Finalcaps

February 24, 2020

1 CapitalOne Data Challenge

1.1 Business Problem

You are consulting for a real estate company that has a niche in purchasing properties to rent out short-term as part of their business model specifically within New York City. The real estate company has already concluded that two bedroom properties are the most profitable; however, they do not know which zip codes are the best to invest in. The real estate company has engaged your firm to build out a data product and provide your conclusions to help them understand which zip codes would generate the most profit on short term rentals within New York City.

1.2 Assumptions

- Investors have already identified that 2 bedroom properties are most profitable and want to invest in only those properties
- Investers are interested to buy properties only in New York City
- The occupancy rate is assumed to be 75%

- The investor will pay for the property in cash (i.e. no mortgage/interest rate will need to be accounted for).
- The time value of money discount rate is 0% (i.e. $1 today is worth the same 100 years from now).
- All properties and all square feet within each locale can be assumed to be homogeneous.
- Only the monthly date will be considered (i.e. all the data will be consolidated into monthly data wherever daily data is provided)

[ ]:
```

```

1.2.1 install and import required packages

[1]:
```
#unccomment the following line to install required packages
#pip install -r requirements.txt
```

[1]:
```
#import all the required packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import gzip


from fbprophet import Prophet
from fbprophet.plot import plot_plotly
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.offline as py
py.init_notebook_mode()
%matplotlib inline


import missingno as msno
import plotly.express as px
```

1.2.2 importing the functions created for the purpose of this analysis

[2]:
```
import functions
```

[ ]:
```

```

1.2.3 Load the dataset

[3]:
```
airbnb = pd.read_csv('listings.csv',low_memory=False) #load the airbnb revenue␣
 ↪dataset
zillow = pd.read_csv('Zip_Zhvi_2bedroom.csv',low_memory=False) # load the␣
 ↪zillow cost dataset
```

.

1.2.4 Quick view of both the dataset

1.2.5 Airbnb data (Revenue data)

[4]:
```
airbnb.head(3)
```

[4]:
```
     id                          listing_url       scrape_id last_scraped  \
0  2539  https://www.airbnb.com/rooms/2539  20190708031610   2019-07-09
1  2595  https://www.airbnb.com/rooms/2595  20190708031610   2019-07-09
2  3647  https://www.airbnb.com/rooms/3647  20190708031610   2019-07-08

                                  name  \
0   Clean & quiet apt home by the park
```

```
1                Skylit Midtown Castle
2  THE VILLAGE OF HARLEM….NEW YORK !

                                             summary  \
0           Renovated apt home in elevator building.
1  Find your romantic getaway to this beautiful, …
2                                                NaN

                                               space  \
0  Spacious, renovated, and clean apt home, one b…
1  - Spacious (500+ft²), immaculate and nicely fu…
2  WELCOME TO OUR INTERNATIONAL URBAN COMMUNITY T…

                                         description experiences_offered  \
0  Renovated apt home in elevator building. Spaci…                none
1  Find your romantic getaway to this beautiful, …                none
2  WELCOME TO OUR INTERNATIONAL URBAN COMMUNITY T…                none

                              neighborhood_overview  … instant_bookable  \
0    Close to Prospect Park and Historic Ditmas Park  …                f
1  Centrally located in the heart of Manhattan ju…  …                f
2                                                NaN  …                f

 is_business_travel_ready          cancellation_policy  \
0                       f                     moderate
1                       f  strict_14_with_grace_period
2                       f  strict_14_with_grace_period

 require_guest_profile_picture require_guest_phone_verification  \
0                            f                                f
1                            t                                t
2                            t                                t

  calculated_host_listings_count  \
0                              6
1                              2
2                              1

  calculated_host_listings_count_entire_homes  \
0                                            0
1                                            1
2                                            0

 calculated_host_listings_count_private_rooms  \
0                                            5
1                                            0
2                                            1
```

```
   calculated_host_listings_count_shared_rooms  reviews_per_month
0                                            1               0.21
1                                            1               0.38
2                                            0                NaN

[3 rows x 106 columns]
```

1.2.6 No of rows and columns in the dataset

[5]:
```
# no of rows and columns present in the airbnb data
airbnb_rows = airbnb.shape[0]
airbnb_columns = airbnb.shape[1]
print('There are {} rows and {} columns in the airbnb listings dataset.'.
 ↪format(airbnb_rows,airbnb_columns))
```

```
There are 48895 rows and 106 columns in the airbnb listings dataset.
```

.

1.2.7 Zillow data (cost data)

[6]:
```
zillow.head()
```

[6]:
```
   RegionID  RegionName      City State     Metro CountyName  SizeRank  \
0     61639       10025  New York    NY  New York   New York         1
1     84654       60657   Chicago    IL   Chicago       Cook         2
2     61637       10023  New York    NY  New York   New York         3
3     84616       60614   Chicago    IL   Chicago       Cook         4
4     93144       79936   El Paso    TX   El Paso    El Paso         5

    1996-04   1996-05   1996-06  …  2016-09  2016-10  2016-11    2016-12  \
0       NaN       NaN       NaN  …  1374400  1364100  1366300  1354800.0
1  167700.0  166400.0  166700.0  …   368600   370200   372300   375300.0
2       NaN       NaN       NaN  …  1993500  1980700  1960900  1951300.0
3  195800.0  193500.0  192600.0  …   398900   401200   403200   405700.0
4   59100.0   60500.0   60900.0  …    82400    82300    82400    82300.0

   2017-01  2017-02  2017-03  2017-04  2017-05  2017-06
0  1327500  1317300  1333700  1352100  1390000  1431000
1   378700   381400   381800   382100   383300   385100
2  1937800  1929800  1955000  2022400  2095000  2142300
3   408300   408800   408000   410100   412200   412200
4    82500    83200    83900    84100    83900    83700

[5 rows x 262 columns]
```

1.2.8 No of rows and columns in the dataset

[7]:
```
# no of rows and columns present in the zillow data
zillow_rows = zillow.shape[0]
zillow_columns = zillow.shape[1]
print('There are {} rows and {} columns in the zillow dataset.'.
 ↪format(zillow_rows,zillow_columns))
```

```
There are 8946 rows and 262 columns in the zillow dataset.
```

.

2 Data Cleaning

2.1 Airbnb data

2.1.1 Data Scrape information

[8]:
```
#unique values in the 'last_scraped column'
scrape = [x for x in airbnb.last_scraped.unique()]
print("The data in airbnb dataset were scraped on: ")
for i in scrape:
    print(i)
```

```
The data in airbnb dataset were scraped on:
2019-07-09
2019-07-08
```

The 'scrape_id' and 'last_scraped' columns provides us the scrape id and the latest date when the data was scraped. This is a valuable information for us as it provides us with the date regarding the data. From the above code we can see that the data has be scraped in two days whose dates are '2019-07-09' and '2019-07-08'. We have already made the assumption that for our analysis we will consolidating the daily data into monthly data. Therefore, it is safe for us to consider the data to be scraped on '2019-07' or 'July 2019' Why do we have to keep this in mind? This is useful if we have to forecast the data from the zillow dataset as it is best for us to have the data from same time period for further analysis.

2.1.2 Inspecting for multiple nation data

[9]:
```
## using the check_non_us function created and saved under the functions module
##check_non_us functions gives the user the numbers of country and country code␣
 ↪present in the data besides the specified ones

df = airbnb
country = 'United States'
country_code = 'US'
functions.check_non_us(df, country, country_code)
```

```
There are 0 countries other than United States in the dataset
There are 0 country_code other than US in the dataset
```

2.2 Basis of preliminary data preprocessing and feature selection

After confirming that we have data only for US cities we can drop country and country_code column. Furthermore, Since the investor wants us to specifically analyse only New York City, we can select features such that we can filter our data for New York city. Also, 'neighbourhood_cleansed', 'neighbourhood_cleansed_grouped' are verified information about neighbourhood so we can just choose those.We will select only the columns required as having too many features that have same information is reduntant.

After inspecting the AirBnB dataset and Metadata document further, it is apparent that there are a lot of columns in the dataset; 106 columns to be exact. Considering the nature of our analysis, instruction from the investors and assumptions, we can conduct a preliminary data preprocessing in order to get a dataset with smaller and relevant features without running any statistical analysis.

The purpose of the analysis is to find the zipcodes which are most profitable. For this, we need to focus only on the macro-economic factors or variables rather than the micro variables. The investing company is looking to buy properties and rent it out through AirBnB so all the variables like , house rules, amenities, access, security deposit, cleaning fee, host related data, host interaction data, extra people and guest related data are not needed as they all vary from renters and the investing company can work to make it the best interest of guest so they sell faster. Also, there are various data related to the urls for the airbnb listing and images which we do not need.

