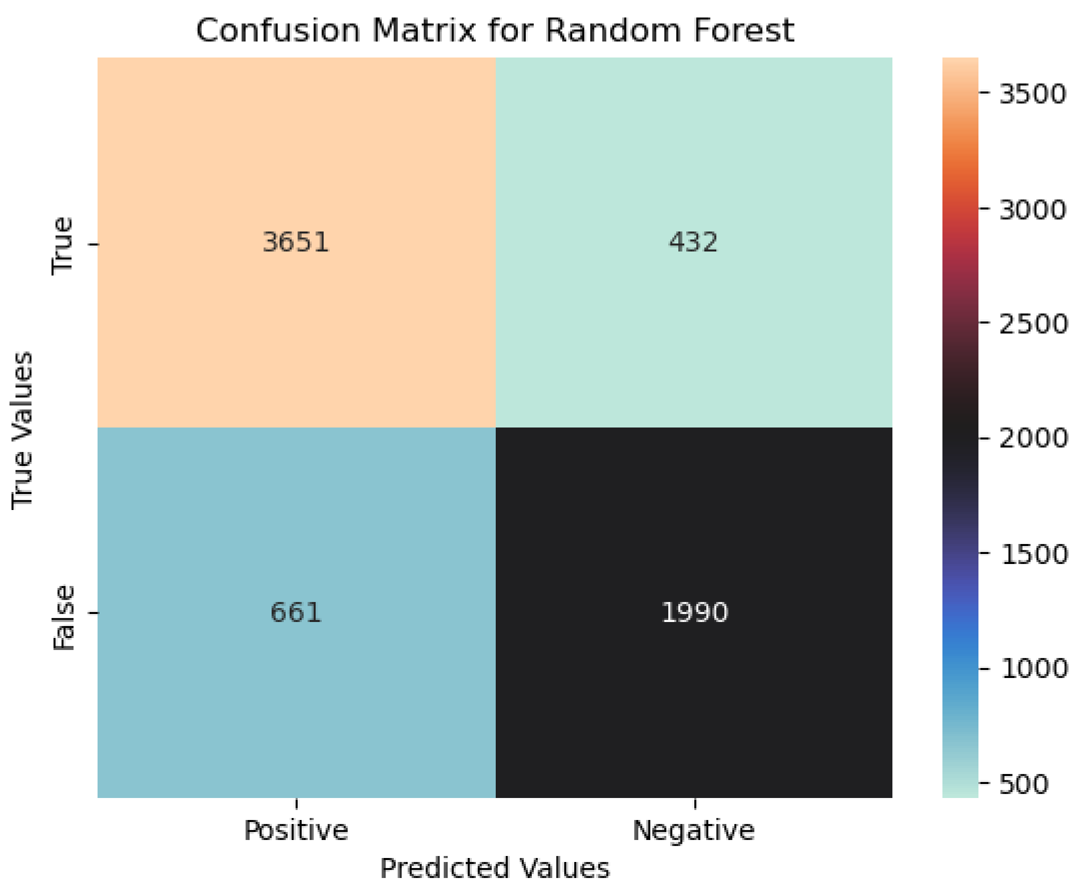
In [371…

```
# Train a random forest classifier
clf = RandomForestClassifier(n_estimators=10, random_state=45)
clf.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred_dt = clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(Y_test, Y_pred_dt)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# Get the classification report
cls_rpt = classification_report(Y_test, Y_pred)
print("\nClassification Report:\n", cls_rpt)

accuracy_scores.append(accuracy_score(Y_test, Y_pred_dt))
```

```
Accuracy: 82.30%

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      4083
           1       0.82      0.75      0.78      2651

    accuracy                           0.84      6734
   macro avg       0.83      0.82      0.83      6734
weighted avg       0.84      0.84      0.84      6734
```

