# Airbnb in Amsterdam

July 16, 2019

#### 0.0.1 Authors: William Bates and Karan Sunil

## 1 Initial Inquiries

On looking through a dataset of Airbnb listings, we noticed that they had a score for the location of a listing. But shouldn't the location variable be dynamic rather than static? A listing could have the best location according to one person and the worst location according to another person. Everyone has different requirements and preferences in a location, so is it possible to devise a score about the location of a listing that gives users the ability to choose what is important to them?

This study will ensure that customers are recommended the best possible listings based on their needs and requirements. This could significantly improve customer satisfaction at Airbnb, which would in turn increase brand loyalty and lead to an increase in return customers.

Secondly, we wanted to get a better understanding of what drives the prices of Airbnb listings. What factors affect price the most? Can we predict the approximate price value of a listing with just some very basic information about the property? Our model would be useful for first time Airbnb renters, as this would help them get an understanding of how their property should be valued in comparison to other listings. In addition to providing renters some help in valuing their homes, this information could give Airbnb the opportunity to regulate against those renters who are exploiting customers.

## 2 Imports

```
In [3]: #importing standard packages
        import os
        import json
        import re

        #importing the required pandas and numpy packages
        import numpy as np
        import pandas as pd
        import geopandas as gpd

        #importing the required plotting packages
        import matplotlib.pyplot as plt
        import folium
        from folium.plugins import FastMarkerCluster
```

```
import seaborn as sns
from branca.colormap import LinearColormap
from shapely.geometry import Point
from shapely.geometry import Polygon

#importing the sklearn packages for machine learning
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer

#importing gis packages and logging onto the server
import arcgis
from arcgis.gis import GIS
from arcgis import geometry
from arcgis.features import GeoAccessor, GeoSeriesAccessor
from arcgis.features import use_proximity
from arcgis.features.manage_data import overlay_layers
from arcgis.geoenrichment import *

#for spatial statistics
import pysal as ps
from shapely.wkt import loads

gis = GIS(username='')  # this requires a GIS account
arcgis.__version__
```

```
Enter password: ůůůůůůůů
```

```
Out[3]: '1.6.0'
```

# 3 Data Sources

- https://www.kaggle.com/erikbruin/airbnb-amsterdam/version/1
- This dataset contains information about airbnb listings in Amsterdam. We also have information about the different neighbourhoods in Amsterdam.
- https://ucsdonline.maps.arcgis.com/home/item.html?id=ecefe11d5118489c9db2f5b8a76686c1
- This feature layer contains the tram and metro stations in Amsterdam. Currently, we weigh all metro and tram stops the same, but in further studies it should be recognized that some stops have access to more locations and should be given greater value.
- https://www.kaggle.com/harshmehta6711/attractions

- This dataset contains information about the different tourist attractions in Amsterdam. It should not be considered a complete list of all reasons to visit Amsterdam, but will suffice for initial modeling.

To recreate this project, these files should be accessed in a 'data' folder, so that they could be retrieved via the following code:

```
In [24]: #retreiving file paths
         fp_listings_details =  os.path.join('data','listings_details.csv')
         fp_attractions = os.path.join('data', 'amsterdam_attraction.csv')
         #loading in dataframes
         listing_details_unfiltered = pd.read_csv(fp_listings_details, engine='python', error_
         attractions_unfiltered = pd.read_csv(fp_attractions, dtype = str)
```

```
Skipping line 10322: unexpected end of data
```

# 4 Data Cleaning

This section is mostly intuitive and aesthetic and therefore not necessary to read, though it does change some variable names and should be run if attempting to recreate this project.

```
In [5]: fp_neighbourhood_gjson = os.path.join('data','neighbourhoods.geojson')
        fp_out_neighbourhoods = 'data/neighbourhoods.shp'
        #converting geojson to a shp file
        neighbourhoods = gpd.read_file(fp_neighbourhood_gjson).drop(columns = ['neighbourhood_g
        neighbourhoods.to_file(fp_out_neighbourhoods)
        neighbourhoods = pd.DataFrame.spatial.from_featureclass(fp_out_neighbourhoods)
        #creating a feature layer allowing us to access the data easier in the future
        neighbourhood_layer =  gis.content.search('d2903211d31c4fa0adfbfb951cea3145')
```

```
In [6]: listings = listing_details_unfiltered[['id',
               'host_id', 'host_since',
               'host_is_superhost', 'host_listings_count',
               'host_identity_verified','latitude', 'longitude',
               'property_type', 'room_type', 'accommodates',
               'bathrooms', 'bedrooms', 'beds', 'bed_type', 'neighbourhood_cleansed',
               'price', 'weekly_price', 'monthly_price', 'security_deposit',
               'cleaning_fee', 'guests_included', 'extra_people', 'minimum_nights',
               'maximum_nights', 'has_availability', 'number_of_reviews','review_scores_rating
               'review_scores_location', 'instant_bookable','is_business_travel_ready',
               'calculated_host_listings_count','neighbourhood']].copy()
```

```
In [7]: #converting all 't' to 1 and 'f' to 0
        listings.host_is_superhost = ([1 if x == 't' else 0 for x in listings.host_is_superhos
        listings.host_identity_verified = ([1 if x == 't' else 0 for x in listings.host_identi
        listings.has_availability = ([1 if x == 't' else 0 for x in listings.has_availability]
        listings.instant_bookable = ([1 if x == 't' else 0 for x in listings.instant_bookable]
        listings.is_business_travel_ready = ([1 if x == 't' else 0 for x in listings.is_busine
        listings.calculated_host_listings_count = ([1 if x == 't' else 0 for x in listings.cal
```

```
In [8]: #filling all null prices with 0 usd
        listings = listings.fillna({'price':'$0', 'weekly_price':'$0','monthly_price':'$0','se
        #filling all null dates with 0000-00-00
        listings = listings.fillna({'host_since':'0000-00-00'})
```

```
In [9]: #converting all prices to floats by stripping dollar sign and ',' signs
        listings.price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.price]
        listings.weekly_price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.wee
        listings.monthly_price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.mon
        listings.security_deposit = [float(re.sub(',','',(x.split('$')[1]))) for x in listings
        listings.cleaning_fee = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.clea
        listings.extra_people = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.ext
```

```
In [10]: #extracting just the year the member has been with airbnb since
         listings.host_since = [int(re.findall('\d{4}', x)[0]) for x in listings.host_since]
         #on checking the number of listings each of the missing hosts have, we can see that w
         listings = listings.fillna({'host_listings_count':1})
         #creating a spatial dataframe from the listing data
         listings = pd.DataFrame.spatial.from_xy(listings,y_column = 'longitude', x_column='lat
         #there are just 24 rows where the following columns are null and we are better of dro
         listings = listings.dropna(subset = ['bathrooms', 'bedrooms', 'beds'])
         listings = listings.reset_index()
```

```
In [11]: #converting sub-categories into much larger more general categories
         nature = ['Park', 'Garden','Scenic Lookout','Lake','River','Trail','Beach','Farm']
         nightlife = ['Beer Garden','Music Venue','Nightclub','Bar','Strip Club','Hookah Bar',
                      'Sake Bar','Gay Bar','Whisky Bar', 'Hotel Bar','Pub','Other Nightlife','
                     'Karaoke Bar','Cocktail Bar','Wine Bar','Rock Club','Speakeasy','Piano Ba
         sports = ['Athletics & Sports','Sports Bar', 'Football Stadium','Baseball Stadium','S
                  'Basketball Stadium','Track Stadium','Racetrack']
         religion = ['Spiritual Center', 'Shrine','Synagogue','Mosque', 'Temple','Church']
         adventure = ['Surf Spot', 'Ski Area', 'Ski Lodge','Outdoors & Recreation']
         history = ['Museum','Historic Site', 'History Museum','Castle','Science Museum']
         arts = ['Theater','Jazz Club', 'Art Museum','Arts & Entertainment','Art Gallery',\
                'Public Art', 'Concert Hall','Comedy Club','Performing Arts Venue','General E
         shopping = ['Flea Market','Mall','Market','Flower Shop','Smoke Shop']
         food = ['Restaurant','Café','Lounge']
         monuments = ['Building','Plaza','Bridge','Harbor / Marina','Sculpture Garden','Lighth
         #filtering out the last 1000 rows as the data does not contain enough information
         attractions_filtered = attractions_unfiltered.loc[:2316]
         #drop unrequired columns
         attractions = attractions_filtered.drop(columns = ['location','category','details', '
         #converting the following to coordinates to floats
         attractions.lng = pd.to_numeric(attractions.lng, errors='coerce')
         attractions.lat = pd.to_numeric(attractions.lat, errors = 'coerce')
         #creating spatial dataframe
         attractions = attractions.spatial.from_xy(attractions, x_column = 'lat', y_column = '
         attractions = attractions.drop(columns = ['address'])
```

```
attractions = attractions.fillna('Other')
attractions.columns = ['lat','lon','NAME', 'CAT', 'SHAPE']
attractions.to_csv('data/attractions_cleaned.csv')

#completing the data cleaning by imputing the median
imp_median = SimpleImputer(missing_values=np.nan, strategy='median')
imp_median.fit(listings[['review_scores_location', 'review_scores_rating']].copy().dro
imputed_vals = imp_median.transform(listings[['review_scores_location', 'review_score
imputed_df = pd.DataFrame(imputed_vals, columns = ['review_scores_location', 'review_s
listings['review_scores_location'] = imputed_df['review_scores_location']
listings['review_scores_rating'] = imputed_df['review_scores_rating']
```

## 5 Descriptive Statistics on Neighborhoods and Listings

To generate some descriptive statistics about the neighborhoods of Amsterdam dataset, we used PySAL to conduct some basic spatial analysis. We got the neighbor weights of each neighborhood, for which we had to flatten the coordinates of each neighborhood polygon from 3-D to 2-D. With only x and y coordinates to consider, we could assess which neighborhoods connected to one another.

About 19% of possible neighborhood intersections did turn out to be neighbor pairs, which we felt was fairly high. Using Queen-wise weights as our standard, the average number of neighbors for a neighborhood was 4.18 with a standard deviation of 1.59. The total number of neighborhoods in the dataset was 22.

We also did K-Nearest-Neighbors analysis on the neighborhoods data to get neighbors by this metric. First, we set the radius to Earth's radius so as to account for the curvature of the Earth in our distance calculations. We set k=3 in this analysis as a choice based on the limited number of neighborhoods in the dataset. This afforded us with a determination of nearby neighborhoods to one another in Amsterdam.

```
In [12]: lats = listings['latitude'].tolist()
         lons = listings['longitude'].tolist()
         locations = list(zip(lats, lons))

         map1 = folium.Map(location=[52.3680, 4.9036], zoom_start=11.5)
         FastMarkerCluster(data=locations).add_to(map1)
         map1

Out[12]: <folium.folium.Map at 0x7fcef5f70400>

In [13]: # Creating DataFrame with mean price for each neighobuurhood
         neighbourhoods = gpd.read_file(fp_neighbourhood_gjson).drop(columns = ['neighbourhood_
         mean_prices = listings.groupby('neighbourhood_cleansed')['price'].mean().sort_values(a
         mean_prices = pd.DataFrame([mean_prices])
         mean_prices = mean_prices.T

         # Merging mean prices with the neighbourhood information
         neighbourhoods = pd.merge(neighbourhoods, mean_prices, right_on='neighbourhood_cleans
         neighbourhoods.rename(columns={'price': 'average_price'}, inplace=True)
```

```
neighbourhoods.average_price = neighbourhoods.average_price.round(decimals=0)

# Creating colormap for the plot
map_dict = neighbourhoods.set_index('neighbourhood')['average_price'].to_dict()
color_scale = LinearColormap(['yellow','red'], vmin = min(map_dict.values()), vmax = m

# Creating function to get the colour to fill the polygon with
def get_color(feature):
    value = map_dict.get(feature['properties']['neighbourhood'])
    return color_scale(value)

# Creating map and plotting results
map2 = folium.Map(location=[52.3680, 4.9036], zoom_start=11)
folium.GeoJson(data=neighbourhoods,
               name='Amsterdam',
               tooltip=folium.features.GeoJsonTooltip(fields=['neighbourhood', 'averag
                                                       labels=True,
                                                       sticky=False),
               style_function= lambda feature: {
                   'fillColor': get_color(feature),
                   'color': 'black',
                   'weight': 1,
                   'dashArray': '5, 5',
                   'fillOpacity':0.5
                   },
               highlight_function=lambda feature: {'weight':3, 'fillColor': get_color
map2
```

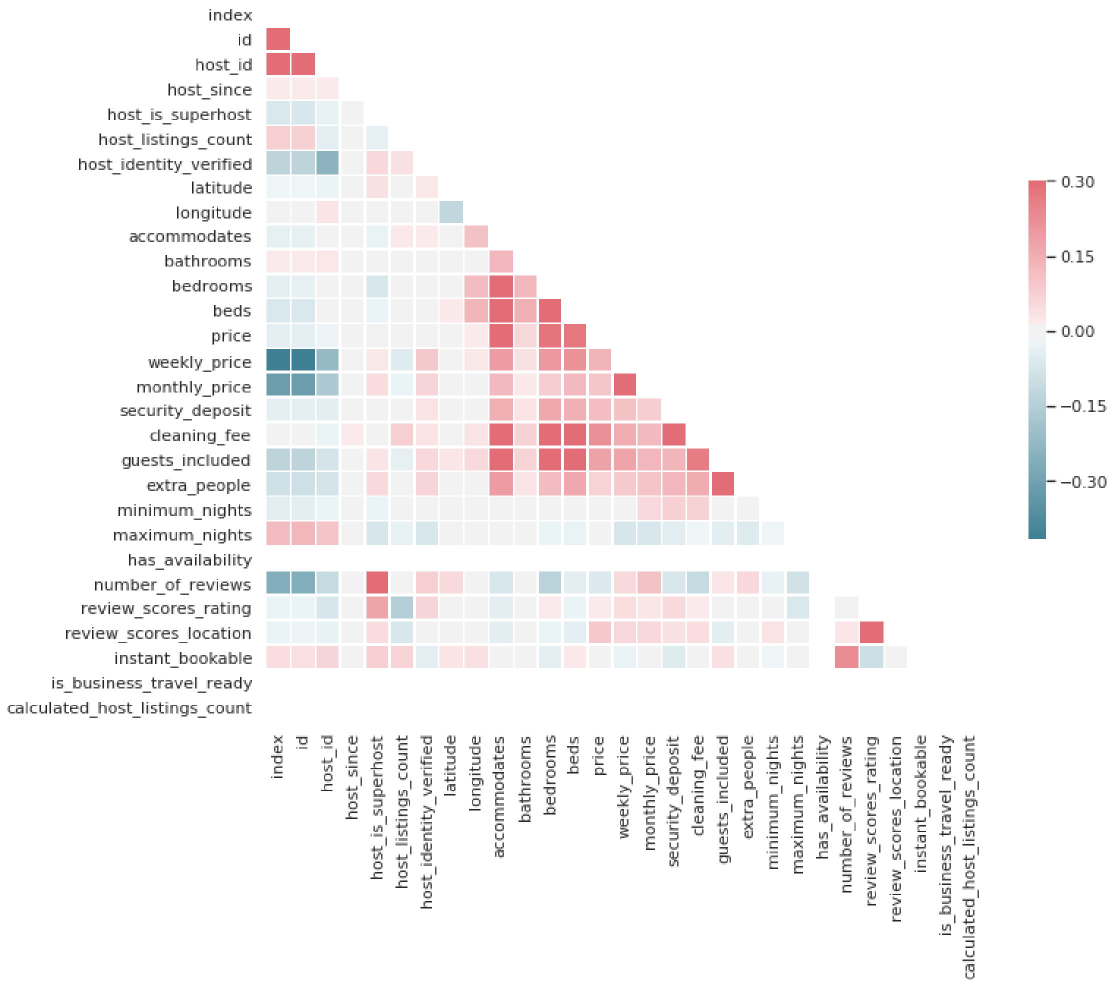
Out[13]: <folium.folium.Map at 0x7fcef5f79c50>

The neighborhoods with the highest average price for a room are Centrum-West and Centrum-Oost. De Wallen, the red light district in Amsterdam, is on the border of these two neighborhoods. This suggests that most people are visiting Amsterdam for the nightlife.

### Correlation Matrix of Relevant Listing Columns

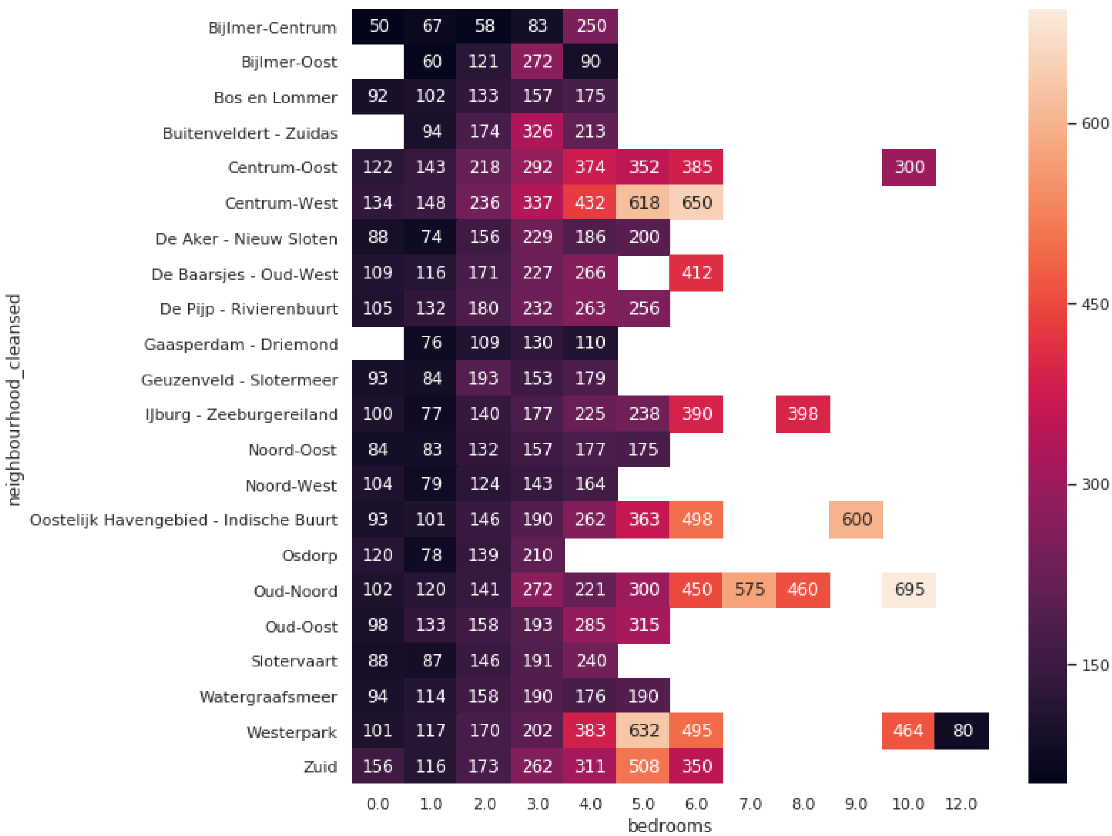
In [14]:
```
sns.set(style="white")

# Compute the correlation matrix
corr = listings.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))
```

```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
ax = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



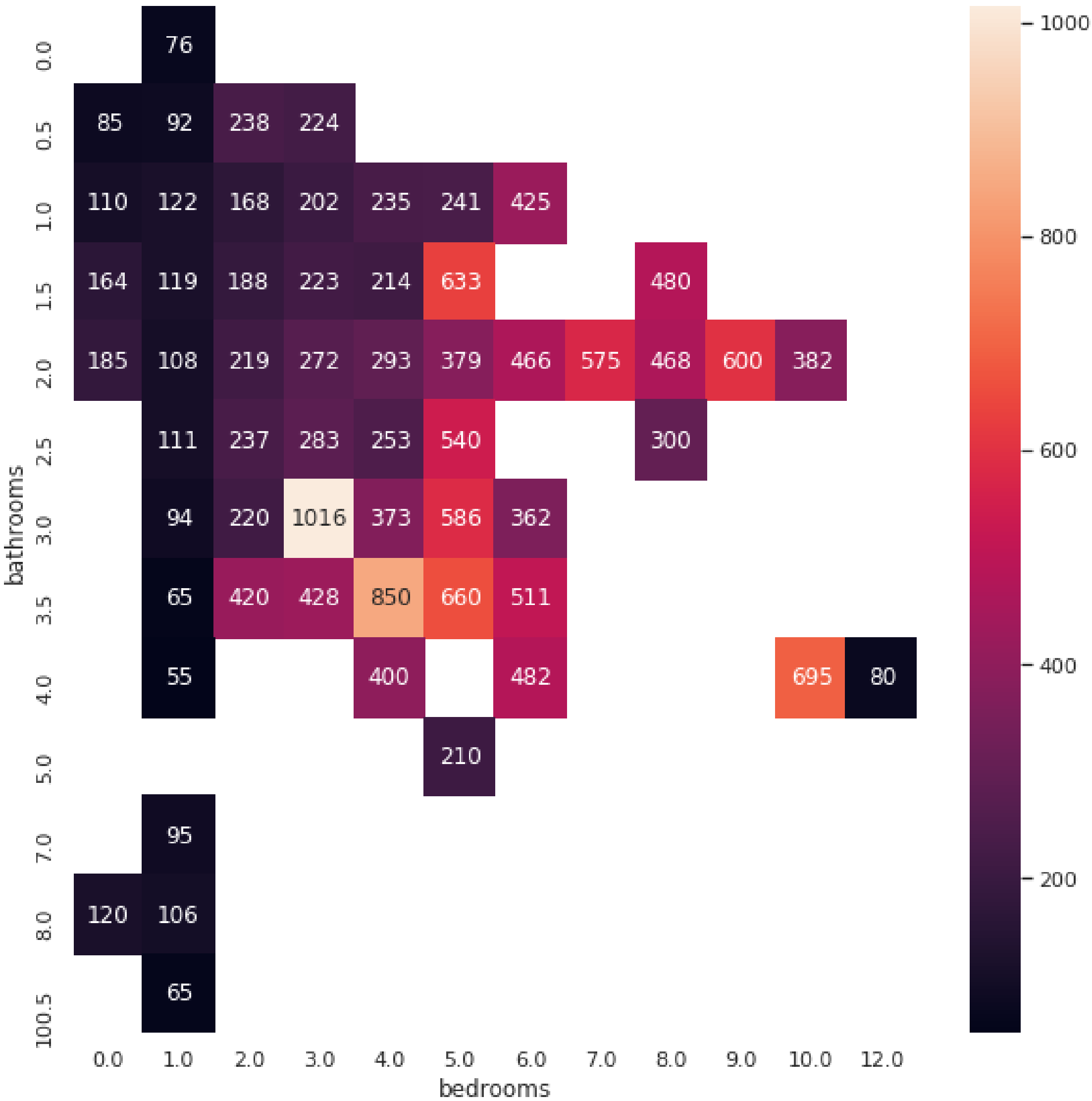

### Heatmap of Housing Prices Based on Bedrooms per Neighborhood

```
In [15]: plt.figure(figsize=(10,10))
         ax = sns.heatmap(listings.groupby([
                 'neighbourhood_cleansed', 'bedrooms']).price.mean().unstack(),annot=True, fmt=
```



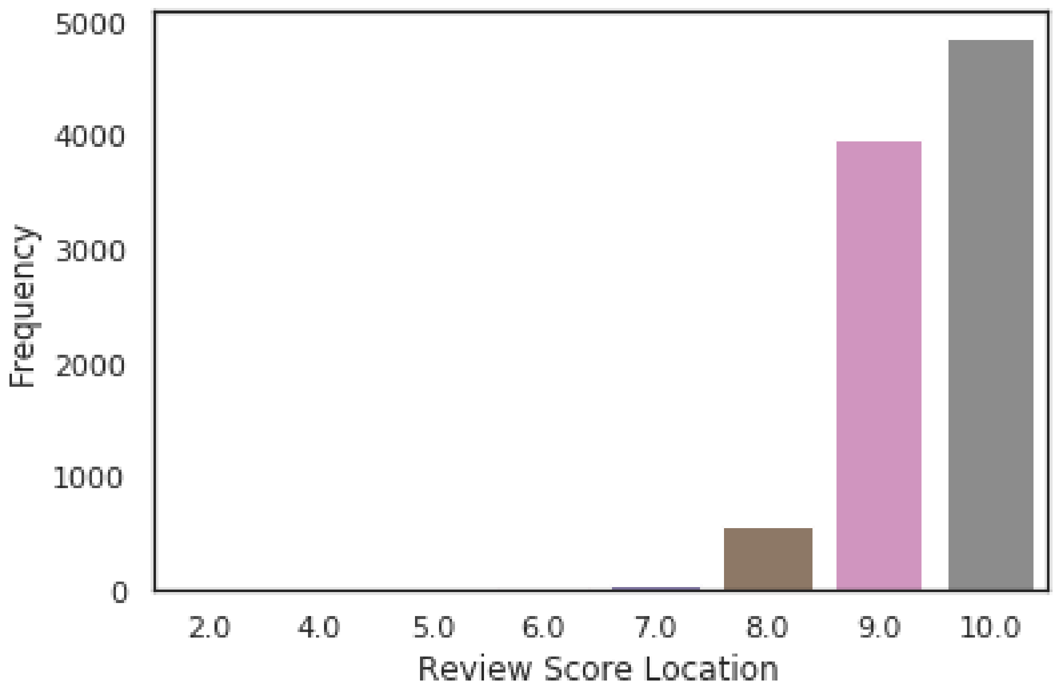

### Heatmap of Housing Prices Based on Bedrooms and Bathrooms

```
In [16]: plt.figure(figsize=(10,10))
         ax = sns.heatmap(listings.groupby([
                 'bathrooms', 'bedrooms']).price.mean().unstack(),annot=True, fmt=".0f")
```




### Abnormal Frequency of Location Review Scores

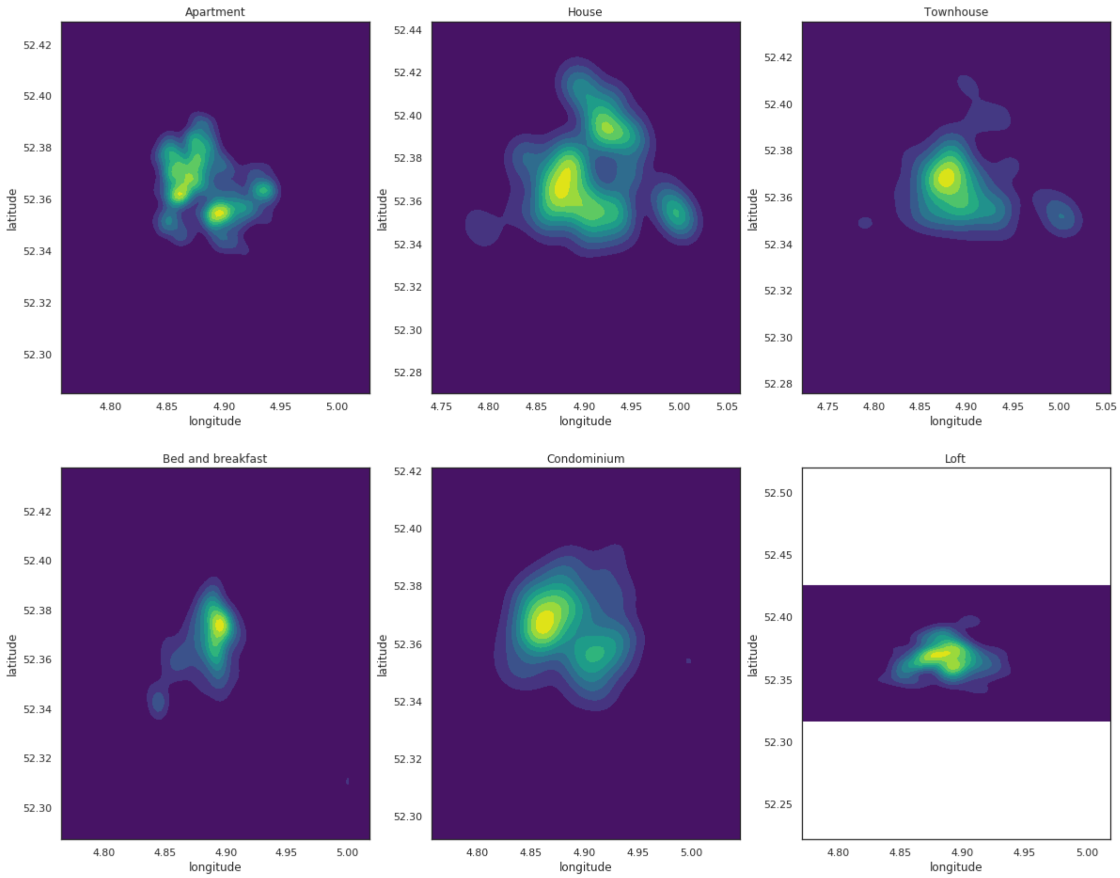
In [17]:
```
ax = sns.barplot(listing_details_unfiltered.review_scores_location.value_counts().inde
ax.set_xlabel('Review Score Location')
y = ax.set_ylabel('Frequency')
```




**KDE Plot of Listings by Latitude and Longitude for each Property Type**

```
In [18]: rank = listings.property_type.groupby(listings.property_type).\
                     count().sort_values(ascending=False)

         f, axs = plt.subplots(2,3,figsize=(20,16))
         axs = axs.flatten()
         for i, sub in enumerate(rank.head(6).index.tolist()):
             ax = axs[i]
             subdf = listings.loc[listings['property_type']==sub,:]
             sns.kdeplot(subdf['longitude'], subdf['latitude'], shade=True,
                         cmap='viridis',ax=ax)
             ax.set_title(sub)
             plt.axis('equal')
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tu
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



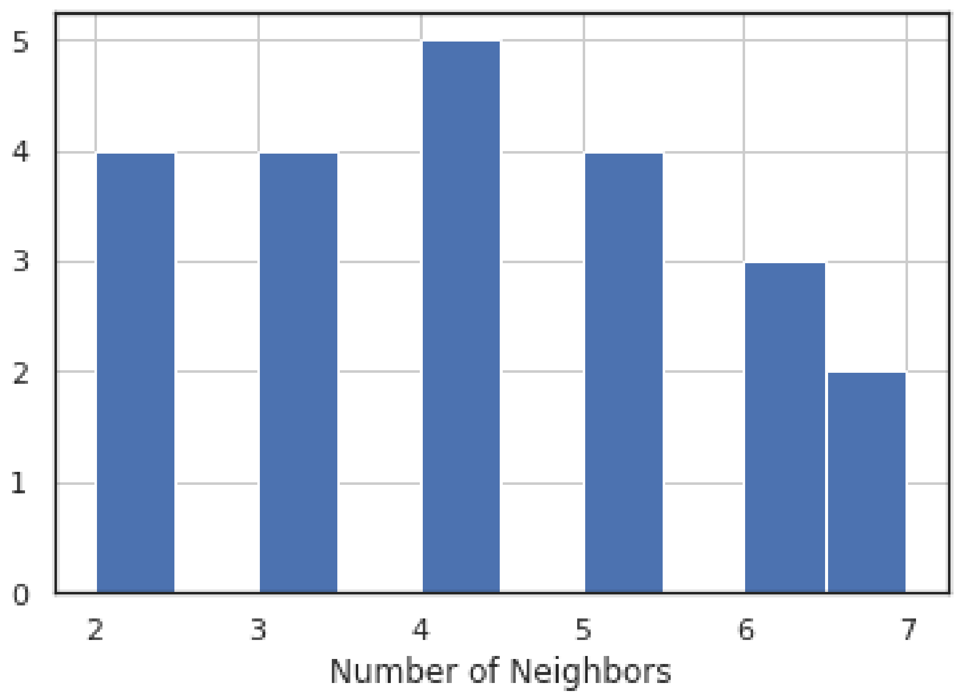

### 5.0.1 PySAL Neighbor Analysis

```
In [19]: # Flatten neighborhoods to 2D for this analysis
         shp_path = "data/neighbourhoods.shp"
         dataframe = gpd.read_file(shp_path)
         dataframe.head()
         var = loads(dataframe.loc[0].geometry.boundary.wkt)
         line_2d = Polygon([xy[0:2] for xy in list(var.coords)])
         coords_2d = list()
         for i in range(len(dataframe.geometry)):
             var = loads(dataframe.loc[i].geometry.boundary.wkt)
             line_2d = Polygon([xy[0:2] for xy in list(var.coords)])
             coords_2d.append(line_2d)

         # Get Queen weights of each neighborhood's coordinates
         dataframe['geometry'] = coords_2d
         dataframe.to_file('2d_hoods.shp')
         shp_path2 = "2d_hoods.shp"
         shp = ps.lib.io.open(shp_path2)
         qW = ps.lib.weights.Queen(shp)
```

```
hoods_df = gpd.read_file(shp_path2)

# Normalize and fill rows, then calculate percent of potential intersections which ar
Wmatrix, ids = qW.full()
print('Percent Non-Zero: ' + str(qW.pct_nonzero))

# Basic Description of Number of Neighbors
n_neighbors = pd.Series(list(Wmatrix.sum(axis=1)))
ax = n_neighbors.hist()
ax.set_xlabel('Number of Neighbors');
print('Mean: ' + str(n_neighbors.mean()))
print('Standard Deviation: ' + str(n_neighbors.std()))
```

```
Percent Non-Zero: 19.00826446280992
Mean: 4.181818181818182
Standard Deviation: 1.592732412176175
```

```
/opt/conda/lib/python3.6/site-packages/pysal/lib/weights/weights.py:170: UserWarning: The weig
  warnings.warn("The weights matrix is not fully connected. There are %d components" % self.n_
```




```
In [20]: # K-Nearest-Neighbors Analysis in accordance with Earth's curvature
         radius = ps.lib.cg.sphere.RADIUS_EARTH_MILES
```

```
knn = ps.lib.weights.KNN.from_shapefile(shp_path2, k=3, radius=radius)

# Neighborhoods closest to Bijlmer-Oost by this method
locs = [0]
for i in knn[0]:
    locs.append(i)
print('Neighborhoods Closest to Bijlmer-Oost:')
hoods_df.loc[locs].reset_index(drop=True)
```

```
Neighborhoods Closest to Bijlmer-Oost:
```

```
Out[20]:             neighbourh                                             geometry
         0          Bijlmer-Oost  POLYGON ((4.991669 52.324436, 4.991756 52.3242...
         1  Gaasperdam - Driemond  POLYGON ((5.021543 52.302457, 5.020643 52.3024...
         2        Bijlmer-Centrum  POLYGON ((4.97184 52.28436, 4.971694 52.284262...
         3        Watergraafsmeer  POLYGON ((4.969713 52.356363, 4.969595 52.3561...
```

# 6 Analysis Preparation

We created two models. The first recommends Airbnb listings based on a users preferences. The second predicts the price of a property based on information about the listing.

In our initial plan, we wanted to convert the neighbourhood field to represent an ordinal variable. To do that we wanted to compute a score for each neighbourhood. To calculate the score we wanted to use factors like the safety, greenery, pollution and number of canals in each neighbourhood. We had attempted to do this using raster layers. However, we did run into numerous problems when working with raster calculators and decided against using this feature.

### Preparation (May Take a While to Run)

```
In [31]: # Getting all data as feature layers
         attractions_fl = gis.content.get('2d917955e3024ee9bf4fb257ff94cd73')
         listing_fl = gis.content.search('419b634ba980445ea29c612aef9c839c')
         tram_metro = gis.content.get('ecefe11d5118489c9db2f5b8a76686c1')

         # Converting listings DataFrame to a GeoDataFrame
         listings = pd.read_csv(fp_listings_details, engine='python', error_bad_lines=False)[[
         listings.to_csv('data/working_listings.csv')
         fp_working_listings =  os.path.join('data','working_listings.csv')
         listings_gpd = gpd.GeoDataFrame.from_csv(fp_working_listings)[['latitude', 'longitude
         #listings_gpd = pd.read_csv(fp_listings_details, engine='python', error_bad_lines=Fal
         listings_gpd['Coordinates'] = list(zip(listings_gpd.longitude, listings_gpd.latitude)
         listings_gpd['Coordinates'] = listings_gpd['Coordinates'].apply(Point)
         listings_gpd = gpd.GeoDataFrame(listings_gpd, geometry='Coordinates')
         listings_gpd.set_geometry('Coordinates')
         listings_gpd.crs = {'init' :'epsg:4326'}
         listings_gpd = listings_gpd.to_crs({'init': 'epsg:2230'})
         listings_gpd.to_csv('data/converted_listings.csv')
```

```
Skipping line 10322: unexpected end of data
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:10: FutureWarning: from_csv is dep
  # Remove the CWD from sys.path while we load stuff.
```

```
In [45]: temp = pd.read_csv('data/converted_listings.csv')
         temp['Coordinates'] = list(zip(temp.longitude, temp.latitude))
         temp['Coordinates'] = temp['Coordinates'].apply(Point)
         #temp['Coordinates'] = temp.Coordinates.apply(lambda x: x.split('(')[1]).apply(lambda
         #temp = temp.loc[:,'id':]
         listings_gpd = gpd.GeoDataFrame(temp, geometry='Coordinates')
         listings_gpd.set_geometry('Coordinates')
         listings_gpd.crs = {'init' :'epsg:2230'}
         listings_gpd.head()
```

```
Out[45]:    Unnamed: 0  index   latitude  longitude  \
         0           0      0  52.365755   4.941419
         1           1      1  52.390225   4.873924
         2           2      2  52.365087   4.893541
         3           3      3  52.373114   4.883668
         4           4      4  52.386727   4.892078

                                        Coordinates
         0  POINT (4.941419235184398 52.36575451387618)
         1  POINT (4.873924094742859 52.39022505041117)
         2   POINT (4.89354100784101 52.36508702680917)
         3  POINT (4.883668196205626 52.37311440038617)
         4  POINT (4.892078070089338 52.38672731612468)
```

**Calculation of Distance to Public Transportation** For our model, we first calculated the distance to the nearest metro/tram station. To do this, we used GeoPandas' distance function. However, the distance between the listing and the nearest tram station is a euclidean distance and not the true walking distance. People who want to use the metro/tram do not have any other means of transportation and using the true walking distance would help increase the accuracy of our results.

```
In [46]: # Converting tram_metro layer to a geopandas
         temp = tram_metro.layers[0].query().sdf
         temp['lon'] = temp.WKT_LNG_LAT.apply(lambda x: float(x.split('(')[1].split(',')[0]))
         temp['lat'] = temp.WKT_LNG_LAT.apply(lambda x: float(x.split('(')[1].split(',')[1][:-
         temp['Coordinates'] = list(zip(temp.lon, temp.lat))
         temp['Coordinates'] = temp['Coordinates'].apply(Point)
         tram_metro_gpd = gpd.GeoDataFrame(temp, geometry='Coordinates')
         tram_metro_gpd.set_geometry('Coordinates')
         tram_metro_gpd.crs = {'init' :'epsg:4326'}
         tram_metro_gpd = tram_metro_gpd.to_crs({'init': 'epsg:2230'})
```

```
In [47]: # Calculating minimum distance
         min_distances = []
```

```
for x in listings_gpd.index:
    min_distances.append(min(tram_metro_gpd.distance(listings_gpd.loc[x,'Coordinates']
```