The listings data also has various identifier columns and scrape information columns. For our purpose, we do not need any unique identifier columns as the data will be grouped by zipcode and there is no need to have granularity in the data beyond the zipcode. Furthermore, there are various review related features but only review score related to location is relevant to our analysis. So, based on above criteria I have done preliminary feature selection with purpose of retaining only the important features.

Also, we only need to find the profitable zipcodes. Therefore, we can remove latitude and longitude from the dataset as we do not need in depth analysis of the location.

Simialrly, the investors already know that 2 bedroom properties are most profitable so their only criteria is 2 bedroom and the type of property doesn't matter. Therefore we can remove the type of property as well as all other features of property exceept the number of bedroom.

From the beginning, one of our assuption is that the occupancy rate of 75%. Therefore, we do not have to take the availability data into consideration and we can drop them.

There are various columns having unique string data and need help of Natural Language Processing models to process those dataand meaninful insights from them. So for our analysis, we will drop those columns as they add no value to us without NLP.

There are various review scores data but we only need the "review_scores_location" columns as it provides us with the customers score of the location. All other review scores are dependent on the accuracy of description posted, cleanliness, communication and so on which depends on the host and after purchasing the property, the investing company can ensure that all of those are in best interest of renters. The only review score that the investor cannot control is the location review score, so we will add this column for our analysis

.

2.2.1 Features selected after preliminary data preprocessing

[10]:
```
#fearutes identified as important ones based on the assumptions and scope of␣
↪analysis
features = ['neighbourhood_group_cleansed', 'bedrooms','city',
       'state', 'zipcode', 'square_feet', 'price', 'weekly_price',
       'monthly_price', 'review_scores_location']
```

2.2.2 creating a dataframe with only the required columns and printing the column names

[11]:
```
#using the prelim_preprocessing function saved under functions module
df1_airbnb = functions.prelim_preprocessing(airbnb, features)
```

```
Features selected after preliminary data preprocessing are:
neighbourhood_group_cleansed
bedrooms
city
state
zipcode
square_feet
price
weekly_price
monthly_price
review_scores_location
```

[12]:
```
#viewing the new dataframe
df1_airbnb
```

[12]:
```
      neighbourhood_group_cleansed  bedrooms      city state zipcode  \
0                         Brooklyn       1.0  Brooklyn    NY   11218
1                        Manhattan       0.0  New York    NY   10018
2                        Manhattan       1.0  New York    NY   10027
3                         Brooklyn       1.0  Brooklyn    NY   11238
4                        Manhattan       NaN  New York    NY   10029
…                              …       …        …   …     …
48890                     Brooklyn       1.0  Brooklyn    NY   11216
48891                     Brooklyn       1.0  Brooklyn    NY   11206
48892                    Manhattan       0.0  New York    NY   10027
48893                    Manhattan       1.0  New York    NY   10036
48894                    Manhattan       1.0  New York    NY   10019

       square_feet    price weekly_price monthly_price  review_scores_location
0              NaN  $149.00      $299.00       $999.00                    10.0
1              NaN  $225.00    $1,995.00           NaN                    10.0
```

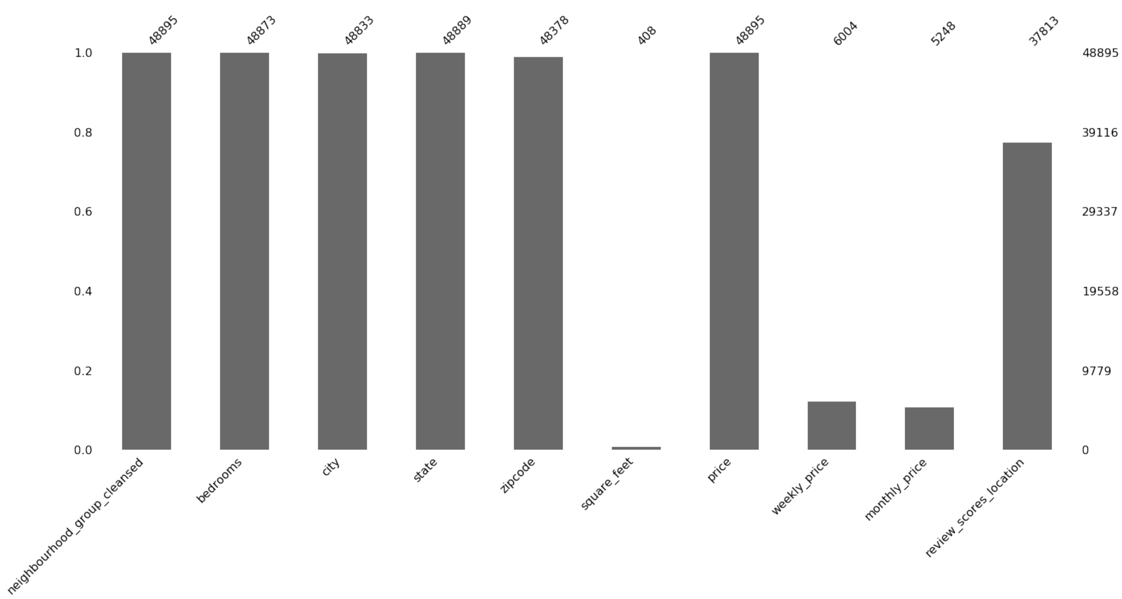
```
2              NaN  $150.00        NaN        NaN             NaN
3            500.0   $89.00    $575.00  $2,100.00            10.0
4              NaN   $80.00    $600.00  $1,600.00             9.0
…              …      …           …          …        …
48890          NaN   $70.00        NaN        NaN             NaN
48891          NaN   $40.00        NaN        NaN             NaN
48892          NaN  $115.00        NaN        NaN             NaN
48893          NaN   $55.00        NaN        NaN             NaN
48894          NaN   $90.00        NaN        NaN             NaN

[48895 rows x 10 columns]
```

[13]:
```
#data = df1_airbnb.isna()
```

2.2.3 Plotting the completeness of the data

[14]:
```
# Visualize the completeness/ missing values, hte bar graphs shows how complete␣
↪the data is
#Higher the bar, higher the data completeness and lower the NULL values and␣
↪vice versa
# values as a bar chart
msno.bar(df1_airbnb)
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2713a972dc8>
```



2.2.4 Plotting the missing/ NULL values

[15]:
```
#using the missing_value function saved under functions module
#missing_value functions takes in a dataframe and gives a table and plot of the␣
↪total nulls and percentage of nulls for each columns in the dataframe
functions.missing_value(df1_airbnb)
```

```
                             null_percentage  total_null_values
neighbourhood_group_cleansed            0.00                  0
bedrooms                                0.04                 22
city                                    0.13                 62
state                                   0.01                  6
zipcode                                 1.06                517
square_feet                            99.17              48487
price                                   0.00                  0
weekly_price                           87.72              42891
monthly_price                          89.27              43647
review_scores_location                 22.66              11082
```

From the above table and bar chart it clear that the columns "square_feet","weekly_price" and "monthly_price" have maximum null values. "square_feet" has 99.17% of the data as null value,"weekly_price" has 87.72% of the data as null value and "monthly_price"has 89.27% of the data as null value and there is not way to impute data for those missing values. Therefore, we should drop those columns from further analysis. Furthermore, 'neighbourhood_group_cleansed' is not necessary as our granulity level is zicode Thus, we will be dropping following columns: - square_feet - weekly_price - monthly_price - neighbourhood_group_cleansed

[16]:
```
df2_airbnb = df1_airbnb.
↪drop(columns=['neighbourhood_group_cleansed','square_feet', 'weekly_price',␣
↪'monthly_price'], axis = 1)
```

[17]:
```
df2_airbnb
```

[17]:
```
       bedrooms       city state zipcode    price  review_scores_location
0           1.0  Brooklyn     NY   11218  $149.00                    10.0
1           0.0   New York    NY   10018  $225.00                    10.0
2           1.0   New York    NY   10027  $150.00                     NaN
3           1.0   Brooklyn    NY   11238   $89.00                    10.0
4           NaN   New York    NY   10029   $80.00                     9.0
…           …         …   …      …       …                      …
48890       1.0   Brooklyn    NY   11216   $70.00                     NaN
48891       1.0  Brooklyn     NY   11206   $40.00                     NaN
48892       0.0   New York    NY   10027  $115.00                     NaN
48893       1.0   New York    NY   10036   $55.00                     NaN
48894       1.0   New York    NY   10019   $90.00                     NaN

[48895 rows x 6 columns]
```

