labassignment3

June 29, 2022

1 Lab Assignment 3: How to Load, Convert, and Write JSON Files in Python

1.1 DS 6001: Practice and Application of Data Science

1.1.1 Instructions

Please answer the following questions as completely as possible using text, code, and the results of code as needed. Format your answers in a Jupyter notebook. To receive full credit, make sure you address every part of the problem, and make sure your document is formatted in a clean and professional way.

1.2 Problem 0

Import the following libraries:

[1]:
```
import numpy as np
import pandas as pd
import requests
import json
import sys
sys.tracebacklimit = 0 # turn off the error tracebacks
```

1.3 Problem 1

JSON and CSV are both text-based formats for the storage of data. It's possible to open either one in a plain text editor. Given this similarity, why does a CSV file usually take less memory than a JSON formatted file for the same data? Under what conditions could a JSON file be smaller in memory than a CSV file for the same data? (2 points)

JSON files take more memory becuase they contain metadata. For examaple each enrty will have Name: xxx when a CSV file will only have name once as a header and then a list of names. This factor makes the CSV file take less memory because there are less characters to store. A JSON would be smaller than a CSV when missing values are involved because CSV stores them as NAN and JSON omits them all together.

1.4 Problem 2

NASA has a dataset of all meteorites that have fallen to Earth between the years A.D. 860 and 2013. The data contain the name of each meteorite, along with the coordinates of the place where

the meteorite hit, the mass of the meteorite, and the date of the collison. The data is stored as a JSON here: https://data.nasa.gov/resource/y77d-th95.json

Look at the data in your web-browser and explain which strategy for loading the JSON into Python makes the most sense and why.

Then write and run the code that will work for loading the data into Python. (2 points)

The best way to load the JSON is the process we used earlier in class with no record_path specified in the pd.json_normalize function becuase there is no path record.

[7]:
```
url = 'https://data.nasa.gov/resource/y77d-th95.json'
user = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,␣
 ↪like Gecko) Chrome/102.0.0.0 Safari/537.36"
r = requests.get(url,headers = {'User-agent':user})
r
```

[7]:
```
<Response [200]>
```

[9]:
```
myjson = json.loads(r.text)
mydf = pd.json_normalize(myjson)
mydf
```

[9]:
```
          name     id nametype                recclass    mass  fall  \
0       Aachen      1    Valid                      L5      21  Fell
1       Aarhus      2    Valid                      H6     720  Fell
2         Abee      6    Valid                     EH4  107000  Fell
3     Acapulco     10    Valid             Acapulcoite    1914  Fell
4      Achiras    370    Valid                      L6     780  Fell
..         …    …       …                        …     …   …
995   Tirupati  24009    Valid                      H6     230  Fell
996    Tissint  54823    Valid  Martian (shergottite)    7000  Fell
997      Tjabe  24011    Valid                      H6   20000  Fell
998   Tjerebon  24012    Valid                      L5   16500  Fell
999  Tomakovka  24019    Valid                     LL6     600  Fell

                         year      reclat      reclong geolocation.type  \
0    1880-01-01T00:00:00.000   50.775000     6.083330            Point
1    1951-01-01T00:00:00.000   56.183330    10.233330            Point
2    1952-01-01T00:00:00.000   54.216670  -113.000000            Point
3    1976-01-01T00:00:00.000   16.883330   -99.900000            Point
4    1902-01-01T00:00:00.000  -33.166670   -64.950000            Point
..                       …         …          …               …
995  1934-01-01T00:00:00.000   13.633330    79.416670            Point
996  2011-01-01T00:00:00.000   29.481950    -7.611230            Point
997  1869-01-01T00:00:00.000   -7.083330   111.533330            Point
998  1922-01-01T00:00:00.000   -6.666670   106.583330            Point
999  1905-01-01T00:00:00.000   47.850000    34.766670            Point
```

```
    geolocation.coordinates :@computed_region_cbhk_fwbd  \
0          [6.08333, 50.775]                          NaN
1        [10.23333, 56.18333]                         NaN
2            [-113, 54.21667]                         NaN
3           [-99.9, 16.88333]                         NaN
4         [-64.95, -33.16667]                         NaN
..                        …                          …
995      [79.41667, 13.63333]                         NaN
996      [-7.61123, 29.48195]                         NaN
997     [111.53333, -7.08333]                         NaN
998     [106.58333, -6.66667]                         NaN
999         [34.76667, 47.85]                         NaN

    :@computed_region_nnqa_25f4
0                           NaN
1                           NaN
2                           NaN
3                           NaN
4                           NaN
..                          …
995                         NaN
996                         NaN
997                         NaN
998                         NaN
999                         NaN

[1000 rows x 13 columns]
```

