# Capstone Project Report

## Applied Data Science

### Introduction to the Business Problem

The problem statement here has been borrowed from online food delivery services (the likes of Swiggy and Zomato). We will first take a quick look into their business model and then discuss the problem statement.

The food delivery companies operate in a tripartite fashion. They facilitate an interaction between two parties (restaurants and consumer) with the help of a third party (delivery boy). A typical transaction starts with the customer going on the online platform (web/app) and placing an order. Once the order has been placed by the customer, two things happen in parallel. On one hand, the restaurant is notified to start preparing the order. On the other hand, a delivery boy is assigned to the order. The delivery boys then reached the restaurant and waits for the food to get prepared. As soon as the food gets prepared, the delivery boy picks it up, goes to the customer location and delivers it. We will not get into the revenue model since that is not relevant to the problem statement we are going to solve for.

Now let's focus on the problem statement we will try and solve for in this project. During the transaction, there are two legs of travel involved for the delivery boy. The first leg, where the DE travels from the location where he was assigned the order to the restaurant location. The second leg of the travel is where the DE moves from restaurant to customer location to deliver the food. Generally, the customer is charged for the second leg of travel and is more-or-less willing to pay it (since he thinks of this as a service he is buying). However, in most of the cases, the customer does not really want to pay for the first leg of travel and it ends up being the company's responsibility to optimize it. Here we will try to optimize the same through clustering the restaurants. We will try to cluster the restaurants based on their geographical location in such a manner that each cluster can be assigned a set of delivery boys exclusively. This way, when a delivery boy is assigned an order, he will only have to move within the cluster, thus reducing the distance ad avoiding long-distance travels in peak times.

The stakeholders for this problem statement will be the Operations team of these companies. They can then use this algo to optimize their travel times for the first leg of these transactions. With the number of transactions ranging to close to ~15-20 lacs per day, a saving of even a single minute per transaction will lead to immensely huge savings (assuming the delivery boys are paid at least 1 Rs per min).

### Data

As stated in the introduction, we are going to rely heavily on the geographical location of restaurants. We will be taking Bangalore as a test city and once we develop the algorithm, the same can be used to get the results for other cities as well.

To identify the geographical location for the restaurant, we will be using the latitude and longitude information for these restaurants. We will use Foursquare Venue APIs to pull this data. However, it is not as straightforward as it looks. The Foursquare Venue API gives only 50 responses in one call. This means that we cannot just run the API once and get the list of all the restaurants across Bangalore. Also, there is a limit of 1000 API calls per account per day. Which basically means we will have to be a little frugal when calling the APIs and will have to get most data with the limited number of API calls.

Here is how we deal with this situation. We basically loop the API call over multiple centre points across the city and keep storing the responses. The trick here is to select the right set of parameters for looping. There are primarily two parameters that govern this loop

1. Range (the upper and lower limits of the loop) – In this case, we will run the loop over a range of ~15 Kms as a aerial radius from the centre of the city. This is a safe assumption to make considering the physical boundaries of the city is well within this radius. In terms of lat/long definition, this comes to be +/- 0.15 lat long points.
2. Step (the step in which the loop is incremented after each run) – In this case, we need to keep it not too small (to avoid hitting the limit) as well as not too large (so that all the restaurants for the given loop and not covered in any other loop don't exceed the limit of responses i.e. 50 rest). For the purpose of this project, we will keep this increment to 1km in each step. This way, the no. of API hits needed will still be within the limits (~900 hits) as it can be safely assumed that a 1 km radius will not have more that 50 restaurants within. In terms of lat/long, this translates to 0.01 lat/long points

To avoid a lot of duplication, we will keep a limit of 1600 m on the API call radius and eventually we will also de-dup the results to ensure that the repeated responses are removed. Finally, we will remove the non-required columns and only keep the columns that either identify the row (id and name of the restaurant) or identify its location (latitude/longitude).

- Here's how the initial data pulled looks like:

```
dataframe.shape
```

```
(6230, 19)
```

```
dataframe.columns
```

```
Index(['categories', 'hasPerk', 'id', 'location.address', 'location.cc',
       'location.city', 'location.country', 'location.crossStreet',
       'location.distance', 'location.formattedAddress',
       'location.labeledLatLngs', 'location.lat', 'location.lng',
       'location.neighborhood', 'location.postalCode', 'location.state',
       'name', 'referralId', 'venuePage.id'],
      dtype='object')
```

```
dataframe.head()
```

| | categories | hasPerk | id | location.address | location.cc | location.city | location.country | location.crossStreet | locat |
|---|---|---|---|---|---|---|---|---|---|
| 0 | [{'id': '4bf58dd8d48988d10f941735', 'name': 'I... | False | 4b9da64ef964a5202db836e3 | Next to Hindu Office, Off Infantry Road | IN | Bangalore | India | NaN | |
| 1 | [{'id': '4bf58dd8d48988d10f941735', 'name': 'I... | False | 4bab6cfbf964a52084a83ae3 | #36, Noronha Road | IN | Bangalore | India | Opp. Russell Market, Shivaji Nagar | |
| 2 | [{'id': '4bf58dd8d48988d10f941735', 'name': 'I... | False | 4cb044fcdd619c748b74e390 | Residency Road, Bengaluru | IN | Bangalore | India | Residency Road, Richmond Circle | |
| 3 | [{'id': '54135bf5e4b08f3d2429dfde', 'name': 'S... | False | 4d1ee872dd6a236a55c82b38 | NaN | IN | NaN | India | NaN | |
| 4 | [{'id': '4bf58dd8d48988d10f941735', | False | 4c3197e766e40f471a43c58b | NaN | IN | Bangalore | India | NaN | |

- Here's how the data looks like after it has been cleaned:

```
[766]: dataframe_limit.shape
```

```
[766]: (975, 4)
```

```
[768]: dataframe_limit.columns
```

```
[768]: Index(['id', 'location.lat', 'location.lng', 'name'], dtype='object')
```

```
[769]: dataframe_limit.head()
```

[769]:

| | id | location.lat | location.lng | name |
|---|---|---|---|---|
| 0 | 4b9da64ef964a5202db836e3 | 12.983456 | 77.598707 | Eden Park Restaurant |
| 1 | 4bab6cfbf964a52084a83ae3 | 12.979717 | 77.602655 | Empire Restaurant |
| 2 | 4cb044fcdd619c748b74e390 | 12.967415 | 77.595938 | Rooftop Garden Restaurant, ITC Royal Gardenia,... |
| 3 | 4d1ee872dd6a236a55c82b38 | 12.972159 | 77.601039 | MTR Restaurant |
| 4 | 4c3197e766e40f471a43c58b | 12.975947 | 77.599496 | Khayal Restaurant |