ADVANCES IN DATA SCIENCE/ARCHITECTURE

Final Project
REPORT

Submitted By:
MEGHA SINGH
SNEHA MALSHETTI

Table of Contents

1. PROBLEM STATEMENT

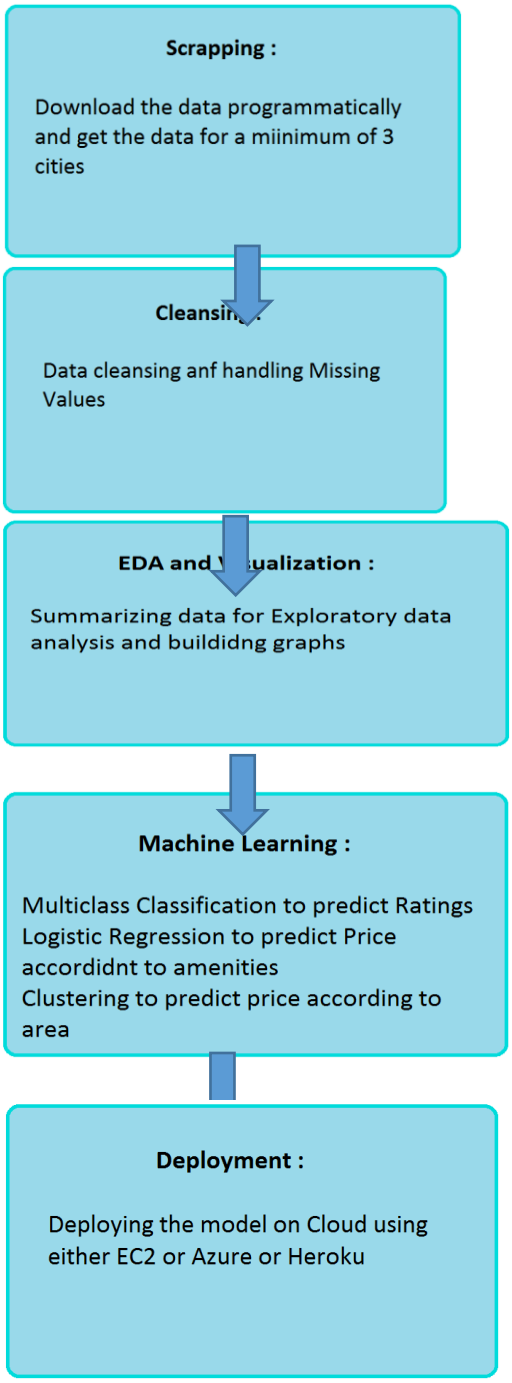
“ Airbnb wants it analysis and prediction for prices available to their customers to make right decisions while choosing venues for providing good customer experience.

Airbnb also to extend its services to owners or hosts by providing their feedback on how they can improve business.

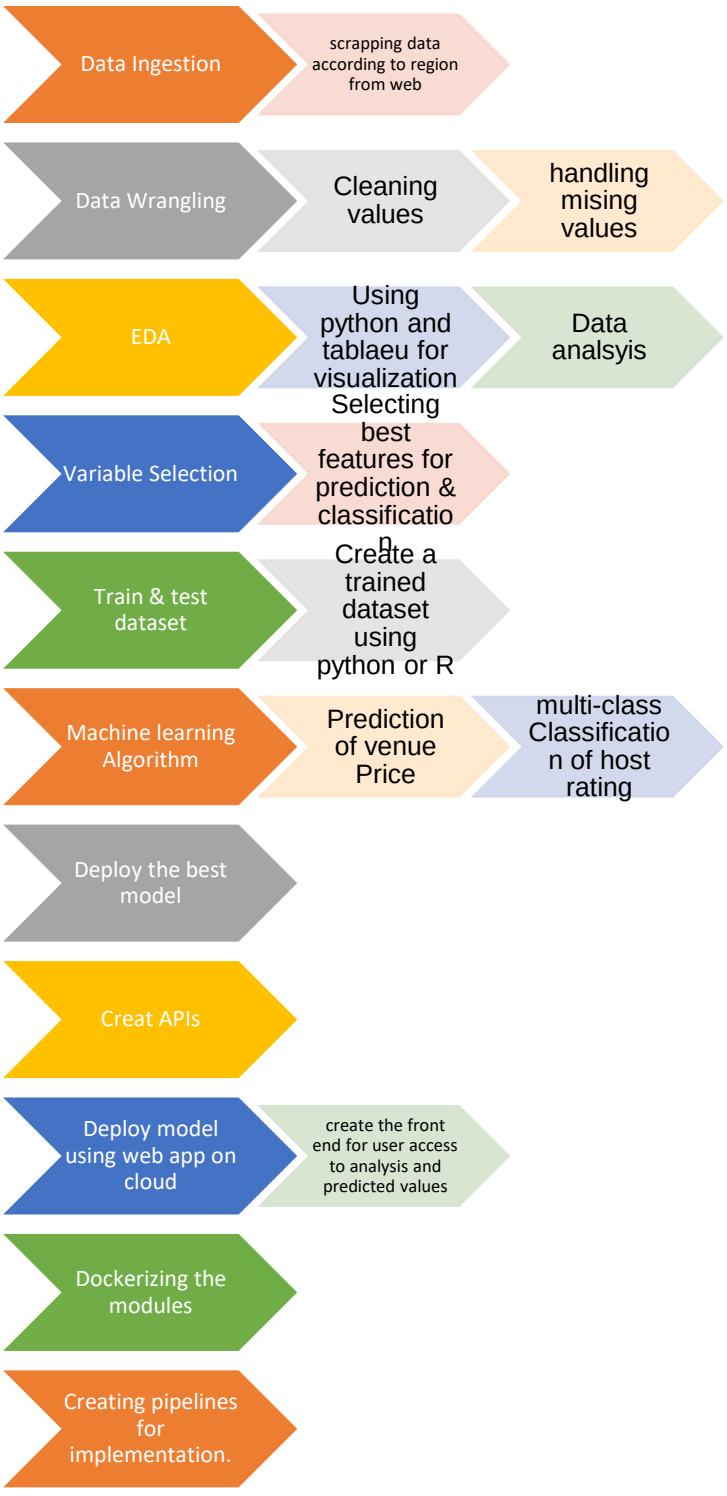
Also, for investors they decide to help them with searching for good investment opportunities based on location. Based on best model evaluation outcome they wish to deploy the services on cloud. “

.

