CS403 Final Project Submission and Presentation

Joshua Thomas
12/10/2022
Professor Trotter

Project Description

For this project, I have decided to use the API-Football API. This API contains numerous amounts of data related to the beautiful game- "football" (soccer). As someone who loves soccer, I'm going to attempt to answer questions related to the Premier League from the 2012-2021 seasons with analyses using:

- 1) Inferential Statistics
- 2) Linear Regression Machine Learning
- 3) Logistic Regression Classification Machine Learning

Questions

1) Question 1: What is the confidence interval at the 95% level for the mean number of goals from the years 2012-2016 and 2017-2021?

2) Question 2: Predict the Goal Difference for a Team in the Premier League using Data from 2012-2021 using Linear Regression

3) Question 3: Predict Whether the Number of Wins for a Team in the Premier League is Greater than the Mean Number of Wins from 2012-2021 using Data from 2012-2021 using Logistic Regression Classification

Data Preparation

In [125…

```
# Import Statements needed for this Question
import statistics
import pandas as pd
import numpy as np
import requests
import json # Needed for parsing

url_team_keys = "https://api-football-v1.p.rapidapi.com/v3/teams"
url_stats = "https://api-football-v1.p.rapidapi.com/v3/teams/statistic"

headers = {
        "X-RapidAPI-Key": "b64524d640msh3d87870dba5544bp111162jsn3e16b1999c06",
        "X-RapidAPI-Host": "api-football-v1.p.rapidapi.com"
}

# Get Team Keys for each Premier League Team from 2012-2021

# 2010-2019 Team Keys for Each Team
querystring_12 = {"league":"39","season":"2012"}
querystring_13 = {"league":"39","season":"2013"}
querystring_14 = {"league":"39","season":"2014"}
querystring_15 = {"league":"39","season":"2015"}
querystring_16 = {"league":"39","season":"2016"}
querystring_17 = {"league":"39","season":"2017"}
querystring_18 = {"league":"39","season":"2018"}
querystring_19 = {"league":"39","season":"2019"}
querystring_20 = {"league":"39","season":"2020"}
querystring_21 = {"league":"39","season":"2021"}

response_12 = requests.request("GET", url_team_keys, headers=headers, params=querystring_12)
response_12 = response_12.text
dictionary_teamId_12 = json.loads(response_12)
dict_12 = dictionary_teamId_12.get('response')

response_13 = requests.request("GET", url_team_keys, headers=headers, params=querystring_13)
response_13 = response_13.text
dictionary_teamId_13 = json.loads(response_13)
dict_13 = dictionary_teamId_13.get('response')

response_14 = requests.request("GET", url_team_keys, headers=headers, params=querystring_14)
response_14 = response_14.text
dictionary_teamId_14 = json.loads(response_14)
dict_14 = dictionary_teamId_14.get('response')

response_15 = requests.request("GET", url_team_keys, headers=headers, params=querystring_15)
response_15 = response_15.text
dictionary_teamId_15 = json.loads(response_15)
dict_15 = dictionary_teamId_15.get('response')
```

```
response_16 = requests.request("GET", url_team_keys, headers=headers, params=querystring_16)
response_16 = response_16.text
dictionary_teamId_16 = json.loads(response_16)
dict_16 = dictionary_teamId_16.get('response')

response_17 = requests.request("GET", url_team_keys, headers=headers, params=querystring_17)
response_17 = response_17.text
dictionary_teamId_17 = json.loads(response_17)
dict_17 = dictionary_teamId_17.get('response')

response_18 = requests.request("GET", url_team_keys, headers=headers, params=querystring_18)
response_18 = response_18.text
dictionary_teamId_18 = json.loads(response_18)
dict_18 = dictionary_teamId_18.get('response')

response_19 = requests.request("GET", url_team_keys, headers=headers, params=querystring_19)
response_19 = response_19.text
dictionary_teamId_19 = json.loads(response_19)
dict_19 = dictionary_teamId_19.get('response')

response_20 = requests.request("GET", url_team_keys, headers=headers, params=querystring_20)
response_20 = response_20.text
dictionary_teamId_20 = json.loads(response_20)
dict_20 = dictionary_teamId_20.get('response')

response_21 = requests.request("GET", url_team_keys, headers=headers, params=querystring_21)
response_21 = response_21.text
dictionary_teamId_21 = json.loads(response_21)
dict_21 = dictionary_teamId_21.get('response')
```