In [372…

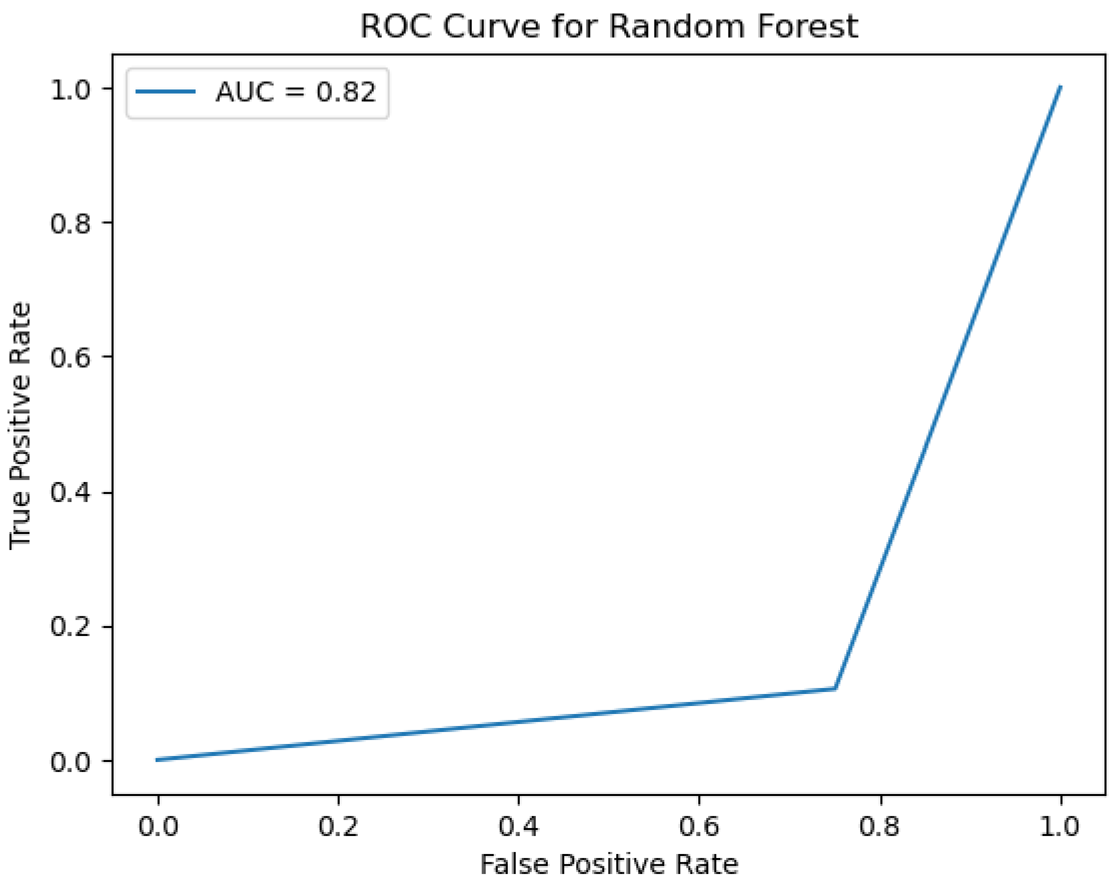
```
confusion = confusion_matrix(Y_test, Y_pred)

# Convert the confusion matrix into a data frame
confusion_df = pd.DataFrame(confusion, index=["True", "False"], columns=["Positive", '

# Plot the confusion matrix as a bar plot
sns.heatmap(confusion_df, annot=True, fmt="d", cmap="icefire")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.title("Confusion Matrix for Random Forest")
plt.show()
```



In [373…

```
y_true = [0 if y == 1.0 else 1 for y in Y_test]

# Get the ROC AUC score
auc = roc_auc_score(Y_test, Y_pred)

# Get the false positive rate and true positive rate for different thresholds
```

```
fpr, tpr, thresholds = roc_curve(y_true, Y_pred)

# Plot the ROC curve
plt.plot(fpr, tpr, label="AUC = {:.2f}".format(auc))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Random Forest")
plt.legend()
plt.show()
```



