The Battle of Neighbourhoods: IBM Data Science Capstone Project

Topic: Exploring the Food Culture in Melbourne
using Foursquare API
and K-Means Clustering

Submitted By: Tanay Arora

Date: 11/01/2021

Table of Contents

1. Introduction

1.1. Background

Melbourne is known as the food capital of Australia. It is considered to be a multicultural melting pot as it is a home to more than 200 nationalities, hence similar number of cuisine inspirations. Quoting from an article on the food culture of Melbourne:
“There is no authentic Melbourne dish or cuisine. Rather, the city's food options are limitless because it is a multicultural melting pot. Food is a portal into culture, and Melbourne's vibrant immigration history remains at the forefront of its culture partly because of its undeniably international cuisine.”

1.2. Problem

While on one hand having a plethora of food options is advantageous in an ethnically diverse city, however on the other hand exploring a wide array of food places might be equally time consuming for someone not so well verse with the city. Moreover, there is a fierce competition among the eateries to attract the foodies in the city, and hence new eateries keep popping on and off on Melbourne’s radar.

The aim of this project is to explore the food preferences/taste of various suburbs in Melbourne by segmenting eateries in various suburbs based on the cuisines which they offer. In this project, the Foursquare’s ‘Places API’ will be utilised to fetch venues in Each sub-urban location and further ‘K-Means’ algorithm will be used to segment these venues into clusters of similar cuisines. Moreover, a profile for each suburb location will be prepared describing the most common type of eateries using Exploratory Data Analysis (EDA), which would help to discover further about the culture and diversity of the neighborhood.

1.3 Stakeholders

Individuals

- The results from this analysis will be useful in understanding the distribution of diverse food cultures in Melbourne, which might make it easier for individuals to choose or navigate to their desired food destinations.

Businesses

- A high level understanding of the distribution of food culture of various suburbs can also be utilised by various business owners who might be planning to expand their ventures to other suburbs or open a new venture in any of the suburbs in Melbourne.

2. Data Acquisition

2.1 Melbourne City Dataset

The Melbourne City Dataset was retrieved from the following link: https://github.com/matthewproctor/australianpostcodes. The data is available in various Formats, however for the sake of simplicity the data was downloaded in form of .csv format.

The table below provides the description of the fields in the Melbourne City dataset along with an example of each field:

Field	Description	Example
id	Primary Key from source database	1
postcode	The postcode numerical format 0000-9999	3000, 3924, 3008
locality	The locality of the postcode - typically the city/suburb or postal distribution centre	Melbourne, Hawthorn, etc
state	The Australian state in which the locality is situated	VIC
long	The longitude of the locality - defaults to 0 when not available	144.956776
lat	The latitude of the locality - defaults to 0 when not available	-37.817403
dc	The Australia Post distribution Centre servicing this postcode - defaults to blank when not available	MELBOURNE
type	The type of locality, such as a delivery area, post office or a "Large Volume Recipient" such as a GPO, defaults to blank when not available	LVR
SA3	The SA3 Statistical Area code	215

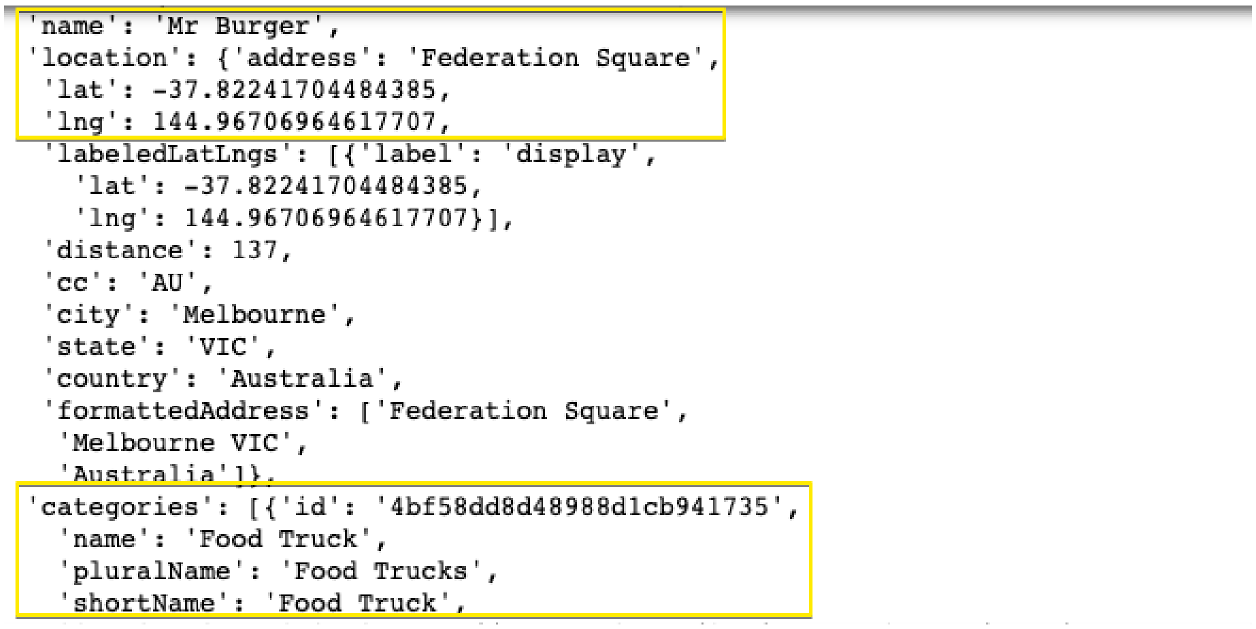
SA3 Name	The name of the SA3 Statistical Area	Melbourne City
SA4	The SA4 Statistical Area code	21501
SA4 Name	The name of the SA4 Statistical Area	Melbourne - Inner
Region	Designated Region Area	R1
status	A note indicating whether the data is new, removed or updated - new column Nov 2018	Updated

2.2 Foursquare Places API

The Places API offers real-time access to Foursquare’s global database of rich venue data and user content. Link: https://developer.foursquare.com/docs/places-api/

In this project, RESTFUL API calls will be made to the Foursquare places to retrieve nearby venues corresponding to a particular geo-location. In particular, the ‘explore’ endpoint of the API will be used, which returns a list of recommended venues in a json format near a particular location.

A sample response from one of the requests is shown below, showing a food venue near SouthBank, Melbourne:



```
'name': 'Mr Burger',
'location': {'address': 'Federation Square',
 'lat': -37.82241704484385,
 'lng': 144.96706964617707,
 'labeledLatLngs': [{'label': 'display',
   'lat': -37.82241704484385,
   'lng': 144.96706964617707}],
 'distance': 137,
 'cc': 'AU',
 'city': 'Melbourne',
 'state': 'VIC',
 'country': 'Australia',
 'formattedAddress': ['Federation Square',
  'Melbourne VIC',
  'Australia']},
'categories': [{'id': '4bf58dd8d48988d1cb941735',
  'name': 'Food Truck',
  'pluralName': 'Food Trucks',
  'shortName': 'Food Truck',
```

Some of the key fields to note here are the name, latitude, longitude values. Moreover, the API returns the categorical information of the venue. These fields will be useful in categorising the venues into different types/cuisines. More on Foursquare categories: https://developer.foursquare.com/docs/build-with-foursquare/categories

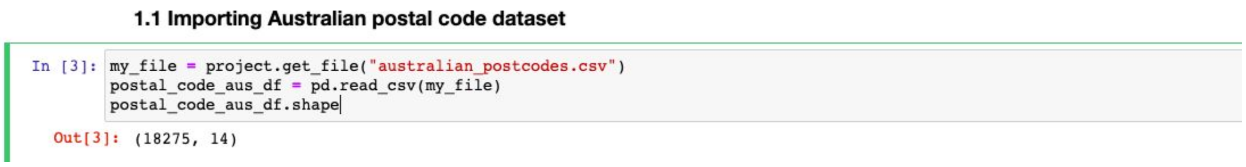
3. Methodology

3.1 Data Preparation

3.1.1 Melbourne Location Data

Importing Melbourne Location Data

The Melbourne city postcode data is downloaded in a **csv** format from the source mentioned above. The original dataset contains location and postal code information of pan Australia neighbourhoods.

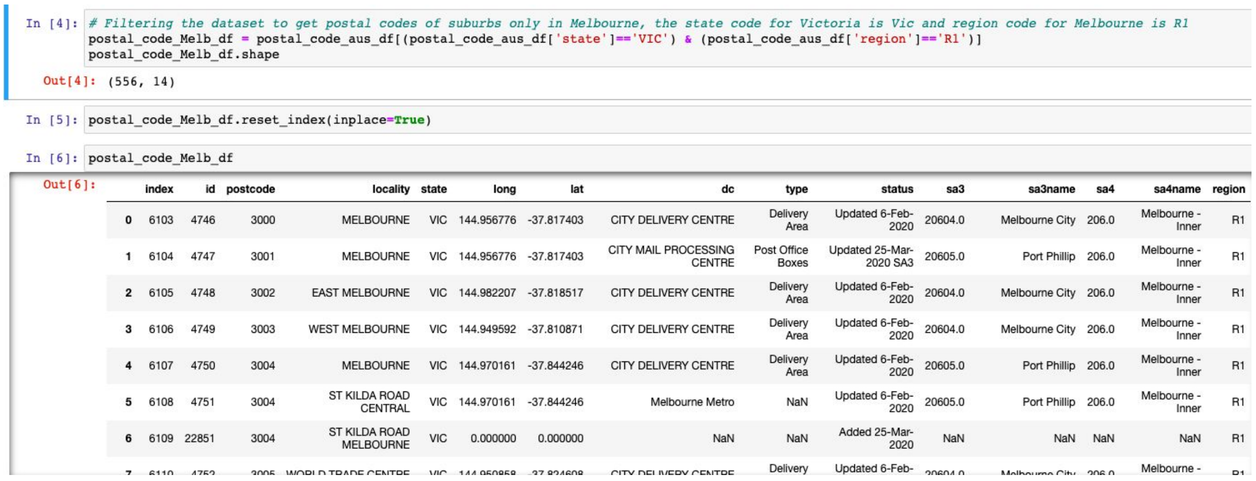


1.1 Importing Australian postal code dataset

```
In [3]: my_file = project.get_file("australian_postcodes.csv")
        postal_code_aus_df = pd.read_csv(my_file)
        postal_code_aus_df.shape

Out[3]: (18275, 14)
```

Figure 1: Importing Australian Postcode Dataset

The overall dataset comprises 18275 samples and 14 features as described above. In order to get the location data for Melbourne this dataset will be filtered over 2 columns i.e. “State =VIC ” which represents “Victoria”, the state in which Melbourne is located and “Region =R1” which represents the greater Melbourne area, other codes such as “R2”, “R3” represent the regional areas which are out of scope of this project.



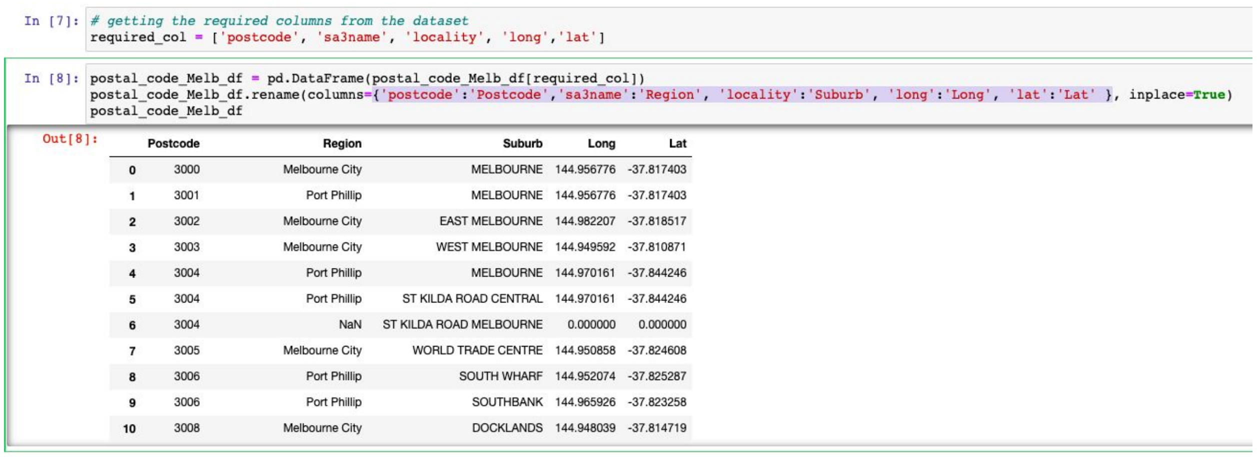
```
In [4]: # Filtering the dataset to get postal codes of suburbs only in Melbourne, the state code for Victoria is Vic and region code for Melbourne is R1
        postal_code_Melb_df = postal_code_aus_df[(postal_code_aus_df['state']=='VIC') & (postal_code_aus_df['region']=='R1')]
        postal_code_Melb_df.shape

Out[4]: (556, 14)

In [5]: postal_code_Melb_df.reset_index(inplace=True)

In [6]: postal_code_Melb_df

Out[6]:
```

	index	id	postcode	locality	state	long	lat	dc	type	status	sa3	sa3name	sa4	sa4name	region
0	6103	4746	3000	MELBOURNE	VIC	144.956776	-37.817403	CITY DELIVERY CENTRE	Delivery Area	Updated 6-Feb-2020	20604.0	Melbourne City	206.0	Melbourne - Inner	R1
1	6104	4747	3001	MELBOURNE	VIC	144.956776	-37.817403	CITY MAIL PROCESSING CENTRE	Post Office Boxes	Updated 25-Mar-2020 SA3	20605.0	Port Phillip	206.0	Melbourne - Inner	R1
2	6105	4748	3002	EAST MELBOURNE	VIC	144.982207	-37.818517	CITY DELIVERY CENTRE	Delivery Area	Updated 6-Feb-2020	20604.0	Melbourne City	206.0	Melbourne - Inner	R1
3	6106	4749	3003	WEST MELBOURNE	VIC	144.949592	-37.810871	CITY DELIVERY CENTRE	Delivery Area	Updated 6-Feb-2020	20604.0	Melbourne City	206.0	Melbourne - Inner	R1
4	6107	4750	3004	MELBOURNE	VIC	144.970161	-37.844246	CITY DELIVERY CENTRE	Delivery Area	Updated 6-Feb-2020	20605.0	Port Phillip	206.0	Melbourne - Inner	R1
5	6108	4751	3004	ST KILDA ROAD CENTRAL	VIC	144.970161	-37.844246	Melbourne Metro	NaN	Updated 6-Feb-2020	20605.0	Port Phillip	206.0	Melbourne - Inner	R1
6	6109	22851	3004	ST KILDA ROAD MELBOURNE	VIC	0.000000	0.000000	NaN	NaN	Added 25-Mar-2020	NaN	NaN	NaN	NaN	R1
7	6110	4752	3005	WORLD TRADE CENTRE	VIC	144.950858	-37.824608	CITY DELIVERY CENTRE	Delivery	Updated 6-Feb-	20604.0	Melbourne City	206.0	Melbourne -	R1

Figure 2: Filtering Melbourne location data

The dataset contains ***556*** unique locations post filtering . As observed above, there are 14 fields present in the dataset out of which only 5 fields are important for this project; they are: 'postcode', 'sa3name', 'locality', 'long','lat'. The required columns are fetched and renamed as follows for consistency:

{'postcode':'Postcode','sa3name':'Region', 'locality':'Suburb', 'long':'Long', 'lat':'Lat' }



```
In [7]: # getting the required columns from the dataset
        required_col = ['postcode', 'sa3name', 'locality', 'long','lat']

In [8]: postal_code_Melb_df = pd.DataFrame(postal_code_Melb_df[required_col])
        postal_code_Melb_df.rename(columns={'postcode':'Postcode','sa3name':'Region', 'locality':'Suburb', 'long':'Long', 'lat':'Lat' }, inplace=True)
        postal_code_Melb_df

Out[8]:
```

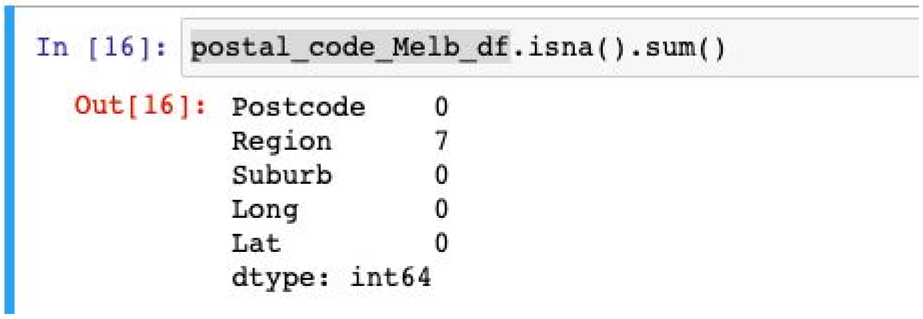
	Postcode	Region	Suburb	Long	Lat
0	3000	Melbourne City	MELBOURNE	144.956776	-37.817403
1	3001	Port Phillip	MELBOURNE	144.956776	-37.817403
2	3002	Melbourne City	EAST MELBOURNE	144.982207	-37.818517
3	3003	Melbourne City	WEST MELBOURNE	144.949592	-37.810871
4	3004	Port Phillip	MELBOURNE	144.970161	-37.844246
5	3004	Port Phillip	ST KILDA ROAD CENTRAL	144.970161	-37.844246
6	3004	NaN	ST KILDA ROAD MELBOURNE	0.000000	0.000000
7	3005	Melbourne City	WORLD TRADE CENTRE	144.950858	-37.824608
8	3006	Port Phillip	SOUTH WHARF	144.952074	-37.825287
9	3006	Port Phillip	SOUTHBANK	144.965926	-37.823258
10	3008	Melbourne City	DOCKLANDS	144.948039	-37.814719

Figure 3: Getting the required columns and renaming them

Replacing missing values

It is essential to check if there are any missing values in the dataset to keep the data set consistent for further analysis.