In [156…

```
# Get Team Keys for each Premier League Team from 2012-2021
def get_id(dict):
    team = []
    for i in range(len(dict)):
        team_id = dict[i].get('team').get('id')
        team.append(team_id)
    return team

id_2012 = get_id(dict_12)
id_2013 = get_id(dict_13)
id_2014 = get_id(dict_14)
id_2015 = get_id(dict_15)
id_2016 = get_id(dict_16)
id_2017 = get_id(dict_17)
id_2018 = get_id(dict_18)
id_2019 = get_id(dict_19)
id_2020 = get_id(dict_20)
id_2021 = get_id(dict_21)

def get_responses(team_Id, season):
    responses = []
    for i in range(len(team_Id)):
        querystring_stats = {"league":"39","season":season,"team": str(team_Id[i])}
        response_stats = requests.request("GET", url_stats, headers=headers, params=querystring_stats)
        response_stats = response_stats.text
        dictionary_stats = json.loads(response_stats)
        dict_response = dictionary_stats.get('response')
        responses.append(dict_response)
    return responses
```

In [ ]:

```
responses_2012 = get_responses(id_2012, "2012")
responses_2012
```

In [ ]:

```
responses_2013 = get_responses(id_2013, "2013")
responses_2013
```

In [ ]:

```
responses_2014 = get_responses(id_2014, "2014")
responses_2014
```

In [ ]:

```
responses_2015 = get_responses(id_2015, "2015")
responses_2015
```

In [ ]:

```
responses_2016 = get_responses(id_2016, "2016")
responses_2016
```

In [ ]:

```
responses_2017 = get_responses(id_2017, "2017")
responses_2017
```

In [ ]:

```
responses_2018 = get_responses(id_2018, "2018")
responses_2018
```

In [ ]:

```
responses_2019 = get_responses(id_2019, "2019")
responses_2019
```

In [ ]:
```
responses_2020 = get_responses(id_2020, "2020")
responses_2020
```

In [ ]:
```
responses_2021 = get_responses(id_2021, "2021")
responses_2021
```

In [230…
```
# Function to get Goals For
def get_goals_for(responses):
    goals_for = []
    for i in range(len(responses)):
        if responses[i].get('goals').get('for').get('total').get('total') is not None:
            data = responses[i].get('goals').get('for').get('total').get('total')
            goals_for.append(data)
        else:
            print("Does not exist")
    return goals_for

# Function to get Goals Against
def get_goals_against(responses):
    goals_against = []
    for i in range(len(responses)):
        if responses[i].get('goals').get('against').get('total').get('total') is not None:
            data = responses[i].get('goals').get('against').get('total').get('total')
            goals_against.append(data)
        else:
            print("Does not exist")
    return goals_against

# Function to get Wins
def get_wins(responses):
    wins = []
    for i in range(len(responses)):
        if responses[i].get('fixtures').get('wins').get('total') is not None:
            data = responses[i].get('fixtures').get('wins').get('total')
            wins.append(data)
        else:
            print("Does not exist")
    return wins

# Function to get Draws
def get_draws(responses):
    draws = []
    for i in range(len(responses)):
        if responses[i].get('fixtures').get('draws').get('total') is not None:
            data = responses[i].get('fixtures').get('draws').get('total')
            draws.append(data)
        else:
            print("Does not exist")
    return draws

# Function to get losses
def get_losses(responses):
    losses = []
    for i in range(len(responses)):
        if responses[i].get('fixtures').get('loses').get('total')is not None:
            data = responses[i].get('fixtures').get('loses').get('total')
            losses.append(data)
        else:
            print("Does not exist")
    return losses

# Function to get season
def get_year(responses):
    years = []
    for i in range(len(responses)):
        if responses[i].get('league').get('season') is not None:
            data = responses[i].get('league').get('season')
            years.append(data)
        else:
            print("Does not exist")
    return years
```

