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

DS 6001: Practice and Application of Data Science

Drew Haynes (rbc6wr)

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.

Problem 0

Import the following libraries:

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

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)

Answer 2

You usually have similarly structured data elements, i.e. the same (more or less) columns for each row. Since CSVs just separate these columns with a single dilmitter, the files can comparatively be more compact than a JSON file that contains a little bit more formatting syntax. However, if you had a lot of rows with attributes unique to that individual row, a JSON file could be smaller. The JSON file would only have the populated attributes per element, while a CSV has to maintain the columns throughout the set regardless if a given row contains data for a column or not.

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)

Answer 2

It seems that the it's appropriate enough to just use the `requests.get` function using the provided url and load the resulting text using the `pd.json_normalize` function since there's not any specific feature we're looking for. Since each element seems to be composed at the highest level, there's no need to pass on a record path.

In [2]:
```
url = "https://data.nasa.gov/resource/y77d-th95.json"
r = requests.get(url, headers = {'User-agent': 'rbc6wr@virginia.edu'})
meteorites = pd.json_normalize(json.loads(r.text))
meteorites
```

Out[2]:

	name	id	nametype	recclass	mass	fall	year	reclat	reclong	geolocation.ty
0	Aachen	1	Valid	L5	21	Fell	1880-01-01T00:00:00.000	50.775000	6.083330	Po
1	Aarhus	2	Valid	H6	720	Fell	1951-01-01T00:00:00.000	56.183330	10.233330	Po
2	Abee	6	Valid	EH4	107000	Fell	1952-01-01T00:00:00.000	54.216670	-113.000000	Po
3	Acapulco	10	Valid	Acapulcoite	1914	Fell	1976-01-01T00:00:00.000	16.883330	-99.900000	Po
4	Achiras	370	Valid	L6	780	Fell	1902-01-01T00:00:00.000	-33.166670	-64.950000	Po
...	...	...	...	...	...	...	...	...	...	
995	Tirupati	24009	Valid	H6	230	Fell	1934-01-01T00:00:00.000	13.633330	79.416670	Po
996	Tissint	54823	Valid	Martian (shergottite)	7000	Fell	2011-01-01T00:00:00.000	29.481950	-7.611230	Po
997	Tjabe	24011	Valid	H6	20000	Fell	1869-01-01T00:00:00.000	-7.083330	111.533330	Po
998	Tjerebon	24012	Valid	L5	16500	Fell	1922-01-01T00:00:00.000	-6.666670	106.583330	Po
999	Tomakovka	24019	Valid	LL6	600	Fell	1905-01-01T00:00:00.000	47.850000	34.766670	Po

1000 rows × 13 columns

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)

Answer 3

In [3]:
```
url = "https://www.reddit.com/r/popular/top.json"
reddit = requests.get(url, headers = {'User-agent': 'rbc6wr@virginia.edu'})
reddit_json = json.loads(reddit.text)
columns = ['subreddit','title','ups','created_utc']

reddit_df = pd.DataFrame(
    [r['data']['subreddit'],r['data']['title'], r['data']['ups'], r['data']['created_utc']] for r
)
reddit_df.columns = columns
reddit_df
```

Out[3]:

	subreddit	title	ups	created_utc
0	nextfuckinglevel	Cat broke into Lynx's cage and now they are be...	103555	1.644737e+09
1	facepalm	Don't have the heart to tell her she's reading...	98962	1.644752e+09
2	nextfuckinglevel	French farmers' art for Tour de France!	102081	1.644782e+09
3	HumansBeingBros	Guy surprised his favourite shopkeeper.	92751	1.644752e+09
4	interestingasfuck	A crowd of angry parents hurl insults at 6 yea...	90383	1.644768e+09
5	meirl	Me irl	89769	1.644743e+09
6	MadeMeSmile	The way she ran to wash her hands first	83450	1.644772e+09
7	memes	Just give me paw!	83200	1.644768e+09
8	nextfuckinglevel	I spent two months on my Valkyrie costume!	79191	1.644763e+09
9	gaming	I'm looking at you CP	77745	1.644739e+09
10	wholesomememes	It's my turn now	77437	1.644740e+09
11	funny	Switch your style up [OC]	73459	1.644767e+09
12	HolUp	those people who push you to be your best	70994	1.644764e+09
13	interestingasfuck	Dog clears sheep traffic jam	69213	1.644758e+09
14	AdviceAnimals	I pointed this out to them and the next mornin...	66438	1.644748e+09
15	oddlysatisfying	Pixelated Gud Boi	65521	1.644754e+09
16	Unexpected	Achievement unlocked.. but what does that mean	64641	1.644740e+09
17	aww	An old woman who used to feed a monkey was bed...	65333	1.644750e+09
18	gaming	Booba	65440	1.644781e+09
19	WatchPeopleDieInside	I embarrassed myself.	60454	1.644753e+09
20	mildlyinfuriating	The "Grilled cheese Reuben" I spent $15 and an...	58683	1.644761e+09
21	me_irl	Me_irl	56901	1.644752e+09

	subreddit	title	ups	created_utc
22	Eyebleach	Platypuses/Platypi are extremely affectionate,...	57701	1.644758e+09
23	antiwork	Humour is allowed, right?	58162	1.644782e+09
24	todayilearned	TIL The only time a chess game has ended in a ...	54541	1.644758e+09

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.

Part a

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

In [4]:
```
url = "https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json"
nba = requests.get(url, headers = {'User-agent': 'rbc6wr@virginia.edu'})
nba_json = json.loads(nba.text)
```

Part b

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

Answer: The path is "resultSets", "rowSet".

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)

In [5]:
```
nba_df = pd.json_normalize(nba_json,
                           record_path = ['resultSets','rowSet'])
nba_df
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	27
0	1610612744	Golden State	Warriors	GSW		82	48.7	114.9	14.9	0.498	...	0.478	21.2	42.5	0.497	2.3
1	1610612759	San Antonio	Spurs	SAS		82	48.3	103.5	14.8	0.481	...	0.506	18.3	39.8	0.460	0.9
2	1610612739	Cleveland	Cavaliers	CLE		82	48.7	104.3	16.9	0.481	...	0.473	18.2	40.7	0.447	1.7
3	1610612746	Los Angeles	Clippers	LAC		82	48.6	104.5	15.0	0.497	...	0.480	18.9	42.0	0.450	2.0

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	27
4	1610612760	Oklahoma City	Thunder	OKC		82	48.6	110.2	16.1	0.480	...	0.497	17.5	38.7	0.451	1.6
5	1610612737	Atlanta	Hawks	ATL		82	48.6	102.8	19.0	0.463	...	0.483	19.4	44.6	0.435	1.0
6	1610612745	Houston	Rockets	HOU		82	48.6	106.5	17.2	0.433	...	0.472	15.5	36.4	0.426	2.3
7	1610612757	Portland	Trail Blazers	POR		82	48.5	105.1	17.5	0.441	...	0.447	18.0	39.8	0.453	1.7
8	1610612758	Sacramento	Kings	SAC		81	48.4	106.7	18.7	0.452	...	0.473	18.1	39.7	0.454	0.9
9	1610612764	Washington	Wizards	WAS		82	48.5	104.1	15.4	0.480	...	0.483	19.5	44.3	0.439	0.7
10	1610612748	Miami	Heat	MIA		82	48.6	100.0	17.9	0.488	...	0.490	15.7	35.2	0.445	0.8
11	1610612761	Toronto	Raptors	TOR		81	48.5	102.7	23.0	0.462	...	0.461	14.1	32.4	0.436	1.8
12	1610612742	Dallas	Mavericks	DAL		82	49.0	102.3	18.2	0.473	...	0.464	17.5	41.4	0.423	1.4
13	1610612766	Charlotte	Hornets	CHA		82	48.6	103.4	16.8	0.459	...	0.449	17.0	39.8	0.427	1.8
14	1610612762	Utah	Jazz	UTA		82	49.0	97.7	18.1	0.445	...	0.468	15.9	37.2	0.426	1.4
15	1610612753	Orlando	Magic	ORL		81	48.7	102.0	18.0	0.456	...	0.475	18.5	42.6	0.435	0.7
16	1610612749	Milwaukee	Bucks	MIL		82	48.7	99.0	17.4	0.463	...	0.477	13.2	29.4	0.448	1.1
17	1610612740	New Orleans	Pelicans	NOP		82	48.5	102.7	19.9	0.458	...	0.460	17.9	41.1	0.434	0.6
18	1610612750	Minnesota	Timberwolves	MIN		82	48.6	102.4	15.1	0.464	...	0.471	16.1	35.4	0.455	0.7
19	1610612754	Indiana	Pacers	IND		82	48.8	102.2	13.7	0.453	...	0.465	16.4	38.1	0.431	1.7
20	1610612751	Brooklyn	Nets	BKN		82	48.4	98.6	14.4	0.457	...	0.464	15.8	36.1	0.438	1.0
21	1610612765	Detroit	Pistons	DET		82	48.7	102.0	17.5	0.464	...	0.452	15.7	37.2	0.422	0.9
22	1610612743	Denver	Nuggets	DEN		82	48.6	101.9	15.9	0.406	...	0.448	16.4	37.8	0.434	1.1
23	1610612738	Boston	Celtics	BOS		81	48.5	105.6	18.9	0.453	...	0.451	16.9	39.9	0.424	1.6
24	1610612741	Chicago	Bulls	CHI		82	48.9	101.6	18.1	0.458	...	0.442	17.0	38.5	0.441	1.3
25	1610612755	Philadelphia	76ers	PHI		82	48.6	97.4	19.7	0.445	...	0.449	15.3	37.4	0.409	1.6
26	1610612756	Phoenix	Suns	PHX		82	48.4	100.9	15.6	0.440	...	0.447	16.6	39.5	0.421	1.4
27	1610612752	New York	Knicks	NYK		82	48.5	98.4	10.4	0.447	...	0.439	15.9	36.4	0.438	1.5
28	1610612763	Memphis	Grizzlies	MEM		82	48.6	99.1	16.4	0.440	...	0.459	16.1	38.5	0.418	0.7
29	1610612747	Los Angeles	Lakers	LAL		82	48.3	97.3	15.6	0.441	...	0.420	14.0	34.5	0.406	2.2

30 rows × 33 columns

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)

In [6]:

```
nba_df.columns = list((pd.json_normalize(nba_json,
                                record_path = ['resultSets','headers']))[0])
```

```
nba_df = nba_df.set_index('TEAM_ID')
nba_df
```

Out[6]:

TEAM_ID	TEAM_CITY	TEAM_NAME	TEAM_ABBREVIATION	TEAM_CODE	GP	MIN	PTS	PTS_DRIVE	FGP_DRIVI
1610612744	Golden State	Warriors	GSW	82	48.7	114.9	14.9	0.498
1610612759	San Antonio	Spurs	SAS	82	48.3	103.5	14.8	0.481
1610612739	Cleveland	Cavaliers	CLE	82	48.7	104.3	16.9	0.481
1610612746	Los Angeles	Clippers	LAC	82	48.6	104.5	15.0	0.497
1610612760	Oklahoma City	Thunder	OKC	82	48.6	110.2	16.1	0.480
1610612737	Atlanta	Hawks	ATL	82	48.6	102.8	19.0	0.463
1610612745	Houston	Rockets	HOU	82	48.6	106.5	17.2	0.433
1610612757	Portland	Trail Blazers	POR	82	48.5	105.1	17.5	0.441
1610612758	Sacramento	Kings	SAC	81	48.4	106.7	18.7	0.452
1610612764	Washington	Wizards	WAS	82	48.5	104.1	15.4	0.480
1610612748	Miami	Heat	MIA	82	48.6	100.0	17.9	0.488
1610612761	Toronto	Raptors	TOR	81	48.5	102.7	23.0	0.462
1610612742	Dallas	Mavericks	DAL	82	49.0	102.3	18.2	0.473
1610612766	Charlotte	Hornets	CHA	82	48.6	103.4	16.8	0.459
1610612762	Utah	Jazz	UTA	82	49.0	97.7	18.1	0.445
1610612753	Orlando	Magic	ORL	81	48.7	102.0	18.0	0.456
1610612749	Milwaukee	Bucks	MIL	82	48.7	99.0	17.4	0.463
1610612740	New Orleans	Pelicans	NOP	82	48.5	102.7	19.9	0.458
1610612750	Minnesota	Timberwolves	MIN	82	48.6	102.4	15.1	0.464
1610612754	Indiana	Pacers	IND	82	48.8	102.2	13.7	0.453
1610612751	Brooklyn	Nets	BKN	82	48.4	98.6	14.4	0.457
1610612765	Detroit	Pistons	DET	82	48.7	102.0	17.5	0.464
1610612743	Denver	Nuggets	DEN	82	48.6	101.9	15.9	0.406
1610612738	Boston	Celtics	BOS	81	48.5	105.6	18.9	0.453
1610612741	Chicago	Bulls	CHI	82	48.9	101.6	18.1	0.458
1610612755	Philadelphia	76ers	PHI	82	48.6	97.4	19.7	0.445
1610612756	Phoenix	Suns	PHX	82	48.4	100.9	15.6	0.440
1610612752	New York	Knicks	NYK	82	48.5	98.4	10.4	0.447
1610612763	Memphis	Grizzlies	MEM	82	48.6	99.1	16.4	0.440
1610612747	Los Angeles	Lakers	LAL	82	48.3	97.3	15.6	0.441

30 rows × 32 columns

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)

In [7]:
```
nba_df.to_json(path_or_buf="nba_df.json", orient='split')
```