2022

UCD Intro to Data Analytics & Intro to Python

PROJECT REPORT

PREPARED BY: CONOR CAHALANE

TUTOR: SHUBHAM JAIN

Contents

GitHub URL

https://github.com/conortcahalane/UCDPythonProject

Abstract

For this project, I chose to do Python analysis on Premier League statistics spanning the last 22 years.

The purpose of the application was to develop a Python project to analyse a real-world scenario and generate valuable insights by visualizing information. The project aims to analyse data from different data sources, manipulate information and visualise to generate insights.

Using Jupyter notebook as my platform and a host of useful resources such as packages NumPy and Pandas, an API from AlphaVantage and Visualization tools from Plotly, I was able to develop and implement the functionality required for the project.

Introduction

As described in paper information provided by University College Dublin, the goal of the assignment is to demonstrate how you are thinking about putting course concepts and learning into practice to demonstrate the course learning outcomes.

Prior to this course, I had no previous experience with the programming language Python or Data Analysis tools such as data visualization. Due to this I knew a substantial portion of time during this project would be dedicated to investigating, understanding, and testing smaller elements of functionality, as well as looking at different implementations to try better understanding how the code was operating and what I thought would be the best solution to each bullet point set out by UCD.

Due to this for my use case I chose Premier league table statistics from 2000-2022. I chose this dataset as I knew there would be so much new concepts and functionality to deal with that my dataset may as well be from a topic in which I am familiar with. Using this dataset allowed me to focus on the coding and functionality aspects of the project more and did not impair me in having to wrap my head around aspects of the dataset.

Dataset

Justification:

I used Kaggle to find a dataset which appealed to me. It is always beneficial to work on topics you are interested in. Due to understanding the topic in detail, more insightful analysis can be conducted on it, thus leading to more applications for the data. This results in a greater number of outcomes that can be gained from the analysis. If I was to choose Cricket instead of Football, then many interesting data points might have gone over my head as I would not have had the background knowledge as to why it was interesting.

The EPL Standings 200-2022 is comprised of 12 columns with the headers:
Season, Pos, Team, Pld, W, L, D, GF, GA, GD, Pts, Qualification or relegation

The dataset is comprised of 440 rows comprising all the competing teams and their statistics for each season of the Premier League.

Source:

https://www.kaggle.com/datasets/quadeer15sh/premier-league-standings-11-seasons-20102021?resource=download

Implementation Process

2. Importing

Firstly, to get our dataset into a Python DataFrame we use Pandas. We import this into our Python code and use it's .read_csv() function to create a DataFrame out of our csv file downloaded from Kaggle. This can be seen below:

```
PL = pd.read_csv("EPL Standings 2000-2022.csv",index_col = 0)
PL
```

Another example of importing in the project is the use of the TIME_SERIES_INTRADAY API courtesy of AlphaVantage. Given the Premier League theme of the project I thought it would be interesting to use this API which retrieves stock data from an online API to pull information on one of the only football clubs in the world that has a ticker symbol on the New York Stock Exchange, Manchester United (Ticker Symbol: MANU) which also happens to be a Premier league club.

This was achieved by using my unique API Key given to me by AlphaVantage and inputting it into the below code. The url was also edited to only retrieve MANU data.

```
import requests

url =
'https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=MANU&interval=5min&apikey=XGA4O6I3IUJX
DA1P'
r = requests.get(url)
data = r.json()

print(data)
```

3. Preparation

Sorting

Utilized Pythons .sort_values() function in order to group together all the individual teams row data to be in sequence, having that data then be sorted by descending years and finally showing which position they came in said year. A code snippet of this can be seen below:

```
PL_Seasons.sort_values(
    by=["Team", "Season","Pos"],
    ascending=False
)[["Team", "Season","Pos"]]
```

Output:

	Team	Season	Pos
429	Wolverhampton Wanderers	2021	10
412	Wolverhampton Wanderers	2020	13
386	Wolverhampton Wanderers	2019	7
366	Wolverhampton Wanderers	2018	7
239	Wolverhampton Wanderers	2011	20
...	...	...	...
81	Arsenal	2004	2
60	Arsenal	2003	1
41	Arsenal	2002	2
20	Arsenal	2001	1
1	Arsenal	2000	2

Indexing

A quick example of some Boolean Indexing involved in the project was my wish to have a view of all the teams that scored over 90 points in a single season. To achieve this I simply indexed the PL_Seasons DataFrame on the column 'Pts' and added a condition to only return the rows that had scored over 90. Below you can find this snippet and output:

```
PL_Seasons[PL_Seasons['Pts']>90]
```

