National College of Ireland

Investigating the validity of FPL data in determining player performance and the most impactful players in the English Premier League teams

MSc Research Project
MSc Data Analytics

Nishant Meena
Student ID: x21221839

School of Computing
National College of Ireland

Supervisor: Vitor Horta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…Nishant Meena………………………………………………………
Student ID:	…x21221839……………………………………………………………………………..….
Programme	……MSc Data Analytics……….. **Year:** …2023………
Module:	………..MSc Research Project…………………………………………………….………
Supervisor:	……………Vitor Horta……………………………………………………….………
Submission Due Date:	………..14-08-2023…………………………………………….………
Project Title:	Investigating the validity of FPL data in determining player performance and the most impactful players in the English Premier League teams.
Word Count:	……1002………………… **Page Count**……………..11………..……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………..Nishant Meena…………………………………………

Date: ………………………..13/08/2023…………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nishant Meena
Student ID:x21221839
MSc Research Project in Data Analytics
14th August 2023

1. Introduction

This manual's goal is to highlight the project's technical side, which includes system requirements and programming snippets that are not mentioned in the main report. The essential system requirements used are covered at the outset of this document, along with a discussion of how the methodology is put into practice.

1.1 System Requirements

- Hardware spec
  1. System Manufacturer: Apple Inc.
  2. Operating System: macOS
  3. Processor: Apple M1 chip, 8-core CPU with 4 performance cores and 4 efficiency cores
  4. Memory: 8GB unified memory

- Software spec
  1. Jupyter notebook
  2. Google colab
  3. Microsoft Excel

2. Project Development

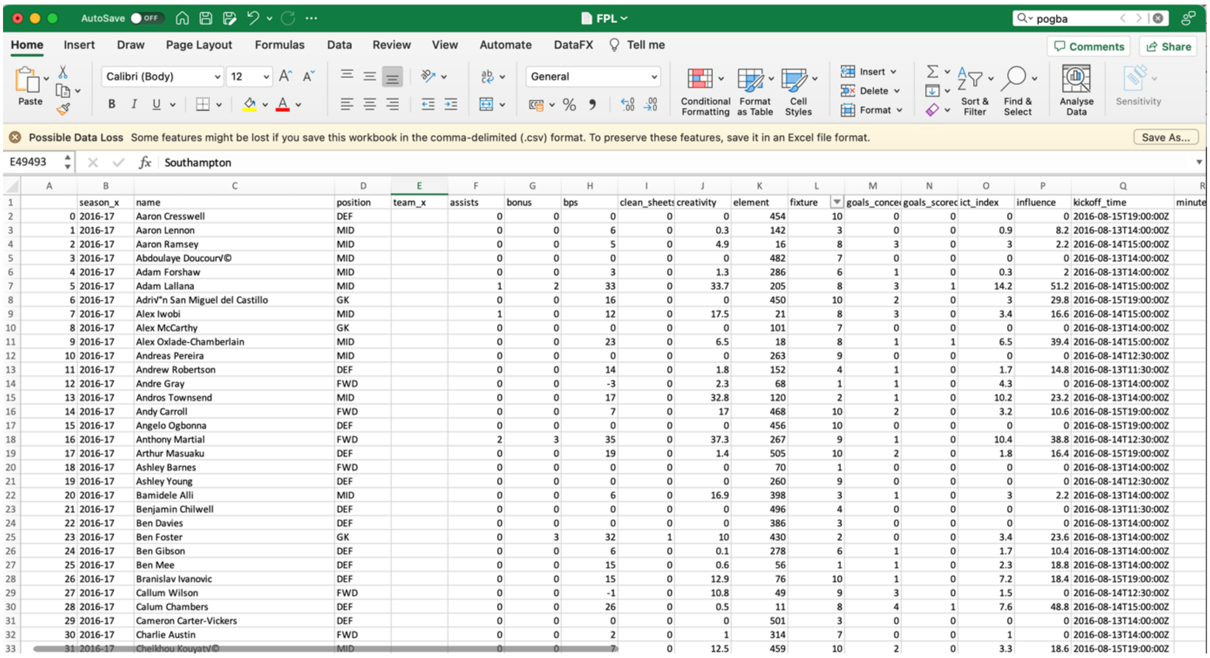
Data preparation is achieved in multiple stages, between Excel, jupyter notebook and google colab. The screenshots have been included in the manual to make everything clear.

