A
Report
on

Football Fever

submitted in fulfillment of the requirements
Software Enginering Mini Project Level II
of

TY COMP

in

Computer Engineering

by

Vedant Deshmukh 112003034
Shubham Jagtap 112003051
Jay Chitale 112003054

Under the guidance of

Tanuja Pattenshetti
Professor
Department of Computer Engineering

Department of Computer Engineering,
COEP Technological University (COEP Tech)
(A Unitary Public University of Govt. of Maharashtra)
Shivajinagar, Pune-411005,Maharashtra, INDIA

May, 2023

1 Problem Statement

In a world where competition ignites passion and boundaries vanish beneath the collective roar of cheering crowds, the immersion of sports transcends mere entertainment, becoming a visceral journey that unites hearts, tests limits, and weaves human endeavor. With the large scale immersion of sports in the lives of people, fans crave a way to understand the nitty-gritties behind the statistics/numbers in sports and make sense of these figures. To contribute to this craving, we, connoisseurs so football, have developed an integrated system to cater to this hunger.

In broad terms, the general problem we are trying so tackle is to create a holistic platform and make it convenient to make sense of and interpret data. Narrowing down, we aim to make football analytics module equipped with the most updated data to be gathered and presented to user. In the presenting the data we intend to prioritize highlighting the crucial, technical data in a easy, pictorial way. We also plan on making an utilising machine learning concepts to build win-or-lose estimator for the English Premiere League bolstered/trained with sufficient and relevant data.

2 Objectives

As of today, our vision isn't exactly paralleled by any other proprietary software. After doing research online, we found that ESPN has some software that somewhat encompasses our idea, but, it largely leaves the user with a ton of numeric data which isn't really easily interpreted. A lot of the new channels also offer insights into player analytics but all of them present numeric data, something that isn't convenient/fun to digest let alone interactive. Furthermore, we observed that other than polls on social media websites, no other platforms offer an outcome/ win-or-lose estimator for a particular team, something we intend on doing efficiently and accurately for the famous English Premier League.

Given this market gap we believe that there is a lot of potential to expand on the easy and interactive consumption of data. Some steps we intend to take on this front:

- Unique visualizations throught radar plots
    - Ability to compare players on important, technical aspects

  - Ability to comapre the performance of teams in the EPL based on important technical factors.

- A one of a kind win-or-lose predictor: Accepts a combination of technical and non-technical parameters and predicts the outcome (i.e. the winning team) with high accuracy

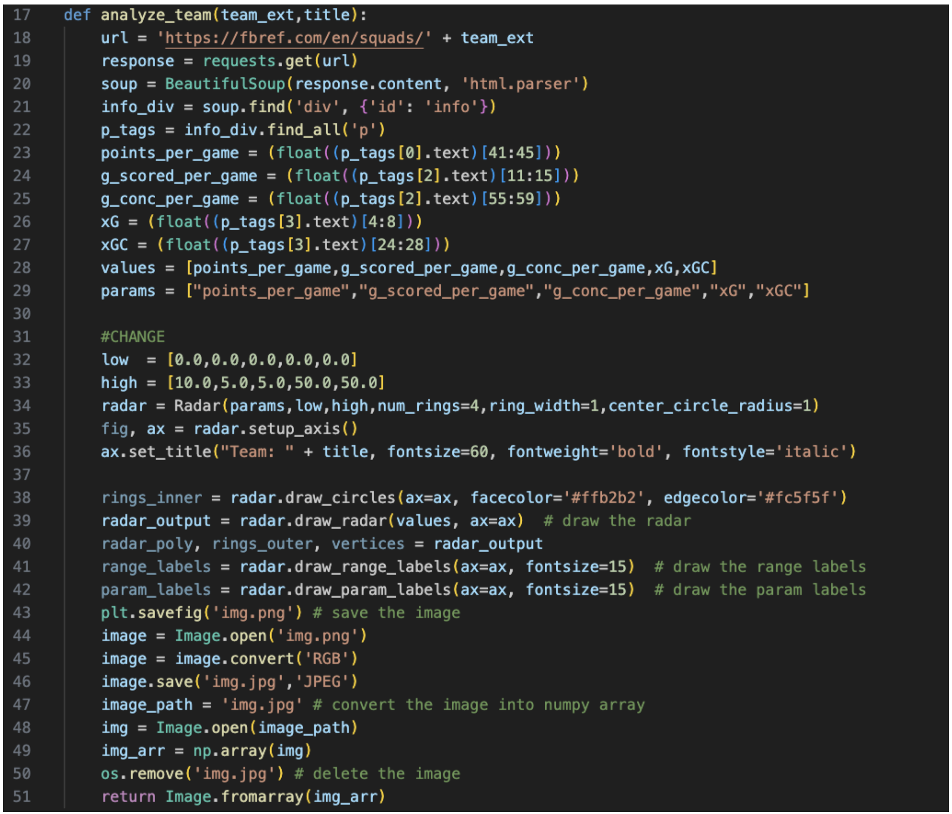
We think that achieving these objectives will give us a head-start over all other existing apps and web-pages and make us truly unique and more digestable, interpretable in nature. Our main target market is specifically millennials of the 21st century characterized by their low attention spans and a craving to learn pictorially without too much unnecessary noise.

3 Functionalities

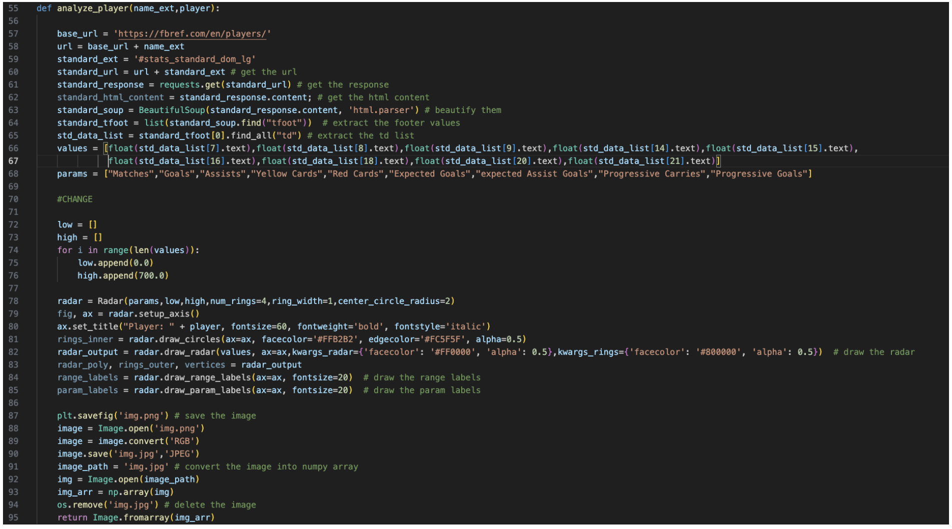
- Predict English Premier League team Win or Lose by analysing technical and non-technical parameters.
- Analyse individual teams and players on a series of the most important features in a visually alluring way
- Analyse multiple teams and players on a variety of attributes in efforts to

4 Coding Screenshots

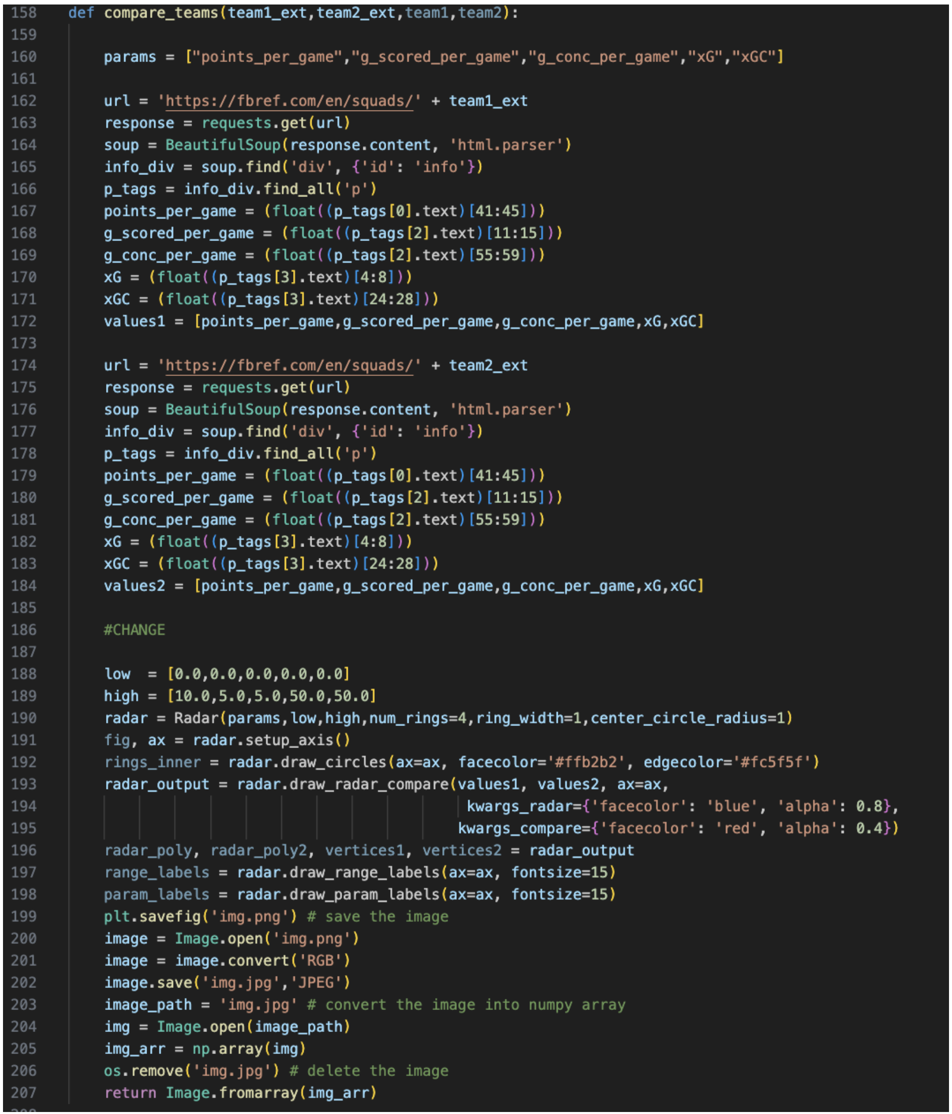
Important Front-end function:



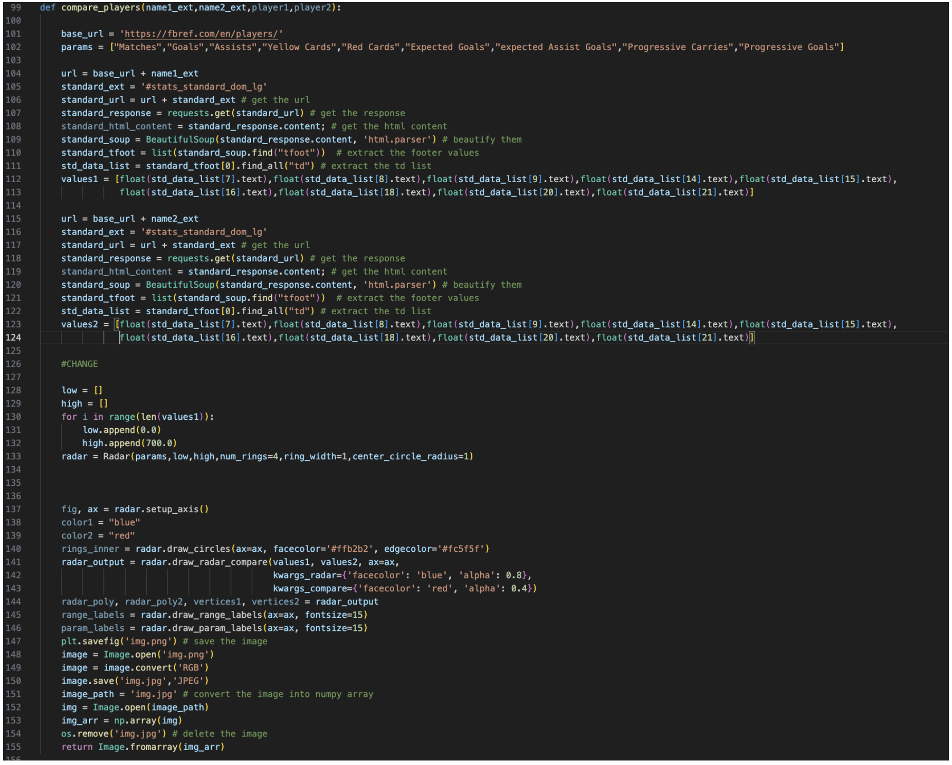
```python
def analyze_team(team_ext,title):
    url = 'https://fbref.com/en/squads/' + team_ext
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    info_div = soup.find('div', {'id': 'info'})
    p_tags = info_div.find_all('p')
    points_per_game = (float((p_tags[0].text)[41:45]))
    g_scored_per_game = (float((p_tags[2].text)[11:15]))
    g_conc_per_game = (float((p_tags[2].text)[55:59]))
    xG = (float((p_tags[3].text)[4:8]))
    xGC = (float((p_tags[3].text)[24:28]))
    values = [points_per_game,g_scored_per_game,g_conc_per_game,xG,xGC]
    params = ["points_per_game","g_scored_per_game","g_conc_per_game","xG","xGC"]

    #CHANGE
    low  = [0.0,0.0,0.0,0.0,0.0]
    high = [10.0,5.0,5.0,50.0,50.0]
    radar = Radar(params,low,high,num_rings=4,ring_width=1,center_circle_radius=1)
    fig, ax = radar.setup_axis()
    ax.set_title("Team: " + title, fontsize=60, fontweight='bold', fontstyle='italic')

    rings_inner = radar.draw_circles(ax=ax, facecolor='#ffb2b2', edgecolor='#fc5f5f')
    radar_output = radar.draw_radar(values, ax=ax)  # draw the radar
    radar_poly, rings_outer, vertices = radar_output
    range_labels = radar.draw_range_labels(ax=ax, fontsize=15)  # draw the range labels
    param_labels = radar.draw_param_labels(ax=ax, fontsize=15)  # draw the param labels
    plt.savefig('img.png') # save the image
    image = Image.open('img.png')
    image = image.convert('RGB')
    image.save('img.jpg','JPEG')
    image_path = 'img.jpg' # convert the image into numpy array
    img = Image.open(image_path)
    img_arr = np.array(img)
    os.remove('img.jpg') # delete the image
    return Image.fromarray(img_arr)
```



```python
def analyze_player(name_ext,player):

    base_url = 'https://fbref.com/en/players/'
    url = base_url + name_ext
    standard_ext = '#stats_standard_dom_lg'
    standard_url = url + standard_ext # get the url
    standard_response = requests.get(standard_url) # get the response
    standard_html_content = standard_response.content; # get the html content
    standard_soup = BeautifulSoup(standard_response.content, 'html.parser') # beautify them
    standard_tfoot = list(standard_soup.find("tfoot"))  # extract the footer values
    std_data_list = standard_tfoot[0].find_all("td") # extract the td list
    values = [float(std_data_list[7].text),float(std_data_list[8].text),float(std_data_list[9].text),float(std_data_list[14].text),float(std_data_list[15].text),
              float(std_data_list[16].text),float(std_data_list[18].text),float(std_data_list[20].text),float(std_data_list[21].text)]
    params = ["Matches","Goals","Assists","Yellow Cards","Red Cards","Expected Goals","expected Assist Goals","Progressive Carries","Progressive Goals"]

    #CHANGE

    low = []
    high = []
    for i in range(len(values)):
        low.append(0.0)
        high.append(700.0)

    radar = Radar(params,low,high,num_rings=4,ring_width=1,center_circle_radius=2)
    fig, ax = radar.setup_axis()
    ax.set_title("Player: " + player, fontsize=60, fontweight='bold', fontstyle='italic')
    rings_inner = radar.draw_circles(ax=ax, facecolor='#FFB2B2', edgecolor='#FC5F5F', alpha=0.5)
    radar_output = radar.draw_radar(values, ax=ax,kwargs_radar={'facecolor': '#FF0000', 'alpha': 0.5},kwargs_rings={'facecolor': '#800000', 'alpha': 0.5})  # draw the radar
    radar_poly, rings_outer, vertices = radar_output
    range_labels = radar.draw_range_labels(ax=ax, fontsize=20)  # draw the range labels
    param_labels = radar.draw_param_labels(ax=ax, fontsize=20)  # draw the param labels

    plt.savefig('img.png') # save the image
    image = Image.open('img.png')
    image = image.convert('RGB')
    image.save('img.jpg','JPEG')
    image_path = 'img.jpg' # convert the image into numpy array
    img = Image.open(image_path)
    img_arr = np.array(img)
    os.remove('img.jpg') # delete the image
    return Image.fromarray(img_arr)
```

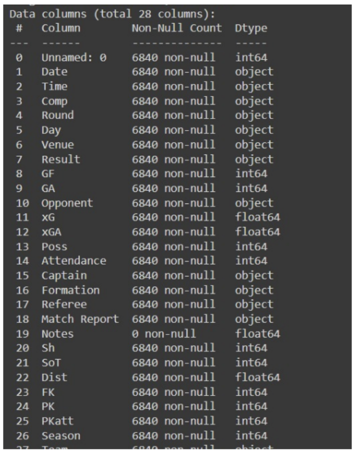


```python
def compare_teams(team1_ext,team2_ext,team1,team2):

    params = ["points_per_game","g_scored_per_game","g_conc_per_game","xG","xGC"]

    url = 'https://fbref.com/en/squads/' + team1_ext
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    info_div = soup.find('div', {'id': 'info'})
    p_tags = info_div.find_all('p')
    points_per_game = (float((p_tags[0].text)[41:45]))
    g_scored_per_game = (float((p_tags[2].text)[11:15]))
    g_conc_per_game = (float((p_tags[2].text)[55:59]))
    xG = (float((p_tags[3].text)[4:8]))
    xGC = (float((p_tags[3].text)[24:28]))
    values1 = [points_per_game,g_scored_per_game,g_conc_per_game,xG,xGC]

    url = 'https://fbref.com/en/squads/' + team2_ext
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    info_div = soup.find('div', {'id': 'info'})
    p_tags = info_div.find_all('p')
    points_per_game = (float((p_tags[0].text)[41:45]))
    g_scored_per_game = (float((p_tags[2].text)[11:15]))
    g_conc_per_game = (float((p_tags[2].text)[55:59]))
    xG = (float((p_tags[3].text)[4:8]))
    xGC = (float((p_tags[3].text)[24:28]))
    values2 = [points_per_game,g_scored_per_game,g_conc_per_game,xG,xGC]

    #CHANGE

    low  = [0.0,0.0,0.0,0.0,0.0]
    high = [10.0,5.0,5.0,50.0,50.0]
    radar = Radar(params,low,high,num_rings=4,ring_width=1,center_circle_radius=1)
    fig, ax = radar.setup_axis()
    rings_inner = radar.draw_circles(ax=ax, facecolor='#ffb2b2', edgecolor='#fc5f5f')
    radar_output = radar.draw_radar_compare(values1, values2, ax=ax,
                                             kwargs_radar={'facecolor': 'blue', 'alpha': 0.8},
                                            kwargs_compare={'facecolor': 'red', 'alpha': 0.4})
    radar_poly, radar_poly2, vertices1, vertices2 = radar_output
    range_labels = radar.draw_range_labels(ax=ax, fontsize=15)
    param_labels = radar.draw_param_labels(ax=ax, fontsize=15)
    plt.savefig('img.png') # save the image
    image = Image.open('img.png')
    image = image.convert('RGB')
    image.save('img.jpg','JPEG')
    image_path = 'img.jpg' # convert the image into numpy array
    img = Image.open(image_path)
    img_arr = np.array(img)
    os.remove('img.jpg') # delete the image
    return Image.fromarray(img_arr)
```