Output:

Season	Pos	Team	Pld	W	D	L	GF	GA	GD	Pts	Qualification or relegation
2004-05	1	Chelsea	38	29	8	1	72	15	57	95	Qualification for the Champions League group s...
2005-06	1	Chelsea	38	29	4	5	72	22	50	91	Qualification for the Champions League group s...
2016-17	1	Chelsea	38	30	3	5	85	33	52	93	Qualification for the Champions League group s...
2017-18	1	Manchester City	38	32	4	2	106	27	79	100	Qualification for the Champions League group s...
2018-19	1	Manchester City	38	32	2	4	95	23	72	98	Qualification for the Champions League group s...
2018-19	2	Liverpool	38	30	7	1	89	22	67	97	Qualification for the Champions League group s...
2019-20	1	Liverpool	38	32	3	3	85	33	52	99	Qualification for the Champions League group s...
2021-22	1	Manchester City	38	29	6	3	99	26	73	93	Qualification for the Champions League group s...
2021-22	2	Liverpool	38	28	8	2	94	26	68	92	Qualification for the Champions League group s...

Grouping

Grouping is an excellent way of providing a lot of data quickly and efficiently. Here I used the .groupby() function in conjunction with the .mean() function on all the teams in the Premier League dataframe to give a quick overview of their average league position, average points, average wins and many more.

```
Teams_Grouped = PL_Seasons.groupby(["Team"]).mean()
Teams_Grouped
```

Output (Not the full output):

Team	Pos	Pld	W	D	L	GF	GA	GD	Pts
Arsenal	3.772727	38.0	21.363636	8.772727	7.863636	70.954545	39.818182	31.136364	72.863636
Aston Villa	11.947368	38.0	11.842105	11.052632	15.105263	45.578947	53.105263	-7.526316	46.578947
Birmingham City	14.142857	38.0	10.428571	11.714286	15.857143	39.000000	51.428571	-12.428571	43.000000
Blackburn Rovers	11.636364	38.0	12.727273	10.000000	15.272727	47.090909	53.818182	-6.727273	48.181818
Blackpool	19.000000	38.0	10.000000	9.000000	19.000000	55.000000	78.000000	-23.000000	39.000000
Bolton Wanderers	12.454545	38.0	12.000000	10.000000	16.000000	45.000000	55.727273	-10.727273	46.000000
Bournemouth	13.800000	38.0	11.200000	8.600000	18.200000	48.200000	66.000000	-17.800000	42.200000
Bradford City	20.000000	38.0	5.000000	11.000000	22.000000	30.000000	70.000000	-40.000000	26.000000
Brentford	13.000000	38.0	13.000000	7.000000	18.000000	48.000000	56.000000	-8.000000	46.000000
Brighton & Hove Albion	14.400000	38.0	9.600000	13.000000	15.400000	38.000000	51.600000	-13.600000	41.800000
Burnley	15.000000	38.0	10.375000	9.500000	18.125000	37.500000	56.875000	-19.375000	40.625000

Drop duplicates, replace missing values

Thankfully my dataset imported with no duplicates or missing values, but these would've been remedied using the below coding functions.
This would have dropped any rows that came back as duplicates:

```
drop_duplicates= PL.drop_duplicates()
```

This is a check to see if there are any data gaps in the code, checking this is important in data analysis to ensure you aren't surprised in the future if you code breaks due to significant data gaps:

```
missing_values_count = PL.isnull().sum()
print(missing_values_count[0:11])
```

Output:

```
Pos                          0
Team                         0
Pld                          0
W                            0
D                            0
L                            0
GF                           0
GA                           0
GD                           0
Pts                          0
Qualification or relegation  0
dtype: int64
```

Merge DataFrames & Lists

For this ask I manually created a DataFrame which contained the Manchester United starting eleven squad and their ages. I initially created lists and added them to an array. I then called the DataFrame function in order to convert the lists. This was needed in order to merge this data to my main PL_Seasons DataFrame. This can be seen below:

```
MU_Player_data = {'Name': ['David De Gea', 'Harry Maguire', 'Victor Lindelof', 'Luke Shaw',
                           'Diogo Dalot', 'Fred', 'Scott McTominay', 'Bruno Fernandes',
                           'Cristiano Ronaldo', 'Marcus Rashford', 'Jadon Sancho'],
                  'Age': [31, 29, 28, 27,
                          23, 29, 25, 27,
                          37, 24, 22],
                  'Team1': ['Manchester United','Manchester United','Manchester United','Manchester United',
                            'Manchester United','Manchester United','Manchester United','Manchester United',
                            'Manchester United','Manchester United','Manchester United']}
# Create DataFrame
MU_Player = pd.DataFrame(MU_Player_data)
```