2.1 Data Preparation

This project focuses on two datasets: The first dataset was obtained from a github repository[1] which has all the FPL data starting from 2016 season to the latest season. All the game week data has been uploaded on this github

[1] https://github.com/vaastav/Fantasy-Premier-League/tree/master

repository. The second dataset was scraped from the website called understat.com, using the documentation available online [2]. The first dataset contains the stats of the players for each game week from game week 1 of 2016 season to game week 38 of 2021 season as seen in fig 1.



	season_x	name	position	team_x	assists	bonus	bps	clean_sheets	creativity	element	fixture	goals_conce	goals_scorec	ict_index	influence	kickoff_time	minute
0	2016-17	Aaron Cresswell	DEF		0	0	0	0	0	454	10	0	0	0	0	2016-08-15T19:00:00Z	
1	2016-17	Aaron Lennon	MID		0	0	6	0	0.3	142	3	0	0	0.9	8.2	2016-08-13T14:00:00Z	
2	2016-17	Aaron Ramsey	MID		0	0	5	0	4.9	16	8	3	0	3	2.2	2016-08-14T15:00:00Z	
3	2016-17	Abdoulaye Doucourv©	MID		0	0	0	0	0	482	7	0	0	0	0	2016-08-13T14:00:00Z	
4	2016-17	Adam Forshaw	MID		0	0	3	0	1.3	286	6	1	0	0.3	2	2016-08-13T14:00:00Z	
5	2016-17	Adam Lallana	MID		1	2	33	0	33.7	205	8	3	1	14.2	51.2	2016-08-14T15:00:00Z	
6	2016-17	Adriv°n San Miguel del Castillo	GK		0	0	16	0	0	450	10	2	0	3	29.8	2016-08-15T19:00:00Z	
7	2016-17	Alex Iwobi	MID		1	0	12	0	17.5	21	8	3	0	3.4	16.6	2016-08-14T15:00:00Z	
8	2016-17	Alex McCarthy	GK		0	0	0	0	0	101	7	0	0	0	0	2016-08-13T14:00:00Z	
9	2016-17	Alex Oxlade-Chamberlain	MID		0	0	23	0	6.5	18	8	1	1	6.5	39.4	2016-08-14T15:00:00Z	
10	2016-17	Andreas Pereira	MID		0	0	0	0	0	263	9	0	0	0	0	2016-08-14T12:30:00Z	
11	2016-17	Andrew Robertson	DEF		0	0	14	0	1.8	152	4	1	0	1.7	14.8	2016-08-13T11:30:00Z	
12	2016-17	Andre Gray	FWD		0	0	-3	0	2.3	68	1	1	0	4.3	0	2016-08-13T14:00:00Z	
13	2016-17	Andros Townsend	MID		0	0	17	0	32.8	120	2	1	0	10.2	23.2	2016-08-13T14:00:00Z	
14	2016-17	Andy Carroll	FWD		0	0	7	0	17	468	10	2	0	3.2	10.6	2016-08-15T19:00:00Z	
15	2016-17	Angelo Ogbonna	DEF		0	0	0	0	0	456	10	0	0	0	0	2016-08-15T19:00:00Z	
16	2016-17	Anthony Martial	FWD		2	3	35	0	37.3	267	9	1	0	10.4	38.8	2016-08-14T12:30:00Z	
17	2016-17	Arthur Masuaku	DEF		0	0	19	0	1.4	505	10	2	0	1.8	16.4	2016-08-15T19:00:00Z	
18	2016-17	Ashley Barnes	FWD		0	0	0	0	0	70	1	0	0	0	0	2016-08-13T14:00:00Z	
19	2016-17	Ashley Young	DEF		0	0	0	0	0	260	9	0	0	0	0	2016-08-14T12:30:00Z	
20	2016-17	Bamidele Alli	MID		0	0	6	0	16.9	398	3	1	0	3	2.2	2016-08-13T14:00:00Z	
21	2016-17	Benjamin Chilwell	DEF		0	0	0	0	0	496	4	0	0	0	0	2016-08-13T11:30:00Z	
22	2016-17	Ben Davies	DEF		0	0	0	0	0	386	3	0	0	0	0	2016-08-13T14:00:00Z	
23	2016-17	Ben Foster	GK		0	3	32	1	10	430	2	0	0	3.4	23.6	2016-08-13T14:00:00Z	
24	2016-17	Ben Gibson	DEF		0	0	6	0	0.1	278	6	1	0	1.7	10.4	2016-08-13T14:00:00Z	
25	2016-17	Ben Mee	DEF		0	0	15	0	0.6	56	1	1	0	2.3	18.8	2016-08-13T14:00:00Z	
26	2016-17	Branislav Ivanovic	DEF		0	0	15	0	12.9	76	10	1	0	7.2	18.4	2016-08-15T19:00:00Z	
27	2016-17	Callum Wilson	FWD		0	0	-1	0	10.8	49	9	3	0	1.5	0	2016-08-14T12:30:00Z	
28	2016-17	Calum Chambers	DEF		0	0	26	0	0.5	11	8	4	1	7.6	48.8	2016-08-14T15:00:00Z	
29	2016-17	Cameron Carter-Vickers	DEF		0	0	0	0	0	501	3	0	0	0	0	2016-08-13T14:00:00Z	
30	2016-17	Charlie Austin	FWD		0	0	2	0	1	314	7	0	0	1	0	2016-08-13T14:00:00Z	
31	2016-17	Cheikhou Kouyatv©	MID		0	0	7	0	12.5	459	10	2	0	3.3	18.6	2016-08-15T19:00:00Z	

