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 [2]:
```
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:

Generally speaking, JSON tends to be bigger because all the keys are repeated in a nested structure. One likely condition when JSON may be smaller than CSV is when there are lot of missing values in the data. JSON will simply omit keys when corresponding values are missing, but CSV will still store NA or an extra comma to indicate that the value is missing. When the size of the data is large, this will lead to a noticeably larger CSV files.

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 (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:

Strategy: JSON has some nesting when it comes to "geolocation", so use normalize with `max_level=1` .

In [208]:
```
# First create `req` object and check its status:
url = "https://data.nasa.gov/resource/y77d-th95.json"
req = requests.get(url, headers = {'User-agent':'Dima'})
req
```

Out[208]:
```
<Response [200]>
```

In [209]:
```
# Second, load text into the memory and check the nested structure:
nasa_json = json.loads(req.text)
nasa_json[:2]
```

Out[209]:
```
[{'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]}},
 {'name': 'Aarhus',
  'id': '2',
  'nametype': 'Valid',
  'recclass': 'H6',
  'mass': '720',
  'fall': 'Fell',
  'year': '1951-01-01T00:00:00.000',
  'reclat': '56.183330',
  'reclong': '10.233330',
  'geolocation': {'type': 'Point', 'coordinates': [10.23333, 56.1833
3]}}]
```

In [211]:
```
# Use pandas normalize with 1 additional level of nesting:
pd.json_normalize(nasa_json, max_level=1)[:5]
```

Out[211]:

	name	id	nametype	recclass	mass	fall	year	reclat	reclong
0	Aachen	1	Valid	L5	21	Fell	1880-01-01T00:00:00.000	50.775000	6.083330
1	Aarhus	2	Valid	H6	720	Fell	1951-01-01T00:00:00.000	56.183330	10.233330
2	Abee	6	Valid	EH4	107000	Fell	1952-01-01T00:00:00.000	54.216670	-113.000000
3	Acapulco	10	Valid	Acapulcoite	1914	Fell	1976-01-01T00:00:00.000	16.883330	-99.900000
4	Achiras	370	Valid	L6	780	Fell	1902-01-01T00:00:00.000	-33.166670	-64.950000

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 (https://www.reddit.com/r/popular/top/). 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 (http://www.reddit.com/r/popular/top.json), and don't forget to specify `headers = {'User-agent': 'DS6001'}` within `requests.get()`. (3 points)

In [9]:
```
# Check status of the request
url2 = 'http://www.reddit.com/r/popular/top.json'
req2 = requests.get(url2, headers = {'User-agent': 'DS6001'})
req2
```

Out[9]:
```
<Response [200]>
```

In [68]:
```
# Load text into the memory
reddit_json = json.loads(req2.text)
```

In [43]:
```
# For the first record, check 'subreddit'
reddit_json['data']['children'][0]['data']['subreddit']
```

Out[43]:
```
'MadeMeSmile'
```

In [44]:
```
# Same, check 'title':
reddit_json['data']['children'][0]['data']['title']
```

Out[44]:
```
'Make sure the human is following our group, we must keep it alive and
safe'
```

In [37]:
```
# Check 'ups'
reddit_json['data']['children'][0]['data']['ups']
```

Out[37]:
```
118314
```

In [39]:
```
# Check 'created_utc' fieled:
reddit_json['data']['children'][0]['data']['created_utc']
```

Out[39]:
```
1644378607.0
```

In [63]:
```
# Now, repeat the above steps in loops for 25 records, add results to a
 dataframe
reddit_df = pd.DataFrame(columns=['subreddit', 'title', 'ups', 'created_
utc'])
reddit_df.subreddit = [j['data']['subreddit'] for j in reddit_json['dat
a']['children']][:25]
reddit_df.title = [j['data']['title'] for j in reddit_json['data']['chil
dren']][:25]
reddit_df.ups = [j['data']['ups'] for j in reddit_json['data']['childre
n']][:25]
reddit_df.created_utc = [j['data']['created_utc'] for j in reddit_json[
'data']['children']][:25]
```

In [64]:
```
# Inspect the resultant dataframe:
reddit_df
```

Out[64]:

	subreddit	title	ups	created_utc
0	MadeMeSmile	Make sure the human is following our group, we...	118314	1.644379e+09
1	nottheonion	Meta's threat to close down Facebook and Insta...	119017	1.644370e+09
2	memes	Oh no anyway	94319	1.644398e+09
3	nextfuckinglevel	When everything goes right	88234	1.644372e+09
4	BlackPeopleTwitter	Granny adding insult to injury	74881	1.644357e+09
5	interestingasfuck	The world's biggest floating crane "Hyundai 10...	74081	1.644370e+09
6	Unexpected	Meanwhile in Japan....	69963	1.644371e+09
7	nextfuckinglevel	Steal my wave, i steal your board	71621	1.644393e+09
8	antiwork	As it should be. 😈	65993	1.644357e+09
9	mildlyinfuriating	Our High school covers the expiration date wit...	65632	1.644356e+09
10	nextfuckinglevel	Forklift certified	65730	1.644363e+09
11	interestingasfuck	This cool image shows the coldest known spot i...	65383	1.644357e+09
12	antiwork	Fucking hypocrites	65064	1.644376e+09
13	meirl	Meirl	64547	1.644356e+09
14	funny	Dakota has been on stall rest for months due t...	65431	1.644381e+09
15	nextfuckinglevel	This is the Real Fair Play	67680	1.644408e+09
16	MurderedByWords	VaCcInEs CaUsE aUtIsM	64106	1.644391e+09
17	WhitePeopleTwitter	Give that man a raise now!	58573	1.644354e+09
18	aww	her cheek squished against his shoulder is so ...	57568	1.644355e+09
19	WhatsWrongWithYourDog	Update for those who asked. BAT HAS BEEN ADOPT...	56769	1.644360e+09
20	antiwork	•₊° -,` ♡ ,´-NO	62483	1.644416e+09
21	FunnyAnimals	found this somewhere	54688	1.644382e+09
22	AnimalsBeingDerps	Kitty enjoying the snow	51021	1.644357e+09
23	iamatotalpieceofshit	Getting away with cheating on the olympic games	50999	1.644367e+09
24	oddlysatisfying	Removing algae from water lily to help it bloom	51367	1.644403e+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 (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 [70]:
```
# Check status of the request object:
url3 = 'https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.jso
n'
req3 = requests.get(url3, headers = {'User-agent': 'Dima'})
req3
```

```
ERROR! Session/line number was not unique in database. History logging
moved to new session 1181
```

Out[70]:
```
<Response [200]>
```

In [73]:
```
# Load text
nba_json = json.loads(req3.text)
```

Part b

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

Answer 4.b:

We need to look at the key `resultSets` and the first element in the corresponding list that will have `headers` with column names and `rowSet` with actual results in a team-by-team fashion. The code below returns results for the first team, Golden State Warriors

In [91]:
```
# Path to get to the first team
nba_json['resultSets'][0]['rowSet'][0]
```

Out[91]:
```
['1610612744',
 'Golden State',
 'Warriors',
 'GSW',
 '',
 82,
 48.7,
 114.9,
 14.9,
 0.498,
 16.7,
 0.645,
 33.7,
 0.428,
 21.5,
 0.418,
 11.0,
 11.1,
 28.3,
 21.5,
 0.563,
 21.4,
 44.8,
 0.478,
 21.2,
 42.5,
 0.497,
 2.3,
 6.3,
 0.363,
 10.8,
 25.3,
 0.429]
```

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 [212]:
```
# Build the dataframe normailzing with respect o `resultSets` and `rowSe
t`
nba_df = pd.json_normalize(nba_json, record_path=['resultSets', 'rowSet'
])

# Check the resultant dataframe:
nba_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 33 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       30 non-null     object
 1   1       30 non-null     object
 2   2       30 non-null     object
 3   3       30 non-null     object
 4   4       30 non-null     object
 5   5       30 non-null     int64
 6   6       30 non-null     float64
 7   7       30 non-null     float64
 8   8       30 non-null     float64
 9   9       30 non-null     float64
 10  10      30 non-null     float64
 11  11      30 non-null     float64
 12  12      30 non-null     float64
 13  13      30 non-null     float64
 14  14      30 non-null     float64
 15  15      30 non-null     float64
 16  16      30 non-null     float64
 17  17      30 non-null     float64
 18  18      30 non-null     float64
 19  19      30 non-null     float64
 20  20      30 non-null     float64
 21  21      30 non-null     float64
 22  22      30 non-null     float64
 23  23      30 non-null     float64
 24  24      30 non-null     float64
 25  25      30 non-null     float64
 26  26      30 non-null     float64
 27  27      30 non-null     float64
 28  28      30 non-null     float64
 29  29      30 non-null     float64
 30  30      30 non-null     float64
 31  31      30 non-null     float64
 32  32      30 non-null     float64
dtypes: float64(27), int64(1), object(5)
memory usage: 7.9+ KB
```

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 [185]:
```
# Grab column names directly from the original JSON using `.values.tolist()`
names_list = pd.json_normalize(nba_json, record_path=['resultSets',['headers']]).values.tolist()

# Flatten the list of lists using `sublist`
flat_list = [item for sublist in names_list for item in sublist]
nba_df.columns = flat_list

# Check the resultant dataframe:
nba_df.head(3)
```

Out[185]:

	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 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 [195]:
```
# Store required lists in a disctionary
nba_dict={}
nba_dict['columns'] = nba_df.columns.to_list()
nba_dict['index'] = nba_df.index.tolist()
nba_dict['data'] = nba_df.values.tolist()
```

In [200]:
```
# Store JSON object to a local text file:
with open('data.json', 'w') as fp:
    json.dump(nba_dict, fp)
```