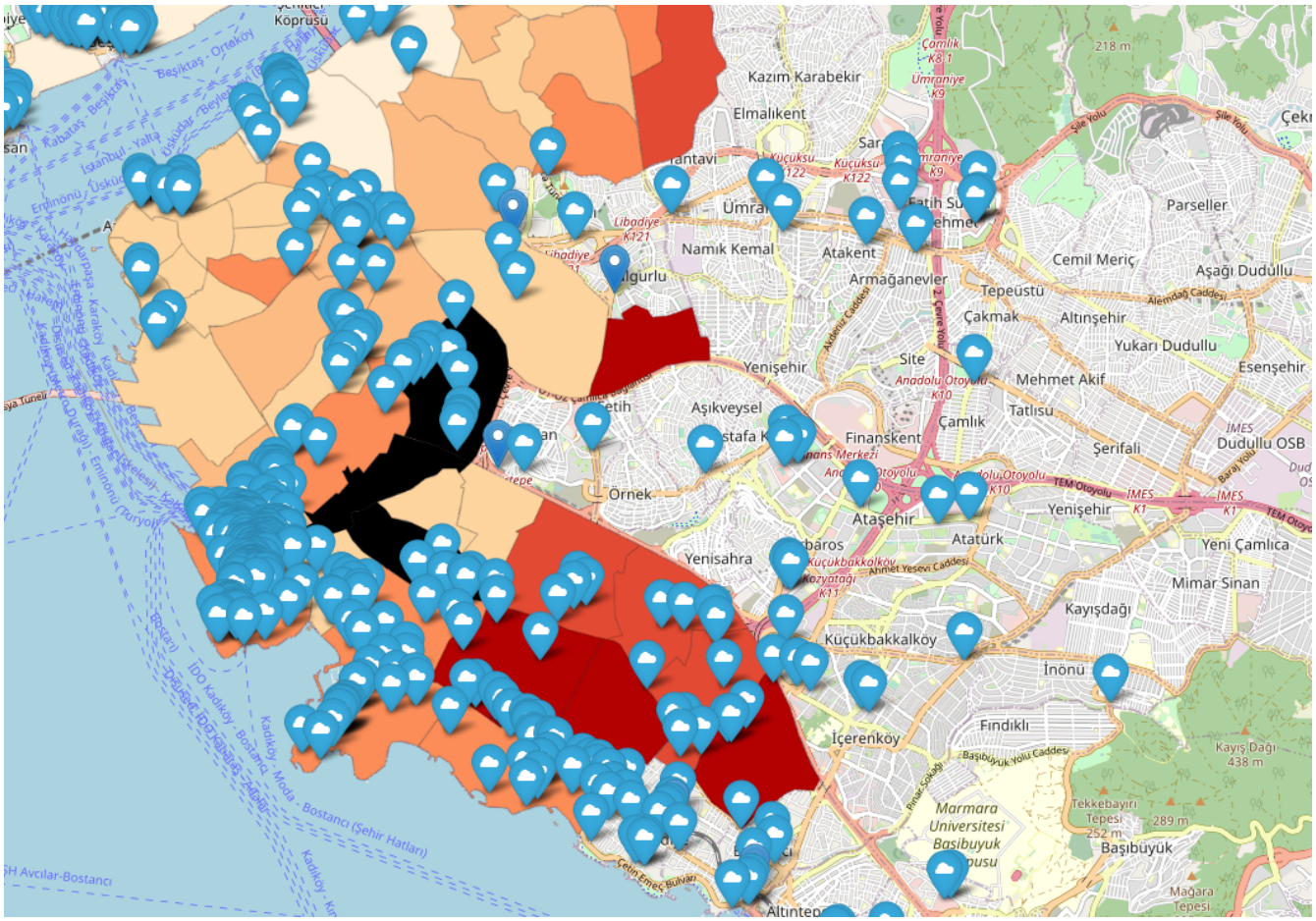
Data-Driven Analysis

into Istanbul Café Locations



In the following script, I will be excavating into the cityscape of Istanbul. The threefold objective is to examine the city's population distribution, the geographical spread of coffee shops, and the average Airbnb rental prices across various neighbourhood.

Data

All the data used in this project were obtained from various sources on the internet. While some were ready to use, others had to be wrangled and cleaned.

-Population & Demographics Data

The population data obtained from Turkish Statistical Institute: here
The dataset contained every neighborhood in Turkey. I have narrowed it down to İstanbul districts.

-Geographical Data

To locate the neighborhoods, I leveraged on Nominatim Open Street maps project. From the API neighborhood boundaries as polygon coordinates which were then converted to geojson files using an API provided by geojson.io

-Location Data

The list of coffee shops was obtained by querying YELP through the API. As I use a free-tier account, the results of my query -coffee- were has their limits.

