An EDA Case Study: Real Estate in Metro Vancouver, BC, Canada

VANCOUVER LOOKOUT AT HARBOUR CENTRE TOWER, Source: [https://www.AAA.com]

Author
Kasra Heidarinezhad

Last Update
Nov-10-2022

Last Data Capture
Jan-16-2023

Contents

Section 1 Introduction

Nowadays, finding an affordable house in Canada is very difficult. Study of Google Trends data[1] shows that British Columbia, especially Metro Vancouver, BC got higher ranking in search **Vancouver real estate** keywords between searchers among all provinces in Canada [Picture 1].

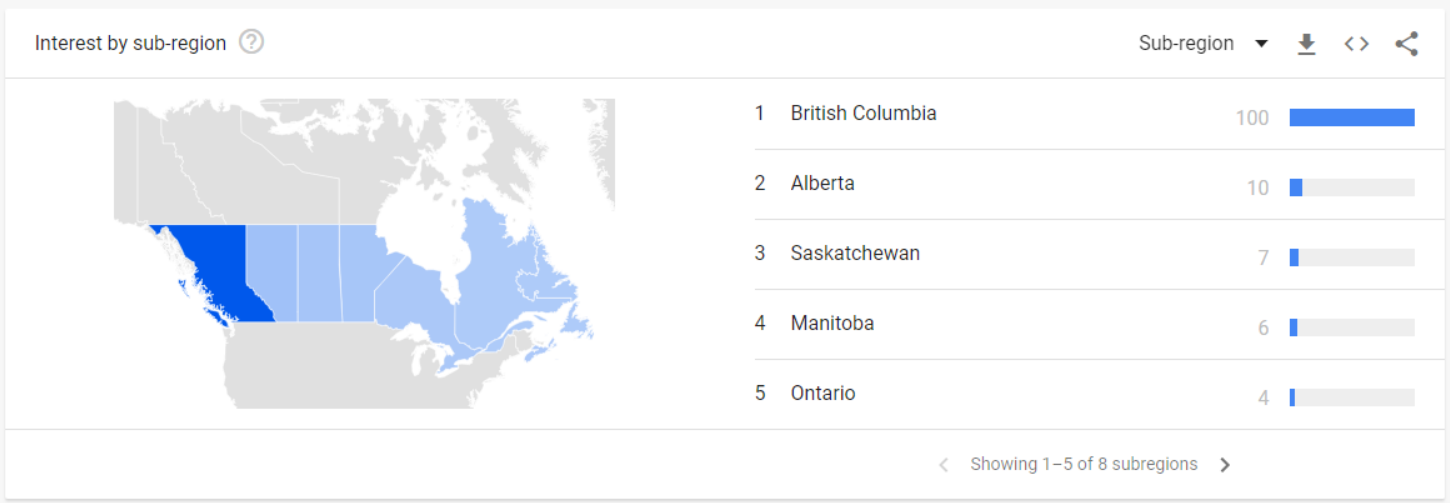


Figure 1 Google Trends output for "Vancouver real estate" comparison in Canada. (Screenshot by the author)

According to Statista[2] , Vancouver, BC is ranked seventh among the world's most expensive residential property markets in 2020, that making it difficult for buyers to find an affordable home. Consequently, this project was initiated to analyze the real estate market in Vancouver, British Columbia, Canada, by retrieving listing details from one of the available resources. The objective of this work is to analyze the housing market, visualize relevant statistics, and ultimately predict the sales price of a given listing using statistical data modeling.

Section 2 Creating data source, data cleaning, missing values

The initial step in this project was to obtain data for analysis. There were two potential methods: linking to a government agency or real estate license, although this data was not up-to-date and did not contain the necessary information such as individual item prices; or scraping data. This article focuses on the latter approach. The next challenge was identifying the best resource from the existing options. To select the optimal choice, two factors were taken into consideration: the comprehensiveness of the data provided by a website, and the difficulty of accessing the data within the website.

After conducting a thorough study and taking the aforementioned factors into consideration, I observed that popular real estate listing websites like Zillow[3] and Realtor[4] only allow access to the first 800 records, which is not sufficient for our data modeling in the last step. So we decided to work with the metrovancouverhomefinder.com[5].

[1] https://trends.google.com
[2] https://www.statista.com/statistics/1040698/most-expensive-property-markets-worldwide
[3] https://Zillow.com
[4] https://www.realtor.ca
[5] https://www.metrovancouverhomefinder.com

2-1 Web Scraping with Python

We made use of Google Colab, a free cloud-based platform for machine learning and data science experimentation. This platform allows users to write and run code, as well as share and collaborate on projects in a Jupyter notebook setting. Free access to GPU and TPU resources is provided, making it an attractive choice for deep learning model training and experimentation. Additionally, Colab integrates with popular libraries and frameworks such as TensorFlow, Keras, and PyTorch, and has the ability to easily import data from Google Drive or Github.

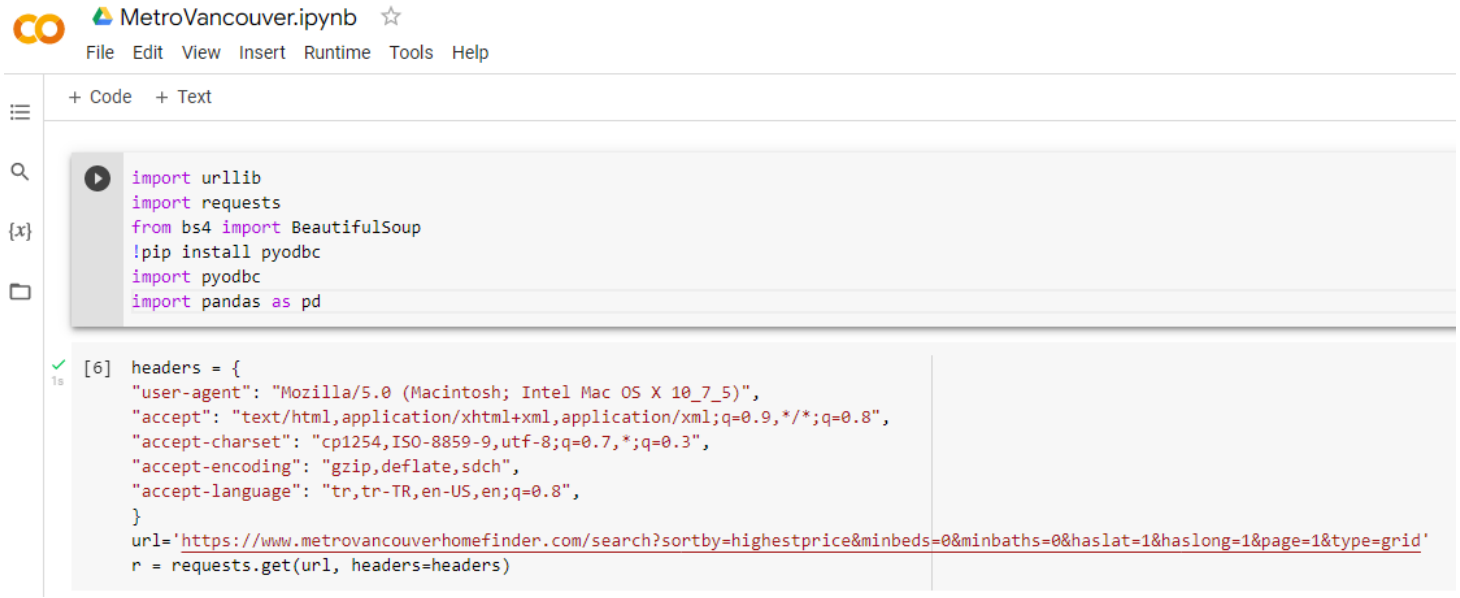
Required Libraries

Python provides libraries, such as Scrapy and BeautifulSoup, which are specifically designed for web scraping. In this instance, I chose to use the first one due to its simplicity. To make the process easier, we can compile a list of all of the necessary libraries in a file called RequiredLibraries.txt, and then use the command 'pip install RequiredLibraries.txt' to install them.

```
import os
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import warnings
import numpy as np
import pandas as pd
import lxml
from lxml.html.soupparser import fromstring
import prettify
import htmltext
import requests
import re
import json
```

Web Headers

Metrovancouverhomefinder utilizes Captchas when attempting to execute a request.get(url) type of function. To bypass this, headers can be added to the request function, as demonstrated below.



Furthermore, following step is done to complete job:

- Parse data from urls in looping through pages
- Create and append Data Frames
- Export Data frame to csv file

Here is complete code and result:

```
if r.status_code != 200:
    print("request denied")
else:
  result = r.content
  addlist = []
  arealist = []
  soup = BeautifulSoup(result)
  add_list = [i.text.strip() for i in soup.find_all('div', {'class': 'property-address'})]
  i=0
  while i < len(add_list):
     addlist.append(add_list[i])
     i = i+1
     arealist.append(add_list[i])
     i= i+1
  pricelist = [i.text.strip() for i in soup.find_all('div', {'class' :'property-price'})]
  bdbalist = [i.text.strip() for i in soup.find_all('div', {'class': 'text-right property-bdba'})]
  brokerlist = [i.text.strip() for i in soup.find_all('div', {'class': "flex2 property-mls"})]
  property_list = pd.DataFrame({
     'address': addlist,
     'area': arealist,
     'price': pricelist,
     'bdba': bdbalist,
     'broker': brokerlist   })

property_list.head()
```

	address	area	price	bdba	broker
0	34687 Ferndale Avenue, Mission, BC V2V 7C8	-- SqFt	$90,000,000	-- Beds | -- Baths	Listing Provided By Royalty Group Realty Inc.
1	4838 Belmont Avenue, Vancouver, BC V6T 1A9	12,410 SqFt	$59,800,000	5 Beds | 8 Baths	Listing Provided By Macdonald Realty
2	5462 Stonebridge Drive, Whistler, BC V8E 0V9	8,700 SqFt	$39,000,000	6 Beds | 10 Baths	Listing Provided By Engel & Volkers Whistler
3	730 Fairmile Road, West Vancouver, BC V7S 1R2	14,759 SqFt	$36,968,000	8 Beds | 11 Baths	Listing Provided By Rennie & Associates Realty...
4	4788 Belmont Avenue, Vancouver, BC V6T 1A9	10,548 SqFt	$33,888,000	7 Beds | 11 Baths	Listing Provided By Sutton Group-West Coast Re...

```
[ ] import csv
    from google.colab import drive
    import pandas as pd
    import time

    #time.sleep(10) # Delay for 10 seconds.
    drive.mount('drive', force_remount=True)

    property_list.to_csv('metrovan.csv', mode='a', header=False)
    !cp metrovan.csv "drive/My Drive/"
```

```
Mounted at drive
```

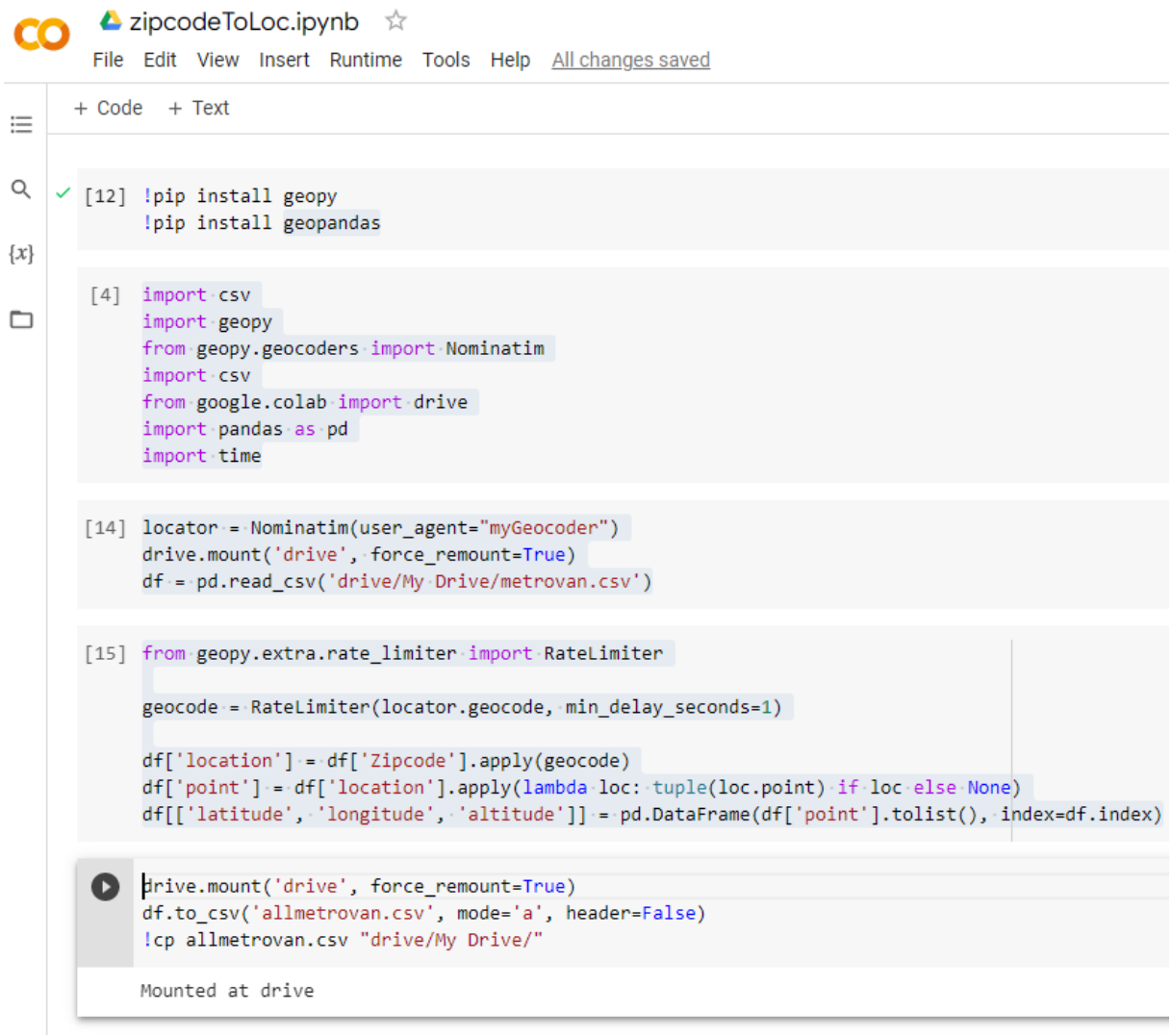
Result: raw data includes 5 columns e.g. address, area, price, bdba and broker and **4000** rows.

Note:

It is important to bear in mind that the website makes use of anti-scraping techniques such as captchas, IP blocking, and honeypot traps in order to protect its data from scraping. It is likely that our IP/device has been blacklisted at this point, but we have successfully completed the first step.

2-2 Data Cleaning and Preprocessing

Data cleaning involves the identification and rectification of any errors or inconsistencies in the data, such as missing values, duplicated records, or incorrect data types. This is a critical step, as it helps to guarantee that the data is complete and accurate, which is essential for constructing dependable models. To this end, we have divided the address attribute into three fields: address, city, and zip code; additionally, we have split the bdba field into two fields: bed and bath, which respectively represent the number of beds and baths; furthermore, we have updated the original data frame by using the *Geopy* library to locate the coordinates of the zip code by adding two new columns: latitude and longitude. We also converted the string types of *price* and *area* to *float* so that we could perform statistical functions on them later.
Here is complete code and result:



2-3 Missing Values

It is essential to substitute missing information with NaN (Not a Number) in order to facilitate the identification of absent values in our data and permit the implementation of appropriate measures expeditiously. Here, the missing values in the columns Town, ZipCode, Broker, Beds, Area, Price, Baths, Latitude and Longitude are addressed by filling it with NaN values.
After data cleaning, shape of data is like result:

```
drive.mount('drive', force_remount=True)
df = pd.read_csv('drive/My Drive/metrovan1.csv')
```

```
Mounted at drive
```

	index	Address	Town	Zipcode	Latitude	Longitude	Area	Price	Beds	Baths	Broker
0	1	111 32085 George Ferguson Way	Abbotsford	V2T 2K7	49.055312	-122.336344	1000	449000	2	2	RE/MAX Westcoast
1	2	20617 102b Avenue	Langley	V1M 3H3	49.190194	-122.651115	1000	3000000	2	1	Homelife Advantage Realty (central Valley) Ltd.
2	3	211 E 38th Avenue	Vancouver	V5W 1H3	49.236586	-123.100463	1000	7100000	3	1	Sutton Group-West Coast Realty
3	4	209 1802 Duthie Avenue	Burnaby	V5A 2R8	49.267361	-122.951989	1001	549900	2	2	RE/MAX Crest Realty
4	5	302 2151 151a Street	Surrey	V4A 7C6	49.040312	-122.802964	1002	655000	2	2	Oakwyn Realty Ltd.

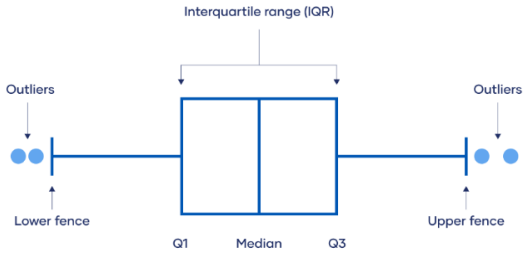
```
[8] df.head(5)
    print(f"Number of samples: {df.shape[0]}")
    print(f"Number of features in dataset: {df.shape[1]}")
    print("Features:")
    print(df.dtypes)
```

```
Number of samples: 3802
Number of features in dataset: 11
Features:
index          int64
Address       object
Town          object
Zipcode       object
Latitude     float64
Longitude    float64
Area           int64
Price          int64
Beds           int64
Baths          int64
Broker        object
dtype: object
```

2-4 Outliers

Outliers are values that are exceptionally distinct from the majority of the data points in a dataset. They can drastically alter the results of any statistical analyses and influence the results of any hypothesis tests. In this work, we utilize the interquartile range (IQR) to identify the range of the middle half of the dataset based on the ***price*** attribute. Additionally, we use the IQR to create *fences* around our data and define outliers as any values that lie outside these boundaries.

In continue we remove any erroneous data while keeping genuine extreme values. It involves deleting anomalous values from a dataset before conducting our statistical analysis. The majority of outliers are located above the upper fence, which is attributed to records related to *land* real estate.



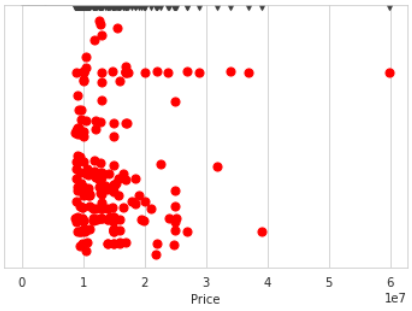
```
# Calculate first and third quartile
first_quartile = df['Price'].quantile(0.25)
third_quartile = df['Price'].quantile(0.75)
# Find interquartile range
IQR = third_quartile - first_quartile
# Find lower and upper bound
lower_bound = first_quartile - (1.5 * IQR)
upper_bound = third_quartile + (1.5 * IQR)

# filtering dataframe
data_outlier = df[(df.Price < lower_bound) | (df.Price > upper_bound)]

# plotting boxplot
sns.boxplot(x=df.Price)

# plotting outliers
plt.scatter(data_outlier.Price, data_outlier.index, color='red', s=50)
```

```
<matplotlib.collections.PathCollection at 0x7f2d2d7cef40>
```




```
plt.rcParams.update({'figure.figsize':(5,7), 'figure.dpi':100})
plt.boxplot(df['Price'], notch=True)
plt.title('Outliers boxplot')
plt.xlabel('Price')
plt.ylabel('Property')
plt.show()
```

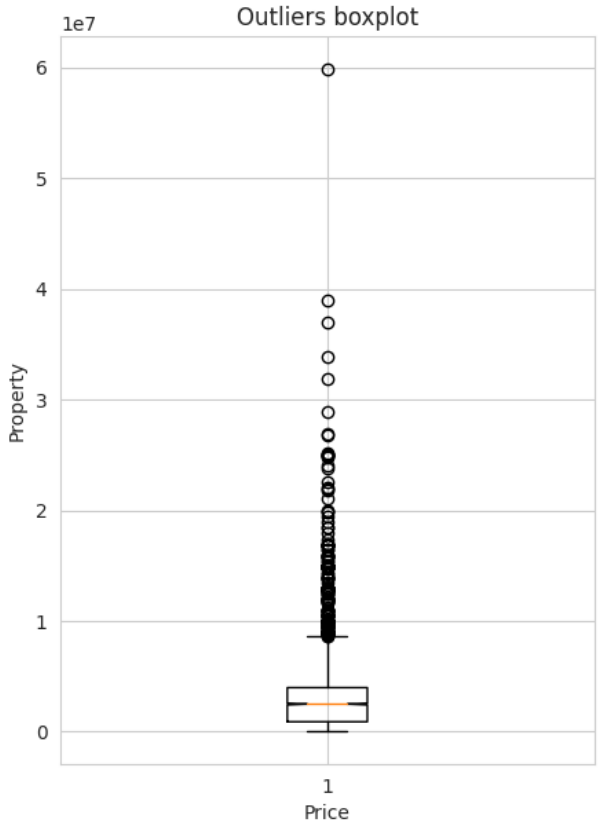


Figure 2 Outliers based on price of real estate in metro Vancouver, BC. (Image by the author)

Section 3 Data visualization

We conducted two forms of data visualization:

1- Data visualization with Python libraries
2- Data visualization with Tableau

3-1 Data visualization with Python libraries

Here we did some codding to create a shape of data and histogram. First of all we convert towns as category attribute to numerical attribute.

```
# Converting Town as a category attribute to numbers
df.Town = pd.Categorical(df.Town)
df['Towncode'] = df.Town.cat.codes

sns.pairplot(df, height=2)
plt.tight_layout()
```

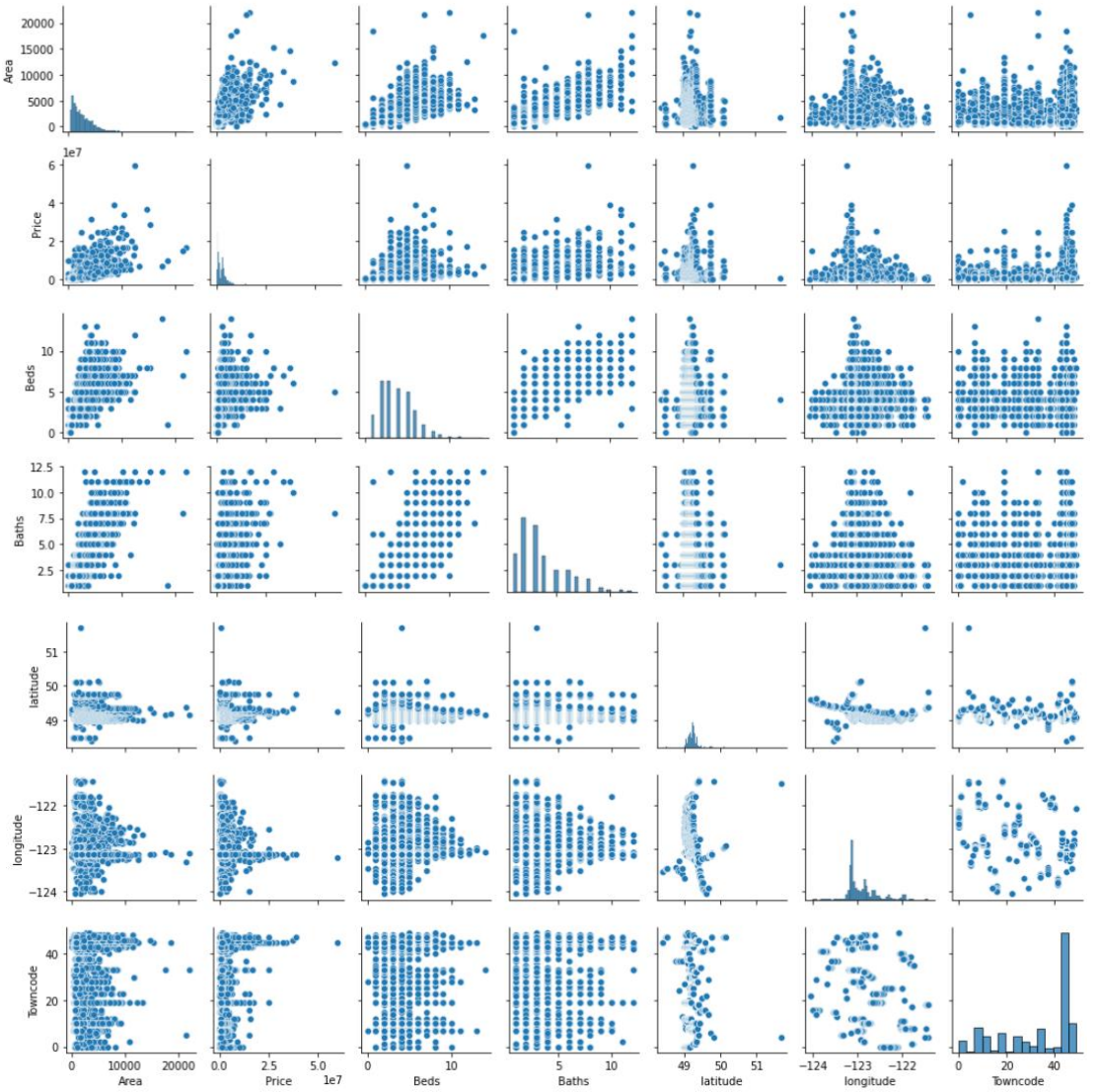


Figure 3 Pairplot of dataset (Image by the author)

Consequently, we draw most important statistics feature i.e. Histogram, extracted by following code, result in [Figure 4]

```
import matplotlib.ticker as tkr

#sns.set_theme()
plt.figure(figsize=(26,9)) # figure ration 16:9
sns.set()
# Modify histogram with bins
graph = sns.displot(data=df, x="Price", kde=True, kind='hist', rug=True, log_scale=True, bins=55, height=8.27, aspect=11.7/8.27)
graph.set(title="Histogram of Sale Price")
for ax in graph.axes.flat:
    ax.xaxis.set_major_formatter(tkr.FuncFormatter(lambda x, p: format(int(x), ',')))
plt.xticks(rotation=30)
plt.show()
```

Result:

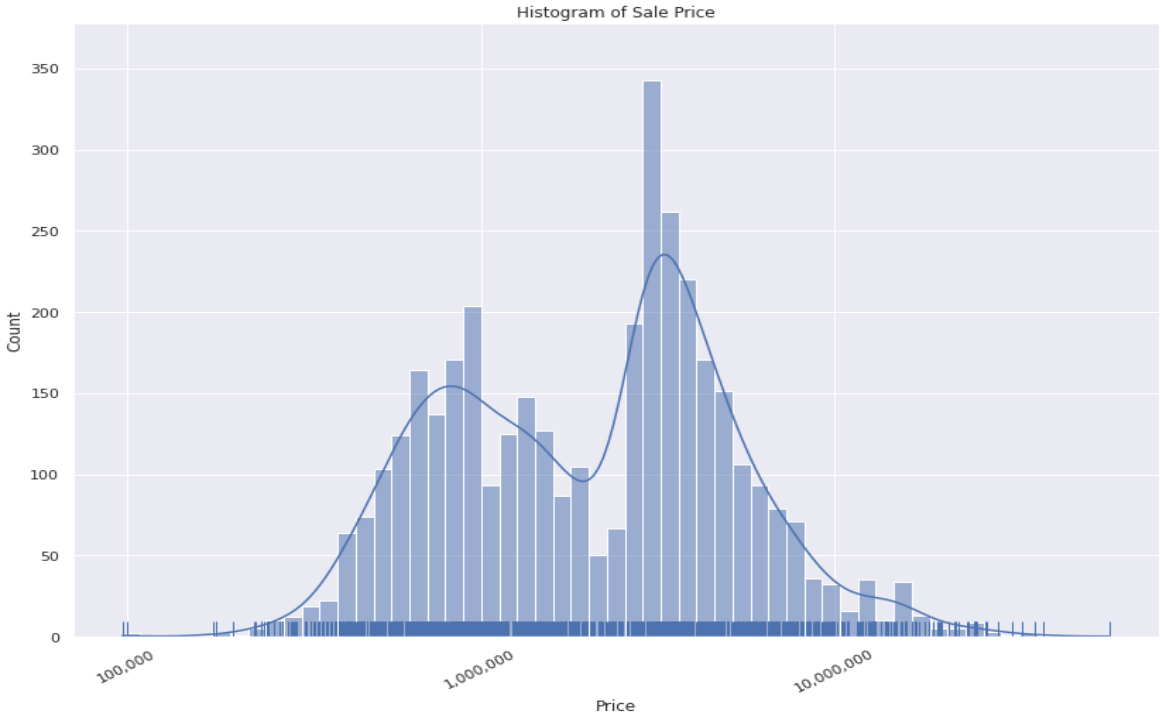


Figure 4 Histogram of real estate price in metro Vancouver, BC. (Image by the author)

Let's have a look in to the other four major statistic parameter:

```
print(f"Number of samples: {df.shape[0]}")
print(f"Number of features in set: {df.shape[1]}")
print("Features:", df.columns.values)
#print(df.columns.values)
Mean = df["Price"].mean()
Median = df["Price"].median()
Skewness = df["Price"].skew()
Kurtosis =df["Price"].kurtosis()
print(f"Mean: {Mean:,.2f}")
print(f"Median: {Median:,.2f}")
print(f"Skewness: {Skewness:.4f}")
print(f"Kurtosis: {Kurtosis:.4f}")
```

```
Number of samples: 3809
Number of features in set: 10
Features: ['Address' 'Town' 'ZipCode' 'Area' 'Price' 'Beds' 'Baths' 'Broker' 'latitude' 'longitude']
Mean: 3,121,455.16
Median: 2,498,000.00
Skewness: 4.2408
Kurtosis: 35.1458
```

Discussion

It shows the distribution of the price of properties in Vancouver, BC, is relatively high positive skewness (the mean is greater than the median). In the prediction model, this will cause, the less accurate in the price prediction. In the other hand, high kurtosis represents heavy tails meaning more outliers.

Correlation:

Correlation is a statistical measure used to quantify the linear relationship between two variables, which is characterized by a constant rate of change. This measure is commonly employed in order to describe the relationship between two variables without inferring causality.

```
plt.figure(figsize=(8,8))
cor = df.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.PuBu)
plt.show()
```

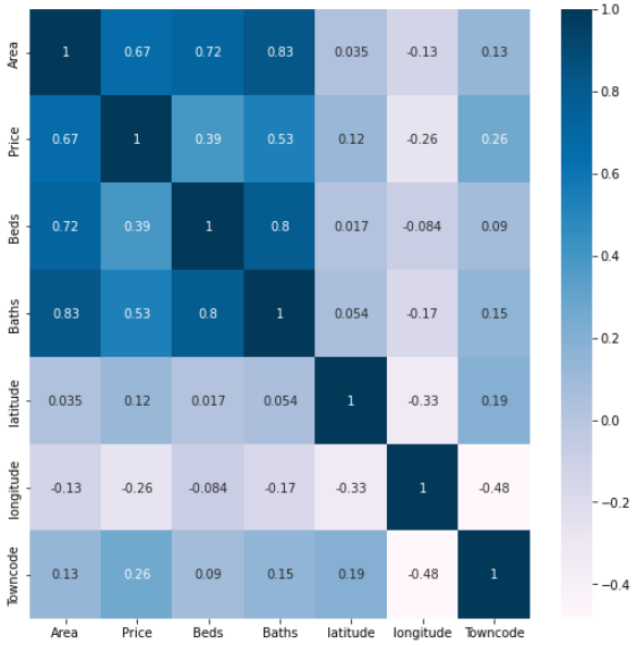


Figure 5 Correlation between attributes (Image by the author)

```
# Absolute value of correlation
cor_target = abs(cor['Price'])
# Highly correlated attributes
relevant_features = cor_target[cor_target>0.2]
# Getting the names of the attributes
names = [index for index, value in relevant_features.iteritems()]
# Erase the target attribute
names.remove('Price')
# printing the correlated attributes
print(names)
print(len(names))
```

```
['Area', 'Beds', 'Baths', 'longitude', 'Towncode']
5
```

More statistic parameter

Statistics per category: P**roperty per town**

Let's take a look to distribution of numbers in listing base of town:

```
from pandas._libs.hashtable import value_count
df['Town'] = df['Town'].str.strip()
type(df['Town'][0])
print('Number of real estate per individual town in the listing: ',df['Town'].nunique(),'\n')
df['Town'].value_counts()
```

```
Number of real estate per individual town in the listing:  50

Vancouver                1049
Surrey                    516
Richmond                  296
West Vancouver            242
Langley                   236
Burnaby                   230
Coquitlam                 134
North Vancouver           133
Abbotsford                133
Maple Ridge               108
Chilliwack                 98
Whistler                   77
Delta                      73
White Rock                 58
Sardis                     52
Mission                    49
New Westminster            49
Squamish                   40
Port Coquitlam             33
Port Moody                 31
Pitt Meadows               21
Sechelt                    17
Gibsons                    16
```

The result suggest, the greatest number of properties in our dataset belong to Vancouver as a considerable portion of our listings.

2- Statistics per category: **Activity of brokers**

Here we did some coding to see distribution of numbers in listing base of brokers:

```
from pandas._libs.hashtable import value_count
df['Broker'] = df['Broker'].str.strip()
type(df['Broker'][0])
df['Broker'].nunique()
df['Broker'].value_counts()
```

```
RE/MAX Crest Realty                    240
Sutton Group-West Coast Realty         185
Royal Pacific Realty Corp.             117
Nu Stream Realty Inc.                   96
Macdonald Realty                        92
                                      ...
Emily Oh Realty                          1
Westmont Realty Inc.                     1
Vantage First Realty                     1
Oak West Realty Ltd.                     1
Engel & Volkers Vancouver (branch)       1
Name: Broker, Length: 261, dtype: int64
```

As results shows, 261 individual brokers are active in our listing means some of them created more than one record in dataset.

In continue, we plot top 30 realtor histogram, Figure 6. We used matplotlib.pyplot to extract more helpful plots.

```
# Histogram top 30 Realtor
from matplotlib.pyplot import figure

figure(figsize=(10, 8), dpi=80)
df['Broker'] = df['Broker'].str.strip()
graph = df['Broker'].value_counts()[:30]
graph.plot(kind='barh', fontsize=12, title="Top 30 listing-creator realtor")
plt.show()
```

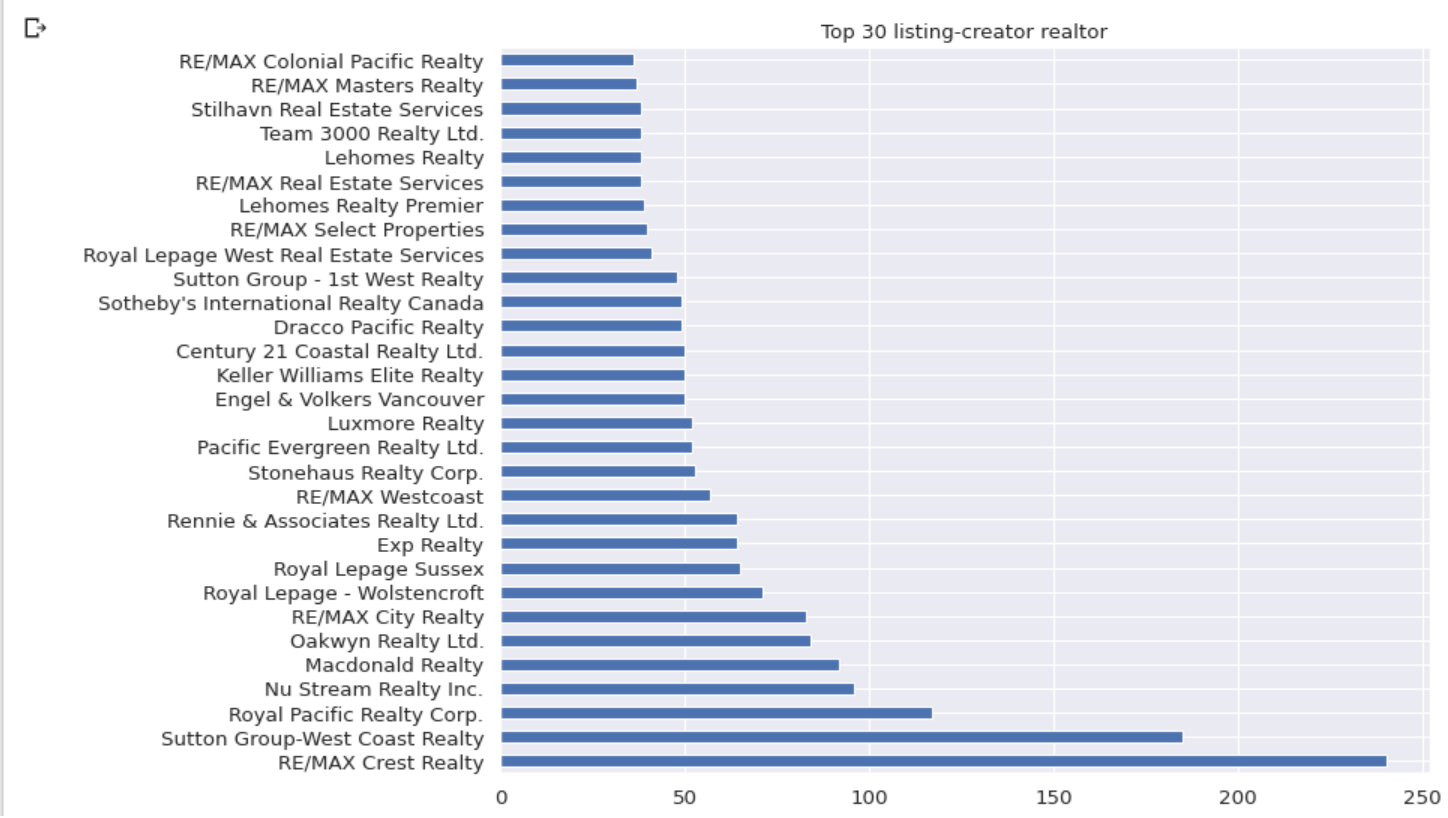


Figure 6 Top 30 realtor histogram (Image by the author)

3- Statistics per category: **Price range**

We continue by creating a price range histogram to observe number of properties. [Figure 7]

```
figure(figsize=(10, 8), dpi=80)
df = df1[['Price', 'Town' ]]
df['binned_price'] = pd.cut(df.Price, [1, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000, 10000000])
ax = df['binned_price'].value_counts().plot(kind='bar', stacked=True, ylabel='Frequency',\
                                            xlabel='Price binned',title='Price group frequency by Town', rot=90, fontsize=9)
```

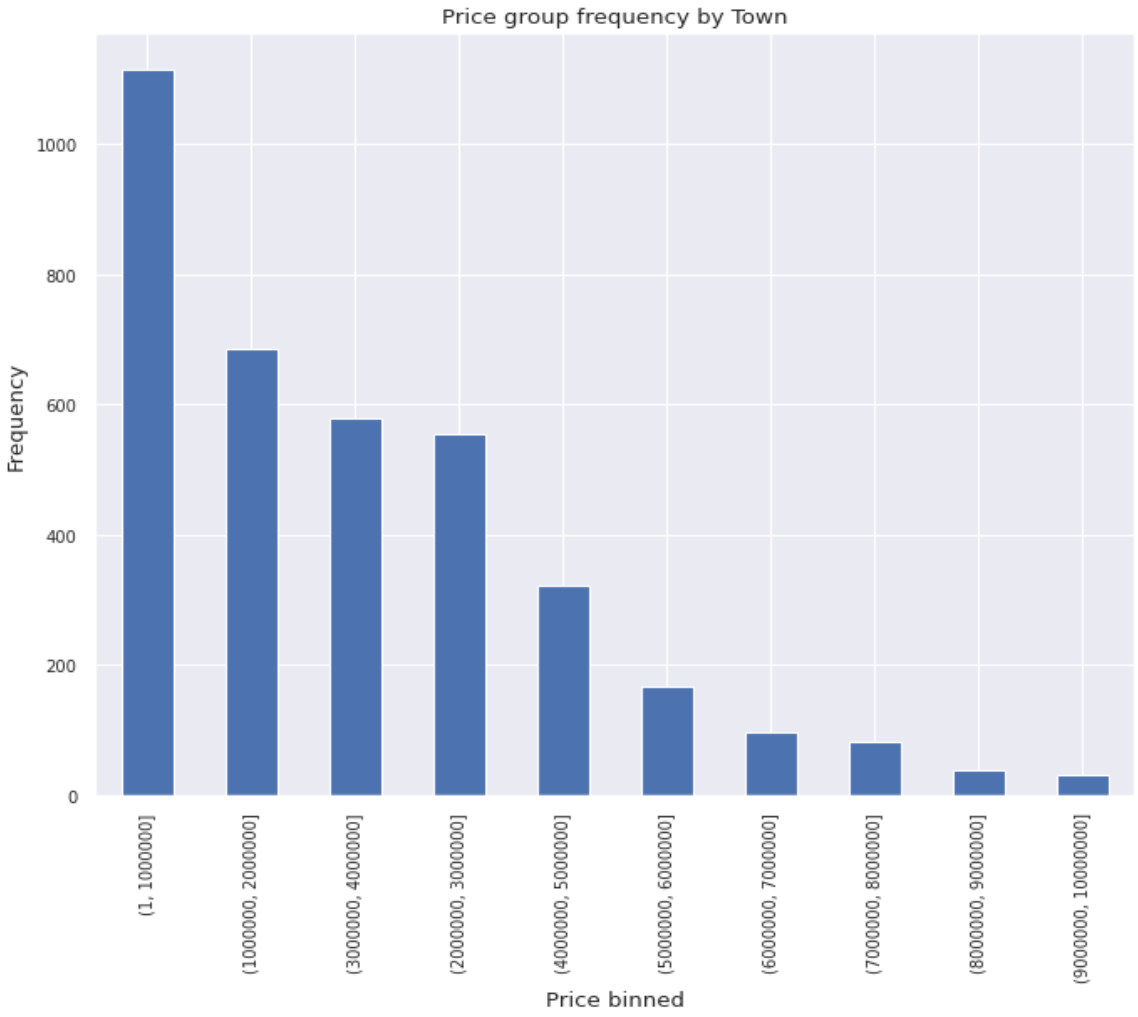


Figure 7 Histogram - Group by: price bin (Image by the author)

4- Scatter plot of distribution Statistics per category: **Area - number of range** [Figure 8]

```
df = pd.read_csv('drive/My Drive/MetroVan.csv')
plt.rcParams.update( {'figure.figsize':(12,6), 'axes.facecolor': 'White' , 'figure.dpi':150})
plt.scatter(x=df.Price, y=df.Area, c=df.Beds, cmap='Spectral')
plt.colorbar(label="Number of Beds")
plt.title('Relationships between Price and Area')
plt.xlabel('Price (CA$)')
plt.ylabel('Property Area (Square feet)')
current_values = plt.gca().get_xticks()
plt.gca().set_xticklabels(['{:,.0f}'.format(x) for x in current_values])
plt.rcParams['axes.facecolor'] = 'black'
plt.show()
```

Figure 8 Scatter plot of distribution of price based of area and number of beds (Image by the author)

5- Geographical property distribution category: `Map`

Finally, let us examine the geographical distribution of properties in the British Columbia. To do this, we employed the Folium library, but first it was necessary to undertake some preliminary processing to differentiate the categories of property. As can be observed, this considerable improved the performance of the visualization.

```
[1] !pip install git+https://github.com/python-visualization/folium
```

```
[2] import pandas as pd
    import folium
    from folium.plugins import MarkerCluster    # Import folium MousePosition plugin
    from folium.plugins import MousePosition    # Import folium DivIcon plugin
    from folium.features import DivIcon
    from google.colab import drive
    import branca.colormap as cm
```

```
[3] drive.mount('drive', force_remount=True)
```

```
[4] df = pd.read_csv('drive/My Drive/MetroVan.csv')
```

```
[5] #Checking latitude and longitude
    df = df.query(' -121 > longitude > -124')
    df = df.query('48 < latitude < 51')
```

```
[6] min_price=df.Price.min()
    max_price=df.Price.max()

    x_start = (df['latitude'].max() + df['latitude'].min()) / 2
    y_start = (df['longitude'].max() + df['longitude'].min()) / 2
    start_coord = (x_start, y_start)
```

```
[7] colormap = cm.LinearColormap(colors=['b','r'],
                                 index = [min_price, max_price],
                                 vmin=min_price,vmax=max_price)

    colormap.caption = 'Color based on price'
```

```
[8] map = folium.Map(location=start_coord, zoom_start=12)

    lat = list(df.latitude)
    lon = list(df.longitude)
    dat = list(df.Price)
```

```
for loc, p in zip(zip(lat, lon), dat):
    folium.Circle(
        location=loc,
        radius=10,
        fill=True,
        color=colormap(p),
        fill_opacity=0.7
    ).add_to(map)

map.add_child(colormap)
```

Result:

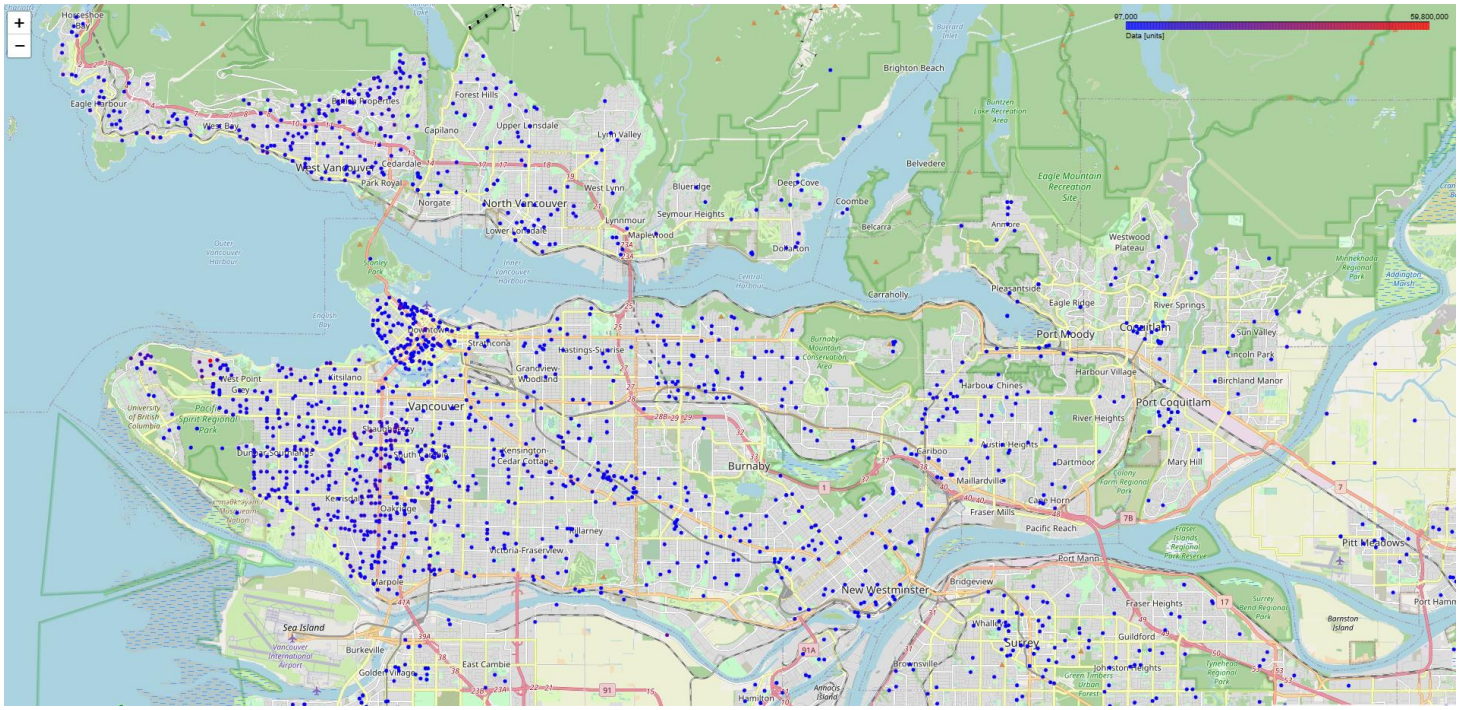


Figure 9 Geographical distribution of real estate in the Metro Vancouver map (Image by the author)

Here also we used plugins from folium library to show distribution of listing in the BC. as cluster view:

```
[ ]  from folium import plugins
```

```
[ ]  map = folium.Map(location=start_coord, zoom_start=12, width="%100",height="%100")

     locations = list(zip(df.latitude, df.longitude))

     cluster = plugins.MarkerCluster(locations=locations,
                    popups=df.Price.tolist())
     map.add_child(cluster)
```

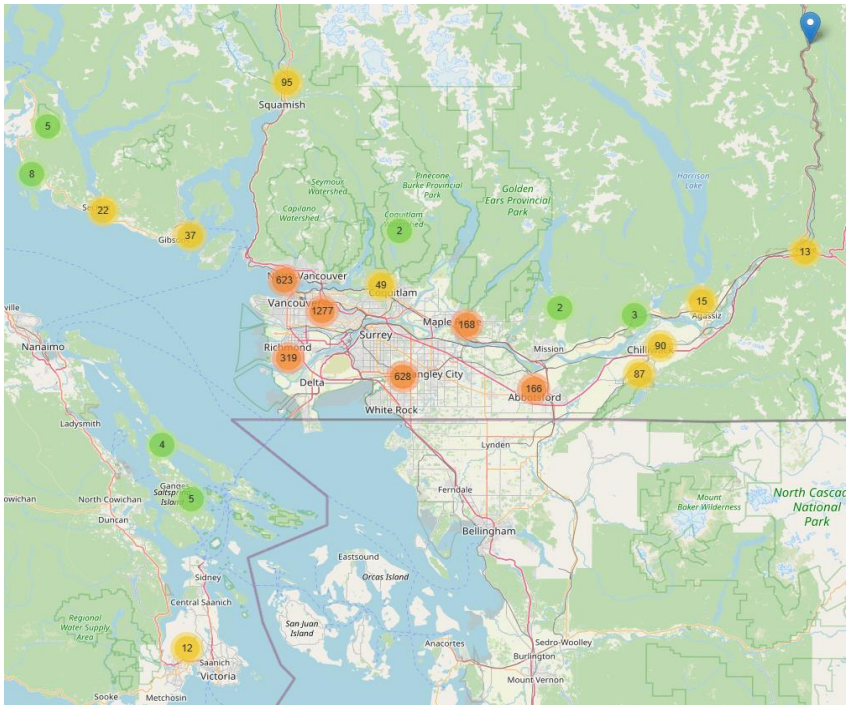


Figure 10 Cluster-based Geo distribution of real estate in the Metro Vancouver map (Image by the author)

6- Geographical Per Canada

This chart is not directly part of this project, but closely shows the ability of `matplotlib` in case of country distribution data.

```
can_map = folium.Map(location=[48, -102], zoom_start=3)   # Create the map object
c_data = {
  'Alberta': 144284.48,
  'British Columbia': 141222.06000000017,
  'Manitoba': 134337.96999999994,
  'New Brunswick': 115727.67000000001,
  'Newfoundland': 6885.140000000001,
  'Northwest Territories': 91755.44000000002,
  'Nova Scotia': 80136.18000000005,
  'Nunavut': 1506.4300000000014,
  'Ontario': 352263.50999999983,
  'Prince Edward Island': 28742.2,
  'Quebec': 138658.87999999998,
  'Saskatchewan': 177314.26000000013,
  'Yukon Territory': 74404.80000000003
}
```

```
folium.GeoJson(gdf).add_to(can_map)  # Add the GeoJSON data to the map

folium.Choropleth(
    geo_data=gdf,
    name="choropleth",
    data=c_data,
    columns=['Province', 'Profit'],
    key_on='feature.properties.NAME',
    fill_color="YlGn",
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name="Imaginary Data",
).add_to(can_map)

can_map
```

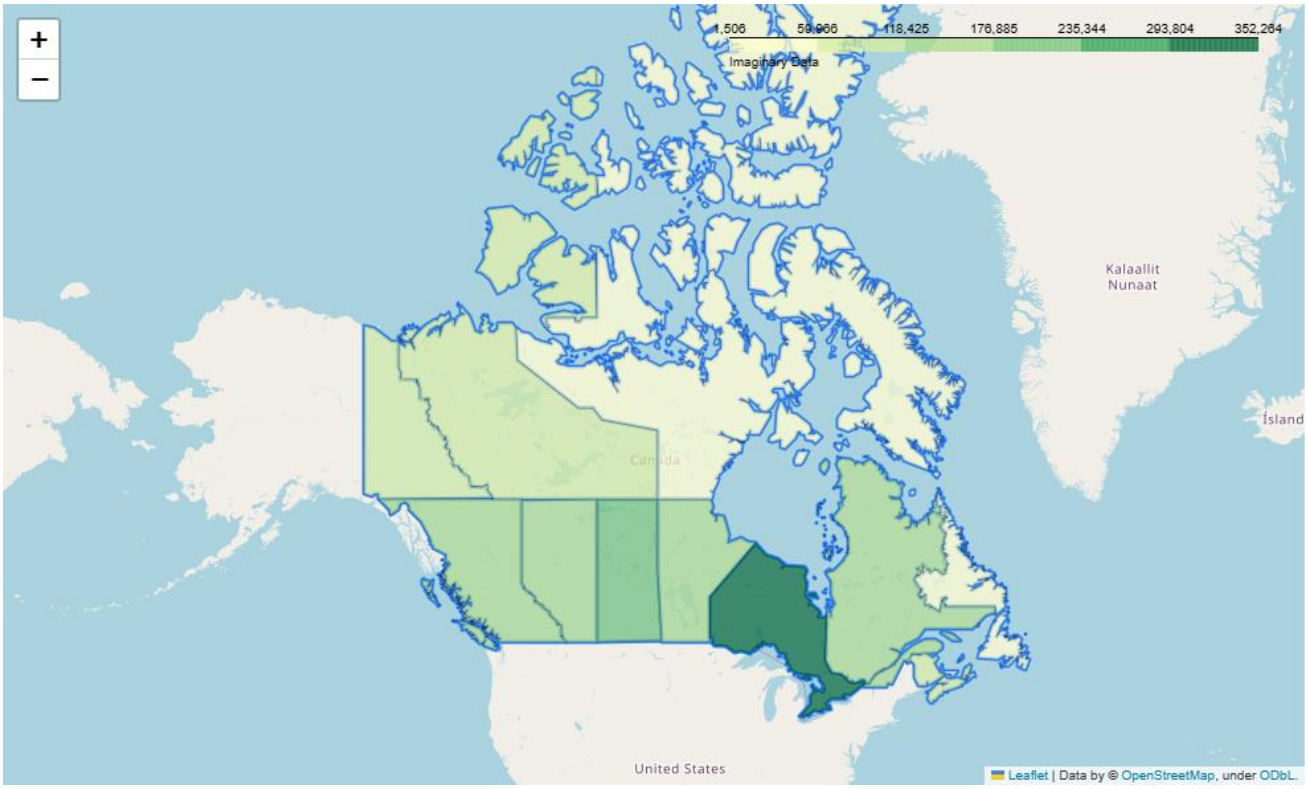


Figure 11 Geographical distribution in Canada categorized by province (Image by the author)

3-2 Data visualization with Tableau

Tableau is a data visualization and business intelligence software that helps users to see and understand their data. It allows to create interactive dashboards, charts, and maps to gain insights from their data. It also has built-in collaboration features, so teams can share their findings and work together on projects. Here we used Tableau (Public) to carry out some visualization.

Chart 1: Percentage of each type of property for sale based on percentage of all

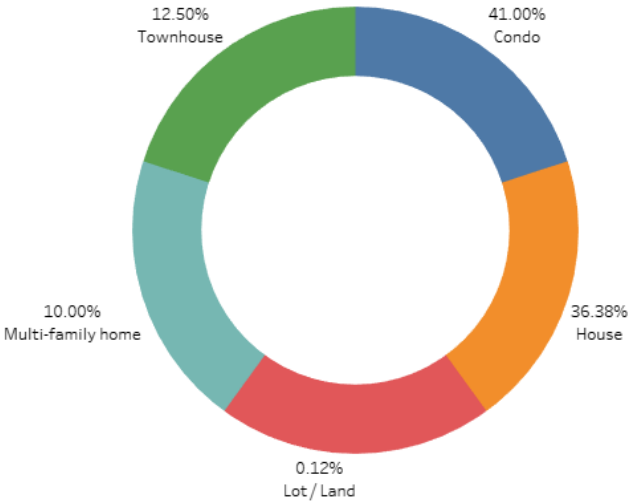


Figure 12 Donut chart shows the percentage of each type of property for sale based on percentage of all (Image by the author)[6]

Chart 2: Categorized histogram displays the distribution of listing Town on Price(bin)

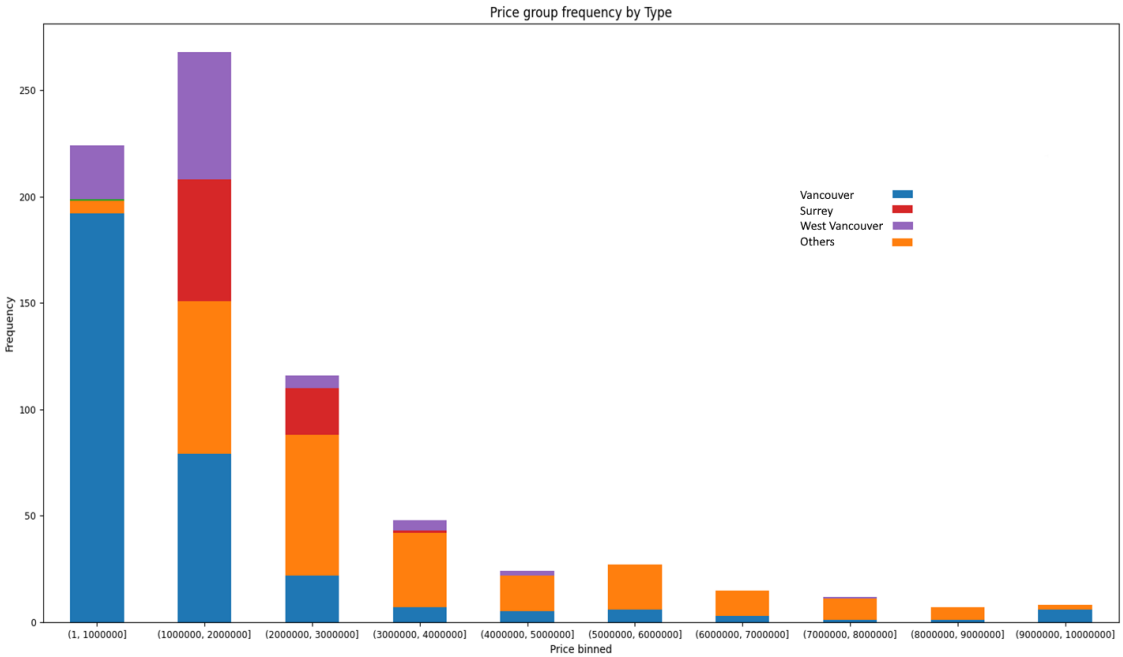


Figure 13 Histogram - Group by: Town / Price bin (Image by the author)[7]

[6] https://public.tableau.com/app/profile/kasra.heidarinezhad

[7] https://public.tableau.com/app/profile/kasra.heidarinezhad

Chart 3: Treemap view to demonstrations the distribution real estate based on price-range category

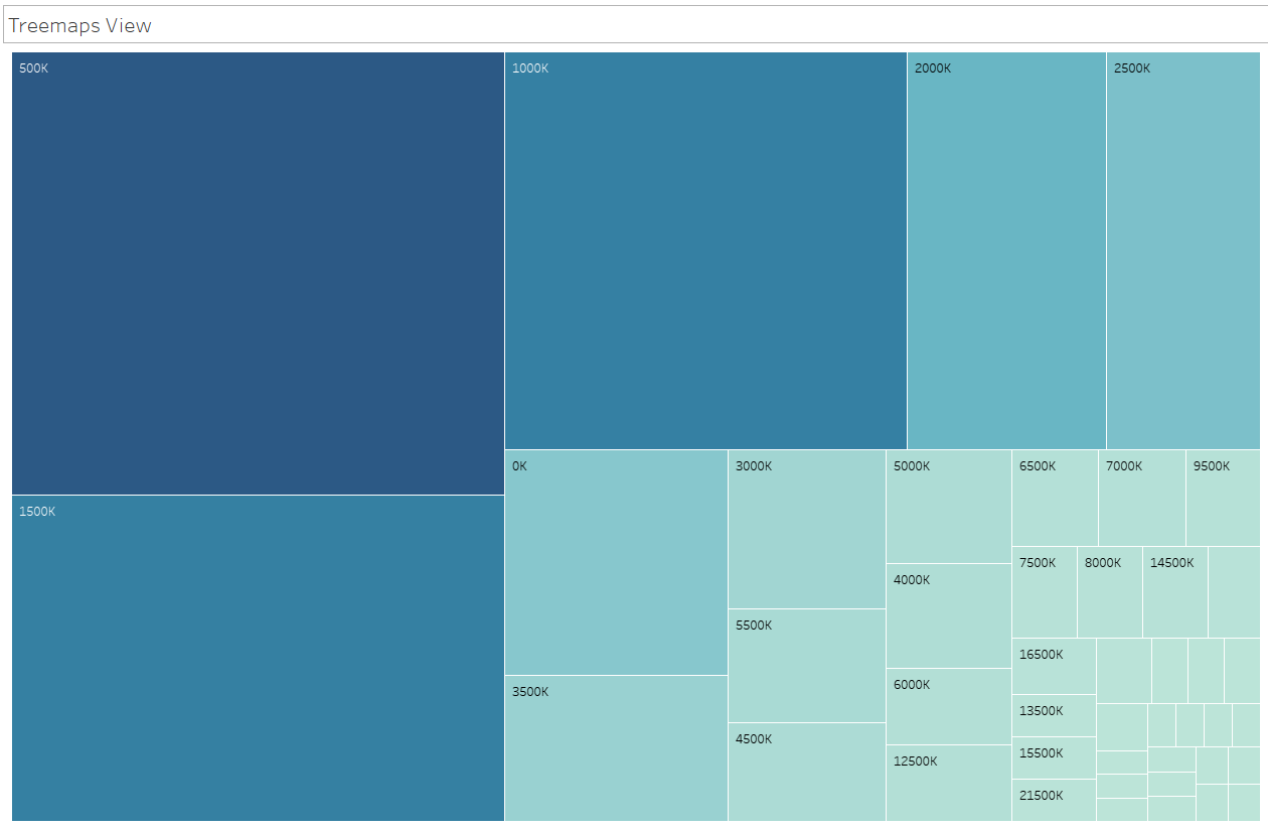


Figure 14 Treemap view shows the distribution based on price-range category (Image by the author)[8]

Chart 4: Average house price by number of items in listing. As evidenced, Vancouver has the highest number of houses listed in the database, yet West Vancouver remains the most expensive area within this listing.

[8] https://public.tableau.com/app/profile/kasra.heidarinezhad

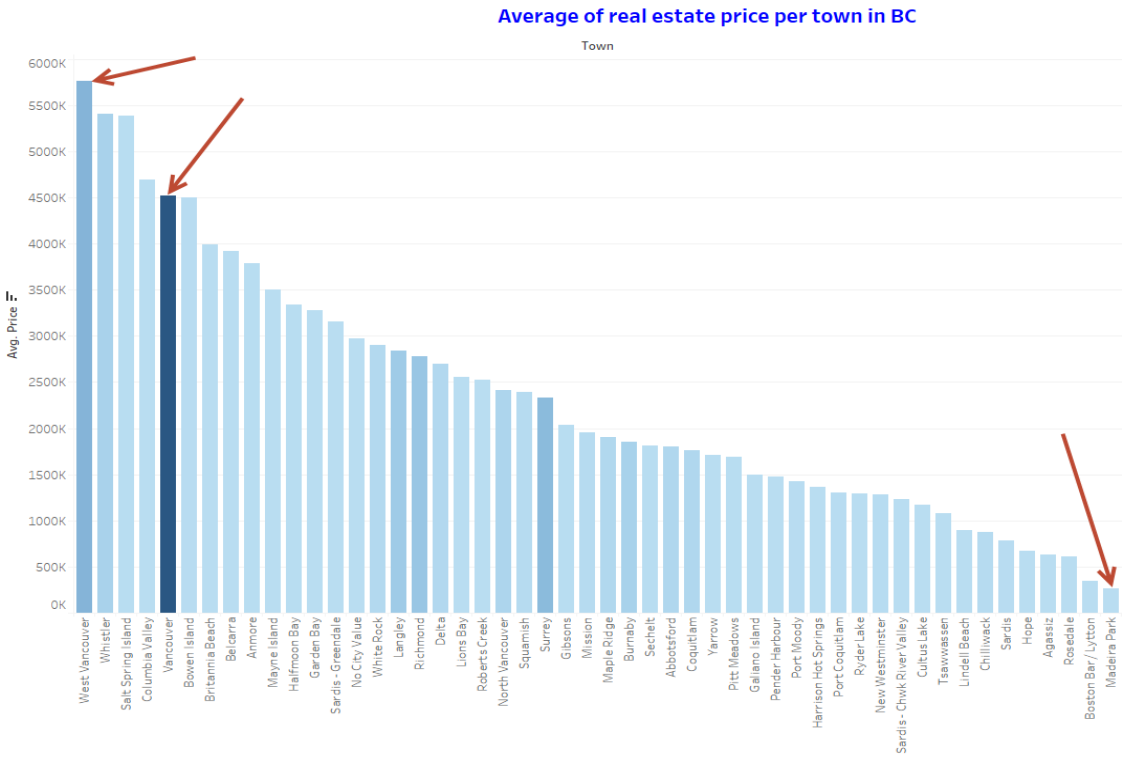


Figure 15 Categorized histogram view shows the distribution based on Price(range)-Type category (Image by the author)[9]

Chart 5: Average Categorized histogram view shows the distribution based on Avg Price(bin), Avg area per Town

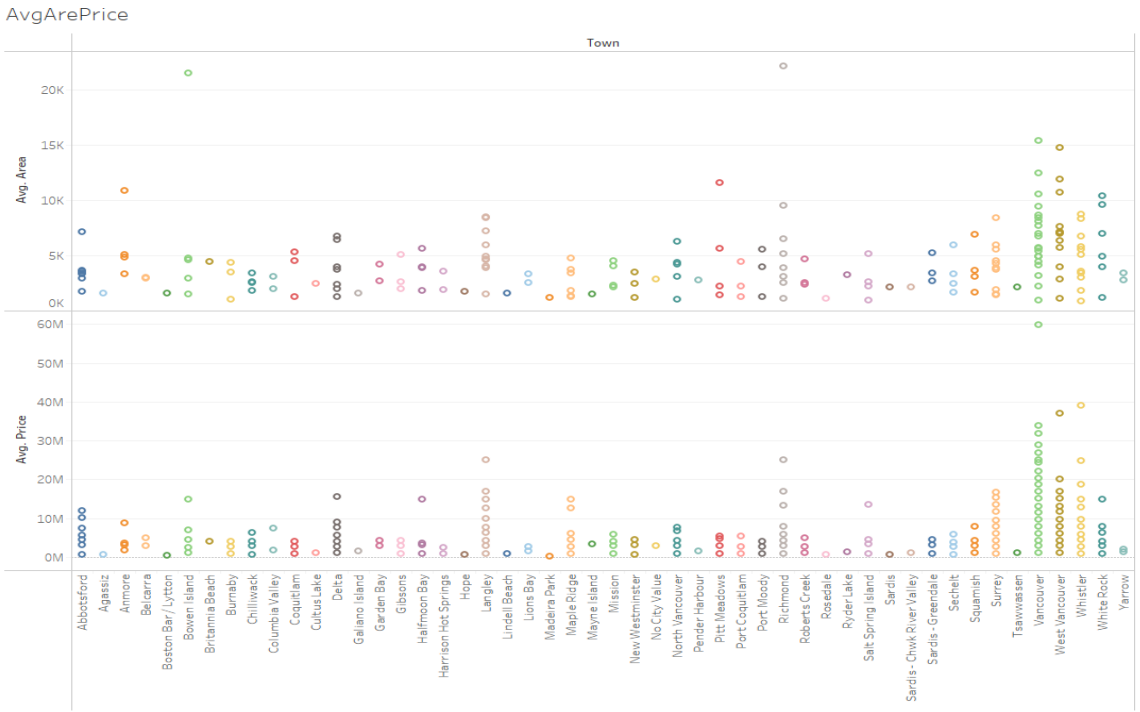


Figure 16 Categorized histogram view shows the distribution based on Avg Price(bin), Avg area per Town (Image by the author)[10]

[9] https://public.tableau.com/app/profile/kasra.heidarinezhad

[10] https://public.tableau.com/app/profile/kasra.heidarinezhad/

Section 4 Database, SQL and Query

To manipulate data, we created a database named: VancouverProperties.db with a table named: VancouverPropertiesTable with sqlite DBMS. Afterward, data is extracted by SELECT statement:

Query 1: A complete list of real estate in table

```
import sqlite3
from google.colab import drive
from pathlib import Path
```

```
drive.mount('drive', force_remount=True)
df = pd.read_csv('drive/My Drive/MetroVan.csv')
```

```
Path('MetroVan.db').touch()
```

```
db_conn = sqlite3.connect('VancouverProperties.db')
db_cursor = db_conn.cursor()
```

```
df.to_sql('VancouverPropertiesTable', db_conn, if_exists='append', index=False)
db_vancouverproperties_init_query = pd.read_sql(''' SELECT * FROM VancouverPropertiesTable ''', db_conn)
db_vancouverproperties_init_query.head(3)
```

	Address	Town	ZipCode	Area	Price	Beds	Baths	Broker	latitude	longitude
0	104 1139 W Cordova Street	Vancouver	V9C 0A2	3788	7870000	4	5	Exp Realty	48.3980	-123.5423
1	4868 Summit Lane	Whistler	V8E 1G4	4349	14995000	5	6	Whistler Real Estate Company Limited	48.4908	-123.4508
2	227 4800 Spearhead Drive	Whistler	V8E 1G1	790	1997000	2	2	RE/MAX Sea To Sky Real Estate	48.4908	-123.4508

Query 2: List of real estate by filtering: Kingsway, and sorting

```
data_stat = pd.read_sql(''' SELECT ZipCode, Area, Price, Beds, Address
                            FROM VancouverPropertiesTable
                            WHERE Address LIKE '%Kingsway%'
                            ORDER BY Price DESC ''', db_conn)
data_stat.head(4)
```

	ZipCode	Area	Price	Beds	Address
0	V5H 0E9	623	768000	1	2202 4688 Kingsway
1	V5R 0C3	771	698000	2	515 2689 Kingsway
2	V5R 5G8	554	619000	1	314 2435 Kingsway
3	V3N 0G9	556	579990	1	1203 7388 Kingsway

Query 3: Average price (CA$) and area (Square feet) of hose for sales in town = Richmond

```
data_stat_Avg = pd.read_sql(''' SELECT AVG(Price) AS Avg_price_richmond , AVG(Area) AS Avg_area
                                FROM VancouverPropertiesTable
                                WHERE Town LIKE '%Richmond%' ''', db_conn)

print(f"Average house price for sale in Richmond is : {data_stat_Avg.Avg_price_richmond[0]:,.2f} CA$")
print(f"Average area of houses for sale in Richmond is : {data_stat_Avg.Avg_area[0]:,.2f}  Sqfeet")
```

```
Average house price for sale in Richmond is : 2,783,492.50 CA$
Average area of houses for sale in Richmond is : 2,731.68  Sqfeet
```

Section 5 Statistical data modeling

Data modeling refer to the process of creating a mathematical representation of an existing data. This representation, known as a model, can be used to make predictions or decisions based on new or unseen data. In data science, statistical models and machine-learning models are two most important types of models that widely used. In this project, we going to first option.

One of most important goals of data modeling is prediction, in our case: prediction of housing price. Here we used multivariable Linear Regression (LR) to predict target variable, and used a filtered attributes as bellow:

1- Numerical attributes: Price, Beds , Area, Baths, Latitude, Longitude
2- Classified attribute: Town, Address, ZipCode, Broker

5-1 Linear Regression Modeling (LRM) using sklearn library

In this phase, we will involve the construction and training of six distinct linear regression models, namely:

1- Ordinary Least Squares (OLS) algorithm
2- Ridge algorithm
3- Lasso algorithm
4- Bayesian algorithm
5- Elastic Net algorithm
6- Support Vector Machine algorithm

To facilitate the implementation of these models, the pre-built algorithms provided by the scikit-learn Python package will be utilized. The process for each model will be consistent, comprising of three steps:

1- Definition of a variable to store the model algorithm
2- Fitting of the train set variables to the model
3- Prediction of values on the test set.

```
df = pd.read_csv('drive/My Drive/MetroVan.csv')
```

```
# Converting Town as a category attribute to numbers
df.Town = pd.Categorical(df.Town)
df['Towncode'] = df.Town.cat.codes

#print(df.describe())
print(df.dtypes)
```

```
Unnamed: 0       int64
Address         object
Town          category
ZipCode         object
Area             int64
Price            int64
Beds             int64
Baths            int64
Broker          object
latitude       float64
longitude      float64
Towncode          int8
```

```
# Absolute value of correlation
cor = df.corr()
cor_target = abs(cor['Price'])
# Highly correlated attributes
relevant_features = cor_target[cor_target>0.2]
# Getting the names of the attributes
names = [index for index, value in relevant_features.iteritems()]
# Erase the target attribute
names.remove('Price')
# printing the correlated attributes
print(names)
print(len(names))
```

```
['Area', 'Beds', 'Baths', 'longitude', 'Towncode']
5
```

```
from sklearn.model_selection import train_test_split

# Droping rows contain NaN and unrequired columns
df.dropna(inplace=True, axis=0)

x = df.drop(['Price', 'Address', 'Town', 'ZipCode', 'Broker'], axis=1)
y = df['Price']

# Dividing the data into training and test set
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=42)

#Print Num of rows and Columns of test and training tet
print("X-Train data set :", x_train.shape)
print("X-Test data set :", x_test.shape)
print("Y-Train data set :", y_train.shape)
print("Y-Test data set :", y_test.shape)
```

```
X-Train data set : (2328, 7)
X-Test data set : (583, 7)
Y-Train data set : (2328,)
Y-Test data set : (583,)
```

```
from sklearn.linear_model import LinearRegression # Ordinary Least Squares(OLS) algorithm
from sklearn.linear_model import Ridge # Ridge algorithm
from sklearn.linear_model import Lasso # Lasso algorithm
from sklearn.linear_model import BayesianRidge # Bayesian algorithm
from sklearn.linear_model import ElasticNet # Elastic Net algorithm
from sklearn.svm import SVR # Support Vector Machine algorithm

# OLS Algorithm
ols = LinearRegression()
ols.fit(x_train, y_train)
ols_yhat = ols.predict(x_test)
print(f'Score of OLS regression : {ols.score(x, y):,.4f}')
print(f'Average value of the house by OLS regression: CA$ {y_test.mean():,.2f}')  # Present Price
print(f'Model predicted average value by OLS regression: CA$ {ols_yhat.mean():,.2f}') # Predicted Price
print('-----------------------------------------------------')
# Ridge Algorithm
ridge = Ridge(alpha = 0.5).fit(x_train, y_train)
ridge_yhat = ridge.predict(x_test)

# Lasso Algorithm
lasso = Lasso(alpha = 0.01).fit(x_train, y_train)
lasso_yhat = lasso.predict(x_test)

# Bayesian Algorithm
bayesian = BayesianRidge().fit(x_train, y_train)
bayesian_yhat = bayesian.predict(x_test)

# Elastic Net Algorithm
en = ElasticNet(alpha = 0.01).fit(x_train, y_train)
en_yhat = en.predict(x_test)

# SVR Algorithm
svm = SVR(C=100000, gamma = 150000).fit(x_train, y_train)
svm_yhat = svm.predict(x_test) #Predictions of House Prices
print(f'Average value of the house by SVM: CA$ {y_test.mean():,.2f}')  # Present Price
print(f'Model predicted average value by: CA$ {svm_yhat.mean():,.2f}') # Predicted Price
```

```
Score of OLS regression : 0.5189
Average value of the house by OLS regression: CA$ 3,082,483.79
Model predicted average value by OLS regression: CA$ 3,157,120.92
-----------------------------------------------------
Average value of the house by SVM: CA$ 3,082,483.79
Model predicted average value by: CA$ 2,747,125.32
```

Evaluation

To check the accuracy of implemented model, we used and explained variance score and Root-squared score. These metric are implemented and is available in scikit-learn library in Python.

```
 # evaluation metric : Explained Variance Score
from sklearn.metrics import explained_variance_score as evs
from sklearn.metrics import mean_squared_error

print('EXPLAINED VARIANCE SCORE values:')
print('Explained Variance Score of OLS model is: {:.4f}'.format(evs(y_test, ols_yhat)))
print('Mean squared error of OLS is: {:,.2f}'.format(mean_squared_error(y_test, ols_yhat,squared = False)))
print('-------------------------------------------------------')
print('Explained Variance Score of Ridge model is: {:.4f}'.format(evs(y_test, ridge_yhat)))
print('Mean squared error of Ridge is: {:,.2f}'.format(mean_squared_error(y_test, ridge_yhat,squared = False)))
print('-------------------------------------------------------')
print('Explained Variance Score of Lasso model is: {:.4f}'.format(evs(y_test, lasso_yhat)))
print('Mean squared error of Lasso is: {:,.2f}'.format(mean_squared_error(y_test, lasso_yhat,squared = False)))
print('-------------------------------------------------------')
print('Explained Variance Score of Bayesian model is: {:.4f}'.format(evs(y_test, bayesian_yhat)))
print('Mean squared error of Bayesian model is: {:,.2f}'.format(mean_squared_error(y_test, bayesian_yhat,squared = False)))
print('-------------------------------------------------------')
print('Explained Variance Score of Elastic Net is: {:.4f}'.format(evs(y_test, en_yhat)))
print('Mean squared error of Elastic Net is: {:,.2f}'.format(mean_squared_error(y_test, en_yhat,squared = False)))
print('-------------------------------------------------------')
print('Explained Variance Score of SVM is: {:.4f}'.format(evs(y_test, svm_yhat)))
print('Mean squared error of SVM is: {:,.4f}'.format(mean_squared_error(y_test, svm_yhat,squared = False)))
```

```
EXPLAINED VARIANCE SCORE values:
Explained Variance Score of OLS model is: 0.6214
Mean squared error of OLS is: 2,097,939.20
-------------------------------------------------------
Explained Variance Score of Ridge model is: 0.6214
Mean squared error of Ridge is: 2,097,955.65
-------------------------------------------------------
Explained Variance Score of Lasso model is: 0.6214
Mean squared error of Lasso is: 2,097,939.20
-------------------------------------------------------
Explained Variance Score of Bayesian model is: 0.6211
Mean squared error of Bayesian model is: 2,098,799.21
-------------------------------------------------------
Explained Variance Score of Elastic Net is: 0.6213
Mean squared error of Elastic Net is: 2,098,213.15
-------------------------------------------------------
Explained Variance Score of SVM is: 0.0000
Mean squared error of SVM is: 3,423,809.3367
```

```
# evaluation metric : R-squared

from sklearn.metrics import r2_score as r2

print('R-SQUARED values:')
print('Explained Variance Score of OLS model is: {:.4f}'.format(r2(y_test, ols_yhat)))
print('Explained Variance Score of Ridge model is: {:.4f}'.format(r2(y_test, ridge_yhat)))
print('Explained Variance Score of Lasso model is: {:.4f}'.format(r2(y_test, lasso_yhat)))
print('Explained Variance Score of Bayesian model is: {:.4f}'.format(r2(y_test, bayesian_yhat)))
print('Explained Variance Score of Elastic Net is: {:.4f}'.format(r2(y_test, en_yhat)))
print('Explained Variance Score of SVM is: {:.4f}'.format(r2(y_test, svm_yhat)))
```

```
R-SQUARED values:
Explained Variance Score of OLS model is: 0.6209
Explained Variance Score of Ridge model is: 0.6209
Explained Variance Score of Lasso model is: 0.6209
Explained Variance Score of Bayesian model is: 0.6206
Explained Variance Score of Elastic Net is: 0.6208
Explained Variance Score of SVM is: -0.0090
```

Discussion

The theoretical ideal values for R-squared and Explained Variance scores are greater than 0.7 and 0.6, respectively. However, our results showed that the scores for all of the models are below these values, indicating that the constructed models based on data set are inadequate for predicting the prices, accurately.

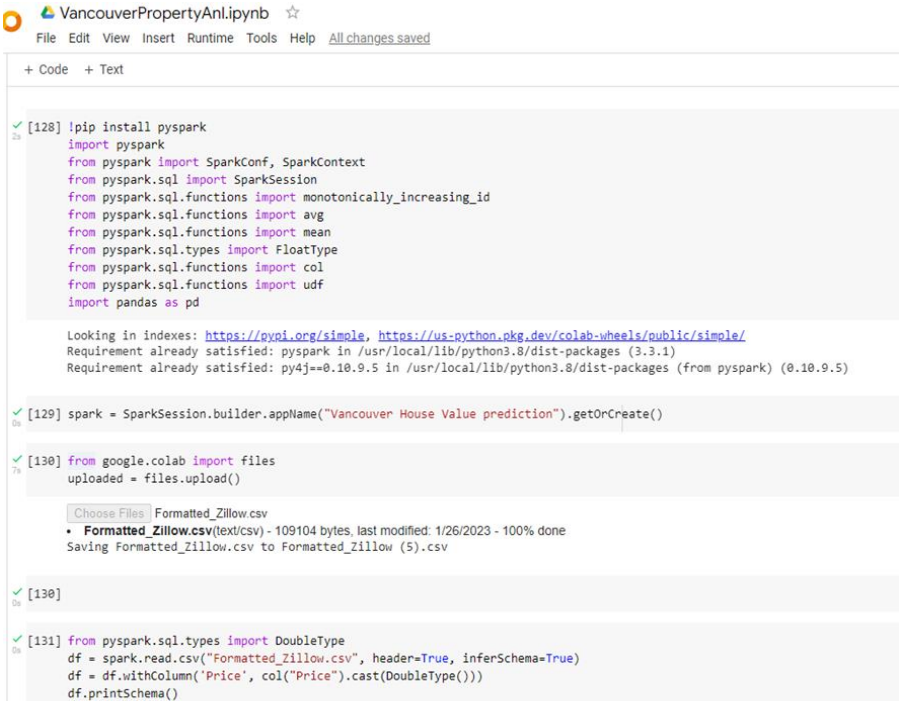
Result

In this section, we successfully developed and evaluated five distinct linear regression models through Python. After that, the most appropriate model based on our dataset is suggested. Nevertheless, this does not end here. Each model we created is underpinned by its own statistical and mathematical principles. We used these five models due to their popularity. However, there are a variety of other linear regression models, such as the Poisson regression model, boosted decision tree model and decision tree model that could be considered in the next version on this work.

5-2 Handle of big data workloads with Apache Spark

Our data in this project are not big to use big data processing tools. But to show ability of one of them, we have used **Apache Spark**. It is an open-source, distributed computing system that allows for large-scale data processing and analysis. It provides a general-purpose cluster-computing framework for a wide range of data processing tasks, such as batch processing, interactive queries, stream processing, and machine learning. Spark uses in-memory data processing, which allows for much faster processing of data than traditional disk-based systems.

Spark provides a programming interface, called **PySpark**, for working with data in the Python programming language. One of the key features of Spark is its ability to process data in parallel across a cluster of machines, which allows for faster processing of large data sets. This makes it a popular choice for big data processing and analytics, especially in industries such as finance and retail. It can also be used in combination with other big data technologies such as Hadoop and Apache Storm.



[131]
```
root
 |-- Type_Property: string (nullable = true)
 |-- Price: double (nullable = true)
 |-- Address: string (nullable = true)
 |-- Address_Street: string (nullable = true)
 |-- Address_Zipcode: string (nullable = true)
 |-- N_Beds: integer (nullable = true)
 |-- N_Baths: integer (nullable = true)
 |-- Area: integer (nullable = true)
 |-- Lat: double (nullable = true)
 |-- Long: double (nullable = true)
 |-- Broker_Name: string (nullable = true)
```

[132]
```
df.show(5)
```

Type_Property	Price	Address	Address_Street	Address_Zipcode	N_Beds	N_Baths	Area	Lat	Long	Broker_Name
Condo	429000.0	4990 McGeer St #1...	4990 McGeer St #114	V5R6C1	1	1	527	49.23952	-123.032814	RE/MAX City Realty
House	5.98E7	4838 Belmont Ave,...	4838 Belmont Ave	V6T1A9	5	8	12410	49.27489	-123.22121	Macdonald Realty
Condo	558000.0	1216 W 11th Ave #...	1216 W 11th Ave #206	V6H1K5	1	1	865	49.261246	-123.13179	Sotheby's Interna...
Condo	638000.0	489 Interurban Wa...	489 Interurban Wa...	V5X0C7	1	1	492	0.0	0.0	Jovi Realty Inc.
Condo	1428000.0	3188 Riverwalk Av...	3188 Riverwalk Av...	V5S0E7	2	2	1131	49.205387	-123.03918	RE/MAX Crest Realty

only showing top 5 rows

[133]
```
df.count()
```

800

[134]
```
df.select(*[mean(c) for c in df.columns]).show()
```

avg(Type_Property)	avg(Price)	avg(Address)	avg(Address_Street)	avg(Address_Zipcode)	avg(N_Beds)	avg(N_Baths)	avg(Area)	avg(Lat)	avg(Long)	avg(Broker_Name)
null	3164698.75375	null	null	null	3.3404255319148937	3.0938673341677094	2196.09625	42.052280633750016	-105.11074844875006	null

[135]
```
df.groupby('Type_Property').agg({col: 'avg' for col in df.columns[0:17]}).show()
```

Type_Property	avg(Area)	avg(N_Beds)	avg(Broker_Name)	avg(Address)	avg(N_Baths)	avg(Lat)	avg(Price)	avg(Type_Property)	avg(Long)	avg(Address_Street)	avg(Addres
Townhouse	1460.36	2.66	null	null	2.62	41.37400926999999	1718102.61	null	-103.41481089000003	null	
Lot / Land	10880.0	null	null	null	null	49.21254	650000.0	null	-123.10618	null	
Condo	1129.8597560975609	1.8353658536585367	null	null	1.8353658536585367	38.001195841463385	1797490.243902439	null	-94.96351820731712	null	
House	3770.498281786942	5.18213058419244	null	null	4.5326460481099655	47.21695499656358	5572019.164948453	null	-118.04527509965634	null	
Multi-family home	1651.9	3.6625	null	null	3.6125	40.63356125	1853304.5625	null	-101.56003080000002	null	

[136]
```
train, test = df.randomSplit([0.7, 0.3])
train, test
```

```
(DataFrame[Type_Property: string, Price: double, Address: string, Address_Street: string, Address_Zipcode: string, N_Beds: int, N_Baths: int, Area: int, Lat: double, Long: double, Broker_Name: string],
 DataFrame[Type_Property: string, Price: double, Address: string, Address_Street: string, Address_Zipcode: string, N_Beds: int, N_Baths: int, Area: int, Lat: double, Long: double, Broker_Name: string])
```

[137]
```
numerical_features_lst = train.columns
numerical_features_lst.remove('Price')
numerical_features_lst.remove('Address')
numerical_features_lst.remove('Address_Zipcode')
numerical_features_lst.remove('Address_Street')
numerical_features_lst.remove('Broker_Name')
numerical_features_lst.remove('Type_Property')
numerical_features_lst
```

```
['N_Beds', 'N_Baths', 'Area', 'Lat', 'Long']
```

[138]
```
from pyspark.ml.feature import Imputer

imputer = Imputer(inputCols=numerical_features_lst,
                            outputCols=numerical_features_lst)
imputer = imputer.fit(train)
train = imputer.transform(train)
test = imputer.transform(test)
train.show(7)
```

[138]

Type_Property	Price	Address	Address_Street	Address_Zipcode	N_Beds	N_Baths	Area	Lat	Long	Broker_Name
Condo	319000.0	1250 Bur0by St #6...	1250 Bur0by St #606	V6E1P6	1	1	560	0.0	0.0	Sutton Group-West...
Condo	339000.0	1251 Cardero St #...	1251 Cardero St #806	V6G2H9	1	1	436	0.0	0.0	Coldwell Banker P...
Condo	349000.0	1250 Burnaby St #...	1250 Burnaby St #607	V6E1P6	1	1	570	49.28147	-123.135605	Sutton Group-West...
Condo	378000.0	1850 Comox St #10...	1850 Comox St #1008	V6G1R3	1	1	546	0.0	0.0	Royal Pacific Tri...
Condo	399000.0	1219 Harwood St #...	1219 Harwood St #603	V6E1S5	1	1	540	0.0	0.0	Sunrich Realty
Condo	399000.0	2165 W 40th Ave #...	2165 W 40th Ave #404	V6M1W4	1	1	560	49.235817	-123.157295	RE/MAX Select Pro...
Condo	405000.0	1850 Comox St #12...	1850 Comox St #1207	V6G1R3	1	1	613	49.288967	-123.14078	Homelife Benchmar...

only showing top 7 rows

[139]
```
from pyspark.ml.feature import VectorAssembler

numerical_vector_assembler = VectorAssembler(inputCols=numerical_features_lst,
                                             outputCol='numerical_feature_vector')

train = numerical_vector_assembler.transform(train)
test = numerical_vector_assembler.transform(test)
train.show(2)
```

Type_Property	Price	Address	Address_Street	Address_Zipcode	N_Beds	N_Baths	Area	Lat	Long	Broker_Name	numerical_feature_vector
Condo	319000.0	1250 Bur0by St #6...	1250 Bur0by St #606	V6E1P6	1	1	560	0.0	0.0	Sutton Group-West...	[1.0,1.0,560.0,0....
Condo	339000.0	1251 Cardero St #...	1251 Cardero St #806	V6G2H9	1	1	436	0.0	0.0	Coldwell Banker P...	[1.0,1.0,436.0,0....

only showing top 2 rows

[140]
```
train.select('numerical_feature_vector').take(2)
```

```
[Row(numerical_feature_vector=DenseVector([1.0, 1.0, 560.0, 0.0, 0.0])),
 Row(numerical_feature_vector=DenseVector([1.0, 1.0, 436.0, 0.0, 0.0]))]
```

```
[141] from pyspark.ml.feature import StandardScaler

      scaler = StandardScaler(inputCol = 'numerical_feature_vector',
                              outputCol= 'scaled_numerical_feature_vector',
                              withStd= True, withMean=True)

      scaler = scaler.fit(train)

      train = scaler.transform(train)
      test = scaler.transform(test)

      train.show(3)
```

```
+-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+
|Type_Property|  Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|   Lat|       Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|
+-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+
|        Condo|319000.0|1250 Bur0by St #6...| 1250 Bur0by St #606|        V6E1P6|     1|     1| 560|   0.0|        0.0|Sutton Group-West...|    [1.0,1.0,560.0,0....|           [-1.2099470025672...|
|        Condo|339000.0|1251 Cardero St #...|1251 Cardero St #806|        V6G2H9|     1|     1| 436|   0.0|        0.0|Coldwell Banker P...|    [1.0,1.0,436.0,0....|           [-1.2099470025672...|
|        Condo|349000.0|1250 Burnaby St #...|1250 Burnaby St #607|        V6E1P6|     1|     1| 570|49.28147|-123.135605|Sutton Group-West...|    [1.0,1.0,570.0,49...|           [-1.2099470025672...|
+-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+
only showing top 3 rows
```

```
[142] train.select('scaled_numerical_feature_vector').take(3)
```

```
[Row(scaled_numerical_feature_vector=DenseVector([-1.2099, -1.0281, -0.8324, -2.4688, 2.4688])),
 Row(scaled_numerical_feature_vector=DenseVector([-1.2099, -1.0281, -0.8943, -2.4688, 2.4688])),
 Row(scaled_numerical_feature_vector=DenseVector([-1.2099, -1.0281, -0.8275, 0.4058, -0.4047]))]
```

```
[143] from pyspark.ml.feature import StringIndexer

      indexer = StringIndexer(inputCol ='Type_Property', outputCol = 'Type_Property_index' )

      indexer = indexer.fit(train)
      train = indexer.transform(train)
      test = indexer.transform(test)

      train.show(3)
```

```
[143] +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      |Type_Property|  Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|   Lat|       Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|Type_Proper
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      |        Condo|319000.0|1250 Bur0by St #6...| 1250 Bur0by St #606|        V6E1P6|     1|     1| 560|   0.0|        0.0|Sutton Group-West...|    [1.0,1.0,560.0,0....|           [-1.2099470025672...|
      |        Condo|339000.0|1251 Cardero St #...|1251 Cardero St #806|        V6G2H9|     1|     1| 436|   0.0|        0.0|Coldwell Banker P...|    [1.0,1.0,436.0,0....|           [-1.2099470025672...|
      |        Condo|349000.0|1250 Burnaby St #...|1250 Burnaby St #607|        V6E1P6|     1|     1| 570|49.28147|-123.135605|Sutton Group-West...|    [1.0,1.0,570.0,49...|           [-1.2099470025672...|
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      only showing top 3 rows
```

```
[144] set(train.select('Type_Property_index').collect())
```

```
{Row(Type_Property_index=0.0),
 Row(Type_Property_index=1.0),
 Row(Type_Property_index=2.0),
 Row(Type_Property_index=3.0)}
```

```
[145] from pyspark.ml.feature import OneHotEncoder

      one_hot_encoder = OneHotEncoder(inputCol='Type_Property_index', outputCol = 'Type_Property_One_Hot' )

      one_hot_encoder = one_hot_encoder.fit(train)

      train = one_hot_encoder.transform(train)
      test = one_hot_encoder.transform(test)

      train.show(3)
```

```
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      |Type_Property|  Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|   Lat|       Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|Type_Proper
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      |        Condo|319000.0|1250 Bur0by St #6...| 1250 Bur0by St #606|        V6E1P6|     1|     1| 560|   0.0|        0.0|Sutton Group-West...|    [1.0,1.0,560.0,0....|           [-1.2099470025672...|
      |        Condo|339000.0|1251 Cardero St #...|1251 Cardero St #806|        V6G2H9|     1|     1| 436|   0.0|        0.0|Coldwell Banker P...|    [1.0,1.0,436.0,0....|           [-1.2099470025672...|
      |        Condo|349000.0|1250 Burnaby St #...|1250 Burnaby St #607|        V6E1P6|     1|     1| 570|49.28147|-123.135605|Sutton Group-West...|    [1.0,1.0,570.0,49...|           [-1.2099470025672...|
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+------+-----------+--------------------+--------------------+----------------------------+----------
      only showing top 3 rows
```

```
[146] assembler = VectorAssembler(inputCols=['scaled_numerical_feature_vector', 'Type_Property_One_Hot'],
                                  outputCol='final_feature_vector')

      train = assembler.transform(train)
      test = assembler.transform(test)
      train.show(2)
```

```
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+---+-----+--------------------+--------------------+----------------------------+-------------------+---
      |Type_Property|  Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|Lat| Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|Type_Property_index|Typ
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+---+-----+--------------------+--------------------+----------------------------+-------------------+---
      |        Condo|319000.0|1250 Bur0by St #6...| 1250 Bur0by St #606|        V6E1P6|     1|     1| 560|0.0|  0.0|Sutton Group-West...|    [1.0,1.0,560.0,0....|           [-1.2099470025672...|                0.0|
      |        Condo|339000.0|1251 Cardero St #...|1251 Cardero St #806|        V6G2H9|     1|     1| 436|0.0|  0.0|Coldwell Banker P...|    [1.0,1.0,436.0,0....|           [-1.2099470025672...|                0.0|
      +-------------+-------+--------------------+--------------------+---------------+------+------+----+---+-----+--------------------+--------------------+----------------------------+-------------------+---
      only showing top 2 rows
```

```
[147] train.select('final_feature_vector').take(2)
```

```
[Row(final_feature_vector=DenseVector([-1.2099, -1.0281, -0.8324, -2.4688, 2.4688, 1.0, 0.0, 0.0])),
 Row(final_feature_vector=DenseVector([-1.2099, -1.0281, -0.8943, -2.4688, 2.4688, 1.0, 0.0, 0.0]))]
```

```
[148] from pyspark.ml.regression import  LinearRegression

      lr = LinearRegression(featuresCol = 'final_feature_vector', labelCol='Price')

      lr
```

```
LinearRegression_2972c72ffff8
```

```
[149] lr = lr.fit(train)
      lr
```

```
LinearRegressionModel: uid=LinearRegression_2972c72ffff8, numFeatures=8
```

```
[150] pred_train_df = lr.transform(train).withColumnRenamed('prediction', 'Predicted_House_Price')
      pred_train_df.show(5)
```

```
[176] +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      |Type_Property|   Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|    Lat|                Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|Type_Proper
      +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      |       Condo|319000.0|1250 Bur0by St #6...| 1250 Bur0by St #606|        V6E1P6|     1|      1| 560|    0.0|                 0.0|Sutton Group-West...|    [1.0,1.0,560.0,0....|           [-1.2422016680011...|
      |       Condo|349000.0|1250 Burnaby St #...|1250 Burnaby St #607|        V6E1P6|     1|      1| 570| 49.28147|-123.135605|Sutton Group-West...|    [1.0,1.0,570.0,49...|           [-1.2422016680011...|
      |       Condo|378000.0|1850 Comox St #10...| 1850 Comox St #1008|        V6G1R3|     1|      1| 546|    0.0|                 0.0|Royal Pacific Tri...|    [1.0,1.0,546.0,0....|           [-1.2422016680011...|
      |       Condo|399000.0|1219 Harwood St #...|1219 Harwood St #603|        V6E1S5|     1|      1| 540|    0.0|                 0.0|      Sunrich Realty|    [1.0,1.0,540.0,0....|           [-1.2422016680011...|
      |       Condo|399000.0|1534 Harwood St #...|1534 Harwood St #...|        V6G1X9|     1|      1| 540|49.283222|-123.140495|Stilhavn Real Est...|    [1.0,1.0,540.0,49...|           [-1.2422016680011...|
      +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      only showing top 5 rows
```

```
[177] pred_test_df = lr.transform(test).withColumnRenamed('prediction', 'Predicted_House_Price')
      pred_test_df.show(5)
```

```
      +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      |Type_Property|   Price|             Address|      Address_Street|Address_Zipcode|N_Beds|N_Baths|Area|    Lat|                Long|         Broker_Name|numerical_feature_vector|scaled_numerical_feature_vector|Type_Proper
      +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      |       Condo|339000.0|1251 Cardero St #...|1251 Cardero St #806|        V6G2H9|     1|      1| 436|    0.0|                 0.0|Coldwell Banker P...|    [1.0,1.0,436.0,0....|           [-1.2422016680011...|
      |       Condo|349000.0|1219 Harwood St #...|1219 Harwood St #604|        V6E1S5|     1|      1| 547|  49.2808| -123.13561|Century 21 In Tow...|    [1.0,1.0,547.0,49...|           [-1.2422016680011...|
      |       Condo|380000.0|2255 Eton St #106...|   2255 Eton St #106|        V5L1C9|     1|      1| 541|49.287766| -123.05868| RE/MAX Crest Realty|    [1.0,1.0,541.0,49...|           [-1.2422016680011...|
      |       Condo|399000.0|2165 W 40th Ave #...|2165 W 40th Ave #404|        V6M1W4|     1|      1| 560|49.235817|-123.157295|RE/MAX Select Pro...|    [1.0,1.0,560.0,49...|           [-1.2422016680011...|
      |       Condo|438800.0|567 Hornby St #12...| 567 Hornby St #1207|        V6C2E8|     1|      1| 383|    0.0|                 0.0|Park Georgia Real...|    [1.0,1.0,383.0,0....|           [-1.2422016680011...|
      +------------+--------+--------------------+--------------------+---------------+------+-------+----+-------+--------------------+--------------------+--------------------+--------------------+
      only showing top 5 rows
```

```
[178] pred_test_pd_df = pred_test_df.toPandas()
      pred_test_pd_df.head(2)
```