2. Objective Flow of the Project



3. Objective Flow And Processes Involved in Project

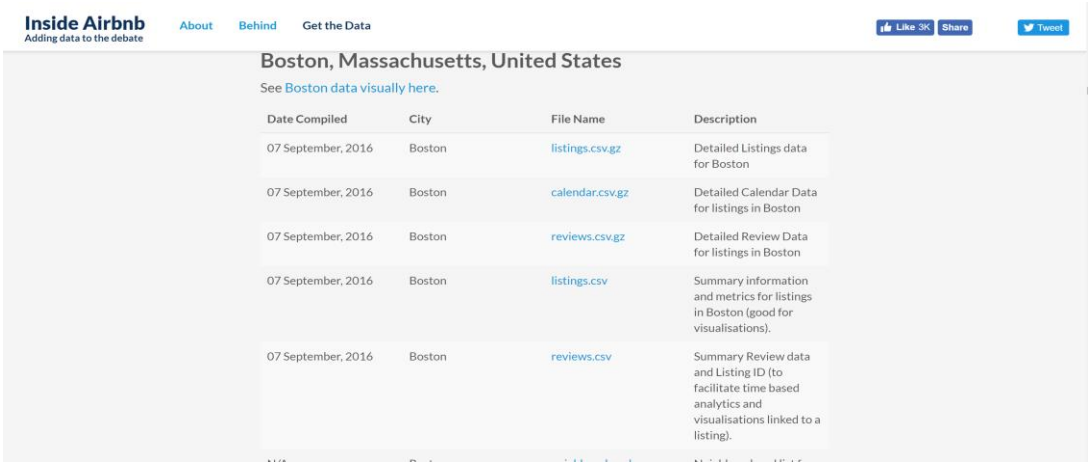


4. Data Ingestion:

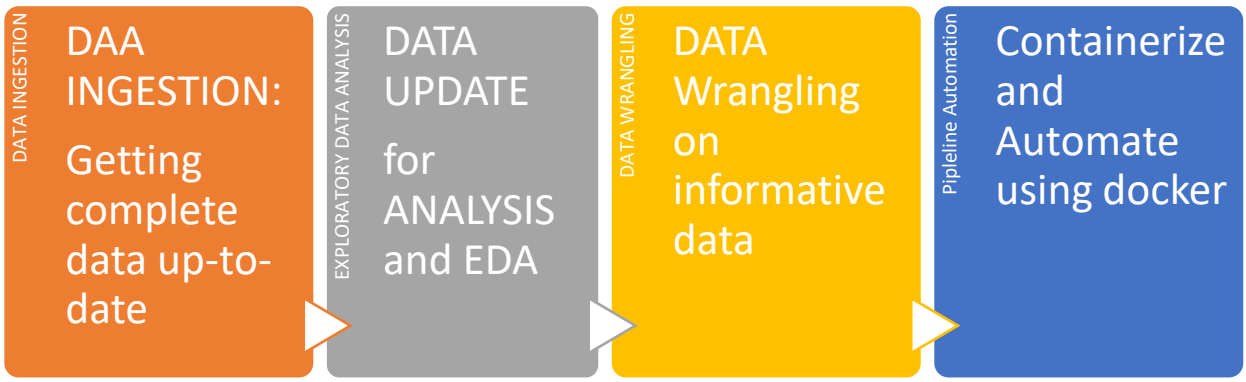
Dataset :

The link for the dataset of Airbnb is as below :
http://insideairbnb.com/get-the-data.html
The Airbnb daaset consists of 3 files namely,
“listings.csv.gz”
“reviews.csv.gz”
“calendar.csv.gz”
The Listings dataset:The listings dataset consists of all current the listings in the Airbnb database for a particular city.

The Reviews dataset: Consists of all the reviews for the respective Listings
The Calendar dataset: Consists of all the listings availability for each day of the year
These files were present for each city and there were archived files for the same as well which were scrapped at different points in the year. We have only picked the latest files for analysis.



2.1Flowchart of Processing of Data Analysis



Flow for the Data Ingestion:

Install all Packages Anaconda, Python, Jupyter Notebook, Shell, AWS account setup, Amazon S3.

Code to Fetch the Real-time data from API using the config files.

- Python script to read a config.json file and source the entire dataset from the FTP link.

Code to Downlaod the data set on local as well On Amazon S3 bucket.

- Python script using boto3 connection create a Bucket on Amazon S3 called Team<No><State>Assignment1" and checks if the bucket already exists.

Update the data set for everyday everytime the code is invoked, on both S3 and local

- Run the code and should fetch the CSV from Api and upload directly to the S3 bucket by creating a new file with specific required name
- Also, If run another day, then it should create a new file. For example, if I run this code tomorrow,it should download the new file and create NJ_12062017_WBAN_5902.csv.

Add the updated data in the CSV based on Station ID and date using the config json file.

- If the code is run after few days it will fetch all the data set and take the existing csv(old) and the new csv and merge into one single data set as a whole,
- Also code shouldn't create another file for the same day.

Create the log for Everytime the Bucket is created, file is downloaded, and error for already Existing file

- Script to creat a log file in format **ddmmyyMMSS.log** everytime the file is uploaded or every instance occurred on S3 bucket, includint the error message of already existing file or already downloaded file.

Create a docker image for this complete script as: DATA INGESTION which can be invoked on running image by using run.sh shell script.

- Create a Docker file with setting the entrypoint for run.sh which will invoke the dataingestion.py script file and execute the whole process, only by running the docker image

1. For Data Ingestion, we had to first of all Scrap the data from website
2. We used parallel processing for fetching data by using threads and multithreading
3. All the files are downloaded and merged on S3 and local
    - create a different name this is to keep all bucket names on the cloud unique.

4. We have used Beautiful soup library to scrape all the required city urls and then downloaded the data for the same.

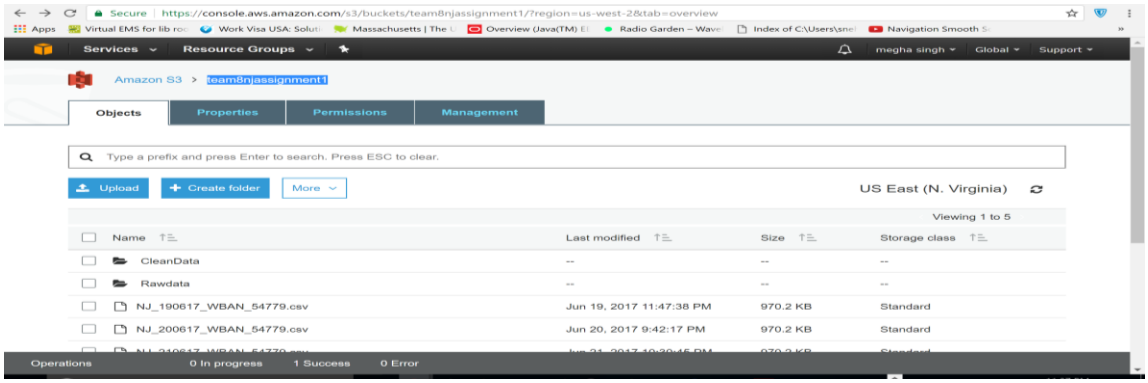
```
def convertstring(city):
    city = city.lower()
    city = city.replace(".", "")
    print (city)
    city = re.sub(' ', '-', city)
    print (city)
    city = re.sub(',', '-', city)
    print (city)

    logprint(city)
    return city
```

```
def getfileurls(city, filetype):

    listname = filetype+"list"
    r = requests.get("http://insideairbnb.com/get-the-data.html")
    soup = BeautifulSoup(r.content)
    links = soup.find_all("a")

    tablename = "table table-hover table-striped " + city
    logprint(tablename)
    g_data = soup.find_all("table", {"class": tablename})
    if len(g_data) == 0:
        logprint("Please Enter Proper City Name")
        my_logger.error("Invalid City Name Entered!")
        exit()

    extractedlist = []
    extratfile = filetype+".csv.gz"
    for items in g_data:
        links2 = items.find_all("a")
        for alllinks in links2:
            if extratfile in alllinks.get("href"):
                if "/united-states/" in alllinks.get("href"):
                    givenlinks = alllinks.get("href")
                    # logprint(givenlistingslink)
                    extractedlist.append(givenlinks)
                else:
                    logprint("Please Enter cities in US only !")
                    my_logger.error("Invalid Country city Entered!")
                    exit()

    for listitems in extractedlist:
```

5. We pull the required city name from a config file and then use that city name to get the files in the above code.
6. The above code only fetchess the list of urls . The code below downloads the csv and gives us a merged file

```
def getmergeddata(scrapedlist, filetype, city):
    mergeddf = pd.DataFrame()
    filename = filetype+".csv.gz"
    for listitems in scrapedlist:
        logprint(listitems)
        # download = urllib.urlretrieve(listitems, 'listings.csv.gz')
        download = urllib.urlretrieve(listitems, filename)

        data = pd.read_csv(filename, compression='gzip', error_bad_lines=False, low_memory=False)
        # print(data)
        logprint(data.shape)
        # print (data.dtypes)

        mergeddf = pd.concat([mergeddf, data], axis=0).drop_duplicates().reset_index(drop=True)
        logprint("Shape of merged file is :" + str(mergeddf.shape))
        mergeddf = mergeddf.drop_duplicates(['id'], keep='first')
        if filetype == "listings":
            mergeddf[["city"]] = city
            logprint(mergeddf[["city"]].head(15))
        # logprint(mergeddf.head(15))
        logprint("Shape of merged file with duplicates removed is :" + str(mergeddf.shape))
        remove_file(filename)
        return mergeddf
```

7. The S3 Bucket we created
   Also, check if the bucket exist it will not be created on S3.
   S3Bucket Created :



8. Each file is converted to csv and sent to S3 after merging and The execution of all the functions is as below :

```
def startexecution(city1, city2):
    # create_directory(DIR_NAME)
    log("Program Execution Begins!")
    logprint("Program Execution Begins!")

    cstring1 = convertstring(city1)
    cstring2 = convertstring(city2)
    log("Getting the files by Scraping!")
    logprint("Getting the files by Scraping!")

    filetype1 = "listings"
    filetype2 = "reviews"
    filetype3 = "calendar"

    city1listingurllist = getfileurls(cstring1, filetype1)
    city2listingurllist = getfileurls(cstring2, filetype1)
    city1reviewurllist = getfileurls(cstring1, filetype2)
    city2reviewurllist = getfileurls(cstring2, filetype2)
    # city1calendarurllist = getfileurls(cstring1, filetype3)
    # city2calendarurllist = getfileurls(cstring2, filetype3)
    log("Retrieving Data!")
    logprint("Retrieving Data!")

    dfcity1listing = getmergeddata(city1listingurllist, filetype1, cstring1)
    dfcity2listing = getmergeddata(city2listingurllist, filetype1,cstring2)
    dfcity1review = getmergeddata(city1reviewurllist, filetype2, cstring1)
    dfcity2review = getmergeddata(city2reviewurllist, filetype2, cstring2)
    # dfcity1calendar = getmergeddata(city1calendarurllist, filetype3, cstring1)
    # dfcity2calendar = getmergeddata(city2calendarurllist, filetype3, cstring2)
    log("Merging Files!")
    logprint("Merging Files!")
```

