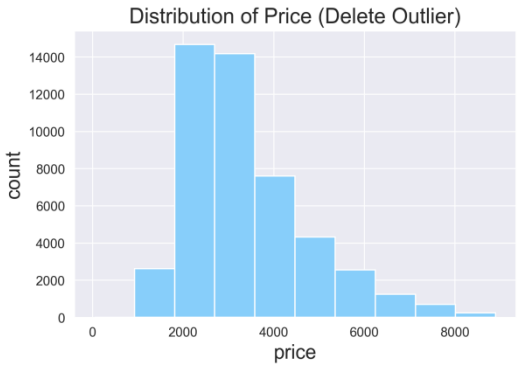
CMPT459 Data Mining Milestone1

1. Exploratory data analysis

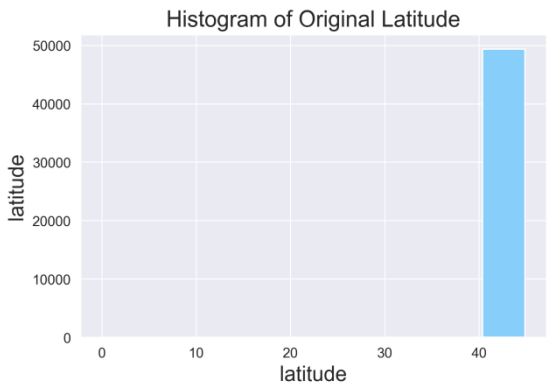
1.1 Plot Histograms for Price, Latitude, Longitude

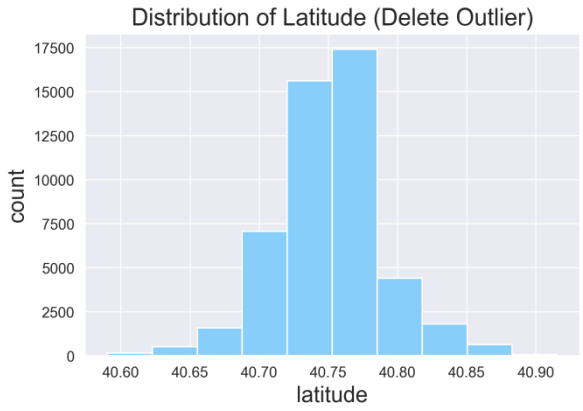
In this subsection, we provide histogram and distribution histograms of three variables: “price”, “longitude” and “latitude”. The plots in this part will help us to build a basic idea of these three variables. Meanwhile, in order to see the data closer, we also plot the histograms without outliers.