index	Type_Property	Price	Address	Address_Street	Address_Zipcode	N_Beds	N_Baths	Area	Lat	Long	Broker_Name	numerical_feature_vector	scaled_numerical_feature_vector
0	Condo	339000.0	1251 Cardero St #806, Vancouver, BC V6G 2H9	1251 Cardero St #806	V6G2H9	1	1	436	0.0	0.0	Coldwell Banker Prestige Realty	[1.0,1.0,436.0,0.0,0.0]	[-1.2422016680011334,-1.0607991189354384,-0.9212716771034052,-2.52652885
1	Condo	349000.0	1219 Harwood St #604, Vancouver, BC V6E 1S5	1219 Harwood St #604	V6E1S5	1	1	547	49.2808	-123.13561	Century 21 In Town Realty	[1.0,1.0,547.0,49.2808,-123.13561]	[-1.2422016680011334,-1.0607991189354384,-0.8641896927378544,0.396553585

Show 25 per page
Like what you see? Visit the data table notebook to learn more about interactive tables.

```
[179] predictions_and_actuals = pred_test_df[['Predicted_House_Price','Price']]
      predictions_and_actuals_rdd = predictions_and_actuals.rdd
      predictions_and_actuals_rdd.take(3)

      [Row(Predicted_House_Price=437382.53898741305, Price=339000.0),
       Row(Predicted_House_Price=793096.2317139432, Price=349000.0),
       Row(Predicted_House_Price=1030928.0085978992, Price=380000.0)]
```

```
[180] predictions_and_actuals = predictions_and_actuals_rdd.map(tuple)
      predictions_and_actuals_rdd.take(2)

      [Row(Predicted_House_Price=437382.53898741305, Price=339000.0),
       Row(Predicted_House_Price=793096.2317139432, Price=349000.0)]
```

```
[181] from pyspark.mllib.evaluation import RegressionMetrics
      metrics = RegressionMetrics(predictions_and_actuals_rdd)

      s = '''
      Mean Squeard Error:      {0}
      Root Mean Squeard Error: {1}
      Mean Absolute Error:     {2}
      R**2:                    {3}
      '''.format(metrics.meanSquaredError,
                 metrics.rootMeanSquaredError,
                 metrics.meanAbsoluteError,
                 metrics.r2)
      print(s)

      /usr/local/lib/python3.8/dist-packages/pyspark/sql/context.py:157: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
        warnings.warn(

      Mean Squeard Error:      7540412179856.261
      Root Mean Squeard Error: 2745981.096048598
      Mean Absolute Error:     1078876.1753957355
      R**2:                    0.7232783849229282
```

5-2 Discussion

To see precise of our modeling, we calculate four following metrics to evaluate the performance of our LR model:

1- Mean Squared Error (MSE)
2- Root of Mean Squared Error (RMSE)
3- Mean Absolut Error (MAE)
4- R^2

It is evident that a high Mean Squared Error (MSE) value indicates that the model is a poorer fit for the data and has a larger discrepancy between the predicated and actual values. It is important to note that MSE is highly sensitive to the scale of the target variable, and consequently, if the target variable is on a large scale, then the MSE will also be large. In order to compare different models or datasets, it is recommended to use relative error metrics, such as Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

The primary reasons for such errors are primarily due to an insufficient data set, a small sample size and the presence of outliers. A comprehensive data set which includes a wide array of information about the properties, such as living room area (in sqft), number of floors, waterfront (yes/ no), view (ranked), condition (ranked), grade (level in building for condo), area of above floor (in sqft), area of basement (in sqft), year of built, renovated (yes/ no), and year of renovation, can allow for more precise price predictions.

A meager sample size of 2900 rows inadequate to split into training and test sets is also a contributing factor. Finally, linear regression is highly sensitive to outliers, and if the data set contains any such points, robust regression methods should be implemented.

Section 6 Building a data pipeline with Google Dataflow and Apache Beam

In this section, we continue our job by creating a data-parallel processing pipeline with GCP [11]. Among existing tools, we work with Google Cloud Dataflow, a fully-managed, serverless data processing service that provides a programming model for batch and stream processing, with support for distributed processing back-end, Apache Beam.

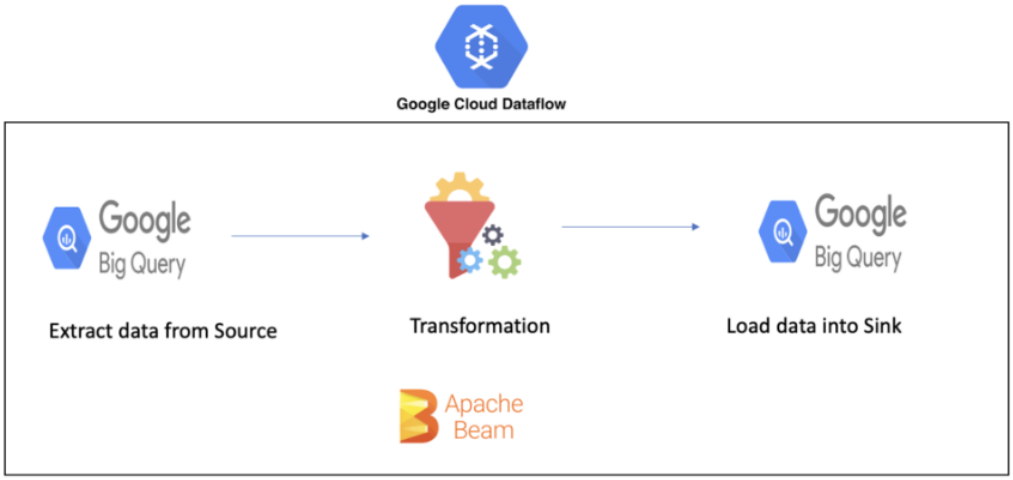


Figure 17 BigQuery data pipeline using Google Cloud Dataflow/Apache Beam[12]

To build and execute data pipeline ETL, we follow next steps.

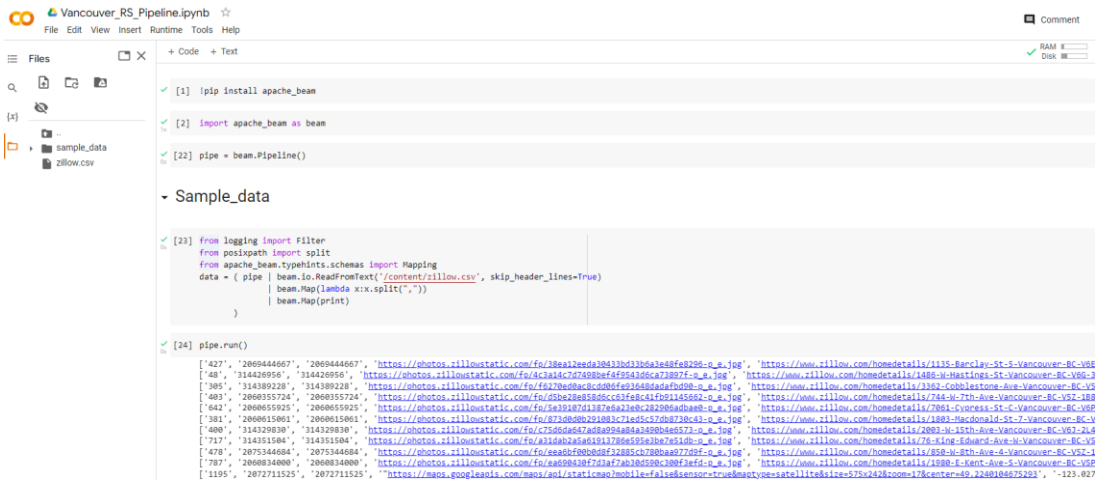
First of all we enabled Dataflow and Bigquery APIs. Afterwards we create a cloud storage bucket in the nearest region and store the .csv data file in the bucket.

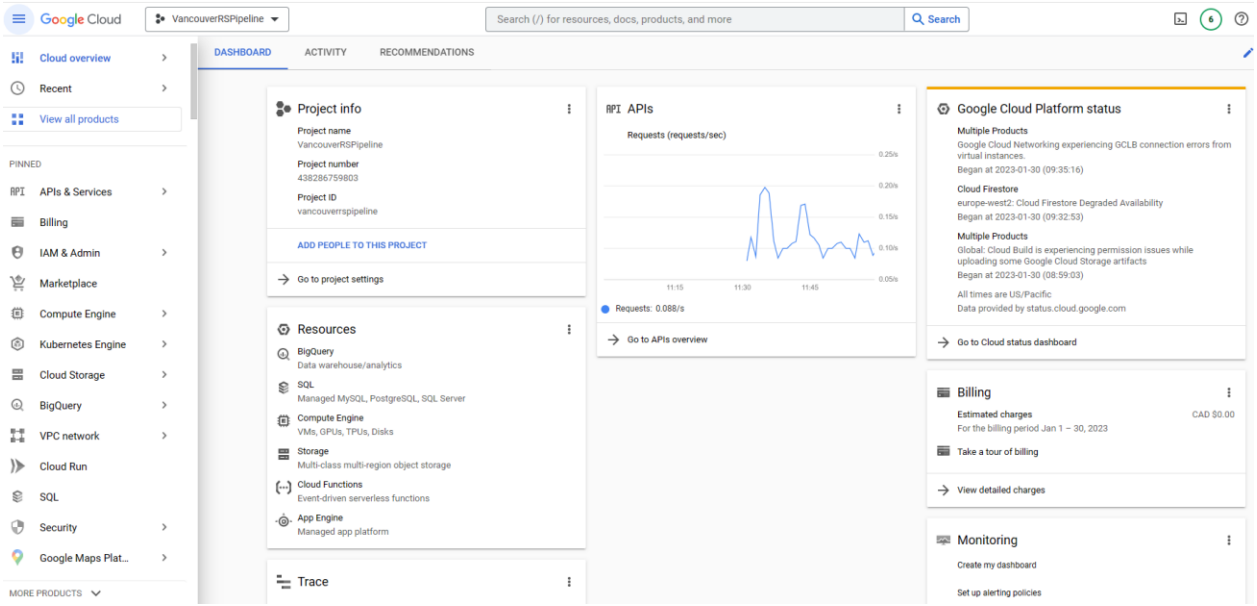
This phase includes following steps:

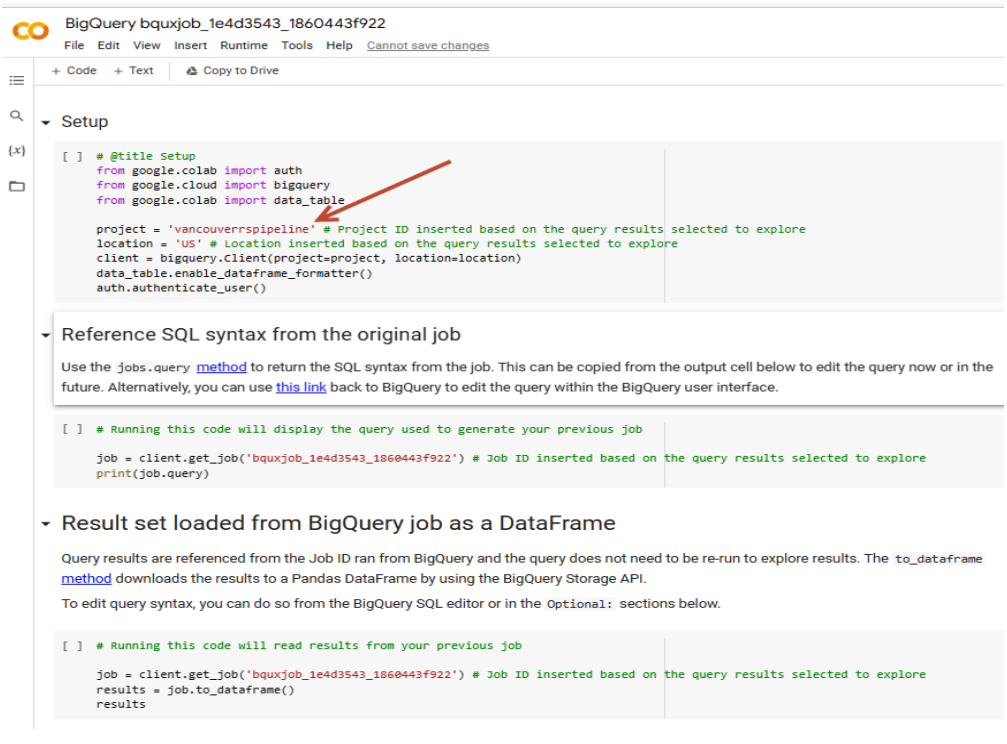
1- Read input data file into a pcollection: Input data for pipelines can be in any form: csv or json. We used .csv that created in previous section.
2- Create new pcollection(s) by applying filters, transforms on existing pcollection
3- Create new pcollection(s) by grouping: GroupByKey
4- Create new pcollection(s) by combining pcollection s: CoGroupByKey
5- Apply aggregate functions: count, sum
6- Mapping the result to convert into a dictionary format
7- Dumping the pipeline output into BigQuery table sink