In [231…
```
# Get Data for each year
goals_for = []
goals_against = []
wins = []
draws = []
losses = []
year = []

#2012
goals_for_2012 = get_goals_for(responses_2012)
goals_against_2012 = get_goals_against(responses_2012)
wins_2012 = get_wins(responses_2012)
draws_2012 = get_draws(responses_2012)
losses_2012 = get_losses(responses_2012)
year_2012 = get_year(responses_2012)
```

```
#2013
goals_for_2013 = get_goals_for(responses_2013)
goals_against_2013 = get_goals_against(responses_2013)
wins_2013 = get_wins(responses_2013)
draws_2013 = get_draws(responses_2013)
losses_2013 = get_losses(responses_2013)
year_2013 = get_year(responses_2013)

#2014
goals_for_2014 = get_goals_for(responses_2014)
goals_against_2014 = get_goals_against(responses_2014)
wins_2014 = get_wins(responses_2014)
draws_2014 = get_draws(responses_2014)
losses_2014 = get_losses(responses_2014)
year_2014 = get_year(responses_2014)

#2015
goals_for_2015 = get_goals_for(responses_2015)
goals_against_2015 = get_goals_against(responses_2015)
wins_2015 = get_wins(responses_2015)
draws_2015 = get_draws(responses_2015)
losses_2015 = get_losses(responses_2015)
year_2015 = get_year(responses_2015)

#2016
goals_for_2016 = get_goals_for(responses_2016)
goals_against_2016 = get_goals_against(responses_2016)
wins_2016 = get_wins(responses_2016)
draws_2016 = get_draws(responses_2016)
losses_2016 = get_losses(responses_2016)
year_2016 = get_year(responses_2016)

#2017
goals_for_2017 = get_goals_for(responses_2017)
goals_against_2017 = get_goals_against(responses_2017)
wins_2017 = get_wins(responses_2017)
draws_2017 = get_draws(responses_2017)
losses_2017 = get_losses(responses_2017)
year_2017 = get_year(responses_2017)

#2018
goals_for_2018 = get_goals_for(responses_2018)
goals_against_2018 = get_goals_against(responses_2018)
wins_2018 = get_wins(responses_2018)
draws_2018 = get_draws(responses_2018)
losses_2018 = get_losses(responses_2018)
year_2018 = get_year(responses_2018)

#2019
goals_for_2019 = get_goals_for(responses_2019)
goals_against_2019 = get_goals_against(responses_2019)
wins_2019 = get_wins(responses_2019)
draws_2019 = get_draws(responses_2019)
losses_2019 = get_losses(responses_2019)
year_2019 = get_year(responses_2019)

#2020
goals_for_2020 = get_goals_for(responses_2020)
goals_against_2020 = get_goals_against(responses_2020)
wins_2020 = get_wins(responses_2020)
draws_2020 = get_draws(responses_2020)
losses_2020 = get_losses(responses_2020)
year_2020 = get_year(responses_2020)

#2021
goals_for_2021 = get_goals_for(responses_2021)
goals_against_2021 = get_goals_against(responses_2021)
wins_2021 = get_wins(responses_2021)
draws_2021 = get_draws(responses_2021)
losses_2021 = get_losses(responses_2021)
year_2021 = get_year(responses_2021)
```

In [232…

```
# Concatenate Data and make DataFrame
goals_for = goals_for_2012 + goals_for_2013 + goals_for_2014 + goals_for_2015 + goals_for_2016 + goals_for_2017 +
goals_against = goals_against_2012 + goals_against_2013 + goals_against_2014 + goals_against_2015 + goals_against
wins = wins_2012 + wins_2013 + wins_2014 + wins_2015 + wins_2016 + wins_2017 + wins_2018 + wins_2019 + wins_2020
draws = draws_2012 + draws_2013 + draws_2014 + draws_2015 + draws_2016 + draws_2017 + draws_2018 + draws_2019 + d
losses = losses_2012 + losses_2013 + losses_2014 + losses_2015 + losses_2016 + losses_2017 + losses_2018 + losses
year = year_2012 + year_2013 + year_2014 + year_2015 + year_2016 + year_2017 + year_2018 + year_2019 + year_2020
```

In [233…

```
data = {'goals_for': goals_for, 'goals_against': goals_against, 'wins': wins, 'draws': draws, 'losses': losses, '
```

In [235…

```
df = pd.DataFrame(data)
```

In [239…

```
# Head of dataframe
df.head()
```

Out[239…

	goals_for	goals_against	wins	draws	losses	year
0	86	43	28	5	5	2012
1	45	68	11	8	19	2012
2	50	60	11	10	17	2012
3	71	43	16	13	9	2012
4	49	60	9	14	15	2012

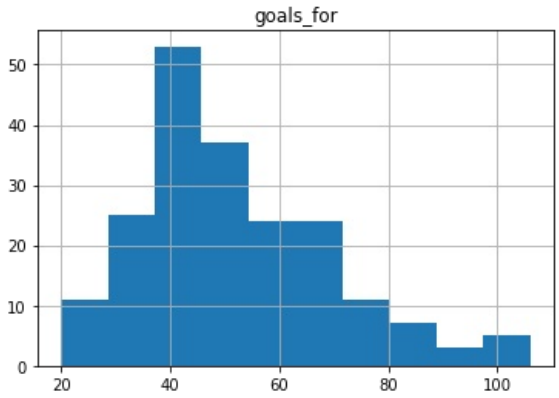
Data Exploration and Filtering

In [241…
```
# Let's get summary stats for each column except year
df['goals_for'].describe()
```

Out[241…
```
count    200.000000
mean      51.995000
std       17.419304
min       20.000000
25%       40.000000
50%       47.500000
75%       62.250000
max      106.000000
Name: goals_for, dtype: float64
```

In [308…
```
df.hist('goals_for')
```

Out[308…
```
array([[<AxesSubplot:title={'center':'goals_for'}>]], dtype=object)
```

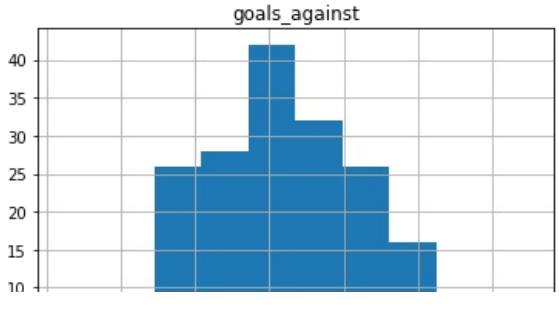


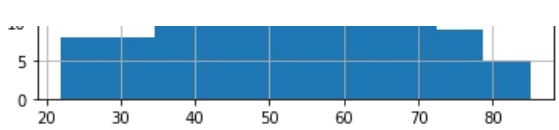
In [242…
```
df['goals_against'].describe()
```

Out[242…
```
count    200.000000
mean      51.995000
std       12.871806
min       22.000000
25%       43.000000
50%       52.000000
75%       60.000000
max       85.000000
Name: goals_against, dtype: float64
```

In [309…
```
df.hist('goals_against')
```

Out[309…
```
array([[<AxesSubplot:title={'center':'goals_against'}>]], dtype=object)
```



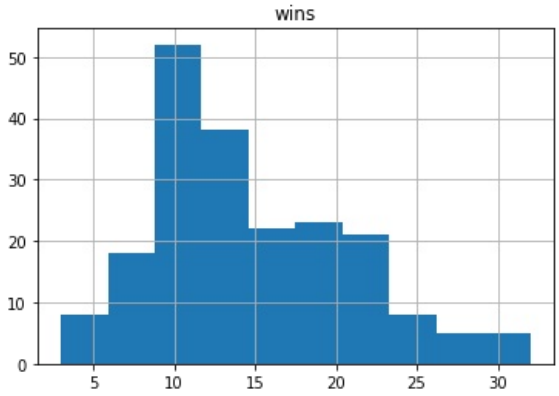


In [243…
```
df['wins'].describe()
```

Out[243…
```
count    200.000000
mean      14.485000
std        6.299061
min        3.000000
25%       10.000000
50%       13.000000
75%       19.000000
max       32.000000
Name: wins, dtype: float64
```

In [310…
```
df.hist('wins')
```

Out[310…
```
array([[<AxesSubplot:title={'center':'wins'}>]], dtype=object)
```

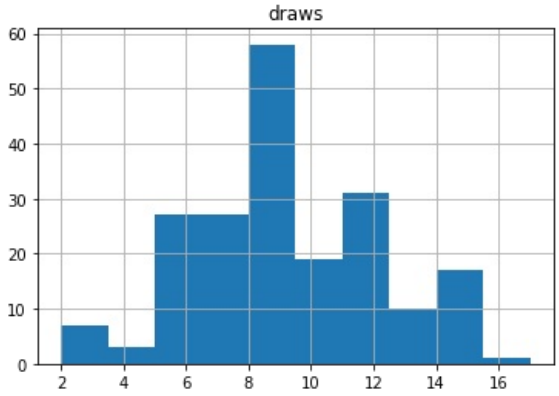


In [244…
```
df['draws'].describe()
```

Out[244…
```
count    200.000000
mean       9.030000
std        2.955125
min        2.000000
25%        7.000000
50%        9.000000
75%       11.000000
max       17.000000
Name: draws, dtype: float64
```

