HSN201 : ADVANCED STATISTICS

INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

DEPARTMENT OF HUMANITIES & SOCIAL SCIENCES

Statistics Project

Predicting game outcomes in Premier League on the basis of FIFA Player Ratings

Authors:
Aniket, Madhav Mukund, Vedant Dwivedi

Enrollment No:
21322005,21322019,21322029

Course Supervisor:
Dr. Abhishek Samantray

November 7, 2022

Contents

1 Introduction **2**
1.1 Aims and Objectives . . . 2
1.2 Literature Review . . . 2
1.3 Model to be used . . . 2

2 Code and Modelling **3**
2.1 Data Scraping . . . 3
2.2 Modelling . . . 10
2.2.1 Final Dataframe . . . 10
2.2.2 Linear Regression . . . 12

3 Results **13**

4 References **13**

1 Introduction

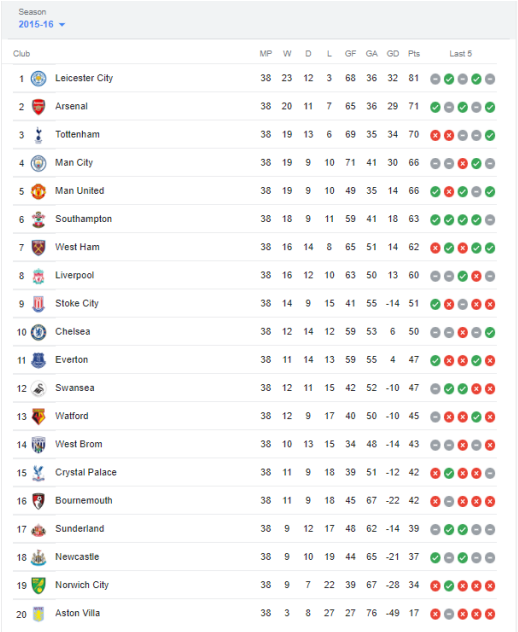
Every year, EA Sports (a popular gaming franchise) releases a new rendition of its FIFA series, and every year more than 24 million other people flock to buy it. In the game, every player is assigned a rating between 0 and 100 based on their performances in the previous season. How specifically the ratings are calculated is somewhat opaque, but it involves some combination of performance statistics and subjective scout reports that are then reviewed by a team of editors at EA. In this project we try and correlate these ratings with actual game results.

1.1 Aims and Objectives

- Analyse Premier league matches for the year 2021-22.
- Compare FIFA ratings for the players to the match analysis[1]
- Use the results to predict the Final Premier league table[2]

1.2 Literature Review

The practice of predicting football matches is used extensively in betting and other odd making activities and even by teams and league statisticians themselves, to improve and manage available resources. Such modelling, however, is done using in game statistics and other advanced stats, which try to measure and compare different unnoticed variables between players.Such models do extensive calculations on all types of statistics, yet they aren't very successful predictors still due to several unexplained factors still being in play. A very notorious example of inability to predict came forward in the 2015-16 season of the Premier League, where Leicester City, a team predicted by Forbes' writers and generally all outlets to be at the 17-18th position, topped the league. The odds for that happening were 5000-1 or 0.02% at the start of the league.



English Premier League 2015-16 Projections

	GP	W	D	L	GF	GA	GD	Pts
Manchester City	38	24	8	6	81	33	+48	80
Chelsea	38	24	8	6	78	32	+46	80
Arsenal	38	23	9	6	70	30	+40	78
Manchester United	38	21	10	7	62	30	+32	73
Liverpool	38	18	9	11	65	46	+19	63
Tottenham Hotspur	38	15	10	13	54	48	+6	55
Southampton	38	15	10	13	53	50	+3	55
Everton	38	14	10	14	52	50	+2	52
Stoke City	38	14	10	14	50	48	+2	52
Crystal Palace	38	13	10	15	46	52	-6	49
Newcastle United	38	12	10	16	48	57	-9	46
Aston Villa	38	12	10	16	44	54	-10	46
West Ham United	38	12	10	16	46	57	-11	46
Swansea City	38	11	10	17	41	55	-14	43
Sunderland	38	10	10	18	40	56	-16	40
West Bromwich Albion	38	9	10	19	37	57	-20	37
Watford	38	9	9	20	39	64	-25	36
Bournemouth	38	8	10	20	36	62	-26	34
Leicester City	38	8	9	21	36	64	-28	33
Norwich City	38	7	9	22	32	65	-33	30

SOCCERMETRICS RESEARCH

Our model, instead of relying on playing data relies on FIFA ratings, which are released 2 months into the season by EA Sports with it's running Fifa game franchise, comprising of player's past performance and expectation level.

1.3 Model to be used

We have used 'Linear Regression' as our prediction model for our dataset. Linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables. In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data.
The assumptions in a multiple linear regression model are :

- There is a linear relationship between the dependent variables and the independent variables

- The independent variables are not too highly correlated with each other
- y observations are selected independently and randomly from the population
- Residuals should be normally distributed with a mean of 0 and variance

Our project has been coded on Python and uses the following libraries :

```
import numpy as np
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup as soup
from urllib.request import urlopen as uReq
import pandas as pd
import unidecode
import seaborn as sns
from scipy import stats
import math
import warnings
```

Numpy NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

matplotlib Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications.

beautifulsoup Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping.

Urlopen It is used to fetch URLs (Uniform Resource Locators). It uses the urlopen function and is able to fetch URLs using a variety of different protocols.

pandas pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Unidecode Unidecode produces better results than simply stripping accents from characters (which can be done in Python with built-in functions). It is based on hand-tuned character mappings that for example also contain ASCII approximations for symbols and non-Latin alphabets.

Seaborn Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Scipy SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

2 Code and Modelling

2.1 Data Scraping

The data required to setup and feed the model is as follows :

- The 380 fixtures of the Premier League in the 2021-22 Season
- The Squad names and home ground data
- The player ratings for each player from Fifa 22

To collect data set for all of the 380 matches, we scraped data from the official Premier League website (https://premierleague.com/results/), while the player rating data set was scraped from a FIFA ratings index (https://www.fifaindex.com/) using Python and it's libraries.

The code for the web scraping is as follows :

```
# set sns color code
sns.set(color_codes=True)

# suprress warnings to make the notebook more readable
warnings.filterwarnings("ignore")
def remove_special_characters(name):
    """This function removes numbers, apostrophes, and addition signs from a
    ↪  string and returns the cleaned string.
    This was specifically written with the format of name strings from the
↪   premier league match summaries in mind.
    Strings of the form "john doe 90 + 4' " are returned as "john doe".

    :param str name: The full string of text containing the player name.
    :return: A string containing the name with the unnecessary characters
↪   stripped.
    :rtype: str

    """
    for c in ['0','1','2','3','4','5','6','7','8','9','+',"'"]:
        name = name.replace(c,'')

    # remove trailing spaces
    name.strip()

    return name



def remove_team_code(name):
    """This function removes team codes from a string containing names+codes.
    The function is limited in that it only does this for teams that were
↪   involved in the premier league
    in the 2017/18 and 2021/19 seasons.

    :param str name: A team name of the form name+code.
    :return: A string containing the team name with the codes removed i.e
↪   BurnleyBUR returns Burnley.
    :rtype: str

    """
    # team codes
    team_codes =
    ↪  ["BUR","BOU","CRY","WBA","HUD","ARS","LIV","BHA","MUN","WAT","NEW",
    "CHE","SOU","MCI","SWA","AVL","TOT","LEI","WHU","EVE","WOL","CAR","FUL",
    "LEE","BRI","FUL","BRE","WAT","NOR"]

    # look for codes in name, if found, remove
    for c in team_codes:
        name = name.replace(c,'')

    return name
def remove_n(team_name):
    to_remove=["\n\n\n\n\n\n\n\n\n\n","\n\n\n"]
    for x in to_remove:
      team_name=team_name.replace(x,'')
      return team_name
```

These are functions designed to remove accents, numbers, spaces and other anomalies to standardize the dataset, as their sources were different. The other function removes team abbreviations from the data, as we identify teams by full names outside of fixtures and as home/away team in fixtures. These small adjustments make the code more readable, and easier to manage.

```
def extract_epl_lineups(team_container, home):
    """This function extract lineups information from the
    ↪  matchLineupTeamContainer class
    as defined on the the premier league results website.

    :param Tag team_container: Contains the matchLineupTeamContainer from the
↪   premier league website.
    :param bool home: A flag for whether team_conatiner contains the home team
↪   lineup or away team lineup
    :return:  A dictionary with the keys = [ "player1", ...., "player 14"], and
↪   the values are [{"name":"John Doe"}....].
             If a team plays with less than 14 players in a game, the remaining
↪   slots are left blank (= '').
    :rtype: dict

    """

    # generate keys for return dict, initialize player names
    if home== True:
        base_string = "home_player"
    else:
        base_string = "away_player"
    keys = [base_string+str(i+1) for i in range(16)]
    player_names = []

    # extract info from the container
    lineup_container = team_container.find_all("div",{"class":"info"})

    # extract player names
    for i in range(len(lineup_container)):
        if i < 11:
            player_names.append(unidecode.unidecode(remove_special_characters
            (lineup_container[i].div.text)))
        else:
            if lineup_container[i].div.div.text != '':
                player_names.append(unidecode.unidecode
                (remove_special_characters(lineup_container[i].div.text)))

    # create dictionary
    player_dict = dict(zip(keys,player_names))

    # if player12, player13, or player14 don't exits, create and and assign ''
    ↪  as value
    for i in range(12,17):
        if base_string+str(i) not in player_dict:
            player_dict[base_string+str(i)]= ""

    return player_dict
```

The code above extracts and forms a player dictionary from the 'matchlineupTeamcontainer' class at the Premier League's website for the 16 (11 playing and 5 reserves) players in every team. It handles the missing data as well.

```
def extract_epl_match_details(match_ids):
    """This function extracts match information for games from the premier
    ↪  league results website
    given their match ids.

    :param list match_ids: A list of match ids for the games you're interested
↪   in. Can be a list of one or empty.
```

```
    :return: A dictionary containing the match information for every match id.
↪   Each unique match
            match id is a key. The value is the corresponding match information
↪   which is a list
            five dictionaries the team names, score, result (encoded as 1 for a
↪   home team win,
            0 for a draw, -1 for an away team win), the home team lineup,
            and the away team line up. The dictionaries are formatted as
↪   follows:
            team_names = {"home_team":"Arsenal", "away_team":"Liverpool"}
            score = {"home_goals":2, "away_goals":3}
            result = {"result": 1}
            home_team = {"home_player1":"Petr Cech", ....., "home_player14":
↪   "Alex Iwobi"}
            away_team = {"away_player1":"Loris Karius", ....,
↪   "away_player14":""}
    :rtype: dict

    """

    # where the match data is being scraped from
    epl_url = "https://www.premierleague.com/match/"

    # initialize a dictionary to store the results
    data = dict()

    # this loop extracts the data through each game in turn
    for match_id in match_ids:

        # url for each math
        match_url = epl_url + str(match_id)

        # download page, call and then close client
        uClient = uReq(match_url)
        results_page = uClient.read()
        uClient.close()

        # extract html
        results_soup = soup(results_page, "html.parser")

        # extract home team name container
        home_team_container = results_soup.findAll("div", {"class":"team home"})

        # extract away team name container
        away_team_container = results_soup.findAll("div", {"class":"team away"})

        # extract score container
        score_container = results_soup.findAll("div",
        ↪  {"class":"matchScoreContainer"})

         # extract lineup containers
        team_lineups_container = results_soup.findAll("div",
        ↪  {"class":"matchLineupTeamContainer"})

        # extract team names from their containers
        team_names = {"home_team":
        ↪  remove_team_code(home_team_container[0].text),\
                    "away_team":remove_team_code(away_team_container[0].text)}

        # extract score from score container
        score = {"home_goals": int(((score_container[0].text).split("-"))[0]),\
```

```
                "away_goals":int(((score_container[0].text).split("-"))[1]) }
        # use score to determine match outcome, Win for home team = 1, Win for
        ↪  away team = -1, draw = 0
        result = {"result":  1 if score["home_goals"] > score["away_goals"]
        ↪  else\
                 -1 if score["away_goals"] > score["home_goals"] else 0 }

        # extract lineups for the home and away team from their containers, this
        ↪  includes substitutes who made appearances
        home_team = extract_epl_lineups(team_lineups_container[0], home = True)
        away_team = extract_epl_lineups(team_lineups_container[1], home = False)

        # append to data dictionary
        data[match_id] = [team_names, score, result, home_team, away_team]

    return data
```

