Geospatial Data Visualization

In [1]:
```
%%html
<style>
    /* Jupyter */
    .rendered_html table,
    /* Jupyter Lab*/
    div[data-mime-type="text-markdown"] table {
        margin-left: 0
    }
</style>
```

Import the necessary Libraries

In [2]:
```
# Basic Libaries
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

## Geospatial Data Visualization Libraries
#!pip install folium - install folium library for mapping
import folium
from folium.plugins import MarkerCluster
```

Import property attributes dataset

In [3]:
```
df = pd.read_csv("innercity.csv")
```

Check a few rows of the loaded dataset to ensure if data is loaded is correctly

In [4]:
```
df.head()
```

Out[4]:

	cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil
0	3034200666	20141107T000000	808100	4	3.25	3020	13457	1.0
1	8731981640	20141204T000000	277500	4	2.50	2550	7500	1.0
2	5104530220	20150420T000000	404000	3	2.50	2370	4324	2.0
3	6145600285	20140529T000000	300000	2	1.00	820	3844	1.0
4	8924100111	20150424T000000	699000	2	1.50	1400	4050	1.0

5 rows × 23 columns

- Dataset is property data from the King county, Washington, USA

- King County is considered the most populous county in Washington, and the 12th-most populous in the United States
- The data is densely clustered around Seattle-Bellevue-Renton-Kent-Federal Way-Tacoma area, an urban conglomeration
- Property data contains 21,613 observations with 23 variables [22 independent and 1 target variable (price)], which includes mostly numerical and date/time attributes and help define property characteristics.

Exploratory Data Analysis

Check the shape of dataframe

In [5]:
```
df.shape
```

Out[5]:
```
(21613, 23)
```

Property data contains **21,613** observations with **23** variables, which includes mostly numerical and date/time attributes and help define property characteristics. The dataset contains **22 independent variables** and **1 target variable (price)**. Each entry represents a property characteristics such as number of bedroom, bathroom, measurements (area, height), year built, aesthetic value (proximity to coast, sight etc.) along with locational attributes (lat, lon, zipcode) according to the set of attributes.

Check the name of fields in data

In [6]:
```
df.columns
```

Out[6]:
```
Index(['cid', 'dayhours', 'price', 'room_bed', 'room_bath', 'living_measu
re',
       'lot_measure', 'ceil', 'coast', 'sight', 'condition', 'quality',
       'ceil_measure', 'basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'living_measure15', 'lot_measure15', 'furnished',
       'total_area'],
      dtype='object')
```

The below table list these attributes with tehir description:

Attribute Information

Attribute	Data Type	Description
cid	Numeric	A notation for a house
dayhours	Date/Time	Date house was sold
price	Numeric	Price is prediction target
room_bed	Numeric	Number of Bedrooms/House

Attribute	Data Type	Description
room_bath	Numeric	Number of bathrooms/bedrooms
living_measure	Numeric	Square footage of the home
lot_measure	Numeric	Square footage of the lot
ceil	Numeric	Total floors (levels) in house
coast	Numeric	House which has a view to a waterfront
sight	Numeric	Has been viewed
condition	Numeric	How good the condition is (Overall)
quality	Numeric	Grade given to the housing unit, based on grading system
ceil_measure	Numeric	Square footage of house apart from basement
basement_measure	Numeric	Square footage of the basement
yr_built	Numeric	Built Year
yr_renovated	Numeric	Year when house was renovated
zipcode	Numeric	Zip
lat	Numeric	Latitude coordinate
long	Numeric	Longitude coordinate
living_measure15	Numeric	Living room area in 2015 (implies-- some renovations) This might or might not have affected the lotsize area
lot_measure15	Numeric	LotSize area in 2015 (implies-- some renovations)
furnished	Numeric	Based on the quality of room
total_area	Numeric	Measure of both living and lot

