# Coursera Capstone Project : Applied Data Science

Dorian Aigner

## Contents

## 1 Introduction

Vienna is one of the most international cities in Europe. Being regularly ranked the city with the best living quality by the Mercer Quality of Living Survey, it attracts people from all over the world. Furthermore, the UN headquarters and the International Atomic Energy Agency are located in the city. Hosting many people from all over the world implies a diversity in ethnic cousins. Additionally, Vienna's population is one of the richest in Europe with an annual average income of more than 55,000 USD (2019). A highly international population with high disposable income for eating out regularly, provides great opportunities for flourishing restaurants and food stalls of all kinds. From restaurants with a proud and long history deeply rooted in the country's traditions to new, flashy fusion cuisine, a vast selection is available.

## 2 Business Problem

Although this is true for the inner districts of the city, several districts may not have access to diverse food choices. Considering the overall high wages and the vast size of middle and high-income households, this fact provides many opportunities to start up an ethnic restaurant in a district less exposed to international cuisine yet.

Thus, the main objective of this project is to cluster Vienna's districts to find out where there is least competition and highest demand for such a restaurant. This data can be highly useful in combination with financial data over the purchasing power of the districts' households and the accessibility to the public transportation system to open a successful business and serve the international community in these needs.

## 3 Data

The data for this project has been retrieved and processed through multiple sources, giving careful considerations to the accuracy of the methods used.

### 3.1 Districts (Neighbourhoods)

The data of the districts in Vienna was accessed to a publicly available database of all districts in Austria. The file contents from zipcodes.at.json is retrieved into a Pandas DataFrame and afterwards sliced to comply to our needs of Viennese districts. The longitude and latitude are converted into floats to be processed later.

```
!wget -q -O 'zipcodes.at.json' https://github.com/zauberware/postal-codes-json-xml-csv/raw/master/data/AT/zipcodes.at.json


with open('zipcodes.at.json') as json_data:

    austria_data = json.load(json_data)


df = pd.DataFrame(austria_data)

df_vienna = df[df.state_code == '09']


def f(x):

    try:

        return np.float(x)

    except:

        return np.nan


df_vienna2["latitude"] = df["latitude"].apply(f)

df_vienna2["longitude"] = df["longitude"].apply(f)
```

### 3.2 Geocoding

The latitude and longitude of Vienna are retrieved using Google Maps Geocoding API. The geometric location values are then stored into the intial dataframe.

```
address = 'Vienna, Austria'


geolocator = Nominatim(user_agent="vienna_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude


# create map of Vienna using latitude and longitude values
map_vienna = folium.Map(location=[latitude, longitude], zoom_start=10)


# add markers to map
for lat, lng, state, place in zip(df_vienna2['latitude'], df_vienna2['longitude'], df_vienna2['state'], df_vienna2['place']):
    label = '{}, {}'.format(place, state)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_vienna)
```

### 3.3 Venue Data

From the location data obtained after Web Scraping and Geocoding, the venue data is accessed by searching in the FourSquare API, and creating another DataFrame to containall the venue details along with the respective districts.

```
CLIENT_ID = 'JBWU13BOHZRXJ0JHGVB0CUI0QKXRS35KSLDCGCKXCNLYZLHO' # your Foursquare ID
CLIENT_SECRET = 'PKO5CYLLADNNGQNFWPWE1DNHADQNBXRE2ABQZBJGPYHT3DMK' # your Foursquare Secret
VERSION = '20180604'
LIMIT = 30

def getNearbyVenues(names, latitudes, longitudes, radius=500):

    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)
        url = 
'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            LIMIT)

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

        # return only relevant information for each nearby venue
        venues_list.append([(
            name,
            lat,
            lng,
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

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

    return(nearby_venues)
```

## 4 Methodology

A thorough analysis of the principles of methods, rules, and postulates employed have been made in order to ensure the inferences to be made are as accurate as possible.

### 4.1 Accuracy of the Geocoding API

In the initial development phase with OpenCage Geocoder API, the number of erroneous results were of an appreciable amount, which led to the development of an algorithm to analyze the accuracy of the Geocoding API used. In the algorithm developed, Geocoding API from various providers were tested, and in the end, Google Maps Geocoder API turned out to have the least number of collisions (errors) in our analysis.

### 4.2 Folium

Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the leaflet.js library. All cluster visualization are done with help of Folium which in turn generates a Leaflet map made using OpenStreetMap technology.

### 4.3 One hot encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. For the K-means Clustering Algorithm, all unique items under Venue Category are one-hot encoded.

```
# one hot encoding
vienna_onehot = pd.get_dummies(vienna_venues[['Venue Category']], prefix="", prefix_sep="")


# add neighborhood column back to dataframe
vienna_onehot['Neighborhood'] = vienna_venues['Neighborhood']


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


vienna_onehot.head()
```

### 4.4 Top 10 most common venues

Due to high variety in the venues, only the top 10 common venues are selected and a new DataFrame is made, which is used to train the K-means Clustering Algorithm.

```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]

num_top_venues = 10

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Neighborhood']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = vienna_grouped['Neighborhood']

for ind in np.arange(vienna_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(vienna_grouped.iloc[ind, :], num_top_venues)

neighborhoods_venues_sorted.head()
```

## 4.5 Optimal number of clusters

Silhouette Score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. Based on the Silhouette Score of various clusters below 20, the optimal cluster size is determined.

```
def plot(x, y, xlabel, ylabel):
    plt.figure(figsize=(20,10))
    plt.plot(np.arange(2, x), y, 'o-')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xticks(np.arange(2, x))
    plt.show()

max_range = 20 # Maximum range of clusters

from sklearn.metrics import silhouette_samples, silhouette_score

indices = []
scores = []

for kclusters in range(2, max_range):

    # Run k-means clustering
    vgc = vienna_grouped_clustering
    kmeans = KMeans(n_clusters = kclusters, init = 'k-means++', random_state = 0).fit_predict(vgc)

    # Gets the score for the clustering operation performed
    score = silhouette_score(vgc, kmeans)

    # Appending the index and score to the respective lists
    indices.append(kclusters)
    scores.append(score)

plot(max_range, scores, "No. of clusters", "Silhouette Score")
```

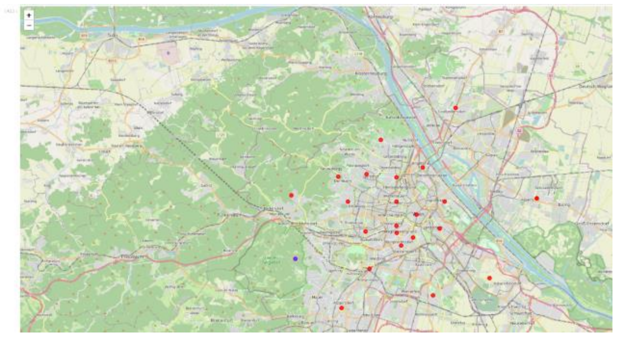
## 4.6 K-means clustering

The venue data is then trained using K-means Clustering Algorithm to get the desired clusters to base the analysis on. K-means was chosen as the variables (Venue Categories) are huge, and in such situations K-means will be computationally faster than other clustering algorithms.

```
# set number of clusters

kclusters = opt

# run k-means clustering

kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(vienna_grouped_clustering)


# check cluster labels generated for each row in the dataframe

kmeans.labels_[0:10]


# add clustering labels


neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)


vienna_merged = df_vienna2


# merge vienna_grouped with df_vienna2 to add latitude/longitude for each neighborhood

vienna_merged = vienna_merged.join(neighborhoods_venues_sorted.set_index('Neighborhood'), on='place')
```

## 5 Results

The neighbourhoods are divided into n clusters where n is the number of clusters found using the optimal approach. The clustered neighbourhoods are visualized using different colours so as to make them distinguishable.



## 6 Discussion

After analyzing the various clusters produced by the Machine learning algorithm, it becomes apparent that the Forequare API provides too limited information for several of the districts to run successfully. Therefore, it is recommended to repeat the described process with data from a different API, if applicable, a more local one.

However, even though the outcomes are only of partial use, the map and location data can be easily used to rebuild the proposed mechanism for other business cases. All in all, it shows that there is vast space for application.

## 7 Conclusion

As the trends described in the introduction have no end in sight, there is definitely demand. Using geodata and API information may allow entrepreneurs finding the best restaurant ideas for the best suitable area and therefore have a share of this rise in consumption. However, the process has shown that API information vary vastly between countries and local solutions of data usage has to be considered when looking deeper into this business idea.