1.5 Problem 3

The textbook chapter for this module shows, as an example, how to pull data in JSON format from Reddit's top 25 posts on /r/popular. The steps outlined there pull all of the features in the data into the dataframe, resulting in a dataframe with 172 columns.

If we only wanted a few features, then looping across elements of the JSON list itself and extracting only the data we want may be a more efficient approach.

Use looping - and not `pd.read_json()` or `pd.json_normalize()` - to create a dataframe with 25 rows (one for each of the top 25 posts), and only columns for `subreddit`, `title`, `ups`, and `created_utc`. The JSON file exists at http://www.reddit.com/r/popular/top.json, and don't forget to specify `headers = {'User-agent': 'DS6001'}` within `requests.get()`. (3 points)

[15]:
```
url2 = 'http://www.reddit.com/r/popular/top.json'
user = "DS6001"
r2 = requests.get(url2,headers = {'User-agent':user})
r2
```

[15]:
```
<Response [200]>
```

[29]:
```
myjson2 = json.loads(r2.text)
columns = ['subreddit','title','ups','created_utc']
df = pd.DataFrame()
ct = 0
for col in columns:
    df[ct] = [ i['data'][col] for i in myjson2['data']['children']]
    ct = ct + 1
df.columns = columns
df
```

[29]:

	subreddit	title	\
0	MadeMeSmile	True freedom …	
1	oddlysatisfying	Easy trick to clean cut carpet flooring around…	
2	WhitePeopleTwitter	Lake Superior hasn't wrecked anyone like this …	
3	worldnews	Supreme Court of New Zealand rules "Family Fir…	
4	interestingasfuck	Drone footage of a dairy farm	
5	memes	"World Peace has been solved…"	
6	HumansBeingBros	Tennis Player Jodie Burrage stopping her Wimbl…	
7	pics	A scene from the reproductive rights marches t…	
8	todayilearned	TIL that during World War II, the United State…	
9	Tinder	this has to be a new low	
10	wholesomememes	They always encourage me no matter what	
11	comics	Doctor Visit [OC]	
12	mildlyinteresting	These urinals where you can look out to the st…	
13	news	8-year-old Florida boy accidentally shoots and…	
14	meirl	me irl	
15	FunnyAnimals	A masterpiece	
16	BlackPeopleTwitter	Give this person a raise.	
17	interestingasfuck	Congobubinga wood has a distinct Red/Pink colo…	
18	coolguides	Just a friendly reminder	
19	MadeMeSmile	The way his face lit up	
20	gaming	One of the most random and downright hilarious…	
21	memes	He did a science	
22	MadeMeSmile	That was his goal!!! He’s having a great day m…	
23	IdiotsInCars	My dad just barely avoided this drunk idiot	
24	AskReddit	If having sex on a plane is called joining the…	

	ups	created_utc
0	117667	1.656366e+09
1	116771	1.656381e+09
2	90969	1.656367e+09
3	82087	1.656375e+09
4	81175	1.656368e+09
5	76339	1.656401e+09
6	70338	1.656366e+09

```
7    68349  1.656371e+09
8    58434  1.656371e+09
9    56662  1.656380e+09
10   54418  1.656362e+09
11   52758  1.656381e+09
12   52393  1.656371e+09
13   51911  1.656365e+09
14   52152  1.656392e+09
15   51699  1.656415e+09
16   47121  1.656356e+09
17   47330  1.656395e+09
18   47430  1.656384e+09
19   49707  1.656414e+09
20   44643  1.656360e+09
21   48129  1.656420e+09
22   43910  1.656373e+09
23   43447  1.656380e+09
24   42661  1.656366e+09
```