 # Check the data type and null values present in fields

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   cid               21613 non-null  int64  
 1   dayhours          21613 non-null  object 
 2   price             21613 non-null  int64  
 3   room_bed          21613 non-null  int64  
 4   room_bath         21613 non-null  float64
 5   living_measure    21613 non-null  int64  
 6   lot_measure       21613 non-null  int64  
 7   ceil              21613 non-null  float64
 8   coast             21613 non-null  int64  
 9   sight             21613 non-null  int64  
 10  condition         21613 non-null  int64  
 11  quality           21613 non-null  int64  
 12  ceil_measure      21613 non-null  int64  
 13  basement          21613 non-null  int64  
 14  yr_built          21613 non-null  int64  
 15  yr_renovated      21613 non-null  int64  
 16  zipcode           21613 non-null  int64  
 17  lat               21613 non-null  float64
 18  long              21613 non-null  float64
 19  living_measure15  21613 non-null  int64  
 20  lot_measure15     21613 non-null  int64  
 21  furnished         21613 non-null  int64  
 22  total_area        21613 non-null  int64  
dtypes: float64(4), int64(18), object(1)
memory usage: 3.8+ MB
```

No null value is present in the data

Count of unique values in each field

In [8]:
```
df.nunique()
```

Out[8]:
```
cid                21436
dayhours             372
price               3625
room_bed              13
room_bath             30
living_measure      1038
lot_measure         9782
ceil                   6
coast                  2
sight                  5
condition              5
quality               12
ceil_measure         946
basement             306
yr_built             116
yr_renovated          70
zipcode               70
lat                 5034
long                 752
living_measure15     777
lot_measure15       8689
furnished              2
total_area         11163
dtype: int64
```

Check for missing values

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
cid                0
dayhours           0
price              0
room_bed           0
room_bath          0
living_measure     0
lot_measure        0
ceil               0
coast              0
sight              0
condition          0
quality            0
ceil_measure       0
basement           0
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
living_measure15   0
lot_measure15      0
furnished          0
total_area         0
dtype: int64
```

No missing value is present in the data

Check rows with missing values

In [10]:
```
df[df.isnull().any(axis=1)]
```

Out[10]:

cid	dayhours	price	room_bed	room_bath	living_measure	lot_measure	ceil	coast	sight	...	l

0 rows × 23 columns

No duplicate rows exists and hence no duplicate removal step is required

Viewing the data statistics

In [11]:
```
df1 = df.describe().transpose()
dfStyler = df1.style.set_properties(**{'text-align': 'left'})
dfStyler.set_table_styles([dict(selector='th', props=[('text-align', 'left'
```

Out[11]:

	count	mean	std	min	25%
cid	21613.000000	4580301520.864988	2876565571.312059	1000102.000000	2123049194
price	21613.000000	540182.158793	367362.231718	75000.000000	321950.000
room_bed	21613.000000	3.370842	0.930062	0.000000	3.000000
room_bath	21613.000000	2.114757	0.770163	0.000000	1.750000
living_measure	21613.000000	2079.899736	918.440897	290.000000	1427.000000
lot_measure	21613.000000	15106.967566	41420.511515	520.000000	5040.000000
ceil	21613.000000	1.494309	0.539989	1.000000	1.000000
coast	21613.000000	0.007542	0.086517	0.000000	0.000000
sight	21613.000000	0.234303	0.766318	0.000000	0.000000
condition	21613.000000	3.409430	0.650743	1.000000	3.000000
quality	21613.000000	7.656873	1.175459	1.000000	7.000000
ceil_measure	21613.000000	1788.390691	828.090978	290.000000	1190.000000
basement	21613.000000	291.509045	442.575043	0.000000	0.000000
yr_built	21613.000000	1971.005136	29.373411	1900.000000	1951.000000
yr_renovated	21613.000000	84.402258	401.679240	0.000000	0.000000
zipcode	21613.000000	98077.939805	53.505026	98001.000000	98033.00000
lat	21613.000000	47.560053	0.138564	47.155900	47.471000
long	21613.000000	-122.213896	0.140828	-122.519000	-122.328000
living_measure15	21613.000000	1986.552492	685.391304	399.000000	1490.000000
lot_measure15	21613.000000	12768.455652	27304.179631	651.000000	5100.000000
furnished	21613.000000	0.196687	0.397503	0.000000	0.000000
total_area	21613.000000	17186.867302	41589.081215	1423.000000	7035.000000

Describe function illustrates that various fields/attributes have 0 values in the data. Whether these Zero are meaningful or require cleansing would require further data exploration. A few outliers are present in the data (for example75% of the data in field room_bed is within limits of 3 bedroom but value such as 33 is seen as max value) which need to be imputed with right strategy. Skewness is present in the data as well.

