BetterDoctor Coding Challenge

Contents

Problem Statement: 2
Approach: 3
Getting started with the project: 8
Output Screenshot: 8
Given the time, additional Enhancement: 9

Problem Statement:

- source_data.json: JSON file that has clean normalized data used as the source for the matching.
- match_file.csv: Raw source data that needs to be parsed and normalized.

Match the data based on the following criteria:

Doctor Match

- NPI
- first name + last name + full address

Practice Match

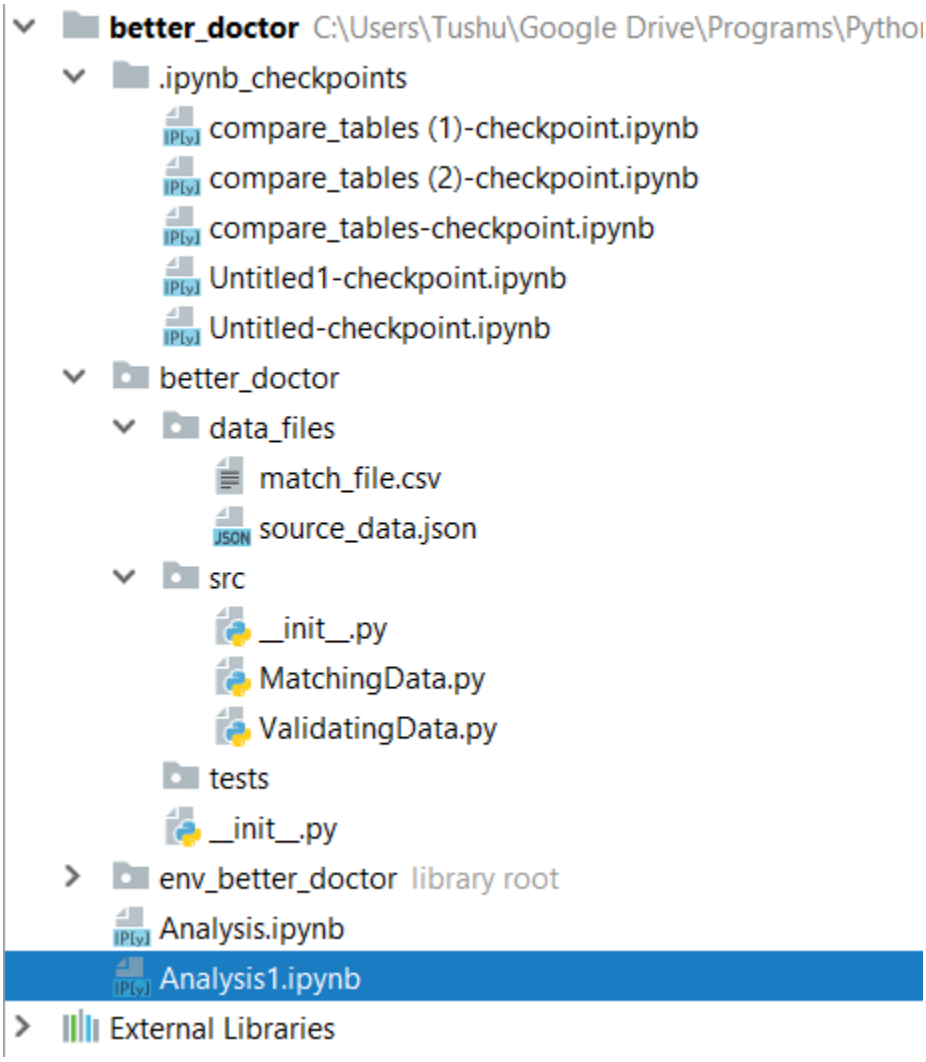
- NPI
- full address

Output should have the following at a minimum

- # of total documents scanned
- # of Doctors matched with NPI
- # of Doctors matched with name and address
- # of Practices matched with NPI
- # of Practice matched with address
- # of documents that could not be matched

Approach:

1. Project tree structure



2. Converting the Json file into flattened dataframe
   The below function extract the practice content of the json file and convert it into Data frame.

Input: Json file


```
{
    "doctor": {
        "first_name": "Sidney",
        "last_name": "Juliana",
        "npi": "264496552149966485
    },
    "practices": [{
        "street": "92136 Welch Cir
        "street_2": "Apt. 554",
        "zip": "76089",
        "city": "Port Herta",
        "state": "WI",
        "lat": "17.180927689819825
        "lon": "25.76177142576256"
    },
    {
        "street": "95750 Goldner H
        "street_2": "Apt. 224",
        "zip": "77798",
        "city": "Keshawnmouth",
        "state": "AR",
        "lat": "71.421222883685",
        "lon": "-79.33047544158335
    }]
{
    "doctor": {
```


Output: Dataframe

	first_name	last_name	npi	lat
0	Dean	Israel	85103080143784778415	-79.8757664338564
1	Quinton	Mollie	36233383542350521233	81.37417480720865
2	Quinton	Mollie	36233383542350521233	69.84837521604314

	lon	street	street_2	city
0	84.31253504872467	271 Annabelle Fort	Apt. 404	Port Demetris
1	-95.33450729432164	8496 Kennedi Inlet	Suite 815	Nealville
2	87.36942972635728	29483 Nader Wall	Apt. 748	Rashadborough
3	177.28706015725533	2122 Wintheiser Valleys	Suite 855	South Daronland

	state	zip
0	LA	53549
1	OR	52665-6811
2	UT	46006-3437
3	AK	99372

Function definition:

```
def extarcting_practice_json(self,file):
    columns = ['first_name', 'last_name', 'npi', 'lat', 'lon', 'street', 'street_2',
'city', 'state', 'zip']
    fname_list = []
    lname_list = []
    npi_list = []
    lat_list = []
    lon_list = []
    street_list = []
    street_2_list = []
    city_list = []
    state_list = []
    zip_list = []
    with open(file, 'r+') as infile:
        for line in infile:
            line = line.replace("\\", r"\\")
            line = json.loads(line)
            for index, elem in enumerate(line['practices']):
                fname_list.append(line['doctor']['first_name'])
                lname_list.append(line['doctor']['last_name'])
                npi_list.append(line['doctor']['npi'])
                lat_list.append(line['practices'][index]['lat'])
                lon_list.append(line['practices'][index]['lon'])
                street_list.append(line['practices'][index]['street'])
                street_2_list.append(line['practices'][index]['street_2'])
                city_list.append(line['practices'][index]['city'])
                state_list.append(line['practices'][index]['state'])
                zip_list.append(line['practices'][index]['zip'])
    new_json_data = pd.DataFrame(
    [fname_list, lname_list, npi_list, lat_list, lon_list, street_list, street_2_list,
city_list,
    state_list,zip_list]).T
    new_json_data.columns =[columns]
    return new_json_data
```

3. The below function extract the doctor content of the JSON file and convert it into Data frame.
   Input: JSON file
   Output: Dataframe

   Function definition:

```
def extarcting_doctor_json(self,file):
    columns = ['first_name', 'last_name', 'npi']
    unique_fname_list = []
    unique_lname_list = []
    unique_npi_list = []
    with open(file, 'r+') as infile:
        for line in infile:
            line = line.replace("\\", r"\\")
            line = json.loads(line)
            for index, elem in enumerate(line['practices']):
                if line['doctor']['npi'] not in unique_npi_list:
                    unique_fname_list.append(line['doctor']['first_name'])
                    unique_lname_list.append(line['doctor']['last_name'])
                    unique_npi_list.append(line['doctor']['npi'])
    new_json_data = pd.DataFrame(
    [unique_fname_list,unique_lname_list,unique_npi_list]).T
    new_json_data.columns =[columns]
    return new_json_data
```

4. Used IPython Notebook to perform exploratory analysis on data. Observed that the JSON file did not had any null values but the CSV file had null values which needs to be handled.

```
In [302]: missing_df
```

Out[302]:

	index	0
0	first_name	0
1	last_name	0
2	npi	287
3	street	230
4	street_2	230
5	city	230
6	state	230
7	zip	230

5. The below function takes a list and clean the data, for example, it changes the data to lower case and then removes any special character in the fields and finally creates a single field with the spaces removed.

   Function definition:

```
def clean_lower_data(self,field):
    field=field.apply(lambda x: x.lower())
    field=field.apply(lambda x: regex.sub('', x))
    field=field.apply(lambda x: x.replace(' ', ''))
    return field
```

6. The below function merged the data of the column specified by the column indices given in the function. It handled 'NaN' values by replacing it with '_'. This column can be used to perform joins.

   Input: **a. File Name**
   b. Column name to be created
   c. List of column indices

   Output: File with new column appended.

   Function definition:

```
def create_merged_column(self,file,column_name,columnList=[]):
    add=[]
    for i in columnList:
        address = []
        address= file.apply(lambda row: row.iloc[i] if pd.notnull(row.iloc[i])
else '_', axis=1 )
        if len(add)==0:
            add=address
        else :
            add=add+address
    file[column_name]=self.clean_lower_data(add)
    return file
```

Getting started with the project:

1. Download the source code from Github
2. Run the ValidatingData.py

Output Screenshot:

```
# of json object scanned: (22443, 10)
# of CSV object scanned: (1265, 8)
# of number of Doctors matched with NPI: 864
# of Practices matched with NPI: 1718
# of Doctors matched with name and address: 921
# of Practice matched with address: 921
# of Practice matched with Name and NPI: 1718
# of unmatched Document when matched on Name and NPI: 21413
```

Output Explanation:

1. 22443 rows of data scanned from Jason file
2. 1265 rows of data scanned from CSV file
3. Number of doctors matched by NPI is 864
4. Number of doctor practices matched by NPI is 1718
5. Number of doctor matched by Name and address is 921
6. Number of doctor matched with Name and NPI us 1718
7. Total number of unmatched document when matched on Name and NPI is 21413.

Unmatched Values:

first_name_x	401
last_name_x	401
npi_x	401
lat	401
lon	401
street_x	401
street_2_x	401
city_x	401
state_x	401
zip_x	401
name_npi	0
first_name_y	20725
last_name_y	20725
npi_y	21012
street_y	21120
street_2_y	21120
city_y	21120
state_y	21120
zip_y	21120

The above table shows the unmatched values in CSV and JSON file and there are higher unmatched values in JSON file, it is evident from the fact that the total rows in CSV is far lesser than that of JSON

Given the time, additional Enhancement:

1. Could have made the code more robust with unit test cases.
2. Could perform more deeper analysis to get more accurate results.