Following, In the below code we can see an outer merge being used with the how parameter, which results in a full outer join. In an outer join all rows from both DataFrames will be present in the new DataFrame. This results in showing the full team for each season Manchester United played and all the other teams, with their player information showing NaN (Not a Number). No rows are lost in an outer join, even when they do not have a match in the other DataFrame.

```
MU_outer_merged = pd.merge(
    PL_Seasons, MU_Player, how="outer", on=["Team1"]
)
MU_outer_merged
```

Output:

	Season	Pos	Team	Pld	W	D	L	GF	GA	GD	Pts	Qualification or relegation	Name	Age
0	2000	1	Manchester United	38	24	8	6	79	31	48	80	Qualification for the Champions League first g...	David De Gea	31.0
1	2000	1	Manchester United	38	24	8	6	79	31	48	80	Qualification for the Champions League first g...	Harry Maguire	29.0
2	2000	1	Manchester United	38	24	8	6	79	31	48	80	Qualification for the Champions League first g...	Victor Lindelof	28.0
3	2000	1	Manchester United	38	24	8	6	79	31	48	80	Qualification for the Champions League first g...	Luke Shaw	27.0
4	2000	1	Manchester United	38	24	8	6	79	31	48	80	Qualification for the Champions League first g...	Diogo Dalot	23.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
655	2020	16	Brighton & Hove Albion	38	9	14	15	40	46	-6	41	Not Applicable	NaN	NaN
656	2021	9	Brighton & Hove Albion	38	12	15	11	42	44	-2	51	Not Applicable	NaN	NaN
657	2017	16	Huddersfield Town	38	9	10	19	28	58	-30	37	Not Applicable	NaN	NaN
658	2018	20	Huddersfield Town	38	3	7	28	22	76	-54	16	Relegation to the EFL Championship	NaN	NaN
659	2021	13	Brentford	38	13	7	18	48	56	-8	46	Not Applicable	NaN	NaN

660 rows × 14 columns

4.Analysis

Conditional statements & Looping & NumPy & Classes

Below we introduce a for loop which will help minimize the amount of code needed in the analysis. This for loop will loop continuously through the number of values in the 'Wins' column until it has passed through them all. For each row of the DataFrame the for loop will check using the condition statement if and elif whether the teams amount of wins that year were above the 22-year PL average. This PL win average was defined earlier in the project as an integer and is now being called back here as it is relevant. If the team had above average wins that year then the loop will return a sentence stating the year, team name and that they have won more than the PL average, else it will return that they won less games than the average. The colour class is also used in the return to emphasise the result and help the readability of the sentence. The NumPy function .any() is used here to test whether any array element along a given axis evaluates to True. Without this function the loop would return an error saying a series cannot be used.

```
for i in range(len(PL_Seasons['W'])):
    if (PL_Seasons['W'][i] > PL_Average_W).any():
        print("In",PL_Seasons['Season'][i],PL_Seasons['Team'][i],"won ",colour.GREEN+color.BOLD+colour.UNDERLINE+"more"+
colour.END," games than the 22 year Premier League average.")

    elif (PL_Seasons['W'][i] < PL_Average_W).any():
        print("In",PL_Seasons['Season'][i],PL_Seasons['Team'][i],"won ",colour.RED+color.BOLD+colour.UNDERLINE+"less"+
colour.END," games than the 22 year Premier League average.")
```

Output (Snippet):

```
In 2000 Coventry City won  less  games than the 22 year Premier League average.
In 2000 Bradford City won  less  games than the 22 year Premier League average.
In 2001 Arsenal won  more  games than the 22 year Premier League average.
In 2001 Liverpool won  more  games than the 22 year Premier League average.
In 2001 Manchester United won  more  games than the 22 year Premier League average.
In 2001 Newcastle United won  more  games than the 22 year Premier League average.
In 2001 Leeds United won  more  games than the 22 year Premier League average.
In 2001 Chelsea won  more  games than the 22 year Premier League average.
In 2001 West Ham United won  more  games than the 22 year Premier League average.
```

Define a custom function to create reusable code

Below can be found a user defined function that checks to see if the team typed into the function played in the premier league. This function uses the conditional statement if to check the DataFrame for the function's argument.