Listing the cafes in the given coordinates

The first part involves listing all the cafes in the given coordinates. The required libraries will be followed.

```
import pandas as pd
import numpy as np
import requests
import json
import pandas as pd
import urllib.parse
import folium
import geopandas as gpd
import csv
import time
import matplotlib.pyplot as plt
import seaborn as sns
```

Leveraging the Yelp API, I have created a query for coffee shops in the neighborhoods of Istanbul.

```
# Yelp API key
API_KEY = 'Fill it in with grace'


# Geographic coordinates for Istanbul
latitude = 40.9819
longitude = 29.0247


# Setting the stage with a 'coffee' query
term = 'coffee'
```

```
# URL for Yelp API
url = 'https://api.yelp.com/v3/businesses/search'

# Headers with API key
headers = {
  'Authorization': 'Bearer %s' % API_KEY,
}
```

The script employs requests and JSON libraries to retrieve and process data from the Yelp API. Yelp limits the results to 50 per query. However, in a city like Istanbul, where cafes are numerous, this limit is insufficient. This requires a combination of offset and while loop as well as time library to fetch more results and not get banned.

The while-loop continuously queries the API for cafes until there are no more left(990 items). For each cafe, the name, coordinates, category, rating, and review count have been extracted. This data is then appended to the data frame, which provides us with a structured dataset of cafes(dfcafes).

```
# A whisper to the oracle
parsed_data = []
offset = 0
while True:
  # The treasures
  params = {
      'term': term,
      'latitude': latitude,
      'longitude': longitude,
      'offset': offset,
  }
```

```
    # Send request and get response
    resp = requests.get(url=url, headers=headers, params=params)
    data = json.loads(resp.text)

    # More treasures
    businesses = data.get('businesses', [])
    if not businesses:
        break

    # Decoding the cryptic messages
    for business in businesses:
        business_name = business['name']
        business_lat = business['coordinates']['latitude']
        business_lng = business['coordinates']['longitude']
        business_category = business['categories'][0]['title'] if business['categories'] else None
        business_rating = business['rating']
        business_review_count = business['review_count']
        parsed_data.append([business_name, business_lat, business_lng, business_category, business_rating, business_review_count])

    # Doing very important stuff
    offset += len(businesses)

    # Paying the homage
    time.sleep(1)

# Convert to pandas DataFrame
import pandas as pd
dfcafes = pd.DataFrame(parsed_data, columns=['Name', 'Latitude', 'Longitude', 'Category', 'Rating', 'ReviewCount'])
```

```
# Get rid of the incompetence
dfcafes = dfcafes.dropna()
```

The explanatory info of the data:

```
Data columns (total 6 columns):
#   Column       Non-Null Count  Dtype
---  ------       --------------  -----
0   Name         1000 non-null   object
1   Latitude     1000 non-null   float64
2   Longitude    1000 non-null   float64
3   Category     1000 non-null   object
4   Rating       1000 non-null   float64
5   ReviewCount  1000 non-null   int64
dtypes: float64(3), int64(1), object(2)
```