In [67]:
```
# Re-acquiring listings dataset
listings = listing_details_unfiltered[['id',
       'host_id', 'host_since',
       'host_is_superhost', 'host_listings_count',
       'host_identity_verified','latitude', 'longitude',
       'property_type', 'room_type', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'bed_type', 'neighbourhood_cleansed',
       'price', 'weekly_price', 'monthly_price', 'security_deposit',
       'cleaning_fee', 'guests_included', 'extra_people', 'minimum_nights',
       'maximum_nights', 'has_availability', 'number_of_reviews','review_scores_rating
       'review_scores_location', 'instant_bookable','is_business_travel_ready',
       'calculated_host_listings_count','neighbourhood']].copy()

#converting all 't' to 1 and 'f' to 0
listings.host_is_superhost = ([1 if x == 't' else 0 for x in listings.host_is_superhos
listings.host_identity_verified = ([1 if x == 't' else 0 for x in listings.host_identi
listings.has_availability = ([1 if x == 't' else 0 for x in listings.has_availability]
listings.instant_bookable = ([1 if x == 't' else 0 for x in listings.instant_bookable]
listings.is_business_travel_ready = ([1 if x == 't' else 0 for x in listings.is_busine
listings.calculated_host_listings_count = ([1 if x == 't' else 0 for x in listings.cal

#filling all null prices with 0 usd
listings = listings.fillna({'price':'$0', 'weekly_price':'$0','monthly_price':'$0','se
#filling all null dates with 0000-00-00
listings = listings.fillna({'host_since':'0000-00-00'})

#converting all prices to floats by stripping dollar sign and ',' signs
listings.price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.price]
listings.weekly_price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.wee
listings.monthly_price = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.mo
listings.security_deposit = [float(re.sub(',','',(x.split('$')[1]))) for x in listings
listings.cleaning_fee = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.cle
listings.extra_people = [float(re.sub(',','',(x.split('$')[1]))) for x in listings.ext

#extracting just the year the member has been with airbnb since
listings.host_since = [int(re.findall('\d{4}', x)[0]) for x in listings.host_since]
#on checking the number of listings each of the missing hosts have, we can see that w
listings = listings.fillna({'host_listings_count':1})
#creating a spatial dataframe from the listing data
listings = pd.DataFrame.spatial.from_xy(listings,y_column = 'longitude', x_column='la

#there are just 24 rows where the following columns are null and we are better of dro
listings = listings.dropna(subset = ['bathrooms', 'bedrooms', 'beds'])
listings = listings.reset_index()
```

In [68]:
```
listings_gpd['metro_dist'] = min_distances
```

```
listing_dist = listings.merge(listings_gpd, on = 'index')
```

```
In [69]: def get_metro_dist(lat,lon):
             pt = Point(lon,lat)
             df = pd.DataFrame([pt], columns = ['Coordinates'])
             df = gpd.GeoDataFrame(df, geometry='Coordinates')
             df.set_geometry('Coordinates')
             df.crs = {'init' :'epsg:4326'}
             df = df.to_crs({'init': 'epsg:2230'})
             return (min(tram_metro_gpd.distance(df.loc[0,'Coordinates'])/5280))
```

**Finding Nearby Attractions**

**Strategy** We started by dividing our attractions into ten sub-categories. Our aim was to get the number of attractions within a 1 km radius of the listing. We used 1-km as that is the average distance someone can walk in 10 minutes. We were able to get the required results using geopandas functions 'buffer' and 'within'. However, this function is not the most time efficient. With the size of our dataset and the computation power on hand, it made sense to avoid this function. Instead, we chose to create a bounding box around the listing, with a side of length 1 km.

```
In [50]: # Creating points from the latitude and longitude coordinates
         attractions.lon = pd.to_numeric(attractions.lon, errors= 'coerce')
         attractions.lat = pd.to_numeric(attractions.lat, errors= 'coerce')
         attractions['Coordinates'] = list(zip((attractions.lon),(attractions.lat)))
         attractions['Coordinates'] = attractions['Coordinates'].apply(Point)
         attractions = attractions.drop(columns = ['SHAPE'])
```

```
In [51]: # Creating subcategory dataframes
         nature_pd = attractions.loc[attractions.CAT.isin(nature)]
         nightlife_pd = attractions.loc[attractions.CAT.isin(nightlife)]
         sports_pd = attractions.loc[attractions.CAT.isin(sports)]
         religion_pd = attractions.loc[attractions.CAT.isin(religion)]
         adventure_pd = attractions.loc[attractions.CAT.isin(adventure)]
         history_pd = attractions.loc[attractions.CAT.isin(history)]
         arts_pd = attractions.loc[attractions.CAT.isin(arts)]
         shopping_pd = attractions.loc[attractions.CAT.isin(shopping)]
         food_pd = attractions.loc[attractions.CAT.isin(food)]
         monuments_pd = attractions.loc[attractions.CAT.isin(monuments)]
```

```
In [52]: # Creating GeoDataFrames for each subcategory
         nature_gpd = gpd.GeoDataFrame(nature_pd, geometry='Coordinates')
         nightlife_gpd = gpd.GeoDataFrame(nightlife_pd, geometry='Coordinates')
         sports_gpd = gpd.GeoDataFrame(sports_pd, geometry='Coordinates')
         religion_gpd = gpd.GeoDataFrame(religion_pd, geometry='Coordinates')
         adventure_gpd = gpd.GeoDataFrame(adventure_pd, geometry='Coordinates')
         history_gpd = gpd.GeoDataFrame(history_pd, geometry='Coordinates')
         arts_gpd = gpd.GeoDataFrame(arts_pd, geometry='Coordinates')
         shopping_gpd = gpd.GeoDataFrame(shopping_pd, geometry='Coordinates')
```

```
food_gpd = gpd.GeoDataFrame(food_pd, geometry='Coordinates')
monuments_gpd = gpd.GeoDataFrame(monuments_pd, geometry='Coordinates')
```

In [54]:
```
# Setting geometry to the coordinate column for each subcategory GDF
nature_gpd.set_geometry('Coordinates')
nightlife_gpd.set_geometry('Coordinates')
sports_gpd.set_geometry('Coordinates')
religion_gpd.set_geometry('Coordinates')
adventure_gpd.set_geometry('Coordinates')
history_gpd.set_geometry('Coordinates')
arts_gpd.set_geometry('Coordinates')
shopping_gpd.set_geometry('Coordinates')
food_gpd.set_geometry('Coordinates')
monuments_gpd.set_geometry('Coordinates')
print('Done')
```

```
Done
```

In [55]:
```
# Setting the crs for each subcategory GDF
nature_gpd.crs = {'init' :'epsg:4326'}
nightlife_gpd.crs = {'init' :'epsg:4326'}
sports_gpd.crs = {'init' :'epsg:4326'}
religion_gpd.crs = {'init' :'epsg:4326'}
adventure_gpd.crs = {'init' :'epsg:4326'}
history_gpd.crs = {'init' :'epsg:4326'}
arts_gpd.crs = {'init' :'epsg:4326'}
shopping_gpd.crs = {'init' :'epsg:4326'}
food_gpd.crs = {'init' :'epsg:4326'}
monuments_gpd.crs = {'init' :'epsg:4326'}
```

In [56]:
```
# Converting crs and reseting index for each subcategory GDF
nature_gpd = nature_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
nightlife_gpd = nightlife_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
sports_gpd = sports_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
religion_gpd = religion_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
adventure_gpd = adventure_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
history_gpd = history_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
arts_gpd = arts_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
shopping_gpd = shopping_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
food_gpd = food_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
monuments_gpd = monuments_gpd.to_crs({'init': 'epsg:2230'}).reset_index()
```

In [70]:
```
# Get number of subcategory datapoints within a bounding box of each listing
def within_radius(df, listings):
    w_radius = []
    for x in listings.index:
        lat = (listings.loc[x,'Coordinates'].x)
        lon = (listings.loc[x,'Coordinates'].y)
```

```
            max_lat = lat+1584
            min_lat = lat-1584
            max_lon = lon+1584
            min_lon = lon-1584
            w_radius.append((df.loc[(df.x >= min_lat) & (df.x <= max_lat) & (df.y >= min_
        return w_radius
```

In [71]:
```
# Getting number of attractions within a 1km distance from the listing per category.
nature_gpd['x'] = nature_gpd['Coordinates'].x
nature_gpd['y'] = nature_gpd['Coordinates'].y
nature_radius = within_radius(nature_gpd,listing_dist)

nightlife_gpd['x'] = nightlife_gpd['Coordinates'].x
nightlife_gpd['y'] = nightlife_gpd['Coordinates'].y
night_radius = within_radius(nightlife_gpd,listing_dist)

sports_gpd['x'] = sports_gpd['Coordinates'].x
sports_gpd['y'] = sports_gpd['Coordinates'].y
sports_radius = within_radius(sports_gpd,listing_dist)

religion_gpd['x'] = religion_gpd['Coordinates'].x
religion_gpd['y'] = religion_gpd['Coordinates'].y
religion_radius = within_radius(religion_gpd,listing_dist)

adventure_gpd['x'] = adventure_gpd['Coordinates'].x
adventure_gpd['y'] = adventure_gpd['Coordinates'].y
adventure_radius = within_radius(adventure_gpd,listing_dist)

history_gpd['x'] = history_gpd['Coordinates'].x
history_gpd['y'] = history_gpd['Coordinates'].y
history_radius = within_radius(history_gpd,listing_dist)

arts_gpd['x'] = arts_gpd['Coordinates'].x
arts_gpd['y'] = arts_gpd['Coordinates'].y
arts_radius = within_radius(arts_gpd,listing_dist)

shopping_gpd['x'] = shopping_gpd['Coordinates'].x
shopping_gpd['y'] = shopping_gpd['Coordinates'].y
shopping_radius = within_radius(shopping_gpd,listing_dist)

food_gpd['x'] = food_gpd['Coordinates'].x
food_gpd['y'] = food_gpd['Coordinates'].y
food_radius = within_radius(food_gpd,listing_dist)

monuments_gpd['x'] = monuments_gpd['Coordinates'].x
monuments_gpd['y'] = monuments_gpd['Coordinates'].y
monuments_radius = within_radius(monuments_gpd,listing_dist)
```

```
listing_dist['nature'] = nature_radius
listing_dist['nightlife'] = night_radius
listing_dist['sports'] = sports_radius
listing_dist['religion'] = religion_radius
listing_dist['adventure'] = adventure_radius
listing_dist['history'] = history_radius
listing_dist['arts'] = arts_radius
listing_dist['shopping'] = shopping_radius
listing_dist['food'] = food_radius
listing_dist['monuments'] = monuments_radius

# Function to remap scores from 0 to 100
def remap(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

Initial GeoScore calculated using the following weights: Nature = 10% Nightlife = 10% Sports = 10% Religion = 10% Adventure = 10% History = 10% Arts = 10% Shopping = 10% Food = 10% Monuments = 10% Metro distance = -10% (as greater distances are worse) We sum each weight to calculate the final score.

Next, we used a rescaling function to convert our score to a value between 0 and 100.

```
In [72]: # Creating a score and then remapping from 0 to 100
         listing_dist['Score'] = (listing_dist.nature*0.10) + (listing_dist.nightlife*0.10) +
         min_score = listing_dist.Score.min()
         max_score = listing_dist.Score.max()
         o_max = 100
         o_min = 0
         listing_dist.Score = remap(listing_dist.Score, min_score,max_score,o_min,o_max)
```

# 7 Creating the Model for Tourists

We then created a function where you could pass in your room requirements along with your favourite attraction sub-category. We would use this to create a new attribute GeoScore.

The score is computed using the following formula:

Tourists favourite type of attractions = 60% Each remaining attraction = 5% Metro distance = -5%

We then remap that score from 0 to 100.

We then queried for the room requirements and sorted our output by the attribute geoScore.

```
In [73]: # Creating a new attribute score and sorting based on that score. Returns at max 20 p
         def get_best(bed,bath, accommodates, reason_for_visit,max_price = 1000000 ,property_t
             cats = ['nature','nightlife','sports','religion','adventure', 'history','arts','s
             cats.remove(reason_for_visit)
             score = listing_dist[reason_for_visit] *0.60
             for x in cats:
                 score += listing_dist[x] * 0.05
             score -= (listing_dist['metro_dist'] *0.05)
             listing_dist['attr_score'] = remap(score, score.min(),score.max(), 0,100)
```

```
        filtered = listing_dist.loc[(listing_dist.bedrooms == bed) & (listing_dist.bathro
        sorted_df = (filtered.sort_values(['attr_score','Score'], ascending = False))[:mi
        return sorted_df
```

## 8 Creating the Model for Renters

For this part, we used some basic information about the listing, like the number of bedrooms, bathrooms, if the host is a superhost along with our features we created to create a predictive model. Our model predicted the price of a listing using this information. We ran our features through a K-Nearest Neighbour regressor to approximate a price. The price predicted depends completely on the training data. Hence, if all the other similar properties are overpriced, we will predict a high price as well.

```
In [80]: # Preparing pipeline
         enc = OneHotEncoder(handle_unknown='ignore')
         enc.fit(listing_dist[['host_is_superhost','property_type','room_type']])
         df = pd.DataFrame(enc.transform(listing_dist[['host_is_superhost','property_type','ro

         x = df.merge(listing_dist[['nature', 'nightlife', 'sports', 'religion', 'adventure',
                      'shopping', 'food', 'monuments', 'Score', 'accommodates', 'bathrooms',
                                  'beds','metro_dist']], left_index= True, right_index = Tru
         y = listing_dist[['price']]

         pl = Pipeline(steps=[('regressor', KNeighborsClassifier(3))])
         X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20)

         pl.fit(X_train,y_train)
         preds = pl.predict(X_test)

         y_test['preds'] = preds
         y_test['diff'] = y_test.price - y_test.preds
         y_test['close'] = [1 if abs(x) < 10 else 0 for x in y_test['diff'].values]

/opt/conda/lib/python3.6/site-packages/sklearn/pipeline.py:267: DataConversionWarning: A colum
  self._final_estimator.fit(Xt, y, **fit_params)
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  from ipykernel import kernelapp as app
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  app.launch_new_instance()
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

In [81]:
```
# Creating predictive model
def get_pred(lat,lon,accommodates,bathrooms,bedrooms,beds,host_is_superhost,property_t
    metro_dist = get_metro_dist(lat,lon)
    df = pd.DataFrame([lat,lon]).T
    df.columns = ['lat', 'lon']
    df['Coordinates'] = Point(lon,lat)
    df = gpd.GeoDataFrame(df, geometry='Coordinates')
    df.set_geometry('Coordinates')
    df.crs = {'init' :'epsg:4326'}
    df = df.to_crs({'init': 'epsg:2230'})
    nature_radius = within_radius(nature_gpd,df)
    night_radius = within_radius(nightlife_gpd,df)
    sports_radius = within_radius(sports_gpd,df)
    religion_radius = within_radius(religion_gpd,df)
    adventure_radius = within_radius(adventure_gpd,df)
    history_radius = within_radius(history_gpd,df)
    arts_radius = within_radius(arts_gpd,df)
    shopping_radius = within_radius(shopping_gpd,df)
    food_radius = within_radius(food_gpd,df)
    monuments_radius = within_radius(monuments_gpd,df)
    df['metro_dist'] = metro_dist
    df['nature'] = nature_radius
    df['nightlife'] = night_radius
    df['sports'] = sports_radius
    df['religion'] = religion_radius
    df['adventure'] = adventure_radius
    df['history'] = history_radius
    df['arts'] = arts_radius
    df['shopping'] = shopping_radius
    df['monuments'] = monuments_radius
    df['food'] = food_radius
    df['Score'] = (df.nature*0.10) + (df.nightlife*0.10) + (df.sports*0.10) + (df.reli
    df['Score'] = remap(df.Score, min_score,max_score,o_min,o_max)
    df['bathrooms'] = bathrooms
    df['bedrooms'] = bedrooms
    df['beds'] = beds
    df['accommodates'] = accommodates
    df['host_is_superhost'] = host_is_superhost
    df['property_type'] = property_type
    df['room_type'] = room_type
    df2 = pd.DataFrame(enc.transform(df[['host_is_superhost','property_type','room_typ
    x = df2.merge(df[['nature', 'nightlife', 'sports', 'religion', 'adventure', 'histo
```

```
                        'shopping', 'food', 'monuments', 'Score', 'accommodates', 'bathrooms',
                                    'beds','metro_dist']], left_index= True, right_index = Tru
        return pl.predict(x)[0],x
```

