## SOURCE CODE

import requests from bs4 import BeautifulSoup import time import pandas as pd import seaborn as sns import matplotlib.pyplot as plt from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor from sklearn.neighbors import KNeighborsRegressor from sklearn.ensemble import AdaBoostRegressor from sklearn.ensemble import GradientBoostingRegressor from xgboost import XGBRegressor from catboost import CatBoostRegressor

years = list(range(2024, 2019, -1))

all\_matches = []

standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

visited\_urls = set()

for year in years:

print(f"Fetching data for season {year}...") try:

# Fetch the standings page data = requests.get(standings\_url) if standings\_url in visited\_urls:

print("Repeated URL detected. Stopping the loop.") break visited\_urls.add(standings\_url)

soup = BeautifulSoup(data.text, 'html.parser') standings\_table = soup.select('table.stats\_table')[0]

# Extract team URLs links = [l.get("href") for l in standings\_table.find\_all('a')] links = [l for l in links if '/squads/' in l] team\_urls = [f"https://fbref.com{l}" for l in links]



# Get the previous season URL prev\_links = soup.select("a.prev") if not prev\_links:

print("No previous season link found. Stopping the loop.") break previous\_season = prev\_links[0].get("href") standings\_url = f"https://fbref.com{previous\_season}"



# Loop through each team and get data for team\_url in team\_urls:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") print(f"Fetching data for {team\_name}...") try:

data = requests.get(team\_url) matches = pd.read\_html(data.text, match="Scores & Fixtures")[0] soup = BeautifulSoup(data.text, 'html.parser')

# Get shooting stats links = [l.get("href") for l in soup.find\_all('a')] links = [l for l in links if l and 'all\_comps/shooting/' in l] if not links: print(f"No shooting data found for {team\_name}") continue

shooting = pd.read\_html(f"https://fbref.com{links[0]}", match="Shooting")[0]



shooting.columns = shooting.columns.droplevel()

# Merge matches and shooting data team\_data = matches.merge(shooting[["Date", "Sh", "SoT", "Dist", "FK", "PK", "PKatt"]], on="Date", how='left') team\_data = team\_data[team\_data["Comp"] == "Premier League"] team\_data["Season"] = year team\_data["Team"] = team\_name all\_matches.append(team\_data)

except Exception as e: print(f"Error fetching data for {team\_name}: {e}")

ti l (10)

time.sleep(10)

except Exception as e:

print(f"Error fetching standings for season {year}: {e}") break

# Combine all data if all\_matches:

final\_df = pd.concat(all\_matches, ignore\_index=True) print("Data fetching completed successfully!") else: print("No data fetched.")

Fetching data for season 2024... Fetching data for Liverpool...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Manchester City...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Nottingham Forest...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Brighton and Hove Albion...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Chelsea...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Arsenal...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Fulham...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Aston Villa...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Tottenham Hotspur...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Brentford...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Bournemouth...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Newcastle United...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Manchester United...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for West Ham United...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Leicester City...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Everton...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Crystal Palace...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Wolverhampton Wanderers...

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an matches = pd.read\_html(data.text, match="Scores & Fixtures")[0]

Fetching data for Ipswich Town... C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\4063153100.py:43: FutureWarning: Passing literal html to 'read\_html' is deprecated an t h d d ht l(d t t t t h "S & i t ")[0]

print(team\_data)