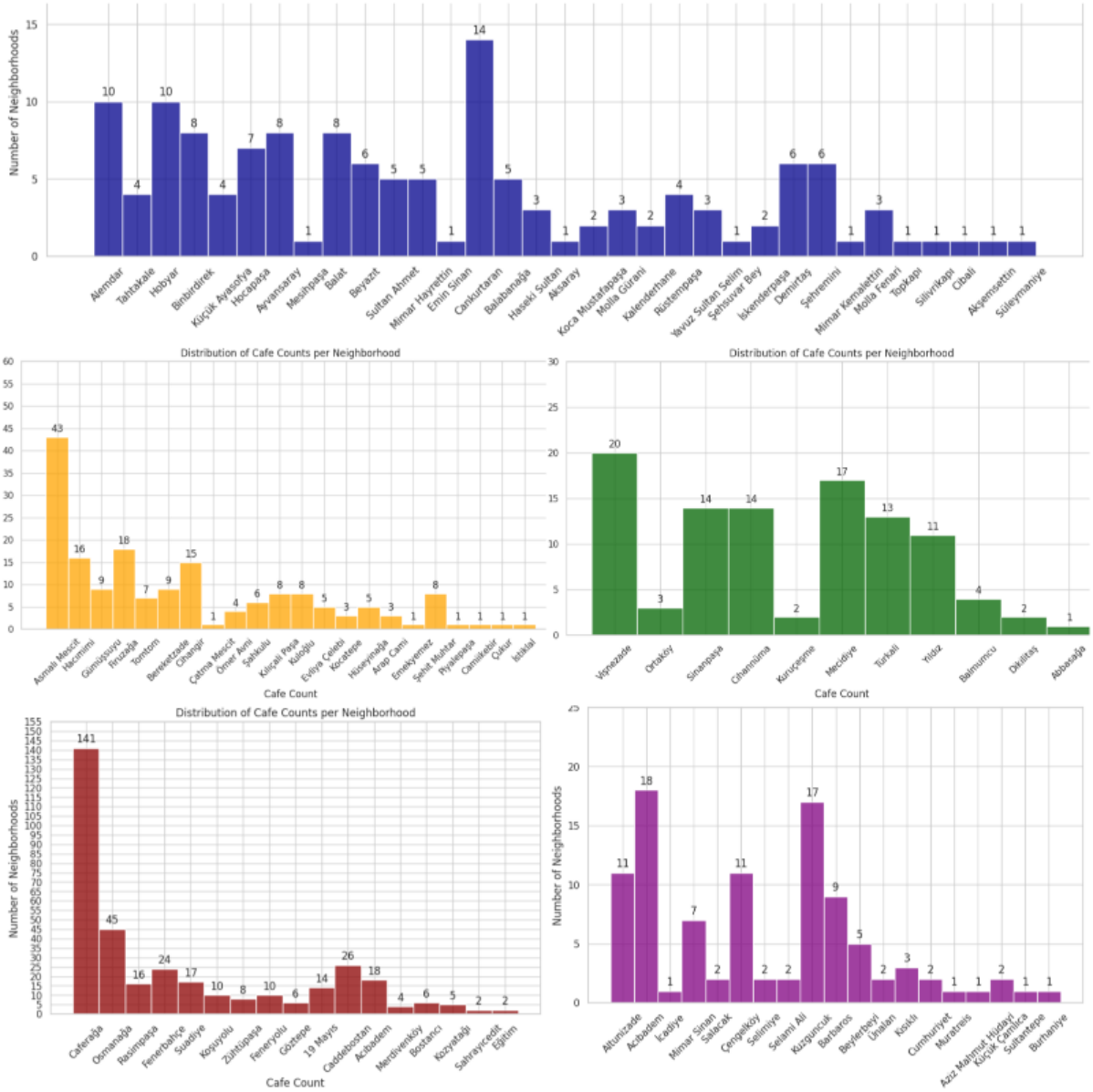
As it is seen dfcafes does not provide neighborhood info. As I will need it through my report, I have written I function, which takes the coordinates of the cafe and through Nominatim API, find the neighborhood.

```
def get_neighborhood(lat, lon):
  url = f"https://nominatim.openstreetmap.org/reverse?lat={lat}&lon={lon}&format=json&zoom=18"
  response = requests.get(url)
  data = response.json()

  if neighborhood is not None:
     neighborhood = neighborhood.replace(" Mahallesi", "")

  return neighborhood

def add_neighborhood(df, lat_col, lon_col):
  neighborhoods = []
```

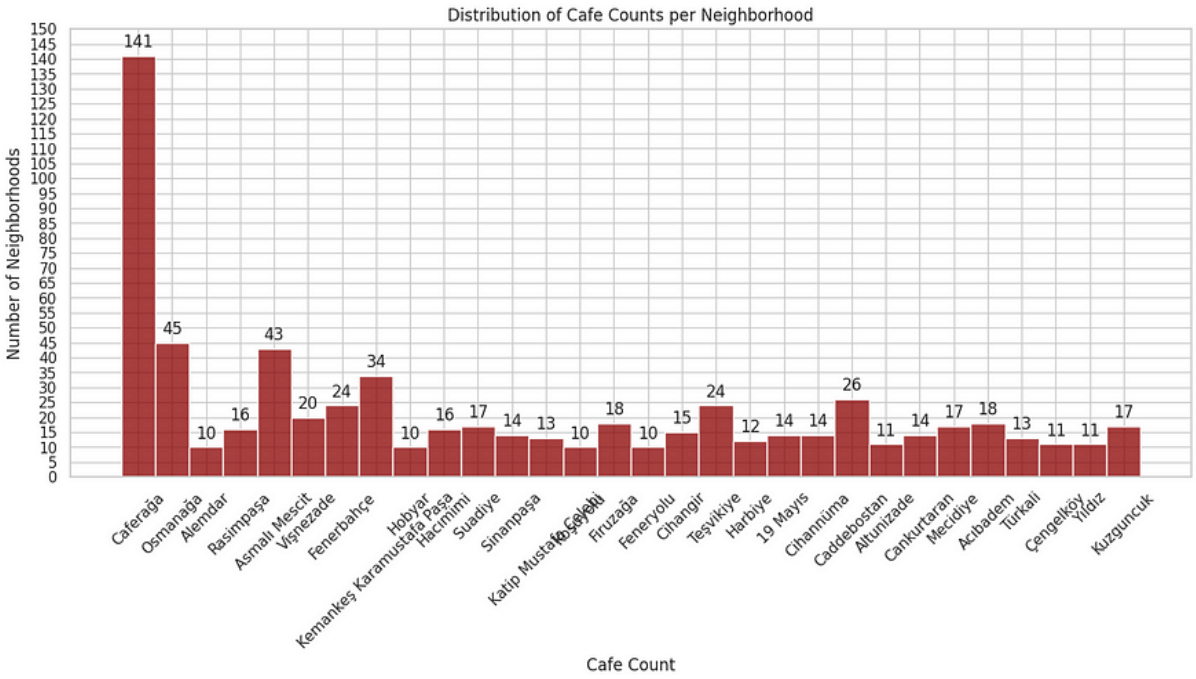
```
    neighborhood = get_neighborhood(row[lat_col], row[lon_col])
    neighborhoods.append(neighborhood)
  df['Neighborhood'] = neighborhoods
  return df
```

```
#1/3 Cafes in Istanbul
dfcafes = add_neighborhood(dfcafes, 'Latitude', 'Longitude')
```

