Milestone 4

July 30, 2023

Milestone 4

[147]:
```python
import requests
import time
import pandas as pd
import json
```

[133]:
```python
# Read in csv data
data = pd.read_csv('Sale_Prices_City.csv')
```

1. Create a function to pull city data from API

[148]:
```python
"""
This function takes the API key from the APIKeys.json in order to secure the␣
↪API key
"""
with open('APIKeys.json') as f:
    keys = json.load(f)
    XRapidAPIKey = keys['X-RapidAPI-Key']
```

[150]:
```python
"""
The function below pulls the api data.
It takes two parameters, the city and key.
The keys can be the following:

country_iso2
country_iso3
country_name
admin_name
latitude
longitude
population
city_name
"""
def get_request(city, key):
    # Defines URL for api
    url = "https://geoapi13.p.rapidapi.com/v1/city/"
    # Creates headers
    headers = {"X-RapidAPI-Key": XRapidAPIKey,
           "X-RapidAPI-Host": "geoapi13.p.rapidapi.com"}
    # Pulls URL and uses city to get request
    response = requests.get(url+city, headers=headers)
    if response.status_code == 200:
        try:
            # Reads json
            data = response.json()
            # Sets the key in where the data lies
            subset = data['cities'][0][key]
            print('Done!')
            # Returns population
            return subset
        # Handle if city does not exist
        except IndexError:
            return 0
    # Handle for 1 second per request limit
    elif response.status_code ==429:
        # Wait for 1 seconds before sending the next request
        time.sleep(1)
        # Attempts to get value again
        return get_request(city, key)
    # Diplays other errors
    else:
        print("Error:", response.status_code, response.reason)
        return None
```

2. Create a list of cities that population will be pulled for

[7]:
```
cities = data['RegionName']
```

[7]:
```
   Unnamed: 0  RegionID   RegionName   StateName  SizeRank   2008-03  \
0           0      6181     New York    New York         1       NaN
1           1     12447  Los Angeles  California         2  507600.0
2           2     39051      Houston       Texas         3  138400.0
3           3     17426      Chicago    Illinois         4  325100.0
4           4      6915  San Antonio       Texas         5  130900.0

    2008-04   2008-05   2008-06   2008-07  …   2019-06   2019-07   2019-08  \
0       NaN       NaN       NaN       NaN  …  563200.0  570500.0  572800.0
1  489600.0  463000.0  453100.0  438100.0  …  706800.0  711800.0  717300.0
2  135500.0  132200.0  131000.0  133400.0  …  209700.0  207400.0  207600.0
3  314800.0  286900.0  274600.0  268500.0  …  271500.0  266500.0  264900.0
4  131300.0  131200.0  131500.0  131600.0  …  197100.0  198700.0  200200.0

    2019-09   2019-10   2019-11   2019-12   2020-01   2020-02   2020-03
0  569900.0  560800.0  571500.0  575100.0  571700.0  568300.0  573600.0
```

```
1  714100.0  711900.0  718400.0  727100.0  738200.0  760200.0       NaN
2  207000.0  211400.0  211500.0  217700.0  219200.0  223800.0       NaN
3  265000.0  264100.0  264300.0  270000.0  281400.0  302900.0  309200.0
4  200800.0  203400.0  203800.0  205400.0  205400.0  208300.0       NaN

[5 rows x 150 columns]
```

[37]:
```
cities = data['RegionName'].tolist()
```

3. Create a loop that will extract population for the cities in the list

[ ]:
```
"""
The populations will need to be added into a list.
The loop below pulls the population from the list of cities and adds it to the␣
 ↪population list
"""
# Creates list
population = []
# Iterates through cities
for city in cities:
    # Extracts population
    number = get_request(city, 'population')
    # Adds population to list
    population.append(number)
```

4. Make the cities and population into a dictionary to prep for merging

[134]:
```
"""
Creates a dictionary of the cities and populations.
This will be needed to merge to the main set.
"""
d = {'RegionName': cities,
       'Population': population}
```

[135]:
```
"""
Since the Cities and Populations are different lengths, we need to account for␣
 ↪all the blanks
when converting into a dataframe.
"""
df = pd.DataFrame(dict([ (k,pd.Series(v)) for k,v in d.items() ]))
```

[136]:
```
df.head()
```

[136]:
```
     RegionName  Population
0      New York  18713220.0
1   Los Angeles  12750807.0
```

```
2      Houston      6430.0
3      Chicago   8604203.0
4  San Antonio     86239.0
```

5. Merge the data on Region Name to add population to the original data

[137]:
```
"""
The following joins the populations to the main dataset.
This is done using RegionName as an index.
"""
data = data.join(df.set_index('RegionName'), on='RegionName')
```

[138]:
```
data.head()
```

[138]:
```
   Unnamed: 0  RegionID   RegionName   StateName  SizeRank   2008-03  \
0           0      6181     New York    New York         1       NaN
1           1     12447  Los Angeles  California         2  507600.0
2           2     39051      Houston       Texas         3  138400.0
3           3     17426      Chicago    Illinois         4  325100.0
4           4      6915  San Antonio       Texas         5  130900.0

    2008-04   2008-05   2008-06   2008-07  …   2019-07   2019-08   2019-09  \
0       NaN       NaN       NaN       NaN  …  570500.0  572800.0  569900.0
1  489600.0  463000.0  453100.0  438100.0  …  711800.0  717300.0  714100.0
2  135500.0  132200.0  131000.0  133400.0  …  207400.0  207600.0  207000.0
3  314800.0  286900.0  274600.0  268500.0  …  266500.0  264900.0  265000.0
4  131300.0  131200.0  131500.0  131600.0  …  198700.0  200200.0  200800.0

    2019-10   2019-11   2019-12   2020-01   2020-02   2020-03  Population
0  560800.0  571500.0  575100.0  571700.0  568300.0  573600.0  18713220.0
1  711900.0  718400.0  727100.0  738200.0  760200.0       NaN  12750807.0
2  211400.0  211500.0  217700.0  219200.0  223800.0       NaN      6430.0
3  264100.0  264300.0  270000.0  281400.0  302900.0  309200.0   8604203.0
4  203400.0  203800.0  205400.0  205400.0  208300.0       NaN     86239.0

[5 rows x 151 columns]
```

6. Remove null from Population column

[139]:
```
"""
Here, fillna is used to fill any blank populations to 0.
This will be useful when doing calculations on the data.
"""
df = df.fillna(0)
```

7. Format the numbers to be readable to match the other datasets

[140]:
```
"""
The code below changes the population column to be formated with a comma at␣
↪each appropriate place.
This is done by using apply, to iterate through the whole column
"""
data['Population'] = data['Population'].apply(lambda x: '{:,}'.format(x))
```

[142]:
```
data.head()
```

[142]:
```
   Unnamed: 0  RegionID   RegionName   StateName  SizeRank   2008-03  \
0           0      6181     New York    New York         1       NaN
1           1     12447  Los Angeles  California         2  507600.0
2           2     39051      Houston       Texas         3  138400.0
3           3     17426      Chicago    Illinois         4  325100.0
4           4      6915  San Antonio       Texas         5  130900.0

    2008-04   2008-05   2008-06   2008-07  …   2019-07   2019-08   2019-09  \
0       NaN       NaN       NaN       NaN  …  570500.0  572800.0  569900.0
1  489600.0  463000.0  453100.0  438100.0  …  711800.0  717300.0  714100.0
2  135500.0  132200.0  131000.0  133400.0  …  207400.0  207600.0  207000.0
3  314800.0  286900.0  274600.0  268500.0  …  266500.0  264900.0  265000.0
4  131300.0  131200.0  131500.0  131600.0  …  198700.0  200200.0  200800.0

    2019-10   2019-11   2019-12   2020-01   2020-02   2020-03    Population
0  560800.0  571500.0  575100.0  571700.0  568300.0  573600.0  18,713,220.0
1  711900.0  718400.0  727100.0  738200.0  760200.0       NaN  12,750,807.0
2  211400.0  211500.0  217700.0  219200.0  223800.0       NaN       6,430.0
3  264100.0  264300.0  270000.0  281400.0  302900.0  309200.0   8,604,203.0
4  203400.0  203800.0  205400.0  205400.0  208300.0       NaN      86,239.0

[5 rows x 151 columns]
```