H. Diana McSpadden (hdm5s)

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 [ ]: 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)

Problem 1 Answer:

The key value pair structure of JSON files may make JSON format more verbose. For example:

```
{
 "name": "John Smith",
 "age": 30,
 "address": {
   "street": "123 Main St",
   "city": "Anytown",
   "state": "CA",
   "zip": "12345"
 },
 "phoneNumbers": [
   {
     "type": "home",
     "number": "555-555-5555"
   },
   {
     "type": "work",
     "number": "555-555-5556"
   }
 ]
}
```

vs. the CSV format:

name	age	street	city	state	zip	type	number
John Smith	30	123	Main St	Anytown, CA	12345	home	555-555-5555
John Smith	30	123	Main St	Anytown, CA	12345	work	555-555-5555

However, JSON could take a smaller amount of memory when the data contains a large number of nested objects such that the CSV format would result in flattening, i.e. repeating, quite a bit of information for all the nested objects.

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:

The data includes name, id, nametype,recclass, mass, fall, year, reclat, reclong, geolocation {type, coordinates:list}. I would want to confirm, but eyeballing seems to show that the geolocation coordinates match the reclat and reclong coordinates.

I will use the requests.get() method to retrieve the json from the url provided.

```
In [ ]: # load the data from the url in problem 2
        nasa_data = requests.get('https://data.nasa.gov/resource/y77d-th95.json').json()
```

Print out the first two results in the json:

```
In [ ]: nasa_data[0:2]
```

```
Out[ ]: [{'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, 56.18333]}}]
```

```
In [ ]: # load the nasadata into a pandas dataframe
        nasa_df = pd.DataFrame(nasa_data)
        nasa_df = nasa_df[['name','id','nametype','recclass','mass','fall','year','reclat',
        nasa_df.head(2)
```

Out[ ]:

	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

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:

```
In [ ]: # create an empty pandas DataFrame
df_reddit = pd.DataFrame()

url = "http://www.reddit.com/r/popular/top.json"
columns = ['subreddit' , 'title' ,'ups' , 'created_utc']
reddit = requests.get(url, headers = {'User-agent': 'DS6001'})
for post in reddit.json()['data']['children'][0:25]:
    # I only want the columns in columns
    post = {k: post['data'][k] for k in columns}
    # add the post to the DataFrame using pd.concat() method to add a new row
    df_reddit = pd.concat([df_reddit, pd.DataFrame(post, index=[0])], ignore_index=

df_reddit
```

Out[ ]:

	subreddit	title	ups	created_utc
0	MadeMeSmile	Mad respect to both of them	101485	1.674849e+09
1	meirl	Meirl	90717	1.674851e+09
2	wholesomememes	Stage 3 breast, stage 1 breast, stage 3 small ...	90428	1.674857e+09
3	nextfuckinglevel	Silverback sees a little girl banging her ches...	84201	1.674841e+09
4	antiwork	very striking	80861	1.674839e+09
5	therewasanattempt	to be a dj	79062	1.674839e+09
6	todayilearned	TIL Fender Guitars did a study and found that ...	73566	1.674847e+09
7	nextfuckinglevel	Cat broke into Lynx's cage and now they are be...	72659	1.674846e+09
8	funny	My mom is diabetic. She eats Rockets to raise ...	71933	1.674833e+09
9	interestingasfuck	On June 27th 1999, Tony Hawk became the worlds...	71457	1.674858e+09
10	news	Tyre Nichols: Memphis police release body cam ...	71422	1.674864e+09
11	worldnews	Japanese Govt Set to Legalize Medical Marijuana	67256	1.674831e+09
12	Unexpected	i would shit my pants	63210	1.674859e+09
13	WhitePeopleTwitter	This is horrific	62688	1.674867e+09
14	ContagiousLaughter	Roll down window prank.	61048	1.674840e+09
15	comics	Any other available job openings?	58269	1.674864e+09
16	Damnthatsinteresting	After the death of her husband &amp; with no b...	56686	1.674846e+09
17	Whatcouldgowrong	WCGW leaving the van in neutral	52861	1.674832e+09
18	Unexpected	Having older brothers.	48652	1.674842e+09
19	wholesomememes	terry crews is a national treasure	49061	1.674854e+09
20	news	‘You’re going to see acts that defy humanity,’...	46475	1.674834e+09
21	IdiotsInCars	Tried to cut me off and instantly regretted it.	44007	1.674849e+09
22	WhitePeopleTwitter	Red state America needs a civics lesson if the...	42251	1.674835e+09
23	MaliciousCompliance	Boss says "If you're 1 minute late I'm docking...	42193	1.674831e+09
24	ContagiousLaughter	This is how you do a travel video	42819	1.674873e+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 [ ]:
```
# load the json file from https://stats.nba.com/js/data/sportvu/2015/shootingTeamDa
the_url = 'https://stats.nba.com/js/data/sportvu/2015/shootingTeamData.json'
r = requests.get(the_url)
nba_json = json.loads(r.text)
```

Part b

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

Answer:

First, the easy part: The column names are in the resultSets[0].headers item, so we can use this as the column names in our dataframe.

The values are in the resultSets[0].rowSet records - these will make up the rows in the dataframe.

```
nba_json['resultSets'][0]['rowSet']
```

will return a list of lists which is acceptable input to create a pandas Dataframe, so this is an easy way to create the dataframe we are looking for.

In [ ]:
```
nba_json['resultSets'][0]['rowSet'][0:1]
```

Out[ ]:
```
[['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]]
```

In [ ]:
```
print("Confirming the count of columns is 33: ", len(nba_json['resultSets'][0]['hea
nba_json['resultSets'][0]['headers']
```

```
Confirming the count of columns is 33:  33
```

```
Out[ ]: ['TEAM_ID',
         'TEAM_CITY',
         'TEAM_NAME',
         'TEAM_ABBREVIATION',
         'TEAM_CODE',
         'GP',
         'MIN',
         'PTS',
         'PTS_DRIVE',
         'FGP_DRIVE',
         'PTS_CLOSE',
         'FGP_CLOSE',
         'PTS_CATCH_SHOOT',
         'FGP_CATCH_SHOOT',
         'PTS_PULL_UP',
         'FGP_PULL_UP',
         'FGA_DRIVE',
         'FGA_CLOSE',
         'FGA_CATCH_SHOOT',
         'FGA_PULL_UP',
         'EFG_PCT',
         'CFGM',
         'CFGA',
         'CFGP',
         'UFGM',
         'UFGA',
         'UFGP',
         'CFG3M',
         'CFG3A',
         'CFG3P',
         'UFG3M',
         'UFG3A',
         'UFG3P']
```

```
In [ ]: # get the count of resultSets[0].rowSet in the data, confirm the count is 30
        print("Confirm the count of teams is 30: ", len(nba_json['resultSets'][0]['rowSet']
```

```
Confirm the count of teams is 30:  30
```

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 [ ]: pd.set_option('display.max_columns', None)
```

In [ ]:
```
# put the json into a dataframe
# Dataframes accept a list of lists, a list of dictionaries, or a dictionary of lis

# this is a different way to create the DataFrame, but not the requested way in the
#df_nba = pd.DataFrame(nba_json['resultSets'][0]['rowSet'], columns=nba_json['resul

# this is the requested way to create the DataFrame
df_nba = pd.json_normalize(nba_json, record_path=['resultSets','rowSet'])
print(df_nba.shape)
df_nba.head()
```

```
(30, 33)
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1610612744	Golden State	Warriors	GSW		82	48.7	114.9	14.9	0.498	16.7	0.645	33.7	0.428
1	1610612759	San Antonio	Spurs	SAS		82	48.3	103.5	14.8	0.481	17.8	0.611	27.1	0.419
2	1610612739	Cleveland	Cavaliers	CLE		82	48.7	104.3	16.9	0.481	14.3	0.622	28.2	0.394
3	1610612746	Los Angeles	Clippers	LAC		82	48.6	104.5	15.0	0.497	12.7	0.712	26.5	0.404
4	1610612760	Oklahoma City	Thunder	OKC		82	48.6	110.2	16.1	0.480	15.3	0.677	24.7	0.402

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 [ ]:
```
# set the column names
df_nba.columns = nba_json['resultSets'][0]['headers']
df_nba.head()
```

Out[ ]:

	TEAM_ID	TEAM_CITY	TEAM_NAME	TEAM_ABBREVIATION	TEAM_CODE	GP	MIN	PTS	P
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	

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)

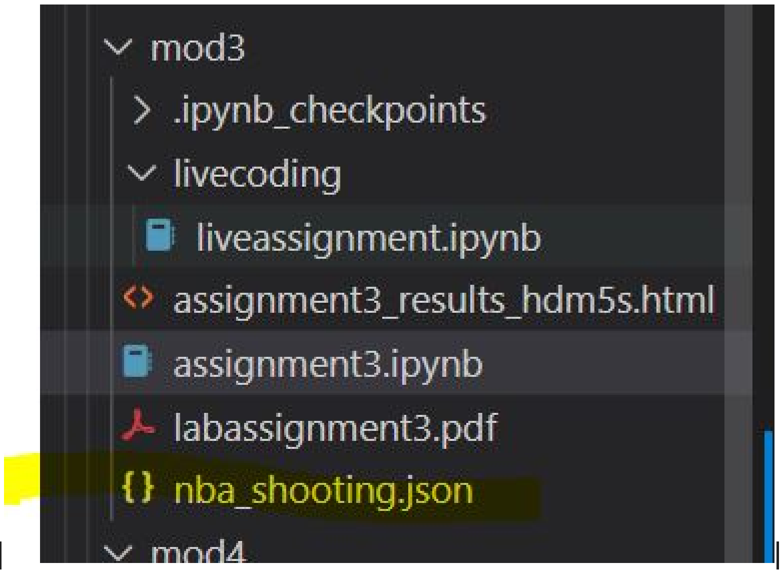
Answer

In [ ]:
```
import os
os.getcwd()
```

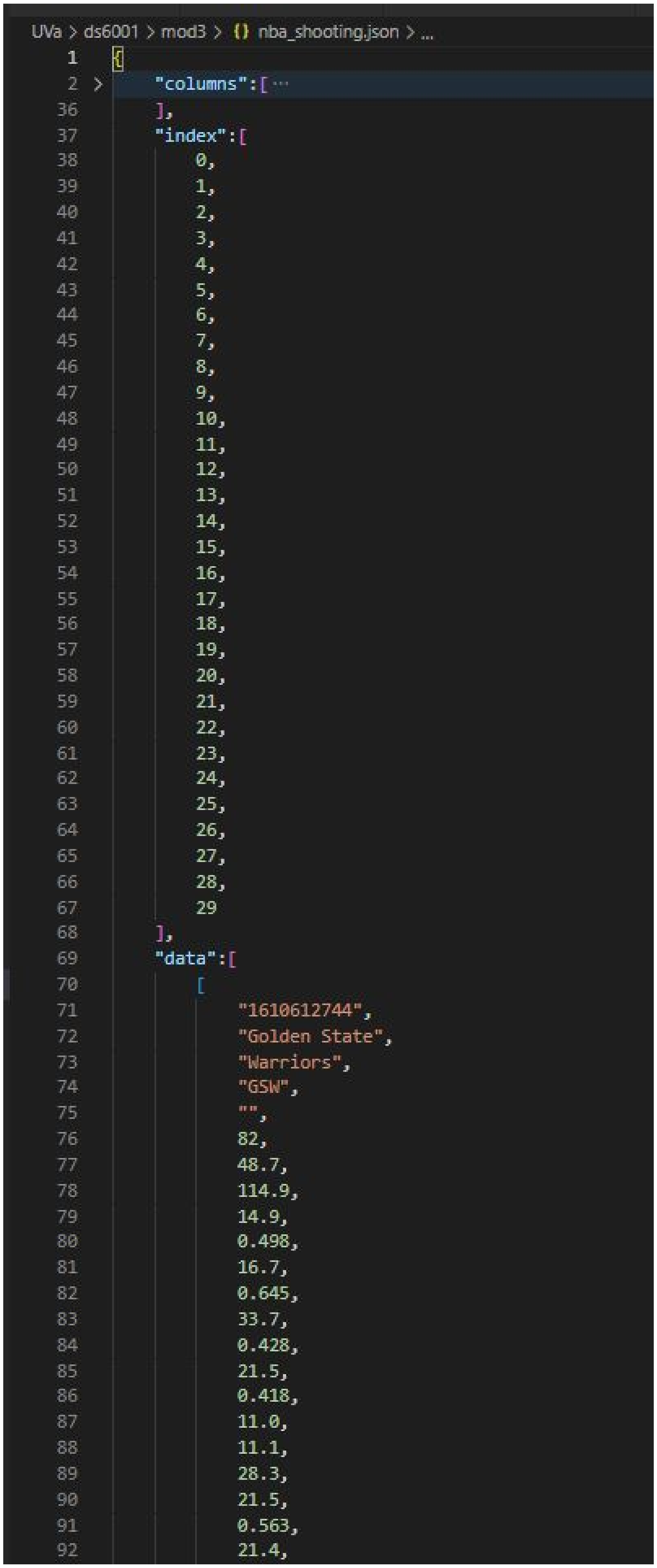
Out[ ]:
```
'c:\\Users\\dianam\\Documents\\jlab_datascience\\PlayGround\\UVa\\ds6001\\mod3'
```

In [ ]:
```
# the orient == split seems to create the requested format described in the problem
# the possible orient options are split, records, index, and table.
df_nba.to_json('nba_shooting.json', orient='split', indent=4)
```

Here are the saved images:

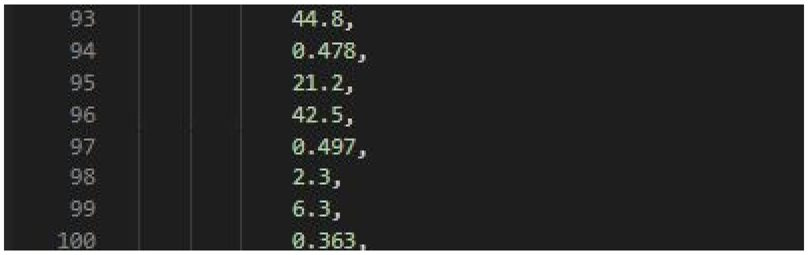
|----|----| | 

mod3
.ipynb_checkpoints
livecoding
liveassignment.ipynb
assignment3_results_hdm5s.html
assignment3.ipynb
labassignment3.pdf
nba_shooting.json
mod4



UVa > ds6001 > mod3 > {} nba_shooting.json > ...

```
{
    "columns":[ ···
    ],
    "index":[
        0,
        1,
        2,
        3,
        4,
        5,
        6,
        7,
        8,
        9,
        10,
        11,
        12,
        13,
        14,
        15,
        16,
        17,
        18,
        19,
        20,
        21,
        22,
        23,
        24,
        25,
        26,
        27,
        28,
        29
    ],
    "data":[
        [
            "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,
```