By neighborhood the cafe counts from YELP, are seen as below.



If we combine all the neighborhoods:



Shaping the borders of neighborhoods of Istanbul

The next challenge is to identify the boundaries of Istanbul's neighborhoods. Unfortunately, no API or data file provides this information. Therefore, I will use the names of the neighborhoods from the Turkish Statistical Institution population data and retrieve the geometry data that encloses each neighborhood.

```
# I have preferred Beşiktaş, Kadıköy, Beyoğlu, Fatih, Üsküdar, you like to proceed with your beloved choices
neighborhood_list = ["Abbasağa, Beşiktaş",
                     "Akat, Beşiktaş",
                     "Arnavutköy, Beşiktaş",
                     "Balmumcu, Beşiktaş",
                     "Bebek, Beşiktaş",
                     "Cihannüma, Beşiktaş"
                     ...]
```

The neighborhood loop iterates through every neighborhood name in the provided list. The names are formatted to be compatible with the API request. The city name, Istanbul, is added to avoid confusion with other cities that might have the same neighborhood names.

The URL is formed to make a search query to the Nominatim API from OpenStreetMap. The “q” parameter stands for the neighborhood name + city, and polygon_geojson=1 is used to get the JSON format of the neighborhood’s geometry.

If the response exists, the code takes the first element, the most relevant one, and constructs a feature dictionary that represents the neighborhood. The feature dictionary stores the name and the geometry of the neighborhood.

```
for neighborhood in neighborhood_list:
  request_text = urllib.parse.quote(neighborhood + " Istanbul")
  request_text =
f"https://nominatim.openstreetmap.org/search?q={request_text}&polygon_geojson=1&format=json"
  response = requests.get(request_text)


  response = response.json()


  if response:
    neighborhood_result = response[0]
    decoded_name = urllib.parse.unquote(neighborhood_result['display_name'])
    feature = {
      "type": "Feature",
      "properties": {
        "name": decoded_name  # Use the decoded name here
```

```
        },
        "geometry": neighborhood_result['geojson']
    }
    result_dict[neighborhood] = feature
  else:
    print(f"No results for {neighborhood}")

with open('resultv2.json', 'w') as fp:
  json.dump(geojson_data, fp)
```

The gpd.read_file() function is provided by the GeoPandas library and is used to read spatial data.

```
districts = gpd.read_file('resultv2.json')
print(districts.head())
                                name  \
0  Abbasağa Mahallesi, Beşiktaş, İstanbul, Marmar...
1  Akat Mahallesi, Beşiktaş, İstanbul, Marmara Bö...
2  Arnavutköy, Bebek Arnavutköy Caddesi, Arnavutk...
3  Balmumcu Mahallesi, Beşiktaş, İstanbul, Marmar...
4  Bebek Mahallesi, Beşiktaş, İstanbul, Marmara B...

                                geometry
0  POLYGON ((29.00319 41.04803, 29.00321 41.04761...
1  POLYGON ((29.01997 41.09076, 29.02036 41.08906...
2                  POINT (29.04327 41.06718)
3  POLYGON ((29.00983 41.05668, 29.01034 41.05653...
4  POLYGON ((29.03265 41.07844, 29.03288 41.07817...
```

Next, reading the population data from a CSV file in order to color the map by referencşng the density of the population.

```
population = pd.read_csv("C:\\Users\...tüiklast2.csv")
```

Then I have merged the population data with the GeoDataFrame data. This allows us to have both polygon coordinates and the population of the neighborhood at the same time.

```
merged = districts.set_index('name').join(population.set_index('name')).reset_index()
```

