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

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

Author

Kasra Heidarinezhad

Last Update

Nov-10-2022

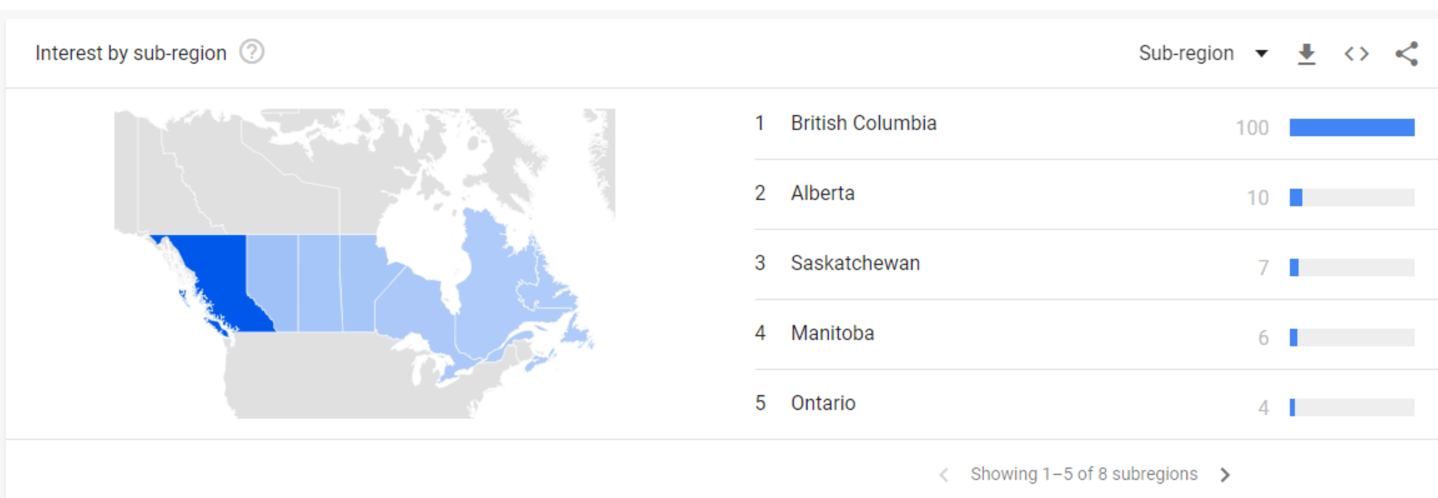
Last Data Capture

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 Vancouver 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)

[1] https://trends.google.com

According to Statista[2] , Vancouver ranks 7th most expensive residential property markets worldwide in 2020. To finding a reasonable price property, there are couple of different resource in the market. For example websites such as Realtor.com and Redfin.com. In this project, we decided to analysis real estate market in Vancouver, BC, Canada with retrieving listing detail from one of existing resource.

Objective of this analyses of housing market, visualize statistic feature and finally predicting sales price in house listing via statistic data modeling.

Section 2 Creating data source, data cleaning, missing values

First step is to get data to analysis. You have two choice here: First is to link a government agency or real estate license to use as a part of research (does not work for me). Second choice is scraping data. In this article, we kept going with the second option. Finding resource is next challenge of this project. To choice a good option among the existing resource, I considered two following options:
1- Comprehensively of data that providing by a website
2- Difficulty of accessing to data within website
After a quite study and considering above mentioned factors, we decided to going to Zillow[3].

2-1 Web Scraping with Python

Required Libraries

Here we list a major required libraries. For easiness, we could throw all of them in a RequiredLibraries.txt file and run: pip install RequiredLibraries.txt.

```
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

Zillow throw Captchas so when you try and run a request.get(url) type of function. So way to get around this is by adding headers to the request function as you can see below:

```
#add headers in case you use chromedriver (captchas are no fun); namely used for chromedriver
req_headers = {
    'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'en-US,en;q=0.8',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
}
```

[2] https://www.statista.com/statistics/1040698/most-expensive-property-markets-worldwide/
[3] https://Zillow.com

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
- Complete source
- Plot to rule them all

Here is complete code and result

```
import requests
import re
import json
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

#add headers in case you use chromedriver (captchas are no fun); namely used for chromedriver
req_headers = {
    'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'en-US,en;q=0.8',
    'upgrade-insecure-requests': '1',
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
}

with requests.Session() as s:
    data_list = []
    resp = s.get('https://www.zillow.com/homes/for_sale/Vancouver,-BC_rb/', headers=req_headers)
    data = json.loads(re.search(r'!--(\{"queryState".*?)-->', resp.text).group(1))
    data_list.append(data)

    for pages in range(1,20):   #just grabbing the first 20 pages
        resp = s.get('https://www.zillow.com/homes/for_sale/Vancouver,-BC_rb/'+str(pages+1)+'_p/', headers=req_headers)
        data = json.loads(re.search(r'!--(\{"queryState".*?)-->', resp.text).group(1))
        data_list.append(data)

df = pd.DataFrame()

def make_frame(frame):
    for i in data_list:
        for item in i['cat1']['searchResults']['listResults']:
            frame = frame.append(item, ignore_index=True)
    return frame

df = make_frame(df)

df = df.drop('hdpData', 1) #drop cols
df = df.drop_duplicates(subset='zpid', keep="last") #drop dupes

#filters
df['zestimate'] = df['zestimate'].fillna(0)
df['best_deal'] = df['unformattedPrice'] - df['zestimate']
df = df.sort_values(by='best_deal',ascending=True)

df.to_csv('zillow_original.csv', encoding='utf-8')
```

Result: raw data

zpid	int64
id	float64
providerListingId	object
imgSrc	object
hasImage	object
carouselPhotos	object
detailUrl	object
statusType	object
statusText	object
countryCurrency	object
price	int64

```
unformattedPrice              object
address                       object
addressStreet                 object
addressCity                   object
addressState                  object
addressZipcode                bool
isUndisclosedAddress          int64
beds                          int64
baths                         float64
area                          int64
latLong                       object
isZillowOwned                 bool
variableData                  object
badgeInfo                     float64
isSaved                       bool
isUserClaimingOwner           bool
isUserConfirmedClaim          bool
pgapt                         object
sgapt                         object
zestimate                     int64
shouldShowZestimateAsPrice    bool
has3DModel                    bool
hasVideo                      bool
isHomeRec                     bool
info1String                   object
brokerName                    object
hasAdditionalAttributions     bool
isFeaturedListing             bool
availabilityDate              float64
list                          bool
relaxed                       bool
hasOpenHouse                  object
openHouseStartDate            object
openHouseEndDate              object
openHouseDescription          object
streetViewMetadataURL         object
streetViewURL                 object
best_deal                     int64
```

Note

Keep in mind that it uses anti-scraping techniques like captchas, IP blocking, and honeypot traps to prevent its data from scraping. We do think our IP/device is probably blacklisted by now
Congratulation: First step done!

2-2 Data Cleaning

Data cleaning involves identifying and correcting any errors or inconsistencies in the data, such as missing values, duplicate records or incorrect data types. This is an important step because it helps to ensure that the data is complete and accurate, which is necessary for building reliable models. For this purpose, some columns that are not likely to help in data analysis and predicting the target variable are dropped from the original data frame.

2-3 Missing Values

It is important to fill in missing data with NaN (Not a Number) because it allows you to identify missing values in your data clearly. Filling missing values with NaN allows us to identify which values are missing and take appropriate action easily. Here, the missing values in the columns 'Volume', 'Interior', 'Availability', 'Garage', 'Upkeep Status,' 'Specification', 'Location Type', 'Number of floors', 'Details of Garden', 'Details of Storage', 'Number of Bedrooms', 'Details of Balcony', Number of Bathrooms and Description of Storage are addressed by filling it with NaN values.

After data cleaning, shape of data is like result:

```
df = pd.read_csv("zillow.csv")
print(f"Number of samples: {df.shape[0]}")
print(f"Number of features in set: {df.shape[1]}")
print("Features:")
print(df.dtypes)
```

Result:
Number of samples: 800
Number of features in set: 18
Features:

index	int64
zpid	int64
id	int64
imgSrc	object
detailUrl	object
TypeofProperty	object
formattedprice	object
Price	int64
address	object
addressStreet	object
addressZipcode	object
beds	float64
baths	float64
area	int64
latLong	object
has3DModel	bool
brokerName	object
best_deal	int64
dtype	object

Section 3 Data visualization

Here we did two Types of visualization:

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

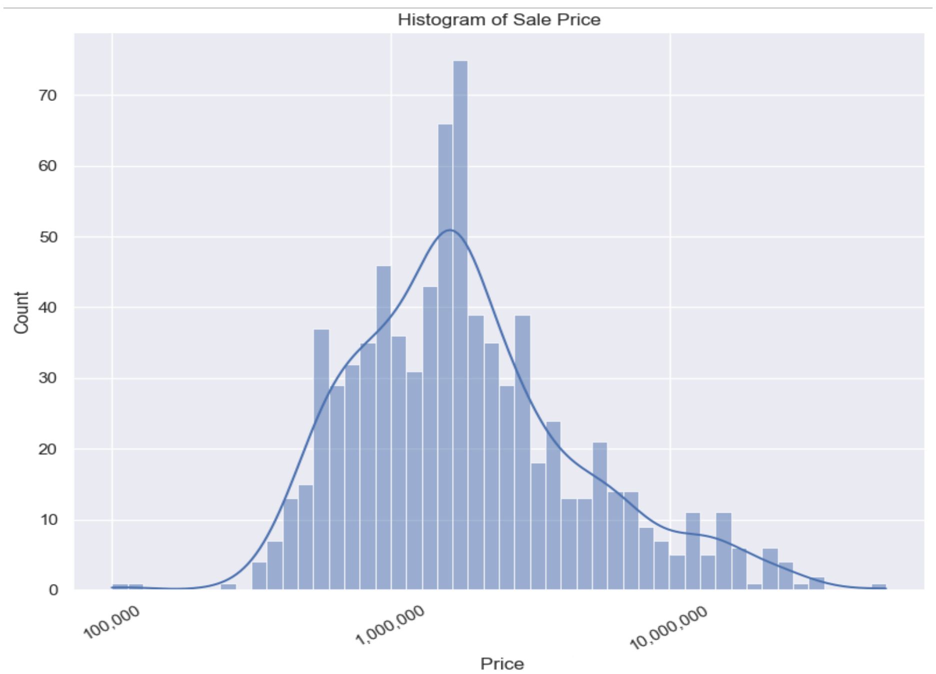
3-1 Data visualization with Python libraries

Here we did some codding to create a shape of data and histogram.

First and most important feature of statistics is Histogram, extracted by following code, result in [Figure 2]

```
# histogram of target=price variable
sns.set_theme()
graph = sns.displot(data=propertyListFrame, x="Price", kde=True, log_scale=True, bins=50)
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 2] Histogram of real estate price in Vancouver. (Image by the author)

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

Mean: 3,164,698.75 CA$
Median: 1,675,000.00 CA$
Skewness: 4.6598
Kurtosis: 33.9900

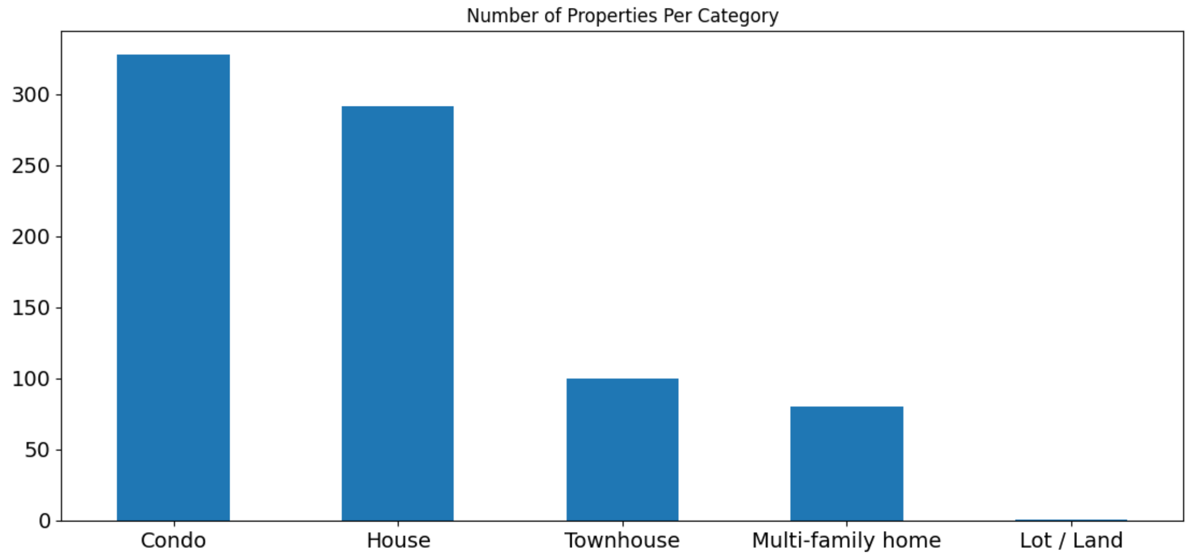
Discussion: It shows the distribution of the price of properties in Vancouver, BC, is relatively high positive skewness (the mean is greater than the median). In the prediction model, this will cause, the less accurate in the price prediction. In the other hand, high kurtosis represents heavy tails meaning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

1- Statistics per category: **Type of property**

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

```
df['TypeofProperty'] = df['TypeofProperty'].str.strip()
graph = df['TypeofProperty'].value_counts()
graph.plot(kind='bar', fontsize=14, title="Number of Properties Per Category", rot=0)
plt.show()
```

Result:



[Figure 3] Number per category: Type of property (Image by the author)

As [Figure 3] suggest, number of Condo (Apartments) and houses are significant part of our listing.

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

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

```
propertyListFrame['brokerName'] = propertyListFrame['brokerName'].str.strip()
type(propertyListFrame['brokerName'][0])
print(propertyListFrame['brokerName'].nunique())
df['TypeofProperty'].value_counts()
```

Result:

119

Condo 328

House 291

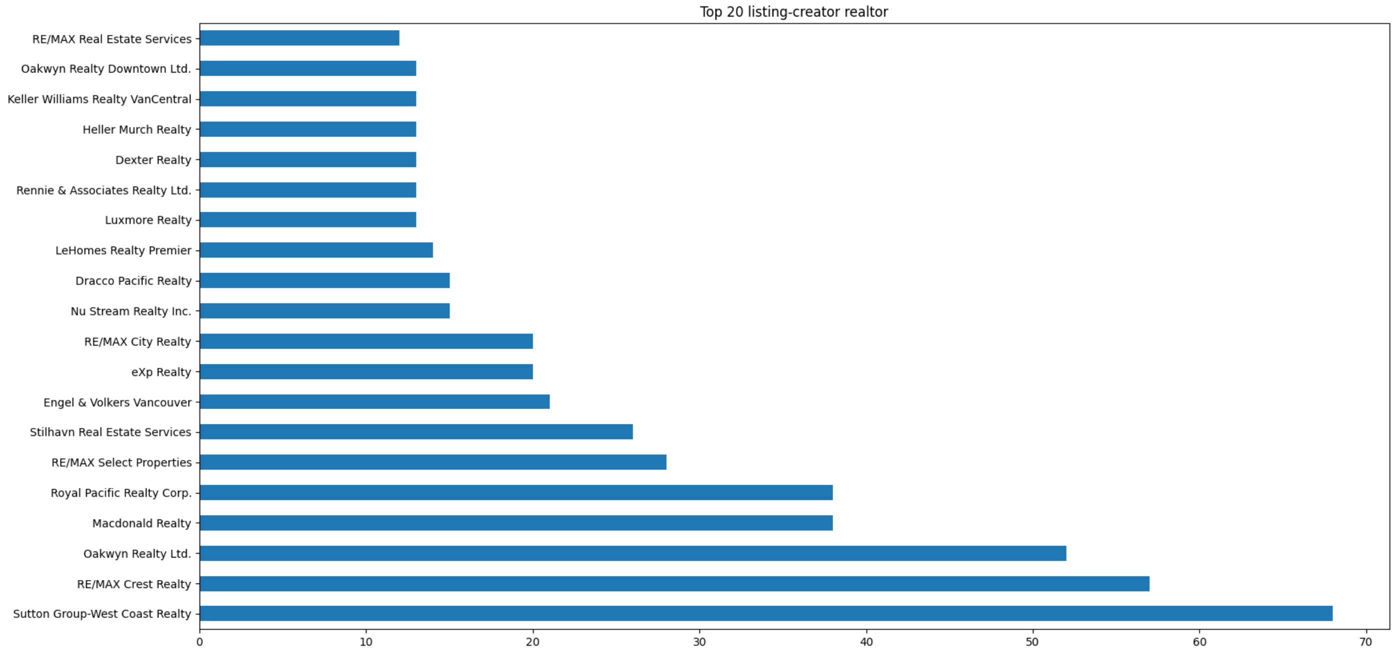
Townhouse 100

Multi-family home 80

As results shows, 119 individual brokers exist in our listing means some of them created more than one record in our property table. In continue, we plot top 20 realtor histogram [Figure 4].
We used matplotlib.pyplot to extract more helpful plots

```
df['brokerName'] = df['brokerName'].str.strip()
#print(df['brokerName'].nunique())    #print number of brokers
graph = df['brokerName'].value_counts()[:20]
graph.plot(kind='barh', fontsize=10, title="Top 20 listing-creator realtor")
plt.show()
```

Result:

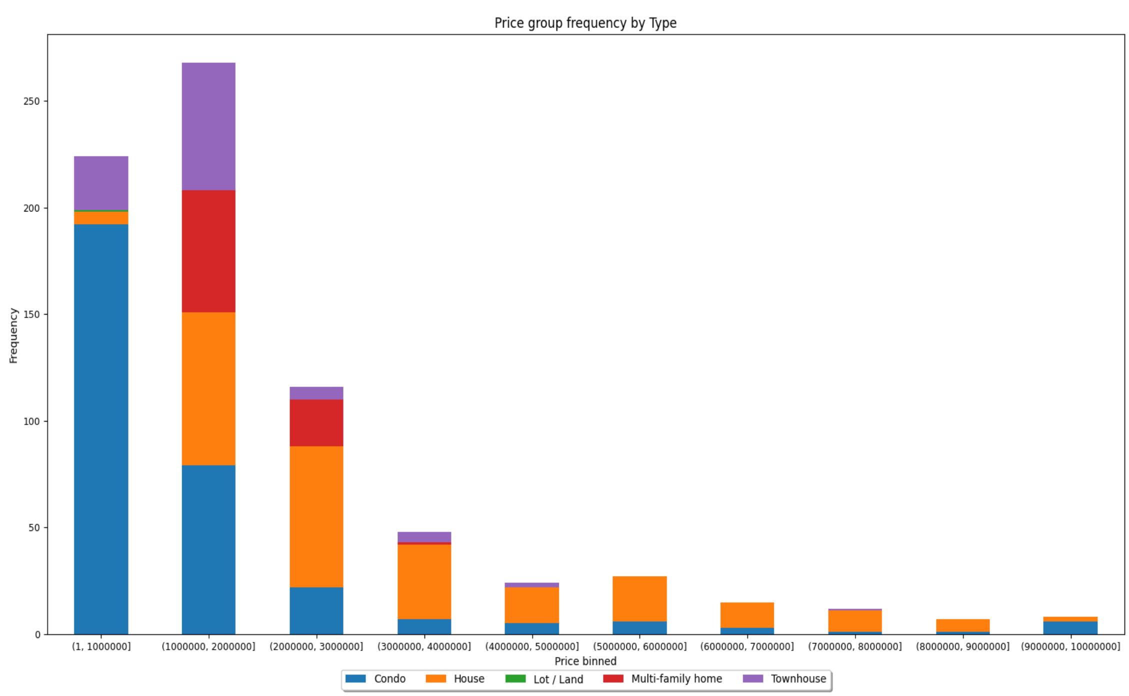


[Figure 4] Top 20 realtor histogram (Image by the author)

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

We continue by creating a price range histogram to observe number of each type of property. [Figure 5]

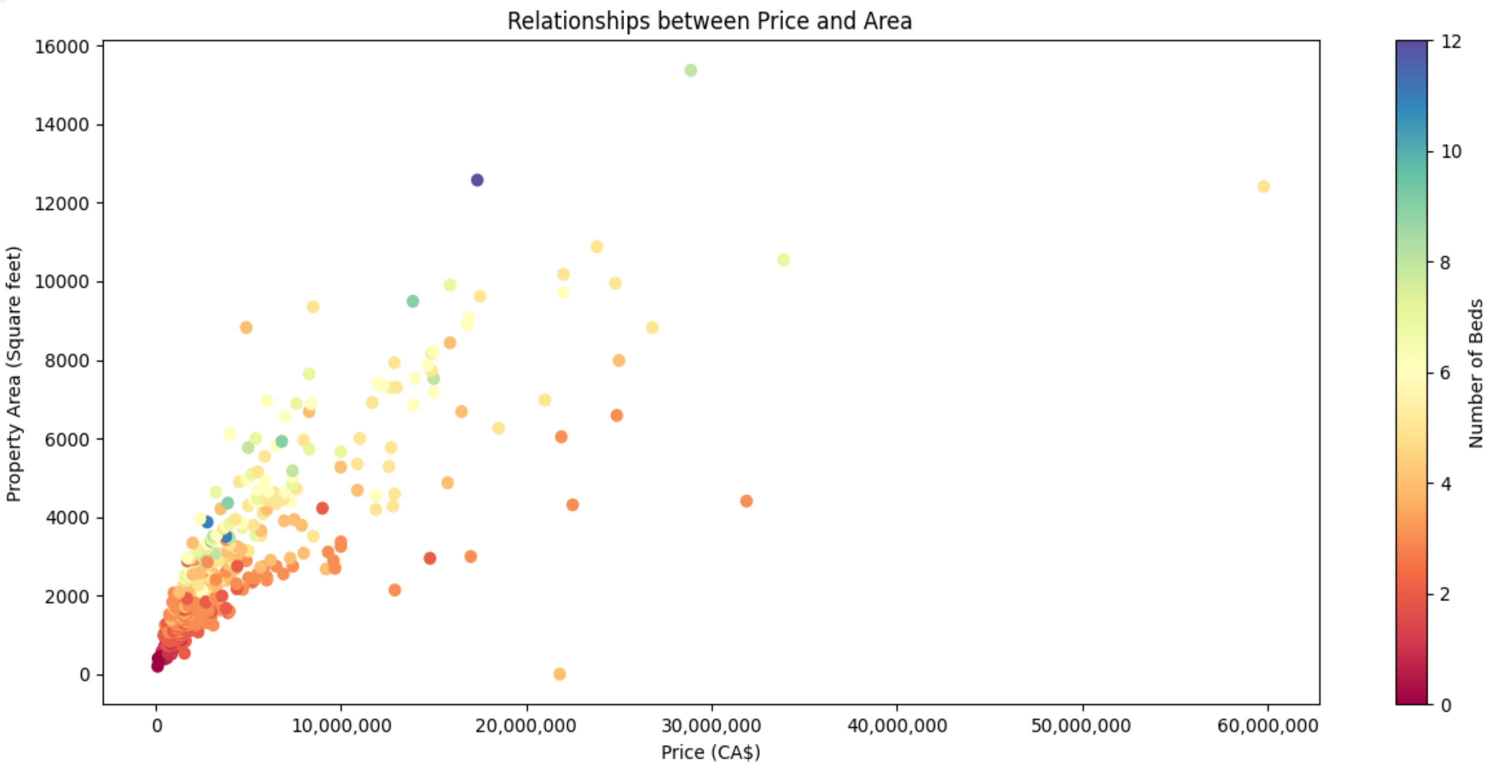
```
df1 = pd.read_csv('zillow.cs
df = df1[['Price', 'TypeofProperty' ]]
df['binned_price'] = pd.cut(df.Price, [1, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000, 10000000])
df.groupby('binned_price')['TypeofProperty'].value_counts()
df.drop('Price',axis=1,inplace=True)
df_reshaped = df.pivot_table(index=['binned_price'], columns=['TypeofProperty'], aggfunc=len)
df_reshaped.plot(kind='bar', stacked=True, ylabel='Frequency', xlabel='Price binned',title='Price group frequency by Type', rot=0, fontsize=9)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True, shadow=True, ncol=8)
plt.show()
```



[Figure 5] Histogram - Group by: Type of Property/ Price range science (Image by the author)

4- Scatter plot of distribution Statistics per category: **Area - number of beds** [Figure 6]

```
properties = pd.read_csv("zillow.csv")
plt.rcParams.update( {'figure.figsize':(10,8), 'figure.dpi':100})
plt.scatter(x=properties.Price, y=properties.area, c=properties.beds, cmap='Spectral')
plt.colorbar(label="Number of Beds")
plt.title('Relationships between Price and Area')
plt.xlabel('Price')
plt.ylabel('Property Area')
current_values = plt.gca().get_xticks()
plt.gca().set_xticklabels(['{:,.0f}'.format(x) for x in current_values])
plt.show()
```



[Figure 6] Scatter plot of distribution of price based of area and number of beds (Image by the author)

5- Geographical property distribution category: **Map – type of peroperty**

Finally, let's take a look to geographical property distribution in the Vancouver area [Figure 7]. For this purpose I used **folium** library. But it is necessary to do some pre-process to separate we used different color for each category of property. As you can see, it increase performance of visualization considerably.

Property location Analysis with Folium

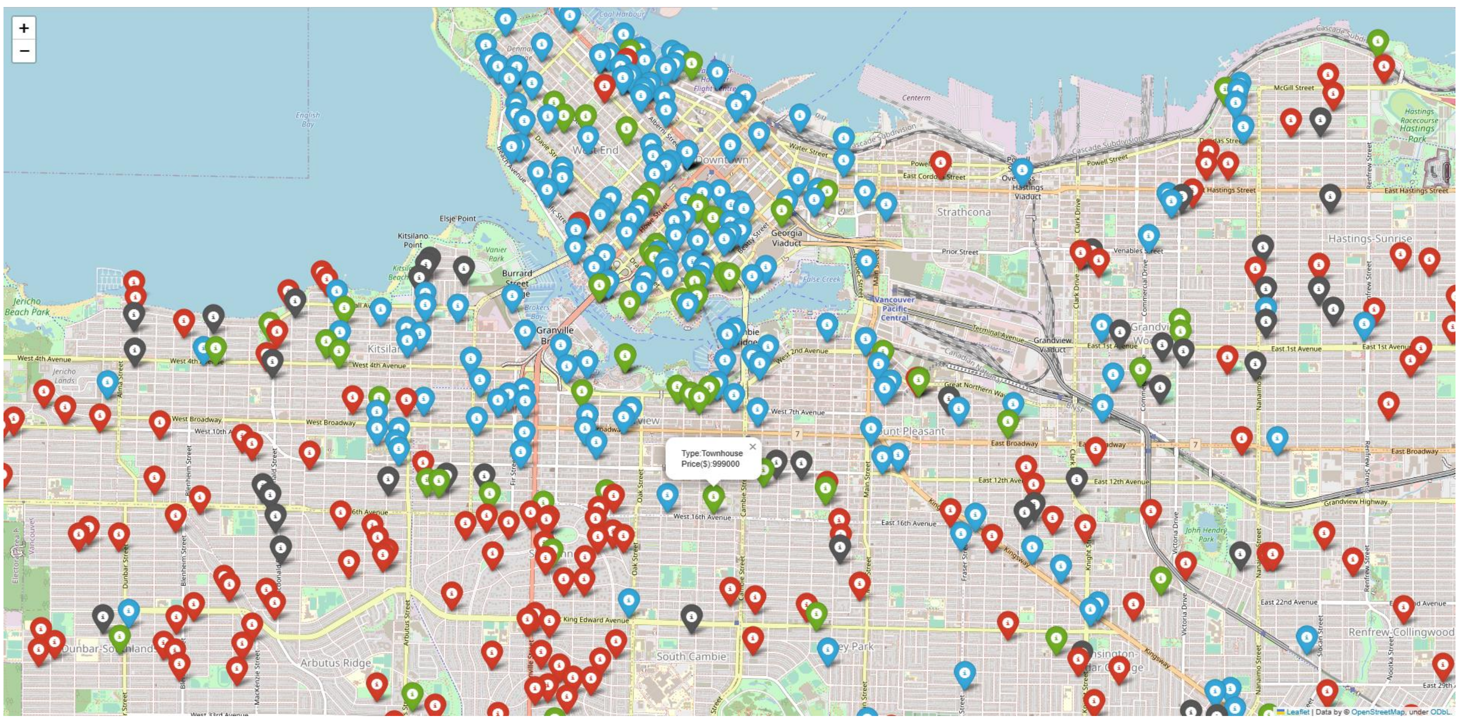
```python
import pandas as pd
import folium
from folium.plugins import MarkerCluster     # Import folium MousePosition plugin
from folium.plugins import MousePosition     # Import folium DivIcon plugin
from folium.features import DivIcon

df = pd.read_csv('zillow.csv')
drawable_df = df[df.Lat > 0.0]

mapit = folium.Map( location=[49.261505,  -123.05453], zoom_start=12 )
for index, drawable_df in drawable_df.iterrows():
    if      drawable_df["TypeofProperty"]=='Condo': c = 'blue'
    elif    drawable_df["TypeofProperty"]=='Townhouse':  c = 'green'
    elif    drawable_df["TypeofProperty"]=='Multi-family home': c = 'gray'
    else:  c = 'red'

    folium.Marker(
                 location = [drawable_df['Lat'], drawable_df['Long']],
                 radius=15 ,
                 icon=folium.Icon(color= c )  ,
                 popup = f'Type:{drawable_df["TypeofProperty"]}\n Price($):{drawable_df["Price"]}'
                 ).add_to(mapit)
mapit.save('map.html')
mapit.show_in_browser()
```

Result:



[Figure 7] Geographical property distribution in the Vancouver map (Image by the author)

6- Geographical Per Canada: **Map – Number**

This chart is not directly part of this project, but closely shows the ability of `matplotlib` in case of country distribution data. [Figure 8]

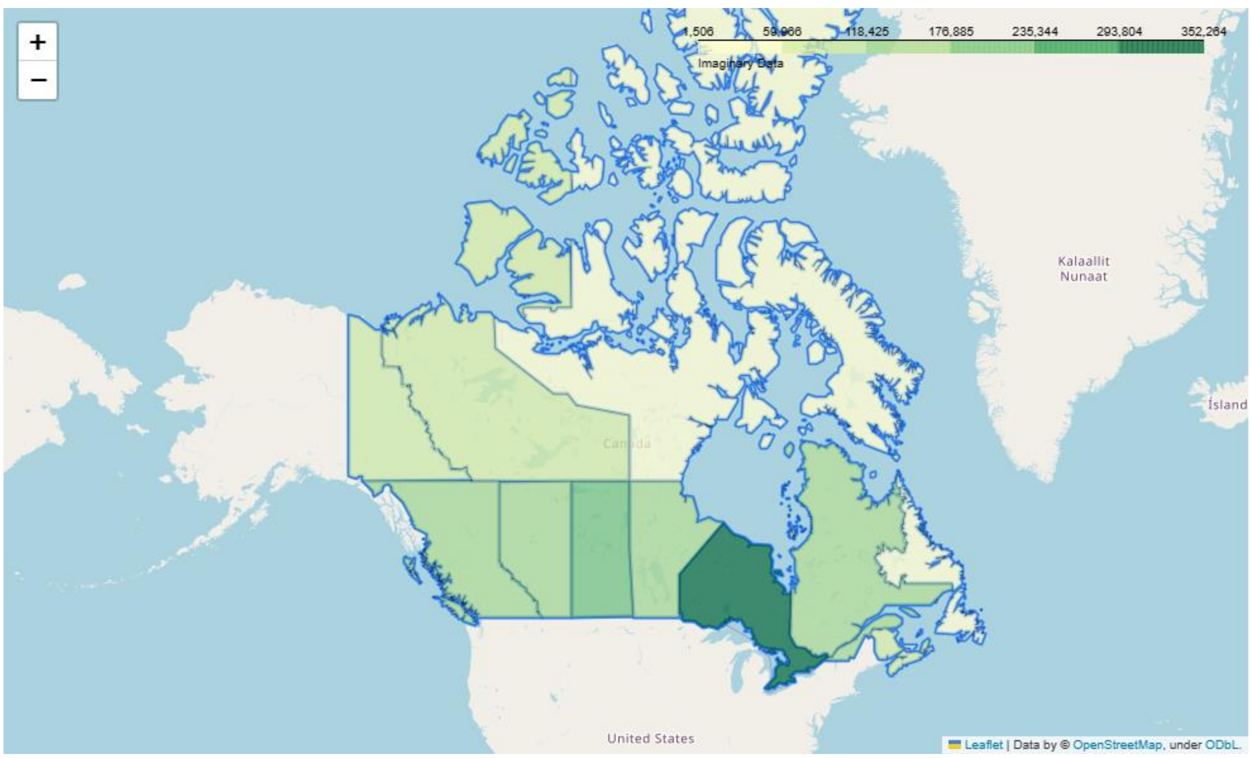
```python
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import folium
```

```python
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
}
```

```python
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
```

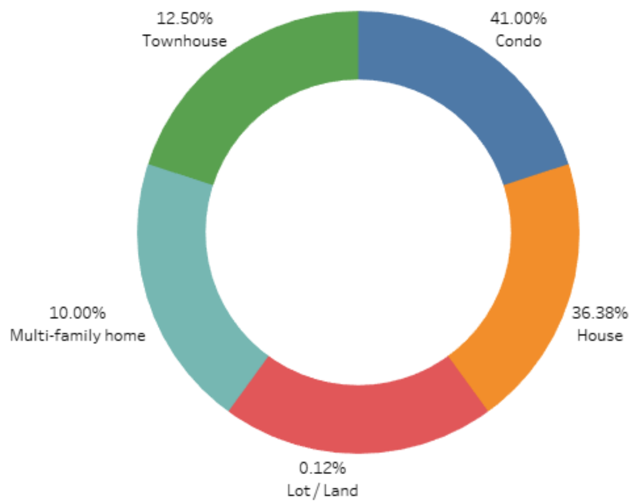


[Figure 8] Geographical distribution in Canada categorized by province (Image by the author)

3-2 Data visualization with Tableau

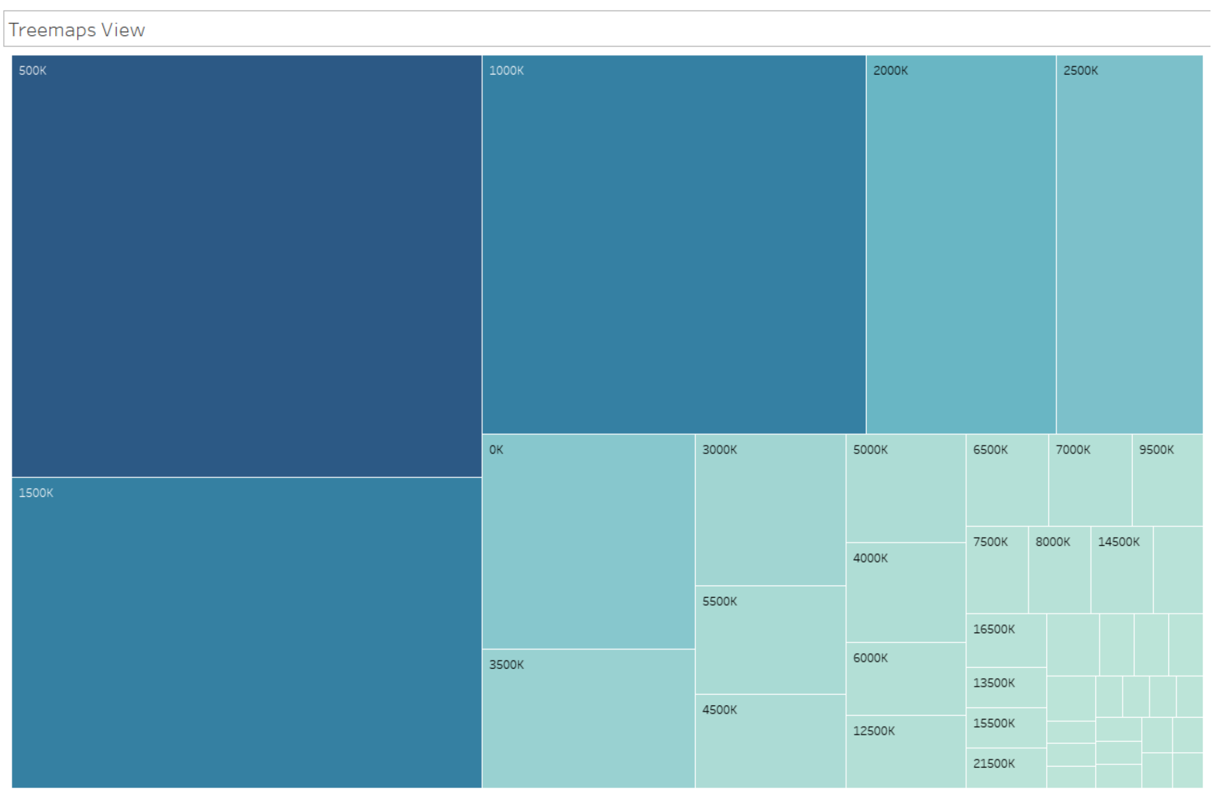
Tableau is a data visualization and business intelligence software that helps users to see and understand their data. It allows to create interactive dashboards, charts, and maps to gain insights from their data. It also has built-in collaboration features, so teams can share their findings and work 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 9]



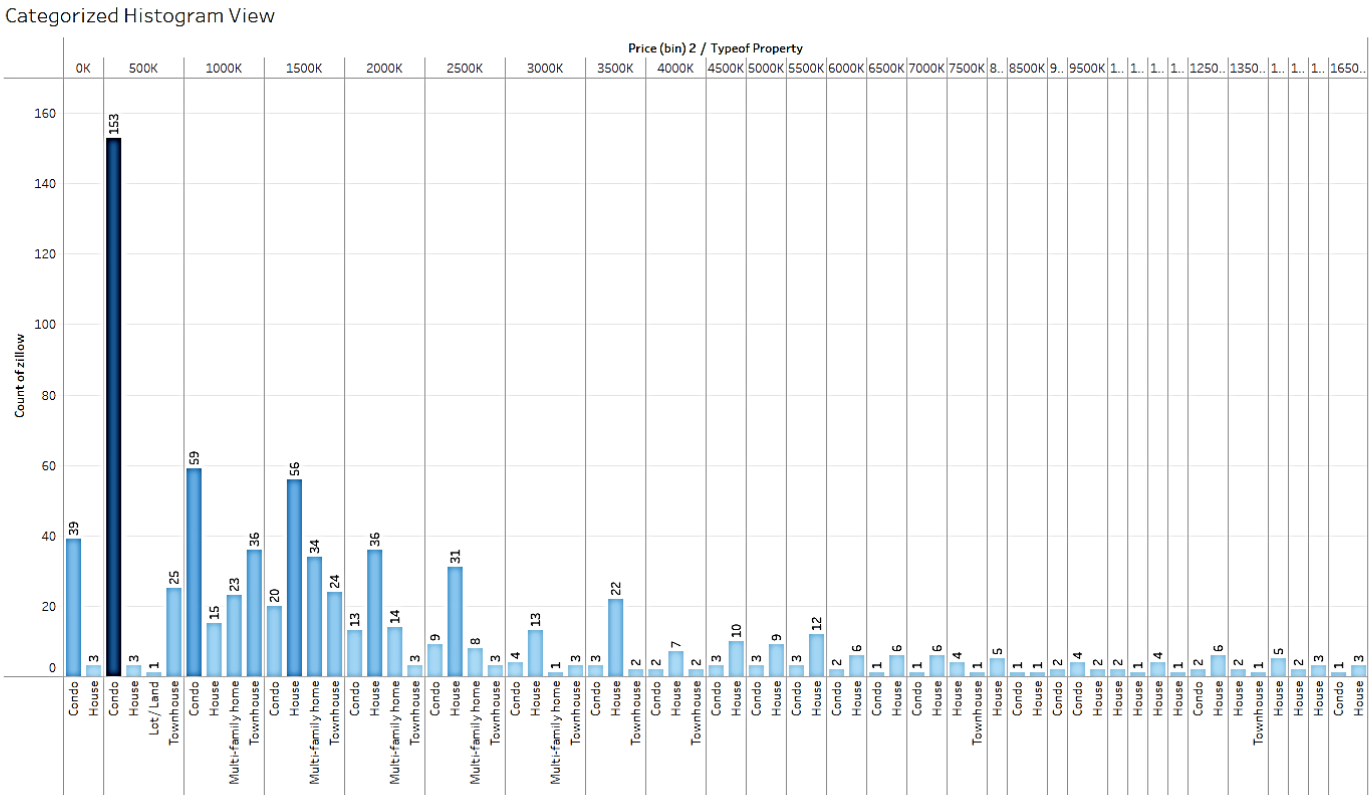
[Figure 9] Donut chart shows the percentage of each type of property for sale based on percentage of all (Image by the author)

Chart 2: Treemap view to demonstrations the distribution real estate based on price-range category [Figure 9]



[Figure 10] Treemap view shows the distribution based on price-range category (Image by the author)

Chart 3: Categorized histogram displays the distribution of listing based on Price (range)-Type category [figure 11]



[Figure 10] Categorized histogram view shows the distribution based on Price(range)-Type category (Image by the author)
[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

```
from pathlib import Path
import sqlite3

properties = pd.read_csv("zillow.csv")
Path('VancouverProperties.db').touch()
db_conn = sqlite3.connect('VancouverProperties.db')
db_cursor = db_conn.cursor()

properties.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
```

Result:

	index	zpid	id	imgSrc	...	latLong	has3DModel	brokerName	best_deal
0	614	2070285804	2070285804	https://photos.zillowstatic.com/fp/f67e9aa377a...	...	{'latitude': 49.286182, 'longitude': -123.11589}	0	None	1058000
1	49	2066436476	2066436476	https://photos.zillowstatic.com/fp/b0831a45b64...	...	{'latitude': 49.261505, 'longitude': -123.05453}	0	eXp Realty	419900

2	136	2067490635	2067490635	https://photos.zillowstatic.com/fp/76674f8faeb...	...	{}	0	Park Georgia Realty Ltd.	438800
3	411	2060499969	2060499969	https://photos.zillowstatic.com/fp/85669420760...	...	{'latitude': 49.243015, 'longitude': -123.05983}	0	Nu Stream Realty Inc.	449000
4	636	2060854306	2060854306	https://photos.zillowstatic.com/fp/91c1798b32d...	...	{'latitude': 49.286224, 'longitude': -123.14085}	0	Team 3000 Realty Ltd.	599900
...	...	...	...	...	...	...	...	...	...
795	323	314427630	314427630	https://photos.zillowstatic.com/fp/02c6a12b34b...	...	{'latitude': 49.289257, 'longitude': -123.12097}	0	Macdonald Realty	4350000
796	201	2061702036	2061702036	https://photos.zillowstatic.com/fp/cc0fd71a6dd...	...	{'latitude': 49.27208, 'longitude': -123.12038}	0	RE/MAX Crest Realty	3250000
797	750	314427938	314427938	https://photos.zillowstatic.com/fp/3dd1f4e2611...	...	{'latitude': 49.274944, 'longitude': -123.12518}	0	Macdonald Realty	4398000
798	632	2071582184	2071582184	https://photos.zillowstatic.com/fp/47de178e771...	...	{}	0	eXp Realty	7870000
799	549	314346775	314346775	https://photos.zillowstatic.com/fp/0d5302ad0c1...	...	{'latitude': 49.256756, 'longitude': -123.13778}	0	Royal Pacific Realty Corp.	13888000

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

```
data_stat = pd.read_sql(''' SELECT zpid, TypeofProperty, Price, beds, addressStreet, area
                            FROM VancouverPropertiesTable
                            WHERE addressStreet LIKE '%Kingsway%'
                            ORDER BY Price DESC ''', db_conn)
data_stat
```

Result:

0	2061711245	Condo	990000	3.0	2220 Kingsway #NE802	1140
1	2063168941	Condo	830000	2.0	2220 Kingsway #1612	812
2	2065888823	Condo	748888	2.0	760 Kingsway #310	880
3	2063497103	Condo	699000	2.0	2689 Kingsway #910	746
4	2060658439	Condo	659900	2.0	488 Kingsway #W407	780
5	2060941923	Townhouse	649800	1.0	2435 Kingsway #204	528
6	2060248755	Condo	618000	2.0	2973 Kingsway #102	861
7	2060723406	Condo	575000	1.0	1239 Kingsway #208	569
8	2060561370	Condo	459000	0.0	1432 Kingsway St #351	459
9	2060499969	Condo	449000	0.0	2239 Kingsway #109	415

Query 3: Average condo price (CA$) and area (Square feet)

```
data_stat_Avg = pd.read_sql(''' SELECT AVG(Price)AS Avg_price_condo , TypeofProperty, AVG(area) AS Avg_area
                                FROM VancouverPropertiesTable
                                WHERE TypeofProperty == 'Condo'  ''', db_conn)
```

Result:

Avg_price_condo	TypeofProperty	Avg_area
1.797490e+06	Condo	1129.859756

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.

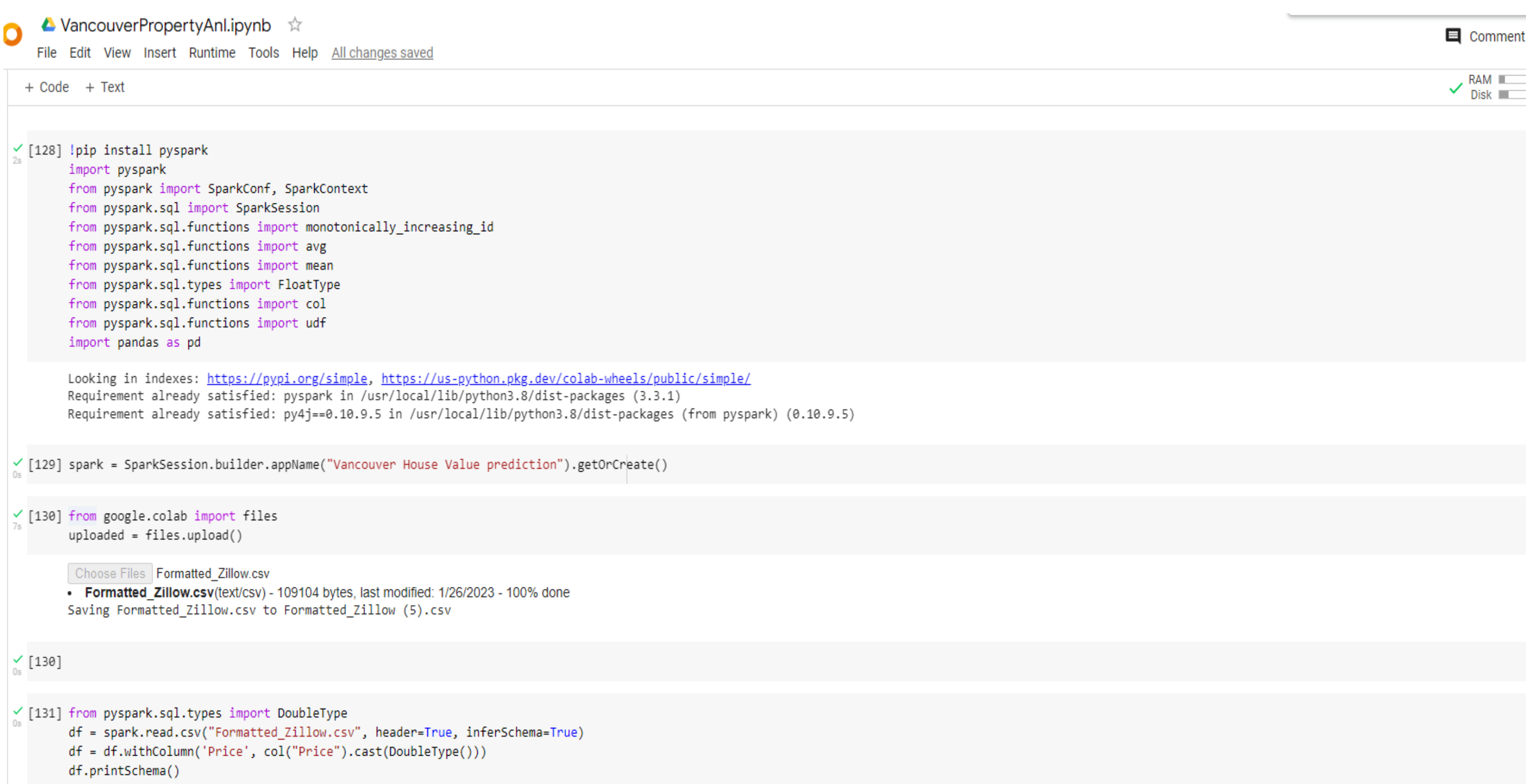
One of most important goals of data modeling is prediction, in our case: prediction of housing price. Here we used multivariable Linear Regression (LR) to predict target variable, and used a filtered attributes as bellow:

1- Numerical attributes: Price, N_Beds , Area, N_Baths, Lat, Long
2- Classified attribute: Type_Property, Address, Address_Street, Address_Zipcode, Broker_Name

5-1 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**. Is 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.

Here we also used **Google Colab**, a free, cloud-based platform for machine learning and data science experimentation. It allows users to write and execute code, as well as share and collaborate on projects, in a Jupyter notebook environment. It provides free access to GPU and TPU resources, making it a popular choice for training and experimenting with deep learning models. Colab also integrates with popular libraries and frameworks such as TensorFlow, Keras, and PyTorch, and can easily access and import data from Google Drive or Github.


VancouverPropertyAnl.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[128] !pip install pyspark
      import pyspark
      from pyspark import SparkConf, SparkContext
      from pyspark.sql import SparkSession
      from pyspark.sql.functions import monotonically_increasing_id
      from pyspark.sql.functions import avg
      from pyspark.sql.functions import mean
      from pyspark.sql.types import FloatType
      from pyspark.sql.functions import col
      from pyspark.sql.functions import udf
      import pandas as pd

      Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
      Requirement already satisfied: pyspark in /usr/local/lib/python3.8/dist-packages (3.3.1)
      Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.8/dist-packages (from pyspark) (0.10.9.5)

[129] spark = SparkSession.builder.appName("Vancouver House Value prediction").getOrCreate()

[130] from google.colab import files
      uploaded = files.upload()

      Choose Files Formatted_Zillow.csv
      • Formatted_Zillow.csv(text/csv) - 109104 bytes, last modified: 1/26/2023 - 100% done
      Saving Formatted_Zillow.csv to Formatted_Zillow (5).csv

[130]

[131] from pyspark.sql.types import DoubleType
      df = spark.read.csv("Formatted_Zillow.csv", header=True, inferSchema=True)
      df = df.withColumn('Price', col("Price").cast(DoubleType()))
      df.printSchema()
```

```
[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 Burøby St #6...| 1250 Burøby 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 Burøby St #6...| 1250 Burøby 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 Burøby St #6...| 1250 Burøby 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 Burøby St #6...| 1250 Burøby 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]
```python
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]
```python
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,-0.8641896927378544,0.39655358

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

[179]
```python
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]
```python
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]
```python
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

As can see in result, high MSE value indicates that the model is a poorer fit for the data and has a larger difference between the predicted and actual values. It is important to note that MSE is sensitive to the scale

of the target variable, thus if the target variable is on a large scale, then MSE will also be large. To compare models or compare with other datasets, one should use relative error metrics like RMSE and MAE.
The most important reason for this error includes:

1- Lake of sufficient data set: worthy note is the fact is very vital to know wide-ranging information regarding to each property to have a more precise price predictions. Example of predictor : area of living room (sqft), number of floors, waterfront (Yes/No), view (ranked), condition (ranked), grade (level in building for condo), area of above floor (sqft), area of basement (sqft), year of built, renovated (Yes/No), year of renovated are very important.
2- Small sample size: our data sets just includes 800 rows that is not enough to split them to train and test set [Figure 2].
3- Outliers: Linear regression is sensitive to outliers, meaning that a few extreme values can have a large impact on the model's parameters. If the data set contains outliers, linear regression may not be appropriate and robust regression methods should be used [Figure 6].

Section 6 Conclusion

In this study, 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.

Multivariable 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. Also, the F-test and R-square help us to understand the explanatory power of statistical models. 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.

In our data set case, using of LR is not recommended, because the direction of the correlation is not clear from the scatter plot. A scatter plot is a useful tool for visualizing the relationship between two variables and can provide insight into whether a linear relationship may exist. 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 6] shows a complex pattern or no clear relationship. So LR model is not appropriate.

Appendix

List of Figures

Keywords

Data pipline, Json, Big data, Data science, Data mining, Github pandas, GeoJson, machine learning, mathplotlib, scraping, pyspark, Apache spark, data visualization, Tableau, folium, seaborn, sqlite3, nampy, web scraping

The Tools

Anaconda, Python, SQL, Tableau (Public), Google Colab, Google Trends

Author
Kasra Heidarinezhad (Data scientist)
(604) 442-3332
Kasra.Heydarinezhad@gmail.com
www.linkedin.com/in/kasra-heidarinezhad
www.Github.com/kasraheidarinezhad