1.6 Problem 4

The NBA has saved data on all 30 teams' shooting statistics for the 2014-2015 season here: https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json. Take a moment and look at this JSON file in your web browser. The structure of this particular JSON is complicated, but see if you can find the team-by-team data. In this problem our goal is to use `pd.json_normalize()` to get the data into a dataframe. The following questions will guide you towards this goal.

1.6.1 Part a

Download the raw text of the NBA JSON file and register it as JSON formatted data in Python's memory. (2 points)

[34]:
```
url3 = 'https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json'
r3 = requests.get(url3,headers = {'User-agent':user})
r3
myjson3 = json.loads(r3.text)
```

1.6.2 Part b

Describe, in words, the path that leads to the team-by-team data. (2 points)

[ ]:
```
resultSets -> 0 -> rowSet
```

1.6.3 Part c

Use the `pd.json_normalize()` function to pull the team-by-team data into a dataframe. This is going to be tricky. You will need to use indexing on the JSON data as well as the `record_path` parameter.

If you are successful, you will have a dataframe with 30 rows and 33 columns. The first row will refer to the Golden State Warriors, the second row will refer to the San Antonio Spurs, and the third row will refer to the Cleveland Cavaliers. The columns will only be named 0, 1, 2, … at this point. (4 points)

[43]:
```
newjson = myjson3['resultSets']
df = pd.json_normalize(newjson, record_path= ['rowSet'])
df
```

[43]:

	0	1	2	3	4	5	6	7	8	\
0	1610612744	Golden State	Warriors	GSW		82	48.7	114.9	14.9	
1	1610612759	San Antonio	Spurs	SAS		82	48.3	103.5	14.8	
2	1610612739	Cleveland	Cavaliers	CLE		82	48.7	104.3	16.9	
3	1610612746	Los Angeles	Clippers	LAC		82	48.6	104.5	15.0	
4	1610612760	Oklahoma City	Thunder	OKC		82	48.6	110.2	16.1	
5	1610612737	Atlanta	Hawks	ATL		82	48.6	102.8	19.0	
6	1610612745	Houston	Rockets	HOU		82	48.6	106.5	17.2	
7	1610612757	Portland	Trail Blazers	POR		82	48.5	105.1	17.5	
8	1610612758	Sacramento	Kings	SAC		81	48.4	106.7	18.7	
9	1610612764	Washington	Wizards	WAS		82	48.5	104.1	15.4	
10	1610612748	Miami	Heat	MIA		82	48.6	100.0	17.9	
11	1610612761	Toronto	Raptors	TOR		81	48.5	102.7	23.0	
12	1610612742	Dallas	Mavericks	DAL		82	49.0	102.3	18.2	
13	1610612766	Charlotte	Hornets	CHA		82	48.6	103.4	16.8	
14	1610612762	Utah	Jazz	UTA		82	49.0	97.7	18.1	
15	1610612753	Orlando	Magic	ORL		81	48.7	102.0	18.0	
16	1610612749	Milwaukee	Bucks	MIL		82	48.7	99.0	17.4	
17	1610612740	New Orleans	Pelicans	NOP		82	48.5	102.7	19.9	
18	1610612750	Minnesota	Timberwolves	MIN		82	48.6	102.4	15.1	
19	1610612754	Indiana	Pacers	IND		82	48.8	102.2	13.7	
20	1610612751	Brooklyn	Nets	BKN		82	48.4	98.6	14.4	
21	1610612765	Detroit	Pistons	DET		82	48.7	102.0	17.5	
22	1610612743	Denver	Nuggets	DEN		82	48.6	101.9	15.9	
23	1610612738	Boston	Celtics	BOS		81	48.5	105.6	18.9	
24	1610612741	Chicago	Bulls	CHI		82	48.9	101.6	18.1	
25	1610612755	Philadelphia	76ers	PHI		82	48.6	97.4	19.7	
26	1610612756	Phoenix	Suns	PHX		82	48.4	100.9	15.6	
27	1610612752	New York	Knicks	NYK		82	48.5	98.4	10.4	
28	1610612763	Memphis	Grizzlies	MEM		82	48.6	99.1	16.4	
29	1610612747	Los Angeles	Lakers	LAL		82	48.3	97.3	15.6	