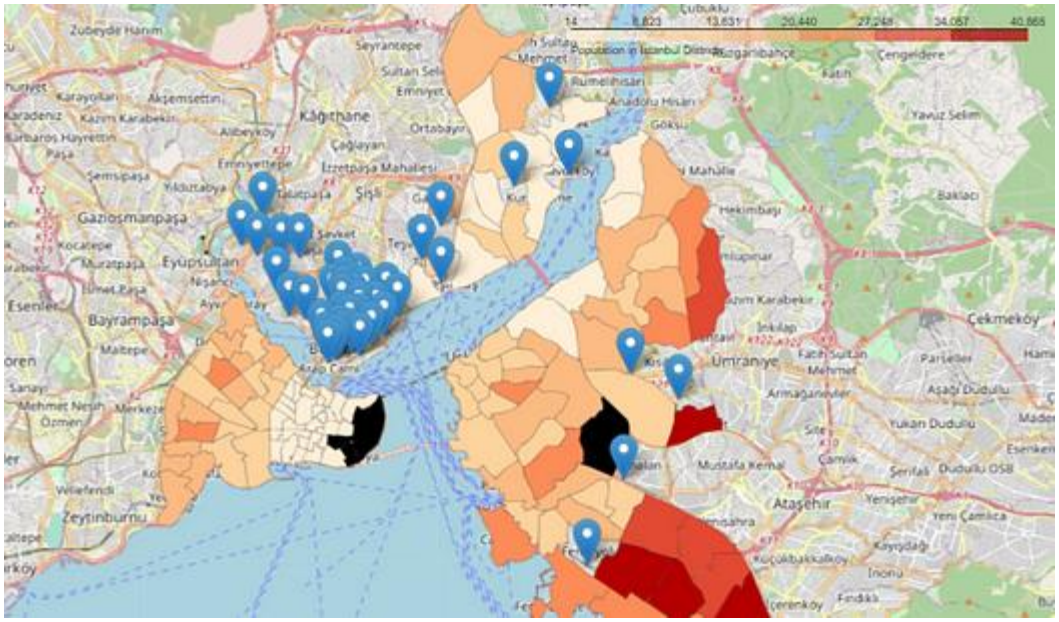
Unfortunately, geojson file does not provide neighborhood name. Yet we have the coordinates

```
def get_neighborhood(lat, lon):
    url = f"https://nominatim.openstreetmap.org/reverse?lat={lat}&lon={lon}&format=json&zoom=18"
    response = requests.get(url)
    data = response.json()

    if neighborhood is not None:
        neighborhood = neighborhood.replace(" Mahallesi", "")

    return neighborhood

def add_neighborhood(df, lat_col, lon_col):
    neighborhoods = []
        neighborhood = get_neighborhood(row[lat_col], row[lon_col])
        neighborhoods.append(neighborhood)
    df['Neighborhood'] = neighborhoods
    return df

#1/3 Cafes in Istanbul
```

```
dfcafes = add_neighborhood(dfcafes, 'Latitude', 'Longitude')
```

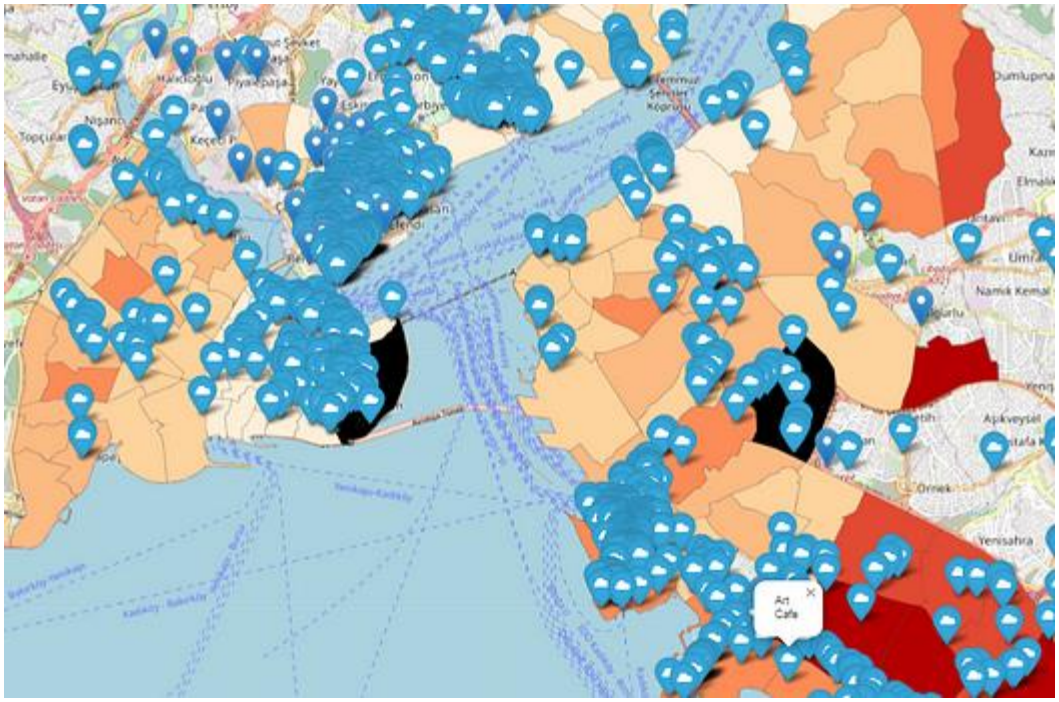
For visualizing the data, I have used the folium library to create a map centered on Istanbul. Choropleth map uses color shading or patterns to represent different values or data for specific geographic regions. The parameters are set according to our requirements.

```
m = folium.Map(location=[41.0082, 28.9784], zoom_start=10)

choropleth = folium.Choropleth(
  geo_data=merged.__geo_interface__,
  name='choropleth',
  data=merged,
  columns=['name', 'TOPLAM'],
  key_on='feature.properties.name',
  fill_color='OrRd',
  fill_opacity=1,
  line_opacity=0.2,
  legend_name='Population in Istanbul Districts'
)

choropleth.add_to(m)
```