Here we have some code snapshot and screenshot of GCP environment:

[11] Google Cloud Platform
[12] source: www.revisitclass.com


Vancouver_RS_Pipeline.ipynb
File Edit View Insert Runtime Tools Help
Comment
Files
+ Code + Text
RAM
Disk
sample_data
zillow.csv
[1] !pip install apache_beam
[2] import apache_beam as beam
[22] pipe = beam.Pipeline()
Sample_data
[23] from logging import Filter
from posixpath import split
from apache_beam.typehints.schemas import Mapping
data = (pipe | beam.io.ReadFromText('/content/zillow.csv', skip_header_lines=True)
| beam.Map(lambda x:x.split(","))
| beam.Map(print)
)
[24] pipe.run()


Google Cloud
VancouverRSPipeline
Search (/) for resources, docs, products, and more
Search
Cloud overview
Recent
View all products
PINNED
APIs & Services
Billing
IAM & Admin
Marketplace
Compute Engine
Kubernetes Engine
Cloud Storage
BigQuery
VPC network
Cloud Run
SQL
Security
Google Maps Plat...
MORE PRODUCTS
DASHBOARD
ACTIVITY
RECOMMENDATIONS
Project info
Project name
VancouverRSPipeline
Project number
438286759803
Project ID
vancouverrspipeline
ADD PEOPLE TO THIS PROJECT
Go to project settings
APIs
Requests (requests/sec)
Requests: 0.088/s
Go to APIs overview
Google Cloud Platform status
Multiple Products
Google Cloud Networking experiencing GCLB connection errors from virtual instances.
Began at 2023-01-30 (09:35:16)
Cloud Firestore
europe-west2: Cloud Firestore Degraded Availability
Began at 2023-01-30 (09:32:53)
Multiple Products
Global: Cloud Build is experiencing permission issues while uploading some Google Cloud Storage artifacts
Began at 2023-01-30 (08:59:03)
All times are US/Pacific
Data provided by status.cloud.google.com
Go to Cloud status dashboard
Resources
BigQuery
Data warehouse/analytics
SQL
Managed MySQL, PostgreSQL, SQL Server
Compute Engine
VMs, GPUs, TPUs, Disks
Storage
Multi-class multi-region object storage
Cloud Functions
Event-driven serverless functions
App Engine
Managed app platform
Billing
Estimated charges
CAD $0.00
For the billing period Jan 1 – 30, 2023
Take a tour of billing
View detailed charges
Trace
Monitoring
Create my dashboard
Set up alerting policies


BigQuery bquxjob_1e4d3543_1860443f922
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
Setup
# @title Setup
from google.colab import auth
from google.cloud import bigquery
from google.colab import data_table
project = 'vancouverrspipeline' # Project ID inserted based on the query results selected to explore
location = 'US' # Location inserted based on the query results selected to explore
client = bigquery.Client(project=project, location=location)
data_table.enable_dataframe_formatter()
auth.authenticate_user()
Reference SQL syntax from the original job
Use the jobs.query method to return the SQL syntax from the job. This can be copied from the output cell below to edit the query now or in the future. Alternatively, you can use this link back to BigQuery to edit the query within the BigQuery user interface.
# Running this code will display the query used to generate your previous job
job = client.get_job('bquxjob_1e4d3543_1860443f922') # Job ID inserted based on the query results selected to explore
print(job.query)
Result set loaded from BigQuery job as a DataFrame
Query results are referenced from the Job ID ran from BigQuery and the query does not need to be re-run to explore results. The to_dataframe method downloads the results to a Pandas DataFrame by using the BigQuery Storage API.
To edit query syntax, you can do so from the BigQuery SQL editor or in the Optional: sections below.
# Running this code will read results from your previous job
job = client.get_job('bquxjob_1e4d3543_1860443f922') # Job ID inserted based on the query results selected to explore
results = job.to_dataframe()
results

Here we can see descriptive result of run bigQuery with above criteria. They include count, mean, std, min, max, 25%, 50%, 75% and max.

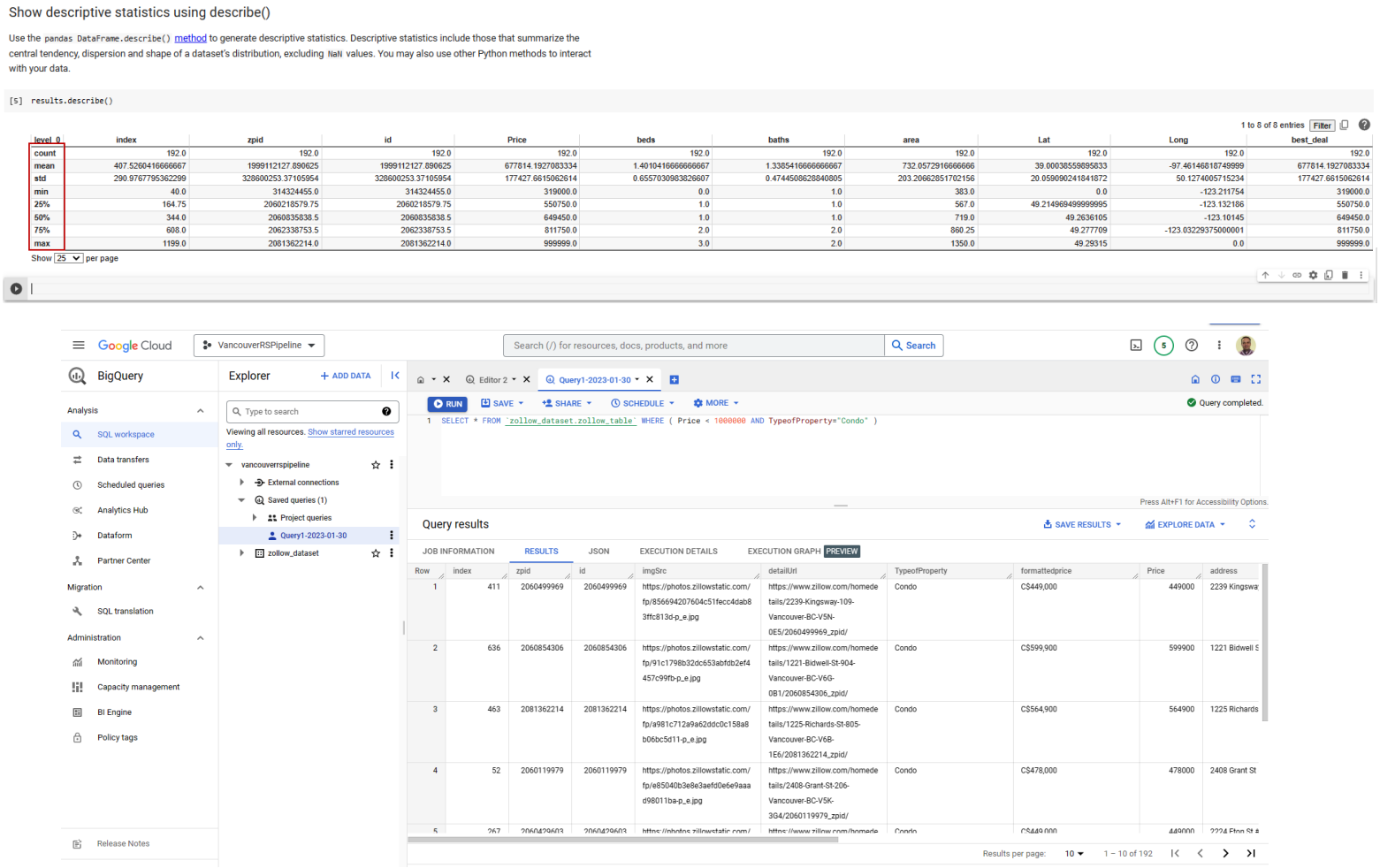


Figure 18 Data analysis with Google BigQuery (Image by the author)

Section 7 Creating Dashboard with Streamlit

Streamlit[13] is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In this part we created a responsive dashboard and app for this project This visualization includes map, line-chart, histogram and… Categorized by _town_ in province.

You can find source and demo here:

Demo link: https://kasraheidarinezhad-vanwebstreamlit-app-819wyp.streamlit.app/

Source link: https://github.com/kasraheidarinezhad/VanWebStreamlit/blob/master/app.py

[13] https://streamlit.io

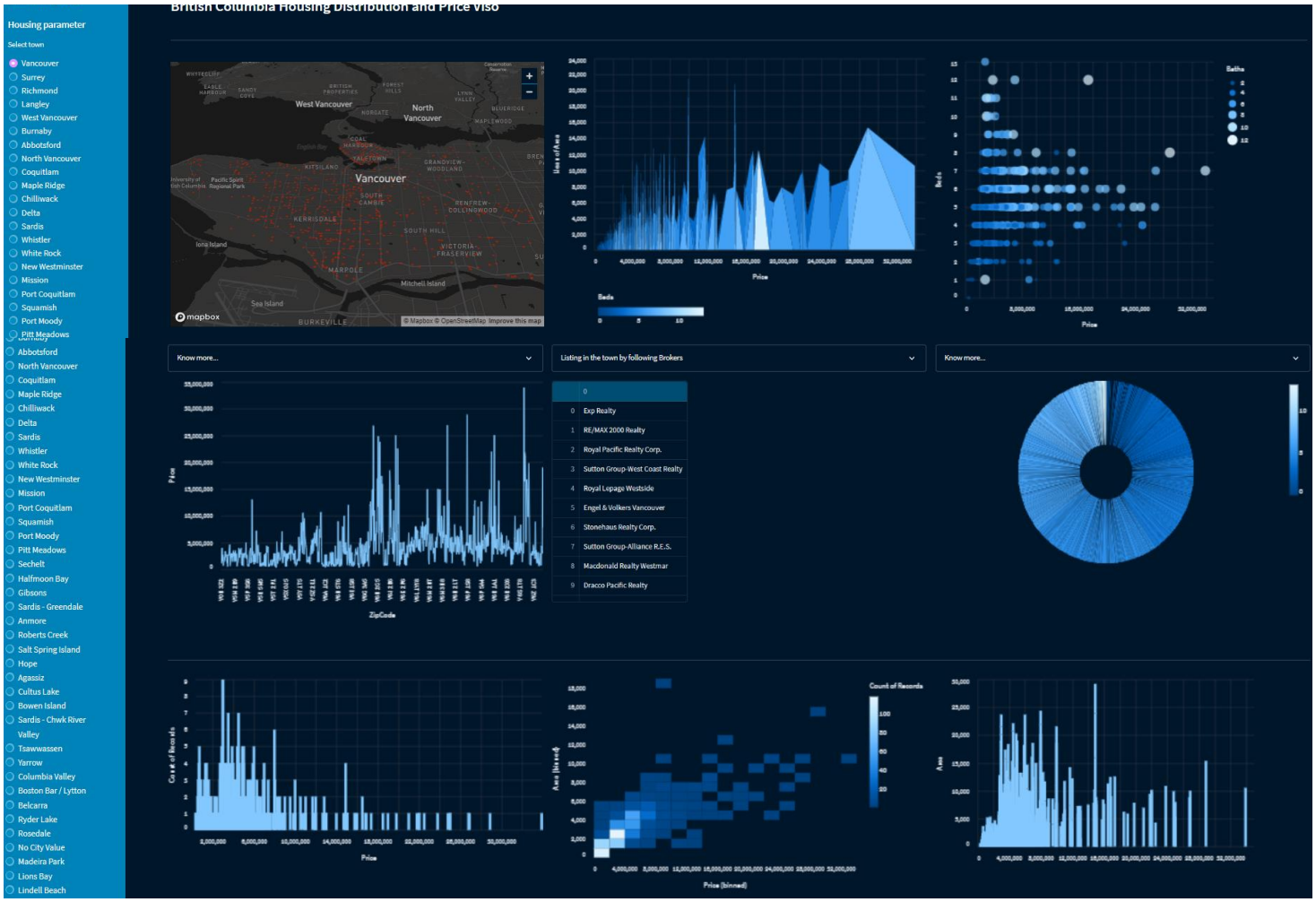


Figure 19 Dashboard with Python/streamlit library (Image by the author)

Section 8 Conclusion

In this project, we have walked through to providing required data about real estate market in Vancouver, BC. Canada, we scraped data, preprocessed and did some basic statistical procedure. Afterward. We had some visualization with Python and Tableau to demonstration feature of distribution of real estate based on diversity and category. We also carried out some data manipulate through database specially SQL and SELECT statement. Finally we tried to predict housing price with LR model based on data in hand.

To be sure, prediction property price is a challenging problem. In our case, it may be more appropriate to use non-linear models or other types of statistical analysis. Additionally, it is important to be aware of the assumptions of LR, such as linearity, independence of errors, and normality of errors and check if these assumptions are met before applying LR. It is also worth considering other factors that may be affecting the relationship between the variables, such as outliers or multi-collinearity.

Linear regression (LR) is a powerful tool that allows researchers to examine the multiple factors that contribute to social experiences and control for the influence of spurious effects. It also helps in creating refined graphs of relationships through regression lines, which can be a straightforward and accessible way of presenting results. Understanding LR coefficients enables us to understand both the direction and strength of the relationship between variables. However, care is needed to examine variables and construct them in forms that are amenable to this approach, such as creating dummy variables. They also need to examine findings carefully and test for concerns such as collinearity or patterns

among residuals. Despite this, LRs are quite forgiving of minor breaches of these assumptions and can produce some of the most useful information on the relationships between variables.

Based on our dataset, using of LR is not recommended, because the direction of the correlation is not clear from the scatter plot. If the scatter plot shows a clear pattern, such as a positive or negative correlation, it is more likely that a LR model will provide useful results. However, in our case, the scatter plot [Figure 8] shows a complex pattern or no clear relationship.

Appendix

List of Figures

Keywords

data pipline, json, big data, bigquery, GCP, data science, data mining, ETL (Extract-Transform-Load), Github, pandas, GeoJson, geopy ,machine learning, mathplotlib, scraping, pyspark, apache spark, apache beam, data visualization, tableau, folium, seaborn, sqlite3, nampy, web scraping, google dataflow, sklearn, streamlit

The Tools

Anaconda, Python, SQL, Tableau (Public), Google Cloud Platform (GCP), Google Colab, Google Trends, streamlit

About the author

Kasra Heidarinezhad has achieved Bachelor and Master's degree in Computer Engineering from the I. Azad University. He is an accomplished Data engineer, having developed effective solutions to challenging data-related problems. His expertise includes Big Data, Machine Learning, and Predictive Analytics, and he is well-versed in various programming languages and software tools which aid in the analysis and engineering of data. Kasra has been involved in data-driven projects in the financial services, retail, gas-oil, and energy industries. He has a comprehensive knowledge of the data lifecycle, from acquisition, cleaning, and analysis to visualization. Additionally, Kasra is highly capable of designing data pipelines, data warehouses, and machine learning models.

Phone: (604) 442-3332 **Email: Kasra.Heidarinezhad@gmail.com**

https://www.Github.com/kasraheidarinezhad

https://www.Linkedin.com/kasra-heidarinezhad

https://public.tableau.com/app/profile/kasra.heidarinezhad