	9	…	23	24	25	26	27	28	29	30	31	32
0	0.498	…	0.478	21.2	42.5	0.497	2.3	6.3	0.363	10.8	25.3	0.429
1	0.481	…	0.506	18.3	39.8	0.460	0.9	2.6	0.341	6.1	15.9	0.381
2	0.481	…	0.473	18.2	40.7	0.447	1.7	5.7	0.299	9.0	23.9	0.378
3	0.497	…	0.480	18.9	42.0	0.450	2.0	6.0	0.334	7.7	20.8	0.373
4	0.480	…	0.497	17.5	38.7	0.451	1.6	5.1	0.321	6.6	18.6	0.356

```
5   0.463  …  0.483  19.4  44.6  0.435  1.0  3.1  0.311   9.0  25.3  0.355
6   0.433  …  0.472  15.5  36.4  0.426  2.3  7.4  0.318   8.4  23.5  0.355
7   0.441  …  0.447  18.0  39.8  0.453  1.7  5.9  0.295   8.8  22.6  0.389
8   0.452  …  0.473  18.1  39.7  0.454  0.9  3.1  0.276   7.2  19.4  0.372
9   0.480  …  0.483  19.5  44.3  0.439  0.7  2.7  0.254   8.0  21.5  0.371
10  0.488  …  0.490  15.7  35.2  0.445  0.8  2.9  0.282   5.3  15.1  0.347
11  0.462  …  0.461  14.1  32.4  0.436  1.8  5.6  0.327   6.8  17.7  0.384
12  0.473  …  0.464  17.5  41.4  0.423  1.4  5.3  0.273   8.4  23.3  0.360
13  0.459  …  0.449  17.0  39.8  0.427  1.8  6.0  0.297   8.9  23.4  0.379
14  0.445  …  0.468  15.9  37.2  0.426  1.4  4.3  0.318   7.1  19.5  0.363
15  0.456  …  0.475  18.5  42.6  0.435  0.7  2.7  0.249   7.1  19.5  0.363
16  0.463  …  0.477  13.2  29.4  0.448  1.1  4.0  0.270   4.3  11.6  0.370
17  0.458  …  0.460  17.9  41.1  0.434  0.6  2.6  0.247   7.9  21.2  0.374
18  0.464  …  0.471  16.1  35.4  0.455  0.7  2.6  0.272   4.8  13.8  0.350
19  0.453  …  0.465  16.4  38.1  0.431  1.7  5.7  0.299   6.4  17.4  0.368
20  0.457  …  0.464  15.8  36.1  0.438  1.0  3.3  0.303   5.5  15.1  0.363
21  0.464  …  0.452  15.7  37.2  0.422  0.9  4.0  0.227   8.1  22.2  0.366
22  0.406  …  0.448  16.4  37.8  0.434  1.1  4.3  0.264   6.9  19.5  0.354
23  0.453  …  0.451  16.9  39.9  0.424  1.6  5.7  0.274   7.1  20.3  0.350
24  0.458  …  0.442  17.0  38.5  0.441  1.3  3.9  0.332   6.6  17.5  0.380
25  0.445  …  0.449  15.3  37.4  0.409  1.6  5.7  0.281   7.7  21.8  0.354
26  0.440  …  0.447  16.6  39.5  0.421  1.4  5.0  0.288   7.6  20.8  0.363
27  0.447  …  0.439  15.9  36.4  0.438  1.5  4.9  0.305   5.9  16.6  0.358
28  0.440  …  0.459  16.1  38.5  0.418  0.7  2.5  0.278   5.4  16.0  0.340
29  0.441  …  0.420  14.0  34.5  0.406  2.2  7.9  0.278   5.6  16.7  0.335

[30 rows x 33 columns]
```