fig 1 FPL dataset

For the second dataset which includes secondary stats (xG and xA) was scraped with the help of the understat documentation. However, to scrape the players stats for each season player id was required which was obtained by running the below mentioned code in figure 2. Further, it is important to know that in English Premier League (EPL), three clubs are relegated (demoted) to lower league and three clubs from the lower leagues are promoted to EPL each season. For the same reason an array of list of teams was manually created to get all the player ids, as seen in figure 2.

[2] https://understat.readthedocs.io/en/latest/contributing/contributing.html

```
import pandas as pd
import asyncio
from understat import Understat

async def fetch_team_data(session, team_name):
    understat = Understat(session)
    players = await understat.get_league_players(
        "epl",
        2018,
        team_title=team_name
    )
    return players

async def main():
    list_teams = ['Manchester United','Chelsea','Tottenham Hotspur','Manchester City','Liverpool','Arsenal','Everton','Southampton','AFC Bournemouth',' West Bromwich Albion',
              'West Ham United','Leicester City','Stoke City','Crystal Palace','Swansea City','Burnley','Watford','Hull City','Middlesbrough','Sunderland','Newcastle United',
              'Brighton & Hove Albion','Huddersfield Town','Wolves','Cardiff City','Sheffield United','Aston Villa','Norwich City','Leeds United','Leeds United','Brentford']  # 1
    list_df = []

    async with aiohttp.ClientSession() as session:
        tasks = [fetch_team_data(session, name) for name in list_teams]
        team_data = await asyncio.gather(*tasks)

    for team_name, data in zip(list_teams, team_data):
        for player in data:
            player["team"] = team_name
            player_name = player["player_name"].replace(" ", "_").lower()
            player["player_name"] = player_name
            list_df.append(player)

    df = pd.DataFrame(list_df)

    return df

# Run the main coroutine using the `await` keyword
result_df = await main()

# Now you can work with the result_df DataFrame
print(result_df)
```

fig 2 Code to obtain the players ids

The next step in scraping the data was to obtain the required stats for the all the players. The focus was to get the 'xG' and 'xA' stats, but the date and seasons were also the fetched as these columns are required to merge with the original FPL dataset, the snippets of the code can be seen in figure 3 below. After which the dataframe is saved to a csv file which is used to merge with the original dataset for second experiment. To merge the scrapped dataset, few changes were made to the name and date column of the FPL dataset to merge the datasets, figure 4. This code removes any numerical characters from the name of the players, in the second line of code, any spaces, dashes, are replaced with underscore, all the non-English characters are converted to their closest English equivalent and finally all the names were converted into lower case. Another change which was made was to extract the date from the kickoff time available in the FPL dataset, as the kickoff time includes date and time of the fixture. The date was extracted and was converted into YYYY-MM-DD to match it with the scraped stats from understat.com, the snippet can be seen below.

```python
import json
import pandas as pd
from bs4 import BeautifulSoup
from urllib.request import urlopen
import unidecode
def scrape(id,name1):
  scrape_url = f"https://understat.com/player/{id}"
  page_connect = urlopen(scrape_url)
  page_html = BeautifulSoup(page_connect, "html.parser")
  page_html.findAll(name="script")
  json_raw_string = page_html.findAll(name="script")[4].string
  print(json_raw_string)
  start_ind = json_raw_string.index("\\")
  stop_ind = json_raw_string.index("')")
  data = json_raw_string[start_ind:stop_ind]
  data = data.encode("utf8").decode("unicode_escape")
  data = json.loads(data)
  df = pd.DataFrame(data)
  dk = df[['xG','xA','date','season']].sort_values(by='date', ascending=False)
  dk['team_id'] = id
  dk['name'] = name1
  return dk
```

```python
combined_df = pd.DataFrame()
for index, row in result_df.iterrows():
    try :
      first_name = row['player_name']
      id_value = row['id']
      print(f"Name: {first_name} , ID: {id_value}")
      combined_df = pd.concat([combined_df, scrape(id_value,first_name)], ignore_index=True)
    except Exception as e:
        print(f"Exception occurred: {e}")
        continue
```