In [311…
```
df.hist('draws')
```

Out[311…
```
array([[<AxesSubplot:title={'center':'draws'}>]], dtype=object)
```

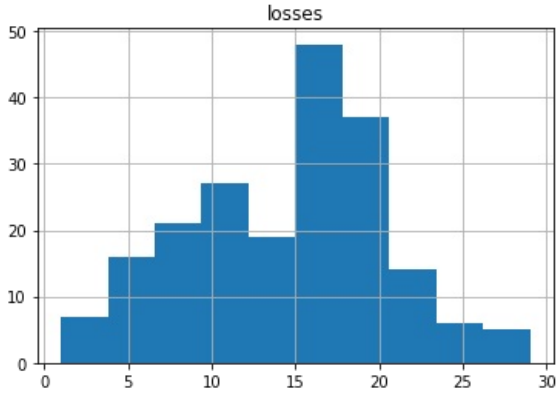


In [245…
```
df['losses'].describe()
```

```
count    200.000000
mean      14.485000
std        5.824034
min        1.000000
25%       10.000000
50%       15.000000
75%       19.000000
max       29.000000
Name: losses, dtype: float64
```

In [312…
```
df.hist('losses')
```

Out[312…
```
array([[<AxesSubplot:title={'center':'losses'}>]], dtype=object)
```

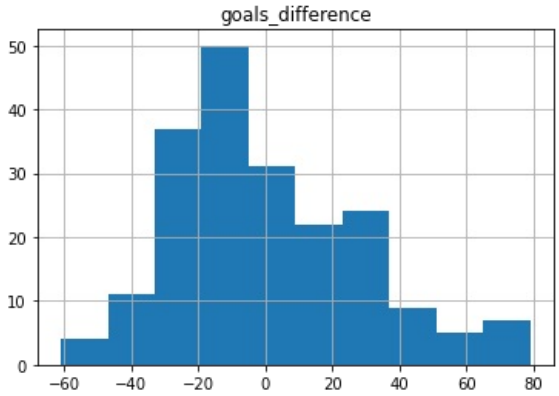


In [249…
```
# Now, let's add a column called 'goal_difference' and get summary stats. Also account for missing values just ir
# no need to account for outliers for the whole data (except for question 1: will explain later)
df['goals_difference'] = df['goals_for'] - df['goals_against']
df['goals_difference'].describe()
```

Out[249…
```
count    200.000000
mean       0.000000
std       27.832052
min      -61.000000
25%      -20.000000
50%       -6.500000
75%       20.250000
max       79.000000
Name: goals_difference, dtype: float64
```

In [313…
```
df.hist('goals_difference')
```

Out[313…
```
array([[<AxesSubplot:title={'center':'goals_difference'}>]], dtype=object)
```



In [250…
```
df = df.dropna()
```

In [251…
```
df.head(20)
```

Out[251…

	goals_for	goals_against	wins	draws	losses	year	goals_difference
0	86	43	28	5	5	2012	43
1	45	68	11	8	19	2012	-23

2	50	60	11	10	17	2012	-10
3	71	43	16	13	9	2012	28
4	49	60	9	14	15	2012	-11
5	72	37	21	10	7	2012	35
6	55	40	16	15	7	2012	15
7	66	46	21	9	8	2012	20
8	45	53	12	10	16	2012	-8
9	75	39	22	9	7	2012	36
10	66	34	23	9	6	2012	32
11	43	73	6	10	22	2012	-30
12	53	57	14	7	17	2012	-4
13	47	73	9	9	20	2012	-26
14	47	69	10	11	17	2012	-22
15	41	58	10	14	14	2012	-17
16	30	60	4	13	21	2012	-30
17	34	45	9	15	14	2012	-11
18	47	51	11	13	14	2012	-4
19	41	54	9	12	17	2012	-13

Question 1: What is the confidence interval at the 95% level for the mean number of goals from the years 2012-2016 and 2017-2021?