9. We have also put in the logging functionality in place and the code for it is s below:

```
# Generating date string

datestr2 = time.strftime("%d%m%Y%H%M%S")
datestr2 = datestr2[0:4] + datestr2[-8:-6]

# log generation of files on local directory

LOG_FILENAME = datestr2 + '.log'
# Set up a specific logger with our desired output level
my_logger = logging.getLogger('MyLogger')

if not my_logger.handlers:
    my_logger.setLevel(logging.DEBUG)

    # Add the log message handler to the logger
    handler = logging.handlers.TimedRotatingFileHandler(filename=LOG_FILENAME, when='d', interval=1,
                                                        backupCount=120)

    my_logger.addHandler(handler)
    # create formatter
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s', datefmt='%m/%d/%Y %I:%M:%S %p')

# add formatter to handler
handler.setFormatter(formatter)

# See what files are created
logfiles = glob.glob('%s*' % LOG_FILENAME)

for filename in logfiles:
    print (filename)


# ..............................................Log file generated.....................................................
```

10. A log file gets generated everyday to track the entire day's activities.
11. The log file logs the activities to the greatest detail in time to the second .

The enteries in the log file look as shown below.

```
08/16/2017 11:39:23 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:44:18 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:44:18 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:44:18 PM - MyLogger - INFO - Thread Execution for Scraping URL data begins!
08/16/2017 11:44:22 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:44:22 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:44:22 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:45:19 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:45:19 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:45:19 PM - MyLogger - INFO - Thread Execution for Scraping URL data begins!
08/16/2017 11:45:20 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:45:21 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:45:21 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:45:52 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:45:52 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:45:52 PM - MyLogger - INFO - Thread Execution for Scraping URL data begins!
08/16/2017 11:45:55 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:45:55 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:45:55 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:50:09 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:50:09 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:50:09 PM - MyLogger - INFO - Thread Execution for Scraping URL data begins!
08/16/2017 11:50:12 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:50:12 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:50:12 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:51:47 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:51:47 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:51:47 PM - MyLogger - INFO - Thread Execution for Scraping URL data begins!
08/16/2017 11:51:50 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:51:50 PM - MyLogger - ERROR - Invalid City Name Entered!
08/16/2017 11:51:50 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:52:08 PM - MyLogger - INFO - Program Execution Begins!
08/16/2017 11:52:08 PM - MyLogger - INFO - Getting the files by Scraping!
08/16/2017 11:52:15 PM - MyLogger - INFO - Retrieving Data!
08/16/2017 11:52:50 PM - MyLogger - INFO - Merging Files!
08/16/2017 11:52:54 PM - MyLogger - INFO - listing.csv Created at 16/08/2017 23:52:54
08/16/2017 11:53:01 PM - MyLogger - INFO - review.csv Created at 16082017235301
08/16/2017 11:53:01 PM - MyLogger - INFO - Uploading Files to S3!
08/16/2017 11:56:59 PM - MyLogger - INFO - Files Uploaded to S3!
08/16/2017 11:56:59 PM - MyLogger - INFO - Program Executed Successfully
```

5. Data Wrangling

1.Retrieving the data from cloud using Boto3 connectecion with s3 bucket:

Getting the Raw data csv form S3

```
In [2]: #pulling from s3

        link="https://s3.amazonaws.com/adsteam8finalairbnb/RawData/listings.csv"
        Airbnb_df =  pd.read_csv(link,low_memory=False)

        print(Airbnb_df.shape)
```

```
(44339, 95)
```

2. The dataframe looks as shown below with 95 columns

Printing the dataframe

```
In [3]: #Airbnb_df= pd.read_csv('listings.csv')
        print(Airbnb_df.head(5))



        #Creating a dataframe for manipulation
        df_new = Airbnb_df
```

```
           id                            listing_url       scrape_id  \
0  12147973  https://www.airbnb.com/rooms/12147973  20160906204935
1   3075044   https://www.airbnb.com/rooms/3075044  20160906204935
2      6976      https://www.airbnb.com/rooms/6976  20160906204935
3   1436513   https://www.airbnb.com/rooms/1436513  20160906204935
4   7651065   https://www.airbnb.com/rooms/7651065  20160906204935

  last_scraped                                      name  \
0   2016-09-07                Sunny Bungalow in the City
1   2016-09-07             Charming room in pet friendly apt
2   2016-09-07              Mexican Folk Art Haven in Boston
3   2016-09-07  Spacious Sunny Bedroom Suite in Historic Home
4   2016-09-07                          Come Home to Boston

                                             summary  \
0  Cozy, sunny, family home.  Master bedroom high...
1  Charming and quiet room in a second floor 1910...
2  Come stay with a friendly, middle-aged guy in ...
3  Come experience the comforts of home away from...
4  My comfy, clean and relaxing home is one block...

                                               space  \
0  Cozy, sunny, family home.  Master bedroom high...
1  Charming and quiet room in a second floor 1910...
2  Come stay with a friendly, middle-aged guy in ...
3  Come experience the comforts of home away from...
4  My comfy, clean and relaxing home is one block...

                                               space  \
0  The house has an open and cozy feel at the sam...
1  Small but cozy and quite room with a full size...
2  Come stay with a friendly, middle-aged guy in ...
3  Most places you find in Boston are small howev...
4  Clean, attractive, private room, one block fro...

                                         description experiences_offered  \
0  Cozy, sunny, family home.  Master bedroom high...                 none
1  Charming and quiet room in a second floor 1910...                 none
2  Come stay with a friendly, middle-aged guy in ...                 none
3  Come experience the comforts of home away from...                 none
4  My comfy, clean and relaxing home is one block...                 none

                              neighborhood_overview        ...          \
0  Roslindale is quiet, convenient and friendly. ...        ...
1  The room is in Roslindale, a diverse and prima...        ...
2  The LOCATION: Roslindale is a safe and diverse...        ...
3  Roslindale is a lovely little neighborhood loc...        ...
4  I love the proximity to downtown, the neighbor...        ...

  review_scores_value requires_license license jurisdiction_names  \
0                 NaN                f     NaN                 NaN
1                 9.0                f     NaN                 NaN
2                10.0                f     NaN                 NaN
3                10.0                f     NaN                 NaN
4                10.0                f     NaN                 NaN

  instant_bookable cancellation_policy require_guest_profile_picture  \
0                f            moderate                             f
```