Geospatial Data Visualization using Folium Library

Map with simple marker

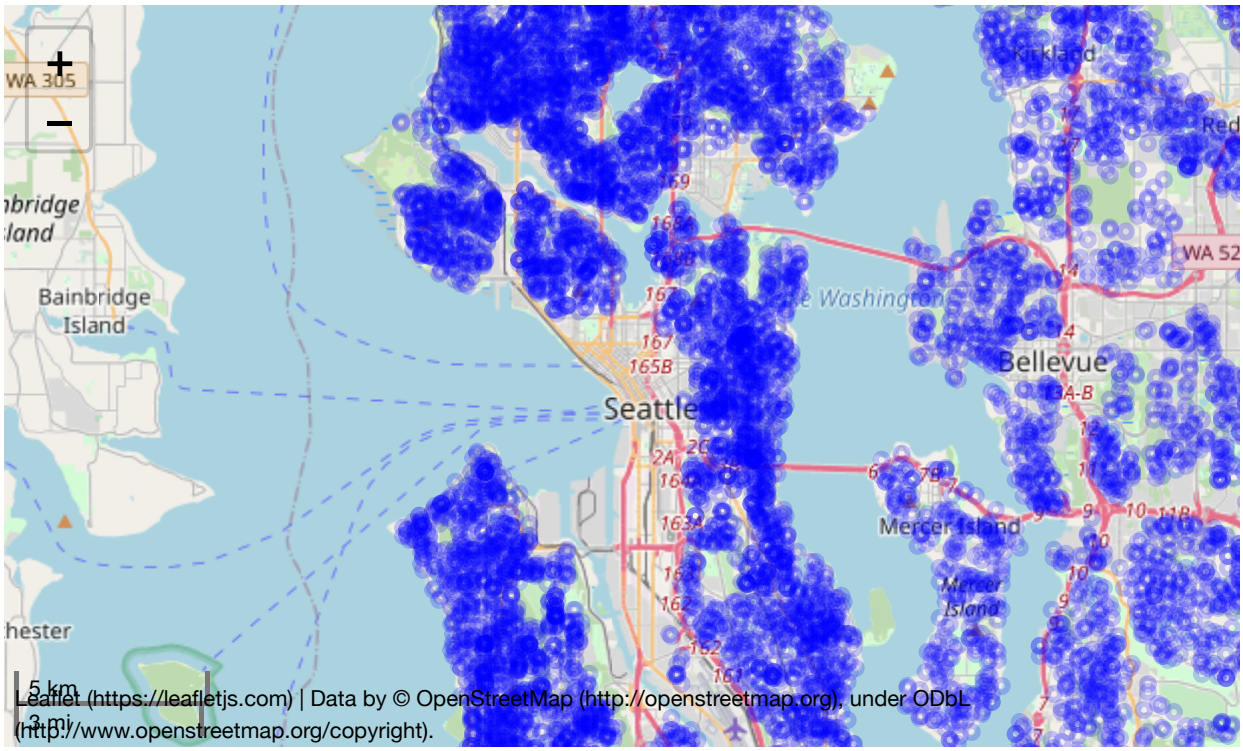
In [12]:
```
# Create Map: Basemap - OpenStreet Map
property_map = folium.Map(
    location=[df['lat'].mean(),
              df['long'].mean()],
    zoom_start=11,
    control_scale=True
)

for i in range(len(df)):

    folium.CircleMarker(
        location = [df.lat.iloc[i], df.long.iloc[i]],
        radius = 3,
        popup = df.cid.iloc[i],
        color = 'blue',
        opacity = 0.2
    ).add_to(property_map)

property_map
```