 Date Time Comp Round Day Venue Result GF GA \

0 2020-09-14 18:00 Premier League Matchweek 1 Mon Home L 0 2

1. 2020-09-21 18:00 Premier League Matchweek 2 Mon Away L 0 1
2. 2020-09-27 12:00 Premier League Matchweek 3 Sun Home L 0 1
3. 2020-10-04 14:00 Premier League Matchweek 4 Sun Away L 1 2
4. 2020-10-18 12:00 Premier League Matchweek 5 Sun Home D 1 1
5. 2020-10-24 20:00 Premier League Matchweek 6 Sat Away L 1 2
6. 2020-10-31 12:30 Premier League Matchweek 7 Sat Home L 0 1
7. 2020-11-07 17:30 Premier League Matchweek 8 Sat Away L 1 4
8. 2020-11-22 14:00 Premier League Matchweek 9 Sun Home L 0 1
9. 2020-11-28 20:00 Premier League Matchweek 10 Sat Away L 0 1
10. 2020-12-06 14:15 Premier League Matchweek 11 Sun Home L 1 2
11. 2020-12-13 12:00 Premier League Matchweek 12 Sun Away L 0 3
12. 2020-12-17 20:00 Premier League Matchweek 13 Thu Home L 2 3
13. 2020-12-20 12:00 Premier League Matchweek 14 Sun Away D 1 1
14. 2020-12-26 20:00 Premier League Matchweek 15 Sat Home L 0 1
15. 2020-12-29 18:00 Premier League Matchweek 16 Tue Away L 0 1
16. 2021-01-02 15:00 Premier League Matchweek 17 Sat Away L 0 2
17. 2021-01-12 18:00 Premier League Matchweek 18 Tue Home W 1 0
18. 2021-01-17 14:00 Premier League Matchweek 19 Sun Home L 1 3
19. 2021-01-27 20:15 Premier League Matchweek 20 Wed Away W 2 1
20. 2021-01-30 15:00 Premier League Matchweek 21 Sat Away L 0 1
21. 2021-02-02 18:00 Premier League Matchweek 22 Tue Home W 2 1
22. 2021-02-07 19:15 Premier League Matchweek 23 Sun Home L 1 2
23. 2021-02-15 18:00 Premier League Matchweek 24 Mon Away L 0 3
24. 2021-02-20 20:00 Premier League Matchweek 25 Sat Away L 0 1
25. 2021-02-28 19:15 Premier League Matchweek 26 Sun Home L 0 2
26. 2021-03-03 18:00 Premier League Matchweek 29 Wed Home W 1 0
27. 2021-03-06 15:00 Premier League Matchweek 27 Sat Home L 0 2
28. 2021-03-14 14:00 Premier League Matchweek 28 Sun Away L 0 5
29. 2021-04-03 15:00 Premier League Matchweek 30 Sat Away L 1 2
30. 2021-04-11 19:00 Premier League Matchweek 31 Sun Home L 0 3
31. 2021-04-17 20:15 Premier League Matchweek 32 Sat Away L 0 1
32. 2021-04-24 20:00 Premier League Matchweek 33 Sat Home W 1 0
33. 2021-05-02 19:15 Premier League Matchweek 34 Sun Away L 0 4
34. 2021-05-08 15:00 Premier League Matchweek 35 Sat Home L 0 2
35. 2021-05-16 19:00 Premier League Matchweek 36 Sun Away W 1 0
36. 2021-05-19 18:00 Premier League Matchweek 37 Wed Away L 0 1
37. 2021-05-23 16:00 Premier League Matchweek 38 Sun Home W 1 0

Opponent ... Match Report Notes Sh SoT Dist FK PK PKatt \

0 Wolves ... Match Report NaN 9 1 12.9 0.0 0 0

1. Aston Villa ... Match Report NaN 3 0 20.2 0.0 0 1
2. Leeds United ... Match Report NaN 14 4 16.7 0.0 0 0
3. Arsenal ... Match Report NaN 6 2 24.7 0.0 0 0
4. Fulham ... Match Report NaN 9 5 20.1 0.0 1 1
5. Liverpool ... Match Report NaN 12 1 18.5 0.0 1 1
6. Manchester City ... Match Report NaN 3 1 18.3 0.0 0 0
7. Chelsea ... Match Report NaN 6 3 18.0 0.0 0 0
8. West Ham ... Match Report NaN 9 5 17.6 0.0 0 0
9. West Brom ... Match Report NaN 21 5 15.0 0.0 0 0
10. Leicester City ... Match Report NaN 4 1 14.9 0.0 0 0
11. Southampton ... Match Report NaN 3 0 11.2 0.0 0 0
12. Manchester Utd ... Match Report NaN 12 6 15.8 1.0 0 0
13. Brighton ... Match Report NaN 5 3 26.0 1.0 0 0
14. Everton ... Match Report NaN 10 1 12.6 0.0 0 0
15. Burnley ... Match Report NaN 7 2 16.6 0.0 0 0
16. Crystal Palace ... Match Report NaN 7 2 19.4 0.0 0 0 ttable = pd.concat(all\_matches)

ttable['GF'] = ttable['GF'].fillna(0).astype(int) ttable['GA'] = ttable['GA'].fillna(0).astype(int) ttable=ttable.reset\_index() ttable.drop('index',axis=1,inplace=True)

ttable.to\_csv('ttable.csv')

years = list(range(2024, 2019, -1)) all\_matches = [] standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

for year in years:

data = requests.get(standings\_url) soup = BeautifulSoup(data.text, 'html.parser') standings\_table = soup.select('table.stats\_table')

if not standings\_table:

print(f"No league table found for {standings\_url}") continue standings\_table = standings\_table[0] links = [l.get("href") for l in standings\_table.find\_all('a')] links = [l for l in links if '/squads/' in l] team\_urls = [f"https://fbref.com{l}" for l in links]

previous\_season = soup.select("a.prev") if previous\_season:

standings\_url = f"https://fbref.com{previous\_season[0].get('href')}" else: print(f"No previous season link found for {standings\_url}") break

for team\_url in team\_urls:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") data = requests.get(team\_url)

matches = pd.read\_html(StringIO(data.text), match="Scores & Fixtures")[0] soup = BeautifulSoup(data.text, 'html.parser')

links = [l.get("href") for l in soup.find\_all('a')] links = [l for l in links if l and 'all\_comps/passing/' in l]

if not links: print(f"No passing data link found for {team\_name} in {year}") continue data = requests.get(f"https://fbref.com{links[0]}") passing = pd.read\_html(StringIO(data.text), match="Passing")[0] passing.columns = passing.columns.droplevel()

try:

team\_data1 = matches.merge( passing[["Date", "KP", "1/3", "PPA", "CrsPA", "PrgP"]], on="Date", how="left"

) except KeyError as e:

print(f"KeyError for merging data of {team\_name} in {year}: {e}") continue team\_data1 = team\_data1[team\_data1["Comp"] == "Premier League"] team\_data1["Season"] = year team\_data1["Team"] = team\_name all\_matches.append(team\_data1) time.sleep(10)

if all\_matches:

ttable = pd.concat(all\_matches, ignore\_index=True) print(f"Collected data for {len(ttable)} matches.") else: print("No matches collected.")

 Collected data for 3800 matches.

assists = pd.concat(all\_matches) assists=assists.reset\_index() assists.drop('index',axis=1,inplace=True)

assists.to\_csv('assists.csv')

years = list(range(2024, 2019, -1)) all\_matches = [] standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

for year in years:

data = requests.get(standings\_url) soup = BeautifulSoup(data.text, 'html.parser') standings\_table = soup.select('table.stats\_table')

if not standings\_table:

print(f"No league table found for {standings\_url}") continue standings\_table = standings\_table[0] links = [l.get("href") for l in standings\_table.find\_all('a')] links = [l for l in links if '/squads/' in l] team\_urls = [f"https://fbref.com{l}" for l in links]

previous\_season = soup.select("a.prev") if previous\_season:

standings\_url = f"https://fbref.com{previous\_season[0].get('href')}" else: print(f"No previous season link found for {standings\_url}") break

for team\_url in team\_urls:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") data = requests.get(team\_url)

matches = pd.read\_html(StringIO(data.text), match="Scores & Fixtures")[0] soup = BeautifulSoup(data.text, 'html.parser')

links = [l.get("href") for l in soup.find\_all('a')] links = [l for l in links if l and 'all\_comps/possession/' in l]

if not links: print(f"No possession data link found for {team\_name} in {year}") continue data = requests.get(f"https://fbref.com{links[0]}") possession = pd.read\_html(StringIO(data.text), match="Possession")[0] possession.columns = possession.columns.droplevel()

required\_columns = ["Att 3rd", "Att Pen", "PrgC"] available\_columns = [col for col in required\_columns if col in possession.columns]

if not available\_columns:

print(f"Required possession columns not found for {team\_name} in {year}") continue

try: team\_data2 = matches.merge(possession[["Date"] + available\_columns], on="Date") except ValueError as e:

print(f"Merge error for {team\_name} in {year}: {e}") continue

team\_data2 = team\_data2[team\_data2["Comp"] == "Premier League"] team\_data2["Season"] = year team\_data2["Team"] = team\_name all\_matches.append(team\_data2) time.sleep(10)

if all\_matches:

ttable = pd.concat(all\_matches, ignore\_index=True) print(f"Collected data for {len(ttable)} matches.") else: print("No matches collected.")

 Collected data for 3252 matches.

posses = pd.concat(all\_matches) posses=posses.reset\_index() posses.drop('index',axis=1,inplace=True)

posses.to\_csv('possession.csv', index=False)

years = list(range(2024, 2019, -1)) all\_matches = [] standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

for year in years:

data = requests.get(standings\_url) soup = BeautifulSoup(data.text, 'html.parser') standings\_table = soup.select('table.stats\_table')

if not standings\_table:

print(f"No league table found for {standings\_url}") continue

standings\_table = standings\_table[0] links = [l.get("href") for l in standings\_table.find\_all('a')] links = [l for l in links if '/squads/' in l] team\_urls = [f"https://fbref.com{l}" for l in links]

previous\_season = soup.select("a.prev") if previous\_season:

standings\_url = f"https://fbref.com{previous\_season[0].get('href')}" else: print(f"No previous season link found for {standings\_url}") break

for team\_url in team\_urls:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") data = requests.get(team\_url)