#Checking for NaN values:

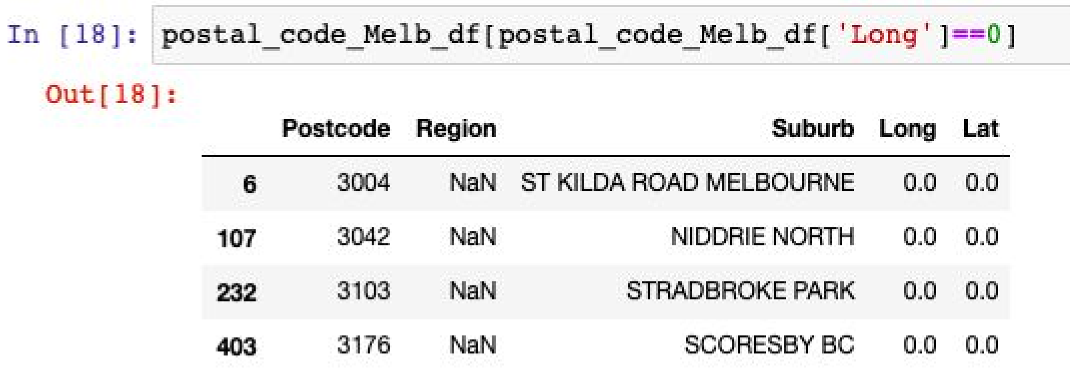


```
In [16]: postal_code_Melb_df.isna().sum()

Out[16]: Postcode    0
         Region      7
         Suburb      0
         Long        0
         Lat         0
         dtype: int64
```

Figure 4: 'NaN' Values Check

Also, it was mentioned by the data provider that there may be some of the suburbs with missing geographical data and the location data of such suburbs is populated a '0'.

# Checking for missing location data

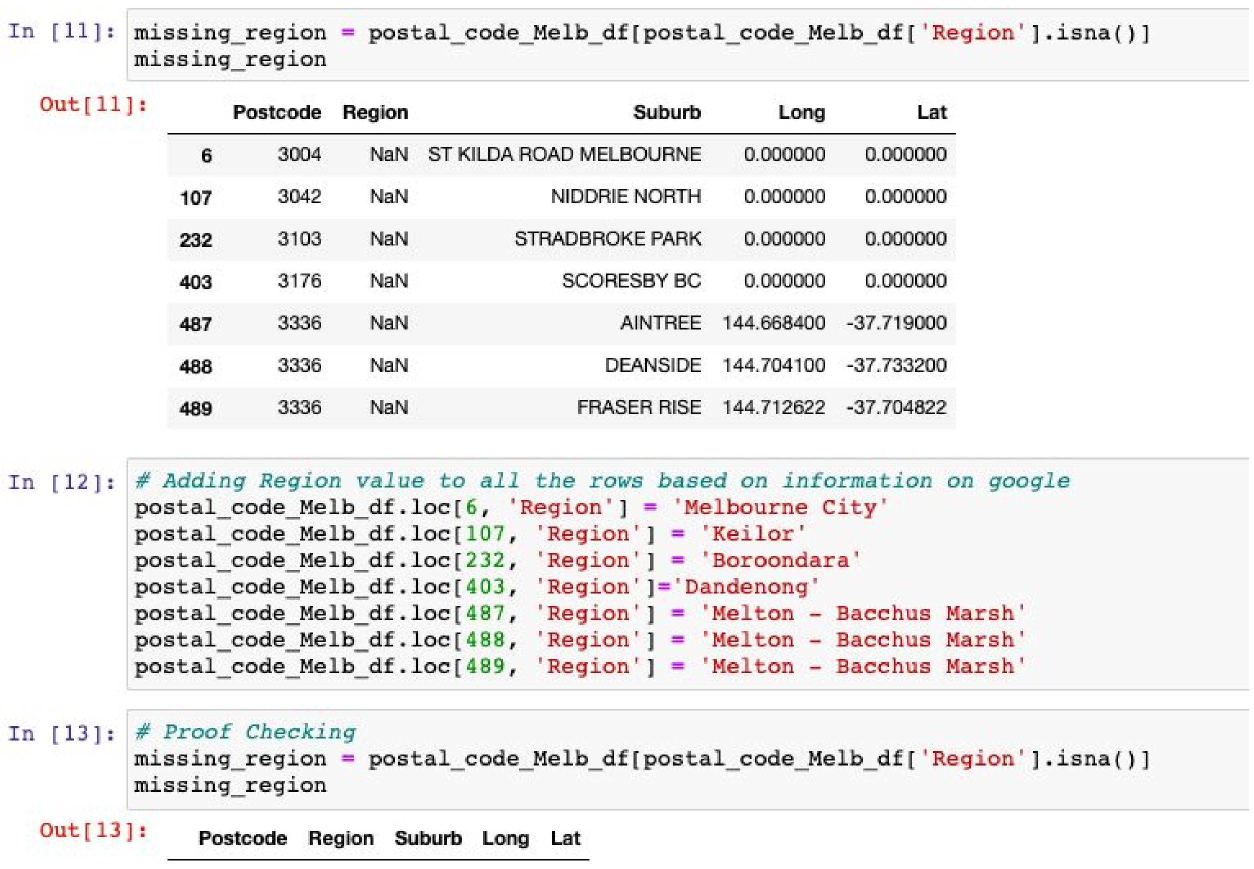


```
In [18]: postal_code_Melb_df[postal_code_Melb_df['Long']==0]

Out[18]:
```

	Postcode	Region	Suburb	Long	Lat
6	3004	NaN	ST KILDA ROAD MELBOURNE	0.0	0.0
107	3042	NaN	NIDDRIE NORTH	0.0	0.0
232	3103	NaN	STRADBROKE PARK	0.0	0.0
403	3176	NaN	SCORESBY BC	0.0	0.0

Figure 5: Locations with Longitude and Latitude Values equal to 0

# Replacing missing Region Values



```
In [11]: missing_region = postal_code_Melb_df[postal_code_Melb_df['Region'].isna()]
         missing_region
```

Out[11]:

	Postcode	Region	Suburb	Long	Lat
6	3004	NaN	ST KILDA ROAD MELBOURNE	0.000000	0.000000
107	3042	NaN	NIDDRIE NORTH	0.000000	0.000000
232	3103	NaN	STRADBROKE PARK	0.000000	0.000000
403	3176	NaN	SCORESBY BC	0.000000	0.000000
487	3336	NaN	AINTREE	144.668400	-37.719000
488	3336	NaN	DEANSIDE	144.704100	-37.733200
489	3336	NaN	FRASER RISE	144.712622	-37.704822

```
In [12]: # Adding Region value to all the rows based on information on google
         postal_code_Melb_df.loc[6, 'Region'] = 'Melbourne City'
         postal_code_Melb_df.loc[107, 'Region'] = 'Keilor'
         postal_code_Melb_df.loc[232, 'Region'] = 'Boroondara'
         postal_code_Melb_df.loc[403, 'Region']='Dandenong'
         postal_code_Melb_df.loc[487, 'Region'] = 'Melton - Bacchus Marsh'
         postal_code_Melb_df.loc[488, 'Region'] = 'Melton - Bacchus Marsh'
         postal_code_Melb_df.loc[489, 'Region'] = 'Melton - Bacchus Marsh'
```

```
In [13]: # Proof Checking
         missing_region = postal_code_Melb_df[postal_code_Melb_df['Region'].isna()]
         missing_region
```

Out[13]:

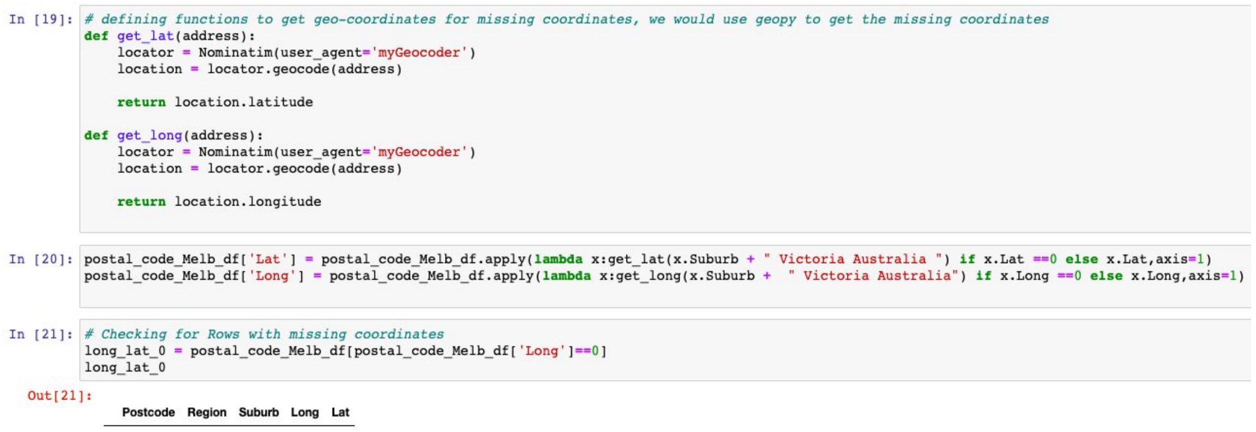
Postcode	Region	Suburb	Long	Lat

Figure 6: Replacing missing Region values

It was observed there were 7 rows without a ‘Region’ value. These value of the ‘Region’ column for these rows were replaced based on popular knowledge from Google as shown in **Figure 6**.

# Replacing missing Location Values

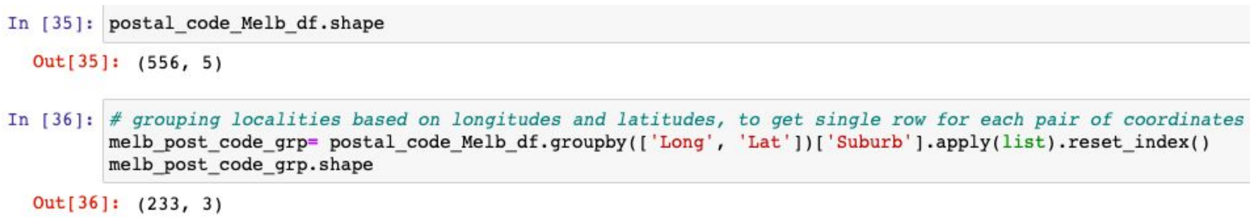
The **geopy** library was used to populate the missing geo coordinates of the suburbs. The addresses of the suburbs in string format were passed to two separate functions to retrieve the Longitudes and Latitudes of each location.



```
In [19]: # defining functions to get geo-coordinates for missing coordinates, we would use geopy to get the missing coordinates
         def get_lat(address):
             locator = Nominatim(user_agent='myGeocoder')
             location = locator.geocode(address)

             return location.latitude

         def get_long(address):
             locator = Nominatim(user_agent='myGeocoder')
             location = locator.geocode(address)

             return location.longitude

In [20]: postal_code_Melb_df['Lat'] = postal_code_Melb_df.apply(lambda x:get_lat(x.Suburb + " Victoria Australia ") if x.Lat ==0 else x.Lat,axis=1)
         postal_code_Melb_df['Long'] = postal_code_Melb_df.apply(lambda x:get_long(x.Suburb +  " Victoria Australia") if x.Long ==0 else x.Long,axis=1)

In [21]: # Checking for Rows with missing coordinates
         long_lat_0 = postal_code_Melb_df[postal_code_Melb_df['Long']==0]
         long_lat_0
Out[21]:
           Postcode  Region  Suburb  Long  Lat
```

Figure 7: Replacing missing geo coordinates values

Preparing Melbourne Location Data
In the Melbourne location data set it was observed that multiple suburbs had the same postal code and hence the same geo coordinates. Since, the aim of the next steps was to get nearby venues corresponding to a particular location, hence in order to avoid repetition of venues, the dataset was grouped on latitude and longitude values.



```
In [35]: postal_code_Melb_df.shape
Out[35]: (556, 5)

In [36]: # grouping localities based on longitudes and latitudes, to get single row for each pair of coordinates
         melb_post_code_grp= postal_code_Melb_df.groupby(['Long', 'Lat'])['Suburb'].apply(list).reset_index()
         melb_post_code_grp.shape
Out[36]: (233, 3)
```

Figure 8: Melbourne Location Dataset Before and After Grouping

The curated dataset of the Melbourne suburb locations was visualised using the ***Folium*** library as shown in **figure 9.**

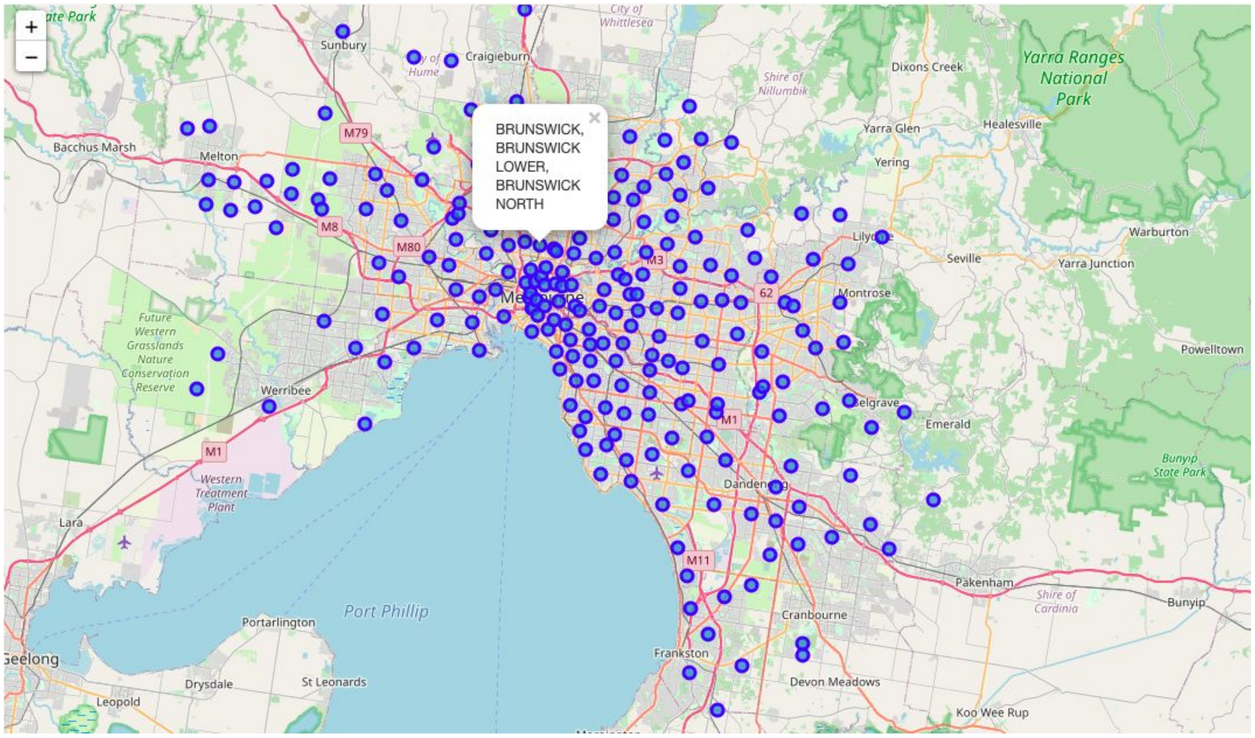


Figure 9: Suburbs' Locations of Melbourne

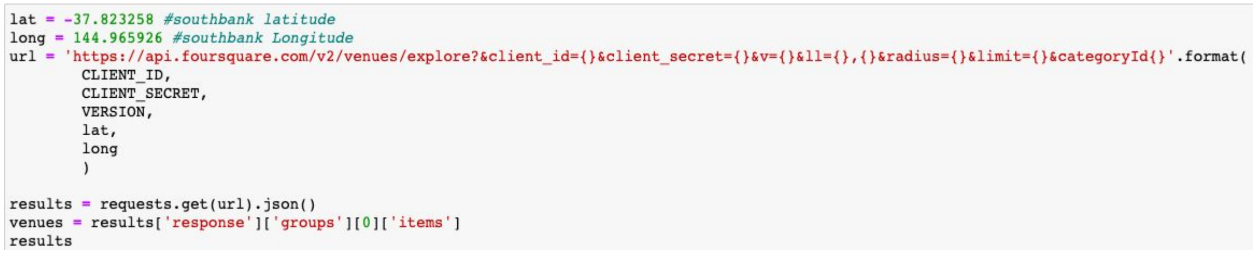
3.1.2 Gathering Data from Foursquare API

The Foursquare REST API was used to gather nearby venues for each geographical Location in the data set above. The key parameter required to call the API are:

{'CLIENT_ID', 'CLIENT_SECRET', 'VERSION', 'Latitude', 'Longitude'}

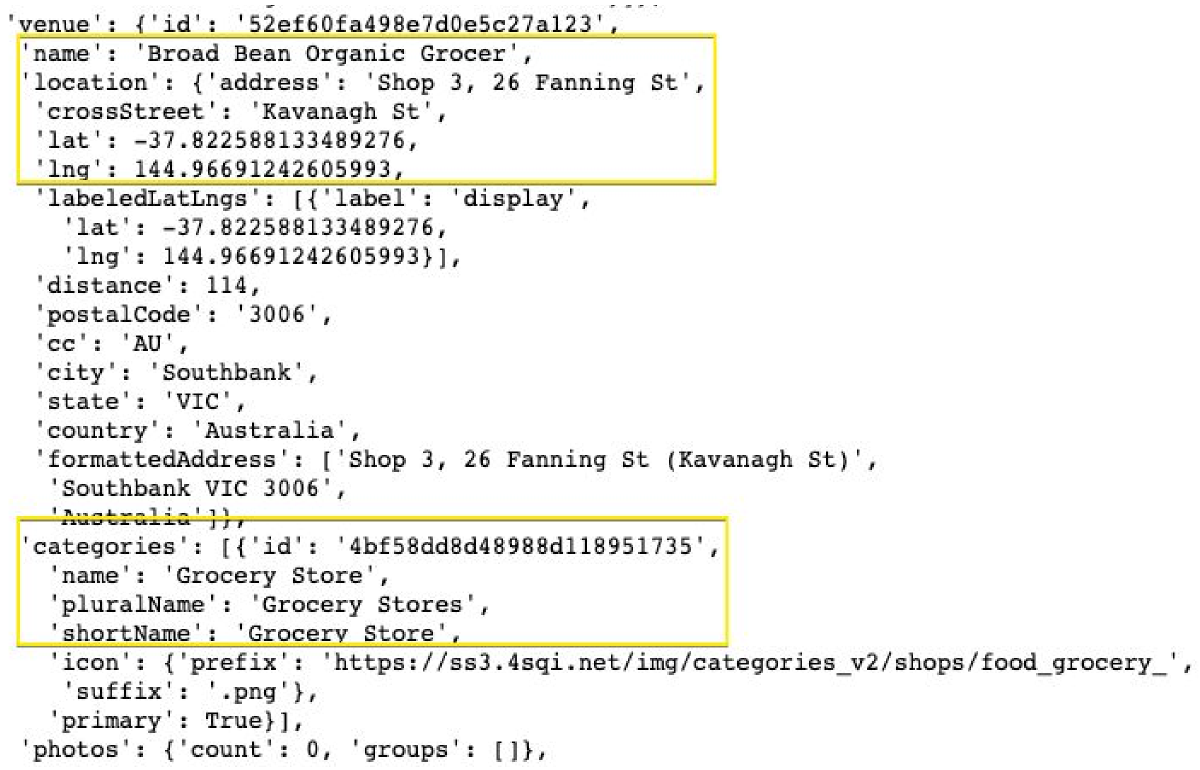
The API offers several endpoints to access the Foursquare databases, however in order to explore to nearby places the **'explore'** endpoint was used for this project.

A sample request to the Foursquare API is shown below:



```
lat = -37.823258 #southbank latitude
long = 144.965926 #southbank Longitude
url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}&categoryId{}'.format(
        CLIENT_ID,
        CLIENT_SECRET,
        VERSION,
        lat,
        long
        )

results = requests.get(url).json()
venues = results['response']['groups'][0]['items']
results
```

Figure 10: Sample GET Request: Foursquare API



```
'venue': {'id': '52ef60fa498e7d0e5c27a123',
 'name': 'Broad Bean Organic Grocer',
 'location': {'address': 'Shop 3, 26 Fanning St',
  'crossStreet': 'Kavanagh St',
  'lat': -37.822588133489276,
  'lng': 144.96691242605993,
  'labeledLatLngs': [{'label': 'display',
    'lat': -37.822588133489276,
    'lng': 144.96691242605993}],
  'distance': 114,
  'postalCode': '3006',
  'cc': 'AU',
  'city': 'Southbank',
  'state': 'VIC',
  'country': 'Australia',
  'formattedAddress': ['Shop 3, 26 Fanning St (Kavanagh St)',
   'Southbank VIC 3006',
   'Australia']},
 'categories': [{'id': '4bf58dd8d48988d118951735',
   'name': 'Grocery Store',
   'pluralName': 'Grocery Stores',
   'shortName': 'Grocery Store',
   'icon': {'prefix': 'https://ss3.4sqi.net/img/categories_v2/shops/food_grocery_',
    'suffix': '.png'},
   'primary': True}],
 'photos': {'count': 0, 'groups': []},
```

Figure 11: Sample Response: Foursquare API

The API returns the response in JSON format consisting of several fields. Among all the fields some of the key fields essential for this analysis as highlighted in **Figure 11** are described in the table below:

Field	Description	Example
“result”.response.groups[].items.venue.name	Name of the venue	Script Bar & Bistro
“result”.response.groups[].items.venue.location.lat	Latitude Value of the location	-37.821
“result”.response.groups[].items.venue.location.lng	Longitude Value of the location	144.968
“result”.response.groups[].items.venue.	Name of the venue	Bar

categories[].name	category	

Table 2: Required fields from Foursquare API response

As observed in the response in **Figure 11,** the first venue retrieved from the Foursquare API:

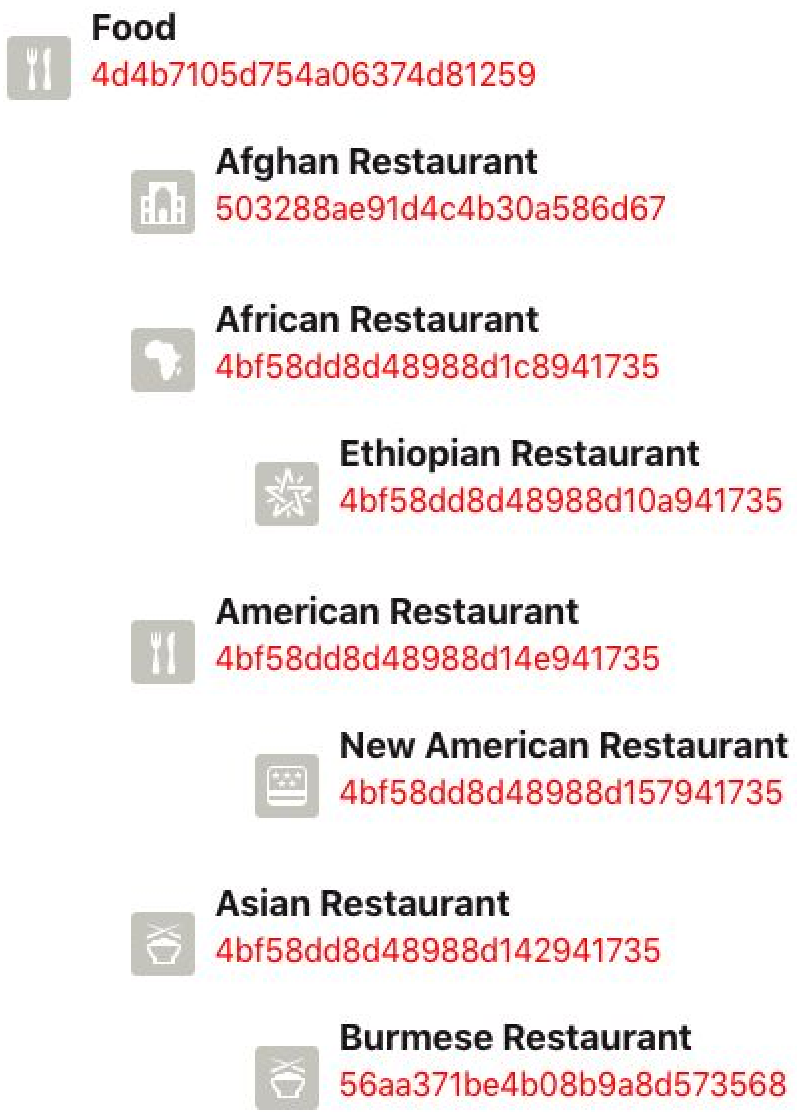
{name: 'Broad Bean Organic Grocer' , lat:-37.822588133489276, lng: 144.96691242605993, categories[].name: 'Grocery Store'}

Since for the scope of this project only food related venues are required, the request will be required to be refined in order to filter out other general categories. This can be achieved by passing the ***categoryID*** as one of the parameters in the GET request. The Foursquare API has a venue category hierarchy as shown in **Figure 12** .

It is inquisitive to know that the Foursquare API returns all sub-categories once the header category is passed into the GET request.

Additionally, it is possible to further refine the results by using a ***RADIUS*** parameter i.e. the radius to search within, in meters. This parameter will heavily influence the number of results returned. If more number of results are desired increase the values of the ***RADIUS*** parameter, in this project ***RADIUS = 1000 (in meters)*** was used to get a decent size of dataset.

Another parameter is the ***LIMIT=100*** i.e. the number of results to return was used in this project.

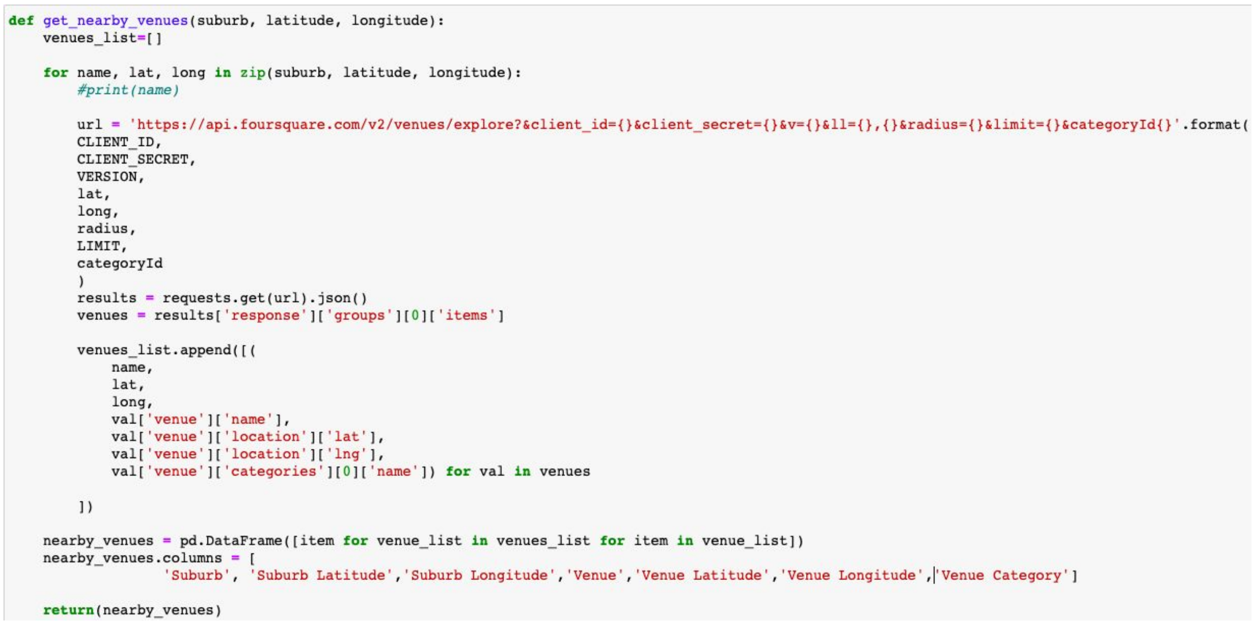


Food
4d4b7105d754a06374d81259

- Afghan Restaurant
  503288ae91d4c4b30a586d67
- African Restaurant
  4bf58dd8d48988d1c8941735
  - Ethiopian Restaurant
    4bf58dd8d48988d10a941735
- American Restaurant
  4bf58dd8d48988d14e941735
  - New American Restaurant
    4bf58dd8d48988d157941735
- Asian Restaurant
  4bf58dd8d48988d142941735
  - Burmese Restaurant
    56aa371be4b08b9a8d573568

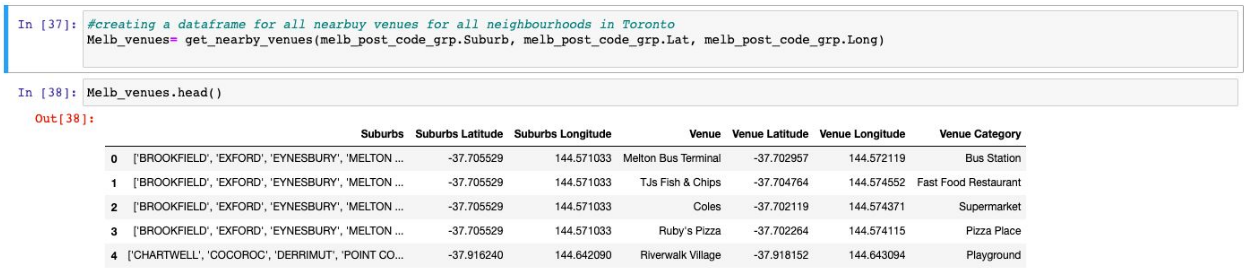
Figure 12: Foursquare API Venue Category Hierarchy

In order to speed up the process to get nearby food venues for each geographical location in the above dataset, a function ***"get_nearby_venues"*** was created with additional parameters as shown in **Figure 13** including the additional parameters ***LIMIT, RADIUS*** and ***CATEGORYID.*** The data retrieved from the GET request is appended to a Python list. The function finally returns a dataframe containing nearby venues for each geographical location as shown in **Figure 14.**



```python
def get_nearby_venues(suburb, latitude, longitude):
    venues_list=[]

    for name, lat, long in zip(suburb, latitude, longitude):
        #print(name)

        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}&categoryId{}'.format(
        CLIENT_ID,
        CLIENT_SECRET,
        VERSION,
        lat,
        long,
        radius,
        LIMIT,
        categoryId
        )
        results = requests.get(url).json()
        venues = results['response']['groups'][0]['items']

        venues_list.append([(
            name,
            lat,
            long,
            val['venue']['name'],
            val['venue']['location']['lat'],
            val['venue']['location']['lng'],
            val['venue']['categories'][0]['name']) for val in venues

        ])

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
    nearby_venues.columns = [
                  'Suburb', 'Suburb Latitude','Suburb Longitude','Venue','Venue Latitude','Venue Longitude','Venue Category']

    return(nearby_venues)
```

Figure 13: Function to get nearby venues



```
In [37]: #creating a dataframe for all nearbuy venues for all neighbourhoods in Toronto
         Melb_venues= get_nearby_venues(melb_post_code_grp.Suburb, melb_post_code_grp.Lat, melb_post_code_grp.Long)

In [38]: Melb_venues.head()
Out[38]:
```

	Suburbs	Suburbs Latitude	Suburbs Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	['BROOKFIELD', 'EXFORD', 'EYNESBURY', 'MELTON ...	-37.705529	144.571033	Melton Bus Terminal	-37.702957	144.572119	Bus Station
1	['BROOKFIELD', 'EXFORD', 'EYNESBURY', 'MELTON ...	-37.705529	144.571033	TJs Fish & Chips	-37.704764	144.574552	Fast Food Restaurant
2	['BROOKFIELD', 'EXFORD', 'EYNESBURY', 'MELTON ...	-37.705529	144.571033	Coles	-37.702119	144.574371	Supermarket
3	['BROOKFIELD', 'EXFORD', 'EYNESBURY', 'MELTON ...	-37.705529	144.571033	Ruby's Pizza	-37.702264	144.574115	Pizza Place
4	['CHARTWELL', 'COCOROC', 'DERRIMUT', 'POINT CO...	-37.916240	144.642090	Riverwalk Village	-37.918152	144.643094	Playground

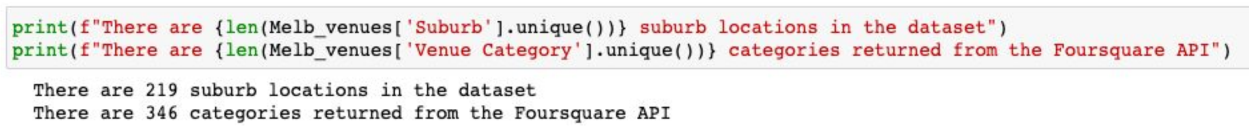
Figure 14: Dataframe with nearby venues for all the locations

Upto this point three dataframe are created:

- **"postal_code_Melb_df"**: Contains the ***post code, longitude, latitude, region, suburb names*** of all the suburbs within Melbourne.
- **"melb_post_code_grp"** : Contains ***longitude, latitude, suburb names*** grouped by ***longitude, latitude*** values.
- **"Melb_venues"**: Contains ***suburbs' names, longitude, latitude*** values and nearby ***venues' name, longitude, latitude*** values along with the name of the ***venue category.***

3.2 Exploratory Data Analysis

In the present dataset, it was observed that there are **346** unique venue categories and **219** uniques suburb locations as shown in **Figure 15**.



```
print(f"There are {len(Melb_venues['Suburb'].unique())} suburb locations in the dataset")
print(f"There are {len(Melb_venues['Venue Category'].unique())} categories returned from the Foursquare API")

  There are 219 suburb locations in the dataset
  There are 346 categories returned from the Foursquare API
```

Figure 15: Number of unique suburb locations and venue categories

3.2.1 Analysing Venue Categories

 First, the value counts of each category were analysed. In the results it some of the top food venues observed were:

{"Pizza Place": 158, "Italian Restaurant" : 133, "Fast Food Restaurant" : 125}

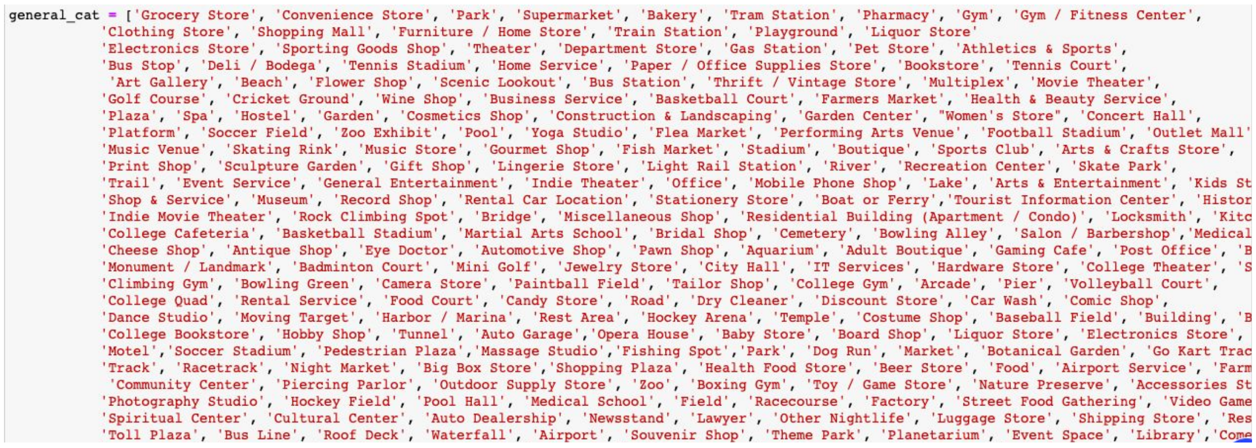
However, categories like ***"Park", "Convenience Store", "Coffee Shop", "Cafe",*** etc were also observed which provide very generic food culture, hence they were filtered out from the dataset.

Data Cleaning: Filtering out general categories

First, a list of general categories is prepared and fed into a Python list manually as described previously. In total there were 278 general categories.

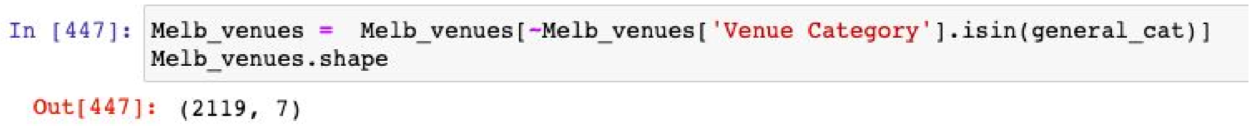
A simple **pandas.DataFrame.isin(***values***)** was applied over the dataframe where *values* is a list passed to the function**.** The **pandas.DataFrame.isin** function checks
whether each element in the DataFrame or a DataFrame column value is contained in values. Since, it was required to keep the rows which were not in the list , a **"~"** symbol is added to the function to achieve the desired results as shown in **Figure 17**.

After filtering out the general categories there were 121 categories that remained in the dataset.



```
general_cat = ['Grocery Store', 'Convenience Store', 'Park', 'Supermarket', 'Bakery', 'Tram Station', 'Pharmacy', 'Gym', 'Gym / Fitness Center',
            'Clothing Store', 'Shopping Mall', 'Furniture / Home Store', 'Train Station', 'Playground', 'Liquor Store'
            'Electronics Store', 'Sporting Goods Shop', 'Theater', 'Department Store', 'Gas Station', 'Pet Store', 'Athletics & Sports',
            'Bus Stop', 'Deli / Bodega', 'Tennis Stadium', 'Home Service', 'Paper / Office Supplies Store', 'Bookstore', 'Tennis Court',
             'Art Gallery', 'Beach', 'Flower Shop', 'Scenic Lookout', 'Bus Station', 'Thrift / Vintage Store', 'Multiplex', 'Movie Theater',
            'Golf Course', 'Cricket Ground', 'Wine Shop', 'Business Service', 'Basketball Court', 'Farmers Market', 'Health & Beauty Service',
            'Plaza', 'Spa', 'Hostel', 'Garden', 'Cosmetics Shop', 'Construction & Landscaping', 'Garden Center', "Women's Store", 'Concert Hall',
            'Platform', 'Soccer Field', 'Zoo Exhibit', 'Pool', 'Yoga Studio', 'Flea Market', 'Performing Arts Venue', 'Football Stadium', 'Outlet Mall'
            'Music Venue', 'Skating Rink', 'Music Store', 'Gourmet Shop', 'Fish Market', 'Stadium', 'Boutique', 'Sports Club', 'Arts & Crafts Store',
            'Print Shop', 'Sculpture Garden', 'Gift Shop', 'Lingerie Store', 'Light Rail Station', 'River', 'Recreation Center', 'Skate Park',
            'Trail', 'Event Service', 'General Entertainment', 'Indie Theater', 'Office', 'Mobile Phone Shop', 'Lake', 'Arts & Entertainment', 'Kids St
            'Shop & Service', 'Museum', 'Record Shop', 'Rental Car Location', 'Stationery Store', 'Boat or Ferry','Tourist Information Center', 'Histor
            'Indie Movie Theater', 'Rock Climbing Spot', 'Bridge', 'Miscellaneous Shop', 'Residential Building (Apartment / Condo)', 'Locksmith', 'Kitc
            'College Cafeteria', 'Basketball Stadium', 'Martial Arts School', 'Bridal Shop', 'Cemetery', 'Bowling Alley', 'Salon / Barbershop','Medical
            'Cheese Shop', 'Antique Shop', 'Eye Doctor', 'Automotive Shop', 'Pawn Shop', 'Aquarium', 'Adult Boutique', 'Gaming Cafe', 'Post Office', 'B
            'Monument / Landmark', 'Badminton Court', 'Mini Golf', 'Jewelry Store', 'City Hall', 'IT Services', 'Hardware Store', 'College Theater', 'S
            'Climbing Gym', 'Bowling Green', 'Camera Store', 'Paintball Field', 'Tailor Shop', 'College Gym', 'Arcade', 'Pier', 'Volleyball Court',
            'College Quad', 'Rental Service', 'Food Court', 'Candy Store', 'Road', 'Dry Cleaner', 'Discount Store', 'Car Wash', 'Comic Shop',
            'Dance Studio', 'Moving Target', 'Harbor / Marina', 'Rest Area', 'Hockey Arena', 'Temple', 'Costume Shop', 'Baseball Field', 'Building', 'B
            'College Bookstore', 'Hobby Shop', 'Tunnel', 'Auto Garage','Opera House', 'Baby Store', 'Board Shop', 'Liquor Store', 'Electronics Store',
            'Motel','Soccer Stadium', 'Pedestrian Plaza','Massage Studio','Fishing Spot','Park', 'Dog Run', 'Market', 'Botanical Garden', 'Go Kart Trac
            'Track', 'Racetrack', 'Night Market', 'Big Box Store','Shopping Plaza', 'Health Food Store', 'Beer Store', 'Food', 'Airport Service', 'Farm
             'Community Center', 'Piercing Parlor', 'Outdoor Supply Store', 'Zoo', 'Boxing Gym', 'Toy / Game Store', 'Nature Preserve', 'Accessories St
            'Photography Studio', 'Hockey Field', 'Pool Hall', 'Medical School', 'Field', 'Racecourse', 'Factory', 'Street Food Gathering', 'Video Game
            'Spiritual Center', 'Cultural Center', 'Auto Dealership', 'Newsstand', 'Lawyer', 'Other Nightlife', 'Luggage Store', 'Shipping Store', 'Res
            'Toll Plaza', 'Bus Line', 'Roof Deck', 'Waterfall', 'Airport', 'Souvenir Shop', 'Theme Park', 'Planetarium', 'Event Space', 'Library','Come
```

Figure 16: List of General Categories



```
In [447]: Melb_venues =  Melb_venues[~Melb_venues['Venue Category'].isin(general_cat)]
          Melb_venues.shape

Out[447]: (2119, 7)
```

Figure 17: Filtering out rows with General Categories

A total of **3851** rows were filtered out after this operation.

3.2.2 Data Visualisation

Visualising top 10 Venue Categories

The value count of the *"Venue Category"* column was taken and the first top 10 rows were selected. The resulting data frame was plotted with the 'Seaborn' library.

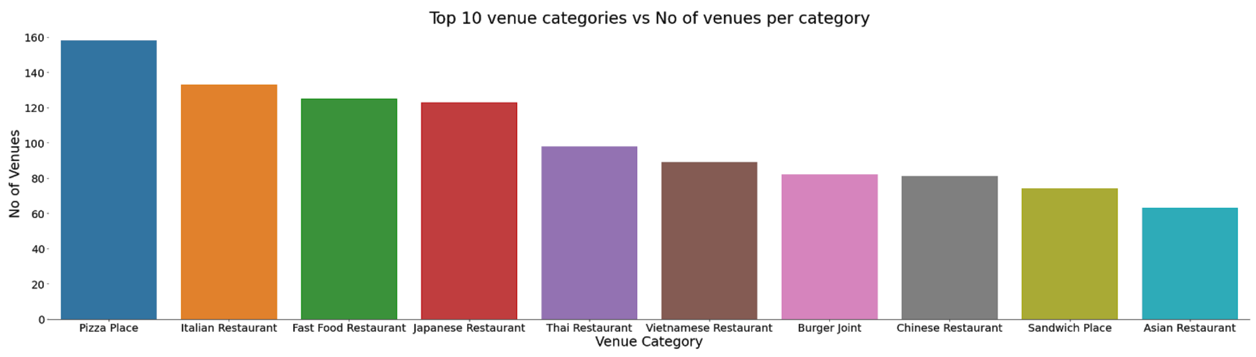
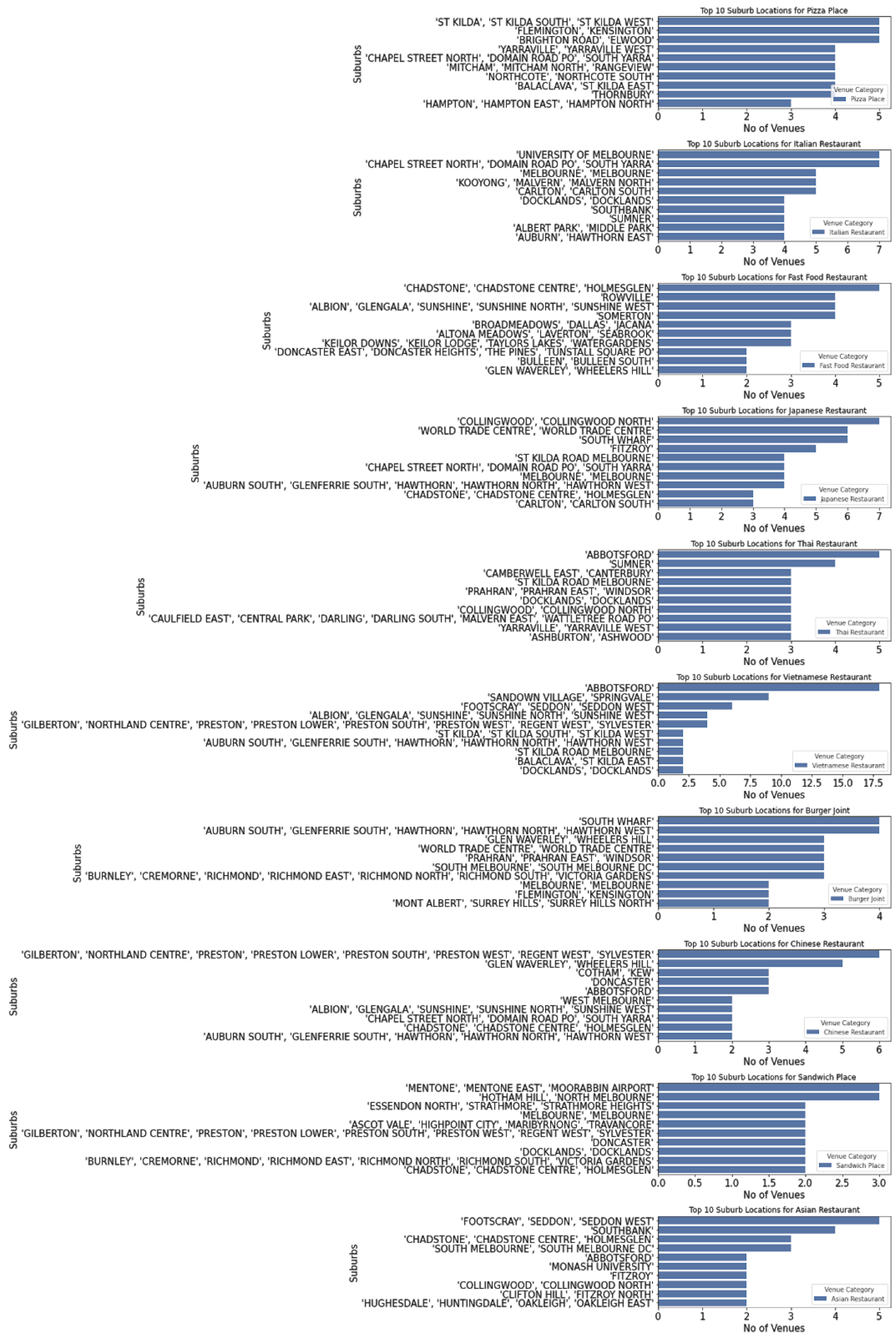