After inspecting the state column, we can see that there are multiple patterns of NY and there are null values as well. Therefore, we will rectify the patterns of state column for NY and finter the dataframe for NY only as required by the investor

[18]:
```
df2_airbnb.state.unique()
```

[18]:
```
array(['NY', nan, 'Ny', 'ny', 'MP', 'CA', 'NJ', 'New York '], dtype=object)
```

[19]:
```
df2_airbnb.bedrooms.dtype
```

[19]:
```
dtype('float64')
```

[20]:
```
#changing the datatype to string
df2_airbnb.state = df2_airbnb.state.astype(str).copy()

#stripping excess whitespaces from the column
df2_airbnb['city'] = df2_airbnb['city'].str.strip()

#stripping excess whitespaces from the column
df2_airbnb['state'] = df2_airbnb['state'].str.strip()

#Makes all "NY"s uniform
df2_airbnb.state = df2_airbnb.state.replace(dict.fromkeys(['NY', 'ny',␣
 ↪'Ny','New York'],'NY'))

#filtering the dataframe to get NewYork data only
df2_airbnb = df2_airbnb[df2_airbnb['state']=='NY']

#filtering the dataframe to get 2 bedroom property only
df2_airbnb = df2_airbnb[df2_airbnb['bedrooms']==2]
```

[21]:
```
#unique values in state column
df2_airbnb.state.unique()

#this shows us that the datafram now has only NewYork data
```

[21]:
```
array(['NY'], dtype=object)
```

[22]:
```
#unique values in bedroom column
df2_airbnb.bedrooms.unique()
#this shows us that the datafram now has only 2 bedroom property
```

[22]:
```
array([2.])
```

[23]:
```
# no of rows and columns present in the airbnb data after filtering with state␣
 ↪= New York
ny_airbnb_rows = df2_airbnb.shape[0]
```

```
ny_airbnb_columns = df2_airbnb.shape[1]
print('There are {} rows airbnb listings dataset after filtering with New York.
 ↪'.format(ny_airbnb_rows))
```

```
There are 6496 rows airbnb listings dataset after filtering with New York.
```

[24]:
```
df2_airbnb.to_csv('df2airbnbnew.csv')
```

[25]:
```
#using the missing_value function saved under functions module
#missing_value functions takes in a dataframe and gives a table and plot of the␣
 ↪total nulls and percentage of nulls for each columns in the dataframe
functions.missing_value(df2_airbnb)
```

	null_percentage	total_null_values
bedrooms	0.00	0
city	0.14	9
state	0.00	0
zipcode	0.77	50
price	0.00	0
review_scores_location	21.34	1386

.

2.2.5 Correcting zipcode and price values and filtering the dataset with state = NewYork

- *There are miltiple entries for zipcode where there is more that 5 digits and also 1.06% of the zidcode data is null. Since, the percentage of null data is very insignificant we will just drop it and reassign the zipcode by fetching only the first 5 digit of the zipcode*
- *We will then remove the '$' and',' from the price data*
- *Also, 0.12% of the city data is null. Since, the percentage of null data is very insignificant we will just drop it*

[26]:
```
#copying the dataframe into a newone
df3_airbnb = df2_airbnb.copy()

#reassigning only the real zipcode (first5 digit of zipcode) to the zipcode␣
 ↪column of the data
df3_airbnb.zipcode = df2_airbnb.zipcode.str[:5].copy()
```

[27]:
```
# filter out rows in dataframe with column zipcode values NA/NAN
df3_airbnb = df3_airbnb[df1_airbnb.zipcode.notnull()]
```

```
2020-02-24 18:12:54,158 [17124] WARNING  py.warnings:110: [JupyterRequire]
C:\Users\Consultant\Anaconda3\lib\site-packages\ipykernel_launcher.py:2:
UserWarning:
```

```
Boolean Series key will be reindexed to match DataFrame index.
```

```
[28]: #inspecting the unique values in zipcode to see if there is any null values or␣
      ↪irrelavant values
      df3_airbnb.zipcode.unique()
```

```
[28]: array(['10029', '11221', '11206', '10001', '10162', '11215', '10075',
             '11211', '10031', '10002', '11217', '11231', '11233', '10009',
             '10023', '11201', '11238', '11249', '10027', '10039', '11385',
             '10013', '10011', '11222', '11216', '10032', '11205', '10003',
             '10012', '10026', '10025', '10128', '10014', '11104', '10022',
             '11225', '11101', '10038', '11213', '11106', '10016', '10036',
             '10463', '10065', '10024', '10455', '10034', '11237', '10469',
             '11235', '10314', '10452', '11103', '11220', '10004', '11226',
             '10282', '10019', '11377', '10033', '10021', '11230', '11214',
             '10037', '10010', '11418', '10030', '10005', '10035', '11218',
             '11105', '11372', '11207', '10028', '10017', '11208', '10040',
             '11412', '11367', '11374', '11209', '11693', '10305', '11109',
             '10304', '11102', '11212', '11232', '11204', '10451', '11369',
             '11234', '10473', '10301', '10044', '10018', '11236', '11203',
             '11373', '10475', '11417', '10459', '10007', '11375', '10280',
             '10069', '11223', '11433', '10454', '11692', '11365', '10308',
             '11210', '11426', '11423', '11434', '11228', '10006', '11435',
             '11379', '11370', '11378', '11368', '10456', '11691', '10303',
             '10460', '11422', '11355', '11416', '11229', '10457', '11224',
             '11358', '10467', '10310', '11414', '11356', '11411', '10307',
             '11436', '11694', '10306', '11429', '11354', '11357', '11413',
             '10464', '10458', '10462', '11219', '10468', '11421', '11361',
             '10466', '11432', '11559', '10461', '10270', '11419', '10281',
             '10472', '11420', '10470', '10453', '10309', '11003', '11428',
             '10471'], dtype=object)
```

```
[29]: #Removing the "$" and "," signs from the price column and changing the datatype␣
      ↪to float
      df3_airbnb['price'] = df3_airbnb['price'].replace( '[\$,)]','', regex=True ).
      ↪astype(float).copy()
```

```
[30]: # filter out rows in dataframe with column city values NA/NAN
      df3_airbnb = df3_airbnb[df1_airbnb.city.notnull()]
```

```
2020-02-24 18:12:55,025 [17124] WARNING  py.warnings:110: [JupyterRequire]
C:\Users\Consultant\Anaconda3\lib\site-packages\ipykernel_launcher.py:2:
UserWarning:

Boolean Series key will be reindexed to match DataFrame index.
```

[31]:
```
df3_airbnb.head(20)
```

[31]:

	bedrooms	city	state	zipcode	price	review_scores_location
19	2.0	New York	NY	10029	190.0	NaN
48	2.0	Brooklyn	NY	11221	115.0	9.0
52	2.0	Brooklyn	NY	11206	228.0	9.0
61	2.0	New York	NY	10001	375.0	10.0
62	2.0	New York	NY	10162	250.0	9.0
66	2.0	Brooklyn	NY	11215	225.0	10.0
76	2.0	New York	NY	10075	200.0	10.0
80	2.0	Brooklyn	NY	11211	145.0	9.0
81	2.0	New York	NY	10031	110.0	8.0
82	2.0	New York	NY	10002	285.0	10.0
93	2.0	Brooklyn	NY	11217	250.0	10.0
101	2.0	Brooklyn	NY	11231	175.0	9.0
106	2.0	Brooklyn	NY	11233	125.0	9.0
114	2.0	New York	NY	10009	350.0	9.0
117	2.0	New York	NY	10023	235.0	10.0
121	2.0	Brooklyn	NY	11215	400.0	10.0
138	2.0	Brooklyn	NY	11233	125.0	9.0
142	2.0	Brooklyn	NY	11201	140.0	10.0
147	2.0	New York	NY	10009	195.0	9.0
153	2.0	Brooklyn	NY	11238	115.0	7.0

[ ]:
```
```

[32]:
```
#check the data
#using the missing_value function saved under functions module
#missing_value functions takes in a dataframe and gives a table and plot of the
↪total nulls and percentage of nulls for each columns in the dataframe
functions.missing_value(df3_airbnb)
```

	null_percentage	total_null_values
bedrooms	0.00	0
city	0.00	0
state	0.00	0
zipcode	0.00	0
price	0.00	0
review_scores_location	21.33	1373