Next, we add a marker for each coffee shop on the map.

```
# Add a marker for each coffee shop
for lat, lon, name in dfcafes[['Latitude', 'Longitude', 'Name']].values.tolist():
  folium.Marker(
      location=[lat, lon],
      popup=name,
      icon=folium.Icon(icon="cloud"),
  ).add_to(m)


# Save to HTML
m.save('ist_pop_cafe.html')
m
```



Airbnb Rental Exploration

In this section, I will add another layer to the analysis — the average Airbnb rental prices for each neighborhood. I will perform some cleaning and filtering operations on the data.

Optionally, filtering further by only including listings that have received reviews, and ones that receive more than 0.5 reviews per month. These filters could help ensure that we're only including popular, actively rented listings in our analysis.

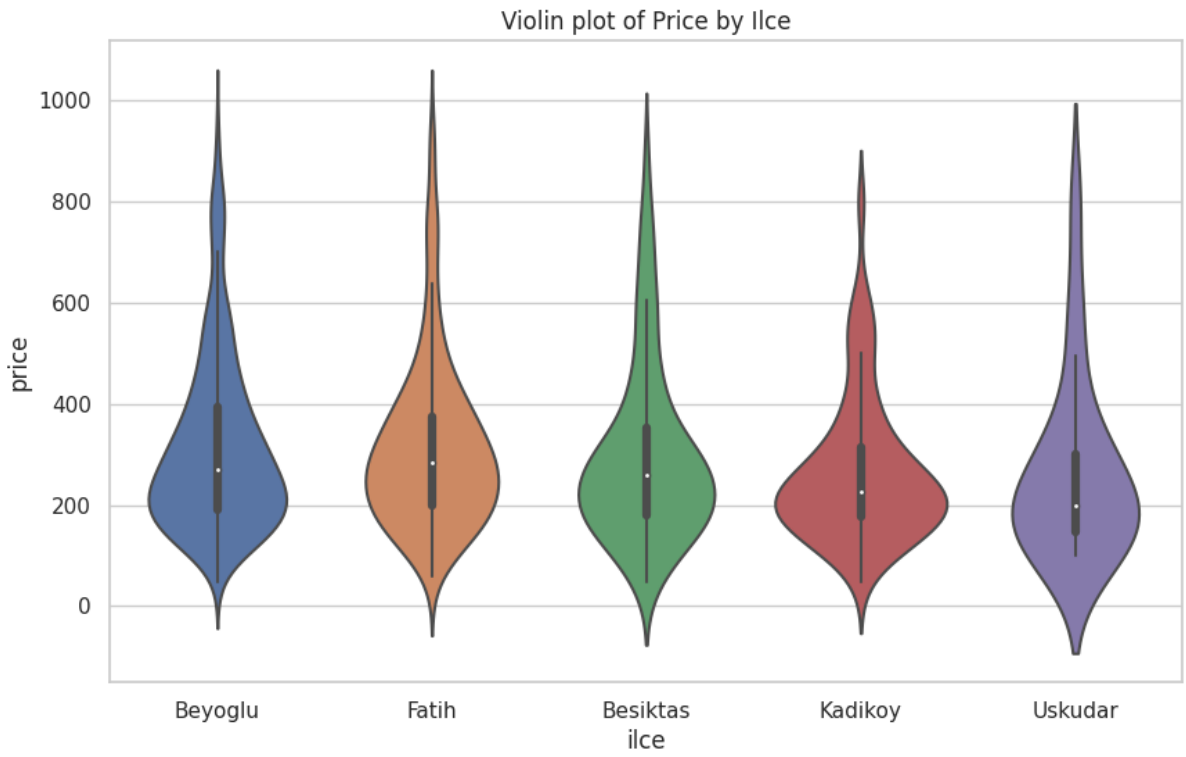
```
# Read the csv file
df = pd.read_csv('C:\\Users\\...AirbnbIstanbul.csv')


# Manipulate the data
df = df.drop_duplicates()
```

```
df = df[df['latitude'] != 'n/a']
df = df[df['longitude'] != 'n/a']
df = df[df['neighbourhood'].isin(['Kadikoy', 'Fatih', "Üsküdar", "Beyoğlu", 'Besiktas'])]
df = df[df['room_type'].isin(['Entire home/apt'])]
df = df[df['price'] < 1000]

#Optinal:
df = df[df['number_of_reviews'] != 0]
df = df[df['reviews_per_month'] > 0.5]

# Save the cleaned data to a new csv file
df.to_csv('final_airbnb.csv', index=False)

df = df.dropna(subset=['neighbourhood'])
```

```
d,name,host_id,host_name,neighbourhood_group,ilce,latitude,longitude,room_type,price,minimum_nights,number_of_reviews,last_review,reviews_per_month,calculated_host_listings_count,availability_365
20815,The Bosphorus from The Comfy Hill,78838,Gülder,,Besiktas,4.106.984,2.904.545,Entire home/apt,100,30,41,2018-11-07,0.38,2,49
25436,House for vacation rental furnutare,105823,Yesim,,Besiktas,4.107.731,2.903.891,Entire home/apt,211,21,0,,,1,83
34177,PETIT HOUSE,147330,Ercan,,Besiktas,4.106.901,2.903.882,Entire home/apt,237,30,8,2016-07-14,0.15,2,357
47264,Kurucesme stunning seaview peacfull Flat,213410,Evrim,,Besiktas,4.106.486,2.903.473,Entire home/apt,395,3,8,2018-09-12,0.08,6,251
53612,Gorgeous Bosphorus View 3BDR Apt with Terrace,250139,Onur,,Besiktas,4.104.439,2.901.266,Entire home/apt,501,30,20,2017-11-27,0.24,1,365
```

In the first plot, Beyoğlu displayed the highest average rental price, followed by Beşiktaş, Kadıköy, Fatih, and Üsküdar, in descending order. The distribution of prices in Beyoğlu showed a single peak, suggesting a unimodal distribution.

Beşiktaş and Kadıköy demonstrated a bimodal distribution with a significant portion of prices at both lower and higher ends.
Fatih and Üsküdar had their prices leaning towards the lower end with a long tail extending towards the higher end, indicating a right-skewed distribution.

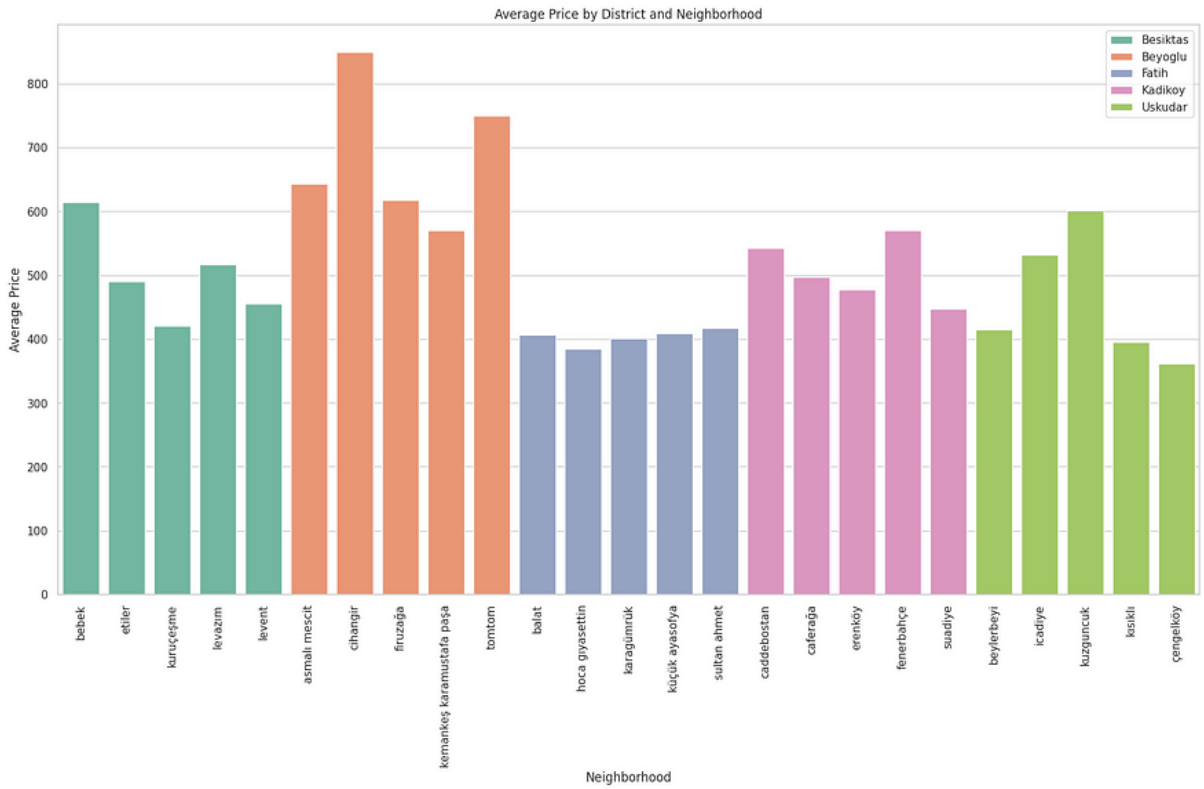


Airbnb data set as it is seen, does also not provide neighborhood values. With the same add_neighborhood function, neighborhood values can be added to the CSV file as well.

```
#2/3 Airbnb data

df_airbnb = add_neighborhood(df_airbnb, 'latitude', 'longitude')
```

So, the second plot breaks down the average rental prices within neighborhoods in these districts.

For Beşiktaş, the highest prices are seen in the Bebek neighborhood, followed by Levazım and Etiler. In Beyoğlu, the Cihangir neighborhood commands the highest prices, followed by Tomtom and Asmalı Mescit. For Fatih, the most expensive neighborhood for Airbnb rentals is Sultan Ahmet, while in Kadıköy, the highest prices are found in Fenerbahçe. Lastly, in Üsküdar, the Kuzguncuk neighborhood has the highest average rental prices.



Here are the

```
# Average
average_prices = {semt: sum(price_list) / len(price_list) for semt, price_list in prices.items()}


# Markers for Airbnb location
for semt, (lat, lon) in locations.items():
  average_price = average_prices[semt]
```

```
    # Map string
    info = f"Semt: {semt}<br>Average Price: {average_price:.2f}"

    # Create a popup
    popup = folium.Popup(info, max_width=250)

    # Circle marker for the average price
    folium.CircleMarker(
        location=[lat, lon],
        radius=average_price / 20,  # you might need to adjust the division factor to get appropriate circle sizes
        color="darkgrey",
        fill=True,
        fill_color="green",
        fill_opacity=0.6,
        popup=popup
    ).add_to(m)

m.save('istanbul_last.html')

m
```

And the representaiton of Istanbul

Finally, the correlation matrix among any potential variable is shown in below.

The number of cafes and population size show a mild positive correlation of 0.304, suggesting that districts with larger populations tend to have more cafes. This could be due to:

Demand and Supply: One straightforward explanation for this positive correlation might be the basic economic principle of supply and demand. In areas with larger populations, there's likely to be greater demand for coffee shops. This demand can sustain a larger number of cafes, allowing more to exist in these areas compared to less populated neighborhoods.

Social Gathering Places: Cafes are often viewed as social gathering places, and a larger population might necessitate more such spaces. As population density increases, cafes become not just venues for coffee but also community hubs for socializing, working, and leisure activities.

Diversity of Preferences: With a larger population, there's likely to be a

wider variety of preferences and tastes. Different preferences can support a larger number of cafes as well.

Higher Foot Traffic: Neighborhoods with larger populations often have higher foot traffic, which can benefit cafes. More people passing by can mean more potential customers, whether it's people heading to work in the morning, meeting friends during the day, or seeking a late-night coffee fix.

However, both the number of cafes and population size show a negative correlation with the mean Airbnb price, -0.237 and -0.296 respectively. This suggests that areas with more cafes and larger populations tend not to have high Airbnb rental prices. This could be due to:

Competition: More cafes could mean more competition, which may keep prices, including Airbnb rentals in lower degree.
Type of Neighborhood: Areas with high cafe density might be more commercial such people prefer to visit rather than reside.

Demographics: Neighborhoods with a higher population might have more varied demographics, including students or younger populations who might not have high spending power.
Preference for Experiences: Airbnb guests sometimes look for ‘authentic’ experiences. Areas with a higher concentration of local cafes might be perceived as more ‘authentic’ compared to areas with more tourist-centric attractions.