In [254…
```
df_20122016 = df[(df['year'] >= 2012) & (df['year'] <= 2016)]
df_20172021 = df[(df['year'] >= 2017) & (df['year'] <= 2021)]
```

In [258…
```
# Get means, standard errpr, and calculate confidence interval without accounting for outliers
df_20122016_mean = df_20122016['goals_for'].mean()
df_20122016_sem = df_20122016['goals_for'].sem()

df_20172021_mean = df_20172021['goals_for'].mean()
df_20172021_sem = df_20172021['goals_for'].sem()

z_score = 1.96
```

In [260…
```
# Generate confidence intervals
CI_20122016 = [df_20122016_mean - z_score * df_20122016_sem, df_20122016_mean + z_score * df_20122016_sem]
CI_20172021 = [df_20172021_mean - z_score * df_20172021_sem, df_20172021_mean + z_score * df_20172021_sem]
```

In [261…
```
print("Confidence Interval 2012-2016:", CI_20122016)
print("Confidence Interval 2017-2021:", CI_20172021)
```

```
Confidence Interval 2012-2016: [48.596182781793445, 55.00381721820655]
Confidence Interval 2017-2021: [48.5618260843752, 55.8181739156248]
```

For the years 2012-2016 in the Premier League, we are 95% confident that the true mean of goals scored falls between the range [48.596182781793445, 55.00381721820655]. For the years 2017-2021 in the Premier League, we are 95% confident that the true mean of goals scored falls between the range [48.5618260843752, 55.8181739156248]. The confidence intervals for both of these year ranges are similar, but the CI for 2012-2016 is much tighter, which indicates slightly less variability

Was it safe to ignore outliers???

In [265…
```
from scipy import stats
# This line of code accounts for all outliers
df_20122016 = df_20122016[(np.abs(stats.zscore(df_20122016['goals_for'])) < 3)]
df_20172021 = df_20172021[(np.abs(stats.zscore(df_20172021['goals_for'])) < 3)]
```

In [267…
```
df_20122016_mean = df_20122016['goals_for'].mean()
df_20172021_mean = df_20172021['goals_for'].mean()

print("Mean 2012-2016 Accounting for Outliers:", df_20122016_mean)
print("Mean 2017-2021 Accounting for Outliers:", df_20172021_mean)
```

```
Mean 2012-2016 Accounting for Outliers: 50.785714285714285
Mean 2017-2021 Accounting for Outliers: 52.19
```

It was safe to ignore outliers as the mean for both these years when outliers were excluded is still within the confidence intervals for these respective year ranges

Question 2: Predict the Goal Difference for a Team in the Premier League using Data from 2012-2021 using Linear Regression

In [268…
```
# Let's get the correlations for the dataframe
df.corr()
```

Out[268…

	goals_for	goals_against	wins	draws	losses	year	goals_difference
goals_for	1.000000	-0.681274	0.920274	-0.286903	-0.849759	0.013174	0.940949
goals_against	-0.681274	1.000000	-0.801520	-0.002506	0.868167	0.017829	-0.888872
wins	0.920274	-0.801520	1.000000	-0.389254	-0.884055	0.033661	0.946662
draws	-0.286903	-0.002506	-0.389254	1.000000	-0.086399	-0.143503	-0.178406
losses	-0.849759	0.868167	-0.884055	-0.086399	1.000000	0.036407	-0.933352
year	0.013174	0.017829	0.033661	-0.143503	0.036407	1.000000	0.000000
goals_difference	0.940949	-0.888872	0.946662	-0.178406	-0.933352	0.000000	1.000000

In [269…
```
import sklearn
import sklearn.linear_model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Develop a train and test set
train_set, test_set = train_test_split(df, train_size=0.7, random_state=42)
```

In [270…
```
train_set.head()
```

Out[270…

	goals_for	goals_against	wins	draws	losses	year	goals_difference
169	68	50	20	6	12	2020	18
97	41	56	11	11	16	2016	-15
31	102	37	27	5	6	2013	65
12	53	57	14	7	17	2012	-4
35	39	61	10	8	20	2013	-22

In [280…
```
X_train = train_set[['wins', 'draws']]
X_test = test_set[['wins', 'draws']]

y_train = train_set[['goals_difference']]
y_test = test_set[['goals_difference']]

reg_model = LinearRegression()
reg_model.fit(X=X_train, y=y_train)

intercept = reg_model.intercept_
slope = reg_model.coef_[0]

print("Regression Equation: y = ", slope[0],"x1 + ", slope[1], "x2 +", intercept[0])
```

```
Regression Equation: y =  4.621502654845676 x1 +  2.098982367938337 x2 + -86.58886635680689
```

