MEZ204 GRADUATION PROJECT

Istanbul Traffic and Venue Distribution Analysis Using Python

06 HAZİRAN 2021

ISTANBUL AYDIN UNIVERSTY
Ahmet Hakan Afşin b1910.033028

Part 1:

Description:
In this project, I have gathered some data from the official website of Istanbul Büyükşehir Belediyesi and made some analysis and manipulation on it to be able to examine the data on an interactive map. Then, I gathered information of the city according to the geospatial coordinates in the data using Foursquare's Web API. I gathered the top 100 venues around those coordinates, took the mode of that data to have some knowledge about the zone structure according to the venue distribution. Having that data lead me to have some opinion about those zones to cluster them logically to identify them as Commercial, Industrial and Green zones. Then, I feed that data into a machine learning algorithm called 'K-Means Clustering' to make the computer cluster them. Then showed them on top of an interactive map again to visualize the data to get some readable information.

Motivation/Purpose:
My main motivation was a project I made for an online course I took this summer. In that project, I used similar tools to visualize the real estate information of Boston. Then, I wanted to make a similar project for Istanbul but not about the real estates, instead the traffic because It's the biggest problem that city has. To be able to visualize the traffic problem, the reasoning behind it and prove it using statistical analysis, I started this project.

Logic and Technologies Used in This Project:

- **1-) Python Programming Language:**

  Python is a Dynamically-Typed, Interpreted, Object-Oriented programming language with a distinct syntax. It's widely used in Data-Science and Machine-Learning projects. It has a wide range of open source libraries for those subjects.
- **2-) Jupyter Lab:**
  Jupyter Lab is a web-based interactive development environment which is again widely used in Data-Science and Machine-Learning projects. It is very flexible and easy to use. It provides such an environment that user can manage code in the runtime. Every instruction line user gives to the computer is run as soon as the user gives it, and user can examine the variables in a great visual environment. I used that tool a lot while working with the dataframes I got to see the changes I made as soon as I made them without being have to give more instructions etc… Also, Jupyter Lab supports html. So In a ipython notebook file, with the visualization of commentation, it becomes to a presentation tool too.

- **3-) Pandas Library:**
  Pandas is an open source Python library which provides DataFrame and Pandas Series structures which are very useful in this subject. I stored the data provided by the Istanbul Büyükşehir Belediyesi in a DataFrame, Created new Pandas Series objects as new columns and added to that DataFrame, removed the unused columns and null rows in that DataFrame using pandas.

- **4-) Matplotlib Library:**
  Matplotlib is the main data visualization tool of many Python programmers and is open-source again. It has three different layers which can be used in different situations: Scripting Layer which is used in basic, theme based visualizations, Artist Layer which provides a lot of visualization options and Backend Layer which handles

the heavy works by communicating the toolkits or drawing languages in the machine. I mostly used another visualization library which is built on top of this called Folium to generate interactive map plottings. But in other visualizations, I mainly used the Scripting layer to create basic visualizations.

- **5-) Folium:**
  Folium again is an open-source Python library which is mainly created for Geographical Plottings which I did a lot in this project. It has a object called Map that gets the coordinates as its location parametes, has a zooming feature. Then, I applied a lambda function that creates a circle marker on that map for each coordinate in the DataFrame.

- **6-) Scikit-Learn:**
  Scikit learn is an open-source Python library that contains a lot of statistical methods and Machine Learning alghorithms in it. I used this tool in my project to apply the K-Means Clustering alghorithm to the data I had.

- **7-) h5py-hdf5 (Hierarchical Data Format):**
  Hierarchical Data Format (HDF5) is a file format designed to store and organize large amounts of data. Its originally developed at National Center for Supercomputing Applications(NCSA) . And h5py is a python library that is built to create hdf5 files and read them. I used this data format to store the result I got after the requests I made to the Foursquare api in this project because my API credentials were limited and could only make a limited number of request in a day. And to be able to work freely, I stored the result Data Frame I created after those requests in a hdf5 file.

- **8-) Branca:**
  Branca is a Python library that used for generating complex HTML and Javascript pages with python. It's the main backend of folium but I only used this library to create a Folium compatible colormap to be able to show the traffic density visually using a cool-warm colormap.

- **9-) Spearman's rho:**
  In statistics, Spearman's rank correlation coefficient is named after Charles Spearman. The Spearman correlation between two variables is equal to Pearson Correlation between rank values of those two wariables, while Pearson's correlation (main correlation method of pandas) assesses a linear relationship, Spearman assesses monotonic relationships.* Both correlation computing methods result a numbering between 1(positively correlated), to -1(negatively correlated). I used this method while creating the final correlation matrix that examines the correlation between the venue types, counts and the number of cars.

  *Source: https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

Part 2

Source Code:

****** To provide the projects source code, I added the notebook in a snippet below. But because pdf files are static and I can not add a html file directly inside that, also I could not convert the html file to a docx file directly, this file has some corrupted parts that are hard to read. I'm including the html file in case you have any difficulties reading.

Istanbul Aydin Universty

Graduation Project

AHMET HAKAN AFSIN

B1910.033028

In [1]:

```
##!pip install pandas,folium,numpy,matplotlib,scikit-learn,scipy,branca,h5py,
```

To be able to re-run this notebook, the necessary libraries can be installed with the commented line above.