Out[12]:



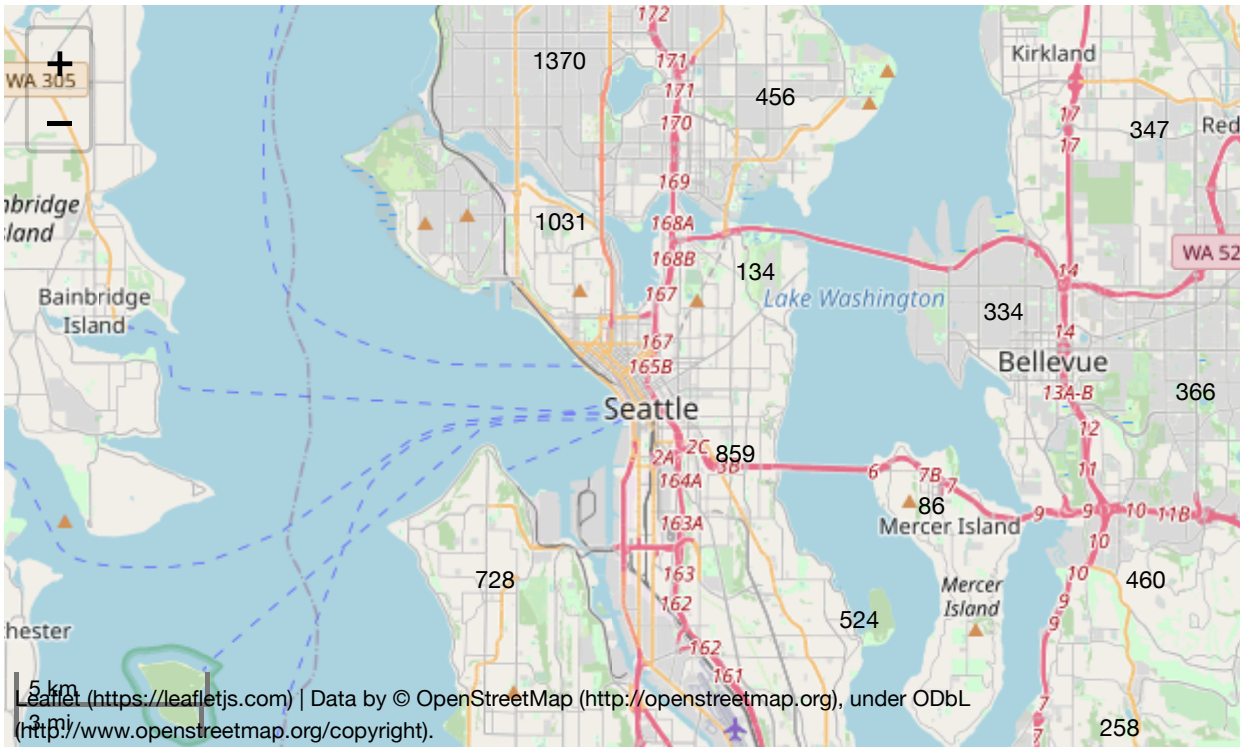
Cluster Map

In [13]:
```
# Create Map with clustering: Basemap - OpenStreet Map
property_map = folium.Map(
    location=[df['lat'].mean(),
              df['long'].mean()],
    zoom_start=11,
    control_scale=True)

mc = MarkerCluster()

#creating a Marker for each point in dataframe. Each point will get a popup
for row in df.itertuples():
    mc.add_child(folium.Marker(location=[row.lat,row.long], popup=row.cid))
property_map.add_child(mc)
property_map
## Other background Maps can be added. Options are
    # "OpenStreetMap" -- default option
    # "Mapbox Bright" (Limited levels of zoom for free tiles)
    # "Mapbox Control Room" (Limited levels of zoom for free tiles)
    # "Stamen" (Terrain, Toner, and Watercolor)
    # "Cloudmade" (Must pass API key)
    # "Mapbox" (Must pass API key)
    # "CartoDB" (positron and dark_matter)
```

Out[13]:



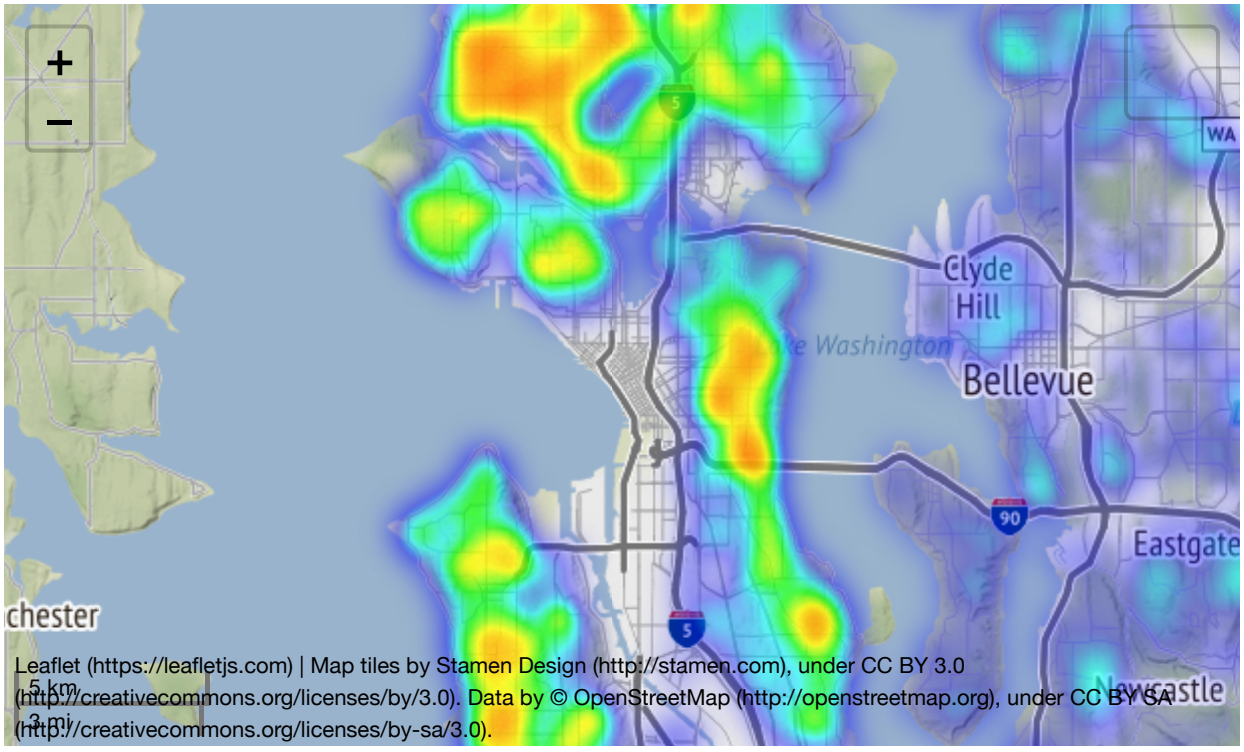
Heatmap

In [14]:
```
# Create HeatMap
from folium.plugins import HeatMap
property_map = folium.Map(
    location=[df['lat'].mean(),
              df['long'].mean()],
    tiles='Stamen Terrain',
    attr='Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMa
    zoom_start=11,
    control_scale=True
)

df['count'] = 1
property_heatmap = HeatMap(
    data=df[['lat', 'long', 'count']].groupby(['lat', 'long']).sum().reset_
    name = 'Heatmap',
    radius = 10,
    min_opacity = 0.1,
    max_zoom=16,
    opacity = 10
).add_to(property_map)

folium.LayerControl().add_to(property_map)
property_map
```

Out[14]:



In [ ]: