BUAN 6340.501 – F19: Programming for Data Science Project

AirBnb Rental Listings for San Francisco, United States

GROUP 11

Yuan Deng

Madhumadhi Sivarama Subramonian

Gautam Ramasamy

Mukil Raj Kannan

INDEX

PROBLEM STATEMENT

Motivation:

Airbnb is a privately-owned accommodation rental website which allows house owners to rent out their properties to guests looking for a place to stay. Airbnb enables a marketplace where people can rent lodging in residential properties. The main idea is a house-sharing service that offers an opportunity for travellers to get cheaper accommodation wherever they travel to. As of 15 September 2019, hosts have earned more than $80bn sharing their homes and spaces on Airbnb. The offers range from shared rooms to full condos/houses and can offer unique and differentiated options as compared to normal traditional hotels. Airbnb has 31 offices across the world and have hosted 400 million guests since its launch. Opportunity for homeowners to make more money from their homes by sharing with travellers. Airbnb offers complete independence to its hosts to decide on the price for their properties, however with only minimal attributes to leverage upon and compare similar previous listings in their neighbourhood, the hosts find it difficult to come up with a competitive price. With the number of hosts using Airbnb increasing, coming up with the right price to remain competitive in a host's neighbourhood is important. On the other side, guests can also find places to stay at a reasonable price and understand the seasonal demands. Airbnb has also seen a potential growth with the number of rentals/listings in its website subsequently increasing each year.

Through our analysis, we want to address the following:

- What time of the year are Airbnb's most popular in San Francisco? Are specific holiday seasons more popular?
- What are the most important characteristics of a listing, and how do they influence price?
- Understand the rental landscape in San Francisco through various static and interactive visualisations.
- How fast is Airbnb growing in San Francisco?
- The exploration results could refer useful knowledge for hosts who search for comparable price and make their listing stand out.
- Predict the Airbnb Listing Price in San Francisco
- We wanted to do a project that addresses a common issue that could either benefit an organization or the general public. The challenge that Airbnb hosts face is determining the optimal nightly rent price.
- Since Airbnb is a marketplace, the amount a host can charge on a nightly basis is closely linked to the dynamics of the marketplace.
- Moreover, we wanted to pick a dataset that gives us some scope to perform various analysis. Therefore, we chose the topic AirBnb Rental Listings for San Francisco, United States.

DATA EXPLORATION

Data set:

- We got the San Francisco listings and neighbourhoods data set from Inside Airbnb as .csv and .geojson files
- Source link: http://insideairbnb.com/get-the-data.html

The dataset comprised of three main tables:

- **listings** - Detailed listings data showing 106 attributes for each of the listings. Some of the attributes which may be used in our analysis are price(continuous), accommodates (the number of guests the rental can accommodate), bedrooms, bathrooms,listing_type (categorical), is_superhost (categorical), neighbourhood(categorical), ratings (continuous) among others.
- **reviews** - Detailed reviews given by the guests with 6 attributes. Key attributes include date(datetime), listing_id(discrete), reviewer_id and comment (textual).
- **calendar** - Provides details about booking for the next year by listing. Four attributes in total including listing_id (discrete), date(datetime), available (categorical) and price (continuous).
- Total number of attributes: 106
- Total number of hosts: 8111
- No of files: 2
    - listings.csv
    - neighbourhoods.geojson

listings.csv: which is the main data table for this project, containing distinct 8111 records with 106 attributes.

```
listings.head(5)
```

	id	listing_url	scrape_id	last_scraped	name	summary	space	description	experiences_offered	neighborhood_c
0	958	https://www.airbnb.com/rooms/958	20191014170858	2019-10-14	Bright, Modern Garden Unit - 1BR/1B	New update: the house next door is under const...	Newly remodeled, modern, and bright garden uni...	New update: the house next door is under const...	none	*Quiet cul friendly neigh
1	3850	https://www.airbnb.com/rooms/3850	20191014170858	2019-10-14	Charming room for two	Your own private room plus access to a shared ...	This room can fit two people. Nobody else will...	Your own private room plus access to a shared ...	none	This is a q neighborh
2	5858	https://www.airbnb.com/rooms/5858	20191014170858	2019-10-14	Creative Sanctuary	NaN	We live in a large Victorian house on a quiet ...	We live in a large Victorian house on a quiet ...	none	I love neighborhood fe
3	7918	https://www.airbnb.com/rooms/7918	20191014170858	2019-10-14	A Friendly Room - UCSF/USF - San Francisco	Nice and good public transportation. 7 minute...	Settle down, S.F. resident, student, hospital,...	Nice and good public transportation. 7 minute...	none	Shopping restaurants, M
4	8142	https://www.airbnb.com/rooms/8142	20191014170858	2019-10-14	Friendly Room Apt. Style - UCSF/USF - San Franc...	Nice and good public transportation. 7 minute...	Settle down, S.F. resident, student, hospital,...	Nice and good public transportation. 7 minute...	none	

```
listings.duplicated().sum()
```

```
0
```

```
listings.shape
```

```
(8111, 106)
```

```
listings.dtypes
```

```
id                          int64
listing_url                object
scrape_id                   int64
last_scraped               object
name                       object
summary                    object
space                      object
description                object
experiences_offered        object
neighborhood_overview      object
notes                      object
transit                    object
access                     object
interaction                object
house_rules                object
thumbnail_url             float64
medium_url                float64
picture_url                object
xl_picture_url            float64
host_id                     int64
```

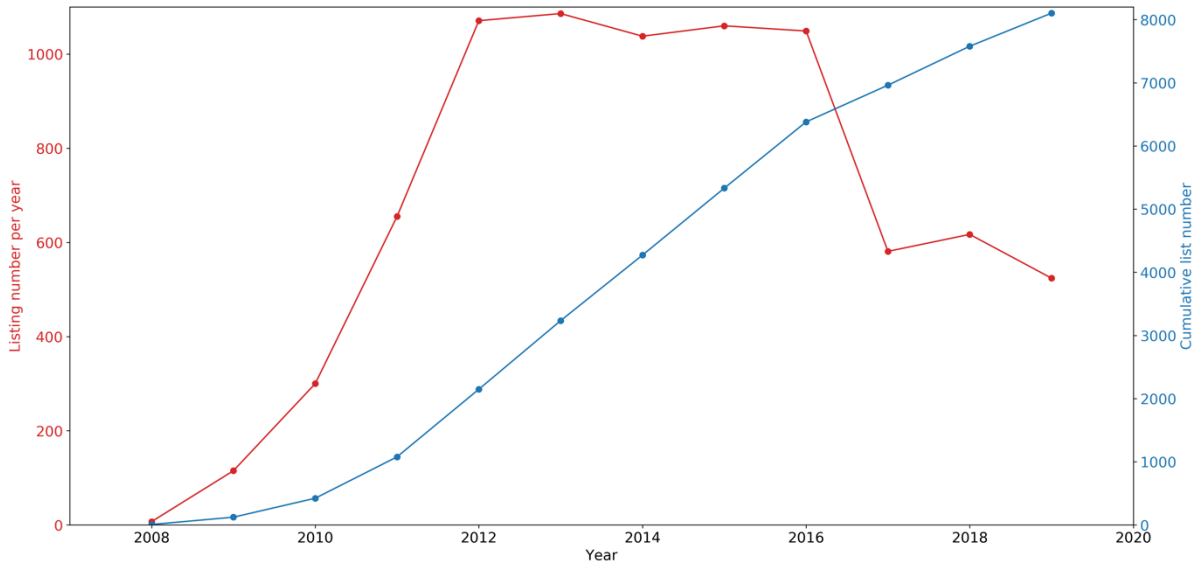
Neighbourboods.geojson: containing the map information of neighbourhoods.

RAW DATA VISUALIZATION

In this section, some attributes have been selected in the visualization to better understand the original data.

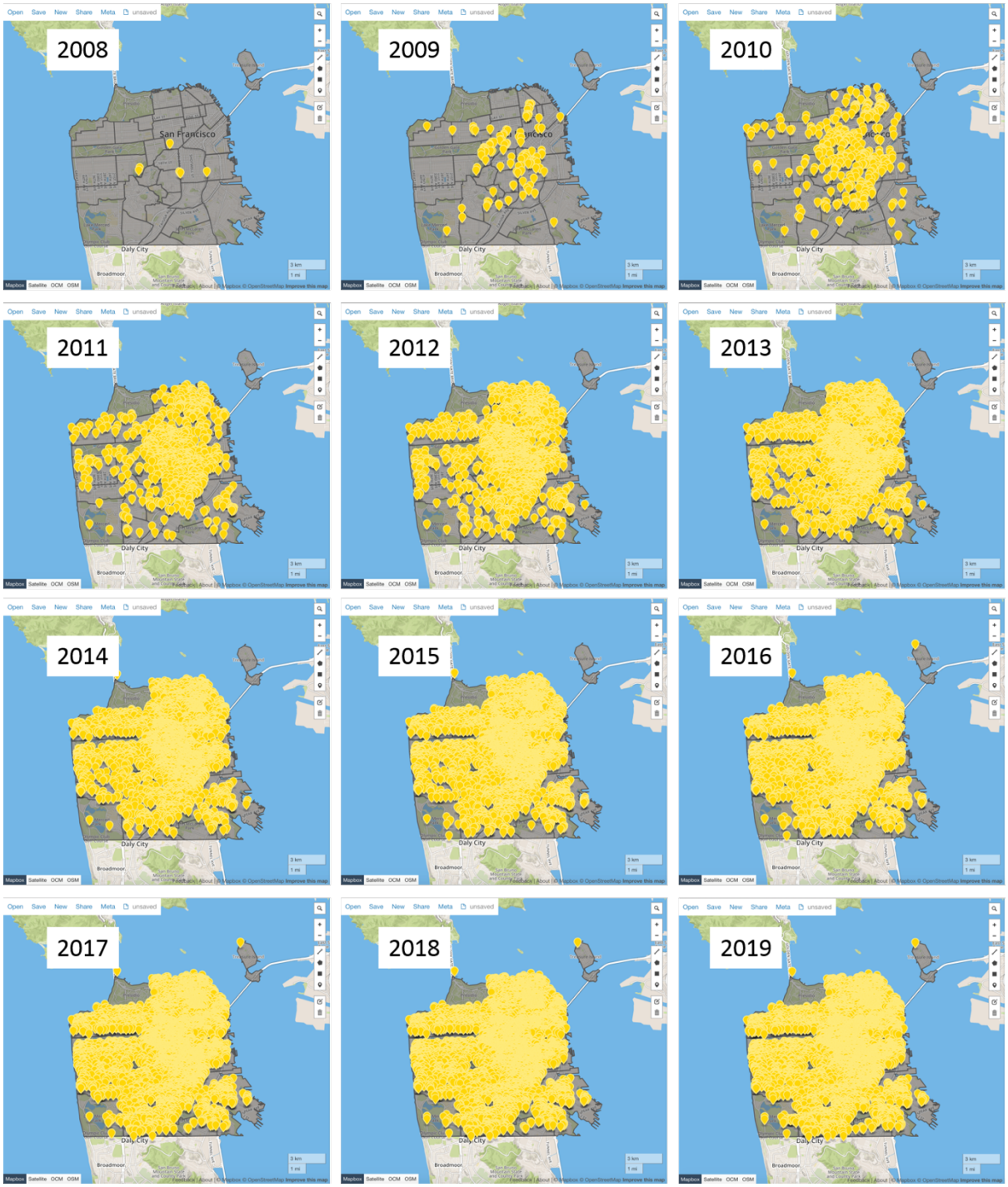
Number of listings over the year:

- Note that there is a sharp increase in the listing number of each year before 2012
- From 2012 to 2016, there are over 1000 new listings in each year
- In recent years, the number of listings in each year has drop below 600
- There has been a gradual increase in the number of listings since 2008 and the total number of listings is beyond 8000 in 2019



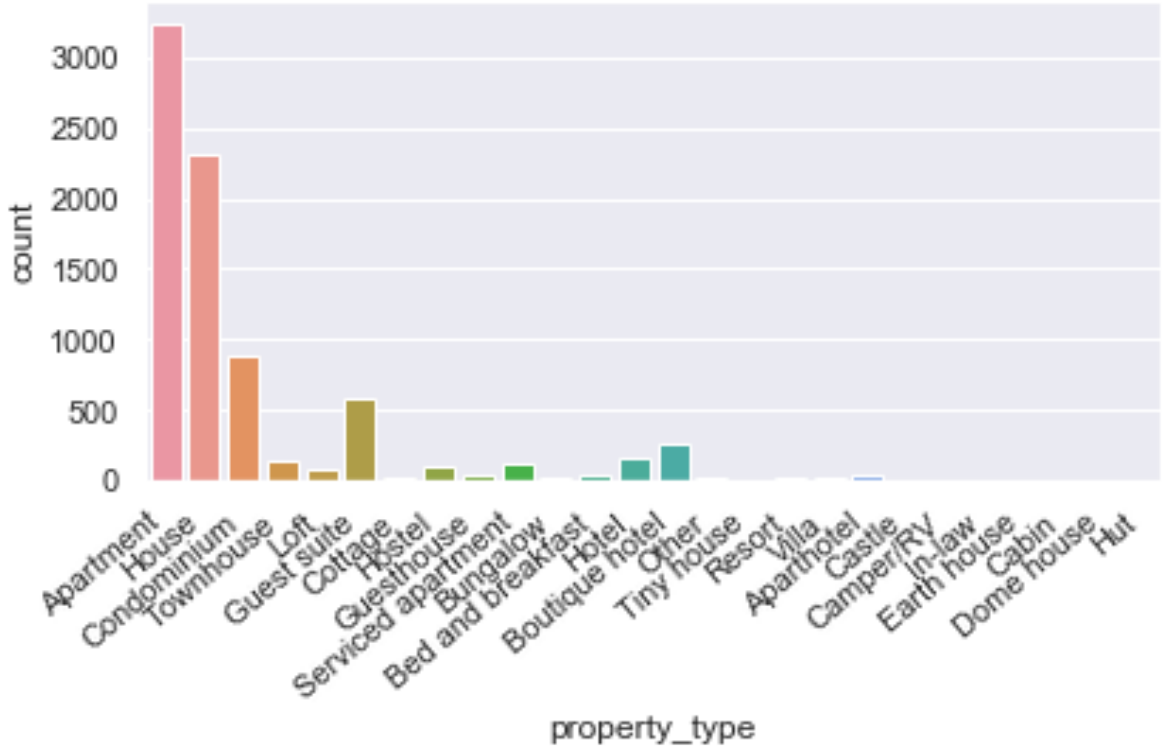
Distribution of listings over the year:

- As can be seen from the distribution map, Airbnb is very popular in San Francisco, and the market has now become a mature market.



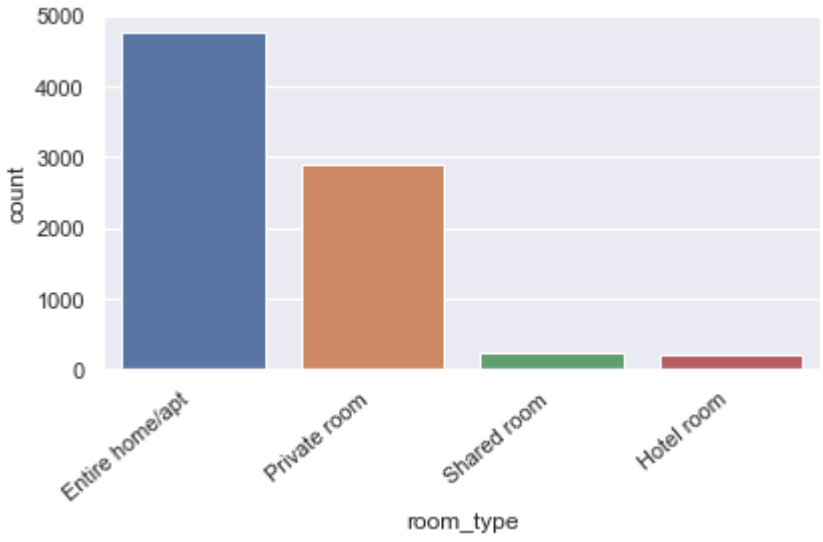
Distribution of property type in all listings:

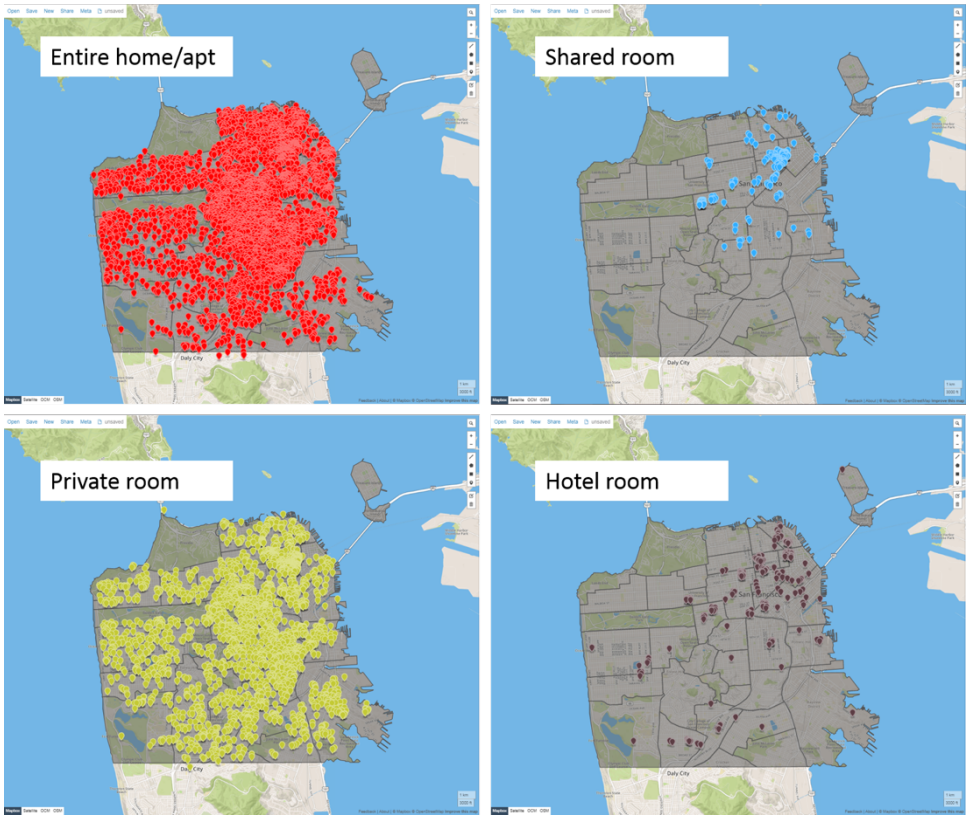
- From all records, majority listings are coming from apartment and house



Distribution of room type in all listings:

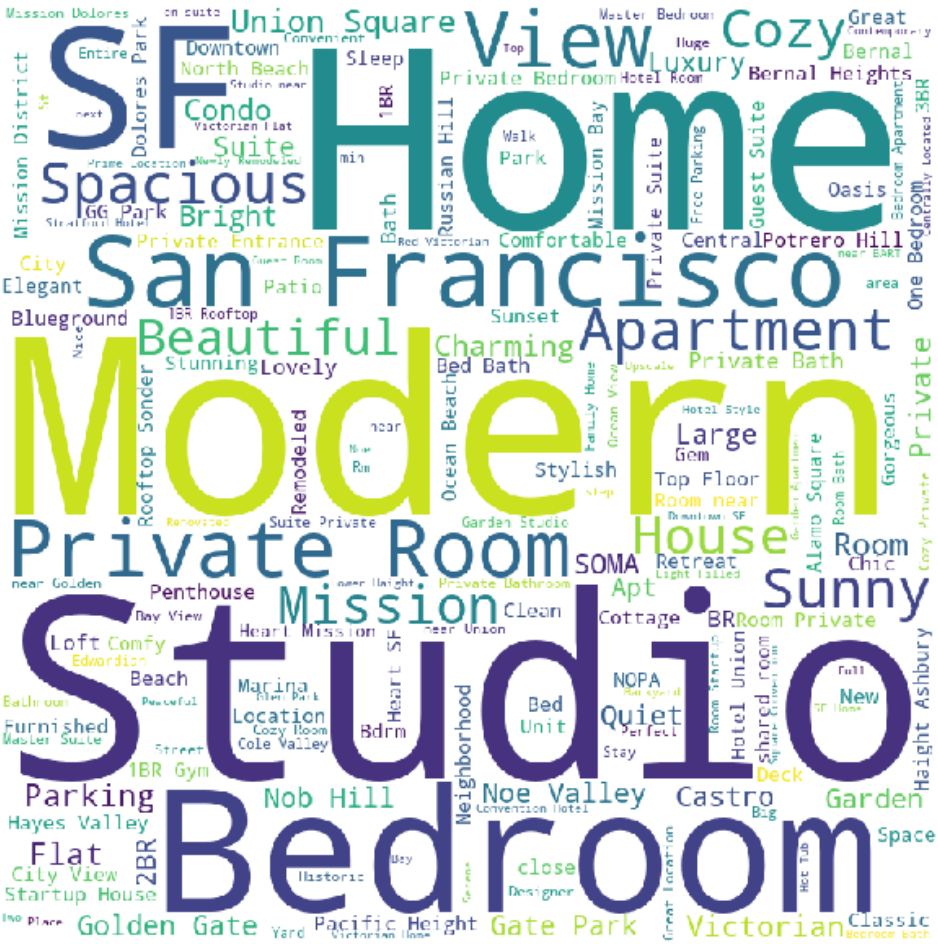
- Only four types of room are available from the listings.csv
- Apparently, shared room and hotel room are not popular comparing with others





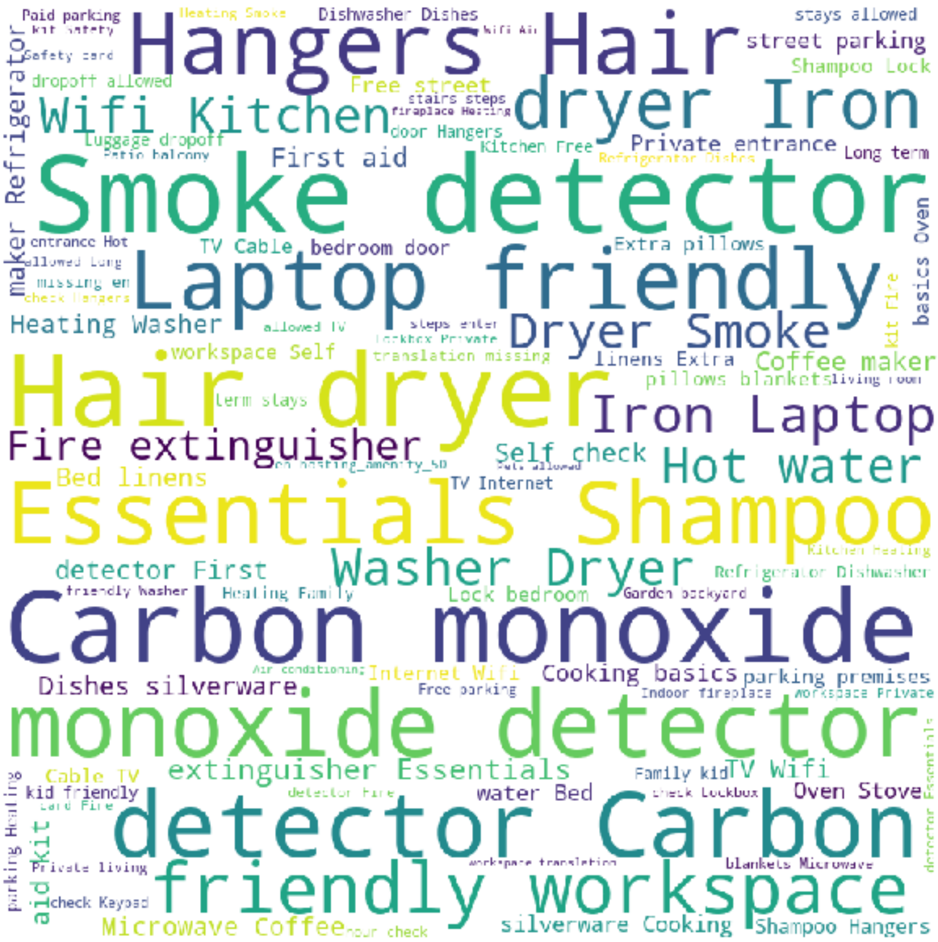
Word cloud of title, what most attract customers:

- "Modern", "Studio", and "Home" are the three most popular words in the listing tile



Word cloud of amenities, what people most care about:

- Apparently, “carbon monoxide”, “monoxide detector”, “Smoke detector” are very popular which means people care about the healthy first


Hangers Hair
dryer Iron
Wifi Kitchen
Smoke detector
Laptop friendly
Dryer Smoke
Hair dryer
Iron Laptop
Fire extinguisher
Hot water
Essentials Shampoo
Washer Dryer
Carbon monoxide
monoxide detector
detector Carbon
friendly workspace
Paid parking
Dishwasher Dishes
stays allowed
street parking
Shampoo Lock
Refrigerator
Free street
First aid
Private entrance
Long term
TV Cable
bedroom door
Extra pillows
Heating Washer
workspace Self
Coffee maker
pillows blankets
Self check
Bed linens
detector First
Lock bedroom
Dishes silverware
Cooking basics
parking premises
Internet Wifi
extinguisher Essentials
TV Wifi
kid friendly
water Bed
Oven Stove
Microwave Coffee
silverware Cooking
Shampoo Hangers

DATA CLEANING

It involved preparing the data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. Certain raw data was not necessary or helpful for analyzing data since it would have hindered the process or provided inaccurate results. Data cleaning attributed to the major part of effort needed for this project until now. The below mentioned steps were involved for cleaning the dataset to extract meaningful insights.

➢ **Drop empty and single value attributes:**

- The original dataset contains 8111 records and 106 attributes
- After dropping, the number of attributes reduced to 92

```
# drop empty and single value columns
listings.drop(listings.columns[listings.nunique() == 0], axis=1, inplace=True)
listings.drop(listings.columns[listings.nunique() == 1], axis=1, inplace=True)
```

➢ **Drop descriptive attributes:**

- Some attributes contain descriptive information, which is not useful in this project.
- E.g., “listing_url”: the URL of the current listing. “summary”: the summary section in the listing, it is all text description
- After dropping, the number of attributes reduced to 73

```
# delete descriptive columns
listings.drop(['listing_url', 'name', 'summary', 'space', 'description', 'neighborhood_overview', 'notes', 'transit',
               'access', 'interaction', 'house_rules', 'picture_url', 'host_url', 'host_about', 'host_thumbnail_url',
              'host_picture_url', 'host_verifications', 'amenities', 'license'], axis=1, inplace=True)
```

```
listings.shape
```

```
(8111, 73)
```

➢ **Drop attributes with a lot of nulls:**

```
listings.isnull().sum()
```

```
square_feet          7987
price                   0
weekly_price         7042
monthly_price        7067
security_deposit     1692
cleaning_fee          924
```

```
# drop columns with a lot of nulls
listings.drop(['square_feet', 'weekly_price', 'monthly_price'], axis=1, inplace=True)
```

➢ **Drop incorrect attributes:**

- By comparing with information from *neighbourhoords.csv*, there attributes are incorrect: 'host_neighbourhood', 'neighbourhood'

```
listings[['host_neighbourhood', 'neighbourhood', 'neighbourhood_cleansed']]
```

	host_neighbourhood	neighbourhood	neighbourhood_cleansed
0	Duboce Triangle	Duboce Triangle	Western Addition
1	Inner Sunset	Inner Sunset	Inner Sunset
2	Bernal Heights	Bernal Heights	Bernal Heights
3	Cole Valley	Cole Valley	Haight Ashbury

```
# drop incorrect information
listings.drop(['host_neighbourhood', 'neighbourhood'], axis=1, inplace=True)
```

- Compared with other columns 'calculated_host_listings_count_*', these columns are incorrect: 'host_listings_count', 'host_total_listings_count'

```
# drop incorrect information
listings.drop(['host_listings_count', 'host_total_listings_count'], axis=1, inplace=True)
```

➢ **Drop redundant attributes:**

- Some not useful: id, host_id, host_name, …
- Some not necessary: state, city, host_has_profile_pirc, …
- Some unclear contents: availability_30, availability_60, …
- Some bias data: bed_type, street…
- The number of attributes reduced to 32

```
# drop redundant information
listings.drop(['id', 'host_id', 'host_name', 'host_since', 'host_location', 'host_has_profile_pic', 'street', 'city',
               'state', 'zipcode', 'market', 'smart_location', 'latitude', 'longitude', 'is_location_exact', 'bed_type',
               'minimum_minimum_nights', 'maximum_minimum_nights', 'minimum_maximum_nights', 'maximum_maximum_nights',
               'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'calendar_updated', 'availability_30', 'availability_60',
               'availability_90', 'availability_365', 'number_of_reviews_ltm', 'first_review', 'last_review',
               'requires_license', 'instant_bookable', 'require_guest_profile_picture', 'require_guest_phone_verification',
              ], axis=1, inplace=True)
```

➢ **Fill null cells with average values:**

- Based on the meaning of attributes, some records with null cells are populated with average values in that attribute

```
listings.fillna({'review_scores_rating':listings['review_scores_rating'].mean()}, inplace=True)
listings.fillna({'review_scores_accuracy':listings['review_scores_accuracy'].mean()}, inplace=True)
listings.fillna({'review_scores_cleanliness':listings['review_scores_cleanliness'].mean()}, inplace=True)
listings.fillna({'review_scores_checkin':listings['review_scores_checkin'].mean()}, inplace=True)
listings.fillna({'review_scores_communication':listings['review_scores_communication'].mean()}, inplace=True)
listings.fillna({'review_scores_location':listings['review_scores_location'].mean()}, inplace=True)
listings.fillna({'review_scores_value':listings['review_scores_value'].mean()}, inplace=True)
```

➢ **Special values are used in certain null cells:**

- Based on the meaning of attributes, "reviews_per_month" should have 0 instead of null as the default value
- Default value should be zero for security_deposit and cleaning_fee

```
listings.fillna({'reviews_per_month':0}, inplace=True)
```

```
# default should be zero
listings.fillna({'security_deposit':0}, inplace=True)
listings.fillna({'cleaning_fee':0}, inplace=True)
```

➢ **Change the data format for some attributes:**

- The price related attributes should have the float format, not string
- Trailing character % should be removed from percentage values

```
# to remove the "$" charactor
listings['price'] = [float(Decimal(sub(r'[^\d.]', '', x))) if type(x) == str else x for x in listings['price']]
listings['extra_people'] = [float(Decimal(sub(r'[^\d.]', '', x))) if type(x) == str else x for x in listings['extra_people']]
listings['security_deposit'] = [float(Decimal(sub(r'[^\d.]', '', x))) if type(x) == str else x for x in listings['security_depos
listings['cleaning_fee'] = [float(Decimal(sub(r'[^\d.]', '', x))) if type(x) == str else x for x in listings['cleaning_fee']]
```

```
listings['host_response_rate'] = [float(x.strip('%')) if type(x) == str else x for x in listings['host_response_rate']]
```

➢ **Drop records with outliers:**

- There should be at least one bed in the listing
- Listings with no bedroom should be ignored from the data set
- The normal range of price should be (0, 500)

```
listings = listings[listings['beds']>0]
```

```
listings = listings[listings['bedrooms']>0]
```

```
listings = listings[listings['price']>0]
listings = listings[listings['price']<500]
```

➢ **Drop remaining null cells to get the final data set:**

- Final dataset: 5906 records, 32 attributes

```
listings.isnull().sum()
```

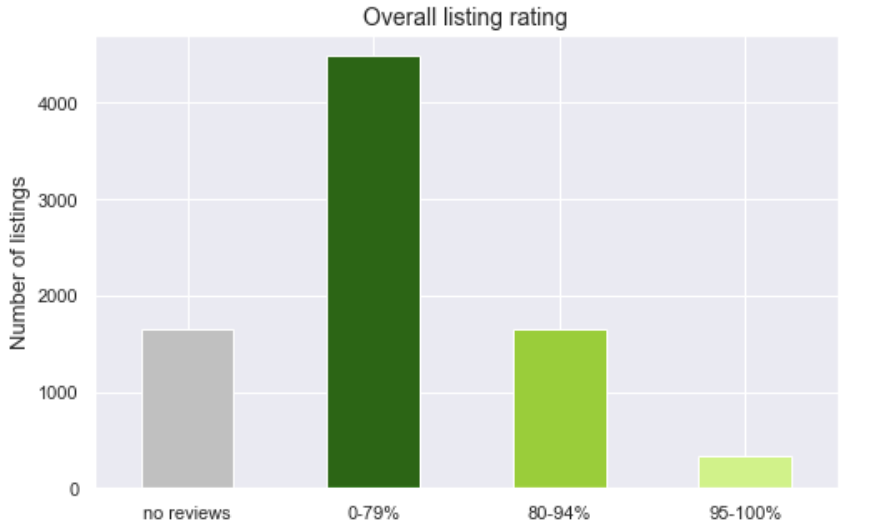
```
host_response_time                              0
host_response_rate                              0
host_is_superhost                               0
host_identity_verified                          0
neighbourhood_cleansed                          0
property_type                                   0
room_type                                       0
accommodates                                    0
bathrooms                                       0
bedrooms                                        0
beds                                            0
price                                           0
security_deposit                                0
cleaning_fee                                    0
guests_included                                 0
extra_people                                    0
minimum_nights                                  0
maximum_nights                                  0
number_of_reviews                               0
review_scores_rating                            0
review_scores_accuracy                          0
review_scores_cleanliness                       0
review_scores_checkin                           0
review_scores_communication                     0
review_scores_location                          0
review_scores_value                             0
cancellation_policy                             0
calculated_host_listings_count                  0
calculated_host_listings_count_entire_homes     0
calculated_host_listings_count_private_rooms    0
calculated_host_listings_count_shared_rooms     0
reviews_per_month                               0
dtype: int64
```

ANALYSIS

After cleaning the dataset; we then analyzed the data in Python using the matplotlib and seaborn libraries to plot and get useful insights for the queries which we initially framed as our problem statement. The below images/visualization answer most of the queries which could be utilized from this data set and then used further for model creation.

- **How long have hosts been listing properties on Airbnb in San Francisco?**
  - The trend lines plotted here depicts the actual exact trend of Airbnb in San Francisco's location market.
  - The peak values of hosts joining Airbnb was recorded in the years 2015-2016 and the trend continues to stay the same after that.
  - Also, the trend lines intimate that the number of listings getting their first reviews are increasing every year which indirectly depicts that number of users using the Airbnb app for their vacation stay are increasing year by year.

- **What is the distribution of reviews like?**
  - To get the overall distribution of reviews, we grouped the users' review_scores_ratings into 4 categories.
  - For the rows which had no reviews, we imputed them with 'no reviews' value and grouped it together to get the needed information.



- **What is the overall distribution of prices?**
  - From the cleaned dataset we had the price feature converted to integer values for interpretation and used it in our analysis to depict the overall distribution of nightly prices in San Francisco.

```
# Overall distribution of Prices.
import seaborn as sns
sns.set()
plt.figure(figsize=(20,4))
listings.price.hist(bins=100, range=(0,1000))
plt.margins(x=0)
plt.title("Advertised nightly prices in San Francisco up to $1000", fontsize=16)
plt.xlabel("Price ($)")
plt.ylabel("Number of listings")
plt.show()
```

DATA TRANSFORMATION

After data cleansing, the final dataset had 5906 records and 32 attributes. To determine the data transformation technique for every attribute, we took a closer look onto the data distribution of each attribute.

Host response time:

- The values of this attribute had four unique values which were 'within a day', 'within an hour', 'within a few hours', 'a few days or more'.

```
listings['host_response_time'].unique()
```

```
array(['within a day', 'within an hour', 'within a few hours',
       'a few days or more'], dtype=object)
```

- We used **one hot encoding** technique to create dummy values for each unique value to transform the data.

```
response_time = pd.get_dummies(listings["host_response_time"],columns = "host_response_time",
                               prefix = "host_response_time")
listings = pd.concat([listings, response_time], axis=1)
listings.drop(['host_response_time'],axis=1,inplace= True)
```

Host is superhost:

- The values of this attribute were either 't' or 'f'.

```
listings["host_is_superhost"].unique()
```

```
array(['t', 'f'], dtype=object)
```

- We used **mapping** technique to transform 't' to 0 and 'f' to 1 value.

```
listings["host_is_superhost"] = listings["host_is_superhost"].map({'f' : 0, 't' : 1}).astype(int)
```

Neighbourhood Cleansed:

- There are 36 unique values in this attribute.

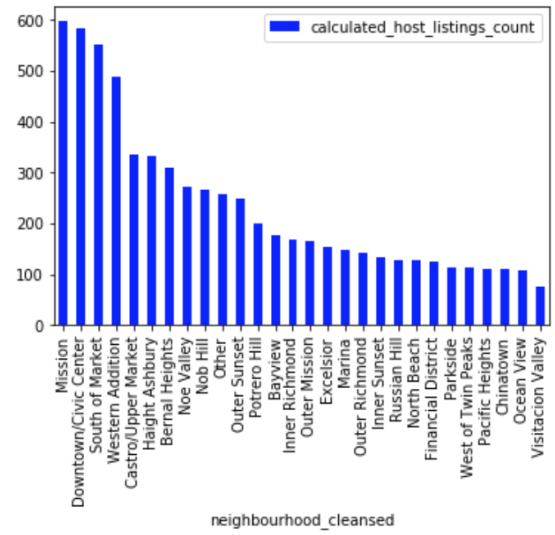
```
listings["neighbourhood_cleansed"].nunique()
```

```
36
```

- We reduced the number of unique values in this attribute by replacing the least count in unique values into 'other'. The unique values were reduced to 28 values.

```
series = pd.value_counts(listings.neighbourhood_cleansed)
mask = (series/series.sum() * 100).lt(1)
listings['neighbourhood_cleansed'] = np.where(listings['neighbourhood_cleansed'].isin(series[mask].index),
                                              'Other',listings['neighbourhood_cleansed'])

pvt = listings.pivot_table(index= "neighbourhood_cleansed", values="calculated_host_listings_count", aggfunc='count')
pvt = pvt.sort_values("calculated_host_listings_count",ascending=False)
pvt
```



.

neighbourhood_cleansed	calculated_host_listings_count
Mission	598
Downtown/Civic Center	583
South of Market	552
Western Addition	488
Castro/Upper Market	336
Haight Ashbury	334
Bernal Heights	311
Noe Valley	271
Nob Hill	266
Other	257
Outer Sunset	248
Potrero Hill	200
Bayview	177
Inner Richmond	167
Outer Mission	164
Excelsior	153
Marina	149
Outer Richmond	143
Inner Sunset	133
Russian Hill	128
North Beach	127
Financial District	126
Parkside	114
West of Twin Peaks	113
Pacific Heights	110
Chinatown	110
Ocean View	108
Visitacion Valley	77

```
print(listings['neighbourhood_cleansed'].nunique())
```

```
28
```

- We used **one hot encoding** technique to create dummy values for each unique value to transform the data.

```
neighborhood = pd.get_dummies(listings["neighbourhood_cleansed"],columns = "neighbourhood_cleansed",
                              prefix = "neighbourhood")
listings = pd.concat([listings, neighborhood], axis=1)
listings.drop(["neighbourhood_cleansed"],axis=1,inplace= True)
listings
```

Is location exact:

- The values of this attribute were either ‘t’ or ‘f’.

```
listings["is_location_exact"].unique()
```

```
array(['t', 'f'], dtype=object)
```

- We used **mapping technique** to transform ‘t’ to 0 and ‘f’ to 1 value.

```
listings["is_location_exact"] = listings["is_location_exact"].map({'f' : 0, 't' : 1}).astype(int)
```

Property Type:

- There are 25 unique values in this attribute.

```
listings["property_type"].unique()
```

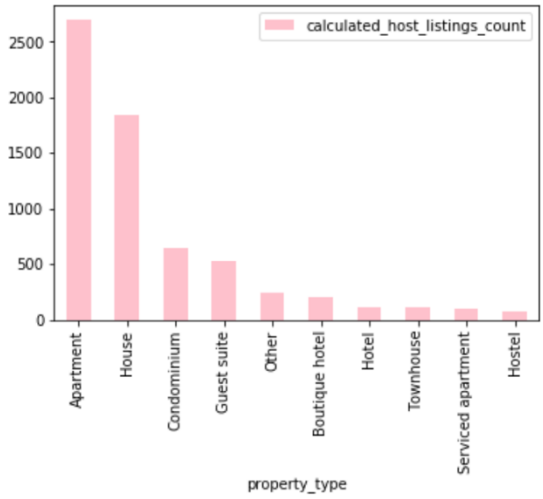
```
array(['Apartment', 'House', 'Condominium', 'Townhouse', 'Guest suite',
       'Cottage', 'Hostel', 'Guesthouse', 'Serviced apartment', 'Loft',
       'Bungalow', 'Bed and breakfast', 'Hotel', 'Boutique hotel',
       'Other', 'Tiny house', 'Villa', 'Resort', 'Aparthotel', 'Castle',
       'Camper/RV', 'In-law', 'Cabin', 'Earth house', 'Dome house'],
      dtype=object)
```

```
listings["property_type"].nunique()
```

```
25
```

- We reduced the number of unique values in this attribute by replacing the least count in unique values into 'other'. The unique values were reduced to 10 values.

property_type	calculated_host_listings_count
Apartment	2696
House	1837
Condominium	640
Guest suite	526
Other	241
Boutique hotel	203
Hotel	112
Townhouse	109
Serviced apartment	103
Hostel	76




```
listings["property_type"].nunique()
```

```
10
```

- We used **one hot encoding** technique to create dummy values for each unique value to transform the data.

```
property_type = pd.get_dummies(listings["property_type"],columns = "property_type", prefix = "property_type")
listings = pd.concat([listings, property_type], axis=1)
listings.drop(['property_type'],axis=1,inplace= True)
listings
```

Room Type:

- The values of this attribute had four unique values which were 'Entire home/apt', 'Private room', 'Shared room', 'Hotel room'.

```
listings["room_type"].unique()
```

```
array(['Entire home/apt', 'Private room', 'Shared room', 'Hotel room'],
      dtype=object)
```

- We used **one hot encoding** technique to create dummy values for each unique value to transform the data.

```
room_type = pd.get_dummies(listings["room_type"],columns = "room_type", prefix = "room_type")
listings = pd.concat([listings, room_type], axis=1)
listings.drop(['room_type'],axis=1,inplace= True)
listings
```

Cancellation Policy:

- The values of this attribute had six unique values which were ‘moderate', 'strict_14_with_grace_period', 'flexible', 'super_strict_30', 'strict', 'super_strict_60'.

```
listings["cancellation_policy"].unique()
```

```
array(['moderate', 'strict_14_with_grace_period', 'flexible',
       'super_strict_30', 'strict', 'super_strict_60'], dtype=object)
```

- We initially changed the two unique values - ‘super_strict_30’ and ‘super_strict_60’ into ‘super_strict’. This reduced the unique values to 5.

cancellation_policy	calculated_host_listings_count
strict_with_grace_period	2862
moderate	2170
flexible	1368
strict	86
super_strict	57

- We used **one hot encoding** technique to create dummy values for each unique value to transform the data.

```
cancellation_policy = pd.get_dummies(listings["cancellation_policy"],columns = "cancellation_policy",
                                     prefix = "cancellation")
listings = pd.concat([listings, cancellation_policy], axis=1)
listings.drop(['cancellation_policy'],axis=1,inplace= True)
listings
```

Host Response rate:

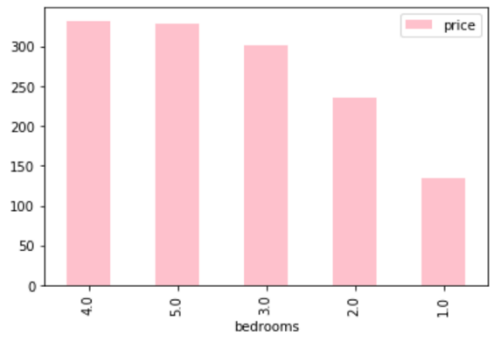
- This attribute is represented in percentage value. We initially changed the data type of the attribute to category and then segregated the values by mapping into six sets: ‘90-100’, ‘80-90’, ‘70-80’, ‘60-70’, ‘<60’.
- We used **one hot encoding** technique to create dummy variables for each unique value to transform the data.

```
host_response_rate = pd.get_dummies(listings["host_response_rate"],columns = "host_response_rate",
                                    prefix = "host_response_rate")
listings = pd.concat([listings, host_response_rate], axis=1)
listings.drop(['host_response_rate'],axis=1,inplace= True)
listings
```

Bedrooms:

- After analysing the data distribution in this attribute, we changed the datatype to category and removed the outliers and anomalies.
- Hence we used only a range of data in this attribute which constitutes to 5 unique values.

```
listings = listings[(listings["bedrooms"] > 0) & (listings["bedrooms"] <=5)]
```



bedrooms	price
4.0	331.761364
5.0	327.750000
3.0	300.757869
2.0	234.850838
1.0	133.771469

- We used **one hot encoding** technique to create dummy variables for each unique value to transform the data.

```
bedrooms = pd.get_dummies(listings["bedrooms"],columns = "bedrooms", prefix = "bedrooms")
listings = pd.concat([listings, bedrooms], axis=1)
listings.drop(['bedrooms'],axis=1,inplace= True)
listings
```

Bathrooms:

- After analysing the data distribution in this attribute, we changed the datatype to category and removed the outliers and anomalies.
- Hence we used only a range of data in this attribute which constitutes to 11 unique values.

```
listings = listings[listings["bathrooms"] <= 5]
```

bathrooms	price
0.0	69.194444
0.5	79.266667
1.0	156.841003
1.5	182.939597
2.0	233.769036
2.5	214.014493
3.0	193.956522
3.5	235.500000
4.0	105.543478
4.5	342.250000
5.0	79.081633

- We used **one hot encoding** technique to create dummy variables for each unique value to transform the data.

```
bathrooms = pd.get_dummies(listings["bathrooms"],columns = "bathrooms", prefix = "bathrooms")
listings = pd.concat([listings, bathrooms], axis=1)
listings.drop(['bathrooms'],axis=1,inplace= True)
listings
```

Attributes transformed using Mapping technique:

- Host is superhost
- Is location exact

Attributes transformed using One-hot encoding technique:

- Host response time
- Neighbourhood cleansed
- Property type
- Room type
- Cancellation policy
- Host response rate
- Bedrooms
- Bathrooms

Final Transformed dataset: 5775 records and 96 attributes

```
Int64Index: 5775 entries, 0 to 8110
Data columns (total 96 columns):
host_is_superhost                              5775 non-null int64
is_location_exact                              5775 non-null int64
accommodates                                   5775 non-null int64
beds                                           5775 non-null int64
price                                          5775 non-null float64
security_deposit                               5775 non-null float64
cleaning_fee                                   5775 non-null float64
guests_included                                5775 non-null int64
extra_people                                   5775 non-null float64
minimum_nights                                 5775 non-null int64
maximum_nights                                 5775 non-null int64
number_of_reviews                              5775 non-null int64
review_scores_rating                           5775 non-null float64
review_scores_accuracy                         5775 non-null float64
review_scores_cleanliness                      5775 non-null float64
review_scores_checkin                          5775 non-null float64
review_scores_communication                    5775 non-null float64
review_scores_location                         5775 non-null float64
review_scores_value                            5775 non-null float64
calculated_host_listings_count                 5775 non-null int64
calculated_host_listings_count_entire_homes    5775 non-null int64
calculated_host_listings_count_private_rooms   5775 non-null int64
calculated_host_listings_count_shared_rooms    5775 non-null int64
reviews_per_month                              5775 non-null float64
host_response_time_a few days or more          5775 non-null uint8
host_response_time_within a day                5775 non-null uint8
host_response_time_within a few hours          5775 non-null uint8
host_response_time_within an hour              5775 non-null uint8
neighbourhood_Bayview                          5775 non-null uint8
neighbourhood_Bernal Heights                   5775 non-null uint8
neighbourhood_Castro/Upper Market              5775 non-null uint8
neighbourhood_Chinatown                        5775 non-null uint8
neighbourhood_Downtown/Civic Center            5775 non-null uint8
neighbourhood_Excelsior                        5775 non-null uint8
neighbourhood_Financial District               5775 non-null uint8
neighbourhood_Haight Ashbury                   5775 non-null uint8
neighbourhood_Inner Richmond                   5775 non-null uint8
neighbourhood_Inner Sunset                     5775 non-null uint8
neighbourhood_Marina                           5775 non-null uint8
neighbourhood_Mission                          5775 non-null uint8
neighbourhood_Nob Hill                         5775 non-null uint8
neighbourhood_Noe Valley                       5775 non-null uint8
neighbourhood_North Beach                      5775 non-null uint8
neighbourhood_Ocean View                       5775 non-null uint8
neighbourhood_Parkside                         5775 non-null uint8
neighbourhood_Potrero Hill                     5775 non-null uint8
neighbourhood_Russian Hill                     5775 non-null uint8
neighbourhood_South of Market                  5775 non-null uint8
neighbourhood_Visitacion Valley                5775 non-null uint8
neighbourhood_West of Twin Peaks               5775 non-null uint8
neighbourhood_Western Addition                 5775 non-null uint8
property_type_Apartment                        5775 non-null uint8
property_type_Boutique hotel                   5775 non-null uint8
property_type_Condominium                      5775 non-null uint8
property_type_Guest suite                      5775 non-null uint8
property_type_Hostel                           5775 non-null uint8
property_type_Hotel                            5775 non-null uint8
property_type_House                            5775 non-null uint8
property_type_Other                            5775 non-null uint8
property_type_Serviced apartment               5775 non-null uint8
property_type_Townhouse                        5775 non-null uint8
room_type_Entire home/apt                      5775 non-null uint8
room_type_Hotel room                           5775 non-null uint8
room_type_Private room                         5775 non-null uint8
room_type_Shared room                          5775 non-null uint8
cancellation_flexible                          5775 non-null uint8
cancellation_moderate                          5775 non-null uint8
cancellation_strict                            5775 non-null uint8
cancellation_strict_with_grace_period          5775 non-null uint8
cancellation_super_strict                      5775 non-null uint8
host_response_rate_60-70                       5775 non-null uint8
host_response_rate_70-80                       5775 non-null uint8
host_response_rate_80-90                       5775 non-null uint8
host_response_rate_90-100                      5775 non-null uint8
host_response_rate_<60                         5775 non-null uint8
bedrooms_1.0                                   5775 non-null uint8
bedrooms_2.0                                   5775 non-null uint8
bedrooms_3.0                                   5775 non-null uint8
bedrooms_4.0                                   5775 non-null uint8
bedrooms_5.0                                   5775 non-null uint8
bathrooms_0.0                                  5775 non-null uint8
bathrooms_0.5                                  5775 non-null uint8
bathrooms_1.0                                  5775 non-null uint8
bathrooms_1.5                                  5775 non-null uint8
bathrooms_2.0                                  5775 non-null uint8
bathrooms_2.5                                  5775 non-null uint8
bathrooms_3.0                                  5775 non-null uint8
bathrooms_3.5                                  5775 non-null uint8
bathrooms_4.0                                  5775 non-null uint8
bathrooms_4.5                                  5775 non-null uint8
bathrooms_5.0                                  5775 non-null uint8
dtypes: float64(12), int64(12), uint8(72)
```

MODELING

Feature Scaling:

Before running the ML models; it is always necessary to ensure that all our features in the dataset have been scaled or normalized in order to have the same magnitude. Feature scaling is a step of data pre-processing which is applied to independent variables or features of data. It basically helps to normalize the data within a particular range. In some cases; it also helps in speeding up the calculations in an algorithm.

FEATURE SCALING

```
listings.drop(columns = ["host_identity_verified"],inplace = True)

y = listings["price"]
X = listings.drop(columns = ["price"])
```

```
X_train_org,X_test_org,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.3)
```

```
X_train_org.shape
```

```
(4041, 94)
```

```
X_test_org.shape
```

```
(1733, 94)
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train_org)
X_test = sc.transform(X_test_org)
```

Linear Regression:

LINEAR REGRESSION

```
train_score = []
test_score = []
model_Scores = []
```

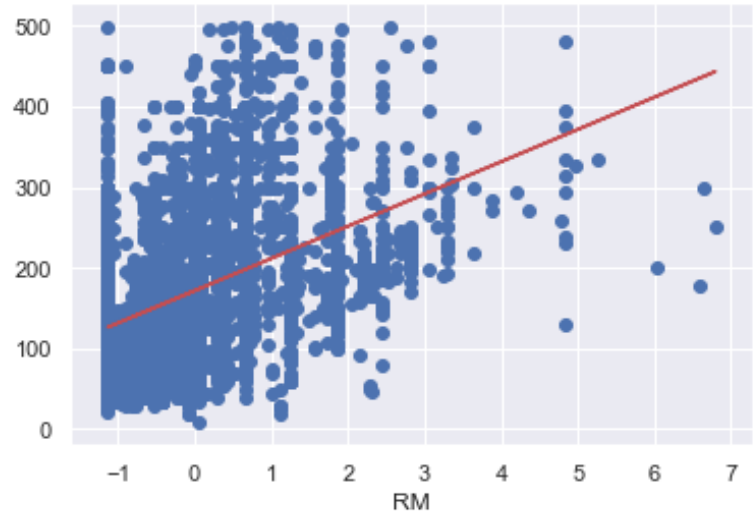
```
from sklearn.linear_model import LinearRegression

lreg = LinearRegression()
lreg.fit(X_train, y_train)
print('Train Score:',lreg.score(X_train, y_train))
train_score.append(lreg.score(X_train, y_train))
test_score.append(lreg.score(X_test, y_test))
train_score = lreg.score(X_train, y_train)
test_score = lreg.score(X_test,y_test)
print("Test Score {:.2f}".format(test_score))
## Appending the results
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Linear Regression',
                     'Parameters' : '-',
                     'Training_score': train_score,
                     'Test Score':test_score})
```

```
Train Score: 0.6277386146120525
Test Score 0.57
```

- The following key metrics/results were obtained after the execution of Linear Regression model.
    - Training Set Score: 0.63
    - Test Score: 0.57
    - Accuracy of the model: 58%

```
Text(0.5, 0, 'RM')
```



Polynomial Regression:

```
train_score_list = []
test_score_list = []

for n in range(2,4):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    lreg.fit(X_train_poly, y_train)
    scores_train = cross_val_score(lreg, X_train_poly, y_train, cv=3)
    scores_test = lreg.score(X_test_poly,y_test)
    test_score_list.append(scores_test)
    train_score_list.append(scores_train.mean())

print("Train Scores", train_score_list)
print("Test Scores", test_score_list)
```

```
Train Scores [-3.5488865482143527e+19, -9.918478435232078e+18]
Test Scores [-1.137344328040649e+20, -1.365666205564552e+18]
```

```
print("Train Scores", train_score_list[1])
print("Test Scores", test_score_list[1])
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Polynomial Regression',
                     'Parameters' : {'Degree':2},
                     'Training_score':train_score_list[1],
                     'Test Score':test_score_list[1]})
```

```
Train Scores -9.918478435232078e+18
Test Scores -1.365666205564552e+18
```

- The following key metrics/results were obtained after the execution of Polynomial Regression model.
    - Training Set Score: -9.918478435232078e+18
    - Test Score: -1.365666205564552e+18

KNN Regressor:

```
def printGridResult (model_def) :
    print("Best CV result: {}".format(model_def.best_score_))
    print("Best parameters: {}".format(model_def.best_params_))
    print("Training Score: {}".format(model_def.score(X_train, y_train)))
    print("Testing Score: {}".format(model_def.score(X_test, y_test)))

#KNN Regressor
knnRegressor = KNeighborsRegressor()
noofNeighbors = {'n_neighbors':[1, 5, 10, 15, 20]}
print("Parameter grid:\n{}".format(noofNeighbors))
knnGridSCV = GridSearchCV(knnRegressor, param_grid = noofNeighbors, cv=5, return_train_score=True)
knnGridSCV.fit(X_train, y_train)

n_neighbors = [1, 5, 10, 15, 20]

n_neighbors = [1, 5, 10, 15, 20]

train_knnRegressor = knnGridSCV.cv_results_["mean_train_score"]

test_knnRegressor = knnGridSCV.cv_results_["mean_test_score"]

printGridResult(knnGridSCV)

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'KNN Regressor',
                     'Parameters' : knnGridSCV.best_params_,
                     'Training_score': knnGridSCV.score(X_train, y_train),
                     'Test Score': knnGridSCV.score(X_test, y_test )})
```

```
Parameter grid:
{'n_neighbors': [1, 5, 10, 15, 20]}
Best CV result: 0.5284313086577018
Best parameters: {'n_neighbors': 10}
Training Score: 0.6393939518337186
Testing Score: 0.5103967496168658
```

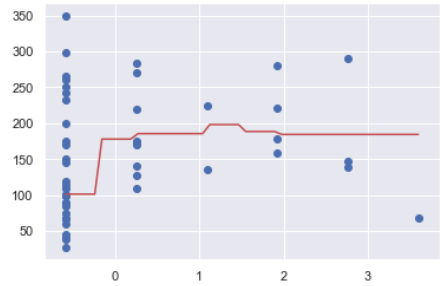
```
X_b = X_train[:50,6].reshape(-1,1)
y_b = y_train[:50]

knn_reg = KNeighborsRegressor(10)
knn_reg.fit(X_b, y_b)

X_new=np.linspace(X_b.min(), X_b.max(), 50).reshape(50, 1)
y_predict = knn_reg.predict(X_new)

plt.plot(X_new, y_predict, c = 'r')
plt.scatter(X_b, y_b)
```

```
<matplotlib.collections.PathCollection at 0x28091cd0be0>
```



- The following key metrics/results were obtained after the execution of KNNRegressor model.
    - Training Set Score: 0.64
    - Test Score: 0.51
    - Best Parameters: n_neighbors: 10

Ridge Regression:

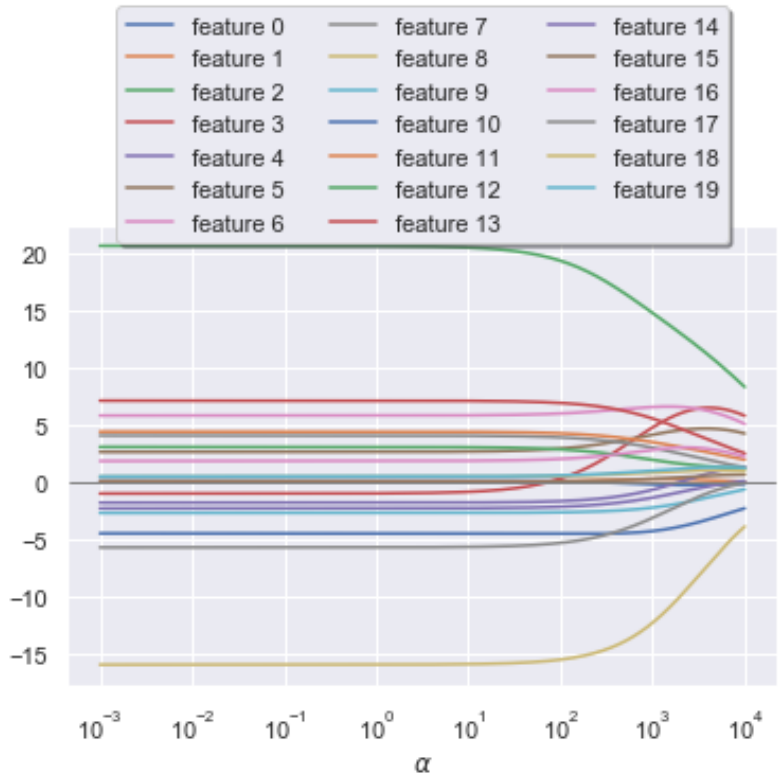
```
# Ridge Regression
ridgeParamGSV = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(ridgeParamGSV))

ridgeGridSV = GridSearchCV(estimator = Ridge(random_state = 0 ),param_grid = ridgeParamGSV,
                           cv=5, return_train_score=True, scoring='r2', n_jobs=1)
ridgeGridSV.fit(X_train, y_train)

ridge_train_scores_mean = ridgeGridSV.cv_results_["mean_train_score"]
ridge_test_scores_mean = ridgeGridSV.cv_results_["mean_test_score"]

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Ridge Regression',
                     'Parameters' : ridgeGridSV.best_params_,
                     'Training_score': ridgeGridSV.score(X_train, y_train),
                     'Test Score': ridgeGridSV.score(X_test, y_test )})
printGridResult(ridgeGridSV)
```

```
Parameter grid:
{'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
Best CV result: 0.6033043158737826
Best parameters: {'alpha': 100}
Training Score: 0.6276089028868538
Testing Score: 0.5750267353139064
```



- The following key metrics/results were obtained after the execution of Ridge Regression model.
    - Training Set Score: 0.63
    - Test Score: 0.58
    - Best Parameters: alpha: 100
    - Accuracy of the model: 56.92%

Lasso Regression:

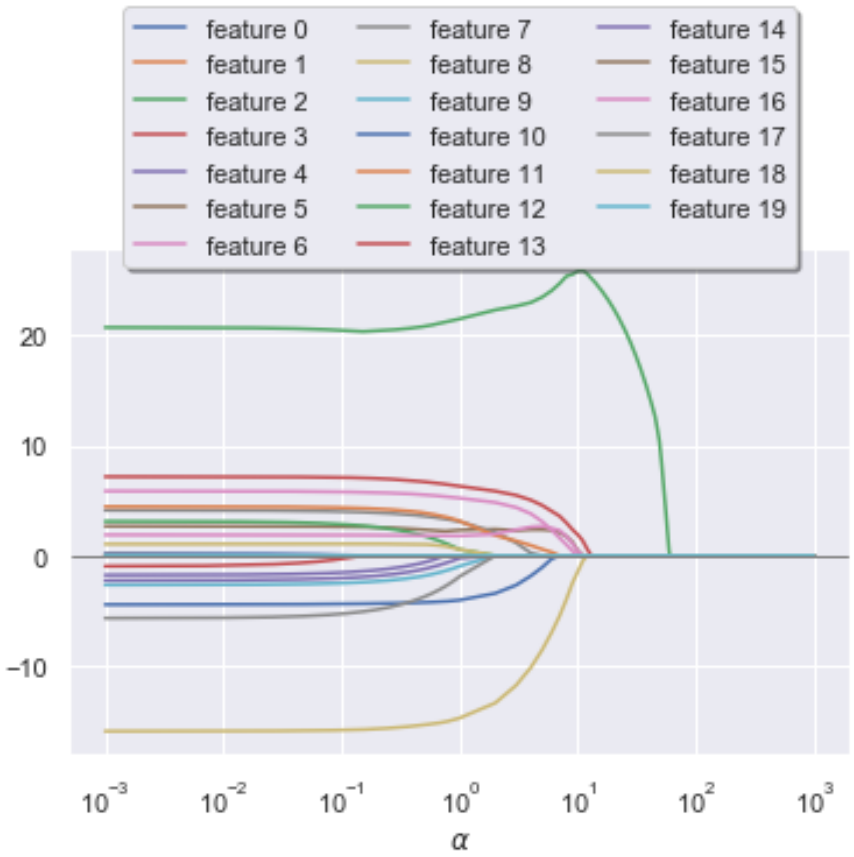
```
#Lasso Regression
lassoParamGSV = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(lassoParamGSV))

lassoGridSV = GridSearchCV(estimator = Lasso(random_state = 0 ),param_grid = lassoParamGSV,
                           cv=5, return_train_score=True, scoring='r2', n_jobs=1)
lassoGridSV.fit(X_train, y_train)

lasso_train_scores_mean = lassoGridSV.cv_results_["mean_train_score"]
lasso_test_scores_mean = lassoGridSV.cv_results_["mean_test_score"]

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Lasso Regression',
                     'Parameters' : lassoGridSV.best_params_,
                     'Training_score': lassoGridSV.score(X_train, y_train),
                     'Test Score': lassoGridSV.score(X_test, y_test )})
printGridResult(lassoGridSV)
```

```
Parameter grid:
{'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
Best CV result: 0.6026840709968957
Best parameters: {'alpha': 0.1}
Training Score: 0.6277062421084709
Testing Score: 0.5747415866899004
```



- The following key metrics/results were obtained after the execution of Lasso Regression model.
    - Training Set Score: 0.63
    - Test Score: 0.57
    - Best Parameters: alpha: 0.1
    - Accuracy of the model: 58.01%

SVM Regression:

SVM With Linear Kernel:

```
#SVM With Linear Kernal
from sklearn.svm import LinearSVR,SVR
linearSVM = SVR(kernel='linear')
param_Linear_SVM = {
    'gamma': [0.001, 0.01, 0.1, 1, 10],
    'C': [0.001, 0.01, 0.1, 1, 10],
}

LinearSVMGrid = GridSearchCV(estimator=linearSVM, param_grid=param_Linear_SVM, cv=3, n_jobs=-1)
LinearSVMGrid.fit(X_train, y_train)

print('Train score:',LinearSVMGrid.score(X_train, y_train))
print('Test score:',LinearSVMGrid.score(X_test, y_test))
print('Best parameters:',LinearSVMGrid.best_params_)

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Linear SVM',
                     'Parameters' : LinearSVMGrid.best_params_,
                     'Training_score': LinearSVMGrid.score(X_train, y_train),
                     'Test Score': LinearSVMGrid.score(X_test, y_test )})
```

```
Train score: 0.608152859484705
Test score: 0.5527286103744447
Best parameters: {'C': 1, 'gamma': 0.001}
```

- The following key metrics/results were obtained after the execution of SVM Regression Model with Linear Kernel.
    - Training Set Score: 0.61
    - Test Score: 0.55
    - Best Parameters: ‘C’:1; ‘gamma’:0.001

SVM With Radial Kernel:

```
#SVM With Radial Kernal
svmRadial = SVR(kernel = 'rbf')
param_Radial_SVM = {'C': [0.001, 0.01, 0.1, 1, 10, 100],'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_Radial_SVM))
svmRadialGridSV  = GridSearchCV(svmRadial, param_grid = param_Radial_SVM, cv=5, return_train_score=True)

svmRadialGridSV.fit(X_train, y_train)

svmRadialGridSV_results = pd.DataFrame(svmRadialGridSV.cv_results_)

printGridResult(svmRadialGridSV)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'SVM Radial',
                     'Parameters' : svmRadialGridSV.best_params_,
                     'Training_score': svmRadialGridSV.score(X_train, y_train),
                     'Test Score': svmRadialGridSV.score(X_test, y_test )})
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
Best CV result: 0.634296648179041
Best parameters: {'C': 100, 'gamma': 0.01}
Training Score: 0.7965109389825114
Testing Score: 0.6129452123567071
```

- The following key metrics/results were obtained after the execution of SVM Regression Model with Radial Kernel.
    - Training Set Score: 0.80
    - Test Score: 0.61
    - Best Parameters: ‘C’:100; ‘gamma’:0.01

SVM With Poly Kernel:

```
#Polynomial SVM Kernal
polySVM = SVR(kernel = 'poly', degree = 2)
paramPolySVM = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(paramPolySVM))
polySVMGridSV = GridSearchCV(polySVM, param_grid = paramPolySVM, cv=5, return_train_score= True)

polySVMGridSV.fit(X_train, y_train)

printGridResult(polySVMGridSV)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Polynomial SVM',
                     'Parameters' : polySVMGridSV.best_params_,
                     'Training_score': polySVMGridSV.score(X_train, y_train),
                     'Test Score': polySVMGridSV.score(X_test, y_test )})
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100]}
Best CV result: 0.547785744088905
Best parameters: {'C': 100}
Training Score: 0.7464654145254829
Testing Score: 0.540805998063183
```

- The following key metrics/results were obtained after the execution of SVM Regression Model with Poly Kernel.
    - Training Set Score: 0.75
    - Test Score: 0.54
    - Best Parameters: 'C':100

XGBRegressor:

```
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import time
from xgboost.sklearn import XGBRegressor

xgb_reg_start = time.time()

xgb_reg = xgb.XGBRegressor()
xgb_reg.fit(X_train, y_train)
training_preds_xgb_reg = xgb_reg.predict(X_train)
val_preds_xgb_reg = xgb_reg.predict(X_test)

xgb_reg_end = time.time()

print(f"Time taken to run: {round((xgb_reg_end - xgb_reg_start)/60,1)} minutes")
print("\nTraining MSE:", round(mean_squared_error(y_train, training_preds_xgb_reg),4))
print("Validation MSE:", round(mean_squared_error(y_test, val_preds_xgb_reg),4))
print("\nTraining r2:", round(r2_score(y_train, training_preds_xgb_reg),4))
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'XGBRegressor',
                     'Parameters' : '-',
                     'Training_score': xgb_reg.score(X_train, y_train),
                     'Test Score': xgb_reg.score(X_test, y_test )})
```

```
[17:36:01] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/objective/regression_obj.cu:152: reg:linear i
s now deprecated in favor of reg:squarederror.
Time taken to run: 0.0 minutes

Training MSE: 2695.2386
Validation MSE: 3490.6037

Training r2: 0.7172
```

- The following key metrics/results were obtained after the execution of XGB Regressor Model.
    - Training Set Score: 0.72
    - Test Score: 0.65
    - R-sq: 0.7172

BEST PERFORMING MODEL:

The below mentioned result grid summarizes/provides the Training and Test Scores of all the models that were executed for this particular dataset in our project.

Model Results:

Model Name	Model Type	Parameters	Test Score	Training_score
Linear Regression	Regression	-	5.746322e-01	6.277386e-01
Polynomial Regression	Regression	{'Degree': 2}	-1.365666e+18	-9.918478e+18
KNN Regressor	Regression	{'n_neighbors': 10}	5.103967e-01	6.393940e-01
Ridge Regression	Regression	{'alpha': 100}	5.750267e-01	6.276089e-01
Lasso Regression	Regression	{'alpha': 0.1}	5.747416e-01	6.277062e-01
Linear SVM	Regression	{'C': 1, 'gamma': 0.001}	5.527286e-01	6.081529e-01
SVM Radial	Regression	{'C': 100, 'gamma': 0.01}	6.129452e-01	7.965109e-01
Polynomial SVM	Regression	{'C': 100}	5.408060e-01	7.464654e-01
XGBRegressor	Regression	NaN	6.489768e-01	7.172015e-01

As it is evident from the summarized results grid of various ML models;

- XGB Regressor is the best performing model in our case based on the obtained Training and Test Scores.
- The next best performing model was SVM Radial with the hyperparameters as 'C': 100 and 'gamma': 0.01.

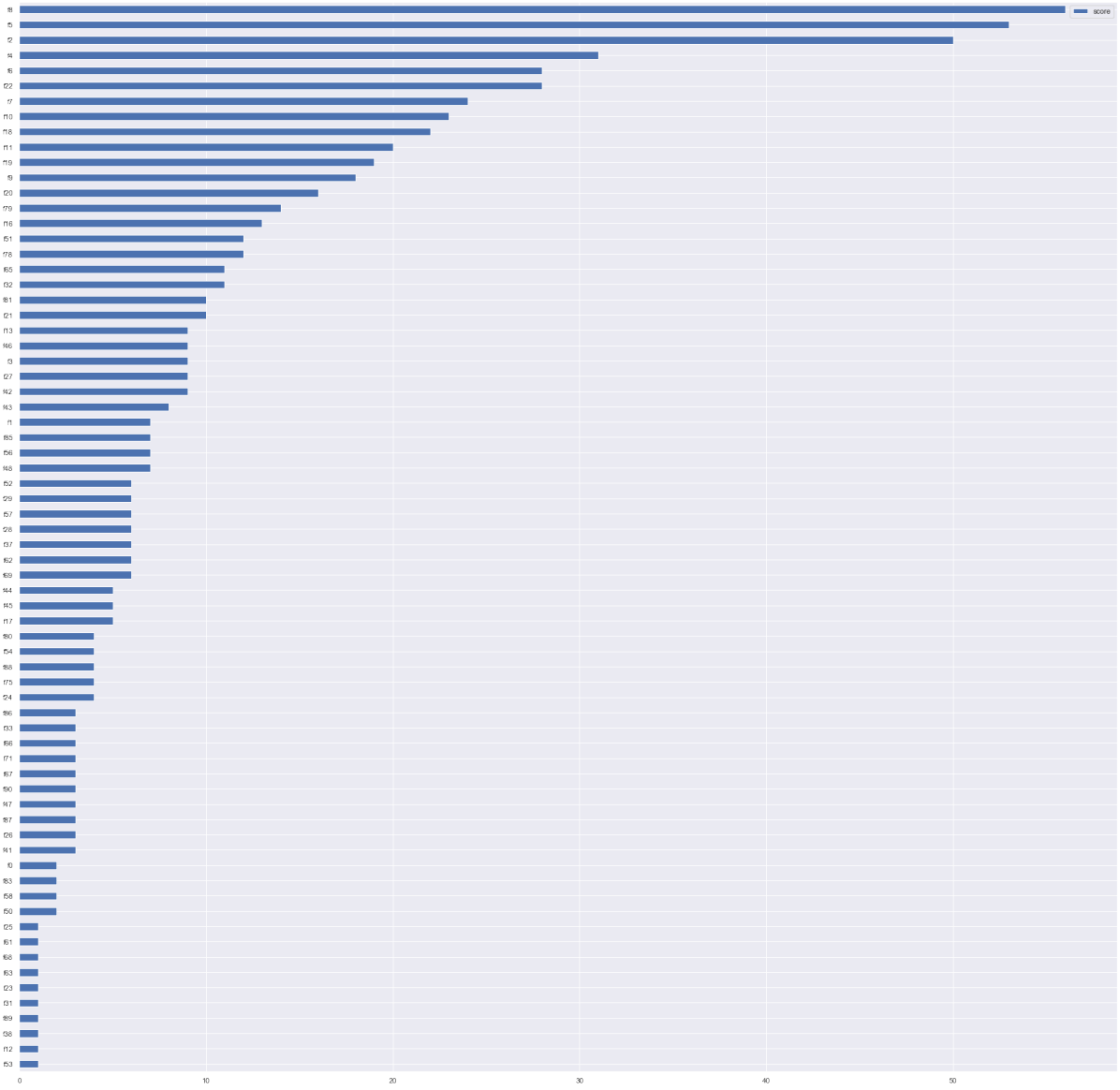
FEATURE IMPORTANCE:

Based on our best performing model; we determined the top 10 important features which would help in determining the price of a new listing for a host.

```
feature_important = xgb_reg.get_booster().get_score(importance_type='weight')
keys = list(feature_important.keys())
values = list(feature_important.values())

data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(by = "score", ascending=True)
print('Feature Importance using XGBRegressor Model')
data.plot(kind='barh',figsize=(30,30))
```

```
Feature Importance using XGBRegressor Model
```



The following ten features would play a major role in determining the price of a new listing for a host:

- minimum_nights
- cleaning_fee
- accommodates
- security_deposit
- guests_included
- reviews_per_month
- extra_people
- number_of_reviews
- calculated_host_listings_count
- review_scores_rating

CONCLUSION

- In our best performing model (XGBRegressor); the model was only able to explain 71% of the variation in price. The remaining 27% may be probably made up of features that were not present in the data. It is also likely that a significant proportion of this unexplained variance may be due to variations in the listing photos.

- Also, we can try to find a way to incorporate image quality into the model.

- Use better quality/more accurate data which includes the actual average prices paid per night.

- Include a wider geographic area.

- Modify/Append the model with natural language processing (NLP) of listing descriptions and/or reviews, e.g. for sentiment analysis.

REFERENCES

- https://rstudio-pubs-static.s3.amazonaws.com/388056_2cf374ef754344669c620f0d24aac5b3.html
- https://www.dataquest.io/blog/machine-learning-tutorial/
- http://insideairbnb.com/get-the-data.html
- https://github.com/L-Lewis/Airbnb-neural-network-price-prediction/blob/master/Airbnb-price-prediction.ipynb

APPENDIX

PYTHON CODE:

Data Manipulation

```
# Importing required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from decimal import Decimal
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from  sklearn.preprocessing  import PolynomialFeatures
from sklearn import svm
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
sns.set()

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

listings = pd.read_csv('listings.csv.gz')
listings.head(5)
listings.duplicated().sum()
listings.shape
listings.dtypes

# delete some information: descriptions, empty ones, same value
listings.drop(['listing_url', 'scrape_id', 'last_scraped', 'name', 'sum
mary', 'space', 'description', 'experiences_offered', 'neighborhood_ove
rview', 'notes', 'transit', 'access', 'interaction','house_rules', 'thu
mbnail_url', 'medium_url', 'picture_url', 'xl_picture_url', 'host_url',
'host_since', 'host_location', 'host_about', 'host_acceptance_rate', 'h
ost_thumbnail_url','host_picture_url', 'host_listings_count', 'host_tot
al_listings_count','host_verifications', 'host_has_profile_pic', 'stree
t', 'neighbourhood_group_cleansed','city', 'state', 'zipcode', 'market'
, 'smart_location', 'country_code', 'country','bed_type', 'amenities',
'square_feet', 'calendar_updated', 'has_availability', 'availability_30
', 'availability_60', 'availability_90', 'availability_365', 'calendar_
last_scraped', 'requires_license','license', 'jurisdiction_names', 'ins
tant_bookable', 'is_business_travel_ready','require_guest_profile_pictu
re', 'require_guest_phone_verification',], axis=1, inplace=True)
```

```
# id: the listing_id used in the reviews.csv
# host_name: not necessary if host_id is used
# weekly_price, monthly_price: too much missing data
# redundant information: minimum_minimum_nights, maximum_minimum_nights
, minimum_maximum_nights, maximum_maximum_nights,
0
#
# first_review, last_review: dates, not useful
# calculated_host_*: statistics of listings for each host,
listings.drop(['host_name', 'weekly_price', 'monthly_price',
               'minimum_minimum_nights', 'maximum_minimum_nights',
               'minimum_maximum_nights', 'maximum_maximum_nights',
               'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm',
               'number_of_reviews_ltm',
               'first_review', 'last_review', #'calculated_host_listings
_count', #'calculated_host_listings_count_entire_homes', 'calculated_ho
st_listings_count_private_rooms', #'calculated_host_listings_count_shar
ed_rooms',], axis=1, inplace=True)

listings['host_neighbourhood'] == listings['neighbourhood']

listings.iloc[11] # the one with False from above commend
                  # based on lat and long, Hayes Valley is the correct
place not Lower Haight,
                  # and there is overlap between "Hayes Valley" and "Lo
wer Haight"
                  # "Western Addition" is quite larger, contains severa
l microhoods
                  # combined with other datat "neightborhood",
                  # "neighbourhood_cleansed" is suggested

listings.drop(['host_neighbourhood', 'neighbourhood'], axis=1, inplace=
True)

listings.drop(['id', 'host_id', 'latitude', 'longitude'], axis=1, inpla
ce=True)

listings.shape

# Superhosts are experienced hosts who provide a shining example for ot
her hosts,
# and extraordinary experiences for their guests.
# review_scores_rating is calculated from other six ratings.
# Overall Experience. What was your guest’s overall experience?
# Cleanliness. Did your guests feel that your space was clean and tidy?
# Accuracy. How accurately did your listing page represent your space?
# Value. Did your guest feel your listing provided good value for the p
rice?
# Communication. How well did you communicate with your guest before an
d during their stay?
# Arrival. How smoothly did their check-in go?
# Location. How did guests feel about your neighborhood?

listings.isnull().sum()

listings.fillna({'review_scores_rating':listings['review_scores_rating'
].mean()}, inplace=True)
```

```
listings.fillna({'review_scores_accuracy':listings['review_scores_accur
acy'].mean()}, inplace=True)
listings.fillna({'review_scores_cleanliness':listings['review_scores_cl
eanliness'].mean()}, inplace=True)
listings.fillna({'review_scores_checkin':listings['review_scores_checki
n'].mean()}, inplace=True)
listings.fillna({'review_scores_communication':listings['review_scores_
communication'].mean()}, inplace=True)
listings.fillna({'review_scores_location':listings['review_scores_locat
ion'].mean()}, inplace=True)
listings.fillna({'review_scores_value':listings['review_scores_value'].
mean()}, inplace=True)

listings[listings['number_of_reviews']==0]['reviews_per_month']

listings.fillna({'reviews_per_month':0}, inplace=True)

# to remove the "$" charactor, we need to change the data type, should
not run the second time
listings['price'] = [float(Decimal(re.sub(r'[^\d.]', '', x))) if type(x
) == str else x for x in listings['price']]
listings['extra_people'] = [float(Decimal(re.sub(r'[^\d.]', '', x))) if
type(x) == str else x for x in listings['extra_people']]
listings['security_deposit'] = [float(Decimal(re.sub(r'[^\d.]', '', x))
) if type(x) == str else x for x in listings['security_deposit']]
listings['cleaning_fee'] = [float(Decimal(re.sub(r'[^\d.]', '', x))) if
type(x) == str else x for x in listings['cleaning_fee']]

# default should be zero
listings.fillna({'security_deposit':0}, inplace=True)
listings.fillna({'cleaning_fee':0}, inplace=True)

listings['host_response_rate'] = [float(x.strip('%')) if type(x) == str
else x for x in listings['host_response_rate']]

listings = listings.dropna(how='any')

listings.shape

listings.isnull().sum()

################ drop some row with strange values ##################

print(listings.describe().loc[['min','max']])

sns.relplot(x="bedrooms", y="room_type", col="host_is_superhost",
            data=listings);

listings_0_bedroom = listings[listings['bedrooms']==0]

listings_0_bedroom.shape

listings_14_bedroom = listings[listings['bedrooms']==14]

listings_14_bedroom.shape

listings = listings[listings['bedrooms']>0]
listings = listings[listings['bedrooms']<14]
```

```
sns.distplot(listings['beds'])

listings_0_bed = listings[listings['beds']==0]

listings_0_bed.shape

listings = listings[listings['beds']>0]

sns.distplot(listings['price'])

print(min(listings['price']), max(listings['price']))

sum(listings['price']>500)

listings = listings[listings['price']>0]
listings = listings[listings['price']<500]

# minimum_nights, maximum_nights: some big values exist, they are stran
ge but should not be directly dropped

# final:
listings.shape

############################## some simple visulization ##############
############################

ax = sns.countplot(x="property_type", data=listings)

ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()

# attributes about the room
listings_room = listings[['room_type', 'accommodates', 'bathrooms', 'be
drooms', 'beds']]
sns.pairplot(data=listings_room, hue="room_type")

# attributes about the reviews
lisitngs_review = listings[['review_scores_rating', 'review_scores_accu
racy', 'review_scores_cleanliness','review_scores_checkin', 'review_sco
res_communication', 'review_scores_location', 'review_scores_value']]

sns.pairplot(data=lisitngs_review)
# note: all review scores are skewed

sns.catplot(x="room_type", y="price", hue="host_is_superhost",
            kind="violin", split=True, data=listings)

sns.catplot(x="bedrooms", y="price", hue="host_is_superhost",
            kind="violin", split=True, data=listings)

sns.relplot(x="number_of_reviews", y="price", col="room_type",
            kind="line", data=listings);

sns.distplot(listings["price"])
plt.show()
sns.boxplot("price",data = listings)
plt.show()
```

```
listings["price"].skew()

# Correlation matrix using scatterplot
corr = listings.corr(method='kendall')
plt.figure(figsize=(15,8))
sns.heatmap(corr, annot=True)
listings.columns

listings['host_response_time'].unique()

response_time = pd.get_dummies(listings["host_response_time"],columns =
"host_response_time", prefix = "host_response_time")
response_time

listings = pd.concat([listings, response_time], axis=1)
listings.drop(['host_response_time'],axis=1,inplace= True)
listings

listings["host_is_superhost"].unique()
listings.describe()

listings["host_is_superhost"] = listings["host_is_superhost"].map({'f'
: 0, 't' : 1}).astype(int)

listings["neighbourhood_cleansed"].nunique()

series = pd.value_counts(listings.neighbourhood_cleansed)
mask = (series/series.sum() * 100).lt(1)
listings['neighbourhood_cleansed'] = np.where(listings['neighbourhood_c
leansed'].isin(series[mask].index),'Other',listings['neighbourhood_clea
nsed'])
print(listings['neighbourhood_cleansed'].nunique())
pvt = listings.pivot_table(index= "neighbourhood_cleansed", values="cal
culated_host_listings_count", aggfunc='count')
pvt = pvt.sort_values("calculated_host_listings_count",ascending=False)
pvt

pvt.plot(kind='bar', color='blue')

neighborhood = pd.get_dummies(listings["neighbourhood_cleansed"],column
s = "neighbourhood_cleansed", prefix = "neighbourhood")
listings = pd.concat([listings, neighborhood], axis=1)
listings.drop(["neighbourhood_cleansed"],axis=1,inplace= True)
listings

listings["is_location_exact"] = listings["is_location_exact"].map({'f'
: 0, 't' : 1}).astype(int)

listings["property_type"].unique()

listings["property_type"].nunique()

pivot = listings.pivot_table(index= "property_type", values="calculated
_host_listings_count", aggfunc='count')
pivot = pivot.sort_values("calculated_host_listings_count",ascending=Fa
lse)
pivot.plot(kind='bar', color='pink')
plt.show()
```

```
pivot

series1 = pd.value_counts(listings.property_type)
mask1 = (series1/series1.sum() * 100).lt(1)
listings['property_type'] = np.where(listings['property_type'].isin(ser
ies1[mask1].index),'Other',listings['property_type'])
print(listings['property_type'].nunique())
pivot1 = listings.pivot_table(index= "property_type", values="calculate
d_host_listings_count", aggfunc='count')
pivot1 = pivot1.sort_values("calculated_host_listings_count",ascending=
False)
pivot1.plot(kind='bar', color='pink')
plt.show()
pivot1

property_type = pd.get_dummies(listings["property_type"],columns = "pro
perty_type", prefix = "property_type")
listings = pd.concat([listings, property_type], axis=1)
listings.drop(['property_type'],axis=1,inplace= True)
listings

listings["room_type"].unique()

room_type = pd.get_dummies(listings["room_type"],columns = "room_type",
prefix = "room_type")
listings = pd.concat([listings, room_type], axis=1)
listings.drop(['room_type'],axis=1,inplace= True)
listings

listings["cancellation_policy"].unique()

pivot4 = listings.pivot_table(index= "cancellation_policy", values="cal
culated_host_listings_count", aggfunc='count')
pivot4 = pivot4.sort_values("calculated_host_listings_count",ascending=
False)
pivot4

listings["cancellation_policy"] = listings["cancellation_policy"].map({
'strict_14_with_grace_period' : 'strict_with_grace_period', 'super_stri
ct_30' : 'super_strict', 'moderate': 'moderate','flexible': 'flexible',
'super_strict_60':'super_strict','strict' : 'strict'})

pivot4 = listings.pivot_table(index= "cancellation_policy", values="cal
culated_host_listings_count", aggfunc='count')
pivot4 = pivot4.sort_values("calculated_host_listings_count",ascending=
False)
pivot4

cancellation_policy = pd.get_dummies(listings["cancellation_policy"],co
lumns = "cancellation_policy", prefix = "cancellation")
listings = pd.concat([listings, cancellation_policy], axis=1)
listings.drop(['cancellation_policy'],axis=1,inplace= True)
listings

listings.info()

df1count = listings[(listings["host_response_rate"] >= 90) &(listings["
host_response_rate"]<=100)].count()
```

```
#listings["host_response_rate"] = listings["host_response_rate"].astype
(object)
df1 = listings[(listings["host_response_rate"] >= 90) &(listings["host_
response_rate"]<=100)]
c1 = df1["host_response_rate"].count()
print(c1)
df2 = listings[(listings["host_response_rate"] < 90) & (listings["host_
response_rate"]>=80)]
c2 = df2.host_response_rate.count()
print(c2)
df3 = listings[(listings["host_response_rate"] < 80) & (listings["host_
response_rate"]>=70)]
c3 = df3.host_response_rate.count()
print(c3)
df4 = listings[(listings["host_response_rate"] < 70) & (listings["host_
response_rate"]>=60)]
c4 = df4.host_response_rate.count()
print(c4)
df5 = listings[(listings["host_response_rate"] < 60)]
c5 = df5.host_response_rate.count()
print(c5)
print(c1+c2+c3+c4+c5)
pivot5 = df5.pivot_table(index= "host_response_rate", values="price", a
ggfunc='mean')
pivot5 = pivot5.sort_values("price",ascending=False)

sns.boxplot(df5["host_response_rate"])
plt.show()
pivot5

listings["host_response_rate"] = np.where(((listings['host_response_rat
e'] >=90) & (listings['host_response_rate'] <=100)), 95, listings['host
_response_rate'])
listings["host_response_rate"] = np.where(((listings['host_response_rat
e'] >=80) & (listings['host_response_rate'] <90)), 85, listings['host_r
esponse_rate'])
listings["host_response_rate"] = np.where(((listings['host_response_rat
e'] >=70) & (listings['host_response_rate'] <80)), 75, listings['host_r
esponse_rate'])
listings["host_response_rate"] = np.where(((listings['host_response_rat
e'] >=60) & (listings['host_response_rate'] <70)), 65, listings['host_r
esponse_rate'])
listings["host_response_rate"] = np.where((listings['host_response_rate
'] <60), 50, listings['host_response_rate'])
listings["host_response_rate"] = listings["host_response_rate"].astype(
object)
listings["host_response_rate"] = listings["host_response_rate"].map({ 9
5.0 : '90-100',85.0 : '80-90', 75.0 : '70-80',65.0: '60-70',50.0:'<60'}
)

host_response_rate = pd.get_dummies(listings["host_response_rate"],colu
mns = "host_response_rate", prefix = "host_response_rate")
listings = pd.concat([listings, host_response_rate], axis=1)
listings.drop(['host_response_rate'],axis=1,inplace= True)
listings

listings = listings[listings["bedrooms"] <= 5]
```

```
print(listings.bedrooms.count())
pivot5 = listings.pivot_table(index= "bedrooms", values="price", aggfun
c='mean')
pivot5 = pivot5.sort_values("price",ascending=False)
pivot5.plot(kind='bar', color='pink')
plt.show()
pivot5

bedrooms = pd.get_dummies(listings["bedrooms"],columns = "bedrooms", pr
efix = "bedrooms")
listings = pd.concat([listings, bedrooms], axis=1)
listings.drop(['bedrooms'],axis=1,inplace= True)
listings

listings = listings[listings["bathrooms"] <= 5]

print(listings.bathrooms.count())
pivot6 = listings.pivot_table(index= "bathrooms", values="price", aggfu
nc='mean')

pivot6.plot(kind='bar', color='pink')
plt.show()
pivot6

bathrooms = pd.get_dummies(listings["bathrooms"],columns = "bathrooms",
prefix = "bathrooms")
listings = pd.concat([listings, bathrooms], axis=1)
listings.drop(['bathrooms'],axis=1,inplace= True)
listings

listings = listings[listings["beds"] <= 6]
listings["beds"] = listings["beds"].astype(int)
print(listings.beds.count())
pivot8 = listings.pivot_table(index= "beds", values="price", aggfunc='m
ean')
pivot8 = pivot8.sort_values("price",ascending=False)
pivot8.plot(kind='bar', color='pink')
plt.show()
pivot8

pivot7 = listings.pivot_table(index = "accommodates", values ="price",
aggfunc ='mean')
pivot7 = pivot7.sort_values("price",ascending = False)
pivot7.plot(kind ='bar', color ='pink')
plt.show()
pivot7

pivot9 = listings.pivot_table(index = "guests_included", values ="price
", aggfunc ='mean')
pivot9 = pivot9.sort_values("price",ascending = False)
pivot9.plot(kind ='bar', color ='pink')
plt.show()
pivot9
listings.info()
```

FEATURE SCALING:

```
listings.drop(columns = ["host_identity_verified"],inplace = True)

y = listings["price"]
X = listings.drop(columns = ["price"])
X_train_org,X_test_org,y_train,y_test=train_test_split(X,y,random_state
=0,test_size=0.3)

X_train_org.shape

X_test_org.shape

sc = StandardScaler()
X_train = sc.fit_transform(X_train_org)
X_test = sc.transform(X_test_org)
```

MODELING:

LINEAR REGRESSION

```
train_score = []
test_score = []
model_Scores = []

from sklearn.linear_model import LinearRegression
lreg = LinearRegression()
lreg.fit(X_train, y_train)
print('Train Score:',lreg.score(X_train, y_train))
train_score.append(lreg.score(X_train, y_train))
test_score.append(lreg.score(X_test, y_test))
train_score = lreg.score(X_train, y_train)
test_score = lreg.score(X_test,y_test)
print("Test Score {:.2f}".format(test_score))
## Appending the results
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Linear Regression',
                     'Parameters' : '-',
                     'Training_score': train_score,
                     'Test Score':test_score})

y_pred_lin = lreg.predict(X_test)
r2_score(y_test,y_pred_lin)

%matplotlib inline
import matplotlib.pyplot as plt
X_train_rm = X_train[:,5].reshape(-1,1)
lreg.fit(X_train_rm, y_train)
y_predict = lreg.predict(X_train_rm)
plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
plt.xlabel('RM')

from sklearn import model_selection
kfold = model_selection.KFold(n_splits = 10, random_state = 100)
model_kfold = LinearRegression()
```

```
results_kfold = model_selection.cross_val_score(model_kfold, X, y, cv =
kfold)
print("Accuracy: %.2f%%" % (results_kfold.mean()*100.0))
```

POLYNOMIAL REGRESSION

```
from  sklearn.preprocessing  import PolynomialFeatures
X_train_1 = X_train[:,5].reshape(-1,1)
plt.scatter(X_train_1,y_train)

train_score_list = []
test_score_list = []
for n in range(2,4):
    poly = PolynomialFeatures(n)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)
    lreg.fit(X_train_poly, y_train)
    scores_train = cross_val_score(lreg, X_train_poly, y_train, cv=3)
    scores_test = lreg.score(X_test_poly,y_test)
    test_score_list.append(scores_test)
    train_score_list.append(scores_train.mean())

print("Train Scores", train_score_list)
print("Test Scores", test_score_list)

print("Train Scores", train_score_list[1])
print("Test Scores", test_score_list[1])
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Polynomial Regression',
                     'Parameters' : {'Degree':2},
                     'Training_score':train_score_list[1],
                     'Test Score':test_score_list[1]})

%matplotlib inline
x_axis = range(1,3)
plt.plot(x_axis, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_list, c = 'b', label = 'Test Score')
plt.xlabel('degree')
plt.ylabel('accuracy')
plt.legend()

poly = PolynomialFeatures(n)
X_train_poly = poly.fit_transform(X_train_1)
lreg.fit(X_train_poly, y_train)
x_axis = np.linspace(0,1,100).reshape(-1,1)
x_poly = poly.transform(x_axis)
y_predict = lreg.predict(x_poly)
X_train_1 = X_train[:,5].reshape(-1,1)
plt.scatter(X_train_1,y_train)
plt.plot(x_axis, y_predict, c = 'r')
```

KNN REGRESSOR

```
%matplotlib inline
train_score_array = []
test_score_array = []

for k in range(1,10):
    knn_reg = KNeighborsRegressor(k)
    knn_reg.fit(X_train, y_train)
    train_score_array.append(knn_reg.score(X_train, y_train))
    test_score_array.append(knn_reg.score(X_test, y_test))

x_axis = range(1,10)
plt.plot(x_axis, train_score_array, c = 'g', label = 'Train Score')
plt.plot(x_axis, test_score_array, c = 'b', label = 'Test Score')
plt.legend()
plt.xlabel('k')
plt.ylabel('MSE')

print(train_score_array)
print(test_score_array)

def printGridResult (model_def) :
    print("Best CV result: {}".format(model_def.best_score_))
    print("Best parameters: {}".format(model_def.best_params_))
    print("Training Score: {}".format(model_def.score(X_train, y_train)
))
    print("Testing Score: {}".format(model_def.score(X_test, y_test)))

#KNN Regressor
knnRegressor = KNeighborsRegressor()
noofNeighbors = {'n_neighbors':[1, 5, 10, 15, 20]}
print("Parameter grid:\n{}".format(noofNeighbors))
knnGridSCV = GridSearchCV(knnRegressor, param_grid = noofNeighbors, cv=
5, return_train_score=True)
knnGridSCV.fit(X_train, y_train)

n_neighbors = [1, 5, 10, 15, 20]
n_neighbors = [1, 5, 10, 15, 20]

train_knnRegressor = knnGridSCV.cv_results_["mean_train_score"]
test_knnRegressor = knnGridSCV.cv_results_["mean_test_score"]
printGridResult(knnGridSCV)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'KNN Regressor',
                     'Parameters' : knnGridSCV.best_params_,
                   'Training_score': knnGridSCV.score(X_train, y_train),
                     'Test Score': knnGridSCV.score(X_test, y_test )})

X_b = X_train[:50,6].reshape(-1,1)
y_b = y_train[:50]
knn_reg = KNeighborsRegressor(10)
knn_reg.fit(X_b, y_b)
X_new=np.linspace(X_b.min(), X_b.max(), 50).reshape(50, 1)
y_predict = knn_reg.predict(X_new)
plt.plot(X_new, y_predict, c = 'r')
plt.scatter(X_b, y_b)
```

RIDGE REGRESSION

```
from  sklearn.linear_model import Ridge
x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    train_score_list.append(ridge.score(X_train,y_train))
    test_score_list.append(ridge.score(X_test, y_test))
print(train_score_list)
print(test_score_list)

%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')

# Ridge Regression
ridgeParamGSV = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(ridgeParamGSV))

ridgeGridSV = GridSearchCV(estimator = Ridge(random_state = 0 ),param_g
rid = ridgeParamGSV,
                           cv=5, return_train_score=True, scoring='r2',
n_jobs=1)
ridgeGridSV.fit(X_train, y_train)

ridge_train_scores_mean = ridgeGridSV.cv_results_["mean_train_score"]
ridge_test_scores_mean = ridgeGridSV.cv_results_["mean_test_score"]

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Ridge Regression',
                     'Parameters' : ridgeGridSV.best_params_,
                  'Training_score': ridgeGridSV.score(X_train, y_train),
                'Test Score': ridgeGridSV.score(X_test, y_test )})
printGridResult(ridgeGridSV)

%matplotlib inline
import numpy as np
x_range1 = np.linspace(0.001, 1, 100).reshape(-1,1)
x_range2 = np.linspace(1, 10000, 10000).reshape(-1,1)
x_range = np.append(x_range1, x_range2)
coeff = []

for alpha in x_range:
    ridge = Ridge(alpha)
    ridge.fit(X_train,y_train)
    coeff.append(ridge.coef_ )

coeff = np.array(coeff)
```

```
for i in range(0,20):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))
plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()

from sklearn import model_selection
kfold = model_selection.KFold(n_splits = 10, random_state = 100)
model_kfold = Ridge(alpha = 100)
results_kfold = model_selection.cross_val_score(model_kfold, X, y, cv=k
fold)
print("Accuracy: %.2f%%" %(results_kfold.mean()*100.0))
```

LASSO REGRESSION

```
from sklearn.linear_model import Lasso
x_range = [0.01, 0.1, 1, 10, 100]
train_score_list = []
test_score_list = []

for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    train_score_list.append(lasso.score(X_train,y_train))
    test_score_list.append(lasso.score(X_test, y_test))
print(train_score_list)
print(test_score_list)

#Lasso Regression
lassoParamGSV = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(lassoParamGSV))

lassoGridSV = GridSearchCV(estimator = Lasso(random_state = 0 ),param_g
rid = lassoParamGSV,
                           cv=5, return_train_score=True, scoring='r2',
n_jobs=1)
lassoGridSV.fit(X_train, y_train)

lasso_train_scores_mean = lassoGridSV.cv_results_["mean_train_score"]
lasso_test_scores_mean = lassoGridSV.cv_results_["mean_test_score"]

model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Lasso Regression',
                     'Parameters' : lassoGridSV.best_params_,
                   'Training_score': lassoGridSV.score(X_train, y_train),
                     'Test Score': lassoGridSV.score(X_test, y_test )})
printGridResult(lassoGridSV)


%matplotlib inline
x_range1 = np.linspace(0.001, 1, 1000).reshape(-1,1)
x_range2 = np.linspace(1, 1000, 1000).reshape(-1,1)
x_range = np.append(x_range1, x_range2)
coeff = []
```

```
for alpha in x_range:
    lasso = Lasso(alpha)
    lasso.fit(X_train,y_train)
    coeff.append(lasso.coef_ )

coeff = np.array(coeff)

for i in range(0,20):
    plt.plot(x_range, coeff[:,i], label = 'feature {:d}'.format(i))

plt.axhline(y=0, xmin=0.001, xmax=9999, linewidth=1, c ='gray')
plt.xlabel(r'$\alpha$')
plt.xscale('log')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.5),
          ncol=3, fancybox=True, shadow=True)
plt.show()

from sklearn import model_selection
kfold = model_selection.KFold(n_splits = 10, random_state = 100)
model_kfold = Lasso(alpha = 0.1)
results_kfold = model_selection.cross_val_score(model_kfold, X, y, cv=k
fold)
print("Accuracy: %.2f%%" %(results_kfold.mean()*100.0))
```

SVM REGRESSION

```
from sklearn import svm
tuning_parameters = [{'C': [0.01, 0.1 ,1, 10, 100], 'kernel': ['linear'
]},
  {'C': [0.01, 0.1 ,1, 10, 100], 'gamma': [0.1,0.01, 0.001, 0.0001, 0.0
0001], 'kernel': ['rbf']}]

svm_grid = GridSearchCV(svm.SVR(), param_grid=tuning_parameters,cv=5)

svm_model=svm_grid.fit(X_train,y_train)
print(svm_model.best_params_)
print('Validation score:', svm_model.best_score_)

#SVM With Linear Kernal
from sklearn.svm import LinearSVR,SVR
linearSVM = SVR(kernel='linear')
param_Linear_SVM = {
    'gamma': [0.001, 0.01, 0.1, 1, 10],
    'C': [0.001, 0.01, 0.1, 1, 10],
}

LinearSVMGrid = GridSearchCV(estimator=linearSVM, param_grid=param_Line
ar_SVM, cv=3, n_jobs=-1)
LinearSVMGrid.fit(X_train, y_train)

print('Train score:',LinearSVMGrid.score(X_train, y_train))
print('Test score:',LinearSVMGrid.score(X_test, y_test))
print('Best parameters:',LinearSVMGrid.best_params_)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Linear SVM',
                     'Parameters' : LinearSVMGrid.best_params_,
```

```
                'Training_score': LinearSVMGrid.score(X_train, y_train),
                   'Test Score': LinearSVMGrid.score(X_test, y_test )})

#SVM With Radial Kernal
svmRadial = SVR(kernel = 'rbf')
param_Radial_SVM = {'C': [0.001, 0.01, 0.1, 1, 10, 100],'gamma': [0.001
, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_Radial_SVM))
svmRadialGridSV  = GridSearchCV(svmRadial, param_grid = param_Radial_SV
M, cv=5, return_train_score=True)

svmRadialGridSV.fit(X_train, y_train)
svmRadialGridSV_results = pd.DataFrame(svmRadialGridSV.cv_results_)
printGridResult(svmRadialGridSV)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'SVM Radial',
                     'Parameters' : svmRadialGridSV.best_params_,
               'Training_score': svmRadialGridSV.score(X_train, y_train),
                   'Test Score': svmRadialGridSV.score(X_test, y_test )})

#Polynomial SVM Kernal
polySVM = SVR(kernel = 'poly', degree = 2)
paramPolySVM = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(paramPolySVM))
polySVMGridSV = GridSearchCV(polySVM, param_grid = paramPolySVM, cv=5,
return_train_score= True)

polySVMGridSV.fit(X_train, y_train)

printGridResult(polySVMGridSV)
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'Polynomial SVM',
                     'Parameters' : polySVMGridSV.best_params_,
                 'Training_score': polySVMGridSV.score(X_train, y_train),
                    'Test Score': polySVMGridSV.score(X_test, y_test )})
```

XGBRegressor

```
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import time
from xgboost.sklearn import XGBRegressor

xgb_reg_start = time.time()

xgb_reg = xgb.XGBRegressor()
xgb_reg.fit(X_train, y_train)
training_preds_xgb_reg = xgb_reg.predict(X_train)
val_preds_xgb_reg = xgb_reg.predict(X_test)

xgb_reg_end = time.time()

print(f"Time taken to run: {round((xgb_reg_end - xgb_reg_start)/60,1)}
minutes")
print("\nTraining MSE:", round(mean_squared_error(y_train, training_pre
ds_xgb_reg),4))
```

```
print("Validation MSE:", round(mean_squared_error(y_test, val_preds_xgb
_reg),4))
print("\nTraining r2:", round(r2_score(y_train, training_preds_xgb_reg)
,4))
model_Scores.append({'Model Type' : 'Regression',
                     'Model Name' : 'XGBRegressor',
                     'Parameters' : '-',
                     'Training_score': xgb_reg.score(X_train, y_train),
                     'Test Score': xgb_reg.score(X_test, y_test )}

feature_important = xgb_reg.get_booster().get_score(importance_type='we
ight')
keys = list(feature_important.keys())
values = list(feature_important.values())
data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_va
lues(by = "score", ascending=True)
print('Feature Importance using XGBRegressor Model')
data.plot(kind='barh',figsize=(30,30))
```

Model Results

```
print("Model Results:")
modelResult = pd.DataFrame(model_Scores)
# Dropped the duplicate column 'Best Parameters'as we already have Para
meters displayed in the Result grid
modelResult.drop('Best Parameters', axis = 1, inplace = True)
modelResult.set_index('Model Name', inplace = True)
modelResult
```