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

DS 6001: Practice and Application of Data Science

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 = 0 # 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)

A CSV file takes less memory than a JSON formatted file for the same data because JSON implements a tree structure and the data constains a lot of brackets, values, lists, and extra information beyond simply the data. CSV files on the other hand only have the data and a separating delimiter.

A JSON file could be smaller in memory than a CSV file for the same data if there are a lot of missing values. This is because missing values in JSON are simply omitted while in a CSV file there are recorded as "NA".

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)

Because this JSON data is nested but does not include metadata, the strategy for loading the JSON into Python that makes the most sense is as follows:

1. Uve `requests.get()` to download the raw JSON data
2. Use json.loads() on the .text attribute of the output from step 1 to register the data as a list in Python
3. Use the pd.json_normalize() function on the list that is the output of step 2

In [2]:
```
url = "https://data.nasa.gov/resource/y77d-th95.json"
nasa = requests.get(url)
nasa_json = json.loads(nasa.text)
nasa_df = pd.json_normalize(nasa_json)
nasa_df
```

Out[2]:

	name	id	nametype	recclass	mass	fall	year	reclat	
0	Aachen	1	Valid	L5	21	Fell	1880-01-01T00:00:00.000	50.775000	6
1	Aarhus	2	Valid	H6	720	Fell	1951-01-01T00:00:00.000	56.183330	10
2	Abee	6	Valid	EH4	107000	Fell	1952-01-01T00:00:00.000	54.216670	-113
3	Acapulco	10	Valid	Acapulcoite	1914	Fell	1976-01-01T00:00:00.000	16.883330	-99
4	Achiras	370	Valid	L6	780	Fell	1902-01-01T00:00:00.000	-33.166670	-64
...	...	...	...	...	...	...	...	...	
995	Tirupati	24009	Valid	H6	230	Fell	1934-01-01T00:00:00.000	13.633330	79
996	Tissint	54823	Valid	Martian (shergottite)	7000	Fell	2011-01-01T00:00:00.000	29.481950	-
997	Tjabe	24011	Valid	H6	20000	Fell	1869-01-01T00:00:00.000	-7.083330	111
998	Tjerebon	24012	Valid	L5	16500	Fell	1922-01-01T00:00:00.000	-6.666670	106
999	Tomakovka	24019	Valid	LL6	600	Fell	1905-01-01T00:00:00.000	47.850000	34

1000 rows × 13 columns

Problem 3

https://open-meteo.com/ provides free and accurate weather forecasts for any location, and shares these forecasts via a free and open API in JSON format. The JSON that contains the next week of forecasts for Charlottesville is here: https://api.open-meteo.com/v1/forecast?latitude=38.03&longitude=-78.48&hourly=temperature_2m,relativehumidity_2m,precipitation,cl
(You can paste this URL to jsonhero.io if you want)

Create a dataframe with 168 rows (one for each hour of each day for the next 7 days) and only columns for features contained within the `hourly` key. [Note: this problem does not require either `pd.read_json()` or `pd.json_normalize()`.]

Also, make sure to include your user-agent string.

As an aside: consider for a moment what we could use access to the API to do. We could write Python code that connects to events on Facebook or Meetup, pulls weather data from this API, and automatically cancels outdoor events that have a high probability of rain in the forecast. Or we can set up automated notifications for stargazing events when the skies will be clear. Maybe we can build a routing app for tornado chasers. Or we can build a model that predicts plant growth from watering times under different weather conditions and notify

a gardener about the ideal times to tend to the plants. Can you think of other potential uses of this fast, free, and accurate data? (3 points)

In [3]:
```
r=requests.get("http://httpbin.org/user-agent")
useragent = json.loads(r.text)['user-agent']
headers={'User-agent': useragent}

url = "https://api.open-meteo.com/v1/forecast?latitude=38.03&longitude=-78.48&h
weather = requests.get(url, headers=headers)
weather_json = json.loads(weather.text)
weather_df = pd.DataFrame(weather_json['hourly'])
weather_df
```

Out[3]:

	time	temperature_2m	relativehumidity_2m	precipitation	cloudcover
0	2023-06-27T00:00	71.0	82	0.0	99
1	2023-06-27T01:00	69.8	91	0.0	99
2	2023-06-27T02:00	69.5	90	0.0	99
3	2023-06-27T03:00	68.1	89	0.0	100
4	2023-06-27T04:00	67.0	89	0.0	99
...	...	...	...	...	...
163	2023-07-03T19:00	88.1	45	0.0	96
164	2023-07-03T20:00	87.9	45	0.0	93
165	2023-07-03T21:00	87.0	47	0.0	89
166	2023-07-03T22:00	85.2	52	0.0	67
167	2023-07-03T23:00	82.8	59	0.0	46

168 rows × 5 columns

Another potential use for this fast, free, and accurate data is using it to automate solar panels to shift direction to gather the most sunlight and avoid sloud cover. Or, this could be used to alert people of the best time to tan and the duration of tanning before injury or burn.