3. Now we select only the required columns for analysis and delete the rest of them :

Selecting Columns with values for analysis

In [4]:
```
#Selecting only columns required
#list(df_new)
print ("The shape of the old file is :", df_new.shape)
print(df_new.head(5))

df_new1= df_new[['id',
 'xl_picture_url',
 'host_id',
 'host_name',
 'host_response_rate',
 'host_total_listings_count',
 'neighbourhood_cleansed',
 'city',
 'state',
 'zipcode',
 'country',
 'latitude',
 'longitude',
 'property_type',
 'room_type',
 'accommodates',
 'bathrooms',
 'beds',
 'amenities',
 'price',
 'cancellation_policy',
 'weekly_price',
 'monthly_price',
 'security_deposit',
 'cleaning_fee',
 'number_of_reviews',
 'review_scores_rating',
 'instant_bookable',
 'host_is_superhost',
```

We have selected the columns with most value of rows and make sense to data danalysis and prediction with our objective.

4. The selected columns are cleaned and converted to appropriate datatypes to perfor further preprocessing on them :

Cleaning data and converting columns into appropriate datatypes

In [5]:
```
#change datatype onject to float

df_new1['host_response_rate'] = df_new1['host_response_rate'].str.replace('%', '')
df_new1['host_response_rate'] = df_new1['host_response_rate'].str.replace(' ', '')
df_new1['host_response_rate'] = df_new1['host_response_rate'].convert_objects(convert_numeric=True)

# df_new1['host_acceptance_rate'] = df_new1['host_acceptance_rate'].str.replace('%', '')
# df_new1['host_acceptance_rate'] = df_new1['host_acceptance_rate'].str.replace(' ', '')
# df_new1['host_acceptance_rate'] = df_new1['host_acceptance_rate'].convert_objects(convert_numeric=True)

df_new1['id'] = df_new1['id'].convert_objects(convert_numeric=True)

df_new1['host_total_listings_count'] = df_new1['host_total_listings_count'].convert_objects(convert_numeric=True)
df_new1['zipcode'] = df_new1['zipcode'].convert_objects(convert_numeric=True)
df_new1['accommodates'] = df_new1['accommodates'].convert_objects(convert_numeric=True)
df_new1['bathrooms'] = df_new1['bathrooms'].convert_objects(convert_numeric=True)
print(df_new1.head(5))

df_new1['id'].dtype
#df_new1['host_acceptance_rate'].dtype
df_new1.dtypes
```

```
C:\Users\sneha\Anaconda3\lib\site-packages\ipykernel\__main__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  app.launch_new_instance()
C:\Users\sneha\Anaconda3\lib\site-packages\ipykernel\__main__.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

5. Replacing all NaN values we get And checking the missing values

replacing NaN values and missing values

In [6]:
```
df_new1.shape
# fill 0  with no data

df_new1.id.fillna(0, inplace=True)

df_new1.reviews_per_month.fillna(0, inplace=True)
df_new1.instant_bookable.fillna('f', inplace=True)
df_new1.host_is_superhost.fillna('f', inplace=True)
df_new1.review_scores_rating.fillna(0, inplace=True)
df_new1.xl_picture_url.fillna(0, inplace=True)
df_new1.host_name.fillna('not available', inplace=True)
df_new1.number_of_reviews.fillna(0, inplace=True)
df_new1.security_deposit.fillna(0, inplace=True)
df_new1.monthly_price.fillna(0, inplace=True)
#df_new1.amenities.fillna('na', inplace=True)
df_new1.amenities.fillna(0, inplace=True)
#df_new1.price.fillna(0, inplace=True)
df_new1.weekly_price.fillna(0, inplace=True)
df_new1.beds.fillna(0, inplace=True)
df_new1.bathrooms.fillna(0, inplace=True)
df_new1.accommodates.fillna(0, inplace=True)
df_new1.room_type.fillna(0, inplace=True)
df_new1.property_type.fillna(0, inplace=True)
# df_new1.longitude.fillna(0, inplace=True)
# df_new1.latitude.fillna(0, inplace=True)
df_new1.country.fillna(0, inplace=True)
df_new1.zipcode.fillna(0, inplace=True)
df_new1.state.fillna(0, inplace=True)
df_new1.city.fillna(0, inplace=True)
df_new1.neighbourhood_cleansed.fillna(0, inplace=True)
df_new1.host_total_listings_count.fillna(0, inplace=True)
#df_new1.host_acceptance_rate.fillna(0, inplace=True)
df_new1.host_response_rate.fillna(0, inplace=True)
df_new1.host_name.fillna(0, inplace=True)
```

6. Now the data set is reduced to 30 columns for further cleaning and we will use these columns for analysis

Listing the datatypes of the various columns of the dataframe

In [8]:
```
df_new1.dtypes
```

Out[8]:
```
id                          float64
xl_picture_url               object
host_id                       int64
host_name                    object
host_response_rate          float64
host_total_listings_count   float64
neighbourhood_cleansed       object
city                         object
state                        object
zipcode                     float64
country                      object
latitude                    float64
longitude                   float64
property_type                object
room_type                    object
accommodates                float64
bathrooms                   float64
beds                        float64
amenities                    object
price                        object
cancellation_policy          object
weekly_price                 object
monthly_price                object
security_deposit             object
cleaning_fee                 object
number_of_reviews           float64
review_scores_rating        float64
instant_bookable             object
host_is_superhost            object
reviews_per_month           float64
dtype: object
```

7. Now we remove the unwanted rows or the rows with NaN values :

Removing unwanted Rows

In [9]:
```
df_new1 = df_new1[df_new1.id != 0]  #remove  rows with no id

df_new1 = df_new1[df_new1.longitude != 0] # remove rows with no latitude for venue
df_new1 = df_new1[df_new1.latitude != 0] # remove rows with no longitude for venue
df_new1 = df_new1[df_new1.zipcode != 0] # remove rows with no longitude for venue
df_new1 = df_new1[df_new1.state != 0] # remove rows with no longitude for venue
df_new1 = df_new1[df_new1.city != 0] # remove rows with no longitude for venue
df_new1 = df_new1[df_new1.host_name != 0] # remove rows with no longitude for venue
df_new1 = df_new1[df_new1.host_id != 0] # remove rows with no longitude for venue
#df_new1.host_id.shape
df_new1 = df_new1[df_new1.xl_picture_url != 0]


print(df_new1.shape)

amenities = df_new1['amenities'].map(lambda d: [amenity.replace('"', "")\
                                                .replace("{", "")\
                                                .replace("}", "") for amenity in d.split(",")])
df_new1['amenities'] = amenities
possible_amenities = set([item for sublist in amenities for item in sublist])
possible_amenities
```

```
(34590, 30)
```

8. The amenites columns have multiple values we need to split all the values into columns and further can be used for analysis
9. The amenities column in the dataset consists of various values and needs to be split into different columns with 0 and 1 as their output.

Splitting the amenities column

In [10]:
```
df_new1['TV'] = np.where(df_new1['amenities'].str.contains("TV"), 0, 1)
df_new1['Kitchen'] = np.where(df_new1['amenities'].str.contains("Kitchen"), 1, 0)
df_new1['Internet'] = np.where(df_new1['amenities'].str.contains("Internet"), 1, 0)
df_new1['Pets'] = np.where(df_new1['amenities'].str.contains("Pets"), 1, 0)
df_new1['Heating'] = np.where(df_new1['amenities'].str.contains("Heating"), 1, 0)
df_new1['Air Conditioning'] = np.where(df_new1['amenities'].str.contains("Air Conditioning"), 1, 0)
df_new1['Washer'] = np.where(df_new1['amenities'].str.contains("Washer"), 1, 0)
df_new1['Dryer'] = np.where(df_new1['amenities'].str.contains("Dryer"), 1, 0)
print(df_new1.shape)
del df_new1["amenities"]
```

```
(34590, 38)
```

10. Create new clumns with separate amaenities values and adding to the dataframe
11. Some columns in the dataset have special chracters like “$” , which have to be removed

Removing symbols from the cleaning fee and other price columns

In [13]:
```
# df_new1 = df_new1[pd.notnull(df_new1['cleaning_fee'])]
df_new1['cleaning_fee'] = df_new1['cleaning_fee'].str.replace(',', '')
df_new1['cleaning_fee'] = df_new1['cleaning_fee'].str.replace('$', '')
df_new1['cleaning_fee'] = df_new1['cleaning_fee'].astype(float)
df_new1.cleaning_fee.fillna(0, inplace=True)

df_new1.cleaning_fee.head(5)
print(df_new1.shape)
```

```
(34590, 37)
```

12. Converting Score Reviews into Ratings, we get values from 0-5

Converting percent ratings into categorical values 1-5

In [20]:
```
#reviews_per_month
print(df_new1.review_scores_rating)


print("After", df_new1.review_scores_rating)


df_new1['review_scores_rating'] =df_new1['review_scores_rating']/20
df_new1['review_scores_rating']=df_new1['review_scores_rating'].apply(np.ceil)


print("After", df_new1.review_scores_rating)
```

```
0          0.0
1         94.0
2         98.0
4         99.0
5        100.0
```

13. The dataframe is converted to csv and sent to s3 to a folder named “CleanData”

Creating a CSV in the local system

In [23]:
```
df_new1.to_csv("cleanlisting.csv", index=None)
print("csv created")
print(df_new1.shape)
```

```
csv created
(34590, 39)
```

Uploading the Clean file on S3

In [24]:
```
############Connection variables for S3

c = boto.connect_s3(AWSAccess, AWSSecret)
conn = S3Connection(AWSAccess, AWSSecret)
bucket = c.get_bucket('adsteam8finalairbnb')
b = c.get_bucket(bucket, validate=False)

b = c.get_bucket(bucket, validate=False)

k = Key(bucket)
k.key = "CleanData/cleanlisting.csv"
url = "https://s3.amazonaws.com/adsteam8finalairbnb/"
r = requests.get(url)

k = Key(b)
k.key = 'CleanData/cleanlisting.csv'

# k.set_contents_from_filename("wrangleddata.csv")
k.content_type = r.headers['content-type']
k.set_contents_from_filename('cleanlisting.csv')
print('successfully uploaded listing.csv to s3')
```