In [374…

```
perm = PermutationImportance(clf, random_state=1).fit(X_test, Y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Out[374]:

Weight	Feature
0.1530 ± 0.0088	overall
0.1185 ± 0.0044	skill_moves
0.0085 ± 0.0050	potential
0.0031 ± 0.0062	weak_foot

In [375…

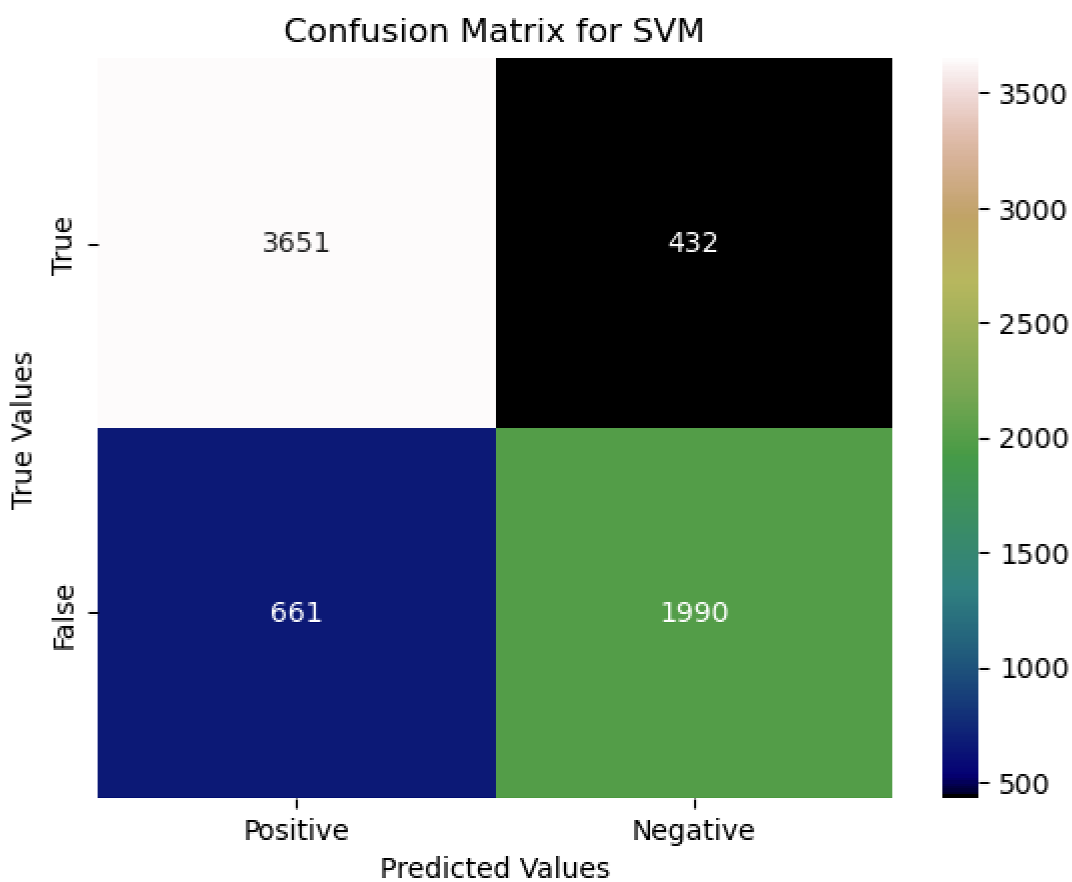
```
# Train a support vector machine classifier
SVM_train = SVC(kernel='rbf', C=1, gamma=0.1, random_state=45)
SVM_train.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred_svm = SVM_train.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(Y_test, Y_pred_svm)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# Get the classification report
```