Importing The Necessary Libraries

In [2]:

```
import pandas as pd
import folium
import numpy as np
import matplotlib.pyplot as plt
import requests
```

Reading the data provided from the governement's website

Step 1: READING AND CLEANING THE DATA

In [3]:

```
cars_df = pd.read_csv("daily_cars.csv" , encoding="ISO-8859-1",sep=";")
```

In [4]:

```
cars_df.columns = ["date","sensor_name","lon","lat","num_cars"]
```

Setting the column names (They are originally in Turkish)

In [5]:

```
cars_df.head()
```

Out[5]:

	date	sensor_name	lon	lat	num_cars
0	1.01.2020	ciragan Cad.	29,016617	41,044845	86521
1	1.01.2020	Kco cekmekoy Kavsagi	29,19354	41,051371	9451
2	1.01.2020	Gunesli 2 Basin Ekspres yolu	28,811125	41,024099	53991
3	1.01.2020	Buyukdere 1.Levent	29,015483	41,073533	102531
4	1.01.2020	Cevizlibag	28,914282	41,018157	129090

Dataset: This dataset contains the number of cars a particular sensor has detected in a day

Date column contains days

Date column contains days

Sensor name contains the name of the place that a particular sensor is located at

Lon stands for longitude

Lat stands for lattitude

Num_cars contains the number of cars

In [6]:

```
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47398 entries, 0 to 47397
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   date         47398 non-null  object
 1   sensor_name  47398 non-null  object
 2   lon          47101 non-null  object
 3   lat          47101 non-null  object
 4   num_cars     47398 non-null  int64 
dtypes: int64(1), object(4)
memory usage: 1.8+ MB
```

1- From that output , we see that there are missing locations and without that info , those rows are useless. We could fill those rows by searching for the sensor's name on the google and looking for coordinates but there's no need for that

2- lat and lon columns are objects because in turkey , we use ',' as the floating point indicator. We need to replace ',' s with '.' s to convert that column to floats

In [7]:

```
cars_df.dropna(inplace=True)
```

In [8]:

```
cars_df.lat = cars_df.lat.apply(lambda x: str(x).replace(",","."))
```

In [9]:

```
cars_df.lon = cars_df.lon.apply(lambda x: str(x).replace(",","."))
```

In [10]:

```
cars_df.lat = cars_df.lat.astype("float64")
cars_df.lon = cars_df.lon.astype("float64")
```

In [11]:

```
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 47101 entries, 0 to 47397
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   date         47101 non-null  object 
 1   sensor_name  47101 non-null  object 
 2   lon          47101 non-null  float64
 3   lat          47101 non-null  float64
 4   num_cars     47101 non-null  int64  
dtypes: float64(2), int64(1), object(2)
memory usage: 2.2+ MB
```

Now , our 'lat' and 'lon' columns are in the form we want them to be. We need to convert the date column to get the day out of it and

after that , we can get the average number of cars passing per day.

In [12]:

```
cars_df["date"] = pd.to_datetime(cars_df["date"])
```

In [13]:

```
cars_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 47101 entries, 0 to 47397
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   date         47101 non-null  datetime64[ns]
 1   sensor_name  47101 non-null  object        
 2   lon          47101 non-null  float64       
 3   lat          47101 non-null  float64       
 4   num_cars     47101 non-null  int64         
dtypes: datetime64[ns](1), float64(2), int64(1), object(1)
memory usage: 2.2+ MB
```

Now we can extract the day and the month

In [14]:

```
cars_df["day"] = cars_df.date.apply(lambda x: x.day)
```

In [15]:

```
cars_df["month"] = cars_df.date.apply(lambda x: x.month)
```

We don't need the date column anymore so let's drop it to save some memory space

In [16]:

```
cars_df.drop("date" , axis=1 , inplace=True)
```

In [17]:

```
cars_df.head()
```

Out[17]:

	sensor_name	lon	lat	num_cars	day	month
0	ciragan Cad.	29.016617	41.044845	86521	1	1
1	Kco cekmekoy Kavsagi	29.193540	41.051371	9451	1	1
2	Gunesli 2 Basin Ekspres yolu	28.811125	41.024099	53991	1	1
3	Buyukdere 1.Levent	29.015483	41.073533	102531	1	1
4	Cevizlibag	28.914282	41.018157	129090	1	1

Now , we can group our dataset by sensor names to get the average number of cars , It's ok to get the average lat and lon because every row contains the same value for them.

In [18]:

```
gby = cars_df.groupby("sensor_name").mean()[["num_cars","lon","lat"]]
```

In [19]:

```
gby.head()
```

```
gby.head()
```

Out[19]:

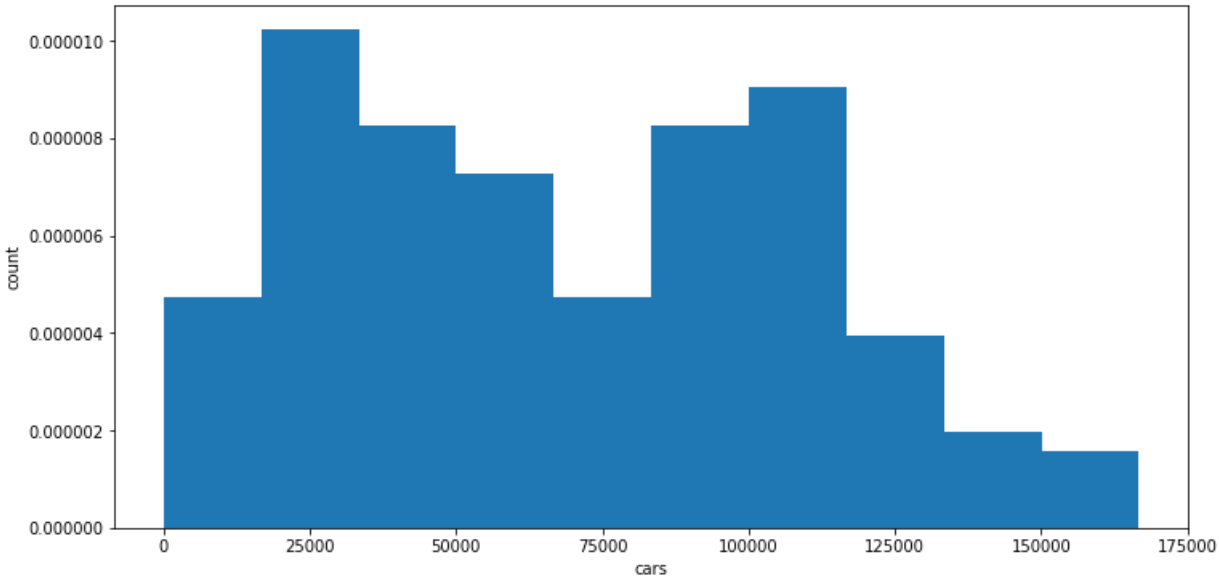
sensor_name	num_cars	lon	lat
Alemdag	11515.674797	29.228200	41.044900
15 Temmuz sehitler Koprusu Anadolu	124149.208054	29.043900	41.035517
15 Temmuz sehitler Koprusu Yildiz Katilimi	104345.873333	29.018432	41.057702
Akom onu	88500.404110	28.961073	41.090599
Alemdag Kavsagi	28806.201550	29.270000	41.028900