User defined function:

```
def PremierLeague_Team_Check(i):
    if (i == PL_Seasons['Team1]).any():
        print("This team", colour.GREEN+color.BOLD+"did"+colour.END, "play in the premier league.")
    else:
        print("This team", colour.RED+color.BOLD+"didn't"+colour.END, "play in the premier league.")
```

Calling the User defined function:

```
#An example of a team that is in the premier league
PremierLeague_Team_Check('Chelsea')
```

Result:

```
This team did play in the premier league.
```

Calling the User defined function:

```
#An example of a team that is in the premier league
PremierLeague_Team_Check('Barcelona')
```

Result:

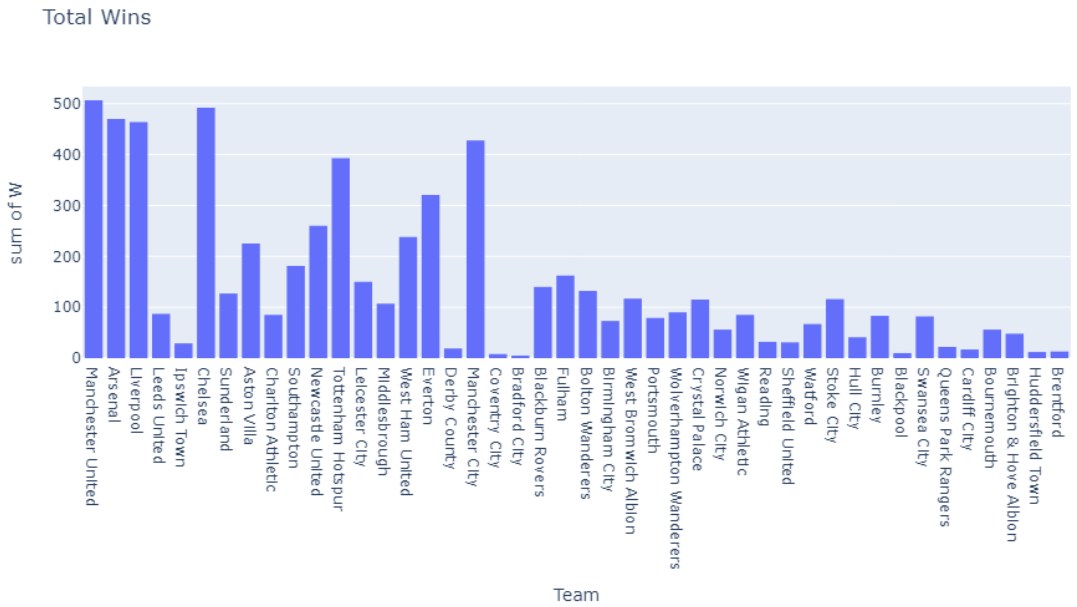
```
This team didn't play in the premier league.
```

Results

5. Visualization

Below we imported Plotly after installing it into our Jupyter through pip in the terminal. Using Plotly we can visualise the DataFrame in a multitude of diverse ways. Firstly, below is a histogram chart of the total wins by each team that participated in the premier league. By simply hovering over each bin the user can get a view of how many wins each team exactly had.

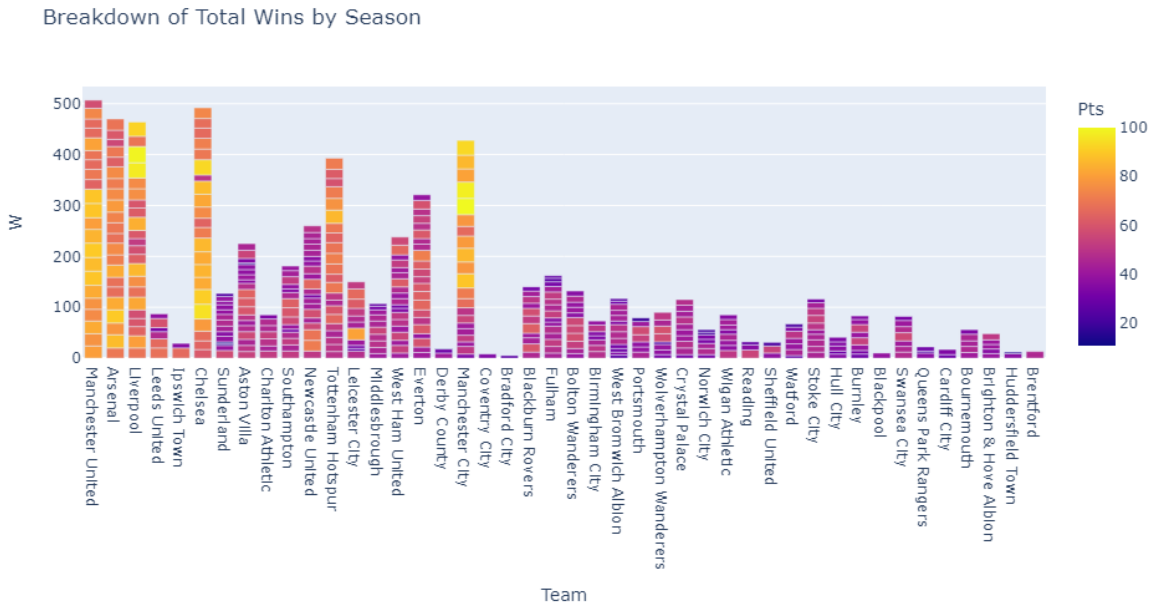
```
import plotly.express as px
df = PL_
fig = px.histogram(df, x='Team1',y='W', title='Total Wins')
fig.show()
```



