

# COURSERA CAPSTONE PROJECT: Applied Data Science

*Tithi Chattopadhyay*

# **1. Introduction**

# **2. Project Question (business problem)**

# **3. Data**

*i. Cities in India*

*ii. State Population*

*iii. Places around Each City*

*iv. State Crime Rates*

# **4. Methodology**

*i. Foursquare API*

*ii. Folium*

# **5. Results**

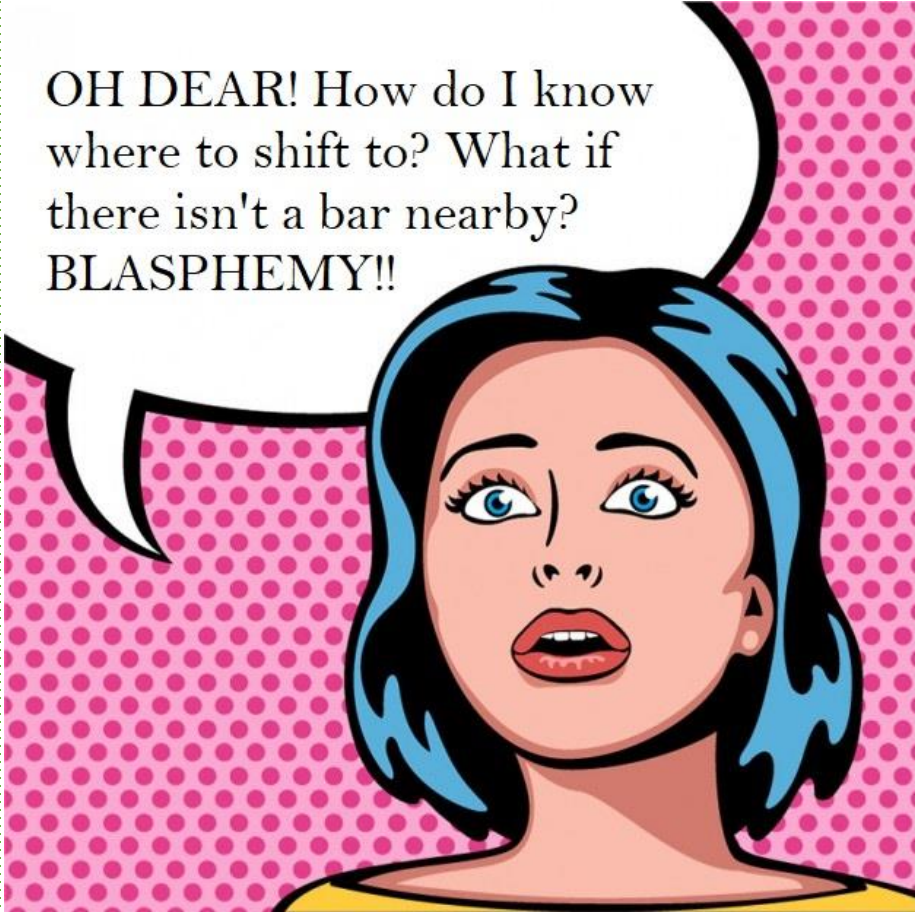
# **6. Conclusions**

# INTRODUCTION

In today's interconnected world, shifting from one place to another is very easy in terms of commute. But, how does one know where to shift to?

Sure, if you know someone in the city, you could ask them. But, in order to keep all your options open, you'll have to know a person from each city in the country then...

Every minute, 25-30 people in India, migrate from rural areas to cities in hopes of a better livelihood and higher standard of living.



OH DEAR! How do I know where to shift to? What if there isn't a bar nearby?  
**BLASPHEMY!!**

# BUSINESS PROBLEM

Where would one shift to, and where *should* one shift to?

If one's shifting for the bustling life and commerce that comes with city life, surely it would be important to ensure that there are enough shops, stations and restaurants nearby.

Moreover, particularly for women, it is also important to ensure that the city one is contemplating is safe enough.

This project aims at providing all the options to people, on where to shift, keeping in mind the criteria provided, so that one can make an informed decision

# DATA

## 1. Cities In India:

794 cities and their coordinates were scraped from LatLong.net using beautiful soup.

This is to help us later when we display the cities on the folium

```
In [8]: x= ['https://www.latlong.net/category/cities-102-15.html', 'https://www.l
```

```
In [9]: from bs4 import BeautifulSoup
```

```
In [10]: n1=0
count=0
for i in range(0,8):
    URL= x[i]
    page = requests.get(URL)
    soup = BeautifulSoup(page.content, 'html.parser')
    a= soup.find('table')
    r= a.find_all('tr')
    for r1 in r:
        r2= r1.find_all('td')
        n2=0
        for r3 in r2:
            count=count+1
            arr[n1][n2]=str(r3)
            n2=n2+1
        n1=n1+1
```

## 2. State Population

Instead of city population, state population was seen then, because the figures of the latter are more reliable.

The data was parsed from the Wikipedia page.

Consequently, all places with very low population ( $<350/\text{km}^2$ ) were removed, with the assumption that one would not want to shift to a very sparsely populated area

```
In [30]: arr= np.full([40, 2], None)
URL= 'https://en.wikipedia.org/wiki/List_of_states_and_union_territories_'
page = requests.get(URL)
soup = BeautifulSoup(page.content, 'html.parser')
a= soup.find('table', class_='wikitable sortable')
```

```
In [ ]:
```

```
In [73]: r= a.find_all('tr')
```

```
In [ ]:
```

```
In [84]: n=0
for r1 in r:
    r2= r1.find_all('td')
    p=0
    for r3 in r2:
        if (p==1): arr[n][0]= str(r3)
        elif (p==10): arr[n][1]= str(r3)
        p+=1
    n+=1
```

### 3. Places around each city

The 'explore' tab of Foursquare API helped look for all places around each of the cities.

As there was a lot of different types of venue, the final output is made more simple by prioritizing— highest priority is to have restaurants, stations and shops near where you live

```
In [107]: import folium
```

```
In [*]: CLIENT_ID = '33CAYK5RUW1XQ5UHL5JKC0TEPD4V3SXZCFR1NSMXJG0BCQAH' # your Fou
CLIENT_SECRET = 'NPVPUQ3CB5DKOAUOBLQKFAQHEUX3JZVTSJF5S43ZI5YNCQBB' # your
VERSION = '20201129' # Foursquare API version
LIMIT = 10000 # A default Foursquare API limit value
radius= 1000
venues=[]

for lat, long, place in zip(mergedf["Latitude"], mergedf["Longitude"], me
url= "https://api.foursquare.com/v2/venues/explore?client_id={}&clien
request= requests.get(url).json()["response"]["groups"][0]["items"]

for venue in request:
    venues.append((place, lat, long, venue['venue']['name'], venue['v
```

#### 4. Crime Rates

Again a Wikipedia page was scraped to find the crime rates in each state.

Since the absolute values are not very reliable, the ranking of each state was used. Here, I used grouping– highest crime rate group, medium crime rate group and low crime rate group.

```
In [130]: URL= 'https://en.wikipedia.org/wiki/Indian_states_and_territories_ranked_  
page= requests.get(URL)  
soup= BeautifulSoup(page.content, 'html.parser')
```

```
In [131]: a= soup.find('table', {'class': 'wikitable sortable'})  
r= a.find_all('tr')  
arr= np.full([len(r), 2], None)
```

```
In [132]: n1=0  
for r1 in r:  
    r2= r1.find_all('td')  
    n2=0  
    for r3 in r2:  
        if (n2==1):  
            arr[n1][0]=str(r3)  
        elif (n2==8):  
            arr[n1][1]=str(r3)  
        n2=n2+1  
    n1=n1+1
```



# METHODOLOGY

## 1. Foursquare API

All places were explored with Foursquare API. What we are most concerned about is the number and types of venues near each city.

```
In [109]: CLIENT_ID = '33CAYK5RUW1XQ5UHL5JKC0TEPD4V3SXZCFR1NSMXJG0BCQAH' # your Fou
CLIENT_SECRET = 'NPVPUQ3CB5DKOAU0BLQKFAQHEUX3JZVTSJF5S43ZI5YNCQBB' # your
VERSION = '20201129' # Foursquare API version
LIMIT = 10000 # A default Foursquare API limit value
radius= 1000
venues=[]

for lat, long, place in zip(mergedf["Latitude"], mergedf["Longitude"], me
url= "https://api.foursquare.com/v2/venues/explore?client_id={}&clien
request= requests.get(url).json()["response"]["groups"][0]["items"]

    for venue in request:
        venues.append((place, lat, long, venue['venue']['name'], venue['v
```

The elimination of cities belonging to states with very low population ensured that the number of API calls were within permissible limits

## 2. Folium

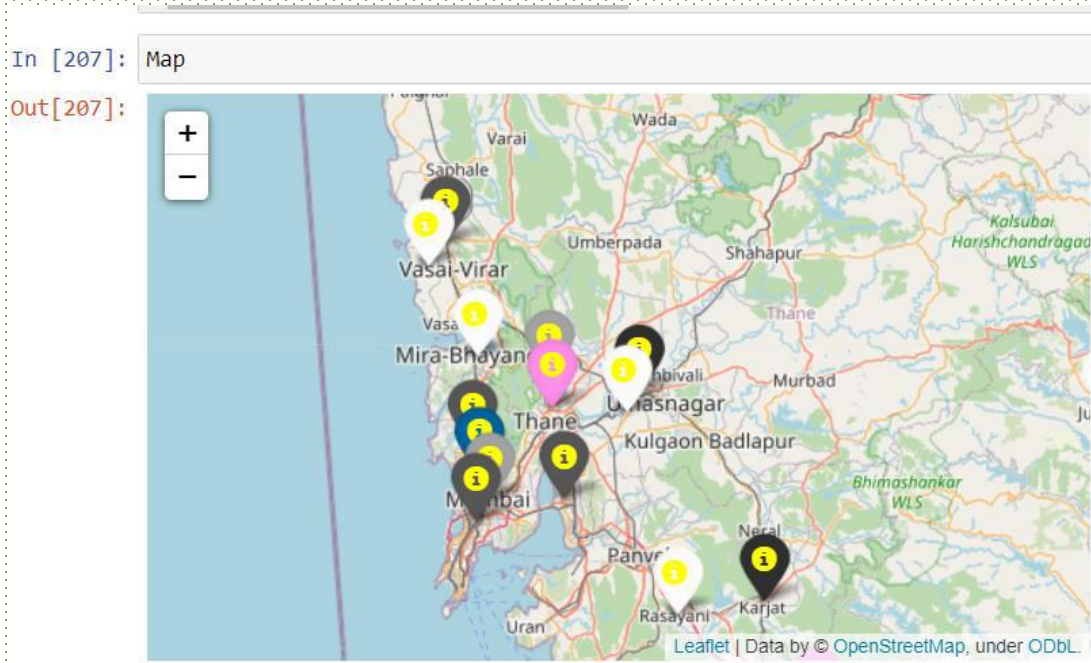
Folium was used to create a map of India, labelling all the cities according to the number of total restaurants, stations or shops nearby, with the icon on each pop-up bubble giving us an idea of the crime rate in the state. Since shops appeared in all the cities, it was assumed so while creating the colour coded markers

```
In [191]: for lat, lan, name, ven, elev, r in zip(dffin['Latitude'], dffin['Longitu']
          for a in ven:
            elev= int(elev)
            if (("Restaurant" in a) and ("Station in a")):
                if elev in range(0,10): col = 'white'
                elif elev in range(10,20): col = 'pink'
                elif elev in range(20,35): col = 'orange'
                else: col = 'lightgreen'
            elif ("Restaurant" in a):
                if elev in range(0,10): col = 'red'
                elif elev in range(10,20): col = 'lightred'
                elif elev in range(20,35): col = 'beige'
                else: col = 'lightblue'
            elif ("Station" in a):
                if elev in range(0,10): col = 'lightgray'
                elif elev in range(10,20): col = 'green'
                elif elev in range(20,35): col = 'blue'
                else: col = 'cadetblue'
            else:
                if elev in range(0,10): col = 'black'
                elif elev in range(10,20): col = 'gray'
                elif elev in range(20,35): col = 'darkblue'
                else: col = 'darkgreen'

            if (r==2.0): c="cloud"
            elif (r==3.0): c= "home"
            else: c= 'info-sign'
            folium.Marker(location=[lat,lan],popup = name,
                          icon= folium.Icon(color=col,
                          icon_color='yellow',icon = c)).add_to(Map)
```

In [192]: Map

# RESULTS



The map thus created is interpretable by the user, by the legend.

EX1: If one is against living in a state with high crime rate, then one would ignore the cities with the ‘!’ sign on the pop-up.

EX2: If one doesn’t want to live in a place with only a few shops, one would ignore the black markers altogether.

Note that the image is only of a small portion of the country, but gives an idea of what the user can expect to see once he/she zooms in on an area of the country.

# CONCLUSION

As the world becomes more interlinked through the internet, we will expect to see more and more people being open to travelling and settling in other parts of the country.

Fresh graduates and even experienced professionals hardly expect to work and live in the same city all their lives.

This project can be easily extended to include a measure of job opportunities by making an index a weighted mean of the companies nearby, weighted by their take-in rate.

Or, for example, this project can be extended by incorporating the quality (by grouping) and number of private (or government) schools near the city.

Hence, according to the needs of the user, this project can be made much more comprehensive