fig 3 Code to scrape the players stats using player IDs

Importing the dataset

```python
In [92]: data = pd.read_csv("/Users/nishant/Desktop/Semester 3/Research Project/FPL.csv")

In [93]: data['full_name'] = data.name.str.replace('_\d+','')
         data['full_name'] = data['full_name'].str.replace(" ", "_").str.replace("-", "_").str.replace('_\d+','')
         data['full_name'] = data['full_name'].apply(lambda x: unidecode.unidecode(x))
         data['full_name'] = data['full_name'].str.lower()

In [108]: data['date1'] = pd.to_datetime(data['kickoff_time']).dt.strftime('%Y-%m-%d')

In [109]: data.head()

Out[109]:
```

nus	bps	clean_sheets	creativity	...	total_points	transfers_balance	transfers_in	transfers_out	value	was_home	yellow_cards	GW	full_name	date1
0	0	0	0.0	...	0	0	0	0	55	False	0	1	aaron_cresswell	2016-08-15
0	6	0	0.3	...	1	0	0	0	60	True	0	1	aaron_lennon	2016-08-13
0	5	0	4.9	...	2	0	0	0	80	True	0	1	aaron_ramsey	2016-08-14
0	0	0	0.0	...	0	0	0	0	50	False	0	1	abdoulaye_doucoure	2016-08-13
0	3	0	1.3	...	1	0	0	0	45	True	1	1	adam_forshaw	2016-08-13

fig 4 Changes made to FPL dataset name and date column

This csv file was saved to local hard disk, which was imported for the second experiment, after which it was merged with the original FPL dataset using the merge function and was joined based on the player's name and the date on which the game was played between the clubs, the snippet of the code can be seen below in figure 5.

Scrapped data from understat

```
In [110]: xa_xg_df = pd.read_csv('/Users/nishant/Downloads/xa_xg (2).csv')

In [111]: xa_xg_df['name1'] = xa_xg_df['name'].str.lower()

In [112]: xa_xg_df.head()
```

Out[112]:

	xG	xA	date	season	team_id	name	name1
0	0.057879	0.043560	2023-05-14	2022	1740	paul_pogba	paul_pogba
1	0.378141	0.000000	2023-05-07	2022	1740	paul_pogba	paul_pogba
2	0.000000	0.313846	2023-05-03	2022	1740	paul_pogba	paul_pogba
3	0.000000	0.000000	2023-04-16	2022	1740	paul_pogba	paul_pogba
4	0.027140	0.000000	2023-03-05	2022	1740	paul_pogba	paul_pogba

```
In [113]: merged_df1 = data.merge(xa_xg_df, left_on=['full_name', 'date1'], right_on=['name1', 'date'],
                         suffixes=('_left', '_right'))

In [114]: merged_df1.head()
```

Out[114]:

x	assists	bonus	bps	clean_sheets	creativity	...	GW	full_name	date1	xG	xA	date	season	team_id	name_right	name1
N	0	0	6	0	0.3	...	1	aaron_lennon	2016-08-13	0.000000	0.000000	2016-08-13	2016	593	aaron_lennon	aaron_lennon
N	0	0	5	0	4.9	...	1	aaron_ramsey	2016-08-14	0.076822	0.000000	2016-08-14	2016	504	aaron_ramsey	aaron_ramsey
N	1	2	33	0	33.7	...	1	adam_lallana	2016-08-14	0.452121	0.177337	2016-08-14	2016	486	adam_lallana	adam_lallana
N	1	0	12	0	17.5	...	1	alex_iwobi	2016-08-14	0.000000	0.072123	2016-08-14	2016	500	alex_iwobi	alex_iwobi
N	0	0	14	0	1.8	...	1	andrew_robertson	2016-08-13	0.000000	0.000000	2016-08-13	2016	1688	andrew_robertson	andrew_robertson

fig 5 Merging the secondary stats with FPL dataset.

The dataset contained some unnecessary columns which were removed like the column 'Unnamed: 0' as it same as the number of rows and the 'kickoff_time' as it was not required for our project. The code can be seen in figure 6.

Data Cleaning

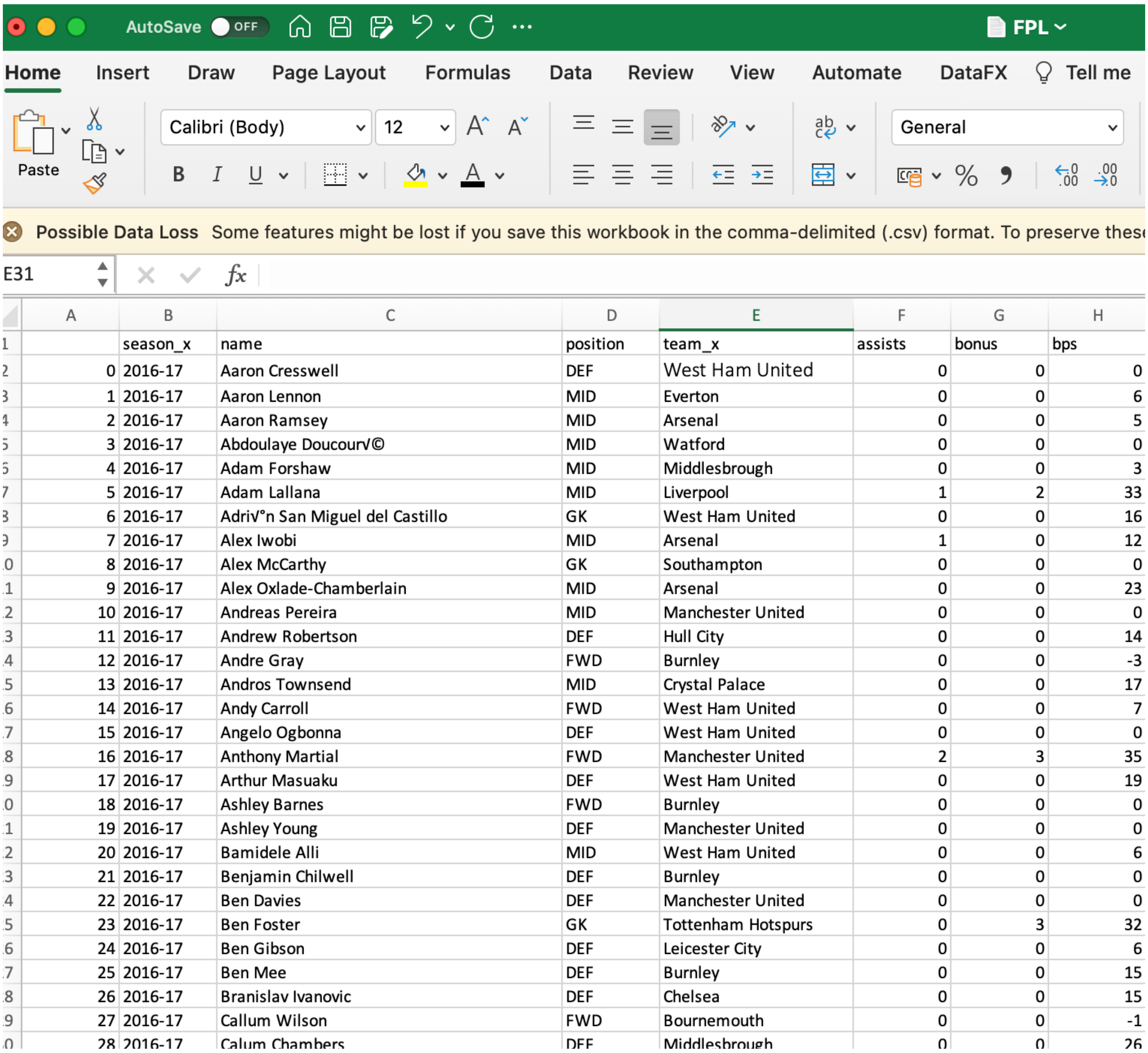
```
In [416]: #removing the unnecessary column
          data = data.drop(['Unnamed: 0','kickoff_time'], axis = 1)

In [417]: #selecting only the players with more than 20 minutes in the match

          data = data[data.minutes>20]
```

fig 6 Cleaning of the dataset

Then the missing values were checked if there were any, it was found out that from season 2016 to 2018 the team names were not available in the dataset an the same were added with the help of Microsoft Excel as seen in figure 7.



	season_x	name	position	team_x	assists	bonus	bps
0	2016-17	Aaron Cresswell	DEF	West Ham United	0	0	0
1	2016-17	Aaron Lennon	MID	Everton	0	0	6
2	2016-17	Aaron Ramsey	MID	Arsenal	0	0	5
3	2016-17	Abdoulaye DoucourV©	MID	Watford	0	0	0
4	2016-17	Adam Forshaw	MID	Middlesbrough	0	0	3
5	2016-17	Adam Lallana	MID	Liverpool	1	2	33
6	2016-17	AdriV°n San Miguel del Castillo	GK	West Ham United	0	0	16
7	2016-17	Alex Iwobi	MID	Arsenal	1	0	12
8	2016-17	Alex McCarthy	GK	Southampton	0	0	0
9	2016-17	Alex Oxlade-Chamberlain	MID	Arsenal	0	0	23
10	2016-17	Andreas Pereira	MID	Manchester United	0	0	0
11	2016-17	Andrew Robertson	DEF	Hull City	0	0	14
12	2016-17	Andre Gray	FWD	Burnley	0	0	-3
13	2016-17	Andros Townsend	MID	Crystal Palace	0	0	17
14	2016-17	Andy Carroll	FWD	West Ham United	0	0	7
15	2016-17	Angelo Ogbonna	DEF	West Ham United	0	0	0
16	2016-17	Anthony Martial	FWD	Manchester United	2	3	35
17	2016-17	Arthur Masuaku	DEF	West Ham United	0	0	19
18	2016-17	Ashley Barnes	FWD	Burnley	0	0	0
19	2016-17	Ashley Young	DEF	Manchester United	0	0	0
20	2016-17	Bamidele Alli	MID	West Ham United	0	0	6
21	2016-17	Benjamin Chilwell	DEF	Burnley	0	0	0
22	2016-17	Ben Davies	DEF	Manchester United	0	0	0
23	2016-17	Ben Foster	GK	Tottenham Hotspurs	0	3	32
24	2016-17	Ben Gibson	DEF	Leicester City	0	0	6
25	2016-17	Ben Mee	DEF	Burnley	0	0	15
26	2016-17	Branislav Ivanovic	DEF	Chelsea	0	0	15
27	2016-17	Callum Wilson	FWD	Bournemouth	0	0	-1
28	2016-17	Calum Chambers	DEF	Middlesbrough	0	0	26

fig 7 Team names added in Excel.

The season's name in the FPL dataset were mentioned as 2016-17, 2017—18 which was changed to 2016, 2017 respectively to increase the readability and to ease of use the code snippet is shown in figure 8.

```
In [421]: #Changes made to the season to increase readability and to make use case easier
          import pandas as pd

          # Sample DataFrame
          sample = {'season': ['2016-17', '2017-18', '2018-19', '2019-20', '2020-21']}
          df = pd.DataFrame(data)

          # Extract the year from the 'season' column
          data['season'] = data['season_x'].str.split('-').str[0]

          # Display the DataFrame

          data = data.drop(['season_x'],axis = 1)
```

fig 8 Transforming the season column to increase readability and use case.

2.2 Data Training and Feature Selection

After doing the introductory exploratory data analysis (EDA), the dataset was split into training and testing dataset, where the season from 2016 to 2020 was for training the dataset and season 2021 as testing dataset as seen in figure 9.

```
In [447]: train_data = data[data.season != '2021']
          test_data = data[data.season == '2021']
```

fig 9 Training and testing dataset

Before training the models, a feature importance technique was applied called Permutation feature importance, which gives the important features for building our models. The snippet of the code and the graph can be seen in the figure 10.

```
In [436]: from sklearn.ensemble import RandomForestClassifier
          import matplotlib.pyplot as plt

          # Initialize and fit a Random Forest model
          forest_model = RandomForestClassifier(n_estimators=100, random_state=42)
          forest_model.fit(X_train, y_train)

          # Get feature importances
          feature_importances = forest_model.feature_importances_

          # Sort features by importance
          sorted_indices = feature_importances.argsort()[::-1]

          # Plot feature importances
          plt.figure(figsize=(10, 6))
          plt.bar(range(X_train.shape[1]), feature_importances[sorted_indices])
          plt.xticks(range(X_train.shape[1]), X_train.columns[sorted_indices], rotation=90)
          plt.title("Feature Importance")
          plt.show()
```

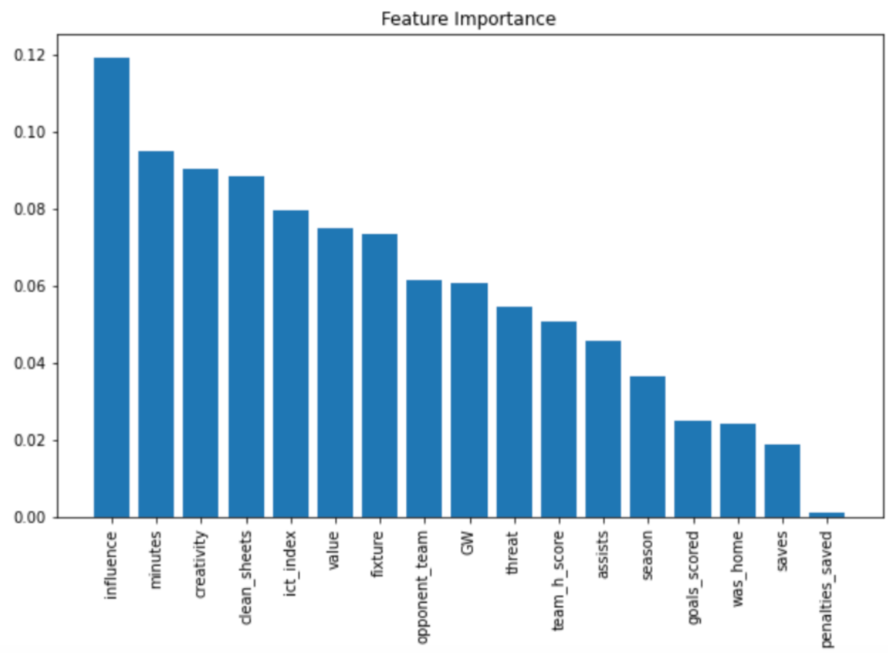


fig 10 Permutation feature importance

Based on the result of feature importance, the important features were included and rest of the features were removed before training of the model.

3. Modeling

Both the experiment involves building three models: Linear Regression, Random Forest and XGBoost. At first snippets of first experiment model building can be seen then after merging the xG and xA stats second experiment snippets are shown below:

Linear Regression (First Experiment)

After selecting the features based on feature importance, the models were build based only on these features for both the experiments in all the three models.

```
: from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model
LR_model = LinearRegression()


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','was_home','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity','season']
X_trainLR = train_data[features]  # Features for training
y_trainLR = train_data['total_points']  # Target variable for training
X_testLR = test_data[features]  # Features for testing
y_testLR = test_data['total_points']  # Target variable for testing

# Train the benchmark model
LR_model.fit(X_trainLR, y_trainLR)

# Predict using the benchmark model
LR_prediction = LR_model.predict(X_testLR)


# Calculate Mean Squared Error for the benchmark model
LR_mse = mean_squared_error(y_testLR, LR_prediction)

# Calculate Root Mean Squared Error (RMSE)
LR_rmse = np.sqrt(LR_mse)
#R square value for Linear Regression
from sklearn.metrics import mean_squared_error, r2_score
r2LR = r2_score(y_testLR, LR_prediction)

# Calculate Mean Absolute Error
maeLR = mean_absolute_error(y_testLR, LR_prediction)
print(f"Mean Absolute Error:", maeLR)
print("Linear Regression Mean Squared Error:", LR_mse)
print("Linear Regression Root Mean Squared Error:", LR_rmse)
print("Linear Regression R-Square value:", r2LR)
```

fig 11 Linear Regression (Exp 1)

Random Forest (First Experiment)

```
: from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model
LR_model = LinearRegression()


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','was_home','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity','season']
X_trainLR = train_data[features]  # Features for training
y_trainLR = train_data['total_points']  # Target variable for training
X_testLR = test_data[features]  # Features for testing
y_testLR = test_data['total_points']  # Target variable for testing

# Train the benchmark model
LR_model.fit(X_trainLR, y_trainLR)

# Predict using the benchmark model
LR_prediction = LR_model.predict(X_testLR)


# Calculate Mean Squared Error for the benchmark model
LR_mse = mean_squared_error(y_testLR, LR_prediction)

# Calculate Root Mean Squared Error (RMSE)
LR_rmse = np.sqrt(LR_mse)
#R square value for Linear Regression
from sklearn.metrics import mean_squared_error, r2_score
r2LR = r2_score(y_testLR, LR_prediction)

# Calculate Mean Absolute Error
maeLR = mean_absolute_error(y_testLR, LR_prediction)
print(f"Mean Absolute Error:", maeLR)
print("Linear Regression Mean Squared Error:", LR_mse)
print("Linear Regression Root Mean Squared Error:", LR_rmse)
print("Linear Regression R-Square value:", r2LR)
```

fig 12 Random Forest (Exp 1)

XGBoost (First Experiment)

XGBoost

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity']
X_train_XG = train_data[features]  # Features for training
y_train_XG = train_data['total_points']  # Target variable for training
X_test_XG = test_data[features]  # Features for testing
y_test_XG = test_data['total_points']  # Target variable for testing
# Convert categorical 'season' column to numerical using one-hot encoding
# X = pd.get_dummies(X, columns=['season'], drop_first=True)

# X = pd.get_dummies(X, columns=['was_home'], drop_first=True)


# Create an XGBoost model
model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, objective='reg:squarederror', random_state=42)
# You can adjust parameters as needed

# Train the model on the training data
model.fit(X_train_XG, y_train_XG)

# Predict on the test data
y_predXG = model.predict(X_test_XG)
```

fig 13 XGBoost (Exp 1)

Linear Regression (Second Experiment)

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
# Initialize the Linear Regression model
LR_model = LinearRegression()


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','was_home','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity','season', 'xA',
           'xG']
X_trainLR = train_data[features]  # Features for training
y_trainLR = train_data['total_points']  # Target variable for training
X_testLR = test_data[features]  # Features for testing
y_testLR = test_data['total_points']  # Target variable for testing

# Train the benchmark model
LR_model.fit(X_trainLR, y_trainLR)

# Predict using the benchmark model
LR_prediction = LR_model.predict(X_testLR)


# Calculate Mean Squared Error for the benchmark model
LR_mse = mean_squared_error(y_testLR, LR_prediction)

# Calculate Root Mean Squared Error (RMSE)
LR_rmse = np.sqrt(LR_mse)
#R square value for Linear Regression
from sklearn.metrics import mean_squared_error, r2_score
r2LR = r2_score(y_testLR, LR_prediction)

# Calculate Mean Absolute Error
maeLR = mean_absolute_error(y_testLR, LR_prediction)
print(f"Mean Absolute Error:", maeLR)
print("Linear Regression Mean Squared Error:", LR_mse)
print("Linear Regression Root Mean Squared Error:", LR_rmse)
print("Linear Regression R-Square value:", r2LR)
```

fig 14 Linear Regression (Exp 2)

Random Forest (Second Experiment)

```
: import pandas as pd
from sklearn.ensemble import RandomForestRegressor  # For regression tasks


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','was_home','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity','season',
           'xG', 'xA']
X_train_RF = train_data[features]  # Features for training
y_train_RF = train_data['total_points']  # Target variable for training
X_test_RF = test_data[features]  # Features for testing
y_test_RF = test_data['total_points']  # Target variable for testing

# Create a new RandomForest model
model = RandomForestRegressor(n_estimators=100, random_state=42)  # You can adjust parameters as needed

# Train the model on the training data
model.fit(X_train_RF, y_train_RF)

# Predict on the test data
y_predRF = model.predict(X_test_RF)
```

fig 15 Random Forest (Exp 2)

XGBoost (Second Experiment)

```
In [159]: import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor


# Select relevant features
features = ['minutes', 'goals_scored', 'assists', 'clean_sheets','GW','value','threat','team_h_score'
           , 'saves','penalties_saved','opponent_team','influence','ict_index','fixture','creativity', 'xG' , 'xA']
X_train_XG = train_data[features]  # Features for training
y_train_XG = train_data['total_points']  # Target variable for training
X_test_XG = test_data[features]  # Features for testing
y_test_XG = test_data['total_points']  # Target variable for testing
# Convert categorical 'season' column to numerical using one-hot encoding
# X = pd.get_dummies(X, columns=['season'], drop_first=True)

# X = pd.get_dummies(X, columns=['was_home'], drop_first=True)


# Create an XGBoost model
model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, objective='reg:squarederror', random_state=42)
# You can adjust parameters as needed

# Train the model on the training data
model.fit(X_train_XG, y_train_XG)

# Predict on the test data
y_predXG = model.predict(X_test_XG)
```

fig 16 Random Forest (Exp 2)

After building the model the best model was selected for prediction of points, which was Random Forest of first experiment. The below mentioned code snippet (figure 17) was used to show the best 11 players based on the predicted points. The user will be asked to enter the game week and the name of the English Premier League club for which they required the best performing players.

```python
# Prompt the user to input the football season
selected_GW = int(input("Enter the Game Week (between 1-38): "))

# Filter the DataFrame for the selected season (assuming you have a DataFrame named data_test)
selected_season_df = data_test[data_test['GW'] == selected_GW]

# Prompt the user to input the team name
selected_team = input("Enter the Team Name: ")

# Filter the DataFrame further for the selected team
selected_season_df = selected_season_df[selected_season_df['team'] == selected_team]

# Sort the DataFrame by predicted_total_points in descending order
selected_season_df = selected_season_df.sort_values(by='predictions', ascending=False)

# Initialize an empty dictionary to store selected players and their predicted points
selected_players = {}

# Loop through the sorted DataFrame to select unique players
for index, row in selected_season_df.iterrows():
    player_name = row['name']
    predicted_points = row['predictions']
    gw = selected_GW
    if player_name not in selected_players:
        selected_players[player_name] = {}
    selected_players[player_name][gw] = predicted_points
    if len(selected_players) == 11:  # Stop when 11 unique players are selected
        break

# Print the selected players along with their predicted points
print("Top 11 unique players from", selected_team, "with maximum predicted points for Game Week
for player, gw_points in selected_players.items():
    print(player)
    for gw, points in gw_points.items():
        print("- Predicted Points:", points, "- GW:", gw)
```

```
Enter the Game Week (between 1-38): 25
Enter the Team Name: Man City
Top 11 unique players from Man City with maximum predicted points for Game Week 25
Raheem Sterling
- Predicted Points: 18.75 - GW: 25
Ederson Santana de Moraes
- Predicted Points: 8.07 - GW: 25
Phil Foden
- Predicted Points: 8.01 - GW: 25
Kyle Walker
- Predicted Points: 6.89 - GW: 25
Rúben Santos Gato Alves Dias
- Predicted Points: 6.64 - GW: 25
Nathan Aké
- Predicted Points: 5.69 - GW: 25
Fernando Luiz Rosa
- Predicted Points: 4.04 - GW: 25
Ilkay Gündogan
- Predicted Points: 3.62 - GW: 25
Oleksandr Zinchenko
- Predicted Points: 3.34 - GW: 25
Bernardo Mota Veiga de Carvalho e Silva
- Predicted Points: 2.85 - GW: 25
Riyad Mahrez
```

fig 17 code for best players in a team for a particular game week