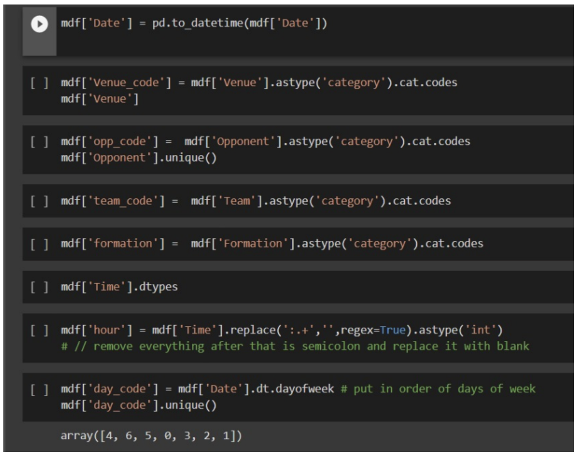


```python
def compare_players(name1_ext,name2_ext,player1,player2):

    base_url = 'https://fbref.com/en/players/'
    params = ["Matches","Goals","Assists","Yellow Cards","Red Cards","Expected Goals","expected Assist Goals","Progressive Carries","Progressive Goals"]

    url = base_url + name1_ext
    standard_ext = '#stats_standard_dom_lg'
    standard_url = url + standard_ext # get the url
    standard_response = requests.get(standard_url) # get the response
    standard_html_content = standard_response.content; # get the html content
    standard_soup = BeautifulSoup(standard_response.content, 'html.parser') # beautify them
    standard_tfoot = list(standard_soup.find("tfoot"))  # extract the footer values
    std_data_list = standard_tfoot[0].find_all("td") # extract the td list
    values1 = [float(std_data_list[7].text),float(std_data_list[8].text),float(std_data_list[9].text),float(std_data_list[14].text),float(std_data_list[15].text),
              float(std_data_list[16].text),float(std_data_list[18].text),float(std_data_list[20].text),float(std_data_list[21].text)]

    url = base_url + name2_ext
    standard_ext = '#stats_standard_dom_lg'
    standard_url = url + standard_ext # get the url
    standard_response = requests.get(standard_url) # get the response
    standard_html_content = standard_response.content; # get the html content
    standard_soup = BeautifulSoup(standard_response.content, 'html.parser') # beautify them
    standard_tfoot = list(standard_soup.find("tfoot"))  # extract the footer values
    std_data_list = standard_tfoot[0].find_all("td") # extract the td list
    values2 = [float(std_data_list[7].text),float(std_data_list[8].text),float(std_data_list[9].text),float(std_data_list[14].text),float(std_data_list[15].text),
              float(std_data_list[16].text),float(std_data_list[18].text),float(std_data_list[20].text),float(std_data_list[21].text)]

    #CHANGE

    low = []
    high = []
    for i in range(len(values1)):
        low.append(0.0)
        high.append(700.0)
    radar = Radar(params,low,high,num_rings=4,ring_width=1,center_circle_radius=1)



    fig, ax = radar.setup_axis()
    color1 = "blue"
    color2 = "red"
    rings_inner = radar.draw_circles(ax=ax, facecolor='#ffb2b2', edgecolor='#fc5f5f')
    radar_output = radar.draw_radar_compare(values1, values2, ax=ax,
                                            kwargs_radar={'facecolor': 'blue', 'alpha': 0.8},
                                            kwargs_compare={'facecolor': 'red', 'alpha': 0.4})
    radar_poly, radar_poly2, vertices1, vertices2 = radar_output
    range_labels = radar.draw_range_labels(ax=ax, fontsize=15)
    param_labels = radar.draw_param_labels(ax=ax, fontsize=15)
    plt.savefig('img.png') # save the image
    image = Image.open('img.png')
    image = image.convert('RGB')
    image.save('img.jpg','JPEG')
    image_path = 'img.jpg' # convert the image into numpy array
    img = Image.open(image_path)
    img_arr = np.array(img)
    os.remove('img.jpg') # delete the image
    return Image.fromarray(img_arr)
```

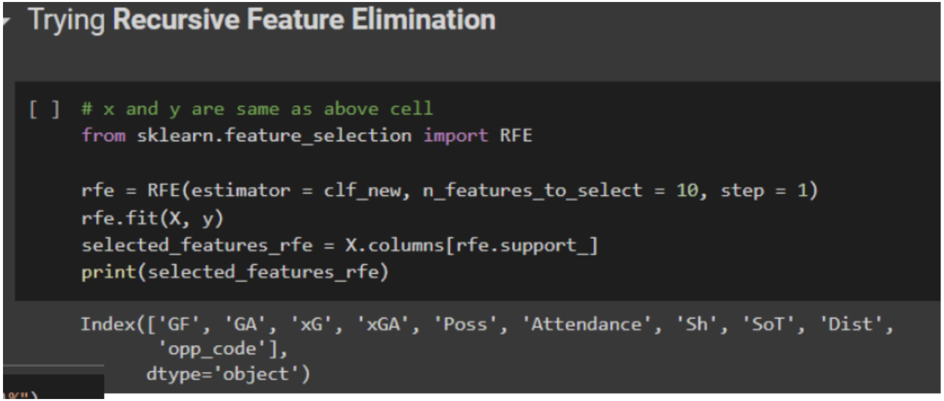
Important predictor code specifics:



```
Data columns (total 28 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Unnamed: 0    6840 non-null   int64  
 1   Date          6840 non-null   object 
 2   Time          6840 non-null   object 
 3   Comp          6840 non-null   object 
 4   Round         6840 non-null   object 
 5   Day           6840 non-null   object 
 6   Venue         6840 non-null   object 
 7   Result        6840 non-null   object 
 8   GF            6840 non-null   int64  
 9   GA            6840 non-null   int64  
 10  Opponent      6840 non-null   object 
 11  xG            6840 non-null   float64
 12  xGA           6840 non-null   float64
 13  Poss          6840 non-null   int64  
 14  Attendance    6840 non-null   int64  
 15  Captain       6840 non-null   object 
 16  Formation     6840 non-null   object 
 17  Referee       6840 non-null   object 
 18  Match Report  6840 non-null   object 
 19  Notes         0 non-null      float64
 20  Sh            6840 non-null   int64  
 21  SoT           6840 non-null   int64  
 22  Dist          6840 non-null   float64
 23  FK            6840 non-null   int64  
 24  PK            6840 non-null   int64  
 25  PKatt         6840 non-null   int64  
 26  Season        6840 non-null   int64  
 27  Team          6840 non-null   object 
```

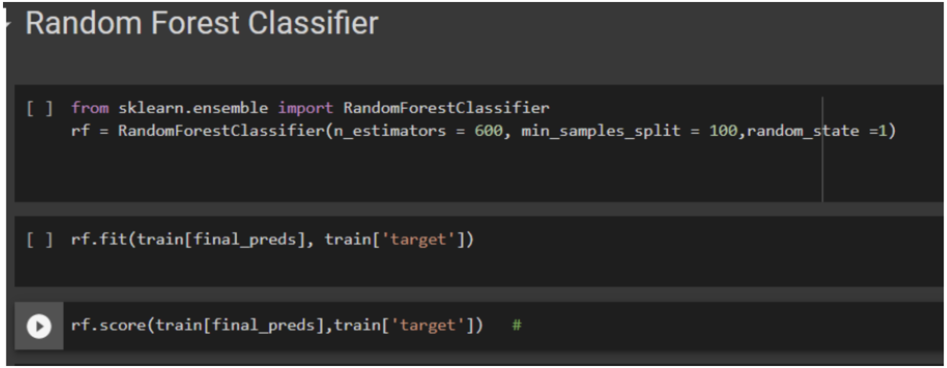


```
mdf['Date'] = pd.to_datetime(mdf['Date'])

[ ] mdf['Venue_code'] = mdf['Venue'].astype('category').cat.codes
    mdf['Venue']

[ ] mdf['opp_code'] =  mdf['Opponent'].astype('category').cat.codes
    mdf['Opponent'].unique()

[ ] mdf['team_code'] =  mdf['Team'].astype('category').cat.codes

[ ] mdf['formation'] =  mdf['Formation'].astype('category').cat.codes

[ ] mdf['Time'].dtypes

[ ] mdf['hour'] = mdf['Time'].replace(':.+','',regex=True).astype('int')
    # // remove everything after that is semicolon and replace it with blank

[ ] mdf['day_code'] = mdf['Date'].dt.dayofweek # put in order of days of week
    mdf['day_code'].unique()

    array([4, 6, 5, 0, 3, 2, 1])
```