matches = pd.read\_html(StringIO(data.text), match="Scores & Fixtures")[0] soup = BeautifulSoup(data.text, 'html.parser') links = [l.get("href") for l in soup.find\_all('a')] links = [l for l in links if l and 'all\_comps/gca/' in l]

if not links: print(f"No goal and shot creation data link found for {team\_name} in {year}") continue data = requests.get(f"https://fbref.com{links[0]}") gfc = pd.read\_html(StringIO(data.text), match="Goal and Shot Creation")[0] gfc.columns = gfc.columns.droplevel()

if "Date" not in gfc.columns or "SCA" not in gfc.columns:

print(f"Required columns not found for {team\_name} in {year}") continue

try: team\_data3 = matches.merge(gfc[["Date", "SCA"]], on="Date") except ValueError as e:

print(f"Merge error for {team\_name} in {year}: {e}") continue

team\_data3 = team\_data3[team\_data3["Comp"] == "Premier League"] team\_data3["Season"] = year team\_data3["Team"] = team\_name all\_matches.append(team\_data3) time.sleep(10)

if all\_matches:

ttable = pd.concat(all\_matches, ignore\_index=True) print(f"Collected data for {len(ttable)} matches.") else: print("No matches collected.")

 Collected data for 3252 matches.

creation = pd.concat(all\_matches) creation=creation.reset\_index() creation.drop('index',axis=1,inplace=True)

creation.to\_csv('creation.csv')

years = list(range(2024, 2019, -1)) all\_matches = [] standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"

for year in years:

data = requests.get(standings\_url) soup = BeautifulSoup(data.text, 'html.parser')

standings\_table = soup.select('table.stats\_table') if not standings\_table:

print(f"No table found for {standings\_url}") continue

standings\_table = standings\_table[0] links = [l.get("href") for l in standings\_table.find\_all('a')] links = [l for l in links if '/squads/' in l] team\_urls = [f"https://fbref.com{l}" for l in links]

previous\_season = soup.select("a.prev") if not previous\_season:

print(f"No previous season link found for {standings\_url}") break standings\_url = f"https://fbref.com{previous\_season[0].get('href')}"

for team\_url in team\_urls:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") data = requests.get(team\_url)

matches = pd.read\_html(StringIO(data.text), match="Scores & Fixtures")[0] soup = BeautifulSoup(data.text, 'html.parser') links = [l.get("href") for l in soup.find\_all('a')]

links = [l for l in links if l and 'all\_comps/passing\_types/' in l]

if not links: print(f"No passing type link found for {team\_name} in {year}") continue data = requests.get(f"https://fbref.com{links[0]}") passing\_type = pd.read\_html(StringIO(data.text), match="Pass Types")[0] passing\_type.columns = passing\_type.columns.droplevel()

if "Date" not in passing\_type.columns or "TB" not in passing\_type.columns:

print(f"Required columns not found for {team\_name} in {year}") continue try: team\_data5 = matches.merge(passing\_type[["Date", "TB"]], on="Date") except ValueError as e:

print(f"Merge error for {team\_name} in {year}: {e}") continue team\_data5 = team\_data5[team\_data5["Comp"] == "Premier League"] team\_data5["Season"] = year team\_data5["Team"] = team\_name all\_matches.append(team\_data5) time.sleep(10)

if all\_matches:

full\_table = pd.concat(all\_matches, ignore\_index=True) print(f"Collected data for {len(full\_table)} matches.") else: print("No matches collected.")

 Collected data for 3252 matches.

back1=pd.concat(all\_matches) back1=back1.reset\_index() back1.drop('index',axis=1,inplace=True)

back1.to\_csv('back1.csv', index=False)

ttable = pd.read\_csv('ttable.csv') assists = pd.read\_csv('assists.csv') posses = pd.read\_csv('possession.csv') creation = pd.read\_csv('creation.csv') back1 = pd.read\_csv('back1.csv')

def season(x):

mainseas = ttable[ttable['Season'] == x] ass = assists[assists['Season'] == x] poss = posses[posses['Season'] == x] cre = creation[creation['Season'] == x] back = back1[back1['Season'] == x]

feat = ["Sh", "SoT", "Dist", "FK", "PK", "PKatt", 'GF', 'GA', 'xG', 'xGA', 'Poss', "KP", "1/3", "PPA", "CrsPA", "Prog", "Att 3rd", "Att Pen", "Prog", 'SCA', 'TB'] df = pd.DataFrame() df['Team'] = mainseas['Team'].unique()

# Initialize columns with zeros for i in feat: df[i] = 0.0

main = ["Sh", "SoT", "Dist", "FK", "PK", "PKatt", 'GF', 'GA', 'xG', 'xGA', 'Poss'] asss = ["KP", "1/3", "PPA", "CrsPA", "PrgP"] posss = ["Att 3rd", "Att Pen", "PrgC"] gfc = ['SCA'] tb = ['TB'] feat1 = [main, asss, poss, gfc, tb] tables = [mainseas, ass, poss, cre, back]

# Assign main features for j in main: for i in range(len(df)): df.loc[i, j] = mainseas[mainseas['Team'] == df.loc[i, 'Team']][j].mean()

# Assign assist features for j in asss: for i in range(len(df)): df.loc[i, j] = ass[ass['Team'] == df.loc[i, 'Team']][j].mean()

# Assign possession features for j in posss: for i in range(len(df)): df.loc[i, j] = poss[poss['Team'] == df.loc[i, 'Team']][j].mean()

# Special feature assignments for i in range(len(df)):

df.loc[i, 'SCA'] = cre[cre['Team'] == df.loc[i, 'Team']]['SCA'].mean() for i in range(len(df)): df.loc[i, 'TB'] = back[back['Team'] == df.loc[i, 'Team']]['TB'].mean()

# Add Season, Points, and Points per match columns df['Season'] = x df['Points'] = 0

df['Points per match'] = 0.0

for i in range(len(df)):

win\_count = mainseas[mainseas['Team'] == df.loc[i, 'Team']]['Result'].value\_counts().get('W', 0) draw\_count = mainseas[mainseas['Team'] == df.loc[i, 'Team']]['Result'].value\_counts().get('D', 0) df.loc[i, 'Points'] = win\_count \* 3 + draw\_count

for i in range(len(df)):

df.loc[i, 'Points per match'] = df.loc[i, 'Points'] / len(mainseas[mainseas['Team'] == df.loc[i, 'Team']])

return df

# Creating DF for different seasons seas19 = season(2019) seas20 = season(2020) seas21 = season(2021) seas22 = season(2022) seas23 = season(2023) seas24 = season(2024)

# Concatenating all the seasons total = pd.concat([seas19, seas20, seas21, seas22, seas23, seas24])

total.to\_csv('Final\_Data.csv')

print(total.columns)

 Index(['Team', 'Sh', 'SoT', 'Dist', 'FK', 'PK', 'PKatt', 'GF', 'GA', 'xG',

'xGA', 'Poss', 'KP', '1/3', 'PPA', 'CrsPA', 'Prog', 'Att 3rd',

'Att Pen', 'SCA', 'TB', 'Season', 'Points', 'Points per match', 'PrgP',

'PrgC'], dtype='object')

print(total.head())

 Team Sh SoT Dist FK PK \

1. Manchester City 15.526316 5.552632 17.257895 0.552632 0.131579
2. Manchester United 13.552632 5.157895 18.118421 0.552632 0.263158
3. Liverpool 15.842105 5.368421 17.342105 0.578947 0.157895
4. Chelsea 14.368421 5.078947 17.371053 0.421053 0.210526
5. Leicester City 12.447368 4.473684 19.039474 0.526316 0.263158

PKatt GF GA xG ... Prog Att 3rd Att Pen \

1. 0.236842 2.184211 0.842105 1.797368 ... 0.0 223.789474 31.473684
2. 0.289474 1.921053 1.157895 1.576316 ... 0.0 192.710526 24.236842
3. 0.157895 1.789474 1.105263 1.781579 ... 0.0 216.894737 31.868421
4. 0.263158 1.526316 0.947368 1.642105 ... 0.0 192.315789 26.763158
5. 0.315789 1.789474 1.315789 1.476316 ... 0.0 149.315789 20.315789

SCA TB Season Points Points per match PrgP PrgC

1. 28.078947 1.000000 2020 86 2.263158 53.973684 28.842105
2. 25.078947 0.894737 2020 74 1.947368 48.210526 23.078947
3. 28.105263 1.157895 2020 69 1.815789 54.131579 23.315789
4. 26.789474 0.763158 2020 67 1.763158 47.315789 23.552632
5. 22.710526 0.921053 2020 66 1.736842 38.526316 17.526316

[5 rows x 26 columns]

def fetch\_url(url, sleep\_time=10):

"""Fetch the URL with retries and delay.""" for \_ in range(3):

response = requests.get(url) if response.status\_code == 200:

return response print(f"Failed request with status {response.status\_code}. Retrying...") time.sleep(sleep\_time) return None

def scrape\_premier\_league\_data():

standings\_url = "https://fbref.com/en/comps/9/Premier-League-Stats"



years = list(range(2024, 2018, -1)) all\_matches = []

for year in years:

print(f"Fetching data for season {year}...") response = fetch\_url(standings\_url) if not response:

print(f"Skipping {year} due to repeated request failures.") continue

time.sleep(10) soup = BeautifulSoup(response.text, "html.parser") standings\_table = soup.select('table.stats\_table') if not standings\_table:

print(f"No standings table found for {year}.") continue

links = [l.get("href") for l in standings\_table[0].find\_all('a') if l.get("href") and '/squads/' in l.get("href")] team\_urls = [f"https://fbref.com{l}" for l in links]



previous\_season = soup.select("a.prev") if previous\_season:

standings\_url = f"https://fbref.com{previous\_season[0].get('href')}"



else:

print(f"No previous season link found for {year}.") break

for team\_url in team\_urls: try:

team\_name = team\_url.split("/")[-1].replace("-Stats", "").replace("-", " ") team\_response = fetch\_url(team\_url) if not team\_response:

print(f"Skipping {team name} due to request failure ")

print(f Skipping {team\_name} due to request failure. ) continue time.sleep(10)

matches = pd.read\_html(team\_response.text, match="Scores & Fixtures")[0]

soup\_team = BeautifulSoup(team\_response.text, "html.parser")

shooting\_links = [l.get("href") for l in soup\_team.find\_all('a') if l and 'all\_comps/shooting/' in l] if shooting\_links:

shooting\_response = fetch\_url(f"https://fbref.com{shooting\_links[0]}")



if shooting\_response:

shooting = pd.read\_html(shooting\_response.text, match="Shooting")[0] shooting.columns = shooting.columns.droplevel() matches = matches.merge(shooting[['Date', 'Sh', 'SoT', 'Dist', 'FK', 'PK', 'PKatt']], on="Date", how="left")

passing\_links = [l.get("href") for l in soup\_team.find\_all('a') if l and 'all\_comps/passing/' in l] if passing\_links:

passing\_response = fetch\_url(f"https://fbref.com{passing\_links[0]}")



if passing\_response:

passing = pd.read\_html(passing\_response.text, match="Passing")[0] passing.columns = passing.columns.droplevel() matches = matches.merge(passing[['Date', 'KP', '1/3', 'PPA', 'CrsPA', 'Prgp']], on="Date", how="left")

matches["Season"] = year matches["Team"] = team\_name all\_matches.append(matches)

except Exception as e:

print(f"Error processing team {team\_name} in {year}: {e}") continue

final\_df = pd.concat(all\_matches, ignore\_index=True) final\_df.columns = [c.lower() for c in final\_df.columns] return final\_df

final\_df.to\_csv('fix.csv')

ttotal = pd.read\_csv(r'FFinal\_Data.csv')

filtered\_total = ttotal.drop(columns=['Prog', 'sno'])

filtered\_total\_numeric = filtered\_total.select\_dtypes(include=['number'])

# Adjust size dynamically based on the number of numeric columns num\_cols = filtered\_total\_numeric.shape[1] figsize = (24, 24 \* 0.6)

# Create the heatmap plt.figure(figsize=figsize) sns.heatmap(filtered\_total\_numeric.corr(), annot=True, cmap='coolwarm') plt.show()



plt.savefig(r'heatmap.png', bbox\_inches='tight')

from sklearn.model\_selection import train\_test\_split X=ttotal.drop(['Points','Season','Team','PpG'],axis=1) y=ttotal['Points']

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

seas24\_test=seas24

print(seas24.head())

 Team Sh SoT Dist FK \

1. Liverpool 13.909091 5.727273 16.436364 0.272727
2. Manchester City 19.181818 6.545455 17.118182 0.545455
3. Nottingham Forest 14.700000 5.300000 18.480000 0.500000
4. Brighton and Hove Albion 14.454545 4.636364 16.800000 0.909091
5. Chelsea 13.600000 5.100000 16.330000 0.400000

PK PKatt GF GA xG ... Prog Att 3rd \

1. 0.181818 0.181818 0.552632 0.157895 1.818182 ... 0.0 162.090909
2. 0.090909 0.090909 0.578947 0.342105 2.000000 ... 0.0 329.272727
3. 0.100000 0.100000 0.368421 0.184211 1.360000 ... 0.0 150.400000
4. 0.000000 0.000000 0.500000 0.394737 1.481818 ... 0.0 159.272727
5. 0.100000 0.100000 0.526316 0.315789 1.820000 ... 0.0 154.800000

Att Pen SCA TB PrgP PrgC Season Points \

1. 29.727273 25.636364 1.636364 42.181818 20.181818 2024 28
2. 46.454545 35.545455 2.363636 58.181818 37.454545 2024 23
3. 25.200000 25.300000 1.300000 29.800000 19.100000 2024 19
4. 26.545455 24.909091 2.363636 44.818182 20.636364 2024 19
5. 26.300000 24.600000 3.000000 37.300000 22.100000 2024 18

Points per match

1. 0.736842
2. 0.605263
3. 0.500000
4. 0.500000
5. 0.473684

[5 rows x 26 columns]

knn = KNeighborsRegressor(n\_neighbors=12) dt = DecisionTreeRegressor(max\_depth = 5) rf = RandomForestRegressor(n\_estimators=100, max\_features= 7) ada = AdaBoostRegressor( n\_estimators=150, learning\_rate =.08) gbr = GradientBoostingRegressor(max\_depth=7, n\_estimators=500, learning\_rate =.05) xgb = XGBRegressor(max\_depth = 7, n\_estimators=500, learning\_rate =.05) cb = CatBoostRegressor(learning\_rate =.07, max\_depth =7, verbose=0)

regressors = [('K Nearest Neighbours', knn),('Decision Tree', dt), ('Random Forest', rf),

('AdaBoost', ada),('Gradient Boosting Regressor', gbr),('XGBRegressor', xgb),('CatBoostRegressor', cb)]

from sklearn.metrics import r2\_score

for regressor\_name, regressor in regressors:

regressor.fit(X\_train, y\_train) y\_pred = regressor.predict(X\_test) accuracy = round(r2\_score(y\_test, y\_pred), 3) \* 100

print('{:s} : {:.0f} %'.format(regressor\_name, accuracy))

random\_color = [random.random() for \_ in range(3)] plt.rcParams["figure.figsize"] = (20, 8)

plt.bar(regressor\_name, accuracy, color=random\_color)

plt.text(regressor\_name, accuracy + 1, f'{accuracy:.0f}%', ha='center', fontsize=12)

plt.title('Model Accuracy Comparison') plt.xlabel('Model') plt.ylabel('Accuracy (%)') plt.show()

 K Nearest Neighbours : 73 %

Decision Tree : 87 %

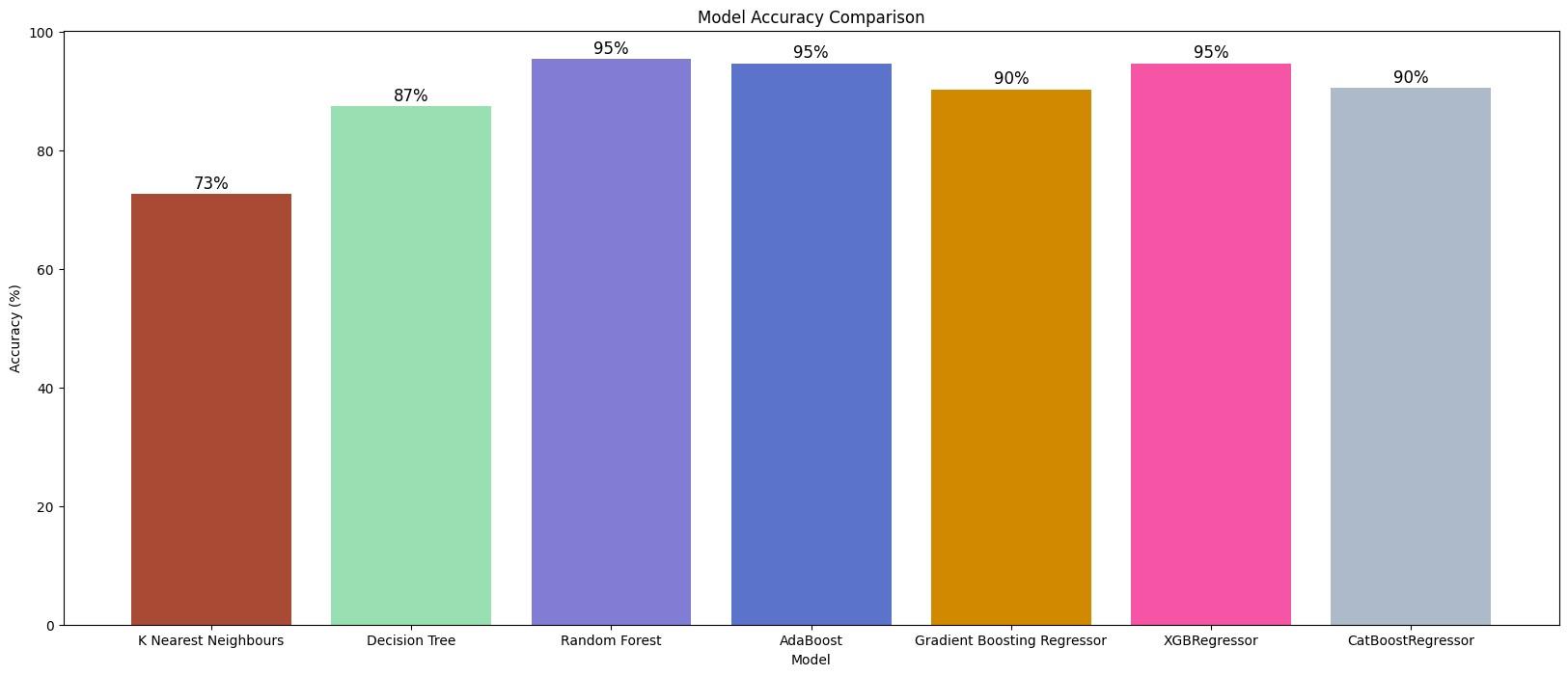
Random Forest : 95 %

AdaBoost : 95 %

Gradient Boosting Regressor : 90 %

XGBRegressor : 95 %

CatBoostRegressor : 90 %



plt.savefig(r'reg\_accu.png', bbox\_inches='tight')

seas24\_copy=seas24\_test

print(seas24\_copy.head())

 Team Sh SoT Dist FK \

1. Liverpool 13.909091 5.727273 16.436364 0.272727
2. Manchester City 19.181818 6.545455 17.118182 0.545455
3. Nottingham Forest 14.700000 5.300000 18.480000 0.500000
4. Brighton and Hove Albion 14.454545 4.636364 16.800000 0.909091
5. Chelsea 13.600000 5.100000 16.330000 0.400000

PK PKatt GF GA xG ... Prog Att 3rd \

1. 0.181818 0.181818 0.552632 0.157895 1.818182 ... 0.0 162.090909
2. 0.090909 0.090909 0.578947 0.342105 2.000000 ... 0.0 329.272727
3. 0.100000 0.100000 0.368421 0.184211 1.360000 ... 0.0 150.400000
4. 0.000000 0.000000 0.500000 0.394737 1.481818 ... 0.0 159.272727
5. 0.100000 0.100000 0.526316 0.315789 1.820000 ... 0.0 154.800000

Att Pen SCA TB PrgP PrgC Season Points \

1. 29.727273 25.636364 1.636364 42.181818 20.181818 2024 28
2. 46.454545 35.545455 2.363636 58.181818 37.454545 2024 23
3. 25.200000 25.300000 1.300000 29.800000 19.100000 2024 19
4. 26.545455 24.909091 2.363636 44.818182 20.636364 2024 19
5. 26.300000 24.600000 3.000000 37.300000 22.100000 2024 18

Points per match

1. 0.736842
2. 0.605263
3. 0.500000
4. 0.500000
5. 0.473684

[5 rows x 26 columns]

seas24\_copy.drop(['Sh', 'SoT', 'Dist', 'FK', 'PK', 'PKatt', 'GF', 'GA', 'xG',

'xGA', 'Poss', 'KP', '1/3', 'PPA', 'CrsPA' , 'Att 3rd',

'Att Pen', 'SCA', 'TB', 'Prog'],axis=1,inplace=True) print(seas24\_copy.head())

 Team PrgP PrgC Season Points \

1. Liverpool 42.181818 20.181818 2024 28
2. Manchester City 58.181818 37.454545 2024 23
3. Nottingham Forest 29.800000 19.100000 2024 19
4. Brighton and Hove Albion 44.818182 20.636364 2024 19
5. Chelsea 37.300000 22.100000 2024 18

Points per match

1. 0.736842
2. 0.605263
3. 0.500000
4. 0.500000
5. 0.473684

fix=pd.read\_csv(r'fix.csv')

seas24\_copy['Predicted Points XGB']=0.0 for i in range(len(seas24\_copy['Team'].unique())): seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Season'

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pand You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pand You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pand You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points XGB'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Se C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\3006510771.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pand You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)seas24\_copy['Predicted Points AdaBoost']=0.0 for i in range(len(seas24\_copy['Team'].unique())): seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fix['Sea

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points AdaBoost'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(fi C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\14045677.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pandas You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points Random Forest']=0.0 for i in range(len(seas24\_copy['Team'].unique())): seas24\_copy['Predicted Points Random Forest'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i])&(f

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\2352499541.py:3: FutureWarning: ChainedAssignmentError: behaviour will change in pand You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will bec

A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row\_indexer] = value Use `df.loc[row\_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the origi See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

seas24\_copy['Predicted Points Random Forest'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i]

C:\Users\admin\AppData\Local\Temp\ipykernel\_5840\2352499541.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) seas24\_copy['Predicted Points Random Forest'][i]=seas24\_copy['Points per match'][i]\*(38-len(fix[(fix['Team']==seas24\_copy['Team'][i]



