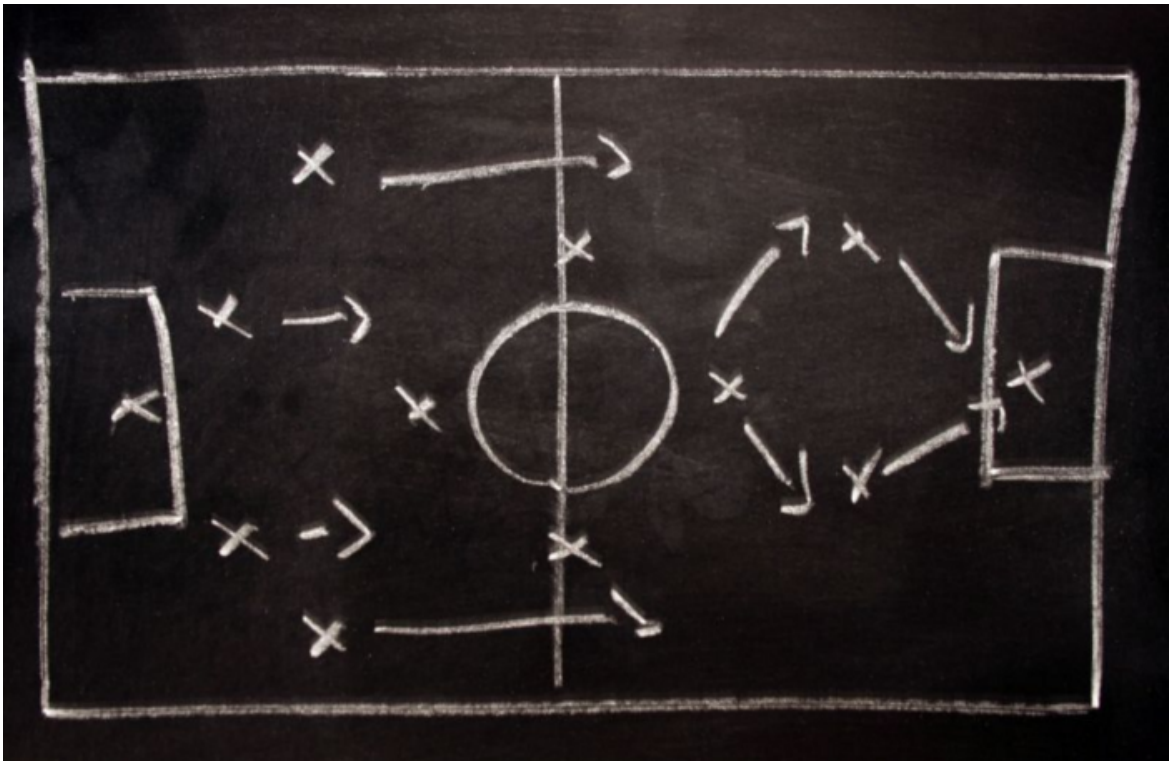
Pandas in the Premier League

How can we get started with Pandas for Data Analysis

Stephen Fordham Follow
Apr 21, 2019 · 7 min read ★



It's one of the closest Premier League title races in years and — with

just a handful of games remaining who will take the honors this season? The race for the illustrious Champions Leagues places are also being closely contested.

With this in mind, I thought I would showcase how we can use the Python Pandas library to parse the Premier League Table, and show how we can start performing some initial exploratory analysis using Python.

Web Scraping

Pandas has a built in function, **read_html()** which uses the libraries lxml and Beautiful Soup to automatically parse data out of HTML files as DataFrame objects. To begin, it is a requirement to install some additional libraries used by **read_html()**. In the terminal, type:

pip install lxml

pip install beautifulsoup4 html5lib

The pandas **read_html** function has a number of custom options, but by default it searches for and attempts to parse all tabular data contained within <table> tags. This results in a list of DataFrame objects.

I import the pandas library and use the **read_html** function to parse the Premier League Table and assign it to the variable prem_table. This returns a list; of which I take the first element which points to the Premier League Table as of 20/04/19 BST 22:00. As much focus

centers around the battle for the Top 4 places, I have decided to visualize the top 6 entries in this Table using the **.head()** method. To further confirm things are looking okay, I use the shape attribute, which tells me that there are 21 rows and 12 columns. This is a useful technique, because Python starts counting the rows from index 0 and there are 20 Premier League Teams, therefore I seem to have one row too many. As things looked good at the head of my DataFrame the problem *must* point to any extra row at the tail of the DataFrame.

```
import pandas as pd

prem_table = pd.read_html('https://www.bbc.co.uk/sport/football/premier-le
Premier_table = prem_table[0]

print(len(prem_table))
print(type(prem_table))

print(Premier_table.head(6))

# Output
1
<class 'list'>
```

	Unnamed: 0	Unnamed: 1	Team	P	W	D	L	F	A	GD	Pts	Form
0	1	team hasn't moved	Man City	34	28	2	4	87	22	65	86	WWon 3 - 1 against Watford on March 9th 2019.W...
1	2	team hasn't moved	Liverpool	34	26	7	1	77	20	57	85	WWon 4 - 2 against Burnley on March 10th 2019....
2	3	team hasn't moved	Tottenham	34	22	1	11	64	35	29	67	LLost 1 - 2 against Southampton on March 9th 2...
3	4	team hasn't moved	Arsenal	33	20	6	7	66	40	26	66	DDrew 1 - 1 against Tottenham Hotspur on March...
4	5	team hasn't moved	Chelsea	34	20	6	8	57	36	21	66	LLost 0 - 2 against Everton on March 17th 2019...
5	6	team hasn't moved	Man Utd	33	19	7	7	63	44	19	64	WWon 3 - 2 against Southampton on March 2nd 20...

As the task is to begin some exploratory analysis, I initially want to ‘clean’ the data. It seems unnecessary to have to column ‘Unnamed: 1’

in the DataFrame. To drop a column, I use the drop method, and pass in the keyword argument axis=1 for a *column-wise* drop.

```
Premier_table.columns
Index(['Unnamed: 0', 'Unnamed: 1', 'Team', 'P', 'W', 'D', 'L', 'F', 'A', '(
       'Pts', 'Form'],
      dtype='object')

Premier_league = Premier_table.drop(['Unnamed: 1'], axis=1)
Premier_league.head(6)
```

	Unnamed: 0	Team	P	W	D	L	F	A	GD	Pts	Form
0	1	Man City	34	28	2	4	87	22	65	86	WWon 3 - 1 against Watford on March 9th 2019.W...
1	2	Liverpool	34	26	7	1	77	20	57	85	WWon 4 - 2 against Burnley on March 10th 2019....
2	3	Tottenham	34	22	1	11	64	35	29	67	LLost 1 - 2 against Southampton on March 9th 2...
3	4	Arsenal	33	20	6	7	66	40	26	66	DDrew 1 - 1 against Tottenham Hotspur on March...
4	5	Chelsea	34	20	6	8	57	36	21	66	LLost 0 - 2 against Everton on March 17th 2019...
5	6	Man Utd	33	19	7	7	63	44	19	64	WWon 3 - 2 against Southampton on March 2nd 20...

The datatypes in the DataFrame

Each column in the DataFrame can be thought of a Series, where each column has to be of the same type. Each row however can be of different types. As the task involves performing some exploratory analysis, I need to decipher what type each Series is. A simple glance at the Table would inform me that the 'Team' column is a Python String. The pandas equivalent to a Python string is a pandas object. Furthermore, I would expect by visual inspection that the 'P' column denoting the number of games played is a Python float or int. However, assumptions aside, it is best to confirm this using either the

dtypes attribute or the **info** method.

```
Table.dtypes
```

```
Unnamed: 0    object
Team          object
P             object
W             object
D             object
L             object
F             object
A             object
GD            object
Pts           object
Form          object
dtype: object
```

This output here informs me that the dtypes of the Series are 'objects', i.e. Python Strings. These types will have to changed in order to perform numeric calculations between the different columns. Thankfully, this is an easy task in Python, we simply use the pandas **to_numeric** method. **pandas.to_numeric()** is one of the general functions in Pandas which is used to convert arguments to a numeric type. Here, I am changing the columns which I suspect I might like to analyse into the Pandas float (float64) data type. To confirm, I can check my DataFrames dtypes:

```
Table[['F', 'A', 'P', 'W', 'D', 'L', 'GD', 'Pts']]\
    = Table[['F', 'A', 'P','W', 'D', 'L', 'GD', 'Pts']].apply(pd.to_numeric, errors='coerce')
```

```
Table.dtypes
```

```
Unnamed: 0     object
Team           object
P             float64
W             float64
D             float64
L             float64
F             float64
A             float64
GD            float64
Pts           float64
Form           object
dtype: object
```

Cleaning the DataFrame

The shape attribute earlier told me I had one too many columns, assuming of course that 20 teams still compete in the English Premier League! As things looked good at the head of my DataFrame, the problem must reside at the tail.

```
Table.tail(3)
```

	Unnamed: 0	Team	P	W	D	L	F	A	GD	Pts	Form
18	19	Fulham	35.0	6.0	5.0	24.0	33.0	76.0	-43.0	23.0	LLost 1 - 2 against Liverpool on March 17th 20...
19	20	Huddersfield	35.0	3.0	5.0	27.0	20.0	69.0	-49.0	14.0	LLost 3 - 4 against West Ham United on March 1...
20	Last updated 20th April 2019 at 19:15	Last updated 20th April 2019 at 19:15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Last updated 20th April 2019 at 19:15

Clearly I have some unnecessary metadata, and a few NaN (not a number entries) populating the columns. Again, this time I can make use of the drop method, but importantly I omit the axis=1 parameter. I want to drop a row on this occasion, *not* a column. Things look much improved now. I can validate this row dropped successfully using the

shape attribute.

	Unnamed: 0	Team	P	W	D	L	F	A	GD	Pts	Form
17	18	Cardiff	34.0	9.0	4.0	21.0	30.0	63.0	-33.0	31.0	WWon 2 - 0 against West Ham United on March 9t...
18	19	Fulham	35.0	6.0	5.0	24.0	33.0	76.0	-43.0	23.0	LLost 1 - 2 against Liverpool on March 17th 20...
19	20	Huddersfield	35.0	3.0	5.0	27.0	20.0	69.0	-49.0	14.0	LLost 3 - 4 against West Ham United on March 1...

Whilst things look improved, it seems odd to have the team's league position column titled 'Unnamed: 0'. It is not representative of the contents of the column. To change this, I use the **Pandas Dataframe.rename()** method, and pass **inplace=True** which makes changes in the original DataFrame if True.

	Position	Team	P	W	D	L	F	A	GD	Pts	Form
0	1	Man City	34.0	28.0	2.0	4.0	87.0	22.0	65.0	86.0	WWon 3 - 1 against Watford on March 9th 2019.W...
1	2	Liverpool	34.0	26.0	7.0	1.0	77.0	20.0	57.0	85.0	WWon 4 - 2 against Burnley on March 10th 2019....
2	3	Tottenham	34.0	22.0	1.0	11.0	64.0	35.0	29.0	67.0	LLost 1 - 2 against Southampton on March 9th 2...
3	4	Arsenal	33.0	20.0	6.0	7.0	66.0	40.0	26.0	66.0	DDrew 1 - 1 against Tottenham Hotspur on March...
4	5	Chelsea	34.0	20.0	6.0	8.0	57.0	36.0	21.0	66.0	LLost 0 - 2 against Everton on March 17th 2019...

Creating New columns

The DataFrame is well set up now. At this point, it might be worthwhile creating some new columns and testing out whether the 'string-to-float' conversion I created earlier will work!

I have created a new columns called 'Goal Ratio'. This divides the number of goals scored against the number of goals conceded. To create a new column, I type the name of my DataFrame, then use

square bracket notation and input my *new* column title in quotations. On the right side of the equation, I select which columns I want from the Table DataFrame using the same technique. Here, I divide each column 'F' entry against each column 'A' entry and round to one decimal place. By default, each new column that is created is appended to the end of the DataFrame.

```
Table['Goal Ratio'] = round(Table['F']/Table['A'], 1)
```

```
Table.head(3)
```

	Position	Team	P	W	D	L	F	A	GD	Pts	Form	Goal Ratio
0	1	Man City	34.0	28.0	2.0	4.0	87.0	22.0	65.0	86.0	WWon 3 - 1 against Watford on March 9th 2019.W...	4.0
1	2	Liverpool	34.0	26.0	7.0	1.0	77.0	20.0	57.0	85.0	WWon 4 - 2 against Burnley on March 10th 2019....	3.8
2	3	Tottenham	34.0	22.0	1.0	11.0	64.0	35.0	29.0	67.0	LLost 1 - 2 against Southampton on March 9th 2...	1.8

```
Table.columns
```

```
Index(['Position', 'Team', 'P', 'W', 'D', 'L', 'F', 'A', 'GD', 'Pts', 'Form',
       'Goal Ratio'],
      dtype='object')
```

To re-order the columns, I simply change the order of the column entries and add them to a list, and re-assign this to the Table Dataframe. The 'Goal Ratio' column fits more appropriately now. I have also removed the 'Form' column to make the Table a little neater.

```
Table = Table[['Position', 'Team', 'P', 'W', 'D', 'L', 'F', 'A','Goal Ratio','GD', 'Pts']]
```

```
Table.head(3)
```

	Position	Team	P	W	D	L	F	A	Goal Ratio	GD	Pts
0	1	Man City	34.0	28.0	2.0	4.0	87.0	22.0	4.0	65.0	86.0
1	2	Liverpool	34.0	26.0	7.0	1.0	77.0	20.0	3.8	57.0	85.0
2	3	Tottenham	34.0	22.0	1.0	11.0	64.0	35.0	1.8	29.0	67.0

Filtering the DataFrame

The Table is well set up and amendable for analysis. Let's start by asking a simple question. Which teams have played one fewer games than their rivals? As teams have either played 34 or 35 games, I simply filter for teams that have played 34 games (as of 20/04/19) and the results appear. To filter, I use Boolean indexing. To see how to do this, read my article on how to 'How to Filter Rows of a Pandas DataFrame by Column Value'.

```
one_less_fixture = Table.P == 34
```

```
Table[one_less_fixture]
```

	Position	Team	P	W	D	L	F	A	Goal Ratio	GD	Pts
0	1	Man City	34.0	28.0	2.0	4.0	87.0	22.0	4.0	65.0	86.0
1	2	Liverpool	34.0	26.0	7.0	1.0	77.0	20.0	3.8	57.0	85.0
2	3	Tottenham	34.0	22.0	1.0	11.0	64.0	35.0	1.8	29.0	67.0
4	5	Chelsea	34.0	20.0	6.0	8.0	57.0	36.0	1.6	21.0	66.0
6	7	Watford	34.0	14.0	7.0	13.0	49.0	49.0	1.0	0.0	49.0
8	9	Wolves	34.0	13.0	9.0	12.0	41.0	42.0	1.0	-1.0	48.0
9	10	Everton	34.0	13.0	7.0	14.0	46.0	44.0	1.0	2.0	46.0
13	14	Crystal Palace	34.0	11.0	6.0	17.0	40.0	46.0	0.9	-6.0	39.0
14	15	Burnley	34.0	11.0	6.0	17.0	42.0	60.0	0.7	-18.0	39.0
15	16	Southampton	34.0	9.0	9.0	16.0	40.0	57.0	0.7	-17.0	36.0
16	17	Brighton	34.0	9.0	7.0	18.0	32.0	53.0	0.6	-21.0	34.0
17	18	Cardiff	34.0	9.0	4.0	21.0	30.0	63.0	0.5	-33.0	31.0

Now let's determine who has the most number of draws.

With Southampton attempting to avoid the drop this season, could all those draws be costly?

```
Table[(Table.D > 7)]
```

	Position	Team	P	W	D	L	F	A	Goal Ratio	GD	Pts
8	9	Wolves	34.0	13.0	9.0	12.0	41.0	42.0	1.0	-1.0	48.0
11	12	Newcastle	35.0	11.0	8.0	16.0	35.0	44.0	0.8	-9.0	41.0
15	16	Southampton	34.0	9.0	9.0	16.0	40.0	57.0	0.7	-17.0	36.0

Finally, lets create two new columns, 'Goals/game' and 'Goal conceded/game' and sort the values, using the **sort_values** method.

With Huddersfield and Fulham already condemned, Cardiff have the third worse Goal ratio, Goals conceded/game and second worst Goals/game. They also have a worst goal difference than their nearest relegation rival, Brighton.

With Cardiff faring the worse here, will they survive the drop? Did their recruits not possess enough of a goal threat, is their defense too vulnerable?

With Python's Pandas, we can find out!

```
Table.sort_values(['Goals/game', 'Goal Ratio']).head(5)
```

	Position	Team	P	W	D	L	F	A	Goal Ratio	GD	Pts	Goals/game	Goals conceded/game
19	20	Huddersfield	35.0	3.0	5.0	27.0	20.0	69.0	0.3	-49.0	14.0	0.6	2.0
18	19	Fulham	35.0	6.0	5.0	24.0	33.0	76.0	0.4	-43.0	23.0	0.9	2.2
17	18	Cardiff	34.0	9.0	4.0	21.0	30.0	63.0	0.5	-33.0	31.0	0.9	1.9
16	17	Brighton	34.0	9.0	7.0	18.0	32.0	53.0	0.6	-21.0	34.0	0.9	1.6
11	12	Newcastle	35.0	11.0	8.0	16.0	35.0	44.0	0.8	-9.0	41.0	1.0	1.3

```
goal_difference = Table.loc[17, 'GD'] - Table.loc[16, 'GD']
```

```
'The goal difference between Cardiff and Brighton is ' + str(goal_difference)
```

```
'The goal difference between Cardiff and Brighton is -12.0'
```


introductory analysis using Pandas. I enjoyed using pandas to delve into the Premier League stats. If you would like to follow me on

Data Science Programming Technology Artificial Intelligence Towards Data Science

Lastly, some advice. I was parsing this Table whilst games where being played and results were coming in. This gave me slightly different results when I ran the analysis, which confused me at first!

For a comfortable life, don't perform the analysis, whilst games are in progress and finishing, put your feet up and watch the game!