2.2.6 Imputing the missing data

After dealing with most of the features, we have one column i.e "review_scores_location" with 22.64% of its data as missing value. This features seems important for analysis and can be easily imputed by using regressor based on zip code and price. We will use MICE to impute the missing *zipcodes*

[33]:
```
#Mice only take numerical values of preparing a dataset for imputation
review=df3_airbnb[['zipcode', 'price','review_scores_location']]
```

[34]:
```
review.head(3)
```

[34]:

	zipcode	price	review_scores_location
19	10029	190.0	NaN
48	11221	115.0	9.0
52	11206	228.0	9.0

[35]:
```
# Import IterativeImputer from fancyimpute
from fancyimpute import IterativeImputer

# Copy diabetes to diabetes_mice_imputed
review_mice_imputed = review.copy(deep=True)

# Initialize IterativeImputer
mice_imputer = IterativeImputer()

# Impute using fit_tranform on diabetes
review_mice_imputed.iloc[:, :] = mice_imputer.fit_transform(review)

#rounding off the imputed data
review_mice_imputed.review_scores_location = round(review_mice_imputed.
↪review_scores_location,0)

#view the data
review_mice_imputed.head(3)
```

```
Using TensorFlow backend.
```

[35]:

	zipcode	price	review_scores_location
19	10029.0	190.0	10.0
48	11221.0	115.0	9.0
52	11206.0	228.0	9.0

[36]:
```
#replacing the null values with the imputed value in the dataset
df3_airbnb.review_scores_location = review_mice_imputed.review_scores_location.
↪copy()
```

[37]:
```
df3_airbnb
```

[37]:

	bedrooms	city	state	zipcode	price	review_scores_location
19	2.0	New York	NY	10029	190.0	10.0
48	2.0	Brooklyn	NY	11221	115.0	9.0
52	2.0	Brooklyn	NY	11206	228.0	9.0

```
61           2.0  New York    NY   10001  375.0                    10.0
62           2.0  New York    NY   10162  250.0                     9.0
…          …       …   …      …     …                    …
48804        2.0  New York    NY   10004   99.0                    10.0
48806        2.0  Brooklyn    NY   11229  140.0                     9.0
48813        2.0    Queens    NY   11691   80.0                     9.0
48860        2.0  New York    NY   10044  145.0                    10.0
48873        2.0  Brooklyn    NY   11234  170.0                     9.0

[6438 rows x 6 columns]
```

[38]:
```
#df3_airbnb.to_csv('df3airbnbnew.csv')
```

.

2.2.7 Dealing with extremely large and zero values in Prices

After inspecting the price column, there were many rows with 0 as price so we will filter the dataset such that it will contain only prices greater than 0

[39]:
```
df3_airbnb = df3_airbnb[df3_airbnb['price']>0].copy()
```

[40]:
```
df3_airbnb
```

[40]:
```
      bedrooms      city state zipcode  price  review_scores_location
19           2.0  New York    NY   10029  190.0                    10.0
48           2.0  Brooklyn    NY   11221  115.0                     9.0
52           2.0  Brooklyn    NY   11206  228.0                     9.0
61           2.0  New York    NY   10001  375.0                    10.0
62           2.0  New York    NY   10162  250.0                     9.0
…          …       …   …      …     …                    …
48804        2.0  New York    NY   10004   99.0                    10.0
48806        2.0  Brooklyn    NY   11229  140.0                     9.0
48813        2.0    Queens    NY   11691   80.0                     9.0
48860        2.0  New York    NY   10044  145.0                    10.0
48873        2.0  Brooklyn    NY   11234  170.0                     9.0

[6437 rows x 6 columns]
```

[ ]:
```

```

2.2.8 Checking for other outliers

[41]:
```
# Percentile of the flattened array
print("\n99th Percentile of price: ",
      np.percentile(df3_airbnb.price, 99))
```

```
99th Percentile of price:  968.1600000000062
```

[42]:
```
import plotly.express as px

fig = px.box(df3_airbnb, y="price", height = 700)
fig.show()
```

After inspecting the price column and plotting the box plot chart it was clear that there are many outliers. After calculating the 99th percentile of the price column we found out that 99th percentile of prices fall below 968. So, I have set the maximum threshold for the price to be 1000 as people will rarely buy any houses costing more that 1000 daily.

[43]:
```
#filtering the data for daily price less than or eual to $1000
df3_airbnb = df3_airbnb[df3_airbnb['price']<=1000].copy()
```

2.2.9 Box plot

[44]:
```
#the following code if for generating box plot with plotly

import plotly.express as px

fig = px.box(df3_airbnb, y="price", height = 700)
fig.show()
```

2.2.10 Grouping the data by zipcode

[45]:
```
#count the number of unique zipcodes in our data and number of records
count = len(df3_airbnb.zipcode.unique())
print('There are {} unique zipcodes.'.format(count))

rows = df3_airbnb.shape[0]
print('There are {} records.'.format(rows))
```

```
There are 168 unique zipcodes.
There are 6390 records.
```

[46]:
```
## Grouping the data
airbnb_grouped = df3_airbnb.groupby('zipcode').mean()
```

[47]:
```
airbnb_grouped
```

[47]:

zipcode	bedrooms	price	review_scores_location
10001	2.0	378.071429	9.857143
10002	2.0	265.219653	9.780347
10003	2.0	293.296296	9.866667

```
10004          2.0  322.714286             9.857143
10005          2.0  357.195122             9.853659
…            …        …                  …
11559          2.0  250.000000             9.000000
11691          2.0  227.000000             9.000000
11692          2.0  171.250000             9.500000
11693          2.0  171.545455             9.454545
11694          2.0  194.750000             9.750000

[168 rows x 3 columns]
```

.

2.3 Zillow Data

[48]:
```
zillow
```

[48]:
```
      RegionID  RegionName             City State            Metro CountyName  \
0        61639       10025         New York    NY         New York   New York
1        84654       60657          Chicago    IL          Chicago       Cook
2        61637       10023         New York    NY         New York   New York
3        84616       60614          Chicago    IL          Chicago       Cook
4        93144       79936          El Paso    TX          El Paso    El Paso
…          …           …              …   …               …          …
8941     93454       80532            Drake    CO     Fort Collins    Larimer
8942     62556       12429        Port Ewen    NY         Kingston     Ulster
8943     99032       97028      Rhododendron    OR         Portland  Clackamas
8944     58333        1338   Shelburne Falls    MA  Greenfield Town   Franklin
8945     59107        3293        Woodstock    NH        Claremont    Grafton

      SizeRank   1996-04   1996-05   1996-06  …  2016-09  2016-10  2016-11  \
0            1       NaN       NaN       NaN  …  1374400  1364100  1366300
1            2  167700.0  166400.0  166700.0  …   368600   370200   372300
2            3       NaN       NaN       NaN  …  1993500  1980700  1960900
3            4  195800.0  193500.0  192600.0  …   398900   401200   403200
4            5   59100.0   60500.0   60900.0  …    82400    82300    82400
…          …        …        …        …     …     …        …
8941      8942       NaN       NaN       NaN  …   270800   272200   274700
8942      8943   64500.0   64000.0   63400.0  …   144500   144600   145600
8943      8944       NaN       NaN       NaN  …   318200   315000   312300
8944      8945   91400.0   91000.0   90600.0  …   185700   184600   184800
8945      8946   71800.0   71800.0   73100.0  …   163500   166300   168900

        2016-12  2017-01  2017-02  2017-03  2017-04  2017-05  2017-06
0     1354800.0  1327500  1317300  1333700  1352100  1390000  1431000
1      375300.0   378700   381400   381800   382100   383300   385100
2     1951300.0  1937800  1929800  1955000  2022400  2095000  2142300
```

```
3       405700.0    408300    408800    408000    410100    412200    412200
4        82300.0     82500     83200     83900     84100     83900     83700
…          …        …        …        …        …        …        …
8941    281300.0    286200    285300    284100    284800    285800    287500
8942    146400.0    146600    147100    149100    151700    153300    153800
8943    308800.0    304700    302400    302300    303300    307400    312600
8944    188600.0    193000    195800    197600    198300    198300    198500
8945    171200.0    172200    171100    168300    165900    165500    165800

[8946 rows x 262 columns]
```

[49]:
```
# no of rows and columns present in the zillow data

zillow_rows = zillow.shape[0]
zillow_columns = zillow.shape[1]
print('There are {} rows and {} columns in the zillow dataset'.
 ↪format(zillow_rows,zillow_columns))
```

```
There are 8946 rows and 262 columns in the zillow dataset
```

.