```
successfully uploaded listing.csv to s3
```

4. Exploratory Data Analysis :

The following are a few plots that we thought would be useful in summarizing the entire 3 CSV
This also helps in Data Analysis. And to make future predictions based on models. The visualization of data also gives us an insight of whether the data has gone bad with a few outliers and missing values or strings in the numeric fields.
We have stored the Data ingestion Files on S3 and now we can easily acces the raw files from S3 storage
Remove redundant or no value columns: For same, First we need to select only the valuable columns and remove the columns which have no values or data lesser than 5%.
Change the datatype to meaningful for analysis After changing and filling the missing data change the data type of column to float, integer or object or Boolean etc.
Fill missing values Nan with mean or Flag: Then we replace all the values to Nan and covert the NaN columns in dataframe into the either flag or mean value of column so it doesn't changes overall values.
For the Exploratory Data Analysis we used Jupyter Notebook:

1. Create a Jupyter notebook to perform EDA on the downloaded data(<state_ddmmyy_stationid>.csv)
2. Using the config json, making the dataframe of the csv dataset: Listing Dataset, Calender Dataset and reviews Dataset
3. Selecting the coloumns with only meaningful data columns

The following graph and code gives the different roomtypes present in the dataset.

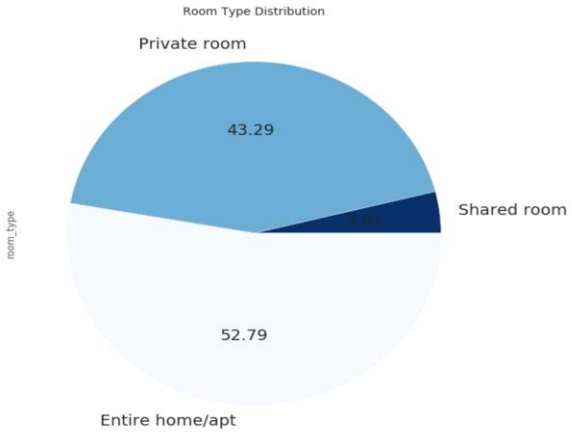
1) Check the type of room available and their count

```
# Categorizing differernt listings based on room_type


#how many types of room
roomTyp=edadata.groupby('room_type').id.count()
roomTyp=roomTyp.reset_index()
roomTyp=roomTyp.rename(columns={'id':'number_Of_Listings'})
roomTyp
```

	room_type	number_Of_Listings
0	Entire home/apt	16617
1	Private room	16820
2	Shared room	1153

Most availavle prperty is type of Apartment or Entire house almost 52% of listing, and rest is shared room and private room type of property in both new york and Boston.

We can check the plot for the same the the room type availbe in listing:



2) Similarly, We can analysie what type of propety and room type are availabe in bot the region and their count for comparison with prices

In [36]:
```
#price o various venur availbale
# analyzing the prices for different room type and property type

venueproperty = edadata.groupby(['property_type','room_type']).price.mean()
venueproperty = venueproperty.reset_index()
venueproperty=venueproperty.sort_values('price',ascending=[0])
venueproperty.head()
```

Out[36]:

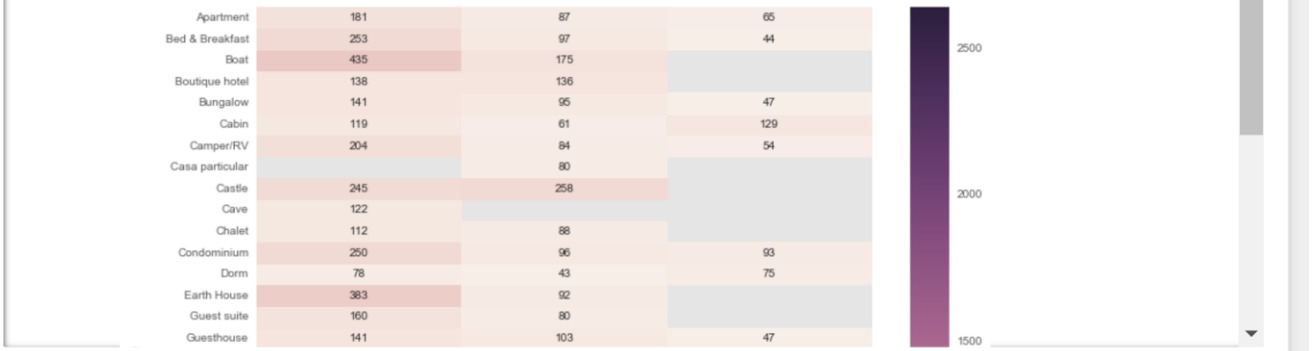
	property_type	room_type	price
80	Yurt	Private room	2642.750000
76	Villa	Entire home/apt	1706.821705
74	Vacation home	Entire home/apt	1322.666667
6	Boat	Entire home/apt	434.707317
31	Earth House	Entire home/apt	383.333333

We can analyse here the most costliest are the Yurt and Villa with almost per day cost is 2642 and 1706 respectively.
This can be further analysed for by using plots.

3) We can now plot the heatmap, to analyse which ones are the costliest combinations by taking 2 parameters in analysis Such as the property type available in area and how much is the count for the same:

```
In [37]: # heat map

         #Plotting the same on a heatMap

         import seaborn as sns

         plt.figure(figsize=(12,12))
         sns.heatmap(edadata.groupby([
                 'property_type', 'room_type']).price.mean().unstack(),annot=True, fmt=".0f")
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x14926528c18>
```



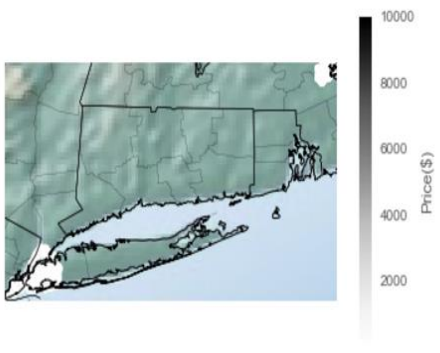
4) We can check which amenities cost more since we have already split the amanities during wrangling,