Figure 18: Top 10 Venue Categories

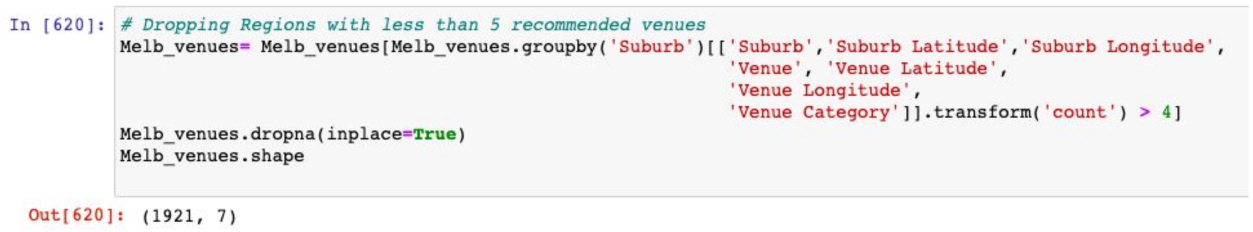
Visualising top 10 Suburbs for each of the top 10 Venue Categories

For this visualisation the suburbs having the most count of the top 10 categories as observed above were taken and the data was plotted using a grid plot as shown in **Figure 19** below.



3.2.3 Analysing Suburbs

In this step, first the value count of the 'Suburb' column was taken. Then the data was grouped by the ***'Suburb'*** column and all the suburbs/groups having less than 4 rows were removed from the dataset as one of the aims of the project is to recommend at least 5 popular venues from each of the neighbourhoods.



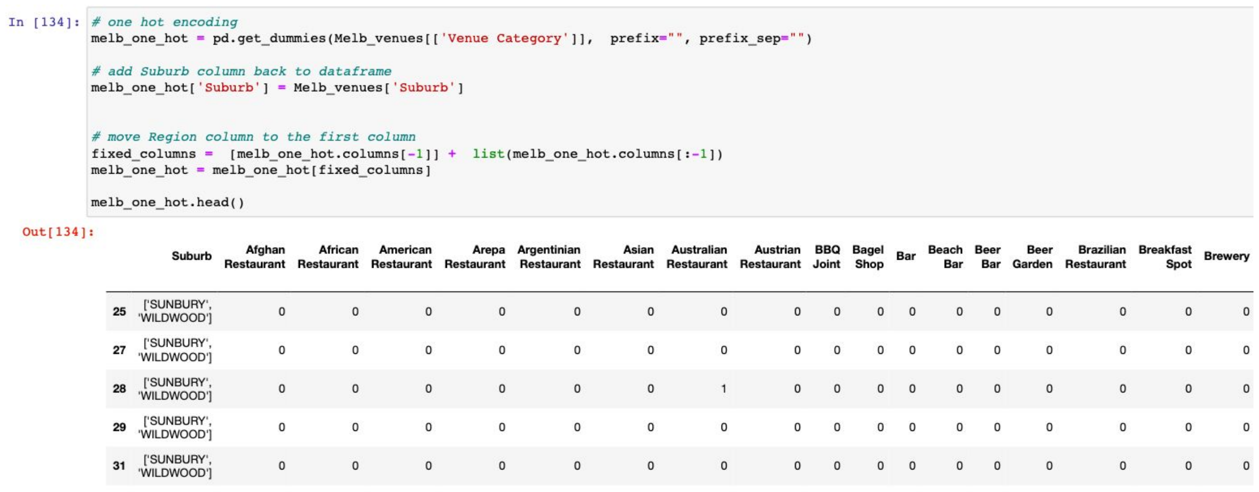
```
In [620]: # Dropping Regions with less than 5 recommended venues
          Melb_venues= Melb_venues[Melb_venues.groupby('Suburb')[['Suburb','Suburb Latitude','Suburb Longitude',
                                                                  'Venue', 'Venue Latitude',
                                                                  'Venue Longitude',
                                                                  'Venue Category']].transform('count') > 4]
          Melb_venues.dropna(inplace=True)
          Melb_venues.shape

Out[620]: (1921, 7)
```

Figure 20: Filtering out Suburbs with less than 5 venues

As seen in **Figure 20, 1921** samples remain after filtering out the Suburbs with less than 5 venues.

3.3 Feature Engineering

For the data to be used with 'K-means' algorithm the categorical data i.e. the venue categories have to be converted to numerical values as the algorithm can't process non-numerical values. The 'Venue Category' column in the dataset is hot encoded using *pandas.dummies* and the 'Suburb' column was added back. The first 5 rows of the resulting data frame are shown in **Figure 21**. The data frame has 1921 samples and 121 venue categories.



```
In [134]: # one hot encoding
          melb_one_hot = pd.get_dummies(Melb_venues[['Venue Category']],  prefix="", prefix_sep="")

          # add Suburb column back to dataframe
          melb_one_hot['Suburb'] = Melb_venues['Suburb']


          # move Region column to the first column
          fixed_columns =  [melb_one_hot.columns[-1]] +  list(melb_one_hot.columns[:-1])
          melb_one_hot = melb_one_hot[fixed_columns]

          melb_one_hot.head()
Out[134]:
```

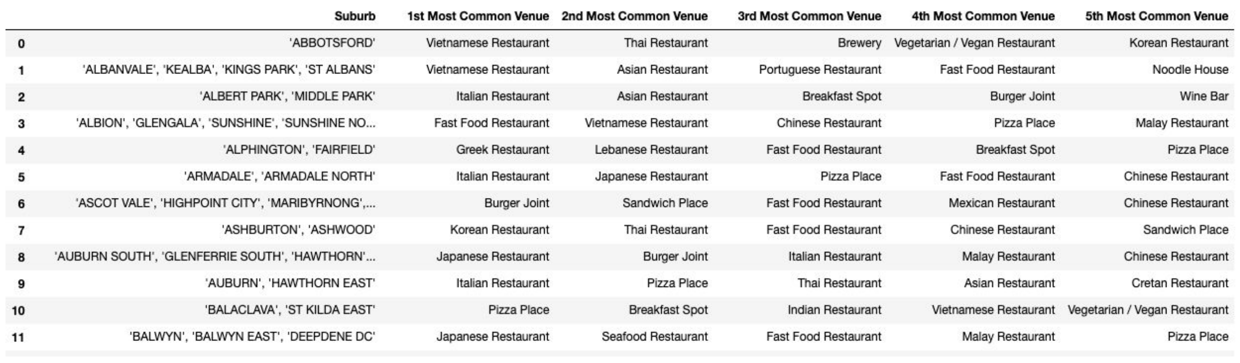
	Suburb	Afghan Restaurant	African Restaurant	American Restaurant	Arepa Restaurant	Argentinian Restaurant	Asian Restaurant	Australian Restaurant	Austrian Restaurant	BBQ Joint	Bagel Shop	Bar	Beach Bar	Beer Bar	Beer Garden	Brazilian Restaurant	Breakfast Spot	Brewery
25	['SUNBURY', 'WILDWOOD']	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	['SUNBURY', 'WILDWOOD']	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	['SUNBURY', 'WILDWOOD']	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
29	['SUNBURY', 'WILDWOOD']	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	['SUNBURY', 'WILDWOOD']	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21: Hot-encoded 'Venue Category'

Following the hot-encoding, the data frame is grouped on the 'Suburb' column and the sum of the occurrence of each category for a suburb was captured.

The dataset upto this point had ***99*** samples and ***120*** features.

Based on this data a profile for each suburb in a separate data frame consisting of the top 5 frequent occurring venues was created as shown in **Figure 23**.



	Suburb	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	'ABBOTSFORD'	Vietnamese Restaurant	Thai Restaurant	Brewery	Vegetarian / Vegan Restaurant	Korean Restaurant
1	'ALBANVALE', 'KEALBA', 'KINGS PARK', 'ST ALBANS'	Vietnamese Restaurant	Asian Restaurant	Portuguese Restaurant	Fast Food Restaurant	Noodle House
2	'ALBERT PARK', 'MIDDLE PARK'	Italian Restaurant	Asian Restaurant	Breakfast Spot	Burger Joint	Wine Bar
3	'ALBION', 'GLENGALA', 'SUNSHINE', 'SUNSHINE NO...	Fast Food Restaurant	Vietnamese Restaurant	Chinese Restaurant	Pizza Place	Malay Restaurant
4	'ALPHINGTON', 'FAIRFIELD'	Greek Restaurant	Lebanese Restaurant	Fast Food Restaurant	Breakfast Spot	Pizza Place
5	'ARMADALE', 'ARMADALE NORTH'	Italian Restaurant	Japanese Restaurant	Pizza Place	Fast Food Restaurant	Chinese Restaurant
6	'ASCOT VALE', 'HIGHPOINT CITY', 'MARIBYRNONG',...	Burger Joint	Sandwich Place	Fast Food Restaurant	Mexican Restaurant	Chinese Restaurant
7	'ASHBURTON', 'ASHWOOD'	Korean Restaurant	Thai Restaurant	Fast Food Restaurant	Chinese Restaurant	Sandwich Place
8	'AUBURN SOUTH', 'GLENFERRIE SOUTH', 'HAWTHORN'...	Japanese Restaurant	Burger Joint	Italian Restaurant	Malay Restaurant	Chinese Restaurant
9	'AUBURN', 'HAWTHORN EAST'	Italian Restaurant	Pizza Place	Thai Restaurant	Asian Restaurant	Cretan Restaurant
10	'BALACLAVA', 'ST KILDA EAST'	Pizza Place	Breakfast Spot	Indian Restaurant	Vietnamese Restaurant	Vegetarian / Vegan Restaurant
11	'BALWYN', 'BALWYN EAST', 'DEEPDENE DC'	Japanese Restaurant	Seafood Restaurant	Fast Food Restaurant	Malay Restaurant	Pizza Place

Figure 22: Profile of each neighbourhood with top 5 venue categories

3.4 Feature Scaling

As an example, the distribution of a few features in the grouped dataset was plotted as shown in **Figure 22.** As it can be observed that, the range of these columns vary for each column.

Therefore, there was a need to scale the values to a common scale for all the columns with higher range would have been given more weight during the clustering operation and would have produced inconsistent clusters.

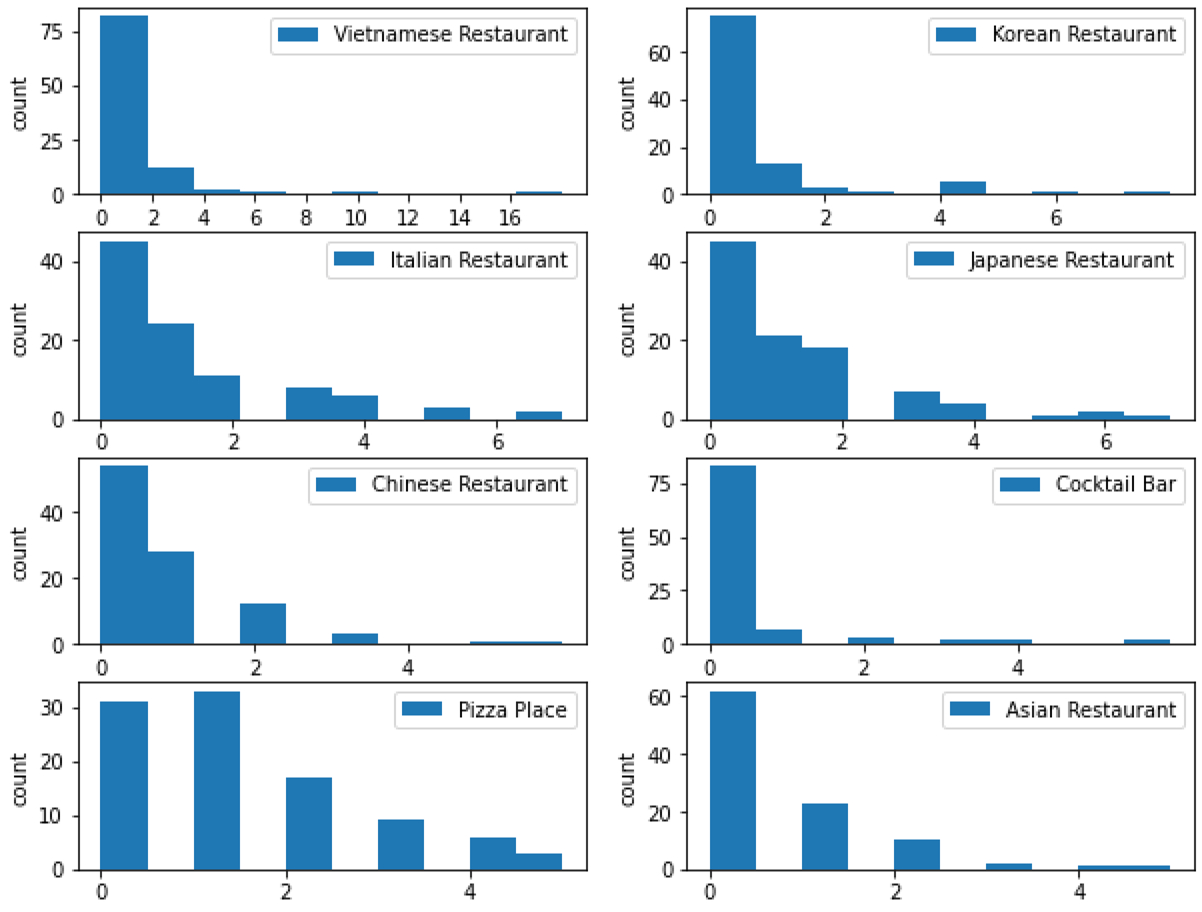
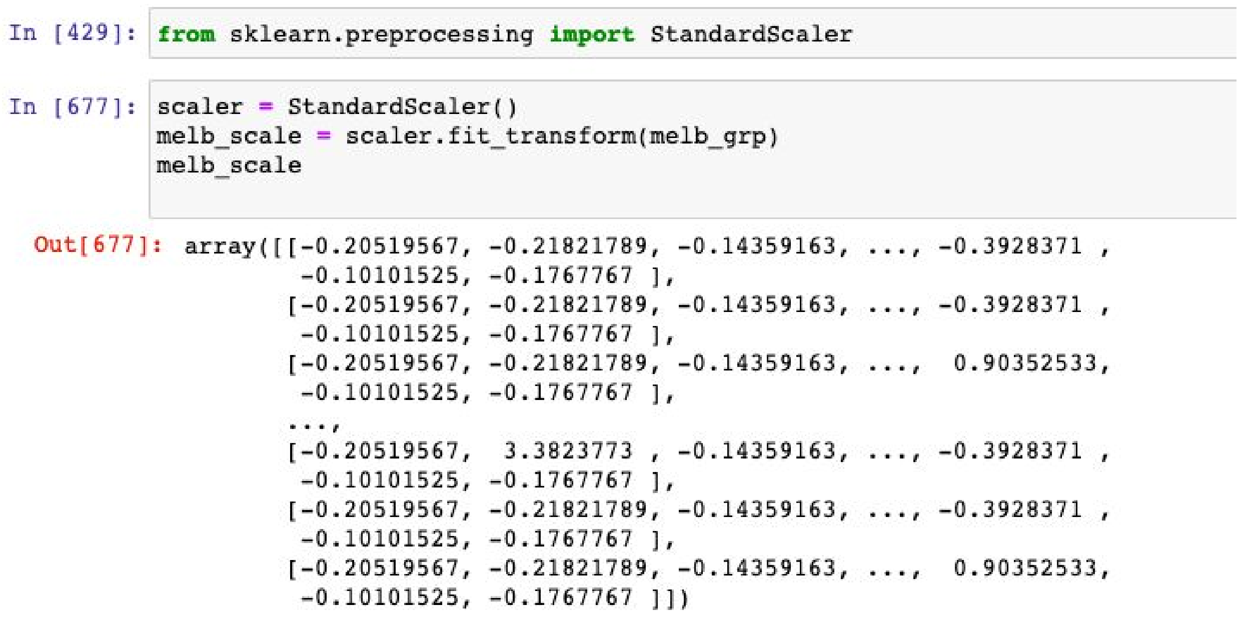