# 9 Summary of Products and Results

**Tourist Model** We tested the results of our model on the following case: 6 friends are traveling to Amsterdam, primarily for the nightlife. They are looking for a 3 bedroom, 2 bathroom Airbnb. Their initial budget is \$500. From our initial EDA, we know that Central-West is the neighbourhood with the best nightlife. The average 3 bedroom is Central-West is $351 and the average 2 bathroom is \$307.

```
In [152]: # Generating recommendations for max prices of $500 vs. $150
          nightlife_recs = get_best(bed = 3,bath=2, accommodates = 6,max_price = 500, reason_fo
          nightlife_recs2 = get_best(bed = 3,bath=2, accommodates = 6,max_price = 150, reason_i

          recs_sdf = pd.DataFrame.spatial.from_xy(nightlife_recs,y_column = 'latitude_y', x_col
          recs2_sdf = pd.DataFrame.spatial.from_xy(nightlife_recs2,y_column = 'latitude_y', x_c
          nightlife_sdf = pd.DataFrame.spatial.from_xy(nightlife_gpd,y_column = 'lat', x_column
```

```
In [153]: # Copying to a DataFrame and displaying recommendations
          shape = list()
          for i in recs_sdf.index:
              point = Point(recs_sdf['latitude_x'][i], recs_sdf['longitude_x'][i])
              shape.append(point)
          recs_sdf['Coordinates'] = shape
          recs_sdf = recs_sdf[['Coordinates', 'neighbourhood', 'property_type', 'Score']]

          shape = list()
          for i in recs2_sdf.index:
              point = Point(recs2_sdf['latitude_x'][i], recs2_sdf['longitude_x'][i])
              shape.append(point)
          recs2_sdf['Coordinates'] = shape
          recs2_sdf = recs2_sdf[['Coordinates', 'neighbourhood', 'property_type', 'Score']]

          print('Max Price of $500:')
          recs_sdf
```

```
Max Price of $500:
```

```
Out[153]:                                     Coordinates            neighbourhood  \
          6956  POINT (52.36422920424003 4.983889112866038)                     Oost
          9789  POINT (52.35754999994857 4.938719918056048)           Watergraafsmeer
          7357   POINT (52.38267421812507 4.90632710231541)      IJplein en Vogelbuurt
          2017  POINT (52.37116815664309 4.909714390196399)      Nieuwmarkt en Lastage
          6686  POINT (52.36839856918552 4.911717244792341)   Weesperbuurt en Plantage
          1384   POINT (52.36168912723696 4.91276522748362)   Weesperbuurt en Plantage
```

```
7090  POINT (52.36768684860621 4.906020737187752)     Nieuwmarkt en Lastage
6274  POINT (52.35703598259818 4.909891283651533)          Oosterparkbuurt
692   POINT (52.35658906379828 4.907892427485184)                      NaN
2523  POINT (52.38444171445879 4.885019791797347)      Westelijke Eilanden

     property_type      Score
6956         House  72.484716
9789     Townhouse  56.069109
7357          Boat  52.254004
2017     Townhouse  50.291153
6686     Townhouse  50.203539
1384     Apartment  48.758448
7090     Apartment  48.168036
6274         House  46.589282
692      Townhouse  45.822593
2523     Apartment  45.818662
```

Above is the map our model recommends with the given conditions. The highlighted neighbourhood is Central-West. We can see that all our recommendations are either in Central-West or on the border, as expected.

```
In [154]: print('Max Price of $150:')
          recs2_sdf
```

```
Max Price of $150:
```

```
Out[154]:                                        Coordinates    neighbourhood  \
          6956  POINT (52.36422920424003 4.983889112866038)             Oost
          9789  POINT (52.35754999994857 4.938719918056048)  Watergraafsmeer
          3421  POINT (52.36668506691566 4.897066186546887)   Grachtengordel

               property_type      Score
          6956         House  72.484716
          9789     Townhouse  56.069109
          3421     Apartment  44.999470
```

This time, our model only returned 3 listings. This makes sense due to the specificity of our search. On checking the average price for a 3 bedroom, 2 bathroom, we learnt that on average it goes for \$284. Hence, our model has generated the expected results.

**Renter Model** Let's say we have 3 bedroom, 2 bathroom property in Centrum-West that we would like to list on Airbnb.

```
In [155]: price, table = get_pred(lat=52.373068, lon=4.899326,accommodates=6,bathrooms=2,bedro
          price
```

```
Out[155]: 85.0
```

```
In [156]: table.loc[:,'nature':]

Out[156]:    nature  nightlife  sports  religion  adventure  history  arts  shopping  \
          0       0         14       0         7          0        3     0         2

             food  monuments         Score  accommodates  bathrooms  bedrooms  beds  \
          0     0          3  1.594726e+10             6          2         3     3

             metro_dist
          0    0.123055
```

We are not focussing on any particular tourist attraction category here and hence none of them are weighted higher than the others. This listing is priced relatively low for the location. The only attraction category that it does well in is nightlife and distance to the nearest metro. Apart from those two it does not do as well. In the next step of our analysis, we should identify which neighbourhoods are specialised for what tourist attractions and see how that affects the price.

## 10 Discussion

- It was interesting to consider our analysis in comparison to the articles and essays we found that talk about Amsterdam's evolving relationship with tourism and Airbnb. It seems that Airbnb still has a massive presence in the city's tourism industry despite some attempts to limit that. Further, tourism still seems to be very present in the areas surrounding the Red Light District, in spite of some pushes to disperse tourism elsewhere throughout the city. Although the prices are still the highest in the commercial center of the city, there are numerous listings elsewhere in the city that seem to reflect some amount of dispersion of tourism. Ultimately, though, many of these assessments would be difficult to confirm without years worth of data to consider.
- During our project, we tried implementing a buffer around a listing, but we were facing problems with the same. Instead, we decided to use a bounding box. This reduced the accuracy of our findings, but not significantly. Further, when we had the idea for the feature we wanted to consider attractions that were within 10 minutes walking. However, all functions that get distances and buffer, consider the euclidean distance. Further, we tried to use a raster calculator to score each neighbourhood. The NDVI imagery layer did not have the servers to allow raster calculations. Hence, we decided not to use the neighbourhood score.

## 11 Conclusions and Future Work

Completely answering our initial question would take months of research. For starters, we need a much more extensive dataset with the attractions in Amsterdam. We would also benefit if we could get star rating for each tourist attraction. That way we can give more popular attractions a higher weight in our score. Further, the restaurants listed in our attractions was very limited. We would benefit if we used a much larger dataset for this. Perhaps scrape some Yelp data for the same. Further, we could weigh restaurants that serve good local food higher, promoting people to expand their culinary palette.

Secondly, when calculating our geoScore, we subset those attractions that are approximately within 1Km from the listing. Our distance function calculates the euclidean distance, rather than the actual walking distance which reduces the accuracy of our results. Further, bicycles are one of the most common modes of transportation in Amsterdam and we did not consider the same for our model. We could perhaps calculate another buffer for attractions that are within a 10 minute bicycle ride from the listing. Other things that we must consider for our geoScore are the distance to train stations, city center, airports, ATM's, supermarkets. It would interesting considering how the safety, pollution levels and beauty of a neighbourhood would affect our model. Sometimes people would rather commute a longer distance in order to stay in a nice, safe neighbourhood. Factors that could affect the beauty of a neighbourhood could include if the listing overlooks canals, and the amount of greenery.

It would be interesting to run a similar model on other cities in the world and see how Amsterdam is similar and different to these other cities. Further, by using this data, Airbnb will understand customer preferences. For example, they might learn that I choose listings with a highly weighted sports geoScore and in turn they could recommend cities with a strong sport culture that I could visit in the future. Renters could also use this information to set fair prices. This is highly useful information as Airbnb is allegedly responsible for price inflation in parts of the world which drives a lack of housing affordability.