- Price

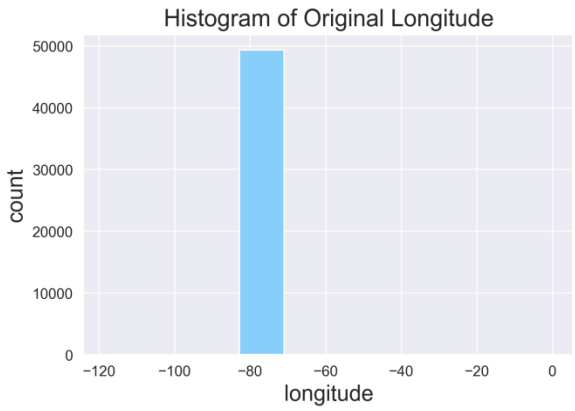


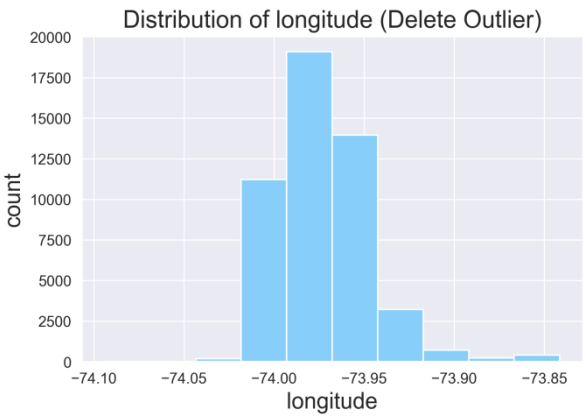
- Latitude



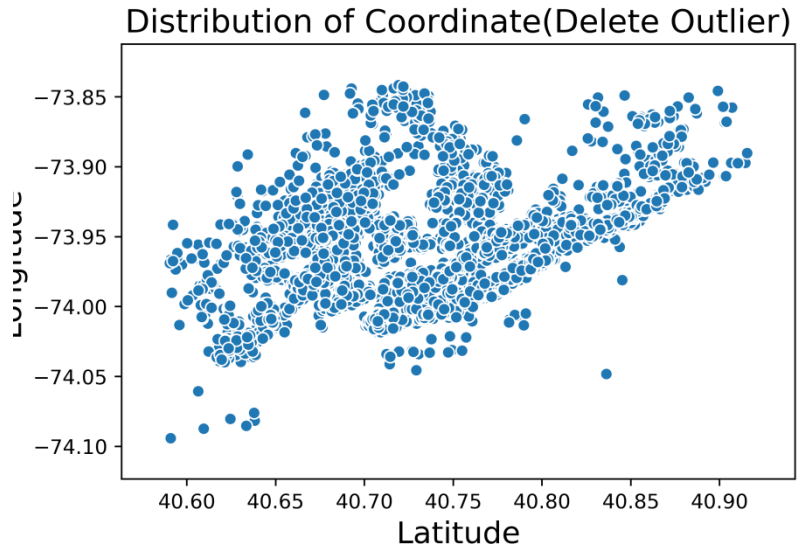


- Longitude

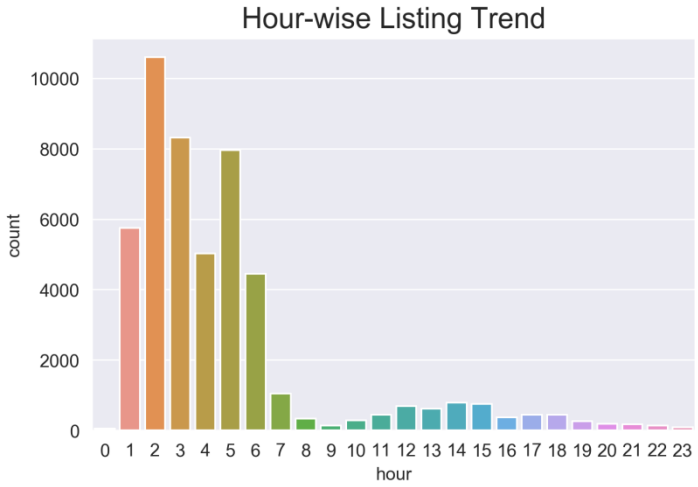




- Scatter of Latitude and Longitude



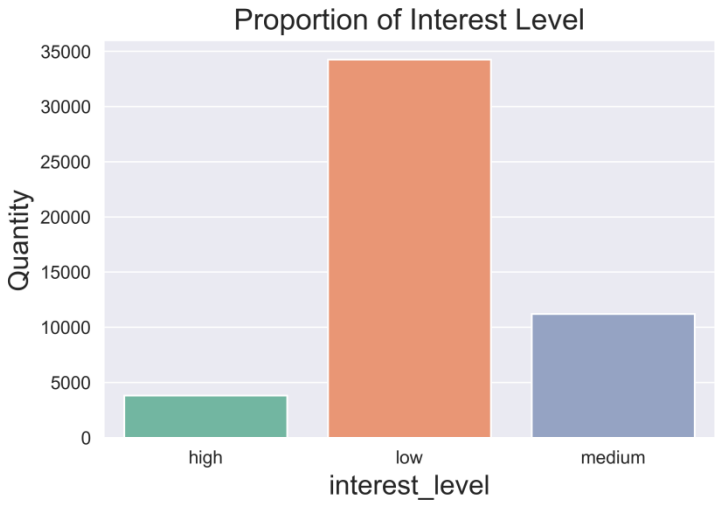
1.2 Plot hour-wise listing trend and find out the top 5 busiest hours of postings.



	hour	created
2	2	10596
3	3	8318
5	5	7954
1	1	5749
4	4	5021

	datehour	created
1122	2016-06-01-5	320
1376	2016-06-14-15	287
856	2016-05-17-14	277
1496	2016-06-21-7	224
1166	2016-06-03-5	212

1.3 Visualization to show the proportion of target variable values



2. Dealing with missing values and outliers

2.1 Find out the number of missing values in each variable.

I find that missing values cannot be found effectively by using function isnull(), so I search missing values for each attributes separately.

I regard building_id=0, latitude=0, longitude=0, length(feature)=0, length(photo)=0, description=“ ”, street_address=“ ”, display_address=“ ” as missing values.

This is the number of missing values:

Attribute	building_id	description	features	display_address
Number of Missing value	8286	1446	3218	135

Attribute	latitude	longitude	photos	street_address
Number of Missing value	12	12	3615	10

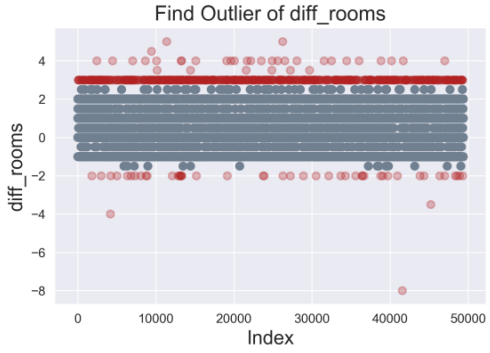
2.2 Find out the number of outliers in each variable. (Red Points are Outliers)

Tukey’s test: Calculate the first quatile Q1,the median Q2, the third quatile Q3, and let interQuatile range IQR be |Q3-Q1|. Then define the the values either Q1-3IQR or above Q3+3IQR as extreme outliers, the values either below Q1-1.5IQR or above Q3+1.5IQR as mild outliers.

In this part, we tried Tukey’s test to help to find a proper range to define outliers in different attributes. We only detect outliers for quantifiable attributes：Bathrooms, Bedrooms, Price, Latitude, Longitude.

- Bathroom & Bedroom

We think that when dealing with the outlier of bathroom and bedroom, they should be considered at the same time. It’s reasonable that one house with 8 bedrooms and 6 bathrooms. So we set a variable for the difference between bedroom and bathroom. Then, using outlier function, we find the Outlier(bedrooms & bathrooms) = 616. We will remove these 616 data, because it’s unreasonable and strange that a house has 6 bedrooms and 1 bathroom.



- Price

By the outlier function, we find the number of price outlier is 1223, whose price higher than 8900.However, by observing the price histogram, we think that it is better to set the price greater than 20000 as the outlier. Therefore, the Outlier(Price) = 109. For these 109 outliers, ridiculously high prices, we removed them.

- Latitude & Longitude

The results of Tukey’s test on longitude and latitude:

Latitude: Q1(25%) = 40.7283 Q2(50%) = 40.7518 Q3(75%) = 40.7743
number of mild outliers (> 40.8433 or < 40.6593) : 1932
number of extreme outliers (> 40.9123 or < 40.5903) : 147

Longitude: Q1(25%) = -73.9917 Q2(50%) = -73.9779 Q3(75%) = -73.9548
number of mild outliers (> -73.89945 or < -74.04705 : 1102
number of extreme outliers (> -73.8441 or < -74.1024) : 411

- Combine longitude and latitude

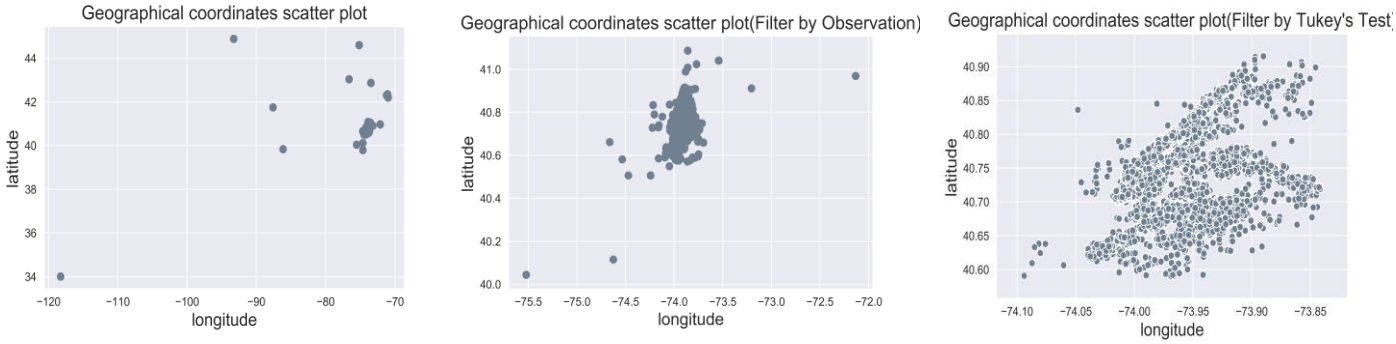


Figure 1: scatter plot of longitudes latitudes of the raw data without missing values

Figure 2: $40 \leq latitude \leq 41.5, -80 \leq longitude \leq -70$. (filter by observation)

Figure 3: $40.59 \leq latitude \leq 40.92, -74.10 \leq longitude \leq -73.84$. (filter by Tukey's test)

By comparing the three pictures above, we can see that the distribution in Figure 3 become quite reasonable. And total number of extreme outliers filtered is 464. We remove these 464 outliers which the houses locate far away from most of the houses.

2.3 Deal with Missing values

- If the missing data is class label (like **interest_level** in this dataset), we can drop that data, because it will not help us in classification.
- If the data are attributes like **building_id**, we can fill it with **"unknown"** or **"NaN"**, because this type of data, which is used for identifying the object we are analyzing (in this case, **house**) helps little in classification.
- If the data are categorical, boolean or numerical data that mean a lot to our classification, we will use **the most probable value** to fill the missing data (regression method). For example, we can try to figure out the missing **price** according to the number of **bedrooms** and **bathrooms** of this listing.
- (continued) If the most probable value is hard to obtain, we can use the attribute **mean** or **median** to fill the missing value, like the case of **longitude** and **latitude**.
- As for other types of missing values, like **photos** and **descriptions**, we can leave it to **feature extraction** first and dealing it after robust features are extracted. In most cases, it is reasonable that some houses are no photo or description, so we can ignore them as well.

Ignore or fill with "NaN"

Attribute	building_id	description	features	display_address
Number of Missing value	8286	1446	3218	135

Attribute	latitude	longitude	photos	street_address
Number of Missing value	12	12	3615	10

Delete or fill with median or mean

Ignore or fill with "NaN"

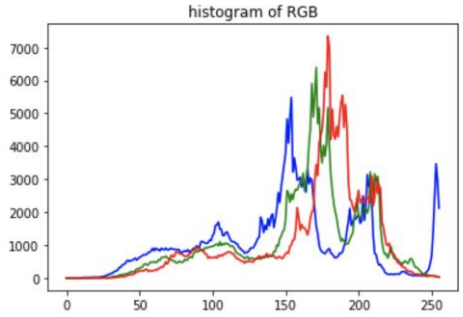
3. Feature extraction from images and text

3.1 Extract features from the images

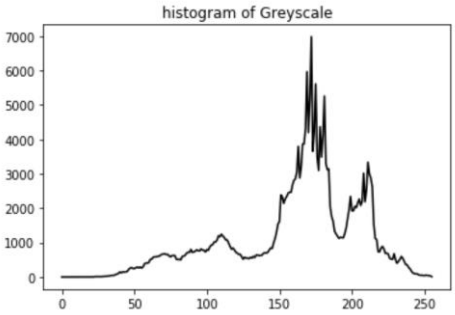
1. Histogram of colors

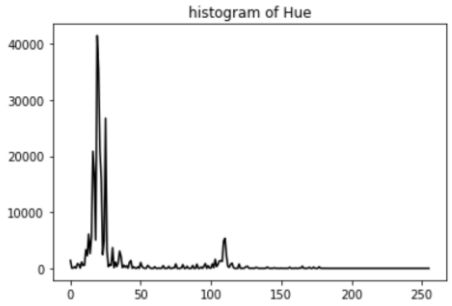
In this part, we extracted several histograms and some corresponding features of the image, with python's tool *cv2.calcHist.*

- Histogram of RGB and greyscale



- Convert the image into greyscale and HSV space





In this part, we also computed the mean and standard deviation of the greyscale graph to have a knowledge on the color distribution. And we record the hue value with the maximum number of pixels, as the main hue of the image.

- Record the features (one row per folder)

	photoNum	RGBHists	greyHists	meanGrey	stdGrey	maxHue
6812051	4	[[[[272.], [561.], [1036.], [1486.], [2182.], ...	[[[2.0], [18.0], [23.0], [83.0], [248.0], [322...	[90.12154622395833, 105.38919596354167, 108.48...	[57.18824893024921, 52.73708558902206, 65.8400...	[57.18824893024921, 52.73708558902206, 65.8400...
6812263	6	[[[[0.], [0.], [0.], [0.], [0.], [0.], [0.], [...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...	[145.28329561281336, 158.02139188718664, 139.9...	[34.13911675322706, 30.348497805058706, 37.375...	[34.13911675322706, 30.348497805058706, 37.375...
6812264	2	[[[[157.], [129.], [152.], [177.], [216.], [35...	[[[0.0], [17.0], [60.0], [124.0], [195.0], [33...	[133.07743019311064, 165.72943808685446]	[64.58462088575449, 52.235490666962164]	[64.58462088575449, 52.235490666962164]
6812002	0	[]	[]	[]	[]	[]
6812005	0	[]	[]	[]	[]	[]
6812208	0	[]	[]	[]	[]	[]
6812201	4	[[[[45.], [16.], [29.], [66.], [86.], [99.], [...	[[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0...	[127.97053990610328, 161.30269953051643, 136.3...	[62.21510358825447, 33.98761893619457, 54.5228...	[62.21510358825447, 33.98761893619457, 54.5228...

photoNum – number of photos in the folder

RGBHists, greyHists – Lists of RGB and greyscale histograms of each photo in this folder

meanGrey, stdGrey, maxHue – List of each value of each photo in this folder

2. Edge Detection using Prewitt kernel

We could identity edges by finding the place of value changes in the pixel table (for example, grayscale or RGB). Prewitt kernel uses values surrounding the selected pixel and multiply it with the kernel, then sum up to get a final value for this pixel.

-1	0	1
-1	0	1
-1	0	1

Prewitt Kernel
X Direction

-1	-1	-1
0	0	0
1	1	1

Prewitt Kernel
Y Direction

Table : Prewitt Kernel in X and Y Direction

This could be achieved by using prewitt_h and prewitt_v, two filter functions in skimage packet. What we have done is:

- First, load an image and convert it to grayscale.
- Second, use Prewitt kernel to detect the edges. Below is one example.

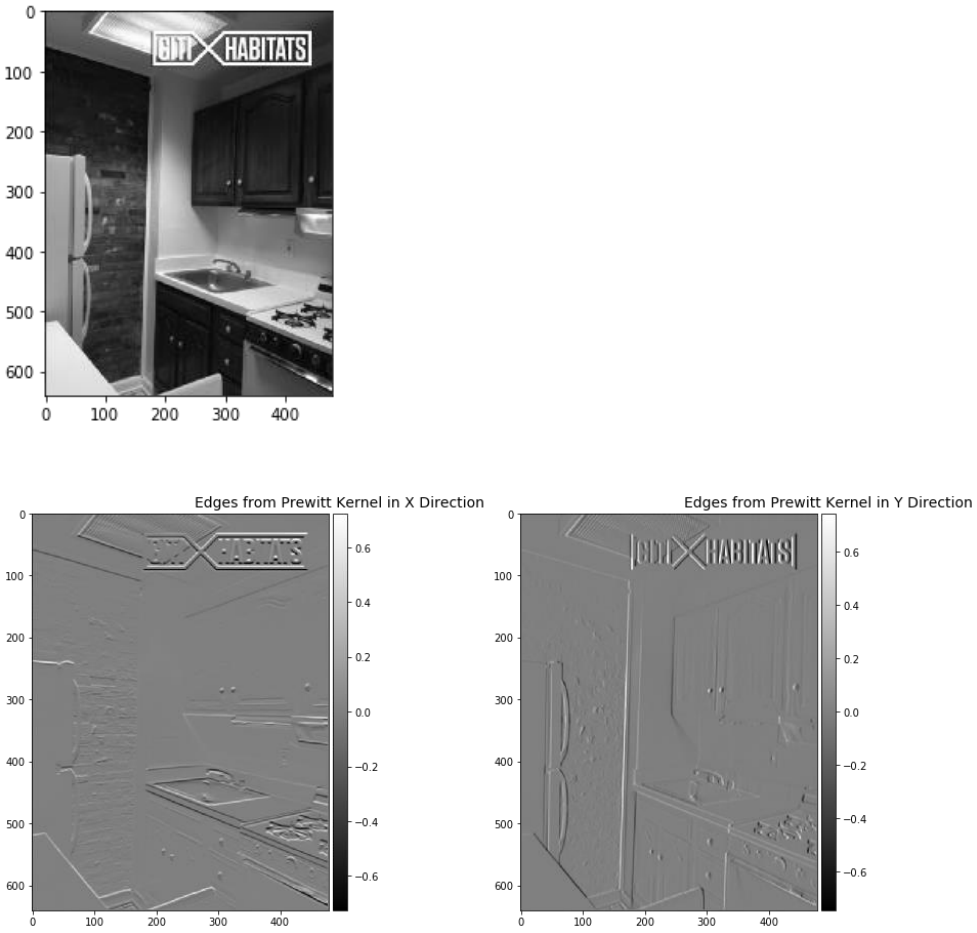


Figure : Example of Horizontal and Vertical Edge Detection

- Third, by parsing all the images in each folder, we can get a table of data we obtain from Prewitt kernel.

	Folder Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	6811957	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None
1	6811958	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None	None
2	6811960	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None
3	6811964	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None
4	6811965	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	6812258	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None	None
96	6812259	None	None	None	None	None	None	None	None	None	None	None	None	None	None
97	6812263	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None
98	6812264	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None	None	None	None
99	6812266	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	None	None	None	None	None	None	None	None	None	None	None	None

100 rows × 15 columns

Table : Dataframe of Result from Edge Detection

With this table, we can choose an image and find its edges by locating the data in the table.

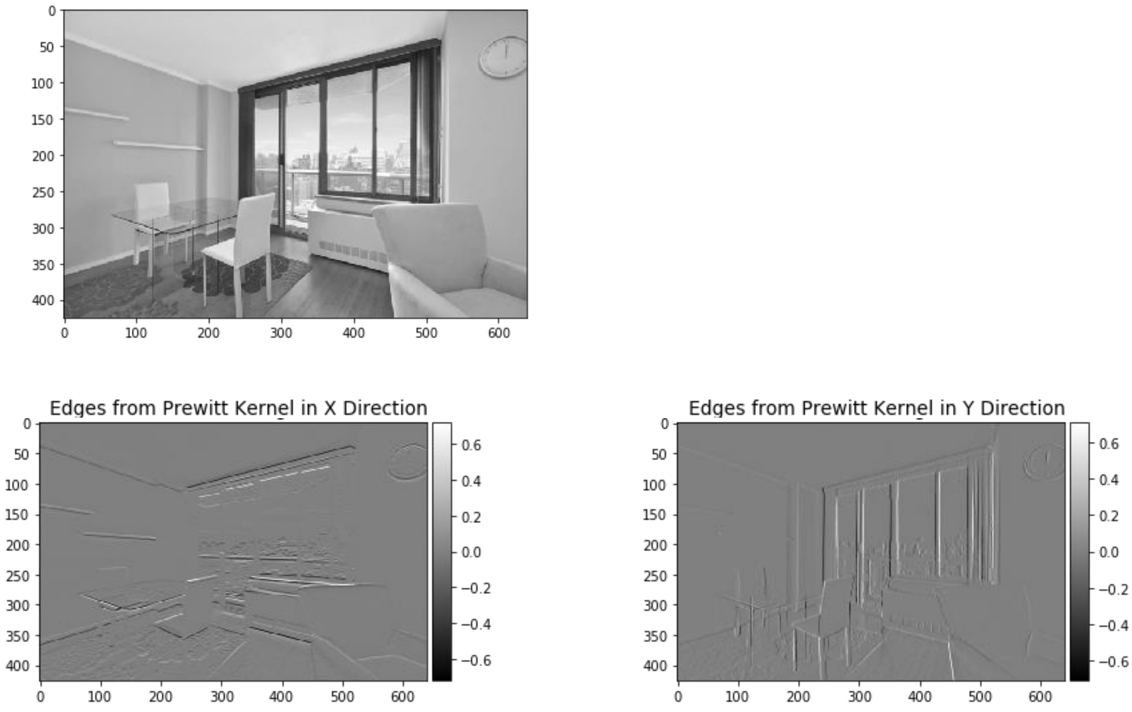


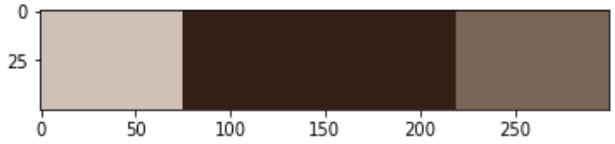
Figure : Result obtained by locating [1][0] in the table above
(the 1^{nd} image in the 2^{nd} folder)

3. Find Primary Colors

To find the primary colors, we apply k-means clustering. This algorithm separates the original n data into k clusters. Data in the same cluster are regarded as "more similar" than data outside this cluster. So, it basically takes 2 steps to find the primary colors: first, cluster the RBG data of an image and second, calculate relative frequency for each cluster.

- First, load an image as RGB
- Apply K-means clustering algorithm to cluster the data into 3 clusters.
- Compute relative frequency by normalizing the number of data in each cluster.
- Take the cluster center as the data to represent each cluster.
- Plot a bar to show the result

Below is an example:



	R	B	G	Relative Frequency
0	208	193	180	0.25
1	51	31	22	0.48
2	121	102	86	0.27

Figure : Example of Finding Primary Colors

- Last, we computer tables of RGB value and relative frequency of the 3 primary color for each image. The table show below:

	Folder Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	6811957	[[160, 160, 159], [94, 88, 72], [224, 224, 224]]	[[97, 81, 69], [167, 162, 159], [216, 217, 221]]	[[163, 163, 162], [119, 119, 119], [206, 209, ...	[[181, 177, 176], [150, 115, 88], [220, 224, 2...	[[213, 214, 214], [171, 158, 145], [119, 98, 76]]	None	None	None	None	None	None	None	None	None
1	6811958	[[101, 92, 77], [177, 166, 145], [207, 209, 211]]	[[136, 149, 169], [65, 60, 56], [207, 211, 215]]	[[252, 252, 252], [5, 5, 5], [131, 131, 131]]	[[187, 183, 175], [75, 60, 50], [145, 131, 106]]	None	None	None	None	None	None	None	None	None	None
2	6811960	[[203, 195, 182], [77, 49, 30], [144, 122, 100]]	[[95, 82, 64], [229, 236, 232], [139, 126, 104]]	[[110, 101, 91], [218, 234, 243], [140, 144, 1...	[[86, 73, 54], [142, 129, 104], [220, 225, 215]]	[[206, 180, 127], [106, 91, 68], [214, 233, 237]]	[[226, 211, 200], [157, 146, 157], [131, 71, 35]]	None	None	None	None	None	None	None	None
3	6811964	[[223, 224, 226], [140, 134, 127], [65, 56, 49]]	[[207, 179, 145], [251, 251, 250], [94, 72, 57]]	[[145, 141, 133], [214, 213, 207], [54, 56, 52]]	[[121, 111, 103], [211, 212, 214], [64, 54, 46]]	[[114, 112, 116], [184, 180, 182], [34, 33, 35]]	None	None	None	None	None	None	None	None	None
4	6811965	[[102, 103, 98], [58, 54, 42], [151, 158, 163]]	[[82, 64, 42], [210, 209, 207], [124, 108, 87]]	None	None	None	None	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	6812258	[[105, 84, 56], [130, 124, 113], [183, 207, 217]]	[[149, 143, 131], [77, 70, 60], [194, 189, 177]]	[[81, 51, 25], [202, 204, 201], [135, 109, 80]]	[[64, 57, 52], [217, 210, 196], [158, 149, 133]]	None	None	None	None	None	None	None	None	None	None
96	6812259	None	None	None	None	None	None	None	None	None	None	None	None	None	None
97	6812263	[[105, 92, 82], [171, 169, 173], [144, 136, 128]]	[[176, 162, 150], [213, 208, 223], [148, 122, ...	[[57, 51, 51], [198, 196, 198], [143, 117, 91]]	[[151, 147, 147], [127, 109, 95], [219, 223, 2...	[[117, 97, 81], [200, 194, 193], [170, 155, 139]]	[[175, 167, 154], [87, 71, 50], [152, 143, 129]]	None	None	None	None	None	None	None	None
98	6812264	[[187, 147, 125], [45, 38, 38], [234, 223, 222]]	[[228, 226, 225], [143, 92, 49], [185, 166, 149]]	None	None	None	None	None	None	None	None	None	None	None	None
99	6812266	[[51, 31, 22], [120, 102, 86], [208, 193, 180]]	[[69, 37, 21], [121, 86, 60], [200, 206, 211]]	None	None	None	None	None	None	None	None	None	None	None	None

100 rows × 15 columns

Table : Three Primary Color in Each Image (RGB)

	Folder Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	6811957	[0.54, 0.08, 0.38]	[0.13, 0.46, 0.41]	[0.38, 0.22, 0.39]	[0.41, 0.25, 0.34]	[0.59, 0.29, 0.12]	None	None	None	None	None	None	None	None	None
1	6811958	[0.22, 0.53, 0.25]	[0.31, 0.4, 0.29]	[0.86, 0.1, 0.04]	[0.62, 0.15, 0.23]	None	None	None	None	None	None	None	None	None	None
2	6811960	[0.58, 0.08, 0.34]	[0.39, 0.24, 0.37]	[0.62, 0.1, 0.27]	[0.41, 0.4, 0.19]	[0.13, 0.77, 0.1]	[0.4, 0.47, 0.14]	None	None	None	None	None	None	None	None
3	6811964	[0.2, 0.64, 0.16]	[0.68, 0.22, 0.1]	[0.46, 0.24, 0.3]	[0.55, 0.18, 0.27]	[0.3, 0.52, 0.18]	None	None	None	None	None	None	None	None	None
4	6811965	[0.44, 0.45, 0.11]	[0.74, 0.08, 0.19]	None	None	None	None	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	6812258	[0.47, 0.37, 0.15]	[0.36, 0.1, 0.54]	[0.24, 0.24, 0.52]	[0.21, 0.37, 0.43]	None	None	None	None	None	None	None	None	None	None
96	6812259	None	None	None	None	None	None	None	None	None	None	None	None	None	None
97	6812263	[0.29, 0.38, 0.33]	[0.57, 0.11, 0.32]	[0.33, 0.28, 0.39]	[0.43, 0.47, 0.1]	[0.31, 0.26, 0.42]	[0.46, 0.14, 0.39]	None	None	None	None	None	None	None	None
98	6812264	[0.61, 0.27, 0.11]	[0.29, 0.32, 0.39]	None	None	None	None	None	None	None	None	None	None	None	None
99	6812266	[0.48, 0.27, 0.25]	[0.42, 0.51, 0.07]	None	None	None	None	None	None	None	None	None	None	None	None

100 rows × 15 columns

Table : Relative Frequency of the Three Primary Color in Each Image

To use the above tables, we can locating values in the same place of the 2 tables, which is a pair of (color, relative frequency), and then use the function *plot_colors* to see the three primary colors and their relative frequency of this image. Here is an example:

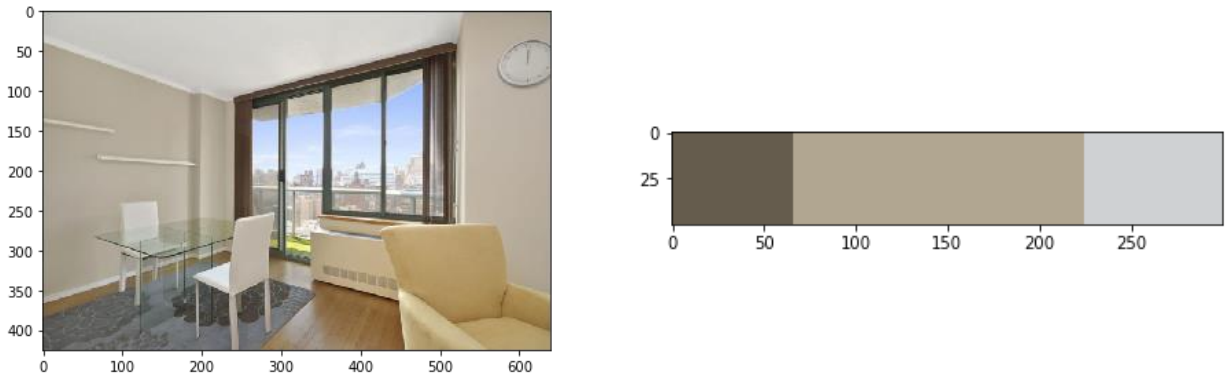


Figure : Result obtained by locating [1][0] in the tables above
(the 1nd image in the 2nd folder)

3.2 Extract features from the text data

We think the text data include 'description' and 'features':

- Description: describe the house with sentence, dtype = string
- Features: describe the house with special label, dtype = list

We clean up the text first, and then create the vector space through the function Tfidfvector() and the function doc2vec(). Besides, we also use wordcloud and artificial extraction. Here are the details about feature extraction.

3.3.1 Description

a. Clean Data

We use regular expressions to replace the text content in 'description', including acronyms, symbols

```
def clean_text(text):
    # acronym
    text = re.sub(r"br\s", "bedroom", text)
    text = re.sub(r"Ave", "avenue ", text)
    text = re.sub(r"n\'t", " not", text)
```

Part code of clean_text()

b. TfidfVector() Analysis

TF - Term Frequency, IDF - Inverse Document Frequency. In short, the Tfidfvector() function can calculate the importance of a word in the whole corpus. This is our result:

```
In [37]: desc_tfidf.vocabulary_

Out[37]: {'spacious': 161,
          'bedroom': 15,
          'bathroom': 12,
          'apartment': 5,
          'features': 64,
          'renovated': 142,
          'kitchen': 95,
          'dishwasher': 44,
          'beautiful': 14,
```

the vocabularies of description

```
data['description_vector']

0        [0.0, 0.0, 0.0, 0.0, 0.0, 0.11263406846043605,...
1        [0.0, 0.0, 0.0, 0.0, 0.0, 0.18452343697075432,...
2        [0.0, 0.0, 0.0, 0.0, 0.0, 0.2184616774192138, ...
3        [0.0, 0.0, 0.0, 0.0, 0.0, 0.11968657472640343,...
4        [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
                                ...
49347    [0.0, 0.0, 0.0, 0.0, 0.0, 0.07784526591216742,...
49348    [0.34680473146403, 0.34684592351133686, 0.1424...
49349    [0.0, 0.0, 0.0, 0.0, 0.16127020904072745, 0.09...
49350    [0.0, 0.0, 0.0, 0.0, 0.0, 0.09374826688913933,...
49351    [0.0, 0.0, 0.0, 0.0, 0.0, 0.12546893962900188,...
```

the vector space of description

c. doc2vec() Analysis (Emotion Analysis for sentence)

First, we need to convert the description data to tagged document, which is suitable for the function doc2vec(). Then we can create a model named doc2vec which can reflect the similarity between texts.

```
doc2vec[2103]

array([-2.04182491e-02, -4.21266183e-02,  3.11049353e-02,  4.07856032e-02,
        5.55825382e-02, -1.98588837e-02, -3.27888466e-02, -3.62423202e-03,
        2.35620178e-02, -3.34877484e-02, -2.25036573e-02,  5.29708490e-02,
        2.30318103e-02, -3.28535661e-02, -1.18135111e-02,  6.27262592e-02,
        2.28677294e-03,  3.50758433e-02, -2.63380203e-02,  3.55886444e-02,
       -2.55907159e-02, -1.49148861e-02,  6.58132508e-02,  3.08807716e-02,
        8.26888382e-02,  2.54716538e-02, -9.34129488e-03,  7.21487105e-02,
       -5.39037846e-02, -3.41544710e-02,  3.64234410e-02, -1.11292906e-01,
       -2.18932424e-02,  1.37084508e-02, -3.75535078e-02, -6.55268924e-03,
```

The model doc2vec

d. Artificial Extraction

In artificial feature extraction, we first classify a small number of high-frequency words, and then extract features from the data. For example, we regard ‘beautiful, great, luxury’ as ‘positive’.

desc_positive	desc_apartment	desc_security	desc_elevator	desc_stainless	desc_steel'	desc_dishwasher	de
0	0	0	1	1	1	1	
1	1	0	1	1	1	1	
1	1	1	1	1	1	1	
1	1	0	1	1	1	1	

Artificial Extraction of Description

3.3.2 Features

a. Clean Data

The data of features has been labeled, so it is much easier for us to clean the data. Because some words exist in the form of phrases, we first process phrases to ensure their integrity, such as ‘cat allowed’. Then, we put all words into a string to construct a corpus, which makes it convenient for us to carry out word frequency analysis.

b. TfidfVector Analysis

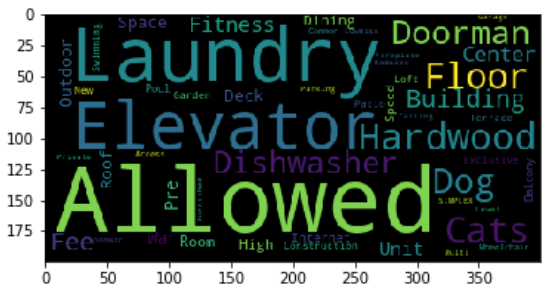
```
In [70]: phr_names
Out[70]: ['24hrdoorman',
          'actualapt',
          'airconditioning',
          'allutilitiesincluded',
          'assigned',
          'attendedlobby',
          'backyard',
          'balconies',
          'balcony',
          'basementstorage',
          'bikeroom',
          'bikestorage',
          'billiardsroom',
          'brownstone',
```

Phrase Analysis

```
In [72]: wd_tfidf.get_feature_names()
Out[72]: ['1br',
          '24hr',
          '2br',
          '3br',
          '4br',
          'accepted',
          'access',
          'actual',
          'air',
          'allowed',
          'appliances',
          'approval',
          'approved',
```

Term Analysis

c. WordCloud



d. Artificial Analysis

Based on the wordcloud and term frequency analysis, we can classify the frequently occurring words according to their attributes, and then extract the data features according to these classifications.

```
In [32]: data
Out[32]:
```

...	feature_allowed	feature_nofee	feature_lowfee	feature_security	feature_laundry	feature_health
...	1	0	0	0	1	0
...	0	1	0	1	1	0
...	0	0	0	1	1	0

Acknowledgement:

Ideas of "Edge Detection" and table 1 comes from 3 Beginner-Friendly Techniques to Extract Features from Image Data using Python; and main part of codes in "Find Primary Color" is based on this web: OpenCV and Python K-Means Color Clustering

Team Members: Jinze Wu Yizhou Chen Luoxi Meng

Code Link: https://github.com/Wukkkinz-0725/CMPT459_Project_RentalListingInquiries