Each match that occurs in the premier league is orderly assigned a match_id, which we have used as that fixture's identifier as well. The data is called from the website through urlib's urlopen command and the names of all the playing players (starting and substitute) and score, alongwith team names is recorded in a dictionary. Hence, we now have a dataset of active players and fixtures and only need to assign them their FIFA ratings.

```
# main functions used for finding fifa ratings

def last_name(name):
    """Extract the last name for a player given their full name.

    :param str name: A string of the full name of the player.
    :return: Returns just the last name of the player. If their full name
            is just one name - "Ronaldinho" for example - return what is passed.
    :rtype: str

    """
    name = unidecode.unidecode(name)
    names = name.split()
    last_name = names[len(names)-1]

    return last_name
def first_name(name) :
    name = unidecode.unidecode(name)
    names = name.split()
    first_name = names[len(names)-2]
    return first_name
def extract_fifa_rating(data_main, data_auxillary, name, club):
    """This function extracts the fifa rating for a player.

    :param DataFrame data_main: A master data store; this is the data from
 ↪  kaggle.
    :param DataFrame data_auxillary: A supplementary dataframe that conatains
 ↪  data missing in data_main.
    :param str name: A string containing the name of the player missing.
    :param str club: A string containing the the player's club.
    :return: The base fifa rating (type int) if found, else returns np.NaN.
    :rtype: int

    """
    rating = 0

    # check the main dataset to see if the full name matches
    if data_main[data_main["NAME"]== name]["RATING"].empty == False:
        rating = data_main[data_main["NAME"]== name]["RATING"].min()
```

```
    # check if the last name and club are enough to match
    elif data_main[data_main["LAST_NAME"]==
    ↪  last_name(name)][fifa_data["CLUB"]==club]["RATING"].empty == False:
        rating = data_main[data_main["LAST_NAME"]==
        ↪  last_name(name)][fifa_data["CLUB"]==club]["RATING"].min()
    elif data_main[data_main["FIRST_NAME"]==
    ↪  first_name(name)][fifa_data["CLUB"]==club]["RATING"].empty == False:
        rating = data_main[data_main["FIRST_NAME"]==
        ↪  first_name(name)][fifa_data["CLUB"]==club]["RATING"].min()

    elif data_auxillary[data_auxillary["NAME"] ==
    ↪  unidecode.unidecode(name)]["RATING"].empty == False:
        rating = data_auxillary[data_auxillary["NAME"] ==
        ↪  unidecode.unidecode(name)]["RATING"].min()
    else:
        rating = np.NaN

    return rating
```

To avoid all discrepancies between datasets (due to different collecting bodies), we had to compare first, last and team names before assigning FIFA ratings to players. The code defines first and last names, and uses unidecode to remove accents, anomalies etc. We created a dataset already of player ratings from the FIFA index website, hence the algorithm compares names and returns the rating.

```
# functions to format the data (list of dictionaries) appropriate as dataframe

def convert_to_dataframe(data, match_ids):
    """This function converts a list called data (see documentation for
    extract_epl_match_details for format) and match_ids to create dataframe of
↪   33 columns.
    The column names are enumerated in the second line of the function.

    :param dict data: The data for all the games fromatted as a dictionary.
    :param list match_ids: A list of all the match ids. These are the keys used
↪   to access the data parameter.
    :return: DataFrame containing 33 columns with every row corresponding to a
↪   single game. The indexes correspond
            to match ids.
    :rtype: DataFrame

    """

    # index and columns used to initialize return dataframe
    index = [0]
    columns = ["home_team", "away_team", "home_goals", "away_goals","result",
              "home_player1","home_player2","home_player3","home_player4",
              ↪  "home_player5","home_player6","home_player7","home_player8",
              "home_player9","home_player10","home_player11","home_player12",
              "home_player13","home_player14","home_player15",
              "away_player1","away_player2","away_player3","away_player4",
              "away_player5","away_player6","away_player7",\
               "away_player8","away_player9","away_player10","away_player11",
               "away_player12","away_player13","away_player14","away_player15"]

    # initialize dataframe with one empty row
    data_df = pd.DataFrame(index=index,columns=columns)

    # append match info for each match in turn to return dataframe
    for match_id in match_ids:
        temp_df = pd.DataFrame.from_dict(data[match_id][0],orient="index")
```

```
        for i in [1,2,3,4]:
            temp_df=
             ↪  pd.concat([temp_df,pd.DataFrame.from_dict(data[match_id][i],
            orient"index")])
        temp_df = temp_df.T
        data_df = data_df.append(temp_df)

    # remove the empty first row
    data_df = data_df.iloc[1:]

    # make the indices equal to the match ids
    data_df.index = match_ids

    return data_df

##### Extract Match Info #####

# there are 380 matches played in a season, there are 20 teams and each team
 ↪  plays every other team twice.
num_matches_2021 = 380 #380

# list of match IDs for the games played last season
match_ids_2021 = [66342+i for i in range(num_matches_2021)]


# extract data for each season
data_2021 = extract_epl_match_details(match_ids_2021)

# convert data from dictionary to dataframe
df_2021 = convert_to_dataframe(data_2021,match_ids_2021)


##### Compile Player Names and Ratings #####

# import master FIFA22 file, includes just fifa ratings
fifa_data =
 ↪  pd.read_csv("https://raw.githubusercontent.com/Madhav-Mukund/myfiles/main/Check2.csv")


# remove special characters from the name column (accented e's etc)
fifa_data["NAME"] = fifa_data["NAME"].transform(unidecode.unidecode)

# add last name column to dataframe to aid querying
fifa_data["LAST_NAME"] = fifa_data["NAME"].transform(last_name)
fifa_data["FIRST_NAME"] = fifa_data["NAME"].transform(first_name)

# import missing fifa data, this is the  missing data file I created
missing_fifa_data =
 ↪  pd.read_csv("https://raw.githubusercontent.com/Madhav-Mukund/myfiles/main/Check2.csv")

# initiaize dictionary to hold player data
name_team_rating_dict = dict()

# extract player name, club, and rating for every player who played any minutes
 ↪  at all in the epl in the last two seasons
for id in (match_ids_2021):
    for i in range(1,16):
        if data[id][3]["home_player"+str(i)] != '':
            name = (data[id][3]["home_player"+str(i)]).strip()
            club2 = remove_team_code(data[id][0]["home_team"])
            club=club2.replace('\n','')
```

```
            rating = extract_fifa_rating(fifa_data, missing_fifa_data, name,
             ↪  club)
            name_team_rating_dict[name] = [club,rating]
    for j in range(1,16):
        if data[id][4]["away_player"+str(j)] != '':
            name = (data[id][4]["away_player"+str(j)]).strip()
            club2 = remove_team_code(data[id][0]["away_team"])
            club=club2.replace('\n','')
            rating = extract_fifa_rating(fifa_data, missing_fifa_data, name,
             ↪  club)
            name_team_rating_dict[name] = [club,rating]

# convert player data dictionary to dataframe
player_data = pd.DataFrame(name_team_rating_dict).T
player_data.columns = ["CLUB", "RATING"]
player_data["CLUB"] = player_data["CLUB"].str.replace("\n", "")
df_2021["home_team"]= df_2021["home_team"]

##### Export Data #####

# export dataframes as csv files
df_2021.to_csv("epl_results_2021", sep = ',')

player_data.to_csv('epl_player_ratings', sep=',')
```

We convert all of our previously assembled dictionaries into data frames, a more manageable and operable data structure. The player names and ratings are called from master file, and the missing data is accounted for by a patch. This data is appended into a dictionary which is then converted to a dataframe.
Running the above segment completely takes around 15-20 minutes of processing time to download all the webpages and scrape the data. Also then to process the same for ratings for all the players in the complete database. For ease of access and saving time we saved the results from the above process and added them to a public Github Repository and will be using the same for further processing.

Finally, the process of data collection is over and we have 2 .csv dataframes, containing fixture results and player ratings database.

2.2 Modelling

Now that we have our dataframes of player ratings and result, we can initiate our model. We will be using average of all the players playing in that match, so we have yet another dataset to generate.

2.2.1 Final Dataframe

After the following code, we will have our final dataset containing average ratings, score, team names, player names and result.

```
 # these functions compute the rating averages for teams, the differences, and
 ↪  append them to the match data dataframes

def compute_team_average(names ,ratings ):
    """This function computes the average rating for a list of players. It
    ↪  accounts for any empty
    strings (i.e. '') in the list.

    :param: list names : A list of names of the players.
    :param: DataFrame ratings : An appropriately formatted dataframe containing
↪   player ratings.
    :return: The average rating of the players passed in.
    :rtype: float

    """
```

```
    #initialize return variable
    average = 0
    count = 0

    # only count players who actually appeared, ignore empty places and NaN
     ↪  values
    for name in names:
        if type(name) == str:
            average = average +
             ↪  ratings[ratings['NAME']==name.strip()]["RATING"].min()
            count = count + 1

    # average contains the sum of all player ratings, thus divide
    average = average/count

    return average


def add_averages_to_df(match_data,ratings):
    """This function appends the following three coloumns to the match data
     ↪  dataframe:
    home_team_rating, away_team_rating, rating_diff. It returns this modified
 ↪  dataframe.

    :param: DataFrame match_data: This dataframe must contain the lineups for
 ↪  each game in the set.
    :param: DataFrame ratings: This dataframe must contain player ratings.
    :return: Appends three columns to the match_data dataframe: one for the
 ↪  average rating for the home team,
            one for the average rating for the away team, and a column for the
 ↪  average difference.
    :rtype: DataFrame

    """

    # initialize lists to store the home team average rating, away team average
     ↪  rating, and average team difference
    home_team_rating = []
    away_team_rating = []
    average_rating_diff =[]

    # populate the above list by match
    for i in match_data.index:

         ↪  home_team_rating.append(compute_team_average(list(match_data.iloc[i][7:22]),
         ↪  ratings))

         ↪  away_team_rating.append(compute_team_average(list(match_data.iloc[i][23:35]),
         ↪  ratings))
        average_rating_diff.append(home_team_rating[i] - away_team_rating[i])

    # turn the three lists into series
    home_average_series = pd.Series(home_team_rating)
    away_average_series = pd.Series(away_team_rating)
    average_rating_diff = pd.Series(average_rating_diff)

    # append the series as columns into the dataframe
    match_data["home_average_rating"] = home_average_series
    match_data["away_average_rating"] = away_average_series
    match_data["rating_diff"] = average_rating_diff
```

```
    return match_data

df_2021.rename(columns = {'Unnamed: 0.1':'Match_IDs'}, inplace = True)
df_2021.rename(columns = {'Unnamed: 0':'Serial_No'}, inplace = True)

# function calls to compute averages, find differences

rating_games_2021 = add_averages_to_df(df_2021,player_data)
```

2.2.2 Linear Regression

Now, that we have our final, single dataset containing everything required for the regression, we can call said function and feed the values. We take 75

```
 #Finding a general model for all teams
from sklearn import linear_model
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import random
complete_list=[]
for i in range(0,380):
  complete_list.append(i)
complete_list=np.array(complete_list)
rand_list=random.sample(range(380), 285)# 75% data for training the model
res_list=np.setdiff1d(complete_list, rand_list)

X=rating_games_2021[['home_goals','away_goals','home_average_rating','away_average_rating'
Y=rating_games_2021['result']
regr = linear_model.LinearRegression()

X_train = pd.DataFrame(columns =
 ↪  ['home_goals','away_goals','home_average_rating','away_average_rating'])
Y_train=[]
for i in rand_list:
  X_train = X_train.append(rating_games_2021.loc[i], ignore_index = True)
  X_train =
   ↪  X_train[['home_goals','away_goals','home_average_rating','away_average_rating']]
  Y_train.append(rating_games_2021.loc[i,'result'])
regr.fit(X_train, Y_train)

X_test = pd.DataFrame(columns =
 ↪  ['home_goals','away_goals','home_average_rating','away_average_rating'])
Y_test=[]
res_list=res_list.tolist()
for i in res_list:
  X_test = X_test.append(rating_games_2021.loc[i], ignore_index = True)
  X_test =
   ↪  X_test[['home_goals','away_goals','home_average_rating','away_average_rating']]
  Y_test.append(rating_games_2021.loc[i,'result'])
Y_pred=regr.predict(X_test)

print(regr.score(X_test, Y_test))

print(regr.coef_)


def home_team_point_counter(Y_pred_home_team):
  #initializing counter variable
  home_team_points =0
  for i in Y_pred_home_team:
```

```
    #winning factor is a number lying between -1 to 1 representing loss and win
    ↪  respectively.
    #Our model predicts such a factor for each game for the home team
    #assuming team wins match if Winning Factor>0.33
    if i > 0.33:
      home_team_points+=3
    #assuming team loses if Winning factor < -0.33
    elif i < -0.33:
      home_team_points+=0
    #draw in all other cases
    else:
      home_team_points+=1

  return home_team_points
```

3 Results

The model predicts a 71.17% accuracy on the test data. It suggests strong indication towards there existing a linear relationship between the match result and the parameters we have chosen (home team goals, away team goals, home team rating, away team rating)

Output of Linear regression model
0.7171361443585236

[0.33584074 -0.38694286 0.00371143 -0.00046501]

4 References

[1] FourFourTwo Staff. *FIFA 22: Player ratings explained*. https://www.fourfourtwo.com/features/fifa-22-player-ratings-explained-chemistry-fut-ultimate-team-pace-passing-speed-card. 2021.

[2] Wikipedia. *Premier League*. https://en.wikipedia.org/wiki/Premier_League.