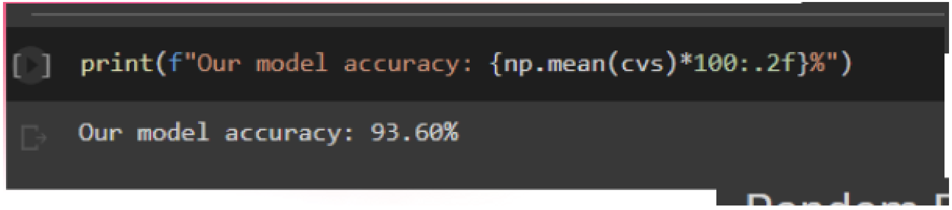
Trying Recursive Feature Elimination

```
[ ] # x and y are same as above cell
    from sklearn.feature_selection import RFE

    rfe = RFE(estimator = clf_new, n_features_to_select = 10, step = 1)
    rfe.fit(X, y)
    selected_features_rfe = X.columns[rfe.support_]
    print(selected_features_rfe)

    Index(['GF', 'GA', 'xG', 'xGA', 'Poss', 'Attendance', 'Sh', 'SoT', 'Dist',
           'opp_code'],
          dtype='object')
```



Random Forest Classifier

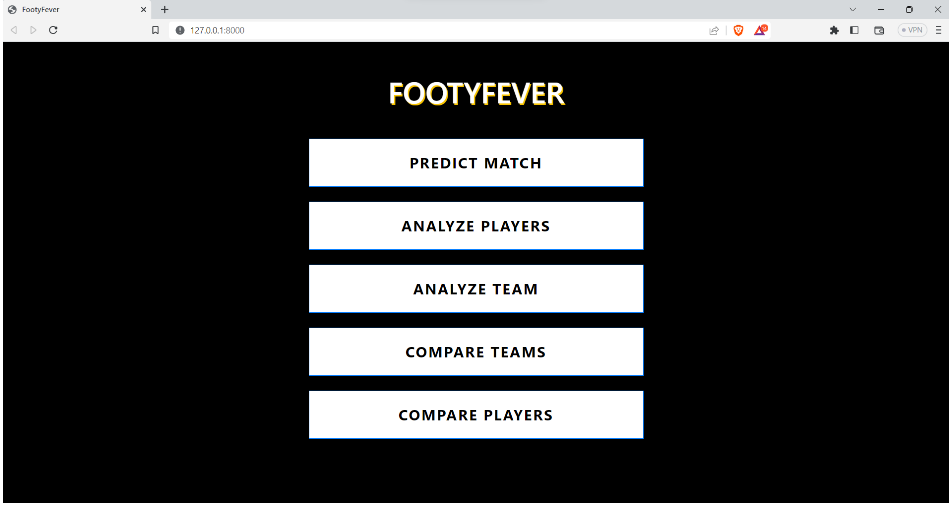
```
[ ] from sklearn.ensemble import RandomForestClassifier
    rf = RandomForestClassifier(n_estimators = 600, min_samples_split = 100,random_state =1)

[ ] rf.fit(train[final_preds], train['target'])

    rf.score(train[final_preds],train['target'])   #
```



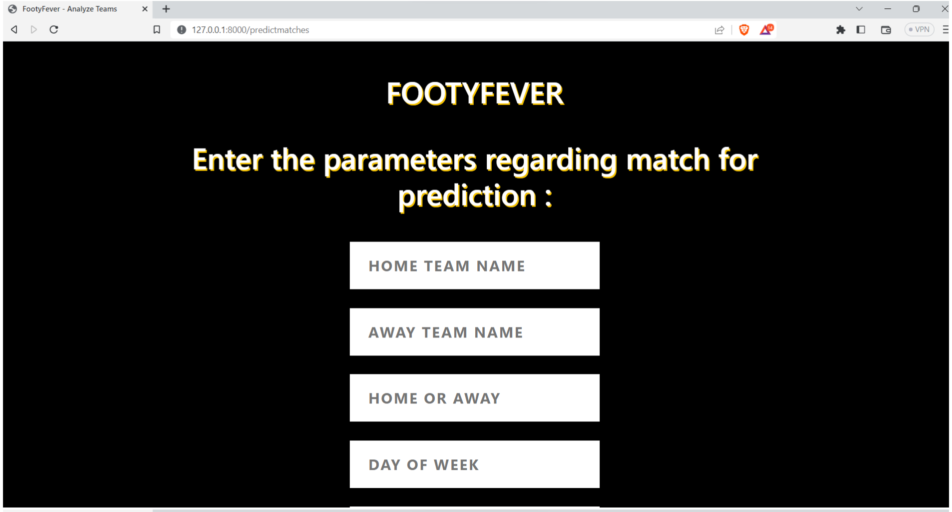
```
[ ] print(f"Our model accuracy: {np.mean(cvs)*100:.2f}%")

    Our model accuracy: 93.60%
```

5 Output Screenshots

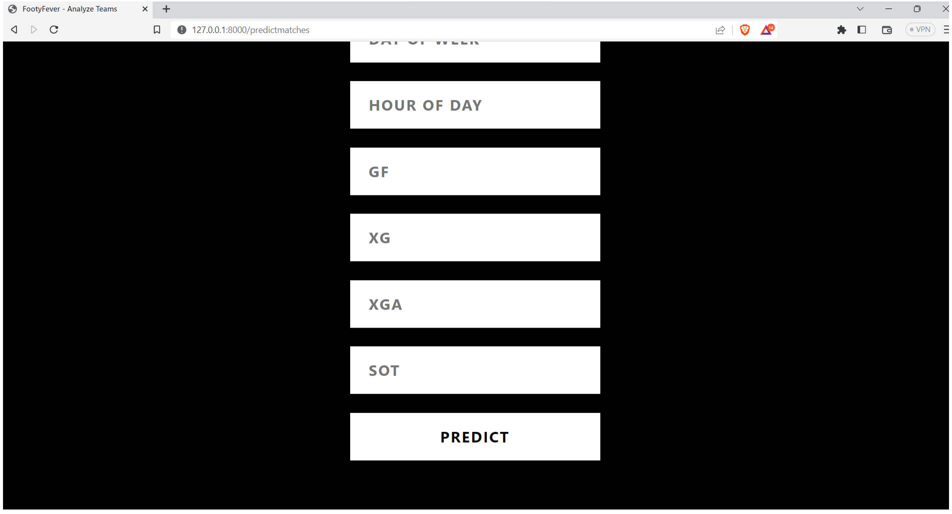
In the beginning, the user is greeted with a very simplistic, yet sophisticated home page displaying all the functionalities of the web-app.



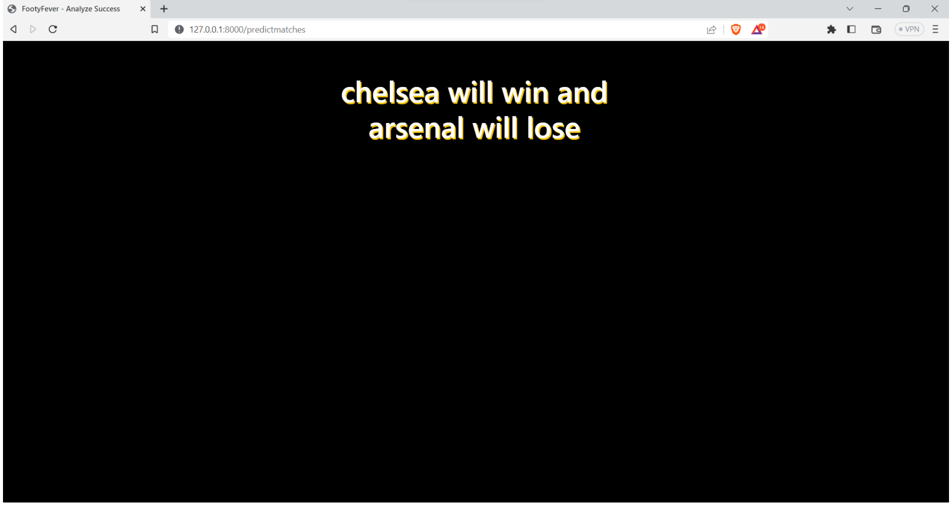
Lets now discover the pages one-by-one. Here is what the GUI of the prediction page looks like:



Enter the parameters regarding match for prediction :

