Project 2: Machine Learning Linear Regression Model Practice

- Name: Joshua Thomas
- Date Completed: 10/04/2022
- Purpose: Make a Machine Learning Model to predict the total wins of a Premier League Team given a correlated variable

Introduction

If there is one sport that I've loved my whole life, it's football (soccer). If there's one competition that has truly resonated with me, it's the English Premier League. As a life long Manchester United fan, I've always been fascinated with the constant competition, the vibrant atmospheres, and the wonderful play of the English game. As an aspiring Data Scientist who is getting his feet wet with Machine Learning, I thought there would be no better way to get acquainted with Supervised Learning than using Premier League data itself. Thus, in my first Machine Learning Model ever, I'm going to buuild a Linear Regression Model which can predict with decent accuracy the final wins of a Premier League Team.

In [1]:

```
# Imports to make Machine Learning Models
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import copy
import math
```

Data Preparation

In [2]:

```
# Step 1: Get the Data to Begin With (Using FBRef Public Database to download csv data)
# Read standard Premier League data of different years into separate dataframes
df2122 = pd.read_csv("Prem-2122.csv")
df2021 = pd.read_csv("Prem-2021.csv")
df1920 = pd.read_csv("Prem-1920.csv")
df1819 = pd.read_csv("Prem-1819.csv")
df1718 = pd.read_csv("Prem-1718.csv")

df_st_frames = [df2122, df2021, df1920, df1819, df1718]

df_standard = pd.concat(df_st_frames)

#Clean up indexes
df_standard.index = np.arange(0,len(df_standard))


# Now I want to load the shooting data
shooting2122 = pd.read_csv("Prem-2122-Shooting.csv")
shooting2021 = pd.read_csv("Prem-2021-Shooting.csv")
shooting1920 = pd.read_csv("Prem-1920-Shooting.csv")
shooting1819 = pd.read_csv("Prem-1819-Shooting.csv")
shooting1718 = pd.read_csv("Prem-1718-Shooting.csv")

shoot_st_frames = [shooting2122, shooting2021, shooting1920, shooting1819, shooting1718]
shooting_standard = pd.concat(shoot_st_frames)
#Bc column already exists
shooting_standard = shooting_standard.drop('xG', axis=1)
shooting_standard.index = np.arange(0, len(shooting_standard))

#Merge shooting data with standard data
df_standard = pd.concat([df_standard, shooting_standard], axis = 1, join = 'inner')

# Now I want to load the shoot creation data
sca2122 = pd.read_csv("Prem-2122-SCA.csv")
sca2021 = pd.read_csv("Prem-2021-SCA.csv")
sca1920 = pd.read_csv("Prem-1920-SCA.csv")
sca1819 = pd.read_csv("Prem-1819-SCA.csv")
sca1718 = pd.read_csv("Prem-1718-SCA.csv")

sca_frames = [sca2122, sca2021, sca1920, sca1819, sca1718]
sca_standard = pd.concat(sca_frames)
sca_standard.index = np.arange(0, len(sca_standard))

# Finally, I want to merge the shot creation data with the main, standard data
df_standard = pd.concat([df_standard, sca_standard], axis = 1, join = 'inner')

df_standard['xG']
```

Out[2]:

```
0      88.5
1      89.2
```

```
2     67.2
3     65.1
4     59.8
      ...
95    31.7
96    42.1
97    30.5
98    37.5
99    34.3
Name: xG, Length: 100, dtype: float64
```

Data Exploration and Filtering

In [3]:
```python
# Snapshot of data
df_standard.head(38)
```

Out[3]:

	Rk	Squad	MP	W	D	L	GF	GA	GD	Pts	...	Fld	Def	GCA	GCA90	PassLive.1	PassDead.1	Drib.1	Sh.1	Fld.1	Def.1
0	1	Manchester City	38	29	6	3	99	26	73	93	...	39	21	88	2.32	60	9	1	10	4	4
1	2	Liverpool	38	28	8	2	94	26	68	92	...	46	12	82	2.16	55	8	7	8	4	0
2	3	Chelsea	38	21	11	6	76	33	43	74	...	40	12	74	1.95	45	7	3	8	10	1
3	4	Tottenham	38	22	5	11	69	40	29	71	...	34	16	64	1.68	43	8	3	5	5	0
4	5	Arsenal	38	22	3	13	61	48	13	69	...	49	11	56	1.47	35	8	5	3	5	0
5	6	Manchester Utd	38	16	10	12	57	57	0	58	...	44	19	124	3.26	90	7	6	9	10	2
6	7	West Ham	38	16	8	14	60	51	9	56	...	37	21	75	1.97	51	3	6	7	6	2
7	8	Leicester City	38	14	10	14	62	59	3	52	...	43	15	64	1.68	43	4	3	6	8	0
8	9	Brighton	38	12	15	11	42	44	-2	51	...	36	21	60	1.58	32	2	8	8	5	5
9	10	Wolves	38	15	6	17	38	43	-5	51	...	32	23	98	2.58	73	3	4	8	4	6
10	11	Newcastle Utd	38	13	10	15	44	62	-18	49	...	37	25	156	4.11	111	8	13	18	5	1
11	12	Crystal Palace	38	11	15	12	50	46	4	48	...	46	13	142	3.74	96	9	7	16	10	4
12	13	Brentford	38	13	7	18	48	56	-8	46	...	40	20	100	2.63	74	6	6	6	5	3
13	14	Aston Villa	38	13	6	19	52	54	-2	45	...	33	18	65	1.71	41	3	3	7	8	3
14	15	Southampton	38	9	13	16	43	67	-24	40	...	40	20	32	0.84	25	2	1	0	2	2
15	16	Everton	38	11	6	21	43	66	-23	39	...	49	24	62	1.63	43	5	1	2	9	2
16	17	Leeds United	38	9	11	18	42	79	-37	38	...	35	12	114	3.00	91	4	5	8	5	1
17	18	Burnley	38	7	14	17	34	53	-19	35	...	41	21	52	1.37	35	4	3	5	2	3
18	19	Watford	38	6	5	27	34	77	-43	23	...	26	17	100	2.63	67	11	5	9	6	2
19	20	Norwich City	38	5	7	26	23	84	-61	22	...	31	21	56	1.47	35	3	4	8	3	3
20	1	Manchester City	38	27	5	6	83	32	51	86	...	46	10	95	2.50	67	2	7	8	10	1
21	2	Manchester Utd	38	21	11	6	73	44	29	74	...	53	12	86	2.26	61	7	2	6	7	3
22	3	Liverpool	38	20	9	9	68	42	26	69	...	36	17	60	1.58	42	5	1	5	6	1
23	4	Chelsea	38	19	10	9	58	36	22	67	...	37	16	46	1.21	25	6	6	3	4	2
24	5	Leicester City	38	20	6	12	68	50	18	66	...	47	17	97	2.55	68	7	5	6	10	1
25	6	West Ham	38	19	8	11	62	47	15	65	...	42	14	70	1.84	45	6	7	4	6	2
26	7	Tottenham	38	18	8	12	68	45	23	62	...	42	10	78	2.05	52	9	2	10	5	0
27	8	Arsenal	38	18	7	13	55	39	16	61	...	32	17	47	1.24	33	3	5	2	3	1
28	9	Leeds United	38	18	5	15	62	54	8	59	...	29	19	108	2.84	79	6	8	8	6	1
29	10	Everton	38	17	8	13	47	48	-1	59	...	44	32	111	2.92	79	3	8	5	11	5
30	11	Aston Villa	38	16	7	15	55	46	9	55	...	41	19	94	2.47	66	6	4	12	5	1
31	12	Newcastle Utd	38	12	9	17	46	62	-16	45	...	47	28	138	3.63	95	4	16	11	7	5
32	13	Wolves	38	12	9	17	36	52	-16	45	...	54	20	122	3.21	87	7	6	8	11	3
33	14	Crystal Palace	38	12	8	18	41	66	-25	44	...	52	19	68	1.79	45	5	4	6	7	1
34	15	Southampton	38	12	7	19	47	68	-21	43	...	16	11	25	0.66	15	2	2	3	3	0
35	16	Brighton	38	9	14	15	40	46	-6	41	...	45	18	80	2.11	47	7	7	7	9	3
36	17	Burnley	38	10	9	19	33	55	-22	39	...	57	22	110	2.89	78	6	10	7	6	3
37	18	Fulham	38	5	13	20	27	53	-26	28	...	33	8	57	1.50	42	3	1	3	5	3

38 rows × 57 columns

In [4]:
```
# Column Names
df_standard.columns
```

Out[4]:
```
Index(['Rk', 'Squad', 'MP', 'W', 'D', 'L', 'GF', 'GA', 'GD', 'Pts', 'Pts/MP',
       'xG', 'xGA', 'xGD', 'xGD/90', 'Attendance', 'Top Team Scorer',
       'Goalkeeper', 'Notes', 'Squad', '# Pl', '90s', 'Gls', 'Sh', 'SoT',
       'SoT%', 'Sh/90', 'SoT/90', 'G/Sh', 'G/SoT', 'Dist', 'FK', 'PK', 'PKatt',
       'npxG', 'npxG/Sh', 'G-xG', 'np:G-xG', 'Squad', '# Pl', '90s', 'SCA',
       'SCA90', 'PassLive', 'PassDead', 'Drib', 'Sh', 'Fld', 'Def', 'GCA',
       'GCA90', 'PassLive.1', 'PassDead.1', 'Drib.1', 'Sh.1', 'Fld.1',
       'Def.1'],
      dtype='object')
```

- That's a lot of columns. However, I only want to look at columns necessary to the task at hand: wins and data related to chance creation
- I'm going to create a new dataframe with relevant columns

In [5]:
```
columns_cc = ['W', 'GF', 'GA', 'GD', 'xG', 'xGA', 'xGD', 'xGD/90', 'SoT', 'SoT%', 'SoT/90', 'G/Sh', 'G/SoT', 'SCA
df_final = df_standard[columns_cc]
df_final.head(10)
```

Out[5]:

	W	GF	GA	GD	xG	xGA	xGD	xGD/90	SoT	SoT%	SoT/90	G/Sh	G/SoT	SCA	SCA90	GCA	GCA90
0	29	99	26	73	88.5	27.1	61.3	1.61	185	31.9	4.87	0.09	0.30	921	24.24	88	2.32
1	28	94	26	68	89.2	33.8	55.4	1.46	159	34.5	4.18	0.10	0.30	725	19.08	82	2.16
2	21	76	33	43	67.2	36.0	31.3	0.82	139	31.5	3.66	0.09	0.29	640	16.84	74	1.95
3	22	69	40	29	65.1	39.4	25.8	0.68	140	29.0	3.68	0.07	0.26	748	19.68	64	1.68
4	22	61	48	13	59.8	47.0	12.9	0.34	117	28.8	3.08	0.08	0.26	579	15.24	56	1.47
5	16	57	57	0	54.8	54.5	0.3	0.01	199	34.0	5.24	0.11	0.34	964	25.37	124	3.26
6	16	60	51	9	49.6	50.7	-1.0	-0.03	141	34.9	3.71	0.11	0.31	629	16.55	75	1.97
7	14	62	59	3	50.8	61.0	-10.1	-0.27	127	29.5	3.34	0.08	0.28	667	17.55	64	1.68
8	12	42	44	-2	46.0	45.4	0.5	0.01	142	29.3	3.74	0.08	0.26	711	18.71	60	1.58
9	15	38	43	-5	35.9	60.3	-24.4	-0.64	163	37.4	4.29	0.14	0.37	653	17.18	98	2.58

In [6]:
```
# Summary Statistics
df_final.describe()
```

Out[6]:

	W	GF	GA	GD	xG	xGA	xGD	xGD/90	SoT	SoT%	SoT/90
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	1.000000e+02	100.000000	100.000000	100.000000
mean	14.670000	52.190000	52.190000	0.000000	50.291000	50.290000	0.004000	-2.553513e-17	155.460000	32.892000	4.090900
std	6.559833	18.511091	13.441752	29.718545	13.934001	10.528859	23.027734	6.060803e-01	38.816174	3.169829	1.021602
min	3.000000	20.000000	22.000000	-61.000000	29.500000	22.800000	-43.600000	-1.150000e+00	92.000000	27.100000	2.420000
25%	10.000000	39.000000	43.750000	-20.000000	40.250000	45.100000	-15.275000	-4.025000e-01	127.000000	30.700000	3.340000
50%	13.500000	48.000000	53.000000	-4.500000	47.450000	51.900000	-5.150000	-1.350000e-01	145.000000	32.800000	3.815000
75%	19.000000	62.250000	61.250000	16.500000	58.050000	57.500000	12.625000	3.300000e-01	179.500000	34.825000	4.722500
max	32.000000	106.000000	84.000000	79.000000	93.100000	76.700000	63.600000	1.670000e+00	255.000000	40.700000	6.710000

- I'm going to explore different variables associated with chance creation which are correlated with wins

In [7]:
```
df_final.corr()
```

Out[7]:

	W	GF	GA	GD	xG	xGA	xGD	xGD/90	SoT	SoT%	SoT/90	G/Sh	G/S
W	1.000000	0.933846	-0.827175	0.955807	0.889816	-0.786718	0.898446	0.898571	0.043168	0.005798	0.042851	-0.010557	0.0201
GF	0.933846	1.000000	-0.723110	0.949944	0.950095	-0.743959	0.915346	0.915367	0.056109	0.007497	0.055920	-0.028230	-0.0037
GA	-0.827175	-0.723110	1.000000	-0.902713	-0.714735	0.878445	-0.834394	-0.834289	-0.009810	0.019736	-0.009516	-0.000824	-0.0370
GD	0.955807	0.949944	-0.902713	1.000000	0.915071	-0.860720	0.947549	0.947514	0.039386	-0.004257	0.039136	-0.017211	0.0143
xG	0.889816	0.950095	-0.714735	0.915071	1.000000	-0.767289	0.956046	0.956063	0.066202	0.009768	0.065931	-0.032323	-0.0143
xGA	-0.786718	-0.743959	0.878445	-0.860720	-0.767289	1.000000	-0.921597	-0.921553	-0.035381	0.031507	-0.034922	0.049683	0.0183
xGD	0.898446	0.915346	-0.834394	0.947549	0.956046	-0.921597	1.000000	0.999989	0.056182	-0.008413	0.055807	-0.042123	-0.0168
xGD/90	0.898571	0.915367	-0.834289	0.947514	0.956063	-0.921553	0.999989	1.000000	0.056203	-0.008360	0.055827	-0.042050	-0.0168

SoT	0.043168	0.056109	-0.009810	0.039386	0.066202	-0.035381	0.056182	0.056203	1.000000	0.700948	0.999996	0.645619	0.4875
SoT%	0.005798	0.007497	0.019736	-0.004257	0.009768	0.031507	-0.008413	-0.008360	0.700948	1.000000	0.700827	0.747143	0.4703
SoT/90	0.042851	0.055920	-0.009516	0.039136	0.065931	-0.034922	0.055807	0.055827	0.999996	0.700827	1.000000	0.645428	0.4873
G/Sh	-0.010557	-0.028230	-0.000824	-0.017211	-0.032323	0.049683	-0.042123	-0.042050	0.645619	0.747143	0.645428	1.000000	0.9208
G/SoT	0.020106	-0.003757	-0.037005	0.014397	-0.014335	0.018321	-0.016811	-0.016852	0.487544	0.470364	0.487361	0.920819	1.0000
SCA	0.077977	0.089603	-0.034352	0.071349	0.109576	-0.078068	0.101921	0.101967	0.945797	0.472082	0.945779	0.488744	0.4044
SCA90	0.077952	0.089611	-0.034332	0.071345	0.109628	-0.078084	0.101960	0.102006	0.945805	0.472059	0.945788	0.488818	0.4045
GCA	0.048588	0.034897	-0.017627	0.029709	0.053743	-0.033795	0.047920	0.047819	0.914104	0.680605	0.913966	0.813487	0.7232
GCA90	0.048408	0.034845	-0.017848	0.029777	0.053758	-0.034282	0.048150	0.048050	0.914225	0.680720	0.914088	0.813446	0.7231

- GD has the highest correlation with wins. Since the other variables are correlated, I'm going to use that as my single feature variable to try and predict wins

Linear Regression Model

In [8]:
```
# Let's get the column data from each dataset for the columns GD and W and
# store them in 2 different variables: x and y respectively
x = []
y = []
```

In [9]:
```
for index in df_final.index:
    x.append(df_final.loc[index, "GD"])
    y.append(df_final.loc[index, "W"])
```

In [10]:
```
# Training variables
x_train = x[:80]
y_train = y[:80]

# Test variables
x_test = x[80:]
y_test = y[80:]
```

In [11]:
```
# The model that predicts the values using a scalar w and intercept b
def model(x, w, b):
    # Number of entries
    m = len(x)
    f_wb = np.zeros(m)
    for i in range(m):
        f_wb[i] = w * x[i] + b
        if f_wb[i] >= 38:
            f_wb[i] = 38
        f_wb[i] = int(round(f_wb[i], 0))
    return f_wb

def predict(val, w, b):
    return val*w + b
```

In [12]:
```
# Function to calculate the difference between the actual and predicted values given a specific w and b value
def cost(x, y, w, b):
    # Number of entries
    m = len(x)
    predict = model(x, w, b)
    cost_sum = 0
    for i in range(m):
        cost = (predict[i] - y[i])**2
        cost_sum = cost_sum + cost
    total_cost = (1/(2*m))*cost_sum
    return total_cost
```

In [13]:
```
# To find line of best fit. Uses gradient descent to determine values of linear regression
slope, intercept, r, p, std_err = stats.linregress(x_train, y_train)
```

In [14]:
```
slope
```

Out[14]:
```
0.20770229337948934
```

In [15]:
```
intercept
```

Out[15]: 14.825

In [16]:
```
cost(x_train, y_train, slope, intercept)
```

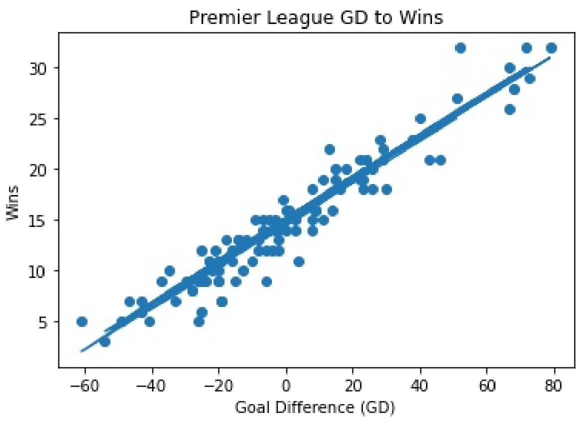
Out[16]: 1.925

- The model with the current w and b values has a minimized cost of 1.925 games, which means that the actual values differ from the predicted values by approximately 2 games, which is pretty good. Therefore, we have great confidence that this model can predict the final wins for a team given their final GD

Plot the Model

In [22]:
```
plt.scatter(x, y)
plt.plot(x, model(x, slope, intercept))
plt.title("Premier League GD to Wins")
plt.xlabel("Goal Difference (GD)")
plt.ylabel("Wins")
```

Out[22]: Text(0, 0.5, 'Wins')



As you can see, the model fits the data VERY well