



Restaurant Recommendation System And Site Selection Minor Project Report

Submitted By: (Group No: 6)

Rishabh Tyagi :- 16803019

Prakhar Vaish :- 16803025

Shubham Aggarwal :- 16803006

Divyansh Dubey :- 16103013

Submitted To:

Mr. Mahendra Gurve

Mrs. Ankita Wadhwa

Abstract

A **recommender system** or a **recommendation system** (sometimes replacing "system" with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender Systems nowadays are very useful as they prove to be an important source of information for the people to discover an item of their choice from selected category.

Restaurant recommendation systems are one such kind of recommenders which prove to be very useful while selecting the restaurants fulfilling certain conditions.

The goal of our project is to develop an efficient recommender system for suggesting the user best restaurant satisfying given criterias of time, categories, and location, number of check-in etc.

Also, This model will help Restaurant owner to marketize and enhance their target customers.

Site selection indicates the practice of new facility location, both for business and government. Site selection involves measuring the needs of a new project against the merits of potential locations.

Developing a site selection system is an integral part of our project. Site selection enables the user to select an appropriate place for opening a new restaurant. It involves the role of different factors like check-in count in a particular venue, population density in the area and density of restaurants in the area.

This model will help the users to select the best site to open a new restaurant so that he can get the maximum number of customers.

Objectives

The restaurant recommendation system is the first part of the project which enables the user to choose the best restaurant of his given category from a number of restaurants based on a number of factors such as location(latitude,longitude), category, number of similar venues in a given radius(venues belonging to the same category) and check-in time.

Also, it enables the restaurant to choose the best of its visitors so that the restaurant can find similar customers and target them in the near future to grow their business.

The Model investigate the checkins dataset containing restaurant check-ins in New York city and report an assessment of human mobility patterns by analyzing the spatial and temporal aspects associated with these footprints. We attempt to explore the behaviours of users in New York city and then visualize the dataset to find interesting patterns. We then present a recommender system which suggests a particular set of venues (such as restaurants) to a user and then further also develop a second model for recommending a set of users to a venue, keeping in mind both user-preferences, users' location and popularity of the venues, population near the restaurant.

The second part of our project includes Site selection.This is now a days a vital approach as it would be really helpful for the people finding suitable location to open new restaurant or cafe. It would analyse may factors like number of checkin in nearby restaurants, availability of the transport facility, restaurants already opened nearby, population where the restaurant is already opened.

The Model investigate the 3 databases which includes check-in count at a restaurant, location of the restaurant and population near the restaurant. By analyzing these parameters, it would recommend a person who wishes to open a new restaurant/cafe taking in calculation all the important parameters for better business growth and sales.

Prerequisites

The algorithm used is implemented using Python version 3.x . Various python libraries like:

- Pandas
- Matplotlib
- Sklearn
- Scipy
- Numpy
- Gmplot
- Web browser

Workload Matrix

Prakhar	Designing and implementation of algorithm for recommending users to a place, visualization on gmap, fitting site selection normalized data on curve , implementation of site selection algorithm
Rishabh	Designing and implementation of algorithm for recommending place to users, Normalization of site selection data , implementation of site selection algorithm
Divyansh	Designing and implementation of algorithm for recommending users to place, regression function for check-ins , implementation of site selection algorithm
Shubham	Designing and implementation of algorithm for recommending place to users, DBScan clustering for site selection , implementation of site selection algorithm

Background Study And Findings

Haversine Distance

The **haversine formula** determines the **great-circle distance** between two points on a **sphere** given their **longitudes** and **latitudes**. Important in **navigation**, it is a special case of a more general formula in **spherical trigonometry**, the **law of haversines**, that relates the sides and angles of spherical triangles.

$$d = 2r \arcsin \left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)} \right)$$

$$= 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\varphi_2 - \varphi_1}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Probabilistic Approach

Probabilistic approaches enable variation and uncertainty to be quantified, mainly by using distributions instead of fixed values in risk assessment. A distribution describes the range of possible values (e.g. for toxicity), and shows which values within the range are most likely.

Flask Api

The flask object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an `__init__.py` file inside) or a standard module (just a `.py` file).

Suitability model

A **suitability model** is a model that weights locations relative to each other based on given criteria. Suitability models might aid in finding a favorable location for a new place using parameters like road facility, population near place, check-in count of a place, number of similar places nearby etc.

There are seven general steps required to create an acceptable suitability model:

1. Define the problem
2. Break the problem into submodels
3. Determine significant layers
4. Reclassify or transform the data within a layer
5. Weight the input layers
6. Add or combine the layers
7. Analyze

Matplotlib Library

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. Conventionally, the package is imported into the Python script by adding the following statement – `from matplotlib import pyplot as plt`

Min-Max Normalization:

Minmax normalization is a normalization strategy which **linearly transforms** x to $y = (x - \min) / (\max - \min)$, where \min and \max are the minimum and maximum values in X , where X is the set of observed values of x .

It can be easily seen that when $x = \min$, then $y = 0$, and When $x = \max$, then $y = 1$. This means, the minimum value in X is mapped to 0 and the maximum value in X is mapped to 1. So, **the entire range of values of X from min to max are mapped to the range 0 to 1.**

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based non-parametric clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

Weighted Average

A **weighted average** is a type of average where each observation in the data set is multiplied by a predetermined weight before calculation. In calculating a simple average (arithmetic mean) all observations are treated equally and assigned equal weight. A weighted average assigns weights that determine the relative importance of each data point. Weightings are the equivalent of having that many like items with the same value involved in the average.

Weighted Average = $(d_1 * w_1 + d_2 * w_2 + d_3 * w_3 + \dots + d_n * w_n) / n$

Curve Fitting :

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints. Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a "smooth" function is constructed that approximately fits the data. A related topic is regression analysis, which focuses more on questions of statistical inference such as how much uncertainty is present in a curve that is fit to data observed with random errors. Fitted curves can be used as an aid for data visualization, to infer values of a function where no data are available, and to summarize the relationships among two or more variables.

Non-Linear Regression :

Nonlinear regression is a regression in which the dependent or criterion variables are modeled as a non-linear function of model parameters and one or more independent variables. There are several common models, such as Asymptotic Regression/Growth Model, which is given by:

$$b_1 + b_2 * \exp(b_3 * x)$$

Logistic Population Growth Model, which is given by:

$$b_1 / (1 + \exp(b_2 + b_3 * x)), \text{ and}$$

Asymptotic Regression/Decay Model, which is given by:

$$b_1 - (b_2 * (b_3 * x)) \text{ etc.}$$

The reason that these models are called nonlinear regression is because the relationships between the dependent and independent parameters are not linear

Background studies and findings :

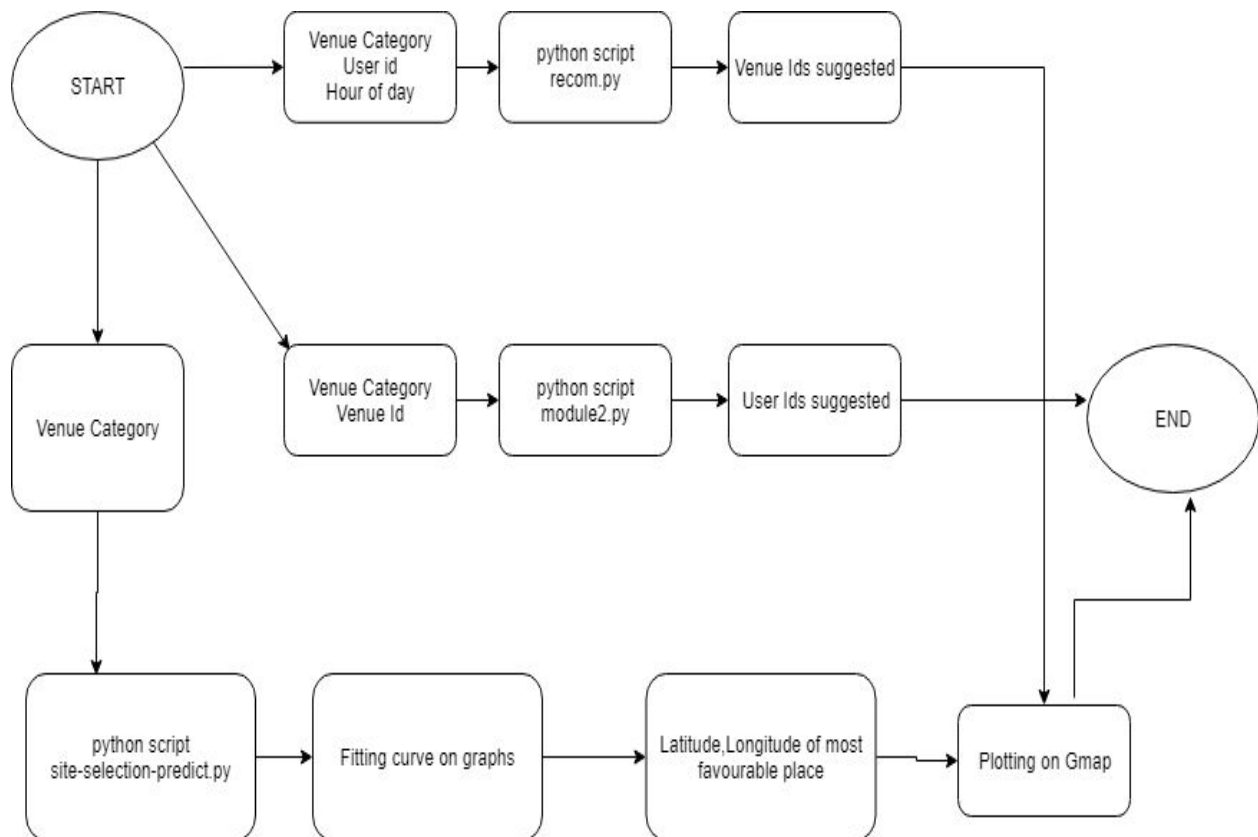
- We read various studies and got different findings such as -- “Euclidean Distance in Recommendation Engine” by A. Jeyasekar*, K. Akshay and Karan, but we found that practically, using Haversine distance is a better option to work on geo-coordinate datasets, which is our use-case.
From blogs on kdnuggets and wiki page, we studied about probabilistic model for proximity matrix vs knowledge model and used the probabilistic model.
- Different studies provided different concepts which we have compiled and used in as dataset encompasses recommendation system for different categories of venues as well.
- For site selection it has been concluded that the denser the place with respect to venues already present , the better we can target the market. To access the density we used clustering algorithm and for spatial data DBSCAN is used. Eg: Suppose there are two competing shops located along the length of a street running north and south, with customers spread equally along the street. Both shop owners want their shops to be where they will get most market share of customers. If both shops sell the same range of goods at the same prices then the locations of the shops are themselves the 'products'. Each customer will always choose the nearer shop as it is disadvantageous to travel to the farther.
- In order to access all the three parameters (population, check ins , location density), we have used suitability analysis and weighted average .
- We needed a continuous function from the discrete data set of ordered values, so we used curve fitting algorithm of non-linear regression for function with two independent variables x,y.

References :

- Using haversine distance :
 - i) Landmark Based Shortest Path Detection by Using A* and Haversine Formula
https://www.researchgate.net/profile/Mangesh_Nichat/publication/282314348_Landmark_based_shortest_path_detection_by_using_A_Algorithm_and_Haversine_Formula/links/56389bb708ae4bde5021b0f5/Landmark-based-shortest-path-detection-by-using-A-Algorithm-and-Haversine-Formula.pdf
 - ii) Privacy-preserving distance computation and proximity testing on earth, done right <https://dl.acm.org/citation.cfm?id=2590307>
- Probabilistic Approach best for Recommendation :
 - i) Personalized Recommendations using Knowledge Graphs: A Probabilistic Logic Programming Approach <https://dl.acm.org/citation.cfm?id=2959131>
- DBScan Clustering :
 - i) Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm
<https://ieeexplore.ieee.org/abstract/document/7588062>
 - ii) Revised DBSCAN algorithm to cluster data with dense adjacent clusters
<https://www.sciencedirect.com/science/article/abs/pii/S0169743912002249>
- Max-Min Normalization :
 - i) Normalization: A Preprocessing Stage :
https://www.researchgate.net/publication/274012376_Normalization_A_Preprocessing_Stage
 - ii) A state-of-the-art survey on the influence of normalization techniques in ranking: Improving the materials selection process in engineering design
<https://www.sciencedirect.com/science/article/pii/S0261306914007122>


- A Hotelling Style Model of Spatial Competition for Convenience Goods
<https://econ.ucalgary.ca/manageprofile/sites/econ.ucalgary.ca.manageprofile/files/unitis/publications/1-4055178/EatonTweedle.pdf>

Designing of System



Testing

Ad-hoc testing



Testing is done based on the skills, intuition, and experience. There are no strong test cases for this type of testing.

An example of ad-hoc testing is *exploratory testing*, which is defined like simultaneous learning, and means that tests are dynamically designed, executed, and modified. When we first look at a new feature or system, we don't know much about the system. We design experiments (or tests) to help us learn more about it. We then explore the system qualities and risks that we believe the customers, users, or other stakeholders may care about.

In our system we are doing the same thing as we don't know what the result will be and on running, the system provides us some result which we learn and analyse for further test cases .

Output of recom.py when given input "Coffee Shop" 642 16.

```
Anaconda Prompt
(base) C:\Users\Shikha\Anaconda3\Spatial-Data-Visualization-and-Recommendation-master\scripts>python recom.py "Coffee Shop" 642 16
in 5

The suggested venueids for you are:
4f3239cd19836c91c7c28fa5
4bb66c0ef159c7488ed75f7
4f874d66e4b0e6f34694b474
4fc94580d4f24895b4467dc0
4e4dcaffbd41b76be933358
4b62f569f964a520cb5a2ae3
4f1c39a5e4b0b4fdea188c7b
4f2ff4084fc63140daf45e23
4f86d1d2e4b0f1893822cc85
50b76ef0e4b0a436037703e9
4f3a5687e4b03d78318d2208
4fd5edce4b08aca4a462878
4c97b94c82b56dcbff32ecaa
4cc19e735684a35dcfd4b90d
4d03e616347da1cd746a288f
50538585e4b0a13301d39387
4e5f499918a870f60f374f78
4df7f3fd164d347cc714631
4f625ddce4b00c2a011b879e
4e651c59e4cdf1e2c0704309

(base) C:\Users\Shikha\Anaconda3\Spatial-Data-Visualization-and-Recommendation-master\scripts>
```

Activate Windows
Go to Settings to activate Windows.

Search the web and Windows

11:34 PM
3/11/2019

Output of module2.py when given input “Coffee Shop” “4ab966c3f964a5203c7f20e3”

Output of module2.py when given input “Gastropub” “4ab966c3f964a5203c7f20e2”

```
Anaconda Prompt
D:\Udemy - machinelearning\Spatial-Data-Visualization-and-Recommendation-master\scripts>python module2.py "Coffee Shop" "4ab966c3f964a5203c7f20e3"

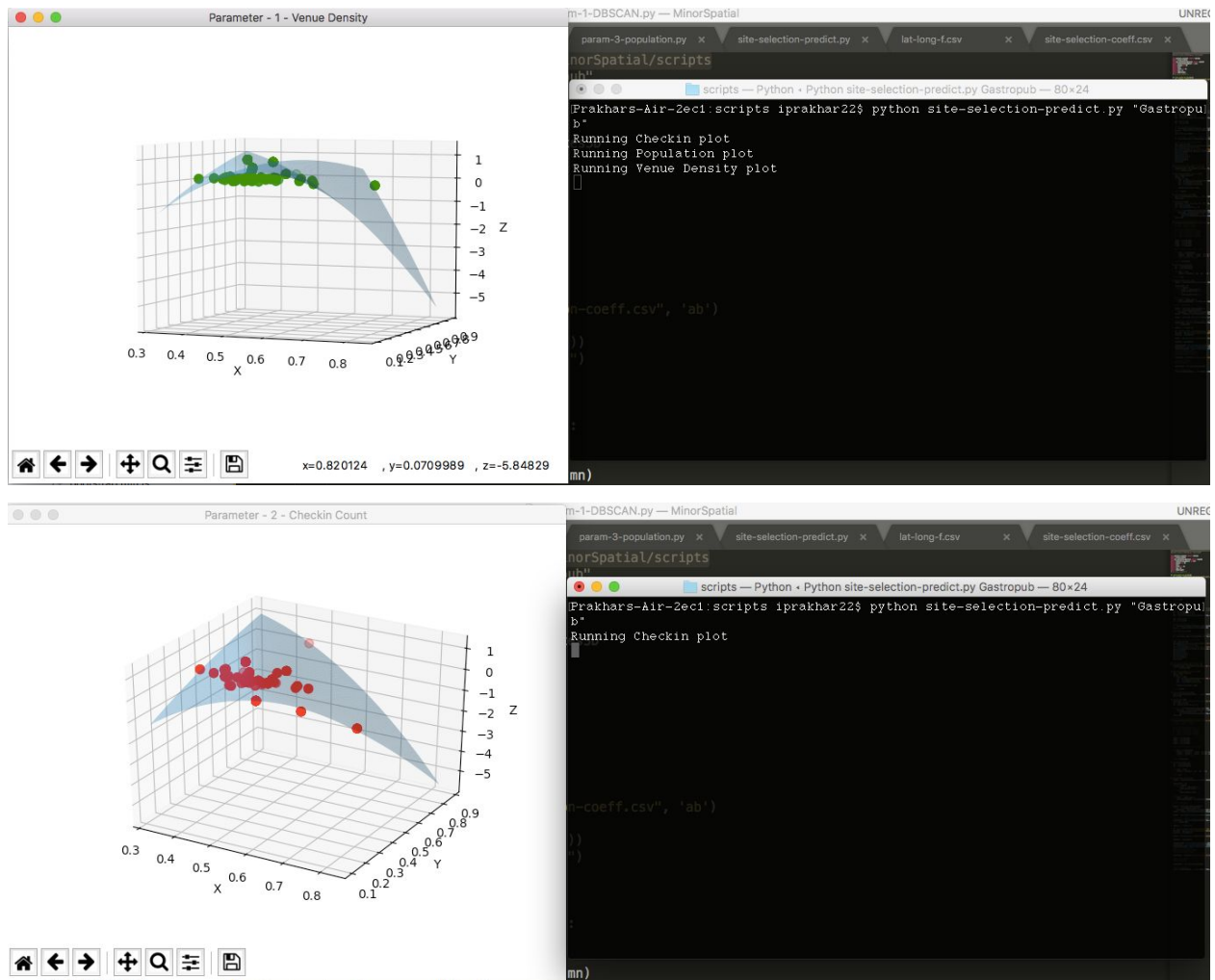
The suggested users for the venue are:
291
920
287
651
285
1006
282
1007
648
278
275
1010
657
271
658
661
919
267
276
296

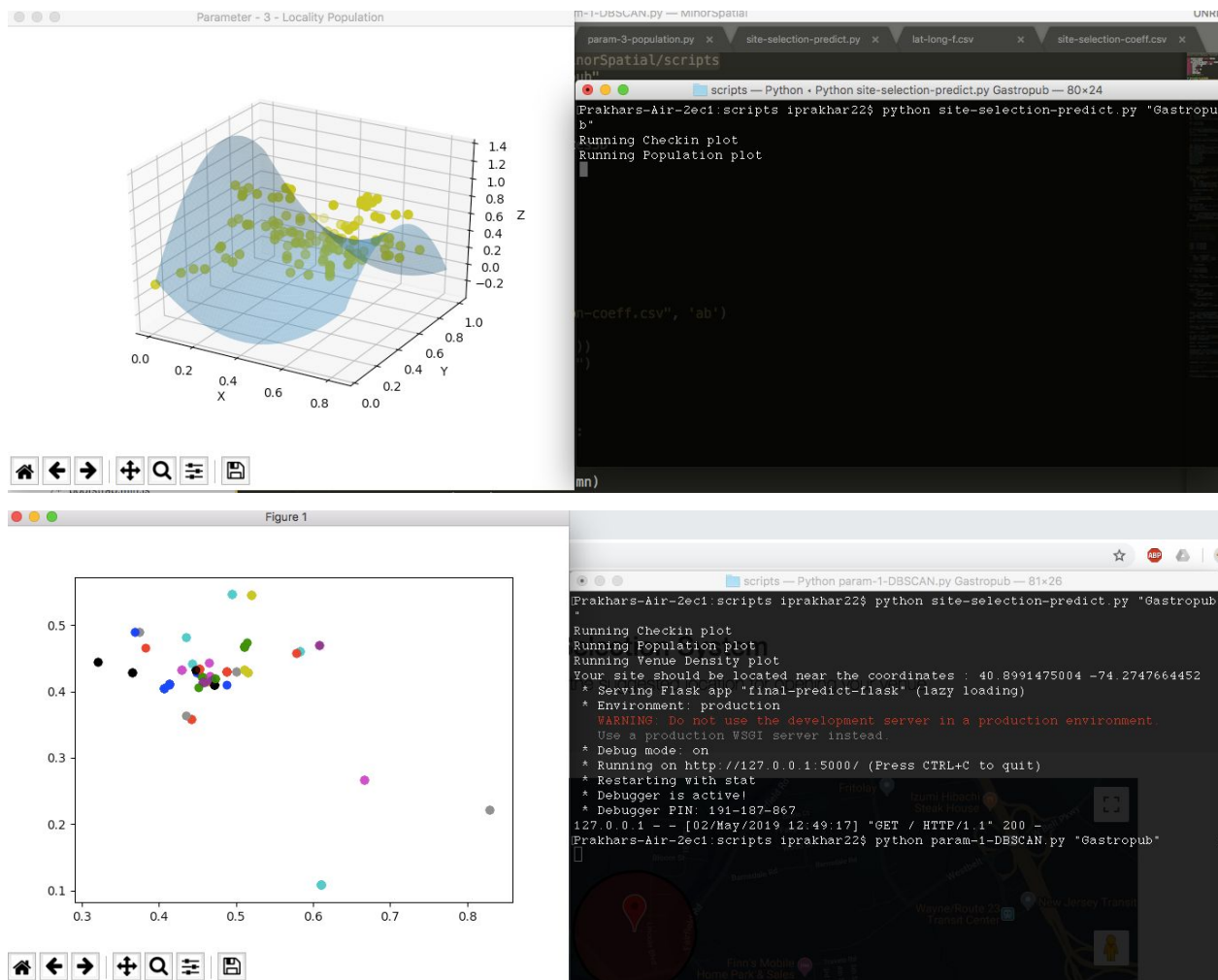
D:\Udemy - machinelearning\Spatial-Data-Visualization-and-Recommendation-master\scripts>python module2.py "Gastropub" "4ab966c3f964a5203c7f20e2"

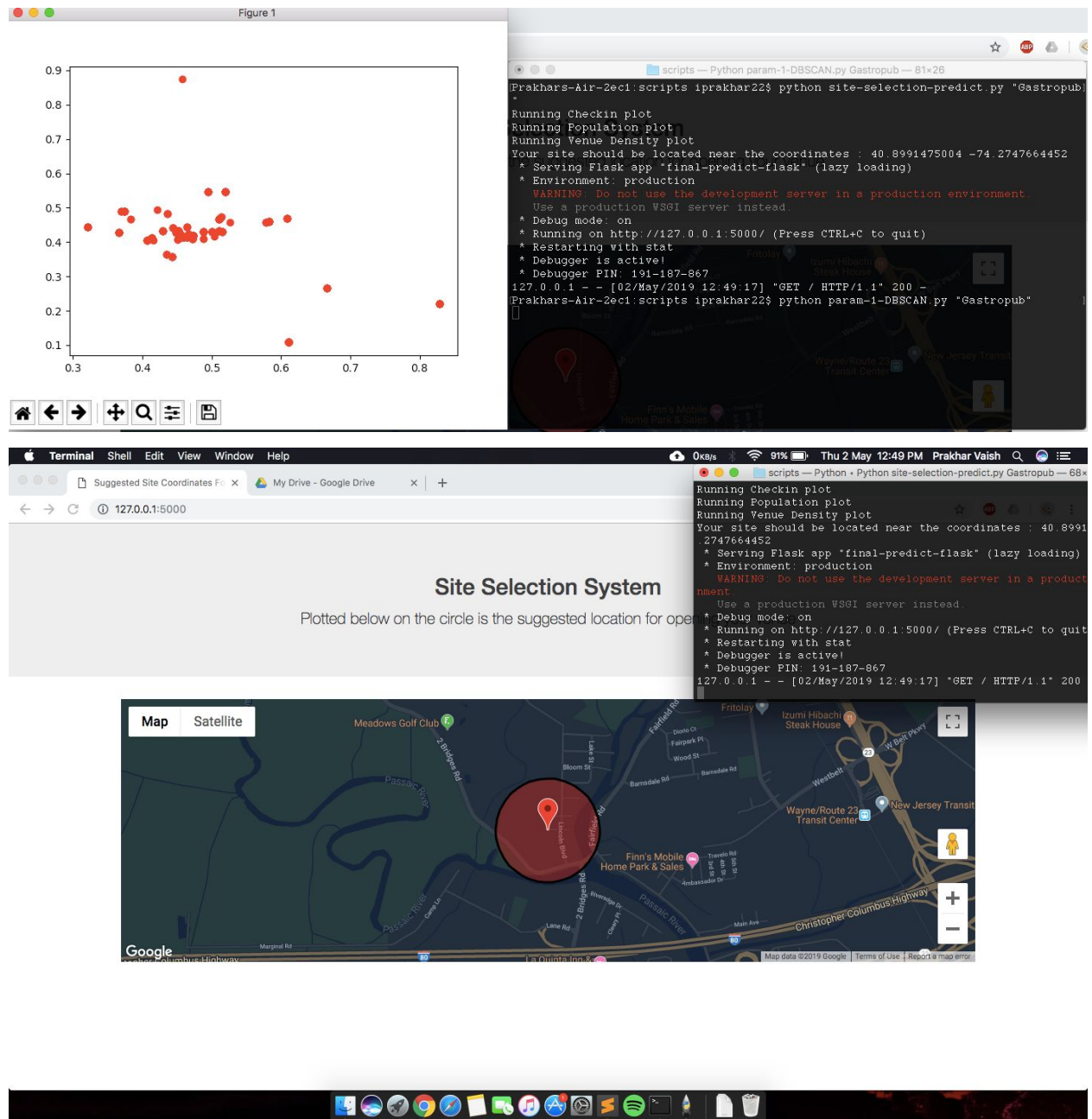
The suggested users for the venue are:
658
659
660
661
662
663
664
665
1036
667
668
669
657
670
672

Activate Windows
Go to Settings to activate Windows.
```

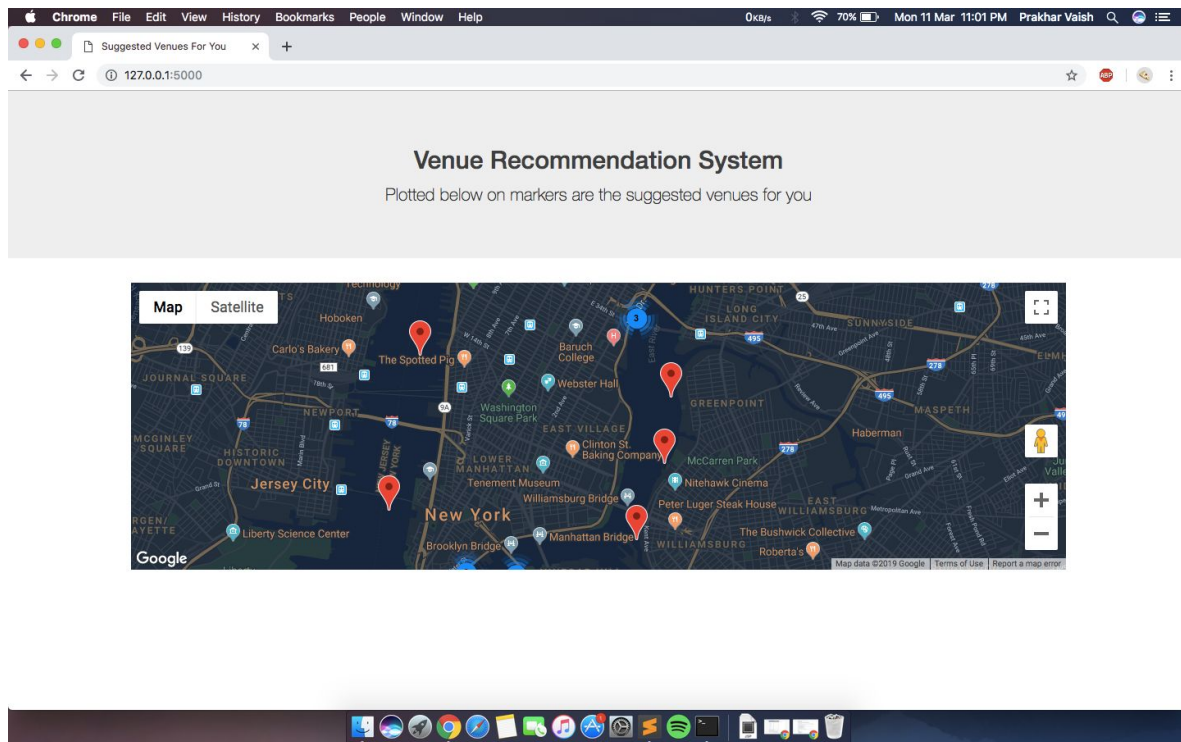
Output of site-selection-predict.py when given input "Gastropub"







Map output :



Conclusion

We have tried our model on various inputs and it worked well. Then we tried reverse of our model which suggested the best users to a particular restaurant. We further enhanced our reverse model so that it gives some more accurate results.

We also created a model for site selection which can predict new sites for new restaurants for most efficient run, providing the restaurant with the best number of customers.

All the three parts of the system are working well and giving fair results for every test case.

Now, we are looking forward to write research paper on it.

References

Site-Selection Research Paper:

Erkut E, Moran SR (1991) Locating obnoxious facilities in the public sector: an application of the hierarchy process to municipal landfill siting decisions. Socioecon Plann Sci 25(2):89–102

<https://www.sciencedirect.com/science/article/pii/003801219190007E>

Landfill site selection using GIS, remote sensing and multicriteria decision analysis: case of the city of Mohammedia, Morocco .

<https://link.springer.com/article/10.1007/s00254-005-0075-2>

Recommendation System Research Paper:

System and method for locating and notifying a user of a person, place or thing having attributes matching the user's stated preferences.

<https://patents.google.com/patent/US7071842B1/en>

A sentiment-enhanced personalized location recommendation system

<https://dl.acm.org/citation.cfm?id=2481505>

The Adaptive Place Advisor: A Conversational Recommendation System

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.1032&rep=rep1&type=pdf>

Other References :

<https://www.kdnuggets.com/2017/08/recommendation-system-algorithms-overview.html>

<https://pdfs.semanticscholar.org/88eb/985e6ed34a5f9497915e3814c562b5a88044.pdf>

https://www.researchgate.net/publication/312829358_A_Restaurant_Recommendation_System_by_Analyzing_Ratings_and_Aspects_in_Reviews

Implementation

Suggesting venues to a user :(recom.py)

```
import pandas as pd
```

```
import math
import numpy as np
import sys, os
import webbrowser

#select all rows with category coffee shop

def get_rows_by_category(df, category):
    category_rows = df.loc[df['venuecatname'] == category]
    return category_rows

#get check incount of each venue with category coffee shop

def get_checkin_count_per_venue_in_category(category_rows):
    temp = category_rows
    time = pd.to_datetime(temp['utctime'])

    hour = []
    for i in time:
        hour.append(i.hour)
    temp.is_copy = False
    temp['hour'] = hour
    category_rows.is_copy = False
    category_rows["count"] = ""
    category_rows["avg_time"] = ""
    venue_group = temp.groupby(['venueid', 'longitude',
'latitude'])['hour'].agg({'count': 'count', 'avg_time': 'mean'}).reset_index()
    #venue_group = pd.DataFrame({'count' : temp.groupby(['venueid', 'latitude',
'longitude']).size()}).reset_index()
    venue_group = venue_group.sort_values(['count'], ascending=False)
    return venue_group
```



```
#count number of venues per category
```

```
def get_number_of_venues_per_category(venue_group):  
    count_venue = len(venue_group['venueid'])  
    return count_venue
```

```
#count of how many times a user has gone to a particular venue
```

```
def get_sorted_user_checkin_count_at_venues(category_rows):  
    user_venue_group = pd.DataFrame({'count' :  
category_rows.groupby(['userid','venueid']).size()}).reset_index()  
    user_venue_group = user_venue_group.sort_values(['count'], ascending=False)  
    return user_venue_group
```

```
def get_number_venues_visited_by_user(user_venue_group, user):  
    user_rows = user_venue_group.loc[df['userid'] == user]  
    count_user_venue = len(set(user_rows['venueid']))  
    return count_user_venue
```

```
#list of all checkins made by a user
```

```
def get_all_user_checkins(df, user):  
    user_rows = df.loc[df['userid'] == user]  
    user_rows = pd.DataFrame({'count' :  
category_rows.groupby(['userid','venueid','latitude','longitude']).size()}).reset_index()  
    return user_rows
```

```
#Calculate the Haversine distance between two geo co-ordinates.
```

```
def haversine_dist(lat1, lon1, lat2, lon2):  
    radius = 1000 # miles  
    dlat = math.radians(lat2 - lat1)
```

```
dlon = math.radians(lon2 - lon1)
```

```
a = math.sin(dlat / 2) * math.sin(dlat / 2) + \
math.cos(math.radians(lat1)) \
* math.cos(math.radians(lat2)) * \
math.sin(dlon / 2) * math.sin(dlon / 2)
```

```
c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
d = radius * c
return d
```

#number of checkins in venue radius / total number of checkins of each user

```
def calculate_p_close(venue_group, user_rows):
```

```
    x = 0
```

```
    d = []
```

```
    x = user_rows
```

```
    denom = sum(user_rows['count'])
```

```
    p_close = []
```

```
    print len(venue_group), len(user_rows)
```

```
    counter = 0
```

```
    for index, venue in venue_group.iterrows():
```

```
        user_rows = x
```

```
        counter += 1
```

```
        print counter
```

```
        for index1, user in user_rows.iterrows():
```

```
            d.append(haversine_dist(venue['latitude'], venue['longitude'],
user['latitude'], user['longitude']))
```

```
            user_rows['distance'] = d
```

```
            maxi = max(d)
```

```

    mini = min(d)
    d = []
    user_rows = user_rows.loc[user_rows['distance'] < (maxi - mini)/3]
    p_close.append(sum(user_rows['count'])/denom)
return p_close

```

#number of checkins at a venue / number of checkins at all venues belonging to the same category

```

def calculate_p_like(category_rows, venue_group, time):
    denom = sum(venue_group['count'])
    avg_time = pd.DataFrame({'average' :
category_rows.groupby(['venueid'])['hour'].mean()}).reset_index()
    p_like = []
    #time penalty to make sure a place popular at nights is not suggested in the
morning
    for index, venue in venue_group.iterrows():
        num = venue['count']
        initial_p_like = (num / denom)
        time_diff = abs(time - venue['avg_time'])
        if time_diff <= 3:
            penalized_p_like = initial_p_like
        elif time_diff <= 6:
            penalized_p_like = initial_p_like * 0.80
        elif time_diff <= 9:
            penalized_p_like = initial_p_like * 0.65
        elif time_diff <= 12:
            penalized_p_like = initial_p_like * 0.50
        else:
            penalized_p_like = initial_p_like * 0.30

```

```

        p_like.append(penalized_p_like)

    return p_like

    #distance between venue coordinates and each of user checkin, filter checkins using
    distance threshold and count #
    """

def calculate_p_checkin(p_like, df_venue):
    for i in range(len(p_like)):
        item = p_like[i]
        count = df_venue.loc[df_venue['venueid'] == item]['checkincount']
        p_like[i].append(count)

    # Binning on checkin-count

    mx_checkin = -1
    mn_checkin = int(1e7)

    for item in p_like:
        mx_checkin = max(mx_checkin,item[2])
        mn_checkin = min(mn_checkin,item[2])

    width = (mx_checkin-mn_checkin)/10

    for item in p_like:
        for i in range(10):
            low = mn_checkin + width*i
            high = low +

    return

    """

```



```

def suggestions(p_checkin, p_close, venue_group):
    w1 = 0.6
    x = np.array(p_checkin)
    y = np.array(p_close)
    p = w1 * x * y
    q = np.argsort(p)
    venueid = []
    end = 21 if len(q) >= 21 else len(q)
    for i in range(1, end):
        index = q[i]
        venueid.append(venue_group.loc[index])
    print('\n')
    print("The suggested venueids for you are:")
    ret = []
    for venue in venueid:
        print(venue.venueid)
        ret.append(venue.venueid)
    return ret

def show_on_map(df_venue, suggested_ids):

    gmap_data = []
    count = 0

    for item in suggested_ids:
        row = df_venue.loc[df_venue['venueid'] == item]
        lat = float(row['latitude'])
        lng = float(row['longitude'])

        gmap_data.append([lat, lng, str(item)])

```

```

pd.DataFrame(gmap_data).to_csv("../data/google-maps-data.csv", index = False)

os.system("python map-flask-api.py")

if __name__ == '__main__':

    df = pd.read_csv("../data/data-ny.csv", sep = ',', header=None, names = ['userid',
    'venueid', 'venuecatid', 'venuecatname', 'latitude', 'longitude', 'timezone', 'utctime'])

    df_venue = pd.read_csv("../data/ny-checkin-count.csv", sep = ',', header=None,
    names = ['venueid', 'latitude', 'longitude', 'checkincount'])
    df_venue = df_venue[1:]

    category = sys.argv[1]
    user = int(sys.argv[2])
    time = int(sys.argv[3])
    category_rows = get_rows_by_category(df, category)
    venue_group = get_checkin_count_per_venue_in_category(category_rows)
    count_venue = get_number_of_venues_per_category(venue_group)
    user_venue_group = get_sorted_user_checkin_count_at_venues(category_rows)
    count_user_venue = get_number_venues_visited_by_user(user_venue_group, user)
    user_rows = get_all_user_checkins(df, user)
    p_close = calculate_p_close(venue_group, user_rows)
    p_like = calculate_p_like(category_rows, venue_group, time)
    suggested_ids = suggestions(p_like, p_close, venue_group)
    show_on_map(df_venue, suggested_ids)

```

Plotting results on google map :(map-flask-api.py)

```

import csv
import pandas as pd

```

```

import numpy as np
import json
from flask import Flask, jsonify, render_template
app = Flask(__name__)

@app.route("/")

def default():

    df = pd.read_csv("../data/google-maps-data.csv")

    lat = df['0'].values.tolist()
    lng = df['1'].values.tolist()
    label = df['2'].values.tolist()
    data = []
    for i in range(len(lat)):
        data.append([lat[i], lng[i], label[i]])

    return render_template('plot_suggested.html', data=map(json.dumps, data))

if __name__ == '__main__':
    app.run(debug=True)

```

Suggesting user to venue :(module2.py)

```

import pandas as pd

import math

```

```
from scipy.spatial import ConvexHull

import numpy as np

import surprise

import sys


def get_list_of_users(df):

    users = pd.DataFrame({'count' : df.groupby(['userid']).size()}).reset_index()

    #print(type(users))

    return users


def get_checkin_count_per_venue_in_category(category_rows):

    #get check in count of each venue with category coffee shop

    temp = category_rows

    time = pd.to_datetime(temp['utctime'])

    hour = []

    for i in time:

        hour.append(i.hour)

    temp.is_copy = False

    temp['hour'] = hour

    category_rows.is_copy = False

    category_rows["count"] = ""

    category_rows["avg_time"] = ""

    venue_group = temp.groupby(['venueid', 'longitude',
'latitude'])['hour'].agg({'count': 'count', 'avg_time': 'mean'}).reset_index()
```

```
#venue_group = pd.DataFrame({'count' : temp.groupby(['venueid', 'latitude',  
'longitude']).size()}).reset_index()  
  
venue_group = venue_group.sort_values(['count'], ascending=False)  
  
return venue_group
```

```
def get_rows_by_category(df, category):  
  
    category_rows = df.loc[df['venuecatname'] == category]  
  
    return category_rows
```

```
def get_user_rows(df, user):  
  
    user_rows = df.loc[df['userid'] == user]  
  
    return user_rows
```

```
def get_number_of_venues_per_category(venue_group):  
  
    #count number of venues per category  
  
    count_venue = len(venue_group['venueid'])  
  
    return count_venue
```

```
def get_number_venues_visited_by_user(user_venue_group, user):  
  
    user_rows = user_venue_group.loc[user_venue_group['userid'] == user]  
  
    count_user_venue = len(set(user_rows['venueid']))  
  
    return count_user_venue
```

```
def get_all_user_checkins(df, user):
    #list of all checkins made by a user

    user_rows = df.loc[df['userid'] == user]

    user_rows = pd.DataFrame({'count' :
category_rows.groupby(['userid','venueid','latitude','longitude']).size()}).reset_index()

    return user_rows
```

```
def get_average_users_per_venue():
    user = pd.DataFrame({'count' : df.groupby(['userid']).size()}).reset_index()

    venue = pd.DataFrame({'count' : df.groupby(['venueid']).size()}).reset_index()

    avg = len(venue) / len(user)

    return avg
```

```
def get_sorted_user_checkin_count_at_venues(category_rows):
    #count of how many times a user has gone to a particular venue

    user_venue_group = pd.DataFrame({'count' :
category_rows.groupby(['userid','venueid']).size()}).reset_index()

    user_venue_group = user_venue_group.sort_values(['count'],
ascending=False)

    return user_venue_group
```

```
def get_users_visited_venue(user_venue_group):
    venue_user = dict()

    for row in user_venue_group.iterrows():
```

```
if row[1].venueid in venue_user:

    s = set()

    s = venue_user[row[1].venueid]

    s.add(row[1].userid)

    venue_user[row[1].venueid] = s

else:

    s = set()

    s.add(row[1].userid)

    venue_user[row[1].venueid] = s


def define_alpha():

    alpha = {

        'food': 1.64,

        'nightlife': 1.61,

        'travel': 2.22,

        'work': 1.62,

        'home': 1.62,

        'shops': 1.64,

        'entertainment': 1.64,

        'art': 1.64,

        'parks': 1.68,

        'education': 1.96

    }

    return alpha
```

```
def center_of_mass(users, df):  
    avg_lat = []  
    avg_long = []  
    _users = users  
    for index, row in _users.iterrows():  
        user_rows = df.loc[df['userid'] == row['userid']]  
        avg_lat.append(pd.DataFrame.mean(user_rows['latitude']))  
        avg_long.append(pd.DataFrame.mean(user_rows['longitude']))  
    #print(avg_lat[:5])  
    _users['latitude'] = avg_lat  
    _users['longitude'] = avg_long  
    return _users  
  
def get_checkins_per_venue(df):  
    venue_checkins = pd.DataFrame({'count': df.groupby(['venueid', 'latitude',  
'longitude']).size())}.reset_index()  
    return venue_checkins  
  
def get_lat_long(venueid, venue_checkins):  
    row = venue_checkins.loc[venue_checkins['venueid'] == venueid ]  
    return row['latitude'], row['longitude']
```



```
def haversine_dist(lat1, lon1, lat2, lon2):

    """Calculate the Haversine distance between two geo co-ordinates."""

    radius = 3959 # miles

    dlat = math.radians(lat2 - lat1)

    dlon = math.radians(lon2 - lon1)

    a = math.sin(dlat / 2) * math.sin(dlat / 2) + math.cos(math.radians(lat1)) *
    math.cos(math.radians(lat2)) * math.sin(dlon / 2) * math.sin(dlon / 2)

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    d = radius * c

    return d
```

```
def get_distance_from_center_of_mass(_users, venue_lat, venue_long):

    dist = []

    for row in range(0, len(_users)):

        dist.append(haversine_dist(_users['latitude'][row],
        _users['longitude'][row], venue_lat, venue_long))

    return dist
```

```
def calculate_p_close(venueid, df, users):

    venue_checkins = get_checkins_per_venue(df)

    venue_lat, venue_long = get_lat_long(venueid, venue_checkins)

    _users = center_of_mass(users, df)

    dist = get_distance_from_center_of_mass(_users, venue_lat, venue_long)

    p_close = [1 / x ** 1.64 for x in dist]
```

```
return p_close
```

```
def calculate_p_go(count_venue, user_venue_group, users):
    #calculalte the probability of each user going to the venue

    p_go = []

    for user in users['userid']:

        count_user_venue =
get_number_venues_visited_by_user(user_venue_group, user)

        p_go.append(count_user_venue / count_venue)

    return p_go
```

```
def calculate_p_like():

    p_like = []

    return p_like
```

```
def suggestions(p_go, p_like, p_close, users):

    x = np.array(p_go)

    y = np.array(p_close)

    z = np.array(p_like)

    p = x * y * z

    q = np.argsort(p)

    userid = []

    end = 21 if len(q)>=21 else len(q)
```

```

for i in range(1,end):

    index = q[i]

    userid.append(users.loc[index])

print('\n')

print("The suggested users for the venue are:")

for user in userid:

    print(int(user.userid))


if __name__ == '__main__':

    df = pd.read_csv("../data/data-ny.csv", sep = ',', header=None, names =
['userid', 'venueid', 'venuecatid',
'venuecatname','latitude','longitude','timezone','utctime'])

    # category = "Coffee Shop"

    # venueid = '4ab966c3f964a5203c7f20e3'

    # user = 642

    category = sys.argv[1]

    venueid = sys.argv[2]

    category_rows = get_rows_by_category(df, category)

    venue_group = get_checkin_count_per_venue_in_category(category_rows)

    count_venue = get_number_of_venues_per_category(venue_group)


    user_venue_group =
get_sorted_user_checkin_count_at_venues(category_rows)

    users = get_list_of_users(df)

    #print(users['userid'])

    p_go = calculate_p_go(count_venue, user_venue_group, users)

```



```
p_close = calculate_p_close(venueid,df,users)
```

```
p_like = p_close
```

```
suggestions(p_go, p_like, p_close, users)
```