STEP 2: BASIC EXAMINATION OF DATA

Let's visualize our data first to have a little insight of what we are facing from the beginning

In [20]:

```
plt.figure(figsize=(12,6))
plt.hist(gby["num_cars"],density=True)
plt.xlabel("cars");
plt.ylabel("count");
```



From the plot above, we can see that most of the censors recorded 25000 to 75000 cars per day, but because there is no normal distrubution, we also have a density between 75000 cars and 125000 cars per day. Even thought those are huge numbers too, our main tagret in this project is the points that have recorded more than 100 000 cars per day. This project will mainly examine those areas and try to figure out what are the main reasons behing this huge numbers.

Note: We haven't cleaned the outliers like the censors on the sea. But because the visualization above is generalizing, its ok to not have those outliers cleaned yet.

STEP 3: VISUALIZATION ON A MAP

Now, we will try to use the coordinates provided in the file to plot our censors on a map to be able to see the distrubution of those censors around the city.

In [21]:

```
gmap = folium.Map(location = (gby.lat[0],gby.lat[1]))
```

In [22]:

```
map_osm = folium.Map(location=[gby["lat"][0],gby["lon"][0]], zoom_start=10)
gby.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row["num_cars"]
,
                                         radius=10)
                                        .add_to(map_osm), axis=1)
map_osm
```

Out[22]:

Make this Notebook Trusted to load map: File -> Trust Notebook

It's almost perfect. We need to get rid of the sensor on the sea because we can't get any information about them from Foursquare but its not necessary now, we will do it later.

Now , we're going to divide our number of cars data to clusters so we can visualize them easily (I coldn't find any way to visualize contunious values with folium)

STEP 4: NORMALIZING OUR DATA AND TURNING THE CONTINIOUS NUM CARS VARIABLE TO DISCRIMINATED DENSITY VALUE.

In [23]:

```
import branca

import branca.colormap as cmap

colormap = cmap.LinearColormap(colors=['green',"yellow",'red'],vmin=0,vmax=175000)
```

we created a colormap to visualize the number of cars in our plot

Note: There was a normalization step where I added dummy variables for representing the density of the area but later, I decided to use the number of cars on its own. That's why there is almost no normalization under that topic.

In [24]:

```
gby.head()
```

Out[24]:

sensor_name	num_cars	lon	lat
Alemdag	11515.674797	29.228200	41.044900
15 Temmuz sehitler Koprusu Anadolu	124149.208054	29.043900	41.035517
15 Temmuz sehitler Koprusu Yildiz Katilimi	104345.873333	29.018432	41.057702
Akom onu	88500.404110	28.961073	41.090599
Alemdag Kavsagi	28806.201550	29.270000	41.028900

Perfect , we now have a density column that describes amount traffic.

Now , we need to create a dictionary that contains a color for every density value we have

In [25]:

```
map_osm = folium.Map(location=[gby["lat"][0],gby["lon"][0]], zoom_start=10)
gby.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row.name , colo
r = colormap(row.num_cars) ,  fill_color =colormap(row.num_cars),
                                            radius=10)
                                           .add_to(map_osm), axis=1)
map_osm
```

Out[25]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looks great , now we can get some data from foursquare to see the relationship between traffic and venues

STEP 5: GETTING THE VENUE INFORMATION FROM FOURSQUARE API

In [26]:

```
client_id = "EAK1OOZAKEURNDYJR0PLZW0E3UK23J5NUCSZP4Y03XZ14LK4"
client_secret = "DCESO2AV0L0CMNDACYKCV3E3MRNKVUPOYTJUMFANW1MG5PAW"
version = "20200710"
```

Those are my credentials for the foursquare api

In [27]:

```
top100 = pd.DataFrame()
for row in gby.iterrows():
    cx = []
    url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}
&radius={}&limit={}'.format(client_id, client_secret, row[1].lat , row[1].lon , version , 500,100)
    responses = requests.get(url).json()["response"]["venues"][0:]
    for response in responses:
        try:
            cx.append(response["categories"][0]["name"])
        except:
            cx.append(np.nan)
    while len(cx) < 100:
        cx.append(np.nan)
    top100[row[0]] = [i for i in cx]
```

This process takes a while because the code above is not so efficient at all. Because of my connection speed it takes around 3-5 minutes.
I will try to improve it further later.

