DATA FOCUSED PYTHON PROJECT

The Nextdoor Get to know the neighborhood!

Feb 25, 2019

Demo video link: https://youtu.be/7iDK-YnOhNw

SUMMARY

This project aims at providing an app that helps neighborhood trackers to better know about their interested neighborhoods in Pittsburgh without visiting in person. To Achieve this result, we use Python scrapped data from multiple data sources and created a web-based app providing analysis from four different dimensions. The final result is an interactive application with dynamic visualization and powerful analysis.

This readme file intends to briefly introduce the flow of this project and give instruction of how to run this app.

Requirements

To run this app, you need to install some modules first:

pip install dash==0.36.0 # The core dash backend pip install dash-html-components==0.13.5 # HTML components

pip install dash-core-components==0.43.0 # Supercharged components

pip install sklearn # Package used to run clustering model

pip install selenium ==3.141.0 #Web-scrapping

pip install beautifulsoup4 ==4.7.1#Web-scrapping

pip install urllib3 ==1.24.1 #Open URL

pip install requests ==2.21.0 #API

pip install pandas ==0.24.1 #Data frame

Instructions

The main programming file for this app is ‘gui.py’. This program requires two support python programming files: ‘cluster.py’ and ‘data_analyze.py’.

Although this app supposed to be able to scrap real-time data to provide useful analysis for customers, this demo gets an overall dataset (‘clean_all.csv) in advance to show its features since real-time web-scrapping could be time-consuming.

Therefore, you have to put these four files (‘gui.py’, ‘cluster.py’, ‘data_analyze.py’ and ‘clean_all.csv’) in the same folder and then run ‘gui.py’ in python IDLE.

After run the main file, you will get an web address in the output (‘http://127.0.0.1:8050/’). Copy this address and paste in your web browser and then you can see this app and interact with it.

Please note that the program will keep running unless you completely quit IDLE. You also need to connect to the Internet to see the app.

Code Flow

To achieve the outcome of this project, we divided into four major parts:

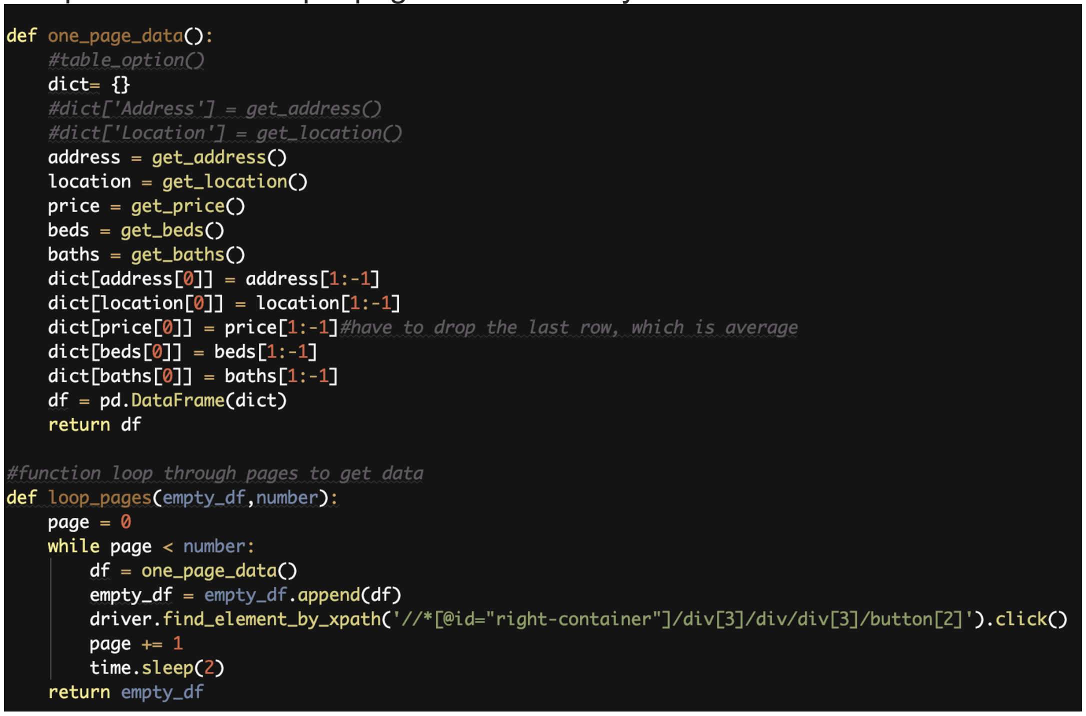
- **Acquire raw data**
    - Acquire data mainly from four different source using different methods: API, selenium, beautifulsoup.
    - Obtain three csv file containing raw data: ‘data_houseprice.csv’, ‘demographic.csv’, ‘playground_park_neighborhood.csv’
- **Clean data**
    - Use package pandas to clean data and merge all datasets using neighborhood name as primary key.
- **Analyze data**
    - Create 8 neighborhood clusters based on all the features.