1.6.4 Part d

Find the path that leads to the headers (the column names), and extract these names as a list. Then set the `.columns` attribute of the dataframe you created in part c equal to this list. The result should be that the dataframe now has the correct column names. (3 points)

[50]:
```
newjson2 = myjson3['resultSets']
cols = newjson2[0]['headers']
df.columns = cols
df
```

[50]:
```
       TEAM_ID      TEAM_CITY      TEAM_NAME TEAM_ABBREVIATION TEAM_CODE  GP  \
0   1610612744   Golden State       Warriors               GSW            82
1   1610612759    San Antonio          Spurs               SAS            82
2   1610612739      Cleveland      Cavaliers               CLE            82
3   1610612746    Los Angeles       Clippers               LAC            82
4   1610612760  Oklahoma City        Thunder               OKC            82
5   1610612737        Atlanta          Hawks               ATL            82
6   1610612745        Houston        Rockets               HOU            82
7   1610612757       Portland  Trail Blazers               POR            82
```

8	1610612758	Sacramento	Kings	SAC	81	
9	1610612764	Washington	Wizards	WAS	82	
10	1610612748	Miami	Heat	MIA	82	
11	1610612761	Toronto	Raptors	TOR	81	
12	1610612742	Dallas	Mavericks	DAL	82	
13	1610612766	Charlotte	Hornets	CHA	82	
14	1610612762	Utah	Jazz	UTA	82	
15	1610612753	Orlando	Magic	ORL	81	
16	1610612749	Milwaukee	Bucks	MIL	82	
17	1610612740	New Orleans	Pelicans	NOP	82	
18	1610612750	Minnesota	Timberwolves	MIN	82	
19	1610612754	Indiana	Pacers	IND	82	
20	1610612751	Brooklyn	Nets	BKN	82	
21	1610612765	Detroit	Pistons	DET	82	
22	1610612743	Denver	Nuggets	DEN	82	
23	1610612738	Boston	Celtics	BOS	81	
24	1610612741	Chicago	Bulls	CHI	82	
25	1610612755	Philadelphia	76ers	PHI	82	
26	1610612756	Phoenix	Suns	PHX	82	
27	1610612752	New York	Knicks	NYK	82	
28	1610612763	Memphis	Grizzlies	MEM	82	
29	1610612747	Los Angeles	Lakers	LAL	82	

	MIN	PTS	PTS_DRIVE	FGP_DRIVE	…	CFGP	UFGM	UFGA	UFGP	CFG3M	\
0	48.7	114.9	14.9	0.498	…	0.478	21.2	42.5	0.497	2.3	
1	48.3	103.5	14.8	0.481	…	0.506	18.3	39.8	0.460	0.9	
2	48.7	104.3	16.9	0.481	…	0.473	18.2	40.7	0.447	1.7	
3	48.6	104.5	15.0	0.497	…	0.480	18.9	42.0	0.450	2.0	
4	48.6	110.2	16.1	0.480	…	0.497	17.5	38.7	0.451	1.6	
5	48.6	102.8	19.0	0.463	…	0.483	19.4	44.6	0.435	1.0	
6	48.6	106.5	17.2	0.433	…	0.472	15.5	36.4	0.426	2.3	
7	48.5	105.1	17.5	0.441	…	0.447	18.0	39.8	0.453	1.7	
8	48.4	106.7	18.7	0.452	…	0.473	18.1	39.7	0.454	0.9	
9	48.5	104.1	15.4	0.480	…	0.483	19.5	44.3	0.439	0.7	
10	48.6	100.0	17.9	0.488	…	0.490	15.7	35.2	0.445	0.8	
11	48.5	102.7	23.0	0.462	…	0.461	14.1	32.4	0.436	1.8	
12	49.0	102.3	18.2	0.473	…	0.464	17.5	41.4	0.423	1.4	
13	48.6	103.4	16.8	0.459	…	0.449	17.0	39.8	0.427	1.8	
14	49.0	97.7	18.1	0.445	…	0.468	15.9	37.2	0.426	1.4	
15	48.7	102.0	18.0	0.456	…	0.475	18.5	42.6	0.435	0.7	
16	48.7	99.0	17.4	0.463	…	0.477	13.2	29.4	0.448	1.1	
17	48.5	102.7	19.9	0.458	…	0.460	17.9	41.1	0.434	0.6	
18	48.6	102.4	15.1	0.464	…	0.471	16.1	35.4	0.455	0.7	
19	48.8	102.2	13.7	0.453	…	0.465	16.4	38.1	0.431	1.7	
20	48.4	98.6	14.4	0.457	…	0.464	15.8	36.1	0.438	1.0	
21	48.7	102.0	17.5	0.464	…	0.452	15.7	37.2	0.422	0.9	
22	48.6	101.9	15.9	0.406	…	0.448	16.4	37.8	0.434	1.1	

```
23  48.5  105.6       18.9      0.453  …  0.451  16.9  39.9  0.424    1.6
24  48.9  101.6       18.1      0.458  …  0.442  17.0  38.5  0.441    1.3
25  48.6   97.4       19.7      0.445  …  0.449  15.3  37.4  0.409    1.6
26  48.4  100.9       15.6      0.440  …  0.447  16.6  39.5  0.421    1.4
27  48.5   98.4       10.4      0.447  …  0.439  15.9  36.4  0.438    1.5
28  48.6   99.1       16.4      0.440  …  0.459  16.1  38.5  0.418    0.7
29  48.3   97.3       15.6      0.441  …  0.420  14.0  34.5  0.406    2.2

    CFG3A  CFG3P  UFG3M  UFG3A  UFG3P
0     6.3  0.363   10.8   25.3  0.429
1     2.6  0.341    6.1   15.9  0.381
2     5.7  0.299    9.0   23.9  0.378
3     6.0  0.334    7.7   20.8  0.373
4     5.1  0.321    6.6   18.6  0.356
5     3.1  0.311    9.0   25.3  0.355
6     7.4  0.318    8.4   23.5  0.355
7     5.9  0.295    8.8   22.6  0.389
8     3.1  0.276    7.2   19.4  0.372
9     2.7  0.254    8.0   21.5  0.371
10    2.9  0.282    5.3   15.1  0.347
11    5.6  0.327    6.8   17.7  0.384
12    5.3  0.273    8.4   23.3  0.360
13    6.0  0.297    8.9   23.4  0.379
14    4.3  0.318    7.1   19.5  0.363
15    2.7  0.249    7.1   19.5  0.363
16    4.0  0.270    4.3   11.6  0.370
17    2.6  0.247    7.9   21.2  0.374
18    2.6  0.272    4.8   13.8  0.350
19    5.7  0.299    6.4   17.4  0.368
20    3.3  0.303    5.5   15.1  0.363
21    4.0  0.227    8.1   22.2  0.366
22    4.3  0.264    6.9   19.5  0.354
23    5.7  0.274    7.1   20.3  0.350
24    3.9  0.332    6.6   17.5  0.380
25    5.7  0.281    7.7   21.8  0.354
26    5.0  0.288    7.6   20.8  0.363
27    4.9  0.305    5.9   16.6  0.358
28    2.5  0.278    5.4   16.0  0.340
29    7.9  0.278    5.6   16.7  0.335

[30 rows x 33 columns]
```

1.7 Problem 5

Save the NBA dataframe you extracted in problem 4 as a JSON-formatted text file on your local machine. Format the JSON so that it is organized as dictionary with three lists: `columns` lists the column names, `index` lists the row names, and `data` is a list-of-lists of data points, one list for each row. (Hint: this is possible with one line of code) (2 points)

[58]:
```
df.to_json("team_data.txt",orient='split')
```