```
cls_rpt = classification_report(Y_test, Y_pred)
print("\nClassification Report:\n", cls_rpt)

accuracy_scores.append(accuracy_score(Y_test, Y_pred_svm))
```

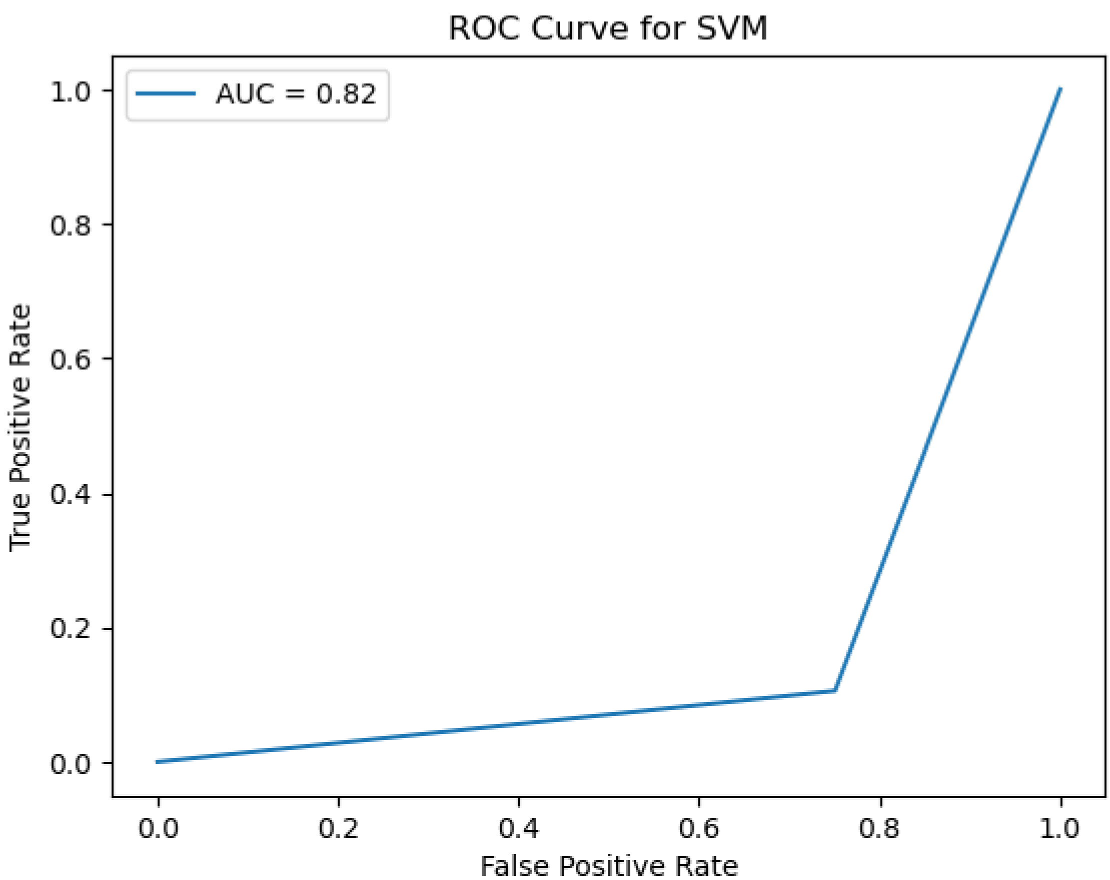
```
Accuracy: 83.77%

Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.89      0.87      4083
           1       0.82      0.75      0.78      2651

    accuracy                           0.84      6734
   macro avg       0.83      0.82      0.83      6734
weighted avg       0.84      0.84      0.84      6734
```

In [376…

```
confusion = confusion_matrix(Y_test, Y_pred)

# Convert the confusion matrix into a data frame
confusion_df = pd.DataFrame(confusion, index=["True", "False"], columns=["Positive", '

# Plot the confusion matrix as a bar plot
sns.heatmap(confusion_df, annot=True, fmt="d", cmap="gist_earth")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.title("Confusion Matrix for SVM")
plt.show()
```



In [377…
```
y_true = [0 if y == 1.0 else 1 for y in Y_test]

# Get the ROC AUC score
auc = roc_auc_score(Y_test, Y_pred)

# Get the false positive rate and true positive rate for different thresholds
fpr, tpr, thresholds = roc_curve(y_true, Y_pred)

# Plot the ROC curve
plt.plot(fpr, tpr, label="AUC = {:.2f}".format(auc))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for SVM")
plt.legend()
plt.show()
```



In [378…
```
perm = PermutationImportance(SVM_train, random_state=1).fit(X_test, Y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Out[378]:

Weight	Feature
0.1488 ± 0.0078	overall
0.1312 ± 0.0075	skill_moves
0.0775 ± 0.0071	potential
0.0004 ± 0.0007	weak_foot

In [384…
```
#Players "Weak_Foot" is making the players to low the skill rates.
#Now by using the machine learning algorithms, we are going to improve the Weak_foot c

low_weak_foot_threshold = 2
```

```
# Update weak_foot for players with a low weak foot value
Fifa_22_database.loc[Fifa_22_database['weak_foot'] < low_weak_foot_threshold, 'weak_fo
```

In [385…

```
# Step 2: Re-run the models with updated data

# Split data into features (X) and targets (Y)
X = Fifa_22_database[['overall', 'potential', 'weak_foot', 'skill_moves']]
Y = Fifa_22_database['Players Skill Set (In Total)']

# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.35, random_state
```

In [386…

```
accuracy_scores_updated=[]

#K neighbors classification
kNN = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
kNN.fit(X_train, Y_train)

Y_pred_knn1 = kNN.predict(X_test)

# Calculate permutation importance
perm = PermutationImportance(kNN, random_state=1).fit(X_test, Y_test)

# Show feature weights
eli5.show_weights(perm, feature_names=X_test.columns.tolist())

# Step 4: Provide targeted training (This step is not implemented in code as it requir

# Step 5: Evaluate the improvement
# Calculate the accuracy score after the improvement
accuracy_after_improvement = accuracy_score(Y_test, Y_pred_knn1)
print("Accuracy after weak foot improvement: {:.2f}%".format(accuracy_after_improvemer

accuracy_scores_updated.append(accuracy_score(Y_test, Y_pred_knn1))
```

```
Accuracy after weak foot improvement: 82.00%
```

In [387…

```
#Random Forest classification
clf = RandomForestClassifier(n_estimators=10, random_state=45)
clf.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred_dt1 = clf.predict(X_test)

# Calculate permutation importance
perm = PermutationImportance(kNN, random_state=1).fit(X_test, Y_test)

# Show feature weights
eli5.show_weights(perm, feature_names=X_test.columns.tolist())

# Step 4: Provide targeted training (This step is not implemented in code as it requir

# Step 5: Evaluate the improvement
# Calculate the accuracy score after the improvement
accuracy_after_improvement = accuracy_score(Y_test, Y_pred_dt1)
print("Accuracy after weak foot improvement: {:.2f}%".format(accuracy_after_improvemer
```

```
accuracy_scores_updated.append(accuracy_score(Y_test, Y_pred_dt1))
```

```
Accuracy after weak foot improvement: 82.31%
```

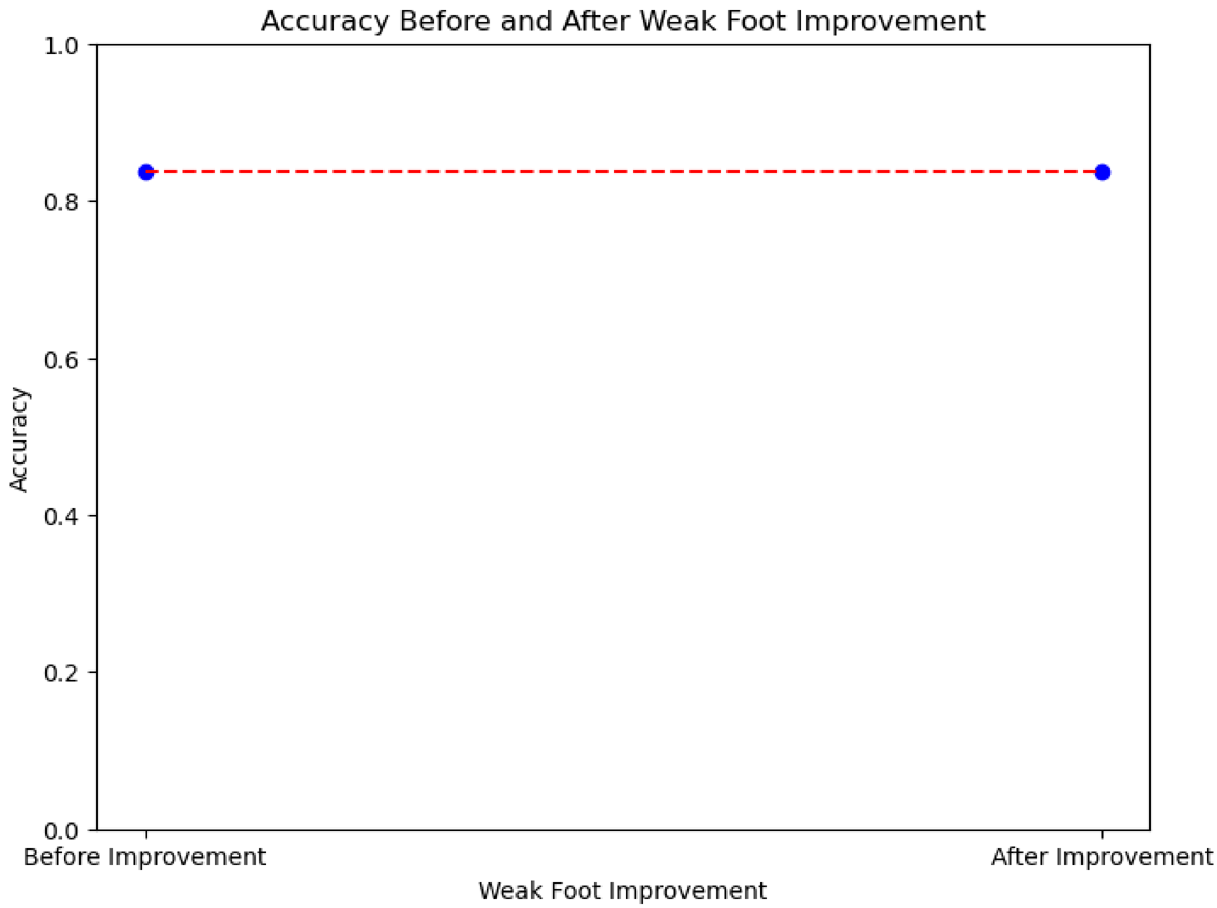
In [388…

```
#SVM classification
SVM_train = SVC(kernel='rbf', C=1, gamma=0.1, random_state=45)
SVM_train.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred_svm1 = SVM_train.predict(X_test)

# Calculate permutation importance
perm = PermutationImportance(kNN, random_state=1).fit(X_test, Y_test)

# Show feature weights
eli5.show_weights(perm, feature_names=X_test.columns.tolist())

# Step 4: Provide targeted training (This step is not implemented in code as it requir

# Step 5: Evaluate the improvement
# Calculate the accuracy score after the improvement
accuracy_after_improvement = accuracy_score(Y_test, Y_pred_svm1)
print("Accuracy after weak foot improvement: {:.2f}%".format(accuracy_after_improvemen

accuracy_scores_updated.append(accuracy_score(Y_test, Y_pred_svm1))
```

```
Accuracy after weak foot improvement: 83.77%
```

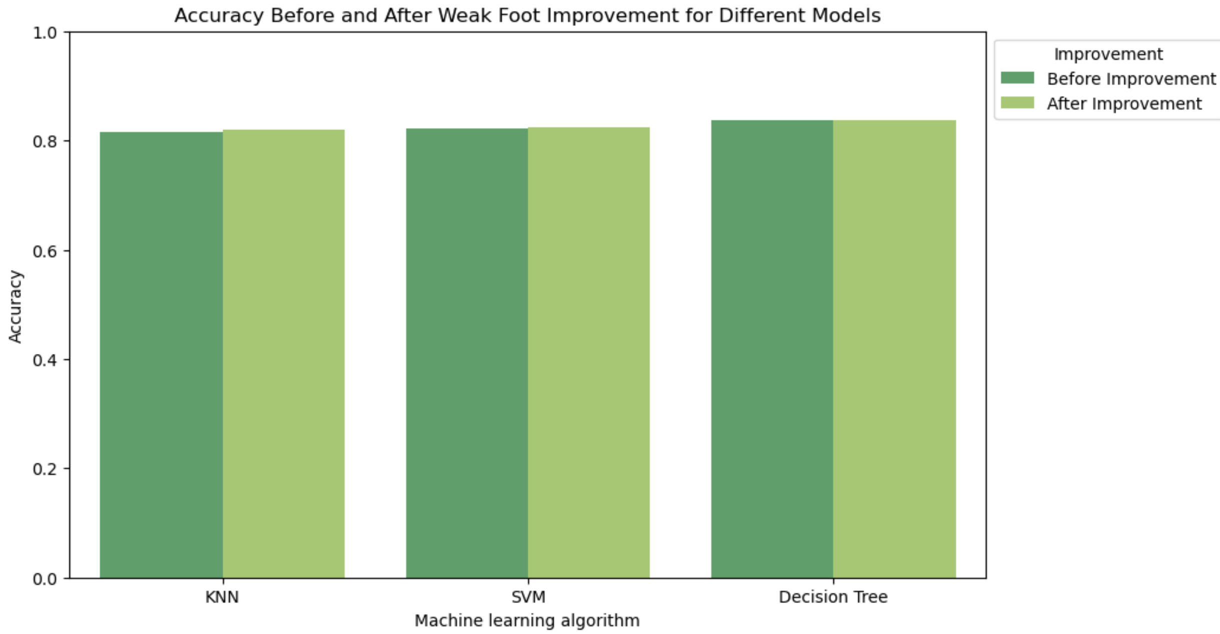
In [390…

```
#From Above SVM is the Best Machine Learning Model for Finding the Player's Performanc

#SVM classification
SVM_train = SVC(kernel='rbf', C=1, gamma=0.1, random_state=45)
SVM_train.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = SVM_train.predict(X_test)

# Plot the scatter plot
x_values = np.array([0, 1])
y_values = np.array([accuracy, accuracy_after_improvement])

plt.figure(figsize=(8, 6))
plt.scatter(x_values, y_values, color='blue')
plt.plot(x_values, y_values, color='red', linestyle='dashed')
plt.xticks(x_values, ['Before Improvement', 'After Improvement'])
plt.xlabel('Weak Foot Improvement')
plt.ylabel('Accuracy')
plt.title('Accuracy Before and After Weak Foot Improvement')
plt.ylim(0, 1)
plt.show()
```



In [398…

```
#Data Visualisation of Weak foot improvement of players for next update release

accuracy_of_all_models = pd.DataFrame({
    'Model': ['KNN', 'SVM', 'Decision Tree'],
    'Before Improvement': accuracy_scores,
    'After Improvement': accuracy_scores_updated
})


# Melt the DataFrame to make it suitable for visualization
accuracy_data_melted = pd.melt(accuracy_of_all_models, id_vars=['Model'], var_name='Im

# Plot the bar chart using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(data=accuracy_data_melted, x='Model', y='Accuracy', hue='Improvement', pal
plt.ylim(0, 1)
plt.xlabel('Machine learning algorithm')
plt.ylabel('Accuracy')
plt.title('Accuracy Before and After Weak Foot Improvement for Different Models')
plt.legend(title='Improvement', loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```


Accuracy Before and After Weak Foot Improvement for Different Models
Improvement
Before Improvement
After Improvement
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
KNN
SVM
Decision Tree
Machine learning algorithm