We are not using data before 2012 in our model for 2 reasons: First, We have lots of null values for these time frame. Second, because of 2008-2011 recession, the prices of real estate properties has declined. If we use this data, It would potentially mislead our model against predicting correctly.

We are considering current year as 2017 and forecasting for July 2019 as our revenue prices were scraped on July 2019 and it would be better to have data from simialr time period for accurate analysis. By using Facebook Prophet model, we will try to predict the value of the properties in year 2018 in order to calculate the profit percentage.

[50]:
```
df = zillow
```

2.3.1 Filtering the data to get New York data

[51]:
```
# zillow_city_filter function takes in  three arguments and returns dataframe␣
 ↪with values filterred based on the arguements provided.
# The first argument df is the dataframe from which the user intends to filter␣
 ↪values
# The second argument State is the name of the state State from which the user␣
 ↪intends to filter values
# The third argument Metro is the Metro name from which the user intends to␣
 ↪filter values

df = zillow
State = "NY"
Metro = "New York"
df1 = functions.zillow_city_filter(df, State, Metro)
```

2.3.2 Preprocessing the data to get median cost values zipcodes and time periods

[52]:
```
# zillow_preprocess function takes in  two arguments and returns dataframe with␣
↪required values for time series forecasting.
# The first argument df is the dataframe which needs to be proprocessed.
# The second argument n is the number of recent months required to be present␣
↪in the dataframe.

#number of months if denoted by n
n = 61

df2 = functions.zillow_preprocess(df1, 61)
```

[53]:
```
df2
```

[53]:
```
      zipcode    2012-06    2012-07    2012-08    2012-09    2012-10  \
0       10025   914000.0   921100.0   923300.0   917300.0   915000.0
2       10023  1376700.0  1378200.0  1378700.0  1375900.0  1366700.0
13      10128  1045000.0  1043400.0  1050300.0  1050500.0  1050700.0
14      10011  1524500.0  1546500.0  1574800.0  1599600.0  1622500.0
20      10003  1364200.0  1376600.0  1384200.0  1387900.0  1404200.0
…         …         …          …          …          …          …
8475    12780   135400.0   134400.0   133900.0   133700.0   136000.0
8501    12583   166800.0   168000.0   168500.0   167800.0   168400.0
8506    12581   212100.0   211800.0   212600.0   213700.0   214500.0
8527    10537   171800.0   170700.0   170800.0   171400.0   171400.0
8749    12729   106300.0   103500.0   102600.0   102400.0   101300.0

         2012-11    2012-12    2013-01    2013-02  …  2016-09  2016-10  \
0       922800.0   929100.0   937700.0   955700.0  …  1374400  1364100
2      1365500.0  1382200.0  1404700.0  1428000.0  …  1993500  1980700
13     1059700.0  1079600.0  1091600.0  1106100.0  …  1526000  1523700
14     1639000.0  1656100.0  1684600.0  1703000.0  …  2354000  2355500
20     1419200.0  1425700.0  1435300.0  1460300.0  …  1932800  1930400
…            …          …          …          …  …      …        …
8475    137500.0   138000.0   139300.0   139400.0  …   123700   123700
8501    170500.0   171600.0   171200.0   170500.0  …   184400   186100
8506    215300.0   215600.0   214600.0   214200.0  …   211000   212400
8527    170500.0   170700.0   171800.0   172300.0  …   176600   177300
8749    100700.0   101000.0   101700.0   101500.0  …   106900   107900

       2016-11    2016-12  2017-01  2017-02  2017-03  2017-04  2017-05  2017-06
0      1366300  1354800.0  1327500  1317300  1333700  1352100  1390000  1431000
2      1960900  1951300.0  1937800  1929800  1955000  2022400  2095000  2142300
13     1527200  1541600.0  1557800  1582900  1598900  1646100  1720500  1787100
14     2352200  2332100.0  2313300  2319600  2342100  2365900  2419700  2480400
20     1937500  1935100.0  1915700  1916500  1965700  2045300  2109100  2147000
```

```
…      …       …        …       …       …       …       …       …
8475   124400  125100.0 124200  122500  121600  122400  125100  128300
8501   189100  190200.0 189600  189300  190500  193900  197100  198200
8506   213700  214500.0 215200  216200  217700  220700  223300  224900
8527   178700  179600.0 179200  179000  180600  182500  183600  184200
8749   109200  109200.0 107700  106500  106700  106800  106400  105800

[156 rows x 62 columns]
```

.

.

2.3.3 Forecasting property cost using Facebook Prophet

[54]:
```
# zillow_prophet function takes in  a preprocessed dataframe and returns the␣
↪dataframe with forecasted current value of the property as "currentPrice".
# Facebook prophet is used for the time series forecasting.
# Facebook prophet is various robust and can handle missing values as well and␣
↪perform very well with large scale of data as well.
```

[55]:
```
#df_currentPrice = functions.zillow_prophet(df2, 26)
```

[57]:
```
df_currentPrice
```

[57]:
```
      zipcode    2012-06    2012-07    2012-08    2012-09    2012-10  \
0       10025   914000.0   921100.0   923300.0   917300.0   915000.0
2       10023  1376700.0  1378200.0  1378700.0  1375900.0  1366700.0
13      10128  1045000.0  1043400.0  1050300.0  1050500.0  1050700.0
14      10011  1524500.0  1546500.0  1574800.0  1599600.0  1622500.0
20      10003  1364200.0  1376600.0  1384200.0  1387900.0  1404200.0
…         …        …          …          …          …          …
8475    12780   135400.0   134400.0   133900.0   133700.0   136000.0
8501    12583   166800.0   168000.0   168500.0   167800.0   168400.0
8506    12581   212100.0   211800.0   212600.0   213700.0   214500.0
8527    10537   171800.0   170700.0   170800.0   171400.0   171400.0
8749    12729   106300.0   103500.0   102600.0   102400.0   101300.0

        2012-11    2012-12    2013-01    2013-02  …  2016-10  2016-11  \
0      922800.0   929100.0   937700.0   955700.0  …  1364100  1366300
2     1365500.0  1382200.0  1404700.0  1428000.0  …  1980700  1960900
13    1059700.0  1079600.0  1091600.0  1106100.0  …  1523700  1527200
14    1639000.0  1656100.0  1684600.0  1703000.0  …  2355500  2352200
20    1419200.0  1425700.0  1435300.0  1460300.0  …  1930400  1937500
…          …          …          …          …   …    …        …
8475   137500.0   138000.0   139300.0   139400.0  …   123700   124400
8501   170500.0   171600.0   171200.0   170500.0  …   186100   189100
8506   215300.0   215600.0   214600.0   214200.0  …   212400   213700
```

```
8527   170500.0   170700.0   171800.0   172300.0  …   177300   178700
8749   100700.0   101000.0   101700.0   101500.0  …   107900   109200

        2016-12  2017-01  2017-02  2017-03  2017-04  2017-05  2017-06  \
0     1354800.0  1327500  1317300  1333700  1352100  1390000  1431000
2     1951300.0  1937800  1929800  1955000  2022400  2095000  2142300
13    1541600.0  1557800  1582900  1598900  1646100  1720500  1787100
14    2332100.0  2313300  2319600  2342100  2365900  2419700  2480400
20    1935100.0  1915700  1916500  1965700  2045300  2109100  2147000
…           …        …       …        …        …        …       …
8475   125100.0   124200   122500   121600   122400   125100   128300
8501   190200.0   189600   189300   190500   193900   197100   198200
8506   214500.0   215200   216200   217700   220700   223300   224900
8527   179600.0   179200   179000   180600   182500   183600   184200
8749   109200.0   107700   106500   106700   106800   106400   105800

      currentPrice
0          1438622
2          2411269
13         1994237
14         2813158
20         2289645
…              …
8475        128275
8501        234412
8506        245348
8527        206023
8749        100688

[156 rows x 63 columns]
```

[58]:
```
#df_currentPrice.to_csv('crrntprce.csv')
```

3 Analysis

3.1 Merge the datasets

[59]:
```
airbnb_grouped
```

[59]:
```
         bedrooms       price  review_scores_location
zipcode
10001         2.0  378.071429                9.857143
10002         2.0  265.219653                9.780347
10003         2.0  293.296296                9.866667
10004         2.0  322.714286                9.857143
10005         2.0  357.195122                9.853659
```

```
…              …          …                         …
11559          2.0   250.000000                9.000000
11691          2.0   227.000000                9.000000
11692          2.0   171.250000                9.500000
11693          2.0   171.545455                9.454545
11694          2.0   194.750000                9.750000

[168 rows x 3 columns]
```

[ ]:

```
[60]: #getting a dataframe with zipcode and property cost
cost =  df_currentPrice[['zipcode', 'currentPrice' ]]
cost = cost.rename(columns={'currentPrice': 'Property_Cost'})
cost
```

```
[60]:      zipcode  Property_Cost
0       10025        1438622
2       10023        2411269
13      10128        1994237
14      10011        2813158
20      10003        2289645
…       …            …
8475    12780         128275
8501    12583         234412
8506    12581         245348
8527    10537         206023
8749    12729         100688

[156 rows x 2 columns]
```

```
[61]: ## Based on our asssumption of 75% occupancy rate, we are calculating occupied␣
 ↪days
occupancy_rate = 0.75
occupancy_days = occupancy_rate*365
```

3.1.1 Calculating Yearly revenue

```
[62]: revenue = airbnb_grouped.copy()
revenue = revenue.reset_index()
revenue=revenue[['zipcode','price','review_scores_location']]
revenue['yearly_revenue']= occupancy_days*revenue['price']
revenue
```

[62]:

```
     zipcode       price  review_scores_location  yearly_revenue
0      10001  378.071429                9.857143   103497.053571
1      10002  265.219653                9.780347    72603.880058
2      10003  293.296296                9.866667    80289.861111
3      10004  322.714286                9.857143    88343.035714
4      10005  357.195122                9.853659    97782.164634
..       …           …                      …             …
163    11559  250.000000                9.000000    68437.500000
164    11691  227.000000                9.000000    62141.250000
165    11692  171.250000                9.500000    46879.687500
166    11693  171.545455                9.454545    46960.568182
167    11694  194.750000                9.750000    53312.812500

[168 rows x 4 columns]
```

3.2 Merging Revenue and Cost dataframe on zipcode

[63]:

```python
#merging two dataframe on a common key
df_merge = pd.merge(cost, revenue, on='zipcode')
```

[64]:

```python
#calculating breakeven period
df_merge['breakeven_period'] = df_merge['Property_Cost']/
 ↪df_merge['yearly_revenue']
```

[65]:

```python
#merged dataset
df_merge
```

[65]:

```
    zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
0     10025        1438622  253.134454                9.840336    69295.556723
1     10023        2411269  276.402597                9.961039    75665.211039
2     10128        1994237  226.156250                9.765625    61910.273438
3     10011        2813158  348.428571                9.952381    95382.321429
4     10003        2289645  293.296296                9.866667    80289.861111
5     11201        1565379  243.682353                9.823529    66708.044118
6     11234         565206  135.111111                9.222222    36986.666667
7     10314         437976   73.000000                9.500000    19983.750000
8     11215        1302932  181.608466                9.671958    49715.317460
9     10028        2588357  273.794521                9.808219    74951.250000
10    10021        2073139  208.884615                9.423077    57182.163462
11    10014        2795098  315.181818                9.988636    86281.022727
12    10036        1668202  330.636986                9.828767    90511.875000
13    11434         493049  136.875000                9.250000    37469.531250
14    10306         425597  117.500000                9.500000    32165.625000
15    10022        2254331  287.823529                9.926471    78791.691176
16    11217        1406859  205.425000                9.750000    56235.093750
17    10013        3098203  363.029703                9.900990    99379.381188
18    11231        1414249  198.500000                9.597826    54339.375000
```

```
19    10304         422999   93.333333                 8.666667    25550.000000
20    10305         563792  132.000000                 9.833333    36135.000000
21    11003         432277  180.000000                10.000000    49275.000000
22    10309         471631   85.000000                10.000000    23268.750000
23    10308         529025  109.500000                10.000000    29975.625000
24    10303         430472  104.000000                 9.250000    28470.000000

    breakeven_period
0          20.760667
1          31.867604
2          32.211730
3          29.493495
4          28.517237
5          23.466120
6          15.281345
7          21.916607
8          26.207858
9          34.533874
10         36.254994
11         32.395281
12         18.430753
13         13.158665
14         13.231423
15         28.611278
16         25.017456
17         31.175511
18         26.026229
19         16.555734
20         15.602380
21          8.772745
22         20.268858
23         17.648506
24         15.120197
```

3.3 Dataframe sorted by breakeven period, cost and revenue

3.3.1 sorted by Breakeven Period

[66]:
```
#sorted by breakeven period
sort_breakeven = df_merge.sort_values(by='breakeven_period', ascending=True)
sort_breakeven
```

[66]:
```
   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
21    11003         432277  180.000000                10.000000    49275.000000
13    11434         493049  136.875000                 9.250000    37469.531250
14    10306         425597  117.500000                 9.500000    32165.625000
24    10303         430472  104.000000                 9.250000    28470.000000
```

6	11234	565206	135.111111	9.222222	36986.666667
20	10305	563792	132.000000	9.833333	36135.000000
19	10304	422999	93.333333	8.666667	25550.000000
23	10308	529025	109.500000	10.000000	29975.625000
12	10036	1668202	330.636986	9.828767	90511.875000
22	10309	471631	85.000000	10.000000	23268.750000
0	10025	1438622	253.134454	9.840336	69295.556723
7	10314	437976	73.000000	9.500000	19983.750000
5	11201	1565379	243.682353	9.823529	66708.044118
16	11217	1406859	205.425000	9.750000	56235.093750
18	11231	1414249	198.500000	9.597826	54339.375000
8	11215	1302932	181.608466	9.671958	49715.317460
4	10003	2289645	293.296296	9.866667	80289.861111
15	10022	2254331	287.823529	9.926471	78791.691176
3	10011	2813158	348.428571	9.952381	95382.321429
17	10013	3098203	363.029703	9.900990	99379.381188
1	10023	2411269	276.402597	9.961039	75665.211039
2	10128	1994237	226.156250	9.765625	61910.273438
11	10014	2795098	315.181818	9.988636	86281.022727
9	10028	2588357	273.794521	9.808219	74951.250000
10	10021	2073139	208.884615	9.423077	57182.163462

	breakeven_period
21	8.772745
13	13.158665
14	13.231423
24	15.120197
6	15.281345
20	15.602380
19	16.555734
23	17.648506
12	18.430753
22	20.268858
0	20.760667
7	21.916607
5	23.466120
16	25.017456
18	26.026229
8	26.207858
4	28.517237
15	28.611278
3	29.493495
17	31.175511
1	31.867604
2	32.211730
11	32.395281
9	34.533874

```
10          36.254994
```

3.3.2 sorted by Yearly Revenue

[67]:
```
#sorted by yearly revenue
sort_revenue = df_merge.sort_values(by='yearly_revenue', ascending=False)
sort_revenue
```

[67]:

	zipcode	Property_Cost	price	review_scores_location	yearly_revenue	\
17	10013	3098203	363.029703	9.900990	99379.381188	
3	10011	2813158	348.428571	9.952381	95382.321429	
12	10036	1668202	330.636986	9.828767	90511.875000	
11	10014	2795098	315.181818	9.988636	86281.022727	
4	10003	2289645	293.296296	9.866667	80289.861111	
15	10022	2254331	287.823529	9.926471	78791.691176	
1	10023	2411269	276.402597	9.961039	75665.211039	
9	10028	2588357	273.794521	9.808219	74951.250000	
0	10025	1438622	253.134454	9.840336	69295.556723	
5	11201	1565379	243.682353	9.823529	66708.044118	
2	10128	1994237	226.156250	9.765625	61910.273438	
10	10021	2073139	208.884615	9.423077	57182.163462	
16	11217	1406859	205.425000	9.750000	56235.093750	
18	11231	1414249	198.500000	9.597826	54339.375000	
8	11215	1302932	181.608466	9.671958	49715.317460	
21	11003	432277	180.000000	10.000000	49275.000000	
13	11434	493049	136.875000	9.250000	37469.531250	
6	11234	565206	135.111111	9.222222	36986.666667	
20	10305	563792	132.000000	9.833333	36135.000000	
14	10306	425597	117.500000	9.500000	32165.625000	
23	10308	529025	109.500000	10.000000	29975.625000	
24	10303	430472	104.000000	9.250000	28470.000000	
19	10304	422999	93.333333	8.666667	25550.000000	
22	10309	471631	85.000000	10.000000	23268.750000	
7	10314	437976	73.000000	9.500000	19983.750000	

	breakeven_period
17	31.175511
3	29.493495
12	18.430753
11	32.395281
4	28.517237
15	28.611278
1	31.867604
9	34.533874
0	20.760667
5	23.466120
2	32.211730

```
10          36.254994
16          25.017456
18          26.026229
8           26.207858
21           8.772745
13          13.158665
6           15.281345
20          15.602380
14          13.231423
23          17.648506
24          15.120197
19          16.555734
22          20.268858
7           21.916607
```

3.3.3 sorted by Property Cost

[68]:
```python
#sorted by property cost
sort_cost = df_merge.sort_values(by='Property_Cost', ascending=False)
sort_cost
```

[68]:

	zipcode	Property_Cost	price	review_scores_location	yearly_revenue \
17	10013	3098203	363.029703	9.900990	99379.381188
3	10011	2813158	348.428571	9.952381	95382.321429
11	10014	2795098	315.181818	9.988636	86281.022727
9	10028	2588357	273.794521	9.808219	74951.250000
1	10023	2411269	276.402597	9.961039	75665.211039
4	10003	2289645	293.296296	9.866667	80289.861111
15	10022	2254331	287.823529	9.926471	78791.691176
10	10021	2073139	208.884615	9.423077	57182.163462
2	10128	1994237	226.156250	9.765625	61910.273438
12	10036	1668202	330.636986	9.828767	90511.875000
5	11201	1565379	243.682353	9.823529	66708.044118
0	10025	1438622	253.134454	9.840336	69295.556723
18	11231	1414249	198.500000	9.597826	54339.375000
16	11217	1406859	205.425000	9.750000	56235.093750
8	11215	1302932	181.608466	9.671958	49715.317460
6	11234	565206	135.111111	9.222222	36986.666667
20	10305	563792	132.000000	9.833333	36135.000000
23	10308	529025	109.500000	10.000000	29975.625000
13	11434	493049	136.875000	9.250000	37469.531250
22	10309	471631	85.000000	10.000000	23268.750000
7	10314	437976	73.000000	9.500000	19983.750000
21	11003	432277	180.000000	10.000000	49275.000000
24	10303	430472	104.000000	9.250000	28470.000000
14	10306	425597	117.500000	9.500000	32165.625000
19	10304	422999	93.333333	8.666667	25550.000000

```
    breakeven_period
17         31.175511
3          29.493495
11         32.395281
9          34.533874
1          31.867604
4          28.517237
15         28.611278
10         36.254994
2          32.211730
12         18.430753
5          23.466120
0          20.760667
18         26.026229
16         25.017456
8          26.207858
6          15.281345
20         15.602380
23         17.648506
13         13.158665
22         20.268858
7          21.916607
21          8.772745
24         15.120197
14         13.231423
19         16.555734
```

3.4 Top 10 zipcodes based on Cost and Revenue

[69]:

```python
sort_cost_top = sort_cost.head(10)
print("Top 10 zipcodes with cost: ")
display(sort_cost_top)

sort_breakeven_top = sort_breakeven.head(10)
print("Top 10 zipcodes with fastest breakeven period: ")
display(sort_breakeven_top)



sort_revenue_top = sort_revenue.head(10)
print("Top 10 zipcodes with highest revenue: ")
display(sort_revenue_top)
```

```
Top 10 zipcodes with cost:

   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
```

```
17   10013         3098203  363.029703                9.900990    99379.381188
3    10011         2813158  348.428571                9.952381    95382.321429
11   10014         2795098  315.181818                9.988636    86281.022727
9    10028         2588357  273.794521                9.808219    74951.250000
1    10023         2411269  276.402597                9.961039    75665.211039
4    10003         2289645  293.296296                9.866667    80289.861111
15   10022         2254331  287.823529                9.926471    78791.691176
10   10021         2073139  208.884615                9.423077    57182.163462
2    10128         1994237  226.156250                9.765625    61910.273438
12   10036         1668202  330.636986                9.828767    90511.875000

    breakeven_period
17         31.175511
3          29.493495
11         32.395281
9          34.533874
1          31.867604
4          28.517237
15         28.611278
10         36.254994
2          32.211730
12         18.430753


Top 10 zipcodes with fastest breakeven period:
   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
21   11003          432277  180.000000               10.000000    49275.000000
13   11434          493049  136.875000                9.250000    37469.531250
14   10306          425597  117.500000                9.500000    32165.625000
24   10303          430472  104.000000                9.250000    28470.000000
6    11234          565206  135.111111                9.222222    36986.666667
20   10305          563792  132.000000                9.833333    36135.000000
19   10304          422999   93.333333                8.666667    25550.000000
23   10308          529025  109.500000               10.000000    29975.625000
12   10036         1668202  330.636986                9.828767    90511.875000
22   10309          471631   85.000000               10.000000    23268.750000

    breakeven_period
21          8.772745
13         13.158665
14         13.231423
24         15.120197
6          15.281345
20         15.602380
19         16.555734
23         17.648506
12         18.430753
22         20.268858
```

```
Top 10 zipcodes with highest revenue:

   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
17   10013        3098203  363.029703                9.900990    99379.381188
3    10011        2813158  348.428571                9.952381    95382.321429
12   10036        1668202  330.636986                9.828767    90511.875000
11   10014        2795098  315.181818                9.988636    86281.022727
4    10003        2289645  293.296296                9.866667    80289.861111
15   10022        2254331  287.823529                9.926471    78791.691176
1    10023        2411269  276.402597                9.961039    75665.211039
9    10028        2588357  273.794521                9.808219    74951.250000
0    10025        1438622  253.134454                9.840336    69295.556723
5    11201        1565379  243.682353                9.823529    66708.044118

    breakeven_period
17         31.175511
3          29.493495
12         18.430753
11         32.395281
4          28.517237
15         28.611278
1          31.867604
9          34.533874
0          20.760667
5          23.466120
```

3.5 Visualizations

3.5.1 Property Cost

The below table and graph identifies the property cost for different zipcodes. Zipcodes 10013, 10011, 10014 cost the highest with 3 Million, 2.8 million and 2.79 million respectively whereas, zipcode 10036, 10128 and 10021 have the lowest cost with values less than 500,000.

[70]:
```
sort_cost_top = sort_cost.head(10)
print("Top 10 zipcodes with cost: ")
display(sort_cost_top)

sort_cost_bot= sort_cost.tail(10)
print("zipcodes with  lowest cost: ")
display(sort_cost_bot)
```

```
Top 10 zipcodes with cost:

   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
17   10013        3098203  363.029703                9.900990    99379.381188
3    10011        2813158  348.428571                9.952381    95382.321429
```

11	10014	2795098	315.181818	9.988636	86281.022727
9	10028	2588357	273.794521	9.808219	74951.250000
1	10023	2411269	276.402597	9.961039	75665.211039
4	10003	2289645	293.296296	9.866667	80289.861111
15	10022	2254331	287.823529	9.926471	78791.691176
10	10021	2073139	208.884615	9.423077	57182.163462
2	10128	1994237	226.156250	9.765625	61910.273438
12	10036	1668202	330.636986	9.828767	90511.875000

	breakeven_period
17	31.175511
3	29.493495
11	32.395281
9	34.533874
1	31.867604
4	28.517237
15	28.611278
10	36.254994
2	32.211730
12	18.430753

zipcodes with lowest cost:

	zipcode	Property_Cost	price	review_scores_location	yearly_revenue \
6	11234	565206	135.111111	9.222222	36986.666667
20	10305	563792	132.000000	9.833333	36135.000000
23	10308	529025	109.500000	10.000000	29975.625000
13	11434	493049	136.875000	9.250000	37469.531250
22	10309	471631	85.000000	10.000000	23268.750000
7	10314	437976	73.000000	9.500000	19983.750000
21	11003	432277	180.000000	10.000000	49275.000000
24	10303	430472	104.000000	9.250000	28470.000000
14	10306	425597	117.500000	9.500000	32165.625000
19	10304	422999	93.333333	8.666667	25550.000000

	breakeven_period
6	15.281345
20	15.602380
23	17.648506
13	13.158665
22	20.268858
7	21.916607
21	8.772745
24	15.120197
14	13.231423
19	16.555734

[71]:
```
import plotly.express as px

fig = px.scatter(sort_revenue, x="Property_Cost", y="zipcode", color="zipcode",
                 size='Property_Cost', hover_data=['Property_Cost'])
fig.show()
```

3.5.2 Yearly revenue

The below tables and graph identifies the yearly revenue for different zipcodes. Zipcodes 10013, 10011, 10036 yield the highest yearly revenue with 99379 dollars, 95382 dollars and 90511 dollars respectively whereas, zipcode 10314, 10309 and 10304 have the yearly revenue with values 19983 dollars, 23268 dollars and 25550 dollars respectively.

[72]:
```
sort_revenue_top = sort_revenue.head(10)
print("Top 10 zipcodes with highest revenue: ")
display(sort_revenue_top)

sort_revenue_bot = sort_revenue.tail(10)
print("Top 10 zipcodes with lowest revenue: ")
display(sort_revenue_bot)
```

```
Top 10 zipcodes with highest revenue:

   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
17   10013        3098203  363.029703                9.900990    99379.381188
3    10011        2813158  348.428571                9.952381    95382.321429
12   10036        1668202  330.636986                9.828767    90511.875000
11   10014        2795098  315.181818                9.988636    86281.022727
4    10003        2289645  293.296296                9.866667    80289.861111
15   10022        2254331  287.823529                9.926471    78791.691176
1    10023        2411269  276.402597                9.961039    75665.211039
9    10028        2588357  273.794521                9.808219    74951.250000
0    10025        1438622  253.134454                9.840336    69295.556723
5    11201        1565379  243.682353                9.823529    66708.044118

    breakeven_period
17         31.175511
3          29.493495
12         18.430753
11         32.395281
4          28.517237
15         28.611278
1          31.867604
9          34.533874
0          20.760667
5          23.466120


Top 10 zipcodes with lowest revenue:
```

```
   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
21   11003         432277  180.000000               10.000000    49275.000000
13   11434         493049  136.875000                9.250000    37469.531250
6    11234         565206  135.111111                9.222222    36986.666667
20   10305         563792  132.000000                9.833333    36135.000000
14   10306         425597  117.500000                9.500000    32165.625000
23   10308         529025  109.500000               10.000000    29975.625000
24   10303         430472  104.000000                9.250000    28470.000000
19   10304         422999   93.333333                8.666667    25550.000000
22   10309         471631   85.000000               10.000000    23268.750000
7    10314         437976   73.000000                9.500000    19983.750000

    breakeven_period
21          8.772745
13         13.158665
6          15.281345
20         15.602380
14         13.231423
23         17.648506
24         15.120197
19         16.555734
22         20.268858
7          21.916607
```

[73]:
```
import plotly.express as px

fig = px.scatter(sort_revenue, x="yearly_revenue", y="zipcode", color="zipcode",
                 size='yearly_revenue', hover_data=['yearly_revenue'])
fig.show()
```

3.5.3 Breakeven period

The below tables and graph identifies the breakeven period for different zipcodes. Zipcodes 10013, 10011, 10036 take the least time to breakeven with breakeven period of 8 years, 13 years and 13 years respectively whereas, zipcode 10021, 10028 and 10014 take the longest to breakeven with breakeven period of 36, 34 and 32 years respectively.

[74]:
```
sort_breakeven_top = sort_breakeven.head(10)
print("Top 10 zipcodes with fastest breakeven period: ")
display(sort_breakeven_top)

sort_breakeven_bot = sort_breakeven.tail(10)
print("Top 10 zipcodes with maximum breakeven period: ")
display(sort_breakeven_bot)
```

```
Top 10 zipcodes with fastest breakeven period:

   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
```

```
21   11003         432277  180.000000               10.000000   49275.000000
13   11434         493049  136.875000                9.250000   37469.531250
14   10306         425597  117.500000                9.500000   32165.625000
24   10303         430472  104.000000                9.250000   28470.000000
6    11234         565206  135.111111                9.222222   36986.666667
20   10305         563792  132.000000                9.833333   36135.000000
19   10304         422999   93.333333                8.666667   25550.000000
23   10308         529025  109.500000               10.000000   29975.625000
12   10036        1668202  330.636986                9.828767   90511.875000
22   10309         471631   85.000000               10.000000   23268.750000

    breakeven_period
21          8.772745
13         13.158665
14         13.231423
24         15.120197
6          15.281345
20         15.602380
19         16.555734
23         17.648506
12         18.430753
22         20.268858


Top 10 zipcodes with maximum breakeven period:
   zipcode  Property_Cost       price  review_scores_location  yearly_revenue  \
8    11215        1302932  181.608466                9.671958    49715.317460
4    10003        2289645  293.296296                9.866667    80289.861111
15   10022        2254331  287.823529                9.926471    78791.691176
3    10011        2813158  348.428571                9.952381    95382.321429
17   10013        3098203  363.029703                9.900990    99379.381188
1    10023        2411269  276.402597                9.961039    75665.211039
2    10128        1994237  226.156250                9.765625    61910.273438
11   10014        2795098  315.181818                9.988636    86281.022727
9    10028        2588357  273.794521                9.808219    74951.250000
10   10021        2073139  208.884615                9.423077    57182.163462

    breakeven_period
8          26.207858
4          28.517237
15         28.611278
3          29.493495
17         31.175511
1          31.867604
2          32.211730
11         32.395281
9          34.533874
10         36.254994
```

[75]:
```
import plotly.express as px

fig = px.scatter(sort_revenue, x="breakeven_period", y="zipcode",␣
↪color="zipcode",
                 size='breakeven_period', hover_data=['breakeven_period'])
fig.show()
```

3.5.4 Location Review Score

3.5.5 Review_score_location and Yearly Revenue

The review score based on the location seems to be all over the chart and is less conclusive in determining the revenue as the properties in zipcodes yeilding least revenue also have high ratings along with low ratings. However, most of the high revenue yielding zipcodes have higher rating

[76]:
```
import plotly.express as px

fig = px.scatter(sort_revenue, x="yearly_revenue", y="review_scores_location",␣
↪color="zipcode",
                 size='review_scores_location',␣
↪hover_data=['review_scores_location'])
fig.show()
```

3.5.6 Review_score_location and Property cost

This chart does show that most of the zipcode having properties that cost high tend to have highers review score based on location. However, there are few zipcode having high review score based on location even when they contain least expensive properties.

[77]:
```
import plotly.express as px

fig = px.scatter(sort_revenue, x="Property_Cost", y="review_scores_location",␣
↪color="zipcode",
                 size='review_scores_location',␣
↪hover_data=['review_scores_location'])
fig.show()
```

[ ]:
```

```

4 Conclusion and Recommendation

The following conclusions can be derived from the analysis:

- Properties in zip codes that represent Manhattan are the most expensive followed by properties in Brooklyn.
- Properties in zipcodes belonging to Manhattan have highest daily price.

- Properties near zipcodes belonging to Manhattan has the highest yearly revenue ranging from 75k-100k dollars per year
- It can also be concluded that zipcodes nearby Central Park cost the most and yeild the highest revenue as well.
- Properties in zipcodes belonging to Staten Island have the least cost as well as yield the least revenue followed by properties in Rochdale.
- Cheaper properties in zip codes belonging to Staten Island and Rochdale acieve breakeven earlier rather than properties in zipcodes belonging to Manhattan.
- Review scores based on location tend to be higher for properties having high cost as well as yeilding high revenue but this relation is not exclusive.

The following Recommendations can be derived from the analysis:

- Since, the investor is seeking High short term Return on Investment, s/he can purchase properties in Manhattan which provide the highest yearly revenue.
- Zipcodes providing highest short-term return on investment with yearly revenue more than 75k dollars are:
  - 10013
  - 10011
  - 10036
  - 10014
  - 10003
  - 10022

5 Further Possibilities

There are additional analyis that can be done to make more concrete and better decisions. Some of the possible additional steps are: - Visuallizations incorporating latitude and longitude data can be used for mapping and better visualization which was not done due to time constraints - Collect and incorporate additional data to make better analysis. Factors like House Price Index and inflation are very important and should be incorporated. - One of the major factors in real state industry is safety. Open source Crime data can be added to the dataseet to further narow down profitable properties in safe neighborhood. - Data related to availability of transportation, natural scenary, entertainment, shopping malls, restaurants, hospitals, groceries among other can affect the choice of property for renters and incorporating such data will provide us with more realistic insights. - Conducting Competitor analysis is very important while making any business decisions. So, adding details/ data regarding nearby competitors can be very useful anf informative. - Segmenting the property type and target customersand conducting analysis for seperately may prove to be more effective. The needs and requirements of a business person will be totally different than that of a student. If we specific customers at a timeand conduct analysis, it may lead to effective insights. - Adding additional data regarding the costs such as property tax, property maintenace cost, utility cost and so on must be incorporated to provide more realistic business insights. - Machine learning models can be used for better and accurate predictions such as property price prediction based on the features rather than just longitudinal data of one feature. - Natural Language Processing algoriths can be used to get better and more insights from the unique text data. - Also, sentiment analysis can be dome from various peoples social media data inorder to get the sentiment of different categories of customers towards different properties which can help us in making more informed decisions.