Secondly, below is a bar chart of the total wins by each team that's participated in the premier league. By simply hovering over each bin the user can get a view of how many wins each team exactly per season, starting with the earliest season they participated in at the bottom. Using the color attribute on points, we can also visually see how each teams

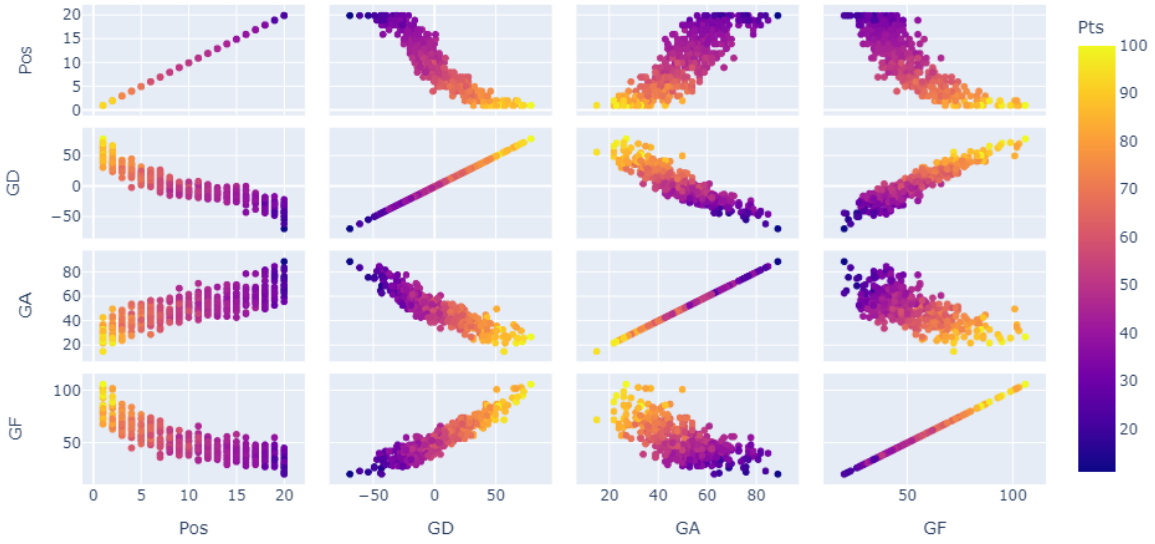
points tally was in each season. The legend to the right of the graph shows that the higher the points tally the brighter the season will display. By defining Hover_data we can expand on the information providing giving users the full W/L/D/Points data for each season.

```
fig = px.bar(PL_Seasons, x='Team', y='W', hover_data=['Team','Season','Pos', 'W', 'D', 'L', 'Pts'],
        title="Breakdown of Total Wins by Season", color = "Pts")
fig.show()
```



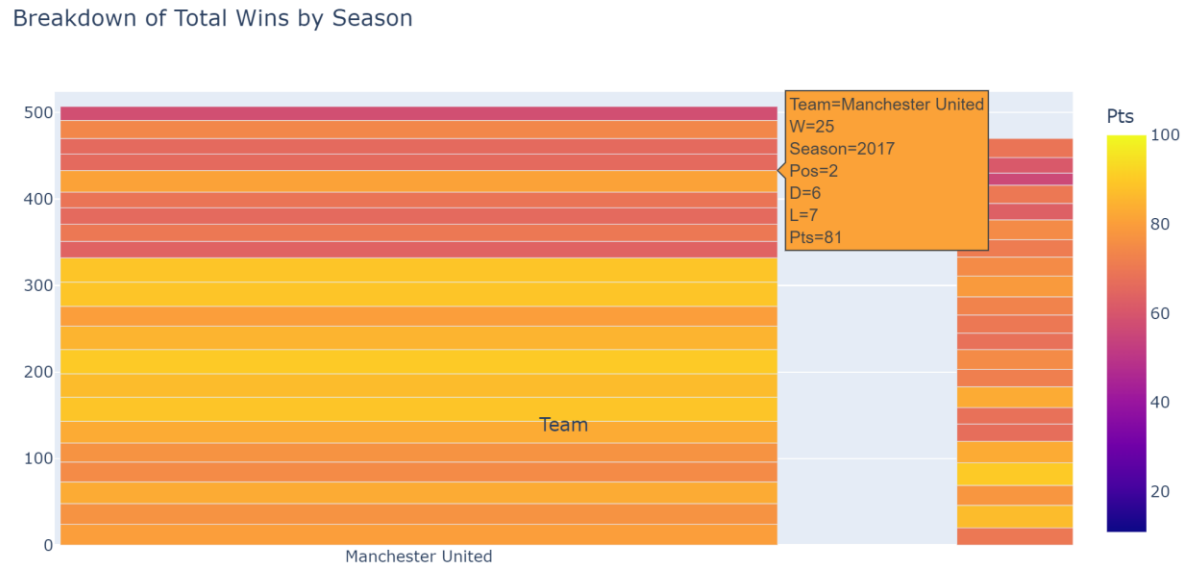
Lastly, here we see the visualization of scatter graphs being used with Goal Difference, Goals Against and Goals For. The colors correspond with the points gained with the more points resulting in brighter datapoints. By defining Hover_data we can expand on the information providing giving users the full Pos/GD/GA/GF data for each season.

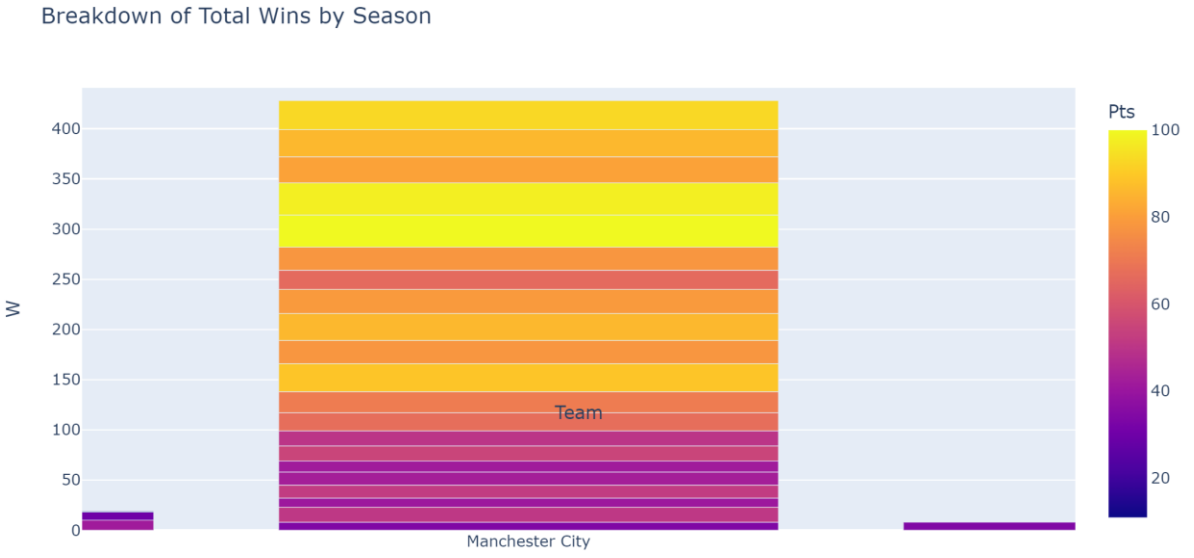
```
fig = px.scatter_matrix(PL_Seasons,dimensions=["Pos", "GD", "GA", "GF"],
   hover_data=['Team','Season', 'GA', 'GF', 'Pts','Pos'],color="Pts")
fig.show()
```



Insights

One of the most striking insights that can be found from the bar chart visualization is the pure hand-over in dominance from Manchester United to Manchester City in the last decade. This can be strikingly seen in the bar chart colour tones. Man United going from bright coloured data points (Indicating very high league positions) to darker tones (indicating lower league positions). This was the opposite case for Man City as can be seen vividly in the two screenshots below:

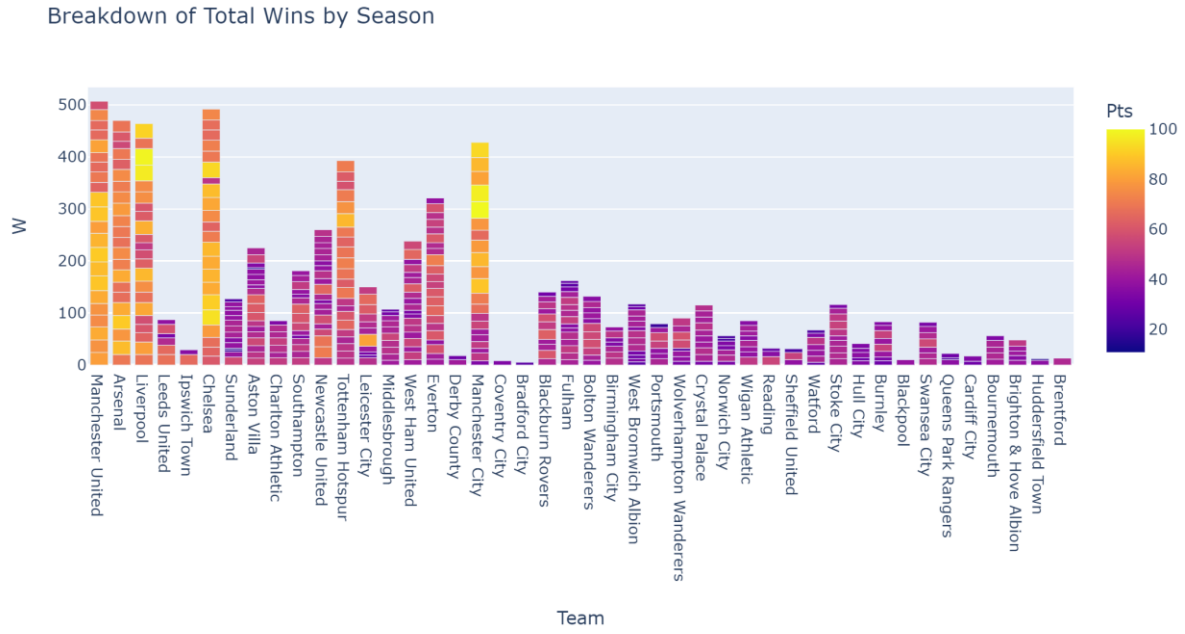




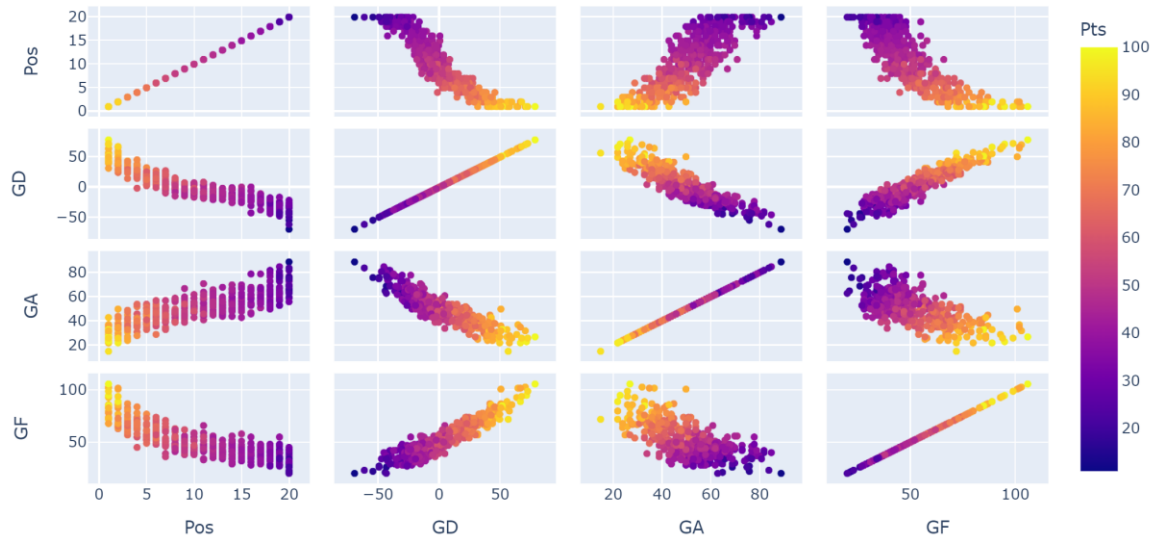
Following, another interesting insight is that in only 7 of the 22 Premier League seasons analyzed ninety points or more were required to win the league outright, with 2 of these seasons having 2 contenders scoring over 90 points.

Season	Pos	Team	Pld	W	D	L	GF	GA	GD	Pts	Qualification or relegation
2004-05	1	Chelsea	38	29	8	1	72	15	57	95	Qualification for the Champions League group s...
2005-06	1	Chelsea	38	29	4	5	72	22	50	91	Qualification for the Champions League group s...
2016-17	1	Chelsea	38	30	3	5	85	33	52	93	Qualification for the Champions League group s...
2017-18	1	Manchester City	38	32	4	2	106	27	79	100	Qualification for the Champions League group s...
2018-19	1	Manchester City	38	32	2	4	95	23	72	98	Qualification for the Champions League group s...
2018-19	2	Liverpool	38	30	7	1	89	22	67	97	Qualification for the Champions League group s...
2019-20	1	Liverpool	38	32	3	3	85	33	52	99	Qualification for the Champions League group s...
2021-22	1	Manchester City	38	29	6	3	99	26	73	93	Qualification for the Champions League group s...
2021-22	2	Liverpool	38	28	8	2	94	26	68	92	Qualification for the Champions League group s...

Furthermore, through our bar chart we can instantly see that of all the teams involved in the Premier League throughout the last 22 year that only 3 of them have been promoted to only instantly be relegated and never to reappear, these teams being Coventry City, Bradford City and Blackpool.



Moving on, we can see using our scatter graph that Goals For, Goals Against and Goal Differences are all heavily linked to league Positions. There being very few outliers to this trend, one of the being Liverpool's 2013 season where they had a very high goals for of 101 and a high goals against of 51 despite still coming 2nd in the league with a goal difference of 51. These stats made them stand out against the rest for still placing high in the leaderboard but conceding an awful amount of goals.



The final insight being a group view of how all the teams ranked when it came to two key attributes, Position and Points throughout the 22 years. This showed that even with the recent failures this decade Manchester United still retain the best Premier League position average and points average.

Team	Pos	Pts
Manchester United	2.863636	77.181818
Chelsea	3.318182	75.681818
Arsenal	3.772727	72.863636
Liverpool	4.136364	72.318182
Manchester City	6.000000	68.571429
Tottenham Hotspur	6.409091	62.272727
Everton	9.181818	54.136364
Leicester City	10.363636	50.363636
Newcastle United	11.100000	48.650000
Ipswich Town	11.500000	51.000000
Leeds United	11.500000	51.833333
Blackburn Rovers	11.636364	48.181818
West Ham United	11.842105	47.105263
Southampton	11.866667	46.933333

Machine learning

Using machine learning in the future any object, picture, video or soundbite will be able to solely be reproduced by machine learning. Once thought only possible in a sci-fi novel is now ever increasingly being proven to be true.

Evidence of this prediction is the recent release of Dall-e 2. DALL·E 2 is a new AI system that can create realistic images and art from a description in natural language. Its results are crisp, accurate and return in seconds. With this open-source machine learning AI users are prompted to type anything with the aim that the AI will output an image of exactly what you asked for. Horse on the moon, Dalle-2 in its current state will output exactly that in under a minute. This AI not only returns what you ask for but completely pixel by pixel creates it for you ensuring that this image has never been seen or presented before. This in the future will be applied to video and paired with audio to have AI create whole videos are completely of the AI's making.

Regression Vs Classification

The biggest difference in the argument of regression versus classification is that regression can aid in the prediction of a continuous quantity, classification labels the classes it predicts and gives a result based on those labels.

In a school setting for example using classification it would return a pass or fail based on the data it was given, yet with regression it would return an accurate percentage. There are appropriate use cases for both models given the circumstances they are being asked of.

References

https://github.com/conortcahalane/UCDPythonProject

https://www.kaggle.com/datasets/quadeer15sh/premier-league-standings-11-seasons-20102021?resource=download

https://jupyter.org/

https://pandas.pydata.org/

https://numpy.org/

https://plotly.com/