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

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 [60]:
```
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)

Answer 1

JSON files are supposed to be more flexible, but because if that, the caveat is that it will probably be harder to split. Compare that to CSV which uses a comma as the separator (CSV is simpler). JSON files are also usually larger files compared to CSV, so it requires more memory than CSV.

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

I would load the JSON file in as a list to make it easier to find information using brackets.

In [61]:
```
url = "https://data.nasa.gov/resource/y77d-th95.json"
r = requests.get(url)
nasa = json.loads(r.text)
nasa[0]
```

Out[61]:
```
{'name': 'Aachen',
 'id': '1',
 'nametype': 'Valid',
 'recclass': 'L5',
 'mass': '21',
 'fall': 'Fell',
 'year': '1880-01-01T00:00:00.000',
 'reclat': '50.775000',
 'reclong': '6.083330',
 'geolocation': {'type': 'Point', 'coordinates': [6.08333, 50.775]}}
```

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,clouc
(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)

Answer 3

In [62]:
```
## go to google and ask "what is my user agent" which is WRONG
## httpbin.org/user-agent use this url to get user agent
r = requests.get("https://httpbin.org/user-agent")
useragent = json.loads(r.text)["user-agent"]
headers = {'User-agent':useragent}
useragent
```

Out[62]:
```
'python-requests/2.28.1'
```

In [63]:
```
url3 = "https://api.open-meteo.com/v1/forecast?latitude=38.03&longitude=-78.48&hourly=
r = requests.get(url3, headers = headers)
weather = json.loads(r.text)
```

In [64]:
```
pd.DataFrame(weather["hourly"])
```

Out[64]:

	time	temperature_2m	relativehumidity_2m	precipitation	cloudcover
0	2023-06-29T00:00	78.3	50	0.0	0
1	2023-06-29T01:00	72.9	60	0.0	0
2	2023-06-29T02:00	70.4	62	0.0	0
3	2023-06-29T03:00	67.8	72	0.0	0
4	2023-06-29T04:00	66.8	69	0.0	0
...	...	...	...	...	...
163	2023-07-05T19:00	92.2	36	0.0	40
164	2023-07-05T20:00	90.3	43	0.0	64
165	2023-07-05T21:00	87.9	49	0.0	87
166	2023-07-05T22:00	85.0	55	0.0	80
167	2023-07-05T23:00	81.7	61	0.0	74

168 rows × 5 columns

My user agent string is: 'python-requests/2.28.1'

Imagine a house or community that uses a combination of renewable resources like solar, wind, and water power and nonrenewable resources (hybrid situation) in order to combat excessive use of nonrenewable resources. I was thinking we could write a program that connects to the systems, pulls weather data from this API, and automatically turns on/off these systems in accordance to the predicted weather. So a really sunny day (cloud cover column), all the solar power systems open up to capture as much solar energy and store for really cloudy days. Or for high temperature days (temperature column), systems that convert heat energy to electric energy will activate (Thermoelectric devices are made from materials that can convert a temperature difference into electricity, without requiring any moving parts). Energy will be

supplemented with nonrenewable sources and we can save more energy (theoretically) by not having these systems running 24/7.

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)

Part b

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

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)

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)

Answer 4a

In [38]:
```
url4a = "https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json"
four = requests.get(url4a)
q4a = json.loads(four.text)
```

Answer 4b

If we go into "resultSets", there's "rowSet" at the 0 index, and through that, it'll have the team-by-team data.

Answer 4c

In [16]:
```
q4c = pd.json_normalize(q4a, record_path=["resultSets","rowSet"])
q4c.head()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26
0	1610612744	Golden State	Warriors	GSW		82	48.7	114.9	14.9	0.498	...	0.478	21.2	42.5	0.497
1	1610612759	San Antonio	Spurs	SAS		82	48.3	103.5	14.8	0.481	...	0.506	18.3	39.8	0.460
2	1610612739	Cleveland	Cavaliers	CLE		82	48.7	104.3	16.9	0.481	...	0.473	18.2	40.7	0.447
3	1610612746	Los Angeles	Clippers	LAC		82	48.6	104.5	15.0	0.497	...	0.480	18.9	42.0	0.450
4	1610612760	Oklahoma City	Thunder	OKC		82	48.6	110.2	16.1	0.480	...	0.497	17.5	38.7	0.451

5 rows × 33 columns

Answer 4d

In [22]:
```
q4d = q4c
q4d.columns = q4a["resultSets"][0]["headers"]
q4d.head()
```

Out[22]:

	TEAM_ID	TEAM_CITY	TEAM_NAME	TEAM_ABBREVIATION	TEAM_CODE	GP	MIN	PTS	PTS_I
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 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 [58]:
```
q5 = q4d.to_json()
# json.loads(q5)
```

In [59]:
```
with open("C:/Users/jacqu/Documents/MSDS/CW/lab3_q5.json", "w") as outfile:
    outfile.write(q5)
```

In [ ]:
```

```