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]:
```
# this dataset lists the column headers first then the data...
url = "https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json"
nba = requests.get(url)
nba_json = json.loads(nba.text)
#nba_json
```

Part b

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

The path that leads to the team-by-team data is by looking in the "resultSets" dictionary, entering the first entry (at index = 0), then entering the "rowSet" disctionary. Every entry, or index, in this rowSet dictionary is one row of the team-by-team data.

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	2
0	1610612744	Golden State	Warriors	GSW		82	48.7	114.9	14.9	0.498	...	0.478	21
1	1610612759	San Antonio	Spurs	SAS		82	48.3	103.5	14.8	0.481	...	0.506	18
2	1610612739	Cleveland	Cavaliers	CLE		82	48.7	104.3	16.9	0.481	...	0.473	18
3	1610612746	Los Angeles	Clippers	LAC		82	48.6	104.5	15.0	0.497	...	0.480	18
4	1610612760	Oklahoma City	Thunder	OKC		82	48.6	110.2	16.1	0.480	...	0.497	17
5	1610612737	Atlanta	Hawks	ATL		82	48.6	102.8	19.0	0.463	...	0.483	19
6	1610612745	Houston	Rockets	HOU		82	48.6	106.5	17.2	0.433	...	0.472	15
7	1610612757	Portland	Trail Blazers	POR		82	48.5	105.1	17.5	0.441	...	0.447	18
8	1610612758	Sacramento	Kings	SAC		81	48.4	106.7	18.7	0.452	...	0.473	18
9	1610612764	Washington	Wizards	WAS		82	48.5	104.1	15.4	0.480	...	0.483	19
10	1610612748	Miami	Heat	MIA		82	48.6	100.0	17.9	0.488	...	0.490	15
11	1610612761	Toronto	Raptors	TOR		81	48.5	102.7	23.0	0.462	...	0.461	14
12	1610612742	Dallas	Mavericks	DAL		82	49.0	102.3	18.2	0.473	...	0.464	17
13	1610612766	Charlotte	Hornets	CHA		82	48.6	103.4	16.8	0.459	...	0.449	17
14	1610612762	Utah	Jazz	UTA		82	49.0	97.7	18.1	0.445	...	0.468	15
15	1610612753	Orlando	Magic	ORL		81	48.7	102.0	18.0	0.456	...	0.475	18
16	1610612749	Milwaukee	Bucks	MIL		82	48.7	99.0	17.4	0.463	...	0.477	13
17	1610612740	New Orleans	Pelicans	NOP		82	48.5	102.7	19.9	0.458	...	0.460	17
18	1610612750	Minnesota	Timberwolves	MIN		82	48.6	102.4	15.1	0.464	...	0.471	16
19	1610612754	Indiana	Pacers	IND		82	48.8	102.2	13.7	0.453	...	0.465	16
20	1610612751	Brooklyn	Nets	BKN		82	48.4	98.6	14.4	0.457	...	0.464	15
21	1610612765	Detroit	Pistons	DET		82	48.7	102.0	17.5	0.464	...	0.452	15
22	1610612743	Denver	Nuggets	DEN		82	48.6	101.9	15.9	0.406	...	0.448	16
23	1610612738	Boston	Celtics	BOS		81	48.5	105.6	18.9	0.453	...	0.451	16
24	1610612741	Chicago	Bulls	CHI		82	48.9	101.6	18.1	0.458	...	0.442	17
25	1610612755	Philadelphia	76ers	PHI		82	48.6	97.4	19.7	0.445	...	0.449	15
26	1610612756	Phoenix	Suns	PHX		82	48.4	100.9	15.6	0.440	...	0.447	16
27	1610612752	New York	Knicks	NYK		82	48.5	98.4	10.4	0.447	...	0.439	15
28	1610612763	Memphis	Grizzlies	MEM		82	48.6	99.1	16.4	0.440	...	0.459	16
29	1610612747	Los Angeles	Lakers	LAL		82	48.3	97.3	15.6	0.441	...	0.420	14

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]:
```
headers = nba_json['resultSets'][0]['headers']
nba_df.columns = headers
nba_df
```

Out[6]:

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

30 rows × 33 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_j = nba_df.to_json(orient="split")

import os
os.chdir("/Users/rachelholman/Desktop/MSDS/DS6001 - Application of DS/Module 3-

with open('nba_j', 'w') as outfile:
     json.dump(nba_j, outfile, sort_keys = True, indent = 4,
               ensure_ascii = False)
```