        - Create data frame analyze method to support GUI.
- **Design interface**
    - Use Plotly and Dash package to create a web app.
    - Four major features:
        - Sort neighborhoods based on selected feature
        - Compare two selected neighborhoods
        - Recommend similar neighborhoods
        - Visualize 8 different neighborhood clusters based on selected features

Part 1: Acquire Raw Data

1. **House price (webscrap_houseprice.py)**

   We used selenium to scrap real-time data about selling house from Redfin (https://www.redfin.com). For this demo, it contains data at 2019.02.22.
   The web scrapping program not only is able to identify specific elements from website and write into data frame but also is able to click "next page" button to scrap data from multiple pages automatically.



```
def one_page_data():
    #table_option()
    dict= {}
    #dict['Address'] = get_address()
    #dict['Location'] = get_location()
    address = get_address()
    location = get_location()
    price = get_price()
    beds = get_beds()
    baths = get_baths()
    dict[address[0]] = address[1:-1]
    dict[location[0]] = location[1:-1]
    dict[price[0]] = price[1:-1]#have to drop the last row, which is average
    dict[beds[0]] = beds[1:-1]
    dict[baths[0]] = baths[1:-1]
    df = pd.DataFrame(dict)
    return df

#function loop through pages to get data
def loop_pages(empty_df,number):
    page = 0
    while page < number:
        df = one_page_data()
        empty_df = empty_df.append(df)
        driver.find_element_by_xpath('//*[@id="right-container"]/div[3]/div/div[3]/button[2]').click()
        page += 1
        time.sleep(2)
    return empty_df
```

2. **Playground & park (api_playground.py)**

   This is program web scrapping of a playground and park dataset of the city of Pittsburgh (http://www.wprdc.org), using geoJSON api convert the geoJSON into csv.
3. **Demographic information1(demographic.py)**

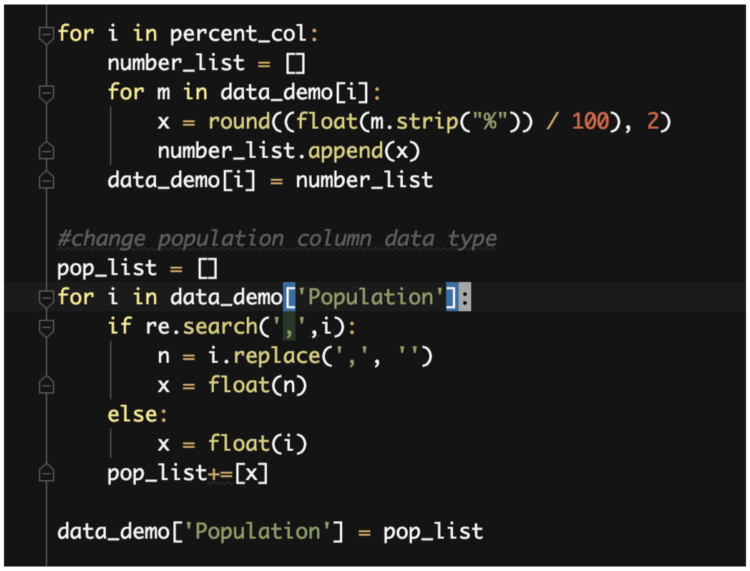
   We use beautifulsoup to scrap data from http://www.city-data.com to get some demographic data.
4. **Demographic information2**

We download xlsx file containing demographic data from www.data.gov. To see this file content, please read the 'Workbook-Final.xlsx' file.

Step 2: Clean Data

In this part, the two major challenges are:

1) Change data type of some columns. There are some columns contains special character, such as % and $. We have to translate data in these columns to number. Here, we use regular expression and string handling method to convert them.

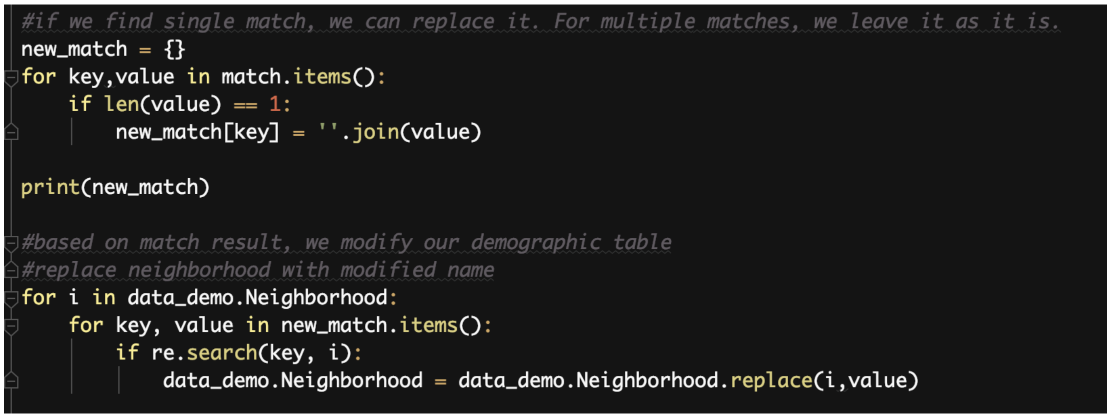


```
for i in percent_col:
    number_list = []
    for m in data_demo[i]:
        x = round((float(m.strip("%")) / 100), 2)
        number_list.append(x)
    data_demo[i] = number_list

#change population column data type
pop_list = []
for i in data_demo['Population']:
    if re.search(',',i):
        n = i.replace(',', '')
        x = float(n)
    else:
        x = float(i)
    pop_list+=[x]

data_demo['Population'] = pop_list
```

2) Merge tables. Although all the dataset contains neighborhood names, I realized that some of them are matching. For example, in demographic dataset, there are 'North Squirrel Hill' and 'South Squirrel Hill'. However, in house price file, there is only 'Squirrel Hill'. To handle this problem, I used regular expression to select all neighborhood names containing 'west', 'east', 'north', 'south', 'central' and compare them with similar names in other files. Then delete duplicate rows.



```
#if we find single match, we can replace it. For multiple matches, we leave it as it is.
new_match = {}
for key,value in match.items():
    if len(value) == 1:
        new_match[key] = ''.join(value)

print(new_match)

#based on match result, we modify our demographic table
#replace neighborhood with modified name
for i in data_demo.Neighborhood:
    for key, value in new_match.items():
        if re.search(key, i):
            data_demo.Neighborhood = data_demo.Neighborhood.replace(i,value)
```

Step 3: Analyze Data

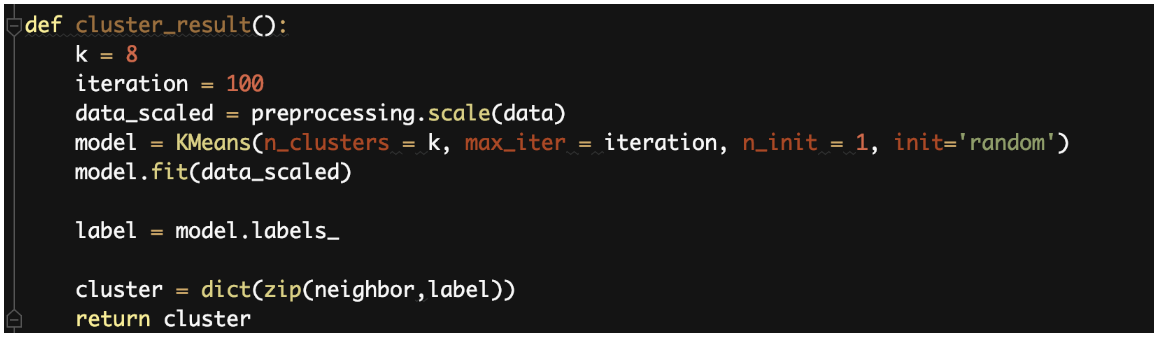
To support GUI, we create two programming files which import as module later.

1. **Data analyze.py**

   This file creates two class objects and several attributes and methods to make it much more easy for the GUI program to obtain data it needs. For example, it has get_row_bar method, which return values of selected columns of a specific row given neighborhood name.

2. **Cluster.py**

   This file used sklearn package created 8 different neighborhood clusters based on all the features from our dataset. Here we used K-means clustering, where k=8.



```
def cluster_result():
    k = 8
    iteration = 100
    data_scaled = preprocessing.scale(data)
    model = KMeans(n_clusters = k, max_iter = iteration, n_init = 1, init='random')
    model.fit(data_scaled)

    label = model.labels_

    cluster = dict(zip(neighbor,label))
    return cluster
```

Step 4: Design Interface

This program uses plotly and dash create a web-based app. It can provide several results based on user's input.