In [38]:

```
top100
```

Out[38]:

	Alemdag	15 Temmuz sehitler Koprusu Anadolu	15 Temmuz sehitler Koprusu Yildiz Katilimi	Akom onu	Alemdag Kavsagi	Alibeykoy TEM Katilimi	Altunizade	Altunizade umraniye Katilimi	Anadolu Feneri	Anadolu Feneri ust Gecit
0	Farm	Bridge	Bridge	Bridge	Coworking Space	Trail	Mini Golf	Karaoke Bar	Cruise Ship	Cruise Ship
1	Soccer Stadium	Scenic Lookout	Rest Area	Other Nightlife	College Residence Hall	Housing Development	Office	Island	Lighthouse	Transportation Service
2	Bridge	Social Club	General Entertainment	Seafood Restaurant	College Engineering Building	Residential Building (Apartment / Condo)	Bridge	Residential Building (Apartment / Condo)	Restaurant	Coworking Space
3	Shopping Plaza	Arcade	Assisted Living	Shopping Mall	Park	Bridge	Bridge	Bridge	Surf Spot	Beach
4	Residential Building (Apartment / Condo)	Bus Stop	Building	Bar	Summer Camp	Park	Newsstand	Bridge	Recreation Center	Restaurant
...	...	...	...	...	...	...	...	...	...	...
95	Housing Development	Coworking Space	Event Space	Residential Building (Apartment / Condo)	College Residence Hall	Advertising Agency	Coworking Space	Business Center	Garden	Farm
96	Factory	Photography Lab	Miscellaneous Shop	Home Service	College Classroom	Café	Furniture / Home Store	Coworking Space	Lighthouse	Farm
97	Factory	Cemetery	Coworking Space	Adult Boutique	Housing Development	Office	Art Gallery	Building	NaN	NaN
98	Building	Auto Dealership	Tech Startup	Coworking Space	Daycare	Assisted Living	Office	Business Center	NaN	Electronics Store
99	Factory	Tech Startup	Office	NaN	Forest	Residential Building (Apartment / Condo)	College Arts Building	Housing Development	Electronics Store	Steakhouse

100 rows × 305 columns

In [29]:

```
!pip install tables
```

```
Requirement already satisfied: tables in c:\users\artemis\.conda\envs\cpu_env\lib\site-packages
(3.6.1)
Requirement already satisfied: numexpr>=2.6.2 in c:\users\artemis\.conda\envs\cpu_env\lib\site-
packages (from tables) (2.7.3)
Requirement already satisfied: numpy>=1.9.3 in c:\users\artemis\.conda\envs\cpu_env\lib\site-
packages (from tables) (1.18.1)
```

Saving the data gathered from the foursquare in case our daily API call limit ends.

In [30]:

```
h5file = "top100.h5"
top100.to_hdf(h5file,"/data/top100")
```

```
C:\Users\Artemis\.conda\envs\cpu_env\lib\site-packages\pandas\core\generic.py:2505:
PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block1_values] [items->Index([' Alemdag', '15 T
emmuz sehitler Koprusu Anadolu',
       '15 Temmuz sehitler Koprusu Yildiz Katilimi', 'Akom onu',
       'Alemdag Kavsagi', 'Alibeykoy TEM Katilimi', 'Altunizade',
       'Altunizade umraniye Katilimi', 'Anadolu Feneri',
       'Anadolu Feneri ust Gecit',
       ...
       'umraniye Kavsagi', 'umraniye Kucuksu', 'umraniye Otopazari',
       'uskudar Sahil Yolu', 'Ýhsaniye Kavsagi', 'Ýkitelli Basin Ekspres yolu',
       'Ýshakli', 'Ýstac Kati Atik Tesisi', 'Ýstanbul Havalimani cikis',
       'Ýstoc'],
      dtype='object', length=304)]

  encoding=encoding,
```

Commented Line below reads the hdf5 file from the folder

In [31]:

```
## top100 = pd.read_hdf(top100.h5,"/data/top100")
```

Perfect. Now we have the first 100 venue types we get from the foursquare api around those censors.
Now we can get the most frequent venues around those censors by easily using the mode function

STEP 6: GETTING THE MOST COMMON VENUE TYPE FROM THE DATA EXTRACTED

In [32]:

```
most_freq = top100.mode()
```

In [33]:

```
most_freq.head()
```

Out[33]:

	Alemdag	15 Temmuz sehitler Koprusu Anadolu	15 Temmuz sehitler Koprusu Yildiz Katilimi	Akom onu	Alemdag Kavsagi	Alibeykoy TEM Katilimi	Altunizade	Altunizade umraniye Katilimi	Anadolu Feneri	Anadolu Feneri ust Gecit	...	umraniye Kavsagi	umraniye Kucuksu
0	Factory	Coworking Space	Office	Residential Building (Apartment	College Residence	Residential Building (Apartment	Office	Office	Farm	Farm	...	Shopping Mall	Office

	Alemdag	15 Temmuz sehitler Koprusu Anadolu	15 Temmuz sehitler Koprusu Yildiz Katilimi	Akom onu	Alemdag Kavsagi	Alibeykoy TEM Katilimi	Altunizade	Altunizade umraniye Katilimi	Anadolu Feneri	Anadolu Feneri ust Gecit	...	umraniye Kavsagi	umraniye Kucuksu
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

5 rows × 305 columns

It's done. We will get rid of those null values and transpose our data frame now.

In [34]:

```
most_freq = most_freq.T
```

In [35]:

```
most_freq = pd.DataFrame(most_freq[0])
```

In [36]:

```
most_freq.columns = ["most_freq"]
```

In [37]:

```
most_freq
```

Out[37]:

	most_freq
Alemdag	Factory
15 Temmuz sehitler Koprusu Anadolu	Coworking Space
15 Temmuz sehitler Koprusu Yildiz Katilimi	Office
Akom onu	Residential Building (Apartment / Condo)
Alemdag Kavsagi	College Residence Hall
...	...
Ýkitelli Basin Ekspres yolu	Factory
Ýshakli	Farm
Ýstac Kati Atik Tesisi	Coworking Space
Ýstanbul Havalimani cikis	Airport Service
Ýstoc	Factory

305 rows × 1 columns

Perfect. Now we can merge our two datasets to one

STEP 7: MERGING THE TWO DATAFRAMES INTO ONE FINAL DATAFRAME

In [39]:

```
final_df = pd.merge(gby , most_freq , left_on = gby.index , right_on = most_freq.index)
```

In [40]:

```
final_df.head()
```

Out[40]:

	key_0	num_cars	lon	lat	most_freq
0	Alemdag	11515.674797	29.228200	41.044900	Factory
1	15 Temmuz sehitler Koprusu Anadolu	124149.208054	29.043900	41.035517	Coworking Space
2	15 Temmuz sehitler Koprusu Yildiz Katilimi	104345.873333	29.018432	41.057702	Office
3	Akom onu	88500.404110	28.961073	41.090599	Residential Building (Apartment / Condo)
4	Alemdag Kavsagi	28806.201550	29.270000	41.028900	College Residence Hall

In [41]:

```
final_df.columns = ["sensor_name","num_cars","lon","lat","most_freq"]
```

Now , lets create a map that popups the most common type of venue around when user clicks

In [42]:

```
map_osm = folium.Map(location=[gby["lat"][0],gby["lon"][0]], zoom_start=10)
final_df.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row["most_
freq"] , color = colormap(row.num_cars) ,  fill_color = colormap(row.num_cars),
                                            radius=10)
                                           .add_to(map_osm), axis=1)
map_osm
```

Out[42]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Perfect. Now , lets cluster our sensors into central places

STEP 8: CLUSTERING THE CENSORS TO TREAT THEM AS AREAS

In [43]:

```
lat_lons = final_df[["lat","lon"]]
```

In [44]:

```
lat_lons
```

Out[44]:

	lat	lon
0	41.044900	29.228200
1	41.035517	29.043900
2	41.057702	29.018432
3	41.090599	28.961073
4	41.028900	29.270000
...	...	...
300	41.056482	28.810554
301	41.130788	29.286221
302	41.203025	28.856191
303	41.246301	28.737081
304	41.067900	28.814800

305 rows × 2 columns

In [45]:

```
from sklearn.cluster import KMeans
```

In [46]:

```
model = KMeans(n_clusters=8)
```

8 clusters would be ok.

In [47]:

```
model.fit(lat_lons)
```

Out[47]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [48]:

```
model.predict(lat_lons)
```

Out[48]:

```
array([4, 1, 7, 7, 4, 7, 1, 1, 4, 4, 7, 2, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 7, 0, 7, 3, 2, 6, 3, 7, 7, 7, 7, 7, 7,
       1, 1, 1, 3, 0, 1, 1, 0, 7, 0, 1, 5, 5, 7, 7, 5, 5, 1, 1, 0, 0, 1,
       1, 7, 0, 7, 5, 5, 0, 0, 3, 7, 0, 3, 5, 1, 1, 7, 1, 0, 0, 1, 3, 3,
       2, 4, 2, 7, 7, 5, 1, 0, 3, 2, 2, 6, 6, 2, 6, 6, 6, 2, 0, 0, 0, 7,
       3, 3, 7, 2, 0, 0, 0, 0, 0, 4, 4, 1, 7, 5, 5, 5, 5, 5, 5, 4, 2, 1,
       4, 4, 7, 7, 7, 6, 1, 6, 6, 1, 6, 1, 1, 1, 7, 2, 0, 7, 1, 0, 2, 2,
       7, 7, 0, 7, 3, 1, 1, 1, 4, 7, 0, 7, 0, 0, 6, 6, 7, 7, 1, 7, 0, 0,
       5, 5, 6, 7, 2, 4, 4, 4, 4, 4, 2, 2, 5, 0, 7, 7, 7, 7, 7, 7, 7, 4,
       5, 0, 7, 1, 1, 5, 1, 3, 0, 3, 3, 3, 3, 1, 3, 2, 0, 7, 7, 7, 0, 7,
       7, 7, 7, 2, 5, 5, 1, 1, 1, 5, 2, 0, 0, 0, 0, 0, 5, 5, 5, 5, 3, 5,
       5, 5, 5, 0, 0, 3, 1, 1, 1, 0, 3, 3, 7, 4, 7, 7, 5, 7, 7, 2, 2, 2,
       7, 0, 6, 2, 5, 2, 7, 0, 0, 1, 0, 0, 7, 7, 7, 7, 7, 1, 1, 1, 1, 4,
       6, 7, 6, 6, 4, 1, 1, 1, 1, 1, 1, 1, 1, 6, 0, 4, 6, 6, 0])
```

Now we have the model generated cluster numbers for each censor.

In [49]:

```
lat_lons["cluster"] = model.predict(lat_lons)
```

```
C:\Users\Artemis\.conda\envs\cpu_env\lib\site-packages\ipykernel_launcher.py:1:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [50]:

```
lat_lons
```

Out[50]:

	lat	lon	cluster
0	41.044900	29.228200	4
1	41.035517	29.043900	1
2	41.057702	29.018432	7
3	41.090599	28.961073	7
4	41.028900	29.270000	4
...	...	...	...
300	41.056482	28.810554	0
301	41.130788	29.286221	4
302	41.203025	28.856191	6
303	41.246301	28.737081	6
304	41.067900	28.814800	0

305 rows × 3 columns

Great. Now we can add cluster column to final_df

In [51]:

```
final_df = pd.concat([final_df , lat_lons["cluster"]],axis=1)
```

In [52]:

```
final_df.head()
```

Out[52]:

	sensor_name	num_cars	lon	lat	most_freq	cluster
0	Alemdag	11515.674797	29.228200	41.044900	Factory	4
1	15 Temmuz sehitler Koprusu Anadolu	124149.208054	29.043900	41.035517	Coworking Space	1
2	15 Temmuz sehitler Koprusu Yildiz Katilimi	104345.873333	29.018432	41.057702	Office	7
3	Akom onu	88500.404110	28.961073	41.090599	Residential Building (Apartment / Condo)	7
4	Alemdag Kavsagi	28806.201550	29.270000	41.028900	College Residence Hall	4

Now let's create a dictionary for cluster colors just like we did before. We are going to pick a color for every unique cluster we have

In [53]:

```
final_df.cluster.unique()
```

Out[53]:

```
array([4, 1, 7, 2, 0, 3, 6, 5])
```

In [54]:

```
cluster_colors = {0:"Blue" , 1: "Green" , 2: "Yellow" , 3: "Red" , 4: "Purple" , 5: "Orange" , 6: "P
ink" , 7: "White" }
```

Let's visualize ,

In [55]:

```
map_osm = folium.Map(location=[final_df["lat"][0],final_df["lon"][0]], zoom_start=10)
final_df.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row["senso
r_name"] , color = colormap(row.num_cars) ,  fill_color = colormap(row.num_cars),
                                              radius=10)
                                             .add_to(map_osm), axis=1)
map_osm
```

Out[55]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Seems working well

In [56]:

```
final_df.head()
```

Out[56]:

	sensor_name	num_cars	lon	lat	most_freq	cluster
0	Alemdag	11515.674797	29.228200	41.044900	Factory	4
1	15 Temmuz sehitler Koprusu Anadolu	124149.208054	29.043900	41.035517	Coworking Space	1
2	15 Temmuz sehitler Koprusu Yildiz Katilimi	104345.873333	29.018432	41.057702	Office	7
3	Akom onu	88500.404110	28.961073	41.090599	Residential Building (Apartment / Condo)	7
4	Alemdag Kavsagi	28806.201550	29.270000	41.028900	College Residence Hall	4

	sensor_name	num_cars	lon	lat	most_freq	cluster
4	Alemdag Kavsagi	28600.201550	29.270000	41.026900	College Residence Hall	4

Let's get rid of the points that are on the sea

STEP 9: CLEANING THE OUTLIERS

In [57]:

```
on_sea = [final_df[(final_df["sensor_name"] == "TEM Kartal") |( final_df["sensor_name"] == "Kasimpa
sa Tunel")]]
```

In [58]:

```
on_sea[0]
```

Out[58]:

	sensor_name	num_cars	lon	lat	most_freq	cluster
129	Kasimpasa Tunel	0.000000	29.457800	41.458700	NaN	4
224	TEM Kartal	98028.890845	29.152771	40.777727	Boat or Ferry	5

NOTE: I don't understand why there are 98 000 cars passing through sea. I don't know where is the TEM Kartal or what it is. But because its most probably a outlier, I decided to get rid of it

In [59]:

```
final_df = final_df.drop(on_sea[0].index , axis=0)
```

In [60]:

```
map_osm = folium.Map(location=[final_df["lat"][0],final_df["lon"][0]], zoom_start=10)
final_df.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row["senso
r_name"] , color = cluster_colors[row["cluster"]] ,
                                              fill_color = colormap(row.num_cars),
                                              fill_opacity = 0.8,
                                              radius=10)
                                             .add_to(map_osm), axis=1)
map_osm
```

Out[60]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looks like it worked.

Now , we can cluster our points again to get the place clusters without points on the sea

In [61]:

```
model.fit(final_df[["lat","lon"]])
```

Out[61]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=8, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [62]:

```
model.predict(final_df[["lat","lon"]])
```

Out[62]:

```
array([1, 7, 2, 2, 1, 2, 7, 7, 5, 5, 2, 4, 7, 0, 0, 0, 7, 0, 0, 0, 0, 0,
       0, 7, 0, 0, 0, 0, 0, 0, 2, 0, 2, 3, 4, 6, 3, 2, 2, 2, 2, 2, 2,
       7, 7, 7, 3, 0, 7, 7, 0, 2, 0, 7, 1, 1, 2, 2, 1, 1, 7, 7, 0, 0, 7,
       7, 2, 0, 2, 1, 1, 0, 0, 3, 2, 0, 3, 1, 7, 7, 2, 7, 0, 0, 7, 3, 3,
       2, 5, 4, 2, 2, 1, 7, 0, 3, 2, 2, 6, 6, 7, 6, 6, 4, 4, 0, 0, 0, 2,
       3, 3, 2, 2, 0, 0, 0, 0, 0, 5, 5, 7, 2, 1, 1, 1, 1, 1, 1, 7, 7, 5,
       7, 2, 2, 2, 6, 7, 6, 4, 7, 6, 7, 7, 7, 2, 4, 0, 2, 7, 0, 2, 2, 2,
       2, 0, 2, 3, 7, 7, 7, 1, 2, 0, 0, 0, 0, 6, 6, 2, 2, 7, 2, 0, 0, 1,
       1, 6, 2, 5, 1, 5, 5, 5, 5, 7, 4, 1, 0, 2, 2, 2, 2, 2, 2, 2, 5, 1,
       0, 2, 7, 7, 1, 7, 3, 0, 3, 3, 3, 3, 7, 3, 7, 0, 0, 2, 2, 0, 2, 2,
       2, 2, 2, 1, 7, 7, 7, 1, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 3, 1, 1, 1,
       1, 0, 0, 3, 7, 7, 7, 0, 3, 3, 2, 1, 2, 0, 1, 2, 2, 4, 4, 4, 2, 0,
       6, 4, 1, 4, 2, 0, 0, 7, 0, 0, 2, 0, 2, 2, 2, 7, 7, 7, 7, 1, 6, 2,
       6, 6, 5, 7, 7, 7, 7, 7, 7, 7, 2, 6, 0, 5, 6, 6, 0])
```

Now, we got our cluster numbers for each location we have, lets add that to our dataframe as a column

In [63]:

```
final_df["cluster"] = model.predict(final_df[["lat","lon"]])
```

In [64]:

```
map_osm = folium.Map(location=[final_df["lat"][0],final_df["lon"][0]], zoom_start=10)
final_df.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]], popup = row["senso
r_name"] , color = cluster_colors[row["cluster"]] ,
                                              fill_color = colormap(row.num_cars),
                                              fill_opacity = 0.8,
                                              radius=10)
                                             .add_to(map_osm), axis=1)
map_osm
```

Out[64]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Perfect. Now we can group our data by place clusters and get the most common venue types

STEP 9: CREATING AREA MARKERS INSTEAD OF VISUALIZING EACH POINT INDIVIDUALLY

In [65]:

```
gby_2 = final_df.groupby("cluster")[["num_cars","lat","lon"]].mean()
```

In [66]:

```
gby_2["cluster"] = gby_2.index
```

After made a groupby operation on our clusters, We get their mean traffic density , mean lat and lon (basically the center of our cluster)

In [67]:

```
gby_2.head()
```

Out[67]:

cluster	num_cars	lat	lon	cluster
0	83095.827270	41.021847	28.826788	0
1	62742.771452	40.942852	29.250016	1
2	77327.549468	41.061036	28.977095	2
3	56068.883995	41.044550	28.594061	3
4	18159.183263	41.217180	29.030919	4

In [68]:

```
gby_2.index.name="index"
```

In [69]:

```
gby_2.head()
```

Out[69]:

index	num_cars	lat	lon	cluster
0	83095.827270	41.021847	28.826788	0
1	62742.771452	40.942852	29.250016	1

	num_cars	lat	lon	cluster
1	62742.771452	40.942852	29.250016	1
2	77327.549468	41.061036	28.977095	2
3	56068.883995	41.044550	28.594061	3
4	18159.183263	41.217180	29.030919	4

index

Let's get the most frequent venue type for each cluster

To get that, we need the mode function from scipy library. We can not use pandas's mode function because it only works on data frames but we need to apply that row-wise

In [70]:

```
from scipy import stats
grouped_by = final_df.groupby("cluster").agg(lambda x: stats.mode(x))[["most_freq"]]
```

In [71]:

```
grouped_by
```

Out[71]:

cluster	most_freq
0	([Residential Building (Apartment / Condo)], [...
1	([Factory], [14])
2	([Office], [27])
3	([Residential Building (Apartment / Condo)], [6])
4	([Farm], [4])
5	([Farm], [10])
6	([Factory], [8])
7	([Office], [24])

In [72]:

```
grouped_by["most_freq"][0][1]
```

Out[72]:

```
array([16])
```

In [73]:

```
grouped_by.iloc[0]["most_freq"][0][0]
```

Out[73]:

```
'Residential Building (Apartment / Condo)'
```

Now we can concat

We are applying the mode function I mentioned above to each row using aggregate function of pandas

In [74]:

```
gby_2 = pd.concat([gby_2 , final_df.groupby("cluster").agg(lambda x: stats.mode(x))[["most_freq"]]]
,axis=1)
```

In [75]:

```
gby_2["Most_frequent_venue"] = grouped_by.apply(lambda x: x[0][0][0] , axis=1)
gby_2["Count"] = grouped_by.apply(lambda x: x[0][1][0] , axis=1)
gby_2.drop(["most_freq"],axis=1,inplace=True)
gby_2
```

Out[75]:

	num_cars	lat	lon	cluster	Most_frequent_venue	Count
0	83095.827270	41.021847	28.826788	0	Residential Building (Apartment / Condo)	16
1	62742.771452	40.942852	29.250016	1	Factory	14
2	77327.549468	41.061036	28.977095	2	Office	27
3	56068.883995	41.044550	28.594061	3	Residential Building (Apartment / Condo)	6
4	18159.183263	41.217180	29.030919	4	Farm	4
5	19920.389914	41.161907	29.235788	5	Farm	10
6	21366.964268	41.201716	28.837974	6	Factory	8
7	88929.834022	41.016909	29.098040	7	Office	24

LAST VISUALIZATION

In [76]:

```
gby_2
```

Out[76]:

	num_cars	lat	lon	cluster	Most_frequent_venue	Count
0	83095.827270	41.021847	28.826788	0	Residential Building (Apartment / Condo)	16
1	62742.771452	40.942852	29.250016	1	Factory	14
2	77327.549468	41.061036	28.977095	2	Office	27
3	56068.883995	41.044550	28.594061	3	Residential Building (Apartment / Condo)	6
4	18159.183263	41.217180	29.030919	4	Farm	4
5	19920.389914	41.161907	29.235788	5	Farm	10
6	21366.964268	41.201716	28.837974	6	Factory	8
7	88929.834022	41.016909	29.098040	7	Office	24

Before we visualize again, we should re-define the colormap because we got rid of the outliers, our maximum and minimum number of cars migh have changed (minimum was 0, it definitely changed but the maximum has not changed actually but we should define them directly using max and min functions to have a clearer colormap).

In [77]:

```
colormap = cmap.LinearColormap(colors=['green',"yellow",'red'] , vmin=gby_2.num_cars.min() , vmax=g
by_2.num_cars.max())
```

In [78]:

```
map_osm = folium.Map(location=[gby_2["lat"][0],gby_2["lon"][0]], zoom_start=10)
gby_2.apply(lambda row:folium.CircleMarker(location=[row["lat"], row["lon"]],
                                           popup =  f"{row.Most_frequent_venue} appeared {row.Count
\n , Average number of cars per day is {row.num_cars} ",
                                           color = cluster_colors[row["cluster"]] ,
                                           fill_color = colormap(row.num_cars),
                                           fill_opacity=0.7,
                                           radius=80)
                                             .add_to(map_osm), axis=1)
map_osm
```

Out[78]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Perfect, from that map we can clearly see that there is a relation between traffic and venues. Even while the venue information from foursquare is not enough, we managed to make our cluster the city on its own (its so close to the way the citizens cluster the districts of the city actually) and get the traffic distrubutions of those districts, and get the data of the most frequent venues on those districts

No matter how many times we run the code, the most appeared types of zones do not change , in the clusters with a lot of traffic. In the green zones, we get farms, beaches and sometimes factories. That makes us able to say that traffic has a correlation with the district planning here, mostly offices. If the districts were planned more fairly, traffic would spread and decrease. But because such a crowded cities most investments were done in a little part of it, it has zones that record around 18 000 cars per day and 91 000 cars per day next to each other (less than 20 km's I suppose). Thats more than 5 times.

In [79]:

```
gby_2
```

Out[79]:

	num_cars	lat	lon	cluster	Most_frequent_venue	Count
0	83095.827270	41.021847	28.826788	0	Residential Building (Apartment / Condo)	16
1	62742.771452	40.942852	29.250016	1	Factory	14
2	77327.549468	41.061036	28.977095	2	Office	27
3	56068.883995	41.044550	28.594061	3	Residential Building (Apartment / Condo)	6
4	18159.183263	41.217180	29.030919	4	Farm	4
5	19920.389914	41.161907	29.235788	5	Farm	10
6	21366.964268	41.201716	28.837974	6	Factory	8
7	88929.834022	41.016909	29.098040	7	Office	24

In [80]:

```
gby_2.corr()
```

Out[80]:

	num_cars num_cars	lat lat	lon lon	cluster cluster	Count Count
num_cars	1.000000	-0.841139	-0.102163	-0.368582	0.795233
lat	-0.841139	1.000000	-0.072988	0.513202	-0.543183
lon	-0.102163	-0.072988	1.000000	0.137627	0.249434
cluster	-0.368582	0.513202	0.137627	1.000000	-0.094309
Count	0.795233	-0.543183	0.249434	-0.094309	1.000000

In [81]:

```
venue_dummies = pd.get_dummies(gby_2["Most_frequent_venue"],prefix = "venue")
venue_dummies
```

Out[81]:

	venue_Factory	venue_Farm	venue_Office	venue_Residential Building (Apartment / Condo)
0	0	0	0	1
1	1	0	0	0
2	0	0	1	0
3	0	0	0	1
4	0	1	0	0
5	0	1	0	0
6	1	0	0	0
7	0	0	1	0

In [85]:

```
correlation_dataFrame = pd.concat([gby_2.drop(["Most_frequent_venue"],axis=1),venue_dummies],axis=
1)
```

In [83]:

```
correlation_dataFrame.drop(["lat","lon","cluster"] , axis = 1,inplace=True)
```

Dropping the columns that contain numerical values that do not actually contain a continious value, specially the coordinate columns here because they are containing numerical values that actually does not contain a continious value that decreases or increases according to something that might have an effect on the traffic density. To be clearer, coordinates can be or not be correlated with traffic, but in this situation, they are not increasing or decreasing according to a variable, they are computed according to their geospatial locations so we can not examine the correlation between location and traffic only with coordinates, we would have needed to calculate the geospatial distance between them and a center or multiple centers like city centers but its not related to our subject.

Dropping the cluster column here again because while its an integer column, the value it contains is actually a label for the cluster. The numbers assigned to the clusters are different each time this cod gets re run.

In [86]:

```
corr = correlation_dataFrame.corr(method = "spearman")
corr.style.background_gradient(cmap='coolwarm')
```

Out[86]:

	num_cars	lat	lon	cluster	Count	venue_Factory	venue_Farm	venue_Office	venue_Residential Building (Apartment / Condo)
num_cars	1.000000	-0.785714	-0.095238	-0.214286	0.833333	-0.125988	-0.755929	0.629941	0.251976
lat	-0.785714	1.000000	-0.214286	0.357143	-0.571429	-0.125988	0.629941	-0.251976	-0.251976

	num_cars	lat	lon	cluster	Count	venue_Factory	venue_Farm	venue_Office	venue_Residential Building (Apartment / Condo)
lon	0.095238	0.214286	1.000000	0.190476	0.190476	-0.251976	-0.377964	0.125988	-0.755929
cluster	-0.214286	0.357143	0.190476	1.000000	0.190476	0.000000	0.251976	0.251976	-0.503953
Count	0.833333	-0.571429	0.190476	-0.190476	1.000000	-0.125988	-0.503953	0.755929	-0.125988
venue_Factory	-0.125988	-0.125988	0.251976	0.000000	-0.125988	1.000000	-0.333333	-0.333333	-0.333333
venue_Farm	-0.755929	0.629941	0.377964	0.251976	-0.503953	-0.333333	1.000000	-0.333333	-0.333333
venue_Office	0.629941	-0.251976	0.125988	0.251976	0.755929	-0.333333	-0.333333	1.000000	-0.333333
venue_Residential Building (Apartment / Condo)	0.251976	-0.251976	-0.755929	-0.503953	-0.125988	-0.333333	-0.333333	-0.333333	1.000000

From the correlation heatmap above, it can be seen that there is a positive correlation between the number of cars and the venue_Office. and a negative correlation between Factories and farms. So we can say that in Istanbul, Whereever the most frequent venue type is office, as the rate increases, traffic density increases. That also proves that the unbalanced distrubution of zones around Istanbul has a high effect on the traffic density.

In [ ]: