Image by Pexels from Pixabay

How to Access the Fantasy Premier League API, Build a Dataframe, and Analyze Using Jupyter, Python, and Pandas

Documentation for building a Pandas Dataframe from the FPL API, and running a value analysis on 19/20 season

David Allen Follow
Jun 15 · 10 min read ★

As of the writing of this tutorial, before match week 30+, I'm ranked #3,919 in the world in Fantasy Premier League Soccer (team: Yin Aubameyang), which equates to the top 0.05% in the world.

It didn't happen by accident, and it wasn't *all* luck.

It's taken several years of playing this game to learn the patience, skill, and strategy required to succeed. From #1,181,262 in 2011/12 to #39,804 in 2018/19, and now #3,919 in 2019/20 with 8 GWs left:

Previous Seasons

Season	Points	Rank
2011/12	1715	1181262
2012/13	1702	1427347
2014/15	1754	1349963
2015/16	1994	678723
2016/17	2151	132305
2017/18	2171	198743
2018/19	2321	39804

A screenshot of the author's previous season performance on fantasy.premierleague.com

Over these several years, I've learned 2 skills really help:

1. Identifying your biases
2. Understanding how to read data

The first will help you stop making stupid mistakes. The second will assist you in uncovering value that other fantasy managers are missing.

This tutorial will help develop these 2 skills by teaching you how to access new data. And then help you look at it in new ways.

This document is far from perfect, but at the very least, it will give you a taste of what is possible with Jupyter Notebooks, Pandas, Python, and a new data source.

By the end of this tutorial, you will have the basic skills to pull down data from the Fantasy Premier League API and work with the data to squeeze out new understandings.

Let's dive in.

. . .

Table of Contents:

. . .

Step 1: Getting Started

This tutorial assumes a little familiarity with Jupyter and Python. If you are just getting started, start with my tutorial here:

Getting Started with Python, Pandas, and Jupyter Notebooks

Documentation for everything you need to set up your machine with Jupyter Notebooks and start programming in Python...

medium.com

Once you complete your setup, come right back to this article!

If you already have Jupyter, Python, and Pandas installed then move on to step 2!

Step 2: Access the API with Requests

First, we'll need our package imports. We really only need 3:

```
import requests
import pandas as pd
import numpy as np
```

Note: If you get an import error, it's probably because you haven't added the package to your environment, or activated your development environment. Head over to your Anaconda Navigator and make sure to

add the package needed to whatever environment you activate for your Jupyter Notebook work. Apply the update, and don't forget to restart your terminal before you activate your development environment!

Next, we'll need the API URL:

```
url = 'https://fantasy.premierleague.com/api/bootstrap-static/'
```

Use the requests package to make a GET request from the API endpoint:

```
r = requests.get(url)
```

Then, transform that request into a json object:

```
json = r.json()
```

Let's look at the json keys, and then we'll create our dataframe(s)

```
json.keys()
```

This returns a list of all the keys to the json:

```
In [1]: import requests
        import pandas as pd
        import numpy as np

In [2]: pd.options.display.max_columns=None

In [3]: url = 'https://fantasy.premierleague.com/api/bootstrap-static/'

In [4]: r = requests.get(url)

In [5]: json = r.json()

In [6]: json.keys()

Out[6]: dict_keys(['events', 'game_settings', 'phases', 'teams', 'total_players', 'elements', 'element_stats', 'element_type
        s'])
```

A screenshot of the author's notebook results

The three keys I really care for this article are `elements` , `element_type` , and `teams` .

Next, we'll create three different dataframes using these three keys, and then map some columns from the `teams` and `element_type` dataframes into our `elements` dataframe.

Step 3: Build a DataFrame from your API request response

Once we build our dataframes, we'll be able to work with the data and do things like sort, filter, map (similar to a v-lookup in Excel), and organize our data into a pivot table.

So let's start by building our dataframes:

```
elements_df = pd.DataFrame(json['elements'])
elements_types_df = pd.DataFrame(json['element_types'])
```

```
teams_df = pd.DataFrame(json['teams'])
```

Preview the top 5 rows of your dataframes with the `head()` method. Like so:

```
elements_df.head()
```

```
elements_df.head()
```

	chance_of_playing_next_round	chance_of_playing_this_round	code	cost_change_event	cost_change_event_fall	cost_change_start	cost_change_start_fall	dr
0	100.0	100.0	69140	0	0	-4	4	
1	100.0	100.0	98745	0	0	0	0	
2	100.0	100.0	111457	0	0	-3	3	
3	100.0	100.0	154043	0	0	-5	5	
4	100.0	100.0	39476	0	0	-2	2	

A screenshot of the author's notebook results

You can also get a glimpse of all of the columns in each dataframe with the `columns` method:

```
elements_df.columns
```

```
elements_df.columns
```

```
Index(['chance_of_playing_next_round', 'chance_of_playing_this_round', 'code',
       'cost_change_event', 'cost_change_event_fall', 'cost_change_start',
       'cost_change_start_fall', 'dreamteam_count', 'element_type', 'ep_next',
       'ep_this', 'event_points', 'first_name', 'form', 'id', 'in_dreamteam',
       'news', 'news_added', 'now_cost', 'photo', 'points_per_game',
       'second_name', 'selected_by_percent', 'special', 'squad_number',
       'status', 'team', 'team_code', 'total_points', 'transfers_in',
       'transfers_in_event', 'transfers_out', 'transfers_out_event',
       'value_form', 'value_season', 'web_name', 'minutes', 'goals_scored',
       'assists', 'clean_sheets', 'goals_conceded', 'own_goals',
       'penalties_saved', 'penalties_missed', 'yellow_cards', 'red_cards',
       'saves', 'bonus', 'bps', 'influence', 'creativity', 'threat',
       'ict_index', 'influence_rank', 'influence_rank_type', 'creativity_rank',
       'creativity_rank_type', 'threat_rank', 'threat_rank_type',
       'ict_index_rank', 'ict_index_rank_type'],
      dtype='object')
```

A screenshot of the author's notebook results

There is an awful lot of data, so I'm going to make a copy of this dataframe, but only copy over some of the columns. I like to prepend `slim_` to these smaller, lighter dataframe copies:

```
slim_elements_df =
elements_df[['second_name','team','element_type','selected_by_p
ercent','now_cost','minutes','transfers_in','value_season','tot
al_points']]
```

Now, most of the data I'm currently interested in is viewable in a single window:

```
slim_elements_df.head()
```

```
slim_elements_df.head()
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points
0	Mustafi	1	2	0.4	51	620	16489	5.1	26
1	Bellerín	1	2	1.3	55	623	156619	4.5	25
2	Kolasinac	1	2	0.5	52	1103	57319	6.7	35
3	Maitland-Niles	1	2	2.3	45	1210	597196	8.0	36
4	Papastathopoulos	1	2	1.5	48	1696	162559	11.9	57

A screenshot of the author's notebook results

Success! Let's move on.

Step 4: Working with the Dataframe

This section will be broken into sub-sections:

- Using map()
- Using astype()
- Using sort_values()
- Using pivot_table()
- Using loc[]
- Using hist()

Using Map()

The first thing I'm going to do is map the position name from the `elements_type_df` to the `slim_elements_df` . If you've ever used Excel, you'll notice that `map()` is very similar to a v-lookup.

```
slim_elements_df['position'] =
slim_elements_df.element_type.map(elements_type_df.set_index('i
d').singular_name)
```

Now that we've done that, take a look at the top few rows:

```
slim_elements_df.head()
```

```
slim_elements_df.head()
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position
0	Mustafi	1	2	0.4	51	620	16491	5.1	26	Defender
1	Bellerín	1	2	1.3	55	623	156671	4.5	25	Defender
2	Kolasinac	1	2	0.5	52	1103	57327	6.7	35	Defender
3	Maitland-Niles	1	2	2.3	45	1210	597216	8.0	36	Defender
4	Papastathopoulos	1	2	1.5	48	1696	162609	11.9	57	Defender

A screenshot of the author's notebook results

Great, now we can easily see the position. This will come in handy when we organize value_season by position.

Next, let's add the team name. We can do that by mapping "name" from our teams_df:

```
slim_elements_df['team'] =
slim_elements_df.team.map(teams_df.set_index('id').name)
```

Voilà:

```
slim_elements_df.head()
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position
0	Mustafi	Arsenal	2	0.4	51	620	16489	5.1	26	Defender
1	Bellerín	Arsenal	2	1.3	55	623	156619	4.5	25	Defender
2	Kolasinac	Arsenal	2	0.5	52	1103	57319	6.7	35	Defender
3	Maitland-Niles	Arsenal	2	2.3	45	1210	597196	8.0	36	Defender
4	Papastathopoulos	Arsenal	2	1.5	48	1696	162559	11.9	57	Defender

A screenshot of the author's notebook results

Next, let's sort this table by `value_season`. I'm curious to look at the top value picks this year.

Using astype()

BUT FIRST, we have to create a new column called `value`.

Why? Because some of the values in the column `value_season` are string values. I know this because I've attempted to sort by `value_season` and got a strange, incorrect result. Unexplained behavior in your dataframe can often be due to conflicting types of data stored in the same column (for example: strings, integers, and float values all stored in the same column).

We need to make sure every single value is of the same type. To do this, we'll create a new column, and use the `astype()` method to set all of the values to a float value (fancy-talk for numbers with decimals, example: 4.5).

```
slim_elements_df['value'] =
slim_elements_df.value_season.astype(float)
```

Now we can sort our dataframe on the column `value`. Let's do it:

Using sort_values()

Sort_values() allows you to, well, sort values :)

By default, Pandas will sort values in the ascending direction from low to high. But we want values sorted in the descending direction: high to low. So we need to explicitly set the `ascending` parameter to `False` like so:

```
slim_elements_df.sort_values('value',ascending=False).head(10)
```

```
slim_elements_df.sort_values('value',ascending=False).head(10)
```

	second_name	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position	value
140	Pope	1	18.5	49	2610	1722481	26.1	128	Goalkeeper	26.1
440	Lundstram	2	39.8	49	2085	3798733	26.1	128	Defender	26.1
195	Guaita	1	2.1	50	2430	353752	23.2	116	Goalkeeper	23.2
377	Dubravka	1	1.8	51	2610	170904	23.1	118	Goalkeeper	23.1
453	Henderson	1	17.3	53	2430	1661752	23.0	122	Goalkeeper	23.0
543	Foster	1	4.1	49	2610	400952	22.7	111	Goalkeeper	22.7
437	Baldock	2	11.7	51	2520	1376609	22.5	115	Defender	22.5
607	dos Santos Patrício	1	9.8	52	2610	1025917	22.3	116	Goalkeeper	22.3
256	Schmeichel	1	14.0	54	2610	1278530	22.0	119	Goalkeeper	22.0
278	van Dijk	2	44.1	65	2610	1780244	21.7	141	Defender	21.7

A screenshot of the author's notebook results

Well, will ya look at that? Two players tied for the top spot. Pope and Lundstram. A goalkeeper and a defender. Nice.

This essentially means Pope and Lundstram have earned the most points per cost, because value is simply calculated by dividing `total_points` by `now_cost` .

$$value = points / cost$$

Did you expect that to be the case? A defender and a goalkeeper? Might this change how you think about team selection next year? Or next week?

Using pivot_table()

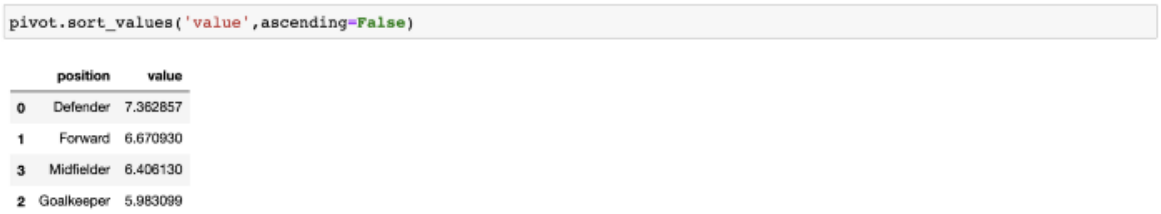
Next, we'll create a pivot_table on the column `position` , and look at value-by-position:

```
slim_elements_df.pivot_table(index='position',values='value',ag
gfunc=np.mean).reset_index()
```

After creating the pivot_table and assigning the pivot_table to the variable `pivot` , we'll sort the pivot table descending by value:

```
pivot.sort_values('value',ascending=False)
```

This gives us:



```
pivot.sort_values('value',ascending=False)
```

	position	value
0	Defender	7.362857
1	Forward	6.670930
3	Midfielder	6.406130
2	Goalkeeper	5.983099

A screenshot of the author's notebook results

Interesting. One of the two highest-value players in the game is a goalkeeper, yet the goalkeeper position contributes the lowest value on average. I think this is likely because all the goalkeepers that play 0 minutes. bring down the average of the starters.

To address this, let's remove all the players from our dataframe that have zero minutes this season. We'll use `loc[]` for the first time:

Using .loc[]

.loc lets you locate specific rows that have specific column values. It's like filtering a spreadsheet on a specific value in a column.

For our current purposes, we want to locate all rows in our dataframe that have a value greater than zero so as to remove all values = 0. We aren't just doing this for keepers. We need to do this for every

position. We write:

```
slim_elements_df = slim_elements_df.loc[slim_elements_df.value
> 0]
```

Now let's run that pivot table back again:

```
pivot =
slim_elements_df.pivot_table(index='position',values='value',ag
gfunc=np.mean).reset_index()
```

and then:

```
pivot.sort_values('value',ascending=False)
```

There you have it:

```
pivot.sort_values('value',ascending=False)
```

	position	value
2	Goalkeeper	11.800000
0	Defender	9.203571
1	Forward	7.968056
3	Midfielder	7.886792

A screenshot of the author's notebook results

Now let's do a different type of pivot table next. This time, we'll look at which teams are providing the most value this year. The only things

that need to change about this pivot_table statement are the index and the pivot_table variable. Take a look:

```
team_pivot =
slim_elements_df.pivot_table(index='team',values='value',aggfun
c=np.mean).reset_index()
```

Now we'll display the team_pivot sorted, from the highest value team to the lowest:

```
team_pivot.sort_values('value',ascending=False)
```

Results are as follows:

```
team_pivot.sort_values('value',ascending=False)
```

	team	value
19	Wolves	11.542105
8	Leicester	11.304545
14	Sheffield Utd	11.109091
4	Burnley	11.055000
9	Liverpool	10.113043
6	Crystal Palace	9.079167
15	Southampton	8.977273
10	Man City	8.904545
12	Newcastle	8.707407
3	Brighton	8.624000
5	Chelsea	8.564000
17	Watford	8.380000
7	Everton	8.359091
11	Man Utd	7.539286
18	West Ham	7.500000
13	Norwich	7.488462
2	Bournemouth	7.468000
0	Arsenal	7.167857
1	Aston Villa	7.114286
16	Spurs	6.618519

A screenshot of the author's notebook results

Wolves fans don't have to worry about their innate biases. Lucky!

Spurs fans, on the other hand, have had to be a little more conscious of their bias towards their own team to succeed in the FPL this year. How stressful.

Using .hist()

Next, let's look at a histogram distribution of value by position. Before we do that, we'll create some filtered dataframes for each position. We will use .loc again to accomplish this:

```
fwd_df = slim_elements_df.loc[slim_elements_df.position ==
'Forward']

mid_df = slim_elements_df.loc[slim_elements_df.position ==
'Midfielder']

def_df = slim_elements_df.loc[slim_elements_df.position ==
'Defender']

goal_df = slim_elements_df.loc[slim_elements_df.position ==
'Goalkeeper']
```

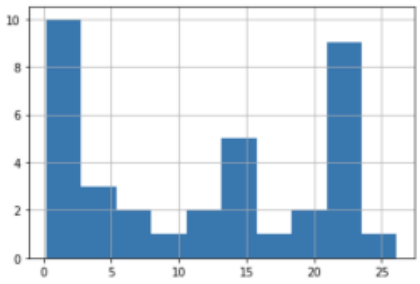
Then, we'll use the `.hist()` method on our `goal_df.value` dataframe column:

```
goal_df.value.hist()
```

This gives us:

```
goal_df.value.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12409cb10>
```



A screenshot of the author's notebook results

There is a decent amount of value here above 20. Let's look at our dataframe, sorted by value to see who these characters are:

```
goal_df.sort_values('value',ascending=False).head(10)
```

```
goal_df.sort_values('value',ascending=False).head(10)
```

	second_name	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position	value
140	Pope	1	18.5	49	2610	1722481	26.1	128	Goalkeeper	26.1
195	Guaita	1	2.1	50	2430	353752	23.2	116	Goalkeeper	23.2
377	Dubravka	1	1.8	51	2610	170904	23.1	118	Goalkeeper	23.1
453	Henderson	1	17.3	53	2430	1661752	23.0	122	Goalkeeper	23.0
543	Foster	1	4.1	49	2610	400952	22.7	111	Goalkeeper	22.7
607	dos Santos Patrício	1	9.8	52	2610	1025917	22.3	116	Goalkeeper	22.3
256	Schmeichel	1	14.0	54	2610	1278530	22.0	119	Goalkeeper	22.0
108	Ryan	1	13.4	47	2610	1144045	21.7	102	Goalkeeper	21.7
13	Leno	1	6.0	50	2520	534244	21.6	108	Goalkeeper	21.6
93	Ramsdale	1	3.6	45	2520	681752	21.3	96	Goalkeeper	21.3

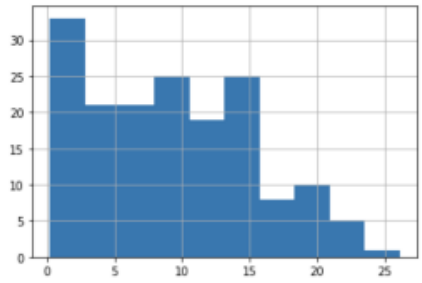
A screenshot of the author's notebook results

Take-away? There's a lot of value available in the goalkeeper position.

Plenty of other ways to cut this cheese, but let's move on to histograms of the defender position:

```
def_df.value.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11cc99310>
```



A screenshot of the author's notebook results

Our defensive histogram points out something interesting. Notice the difference between the goalkeeper histogram and the defender histogram?

For a goalkeeper, there is a "best" choice, but we don't really know who that is going to be starting the season. It turns out, goalkeeper selection was pretty forgiving seeing how many goalkeepers offered value between 20 and 25:

```
goal_df.sort_values('value',ascending=False).head(10)
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position	value
140	Pope	Burnley	1	18.4	49	2610	1740813	26.1	128	Goalkeeper	26.1
195	Guaita	Crystal Palace	1	2.0	50	2430	355326	23.2	116	Goalkeeper	23.2
377	Dubravka	Newcastle	1	1.8	51	2610	174431	23.1	118	Goalkeeper	23.1
453	Henderson	Sheffield Utd	1	17.7	53	2430	1701795	23.0	122	Goalkeeper	23.0
543	Foster	Watford	1	4.0	49	2610	402879	22.7	111	Goalkeeper	22.7
607	dos Santos Patrício	Wolves	1	9.7	52	2610	1032428	22.3	116	Goalkeeper	22.3
256	Schmeichel	Leicester	1	13.9	54	2610	1282172	22.0	119	Goalkeeper	22.0
108	Ryan	Brighton	1	13.2	47	2610	1147253	21.7	102	Goalkeeper	21.7
13	Leno	Arsenal	1	6.3	50	2520	559444	21.6	108	Goalkeeper	21.6
93	Ramsdale	Bournemouth	1	3.6	45	2520	689018	21.3	96	Goalkeeper	21.3

A screenshot of the author's notebook results

Defenders, on the other hand, had just one clear winner this year, and only a few others in the 20–25 value range that stood out from the pack. Let's look at the top of the defender value table:

```
def_df.sort_values('value',ascending=False).head(10)
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position	value
440	Lundstram	Sheffield Utd	2	40.2	49	2065	3839125	26.1	128	Defender	26.1
437	Baldock	Sheffield Utd	2	11.9	51	2520	1406803	22.5	115	Defender	22.5
278	van Dijk	Liverpool	2	44.0	65	2610	1795348	21.7	141	Defender	21.7
277	Alexander-Arnold	Liverpool	2	43.8	78	2549	2802716	21.3	166	Defender	21.3
434	Stevens	Sheffield Utd	2	3.4	52	2445	478241	21.2	110	Defender	21.2
252	Söyüncü	Leicester	2	15.8	49	2520	2193464	21.0	103	Defender	21.0
438	Egan	Sheffield Utd	2	3.0	46	2418	306440	20.4	94	Defender	20.4
436	O'Connell	Sheffield Utd	2	3.9	47	2520	481381	20.4	96	Defender	20.4
250	Evans	Leicester	2	6.2	53	2574	795099	20.4	108	Defender	20.4
447	Basham	Sheffield Utd	2	1.5	46	2432	168733	19.6	90	Defender	19.6

A screenshot of the author's notebook results

Lundstram was THE essential defender this year, offering a huge amount of value per cost: 26.1. Impressive!

The most important takeaway from this whole analysis is that Goal Keepers and Defenders offer more value than Midfielders and

Forwards. But if you review FPL Twitter, you'll see many FPL managers that don't quite grasp this insight. That, my friend, is called "opportunity."

It's not *all* about value of course. It's a team optimization challenge. You have 100 currency units to spend, and a team of high-value returners may not score enough points to challenge for a high finish in your mini-leagues or overall. A team of the highest-value returners will leave you with some money in the bank. And the game is to leverage your money for POINTS, not VALUE.

The End

Let's look at the top midfielders, for example:

There is a lot more to do here, but as with a good lunch, it's important to not overall eat the sandwich whole. We'll be back with more often here in this tutorial? Share you request in a comment here or reach out on twitter @ deallen7

```
mid_df.sort_values('value',ascending=False).head(10)
```

	second_name	team	element_type	selected_by_percent	now_cost	minutes	transfers_in	value_season	total_points	position	value
418	Cantwell	Norwich	3	19.7	47	2067	3670901	21.3	100	Midfielder	21.3
613	Traoré	Wolves	3	16.7	57	2068	2642804	18.9	108	Midfielder	18.9
444	Fleck	Sheffield Utd	3	6.3	50	2188	908581	18.8	94	Midfielder	18.8
293	Henderson	Liverpool	3	1.6	53	1883	296850	18.7	99	Midfielder	18.7
41	Grealish	Aston Villa	3	22.5	64	2333	2882331	18.4	118	Midfielder	18.4
260	Barnes	Leicester	3	4.1	62	1705	565356	18.1	112	Midfielder	18.1
180	Mount	Chelsea	3	15.4	62	2296	4062576	17.9	111	Midfielder	17.9
269	Pérez	Leicester	3	7.5	62	1578	1281159	17.7	110	Midfielder	17.7
611	Santos Moutinho	Wolves	3	5.1	54	2434	620494	17.4	94	Midfielder	17.4
319	De Bruyne	Man City	3	47.5	106	2148	5405926	16.8	178	Midfielder	16.8

A screenshot of the author's notebook results

Cantwell offers the highest value out of all midfielders, but he still has just 100 points. Meanwhile, De Bruyne has a value of just 16.8, but 178 total points.

Cheers.

techniques delivered Monday to Thursday. Make learning your daily ritual. Take a look

Get this newsletter

Emails will be sent to enunenun21@gmail.com.
Not you?

Data Analysis Programming Python Data Science Soccer Analytics

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

About Help Legal