Figure 23: Sample distribution of the columns

For the scaling purpose, the standard scaler from the scikit learn library was used. The standard scaler results in the features having a mean of 0 and a unit standard deviation across all the samples. As a result the clustering algorithm would be able to give equal weights to all the features. **Figure 24** showcases the process of scaling of the dataset



```
In [429]: from sklearn.preprocessing import StandardScaler

In [677]: scaler = StandardScaler()
          melb_scale = scaler.fit_transform(melb_grp)
          melb_scale

Out[677]: array([[-0.20519567, -0.21821789, -0.14359163, ..., -0.3928371 ,
                  -0.10101525, -0.1767767 ],
                 [-0.20519567, -0.21821789, -0.14359163, ..., -0.3928371 ,
                  -0.10101525, -0.1767767 ],
                 [-0.20519567, -0.21821789, -0.14359163, ...,  0.90352533,
                  -0.10101525, -0.1767767 ],
                 ...,
                 [-0.20519567,  3.3823773 , -0.14359163, ..., -0.3928371 ,
                  -0.10101525, -0.1767767 ],
                 [-0.20519567, -0.21821789, -0.14359163, ..., -0.3928371 ,
                  -0.10101525, -0.1767767 ],
                 [-0.20519567, -0.21821789, -0.14359163, ...,  0.90352533,
                  -0.10101525, -0.1767767 ]])
```

Figure 24: Feature Scaling of the dataset

3.5 Cluster Analysis

As stated earlier, the goal of this project is to segment or cluster suburbs together which have similar food culture. For this analysis the **'K-Means'** algorithm was used.

'K-Means' algorithm is an unsupervised machine algorithm which aims to segment 'n' observations into 'K' clusters by minimizing the distance between the centroid of the cluster and the data point such that each data point belongs to a cluster.

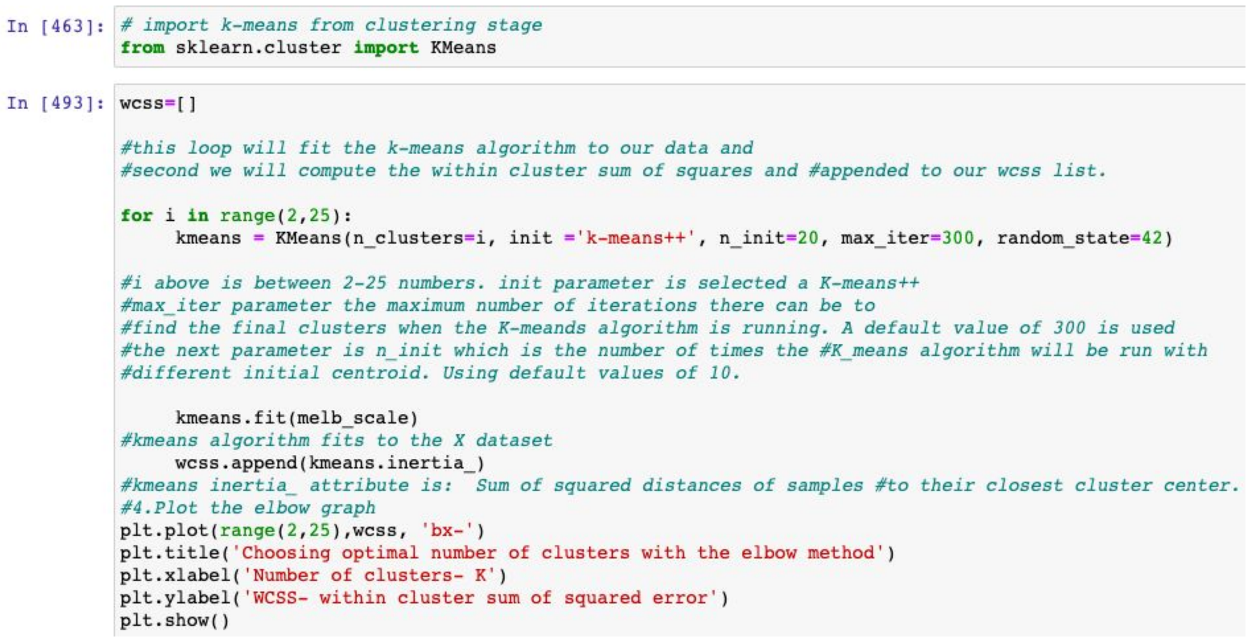
In other words, the **K-means** algorithm finds **K** centroids and allocates each data point to the nearest centroid by minimising the distance between the point and the centroid.

However, the foremost step before applying the 'K-Means' algorithm is to figure out the optimal value of 'K' as too few clusters may not segment the clusters properly and too many clusters might make it difficult to get any inference from them.

There were two methods used to find the optimal value of **'K',** which are as follows:

1. **Elbow Method:**
   The basic intuition behind using the elbow method is to find the optimal value of **'K'** from a range of values of **'K',** such that the intra-cluster variation or total within clusters

sum of squares is minimised. The optimal number of clusters is the value after which there is no significant decrease in the sum of squared distances. For this project, the K-means algorithm was run for a range for values of **'K'** between 2 and 25, for each value and the values of sum of squared distances was plotted against the number of clusters as shown below:



```
In [463]: # import k-means from clustering stage
          from sklearn.cluster import KMeans

In [493]: wcss=[]

          #this loop will fit the k-means algorithm to our data and
          #second we will compute the within cluster sum of squares and #appended to our wcss list.

          for i in range(2,25):
               kmeans = KMeans(n_clusters=i, init ='k-means++', n_init=20, max_iter=300, random_state=42)

          #i above is between 2-25 numbers. init parameter is selected a K-means++
          #max_iter parameter the maximum number of iterations there can be to
          #find the final clusters when the K-meands algorithm is running. A default value of 300 is used
          #the next parameter is n_init which is the number of times the #K_means algorithm will be run with
          #different initial centroid. Using default values of 10.

               kmeans.fit(melb_scale)
          #kmeans algorithm fits to the X dataset
               wcss.append(kmeans.inertia_)
          #kmeans inertia_ attribute is:  Sum of squared distances of samples #to their closest cluster center.
          #4.Plot the elbow graph
          plt.plot(range(2,25),wcss, 'bx-')
          plt.title('Choosing optimal number of clusters with the elbow method')
          plt.xlabel('Number of clusters- K')
          plt.ylabel('WCSS- within cluster sum of squared error')
          plt.show()
```

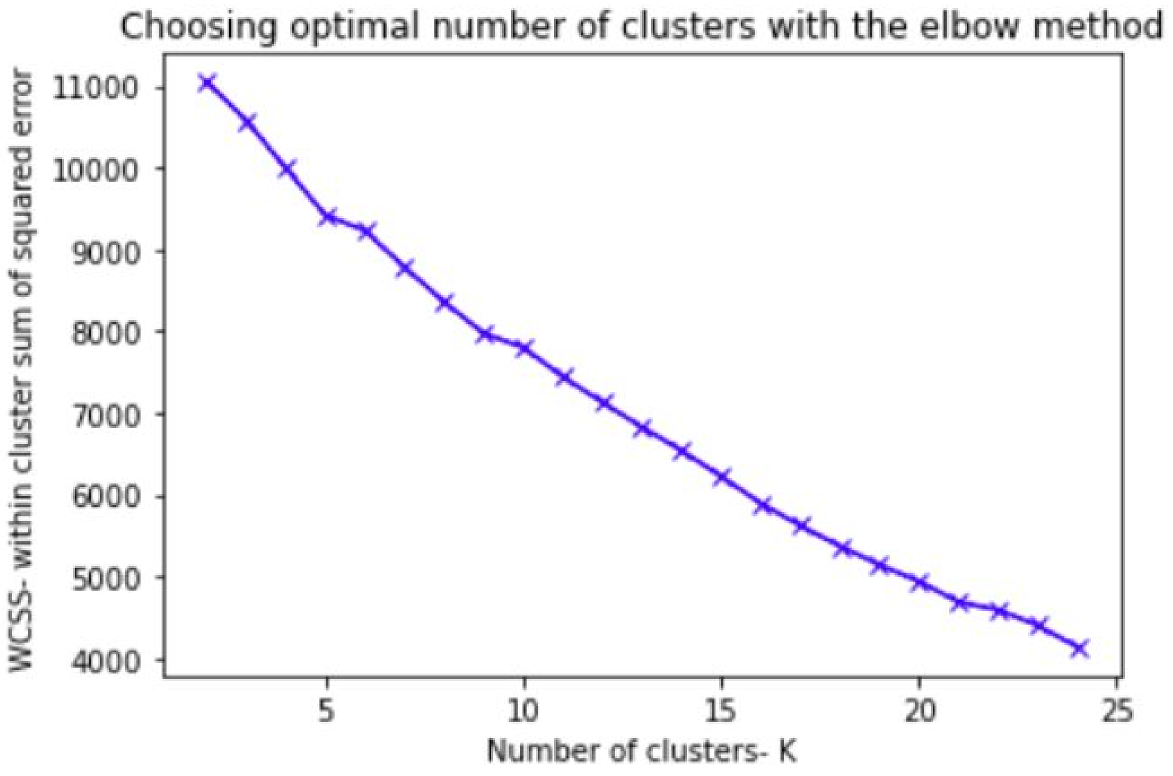


Figure 25: Elbow Method for choosing optimal clusters

Although there is a gradual decrease in the sum of squared distances, however as seen in **Figure 25**, it is difficult to pick an optimal values of K as there are multiple elbow points observed at *K= 5, 9.* This is normal to happen with the elbow method, Hence another method i.e. the *average silhouette score* method was used.

2. **Average Silhouette Score:**
   The silhouette score measures the cohesion i.e. how similar a point is to its own clusters in comparison to the other clusters. The range of silhouette score ranges between -1 and 1, a silhouette score closer to 1 indicates that the point is well clustered within a cluster and well separated from other clusters.

   The formula for calculating silhouette coefficient is given by:

$$s(\boldsymbol{o}) = \frac{b(\boldsymbol{o}) - a(\boldsymbol{o})}{\max\{a(\boldsymbol{o}), b(\boldsymbol{o})\}}$$

where,

s(o): is the silhouette coefficient of the point *o.*

b(o): *average inter-cluster distance between o and rest of the points within the cluster in which o does not belong to.*

a(o): *average intra-cluster distance between o and rest of the points within the cluster in which o belongs to.*

For this project, the K-means algorithm was run for a range for values of **‘K’** between 2 and 25, for each value and the average of silhouette score over all samples was plotted against the number of clusters as shown below

```
from sklearn.metrics import silhouette_score

sil = []
K_sil = range(2,25)
# minimum 2 clusters required, to define dissimilarity
for k in K_sil:
    print(k, end=' ')
    kmeans = KMeans(init="k-means++", n_clusters=k,n_init=50, max_iter=300, random_state=42).fit(melb_scale)
    labels = kmeans.labels_
    sil.append(silhouette_score(melb_scale, labels, metric = 'euclidean'))

plt.plot(K_sil, sil, 'bx-')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Average Silhouette Score')
plt.title('Average Silhouette score vs  Number of clusters (K)')
plt.show()
```

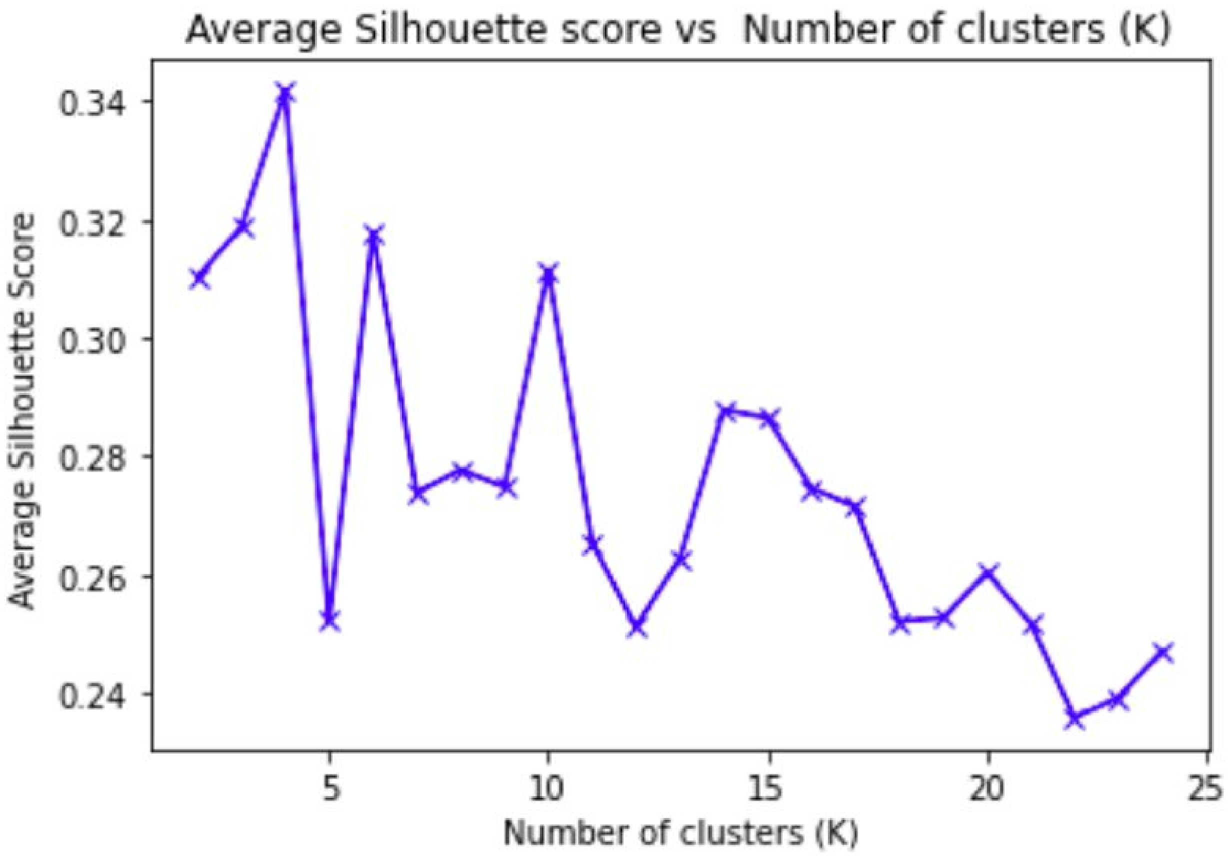
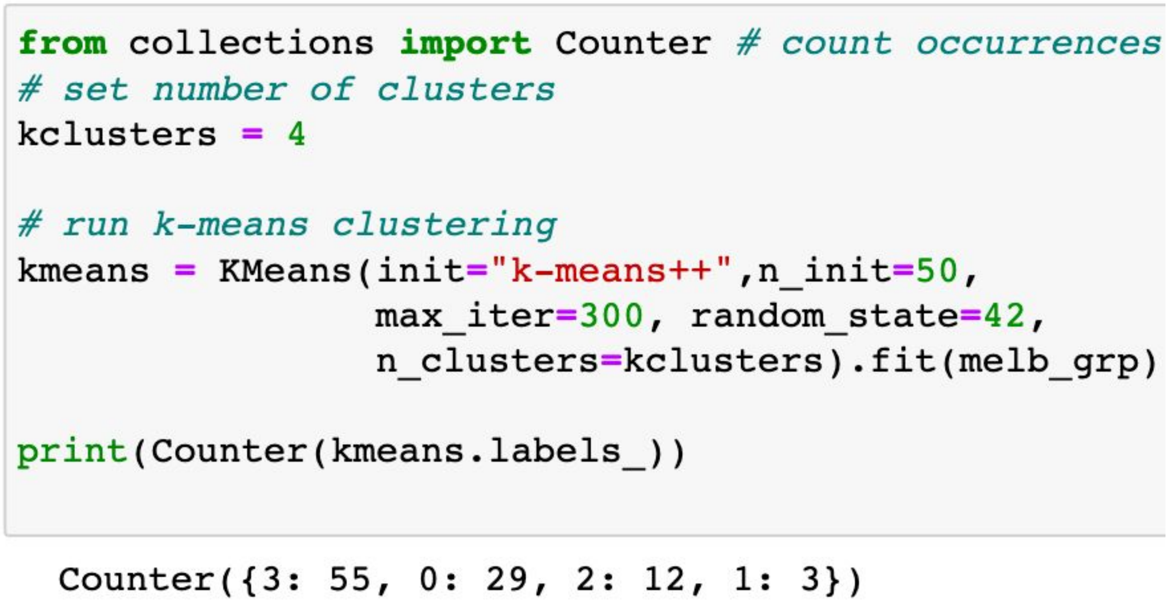


Figure 26: Average Silhouette Score vs Number of Clusters

As per **Figure 26**, the maximum score was observed at *K = 4,* hence this value was chosen for the optimal value for the number of clusters.

3.5.1 K-Means Clustering

The *sklearn* library implementation of the K-means algorithm was used with num of clusters=4. The following code block implements the algorithm and returns the number of samples in each cluster and the cluster labels.



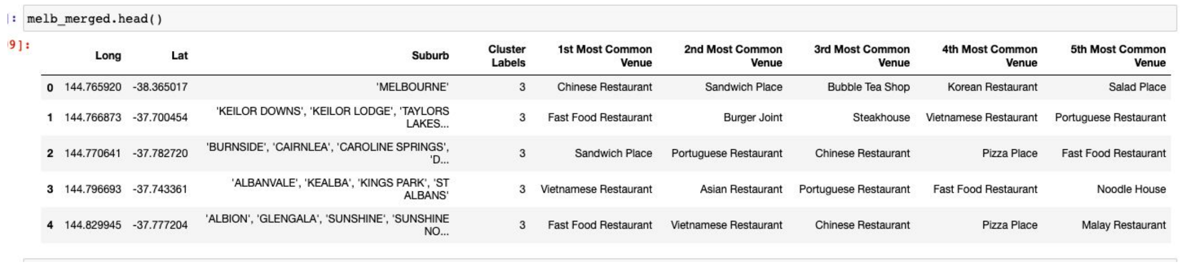
```
from collections import Counter # count occurrences
# set number of clusters
kclusters = 4

# run k-means clustering
kmeans = KMeans(init="k-means++",n_init=50,
                max_iter=300, random_state=42,
                n_clusters=kclusters).fit(melb_grp)

print(Counter(kmeans.labels_))
```

```
Counter({3: 55, 0: 29, 2: 12, 1: 3})
```

Figure 26: K-means Implementation with K=4

4. Results

The cluster labels were added to the dataframe and this dataset is merged with dataframe containing the geo location data. The resulting data frame with cluster labels and geo location is shown below:



```
]: melb_merged.head()
```

9]:

	Long	Lat	Suburb	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	144.765920	-38.365017	'MELBOURNE'	3	Chinese Restaurant	Sandwich Place	Bubble Tea Shop	Korean Restaurant	Salad Place
1	144.766873	-37.700454	'KEILOR DOWNS', 'KEILOR LODGE', 'TAYLORS LAKES...	3	Fast Food Restaurant	Burger Joint	Steakhouse	Vietnamese Restaurant	Portuguese Restaurant
2	144.770641	-37.782720	'BURNSIDE', 'CAIRNLEA', 'CAROLINE SPRINGS', 'D...	3	Sandwich Place	Portuguese Restaurant	Chinese Restaurant	Pizza Place	Fast Food Restaurant
3	144.796693	-37.743361	'ALBANVALE', 'KEALBA', 'KINGS PARK', 'ST ALBANS'	3	Vietnamese Restaurant	Asian Restaurant	Portuguese Restaurant	Fast Food Restaurant	Noodle House
4	144.829945	-37.777204	'ALBION', 'GLENGALA', 'SUNSHINE', 'SUNSHINE NO...	3	Fast Food Restaurant	Vietnamese Restaurant	Chinese Restaurant	Pizza Place	Malay Restaurant

Figure 27: Merged Data frame with geolocations, cluster labels & top venues

Further, the clusters are visualized on a map using the Folium library as shown in **Figure 28**. This map represents the segmenting of Melbourne suburbs based on the food culture in the respective suburbs.

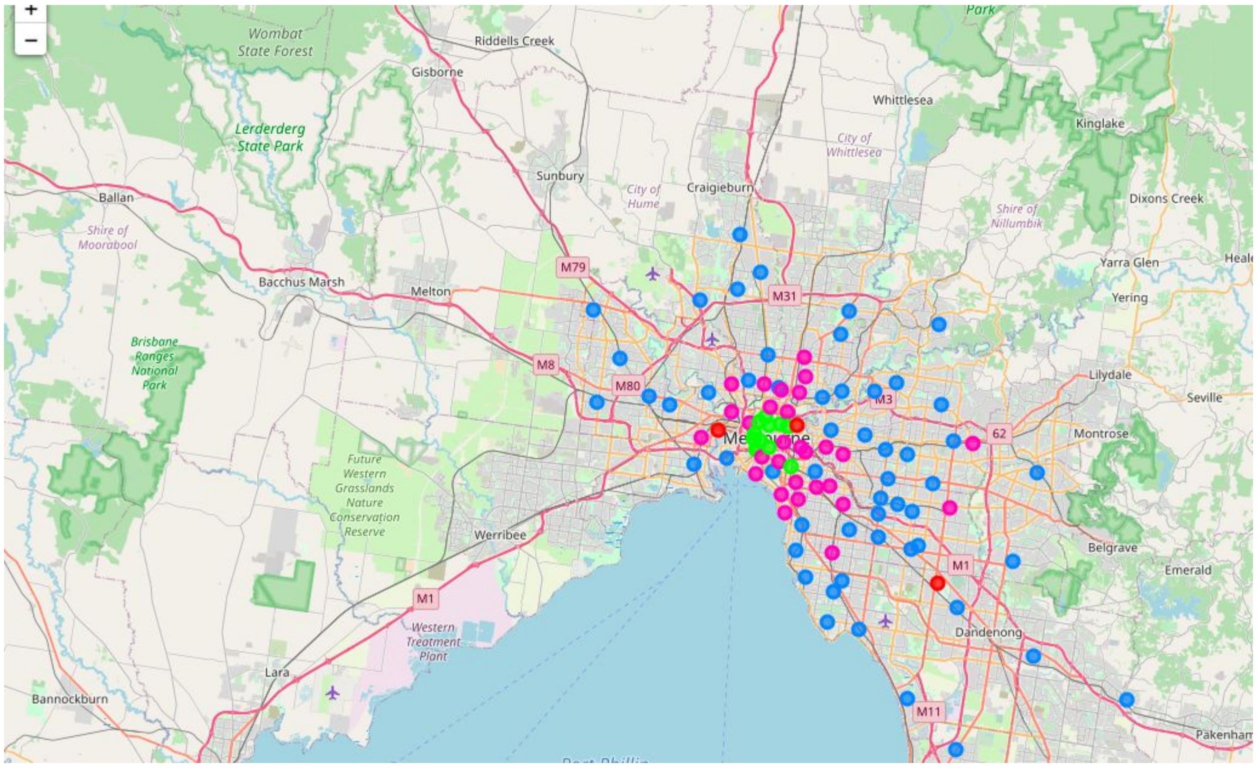


Figure 28: Visualisation of clusters based on food culture in Melbourne

In order to understand the theme of the food culture within each cluster, first a wordcloud of all the suburbs within the clusters was created. Thereafter, from each cluster top 5 venue categories were picked.

The overall cluster analysis is broken down by clusters as described below.

Cluster 1:

Cluster 1 mostly consists of suburbs of Melbourne which are in close proximity to the central business district of Melbourne city along with other suburbs which form the inner suburbs of Melbourne and are equally commercially developed.

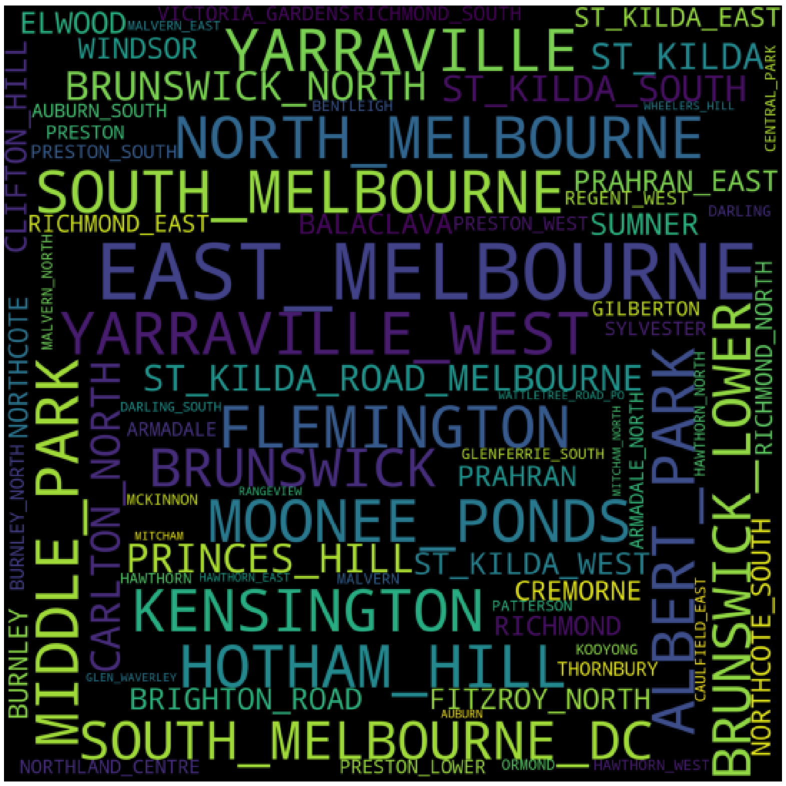


Figure 29: Suburbs in Cluster 1

One thing to note here is that according to the Australian census 2016, the majority of suburbs in this cluster are multi-ethnic, however they are majorly composed of Australian and other western European ethnicities.

Among the top 5 venue categories , ***"Pizza Place", "Italian Restaurants"*** constitute a majority of venues in this cluster, this can be attributed to the high influence of Italian cuisine on Melbourne's food culture for centuries. The categories are followed by ***"Japanese"*** ,***"Thai"*** which also are a key part of the Melbourne food culture along with ***"Burger Joints"*** which are famous for Aussie beef burgers.

Therefore, looking at the venue categories and the cultural history this cluster can be termed as ***"Melbournian"*** .

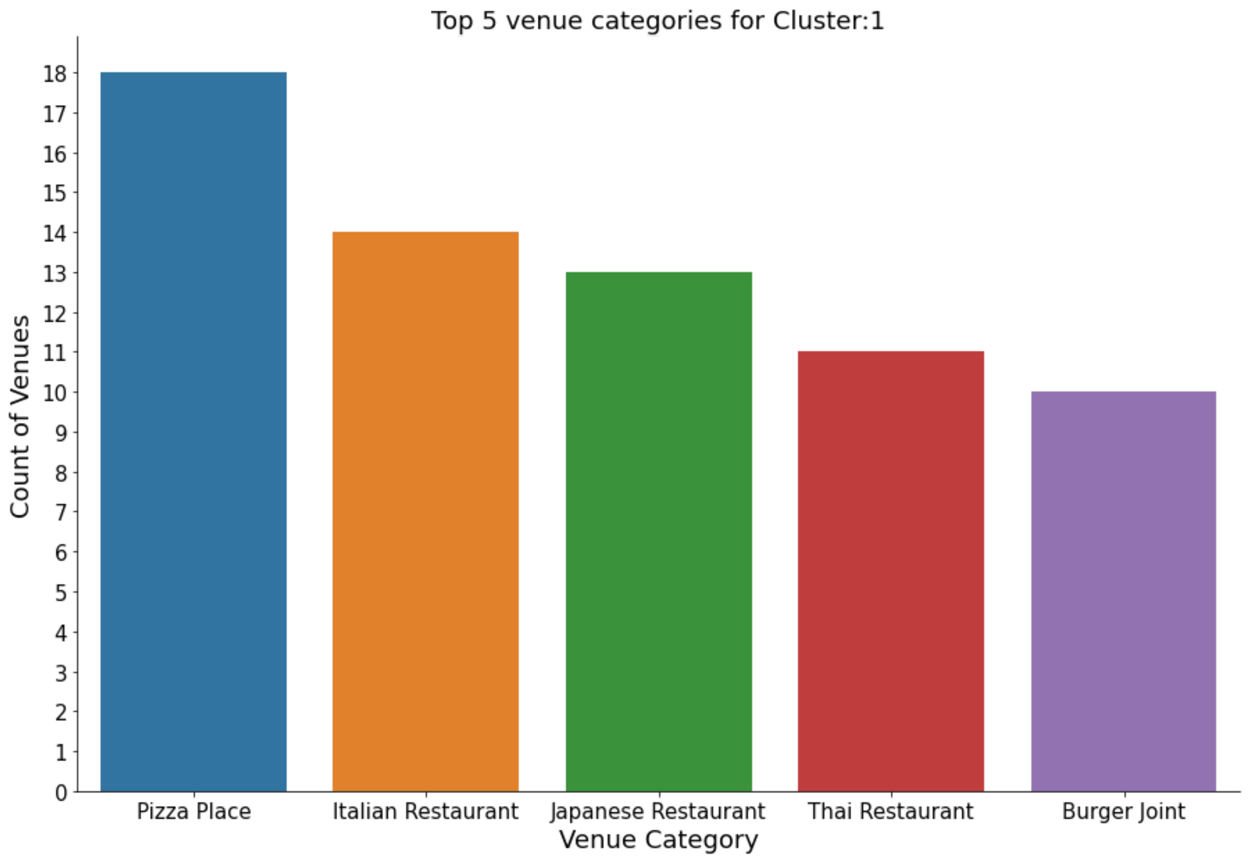


Figure 30: Top Venue Categories in Cluster 1

Cluster 2:
The suburbs in this cluster mainly consist mostly of north-western suburbs of Melbourne. Although there are only few suburbs in this cluster, mainly because there wasn't enough data returned from the Foursquare API for these suburbs.

However, as per the 2016 Australian Census, there is a high percentage of Vietnamese along with Chinese ethnicities form a major part of the demographics in these regions.

Therefore due to the distribution of the demographics, it can be assumed that ***"Vietnamese", "Chinese"*** and other kinds of ***"Asian"*** venues are more prominent within these suburbs.

Hence, this cluster can be categorised as predominantly ***"South-Asian"*** .

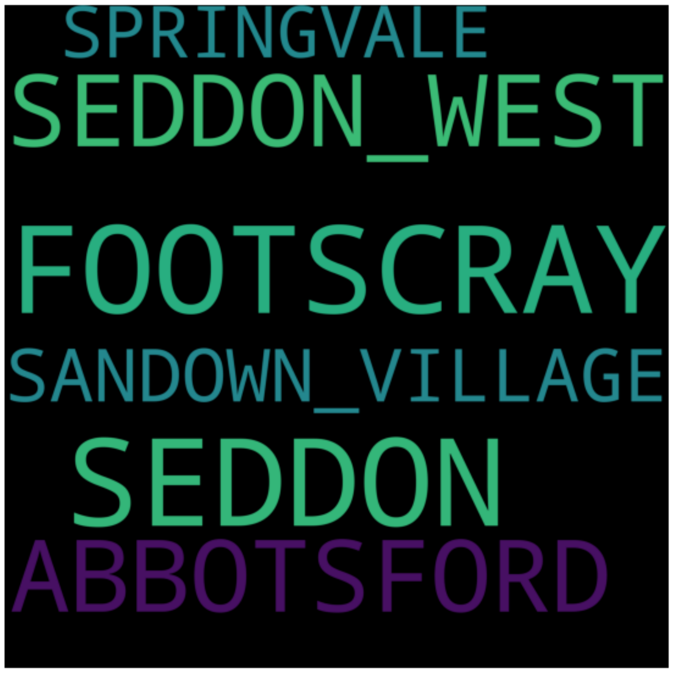


Figure 31: Suburbs in Cluster 2

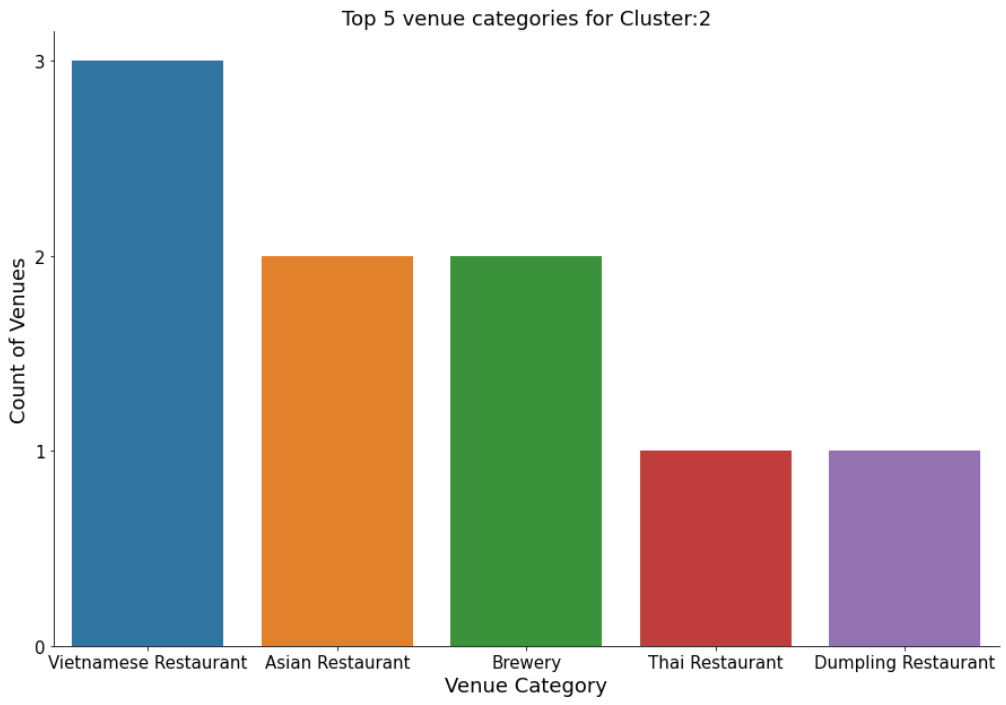


Figure 32: Top Venue Categories in Cluster 2

Cluster 3:
Most of the areas in this cluster are a hub of commercial activity in Melbourne and also various avenues for nightlife as well.

The areas in these clusters are multi-ethnic due as these areas offer a lot of work and academic opportunities for the multi-national ethnicities.

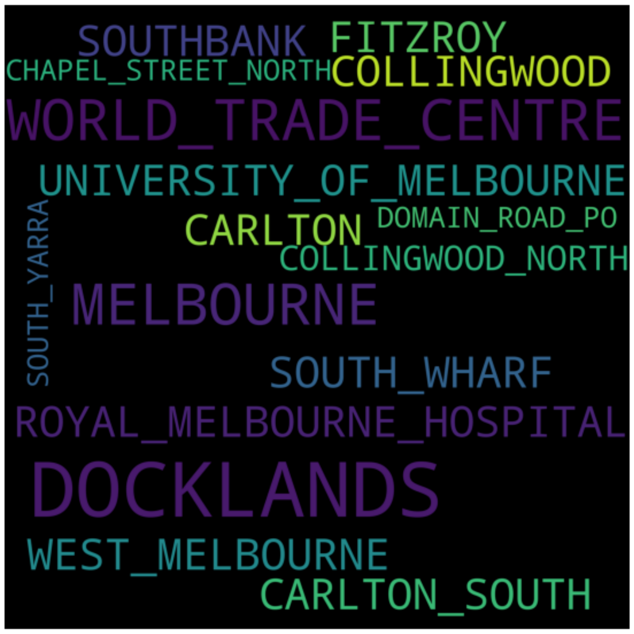


Figure 33: Suburbs in Cluster 3

The top venue categories in these suburbs are multi-ethnic in nature, hence this cluster can be termed as ***"multi-ethnic",*** however ***"Italian"*** and ***"Japanese"*** cuisines are the two prominent venue categories in this cluster.

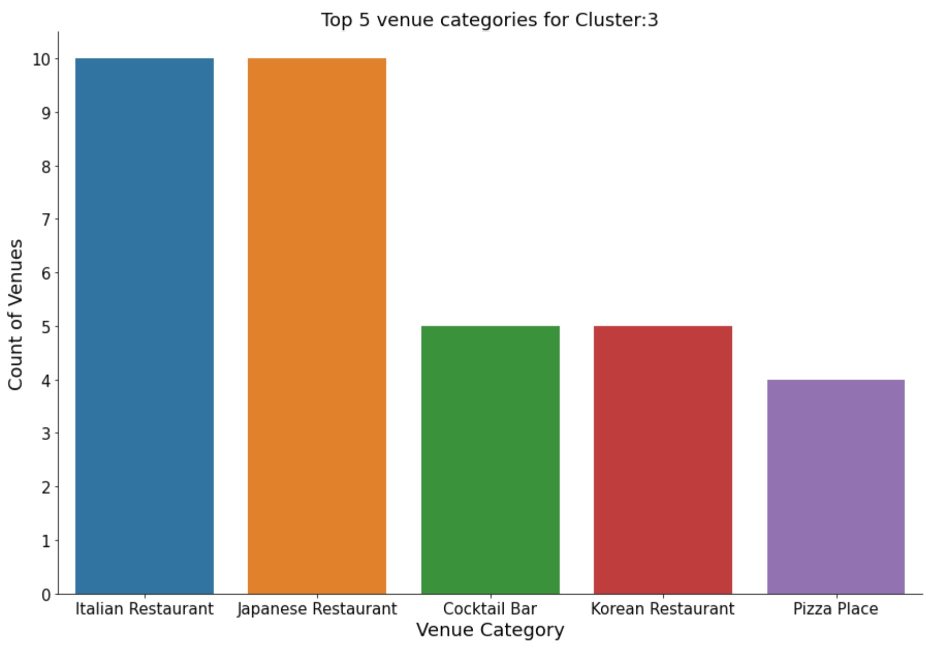


Figure 34: Top Venue Categories in Cluster 3

Cluster 4:

This cluster contains the most number of suburbs, a majority of the suburbs in this cluster lie towards the outer regions of Melbourne City. It can be inferred from **Figure 29** and **Figure 35**, the influence of the ethnicities decreases or food culture as one moves towards the outer regions of Melbourne.

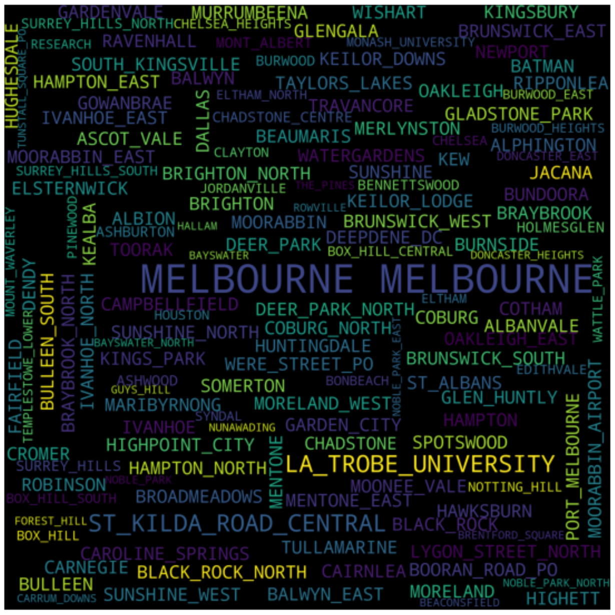


Figure 35: Suburbs in Cluster 4

Based on the data in **Figure 36,** this cluster predominantly contains ***"Fast-Food", "Pizza" and "Sandwich"*** joints. Hence, this cluster can be named as ***"Fast-Food"*** cluster.

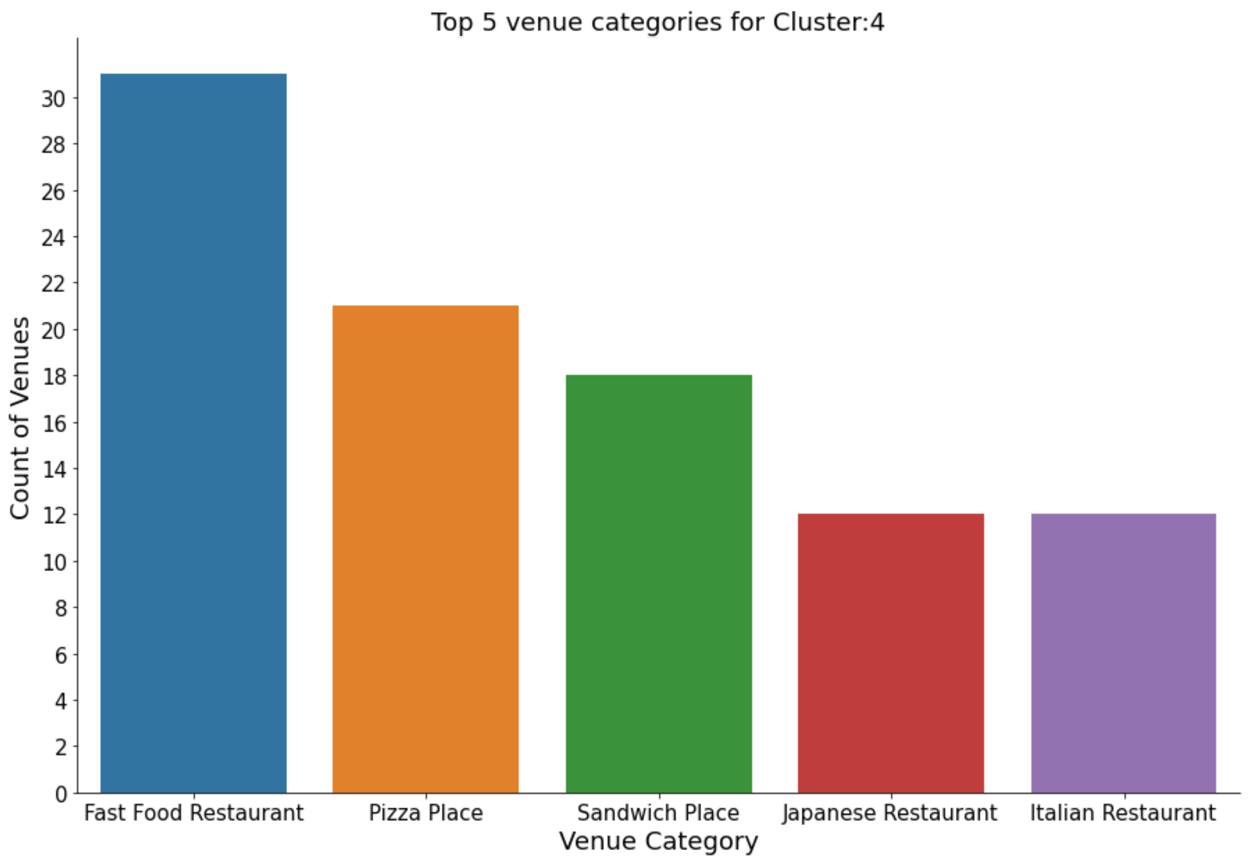


Figure 36: Top Venue Categories in Cluster 4

5. Discussion

In this project I have segmented the suburbs based on their food culture. There were mainly six clusters determined post implementation of K-means clustering on the data gathered about Melbourne postcode and the location based data from the Foursquare API.
Each of the clusters determined in this analysis has its unique characteristics as summarised in the table below:

Cluster Number	**Location Characteristics**	**Top Venue Characteristics**
1	Inner Suburbs: Predominantly Australian and Western-European Ethnicities	Pizza, Italian, Japanese, Thai, Burger
2	North- Western, Inner Western Suburbs: South Asian Ethnicities	Vietnamese, Chinese, Asian
3	Central Business District (CBD): Multi-Ethnicities	Italian, Japanese, Cocktail Bars, Korean, Pizza
4	Outer Suburbs: Multi-Ethnicities	Fast-Food

Table 4: Characteristics of Each Cluster

From this analysis a high level argument can be made that the demographics of suburbs may influence the food culture in different regions of the Melbourne city. However, this may require further investigation probably by integrating the census data in future.

However, from an individual standpoint one can get a fair understanding of the distribution of the food culture in Melbourne, which is:

- Most of the popular Australian cuisines can be found near the main city.
- The north western and western suburbs offer more ***"south-asian"*** cuisines.
- The further one moves away from the city, the food options are limited to ***"Fast-Food".***

Moreover, from a business standpoint the following can be inferred:

- The outer suburbs of Melbourne, although not too far from the city may be good options if a business is planning to expand further or anyone planning to start a new venture as the options there are limited to fast-food.
- Any food venture should foray into cuisines apart from Italian, Japanese, and Thai as there are abundant venues of these kinds in and around the city.

6. Conclusion

The aim of this project was to understand the distribution of the food culture in Melbourne city. Melbourne as a city offers a plethora of food venues, although some areas do specialise in a particular kind of cuisine which may be representative of the demographics or popular culture in the suburbs. Although this may require further investigation.

On the other hand, there were certain limitations in the project pertaining to the data collected from the Foursquare API as it was observed that there was a limited amount of data about some categories of cuisines or locations that could be gathered from the API; most of the venue categories returned from the API were non-food related or popular food categories. However, Since Melbourne has a wide array of food ventures, it might be possible that the data about these places may not have been available with the Foursquare platform.

Having said that, applications like Yelp, Zomato, or even Google Places API should be able to provide a wide variety of data about the food related venues. This possibly could be a potential opportunity to be a part of the future scope of the analysis.

Additionally future works can also integrate the census data from Australian Bureau of Statistics should be able to shed more light on the influence of demographics on the food culture in Melbourne.

7. References

1. Notebook created by Alex Aklson and Polong Lin for the ‘Applied Data Science Capstone’ course on Coursera

2. 2016 Census QuickStats: Australia. (2016). Retrieved 13 January 2021, from https://quickstats.censusdata.abs.gov.au/census_services/getproduct/census/2016/quickstat/036

3. Places API. Retrieved 2 January 2021, from https://developer.foursquare.com/docs/places-api/