In [281…
```
# Determine R-Squared
y_predicted = reg_model.predict(X_test)
r2_single = r2_score(y_test, y_predicted)
r2_single
```

Out[281… `0.9412237884147182`

My Linear Regression Model to predict the goal difference for a team using the wins and draws for a team as features had a r2 value of approx .94. A high r2 value signifies a good fit with the regression line which thus indicates a very good predictive power for my regression model

Question 3: Predict Whether the Number of Wins for a Team in the Premier League is Greater than the Mean Number of Wins from 2012-2021 using Data from 2012-2021 using Logistic Regression

Classification

In [314…
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, f1_score, recall_score, precision_score
train_set['wins_greater_mean'] = train_set['wins'] > df['wins'].mean()
test_set['wins_greater_mean']= test_set['wins'] > df['wins'].mean()
```

```
<ipython-input-314-4a82b722dbe3>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_set['wins_greater_mean'] = train_set['wins'] > df['wins'].mean()
<ipython-input-314-4a82b722dbe3>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_set['wins_greater_mean']= test_set['wins'] > df['wins'].mean()
```

In [315…
```
train_set.head()
```

Out[315…

	goals_for	goals_against	wins	draws	losses	year	goals_difference	wins_greater_mean
169	68	50	20	6	12	2020	18	True
97	41	56	11	11	16	2016	-15	False
31	102	37	27	5	6	2013	65	True
12	53	57	14	7	17	2012	-4	False
35	39	61	10	8	20	2013	-22	False

In [316…
```
train_set.corr()
```

Out[316…

	goals_for	goals_against	wins	draws	losses	year	goals_difference	wins_greater_mean
goals_for	1.000000	-0.657017	0.914149	-0.322689	-0.844665	-0.001458	0.939926	0.758423
goals_against	-0.657017	1.000000	-0.792932	0.045427	0.854825	-0.019252	-0.874903	-0.676598
wins	0.914149	-0.792932	1.000000	-0.432154	-0.882747	0.020468	0.946321	0.830924
draws	-0.322689	0.045427	-0.432154	1.000000	-0.042227	-0.090147	-0.227869	-0.324554
losses	-0.844665	0.854825	-0.882747	-0.042227	1.000000	0.024291	-0.929711	-0.751487
year	-0.001458	-0.019252	0.020468	-0.090147	0.024291	1.000000	0.007781	0.030943
goals_difference	0.939926	-0.874903	0.946321	-0.227869	-0.929711	0.007781	1.000000	0.793602
wins_greater_mean	0.758423	-0.676598	0.830924	-0.324554	-0.751487	0.030943	0.793602	1.000000

In [317…
```
# I didn't pick wins despite having the highest correlation because based off the value for wins,
# you can easily see whether it is greater than the mean
X_train_log = train_set[['goals_difference', 'draws']]
X_test_log = test_set[['goals_difference', 'draws']]

y_train_log = train_set[['wins_greater_mean']]
y_test_log = test_set[['wins_greater_mean']]

logistic_model = LogisticRegression()
logistic_model.fit(X_train_log, y_train_log)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

Out[317… `LogisticRegression()`

In [318…
```
y_pred_log = logistic_model.predict(X_test_log)
f1 = f1_score(y_test_log, y_pred_log)
recall = recall_score(y_test_log, y_pred_log)
precision = precision_score(y_test_log, y_pred_log)
accuracy = logistic_model.score(X_test_log, y_test_log)
```

In [319…

```
print("accuracy:", accuracy)
print("f1 score:", f1)
print("recall score:", recall)
print("precision:", precision)
```

```
accuracy: 0.9666666666666667
f1 score: 0.9545454545454546
recall score: 0.9545454545454546
precision: 0.9545454545454546
```

My logistic regression model using the two features of goal difference and draws to predict whether the number of wins for a Premier League team is greater than the mean number of wins from 2012-2021 had an accuracy score of 0.9666666666666667, which shows that the model fit the data very well. My model had an f1 score, which is the performance metric for classification and is calculated as the harmonic mean of precision and recall, of 0.9545454545454546, which again shows that my model has good precision. Additionally, the precision score for my model was 0.9545454545454546 and the recall score was 0.9545454545454546, which again signifies that this model can be used for predictions and fits the data quite well.