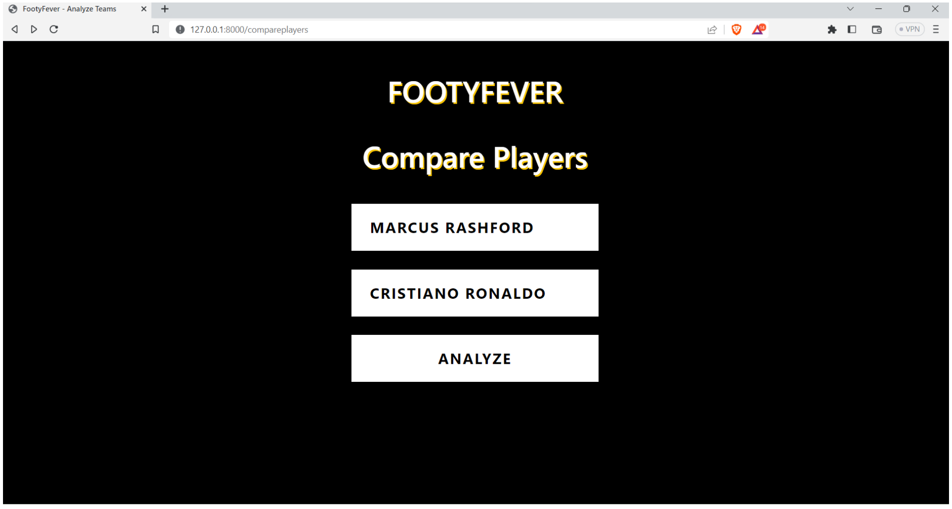


Once the user enters these fields and hit predict, in this case we analysed the output of Chealsea and Arsenel giving their necessary parameters to the predictor and are displayed with the following output



chelsea will win and
arsenal will lose

Similarly, lets now discover the compare players functionality. Inputting the names of 2 players in the respective fields:



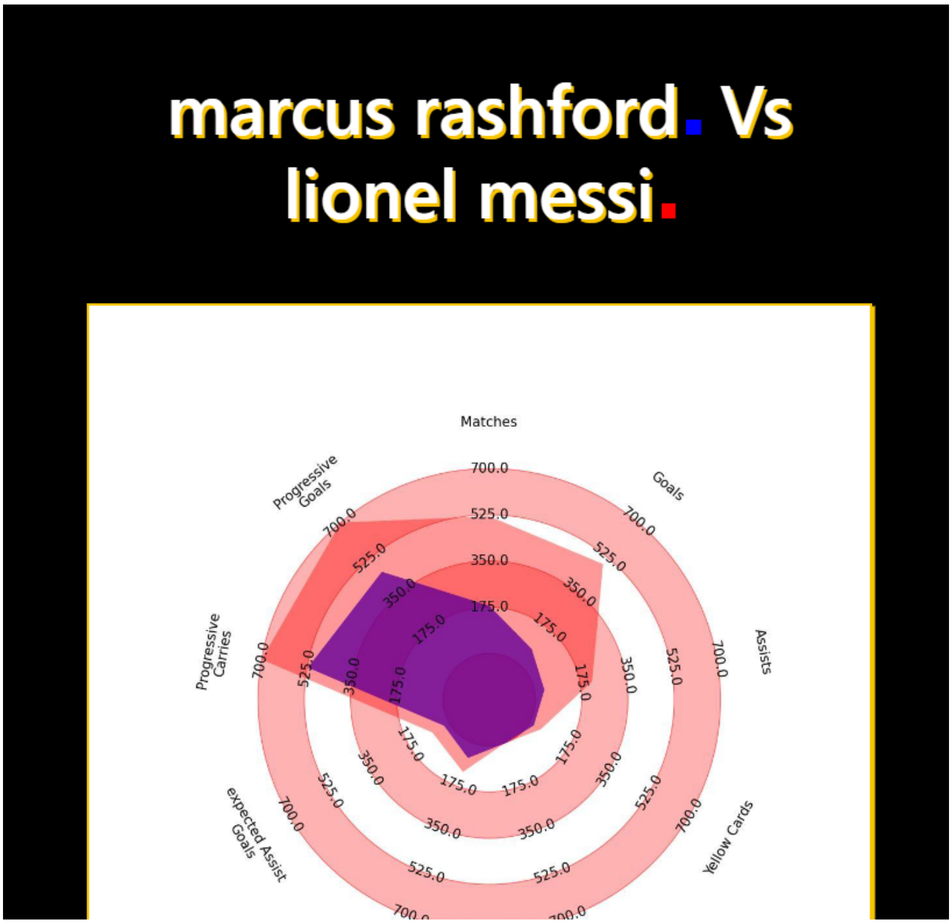
FOOTYFEVER

Compare Players

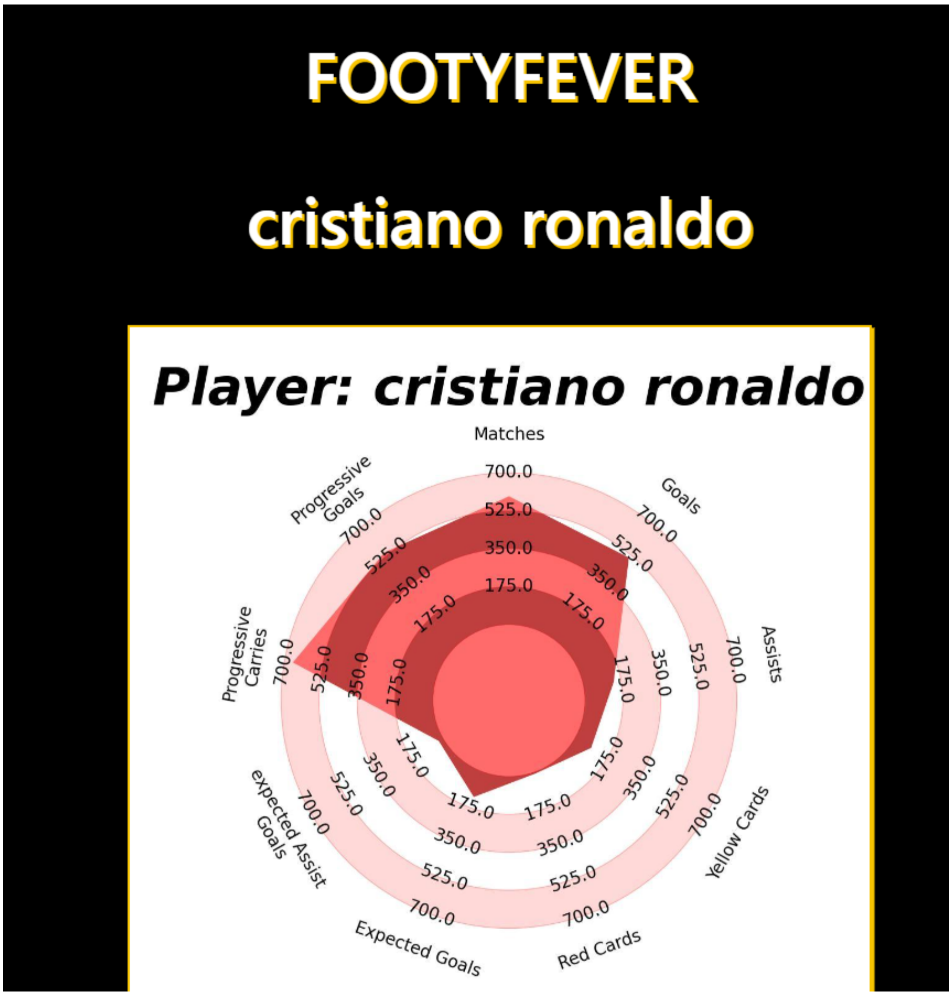
MARCUS RASHFORD

CRISTIANO RONALDO

This will yield the following result:



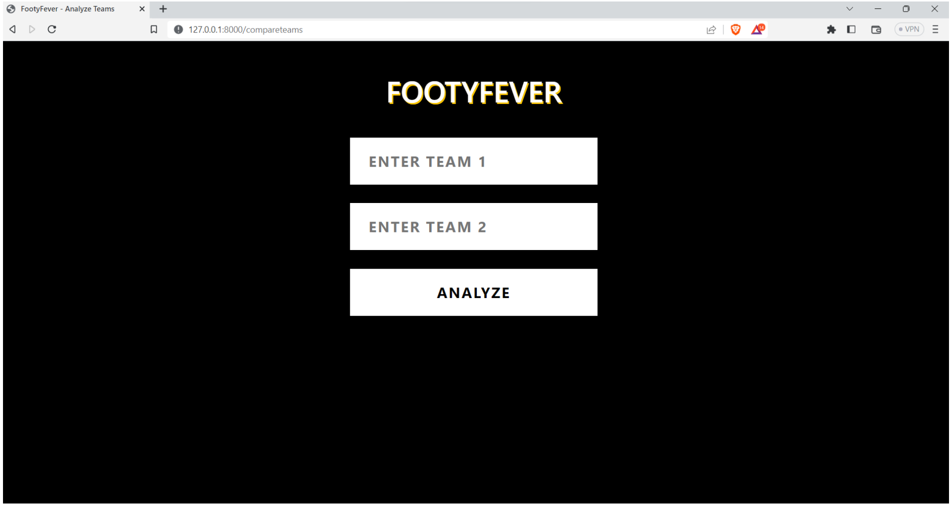
A functionality also allows you to analyze individual players on the giving their name as input.



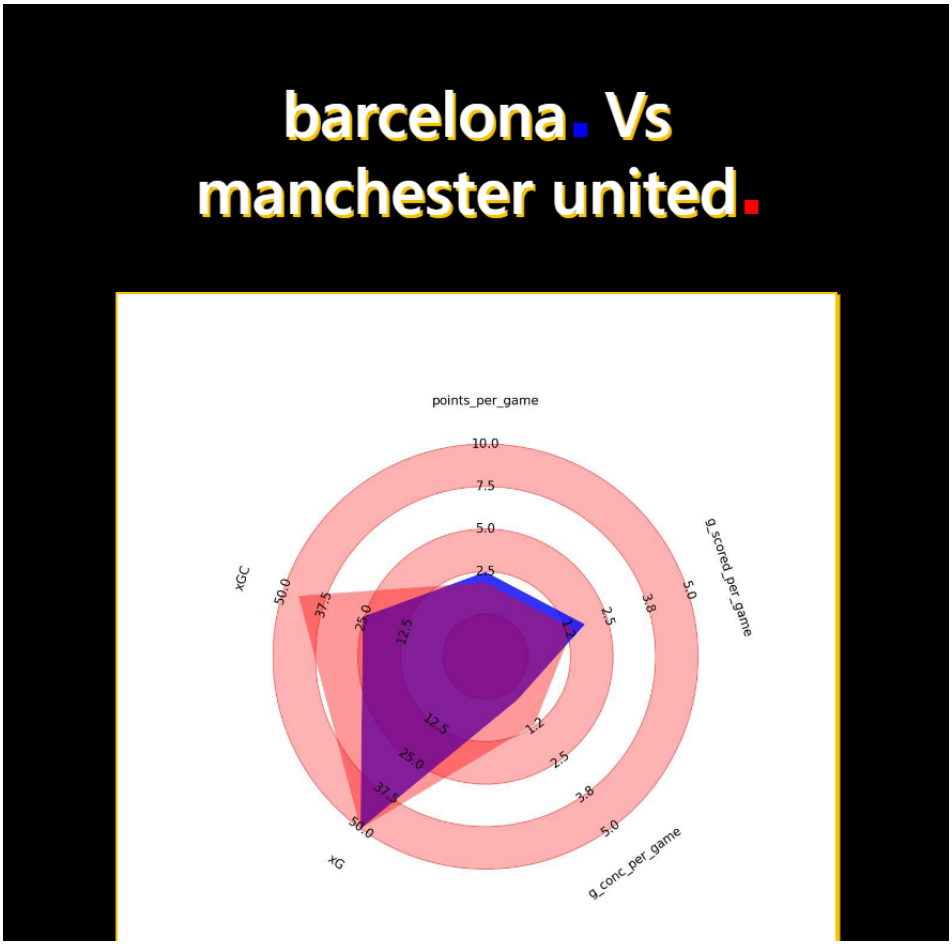
FOOTYFEVER

cristiano ronaldo

Similar functionalities are also implemented for analyzing and comparing teams respectively.

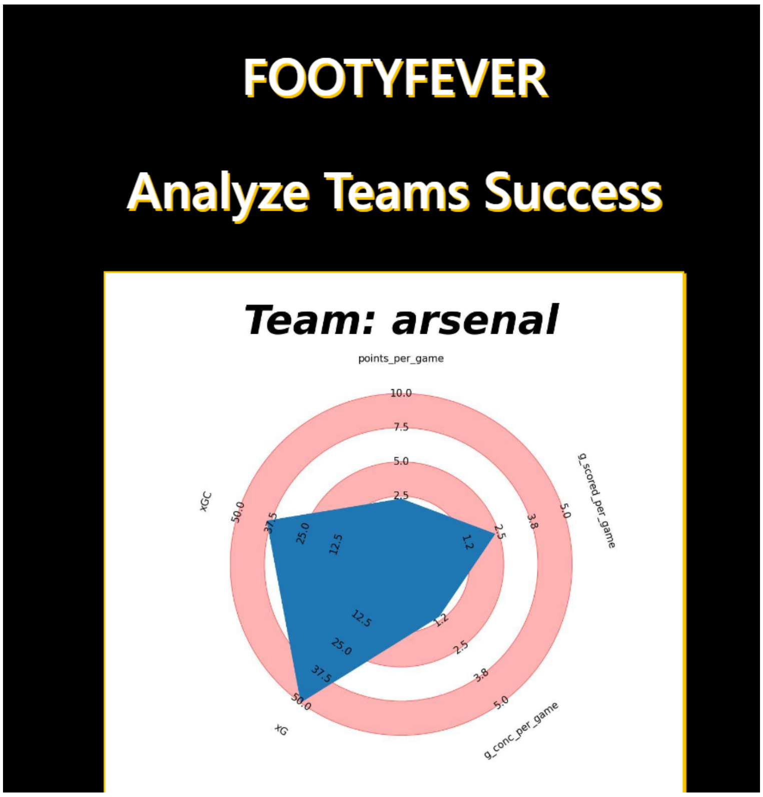


FOOTYFEVER



barcelona 🟦 Vs
manchester united 🟥

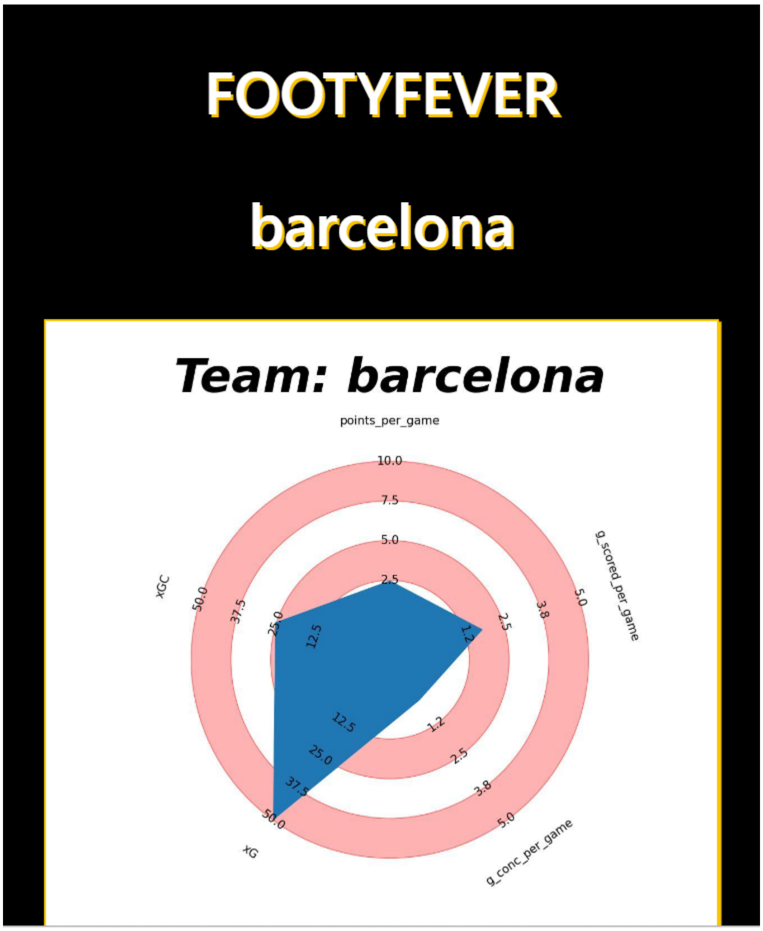
For analysing individual teams:



FOOTYFEVER

Analyze Teams Success

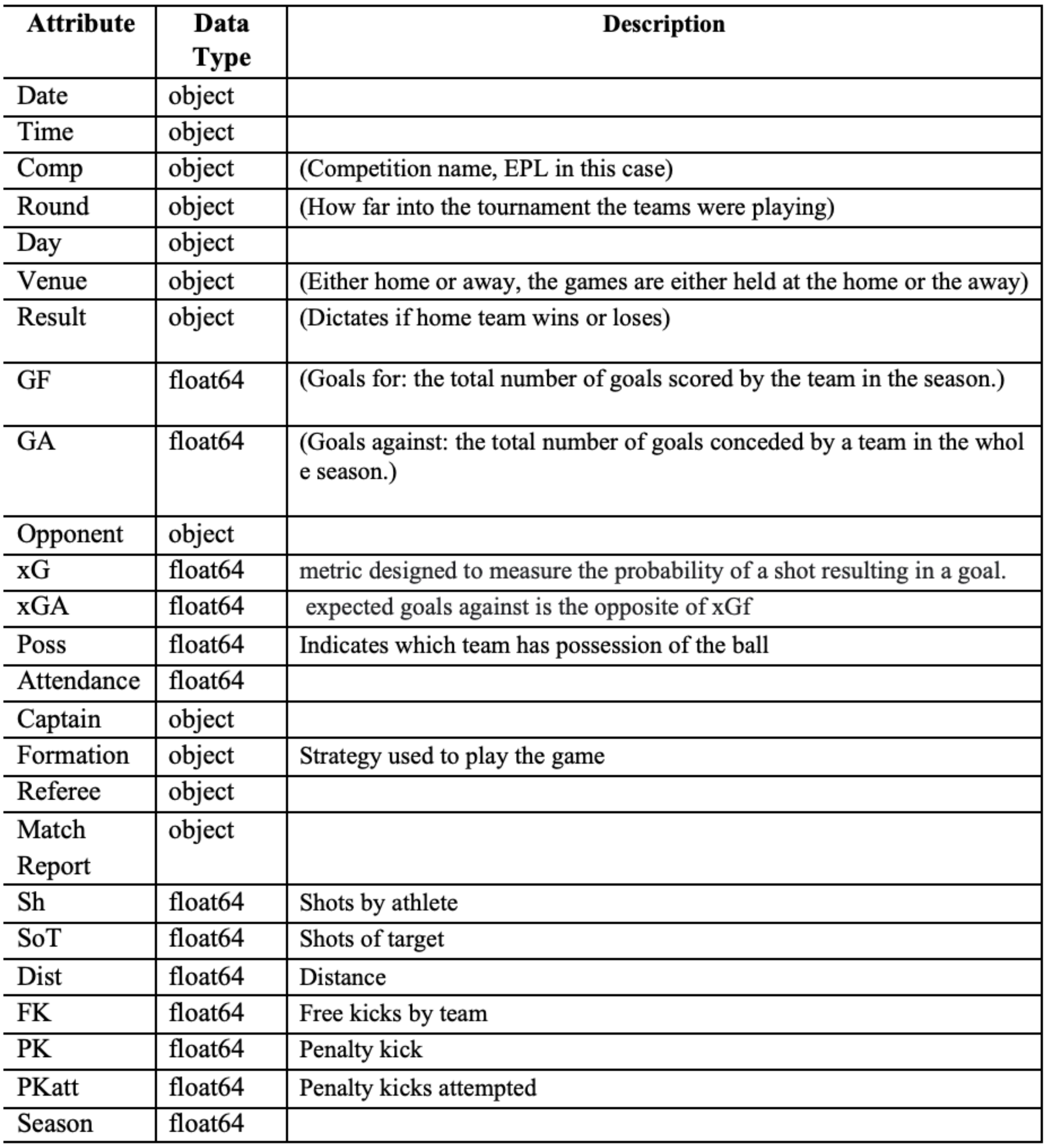
Team: arsenal



6 Description and Analysis of Parameters used in Project

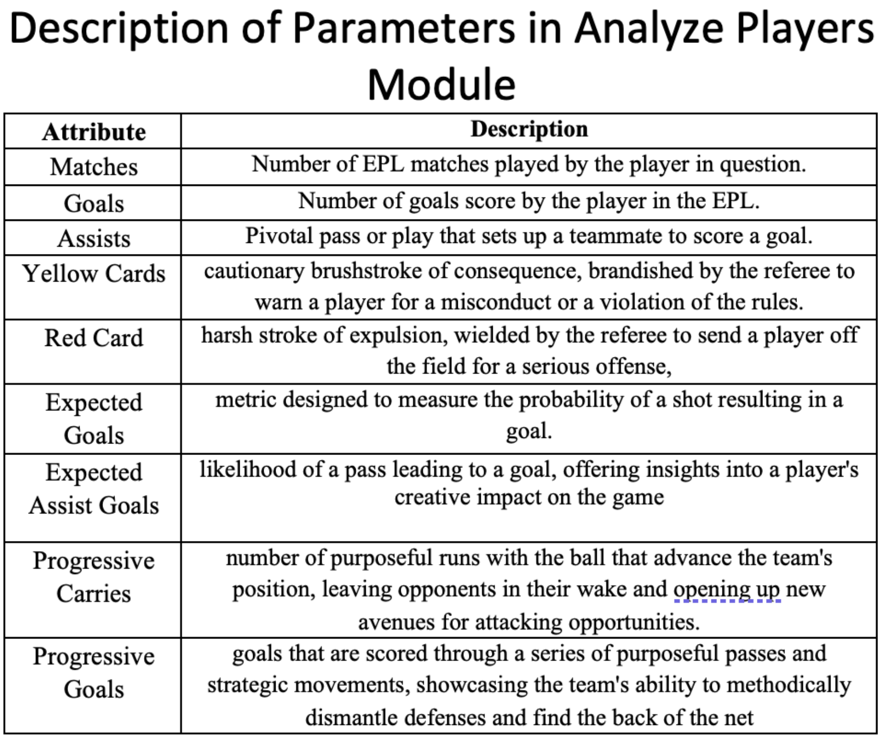
To provide a description of the football technicalities used in the project, let us look at each of the parameters and understand its meaning from a lay mans perspective.

To start with, consider the parameters used in the Win - Lose predictor



Attribute	Data Type	Description
Date	object	
Time	object	
Comp	object	(Competition name, EPL in this case)
Round	object	(How far into the tournament the teams were playing)
Day	object	
Venue	object	(Either home or away, the games are either held at the home or the away)
Result	object	(Dictates if home team wins or loses)
GF	float64	(Goals for: the total number of goals scored by the team in the season.)
GA	float64	(Goals against: the total number of goals conceded by a team in the whol e season.)
Opponent	object	
xG	float64	metric designed to measure the probability of a shot resulting in a goal.
xGA	float64	expected goals against is the opposite of xGf
Poss	float64	Indicates which team has possession of the ball
Attendance	float64	
Captain	object	
Formation	object	Strategy used to play the game
Referee	object	
Match Report	object	
Sh	float64	Shots by athlete
SoT	float64	Shots of target
Dist	float64	Distance
FK	float64	Free kicks by team
PK	float64	Penalty kick
PKatt	float64	Penalty kicks attempted
Season	float64	

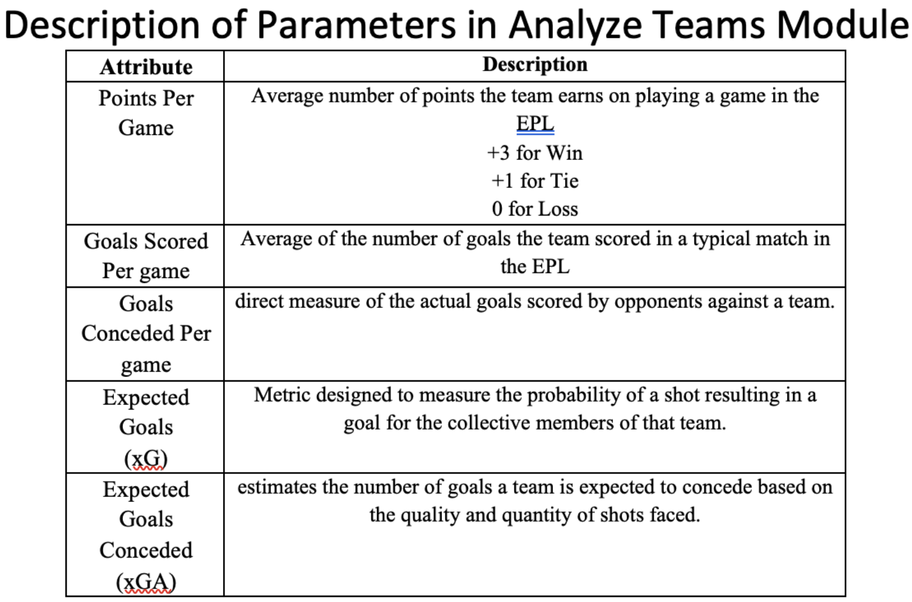
Next, lets delve into some of the possibly baffling characteristics of the compare players module



Description of Parameters in Analyze Players Module

Attribute	Description
Matches	Number of EPL matches played by the player in question.
Goals	Number of goals score by the player in the EPL.
Assists	Pivotal pass or play that sets up a teammate to score a goal.
Yellow Cards	cautionary brushstroke of consequence, brandished by the referee to warn a player for a misconduct or a violation of the rules.
Red Card	harsh stroke of expulsion, wielded by the referee to send a player off the field for a serious offense,
Expected Goals	metric designed to measure the probability of a shot resulting in a goal.
Expected Assist Goals	likelihood of a pass leading to a goal, offering insights into a player's creative impact on the game
Progressive Carries	number of purposeful runs with the ball that advance the team's position, leaving opponents in their wake and opening up new avenues for attacking opportunities.
Progressive Goals	goals that are scored through a series of purposeful passes and strategic movements, showcasing the team's ability to methodically dismantle defenses and find the back of the net

Finally, lets learn more about the different parameter of the analyze teams module



Description of Parameters in Analyze Teams Module

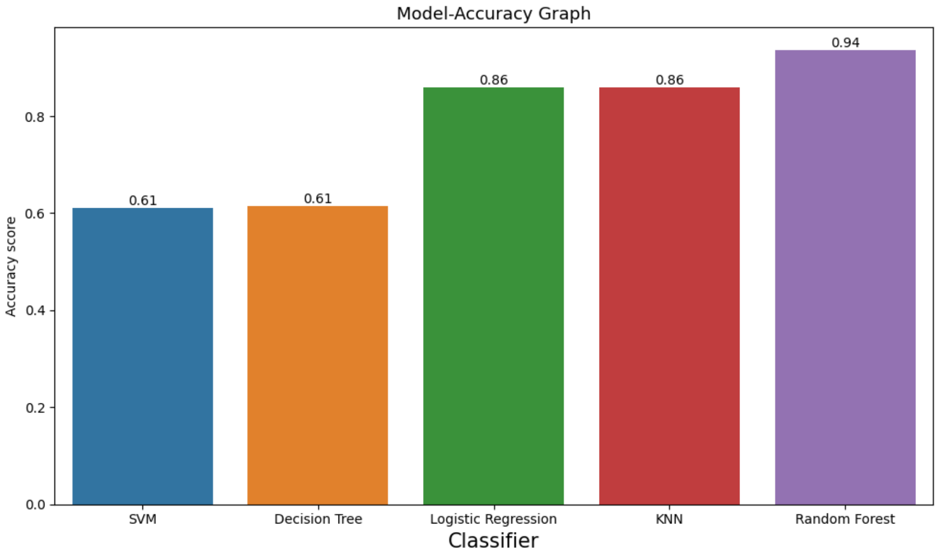
Attribute	Description
Points Per Game	Average number of points the team earns on playing a game in the EPL +3 for Win +1 for Tie 0 for Loss
Goals Scored Per game	Average of the number of goals the team scored in a typical match in the EPL
Goals Conceded Per game	direct measure of the actual goals scored by opponents against a team.
Expected Goals (xG)	Metric designed to measure the probability of a shot resulting in a goal for the collective members of that team.
Expected Goals Conceded (xGA)	estimates the number of goals a team is expected to concede based on the quality and quantity of shots faced.

7 Results

We are extremely proud to have successfully implemented what we set out to develop. Before quantitatively identifying some figures relating to our project, we are elated to have developed the infrastructure of the webapp which can be easily expanded upon and made better going ahead into the future.

This can be better elaborated in the subsequent section but to provide a brief example, the same base (for the webapp) can be expanded upon to encompass other sports like cricket, basketball. Encompassing such sports will provide for a future proof sport consumption app encompassing all sports in general.

Coming to the more quantitative analysis of our implementation, specifically the machine learning model:
Initially, we played around with what models we could use. After having narrowed down on some of the possible classifiers we could use, we put each to the test. Here is a brief accuracy summary we got for the models we short-listed by training them and later testing them on our data.

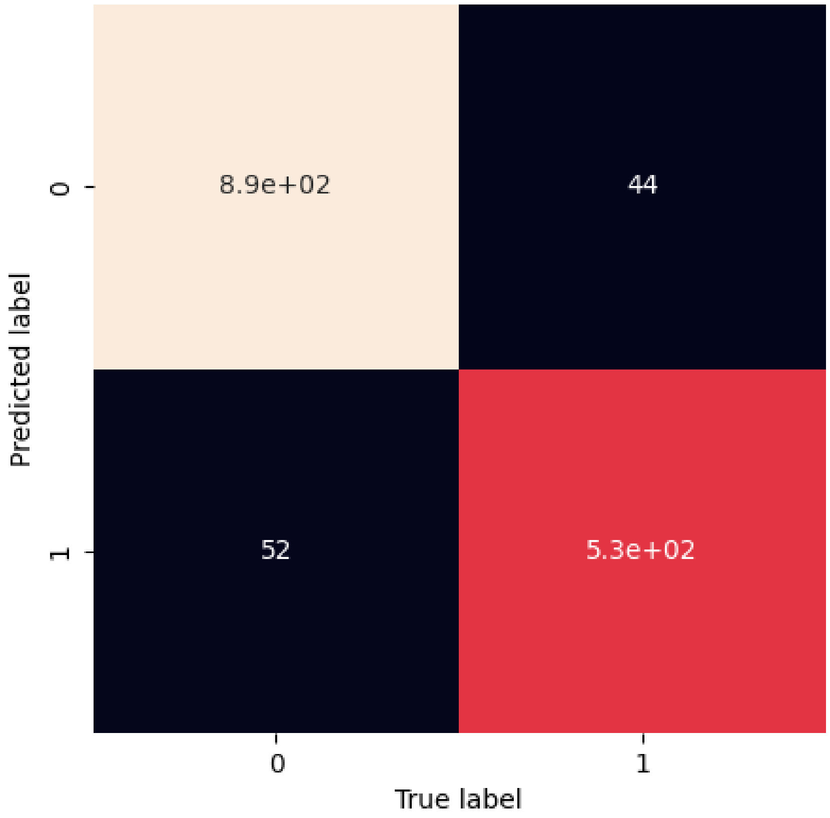


Since the Random Forest Classifier yielded the highest accuracy, we decided to go ahead with it.
Narrowing down to the performance of our model specifically, it was yielded some of the following scores.

- Accuracy: 93.68%
- Precision: 92%
- Recall: 91%

- F1 score: 92%

A more accepted, liquid form of validation can be summarized through a confusion matrix, the same is displayed below.



Given the nature and the proximity of these number, we can be sure that the fitting of the model with the data is free from factors such as over-fitting etc.
This too can be justified since with the cross validation that we performed.

Cross-validation, is a technique that uses multiple validation sets to estimate the generalization error of the model. In cross-validation, the data is split into K equal-sized folds, and the model is trained and validated K times, with each fold used as the validation set once, and the remaining K-1 folds used as the training set. The performance of the model is then averaged over the K iterations to get a more reliable estimate of the generalization error.

The main advantage of cross-validation over validation is that it provides a more accurate estimate of the generalization error, since it uses multiple validation sets instead of a single one. This is particularly useful when the amount of data is limited, or when the data is highly imbalanced or heterogeneous. Cross-validation can also be used to tune hyperparameters, which are parameters that are not learned by the model but must be set by the user, such as the regularization parameter or the learning rate.

Its derivation from the validation can be observed by the picture below, as well as its performance on our model:

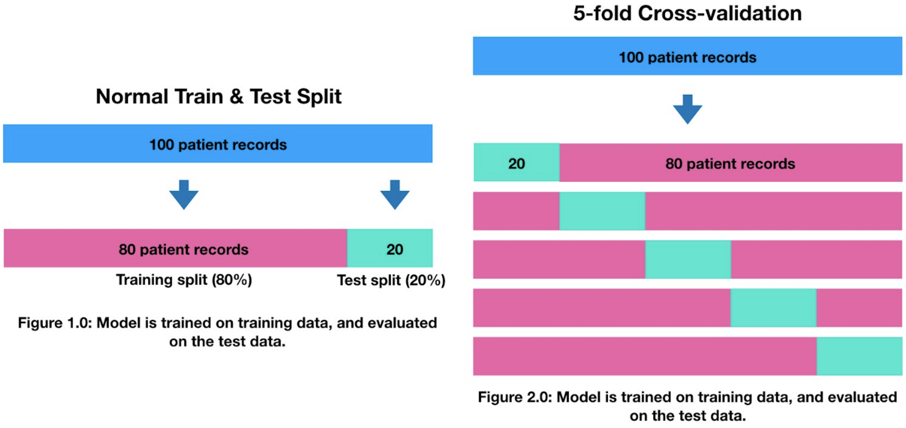
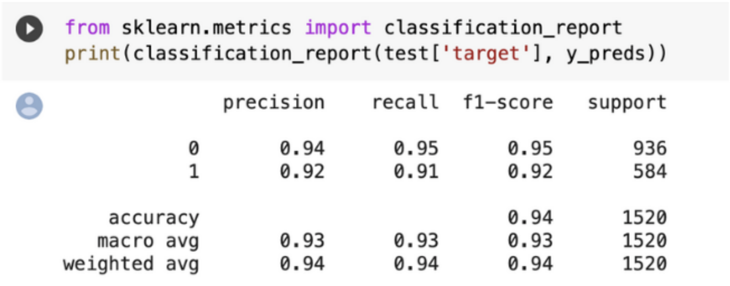


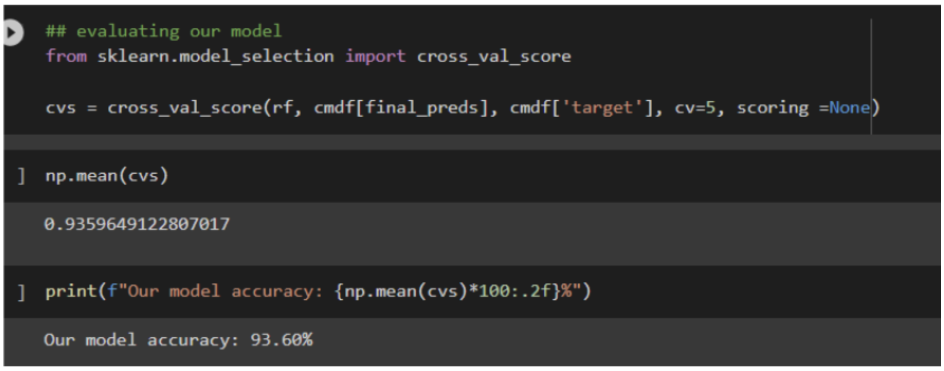
Figure 1.0: Model is trained on training data, and evaluated on the test data.

Figure 2.0: Model is trained on training data, and evaluated on the test data.



```
from sklearn.metrics import classification_report
print(classification_report(test['target'], y_preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       936
           1       0.92      0.91      0.92       584

    accuracy                           0.94      1520
   macro avg       0.93      0.93      0.93      1520
weighted avg       0.94      0.94      0.94      1520
```



```
## evaluating our model
from sklearn.model_selection import cross_val_score

cvs = cross_val_score(rf, cmdf[final_preds], cmdf['target'], cv=5, scoring =None)
```

```
np.mean(cvs)
```

```
0.9359649122807017
```

```
print(f"Our model accuracy: {np.mean(cvs)*100:.2f}%")
```

```
Our model accuracy: 93.60%
```

8 Future Scope

- Like mentioned in the above section, the project not only serves its purpose of making data more consumable, but we've ensured that we have the essential infrastructure to be easily expanded to encompass other sports, or other events that require an analysis of data and predicting models.

- If taken seriously to where all sports are encompassed in this one app, it can be expanded to create a rival, more engaging platform than ESPN, and other, largely outdated sport statistics monopolies.
- Furthermore, an easy fix in the analysis model can be added where more than 2 players/teams can be compared at a time. Perhaps using a different comparison metric (i.e. other than the all encompassing radar plot).
- Although the accuracy of the predictor model is already at whopping 94% it can possibly be pushed further.