```
     bathrooms  beds   price
109        6.5   7.0  5700.0
89         4.5   5.0  3500.0
87         4.0  11.0  3025.0
102        5.5   6.0  2750.0
106        6.0   6.0  2000.0
```

7]:

	Dryer	Washer	price
0	1	1	138.148077

5) We can now analyse which state is having more number of listing or property available by showing with price

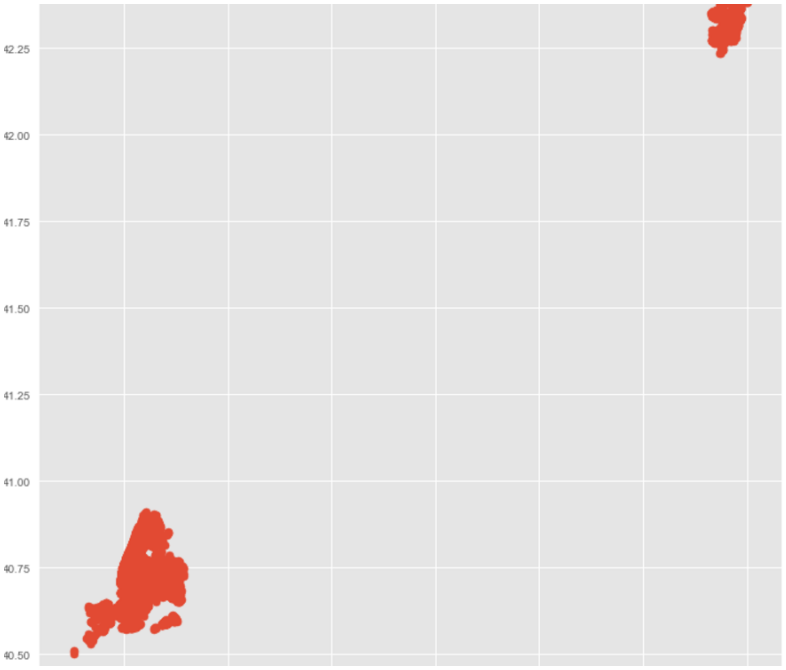



```
plt.rcParams["figure.figsize"] = [15,7]
cb1 = plt.colorbar(sp1)
cb1.set_label('Price($)')
plt.show()
```


```
<matplotlib.figure.Figure at 0x18ad4198>
```

We checked the price based aread, where most high priced properties present.

6) Now anaother observation can be made by comparing whether boston is having more properties or New york



With this graph the east coast is comparitivly less with availabe listing than west coast

```
In [45]: # fig=edadata.hist()

         fig = plt.figure(figsize = (15,20))
         ax = fig.gca()
         edadata.hist(ax = ax)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2881: UserWarning: To output multiple subplots, the
figure containing the passed axes is being cleared
  exec(code_obj, self.user_global_ns, self.user_ns)
```

2) Now we can chck the count and value of all the amenites and dataset

Amenities such, pets, Internet ,TV, heating, dryer, washer are used and most available may show the people prefers properties with these amaneities


('Pets', 'Cat(s)')
('TV', 'Cable TV')
Air Conditioning
Internet
Kitchen
Washer
Dryer
Heating
beds
cleaning_fee
host_acceptance_rate
host_id
host_response_rate
host_total_listings_count
id
latitude
longitude
monthly_price
number_of_reviews
price
review_scores_rating
reviews_per_month
security_deposit
weekly_price
zipcode

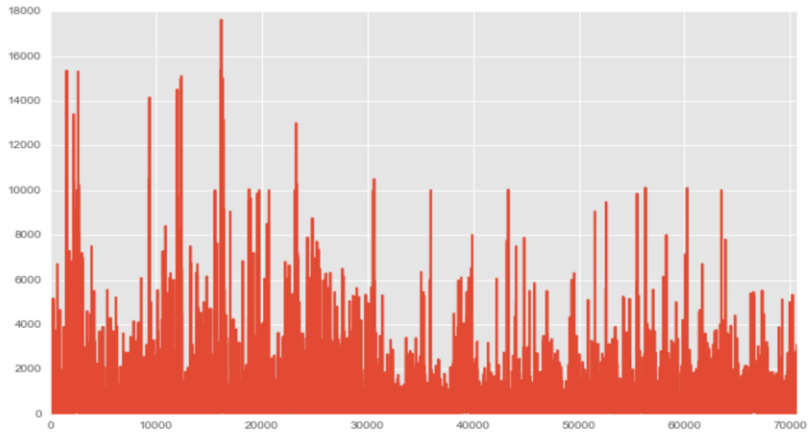
3) Another interesting calculation can be checked how much is the total price for everypropty by adding the security and cleaning fee and the comparing through a graph

```
]: # Total price and adding a new column
edadata['Totalprice'] = edadata['security_deposit'] + edadata['cleaning_fee']+edadata['price']
print(edadata.Totalprice.head())
```

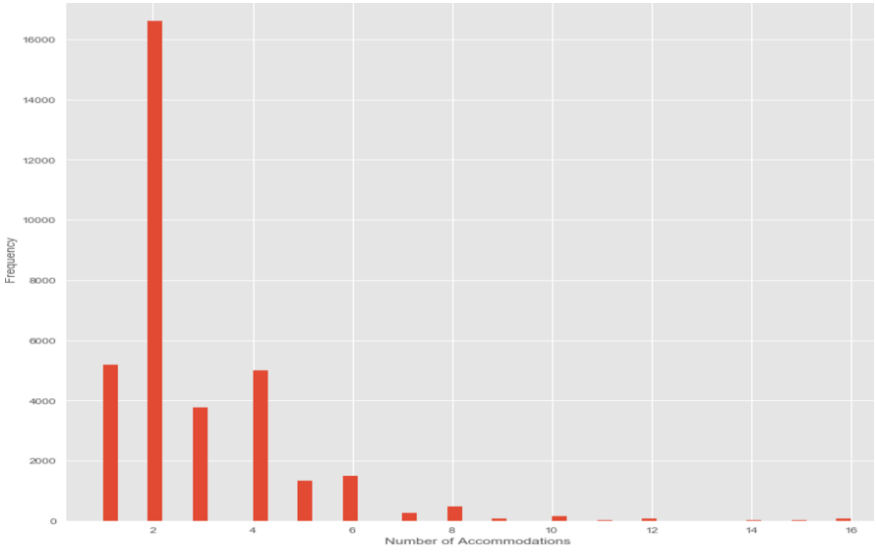
```
0    285.0
1    170.0
2     65.0
3     94.0
4    105.0
Name: Totalprice, dtype: float64
```

4) WE can see the total price distribution through the graph

```
In [47]: edadata.Totalprice.plot.line() # price
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1492c65c780>
```



5) Now we can also analysis how many accomodates available in datset

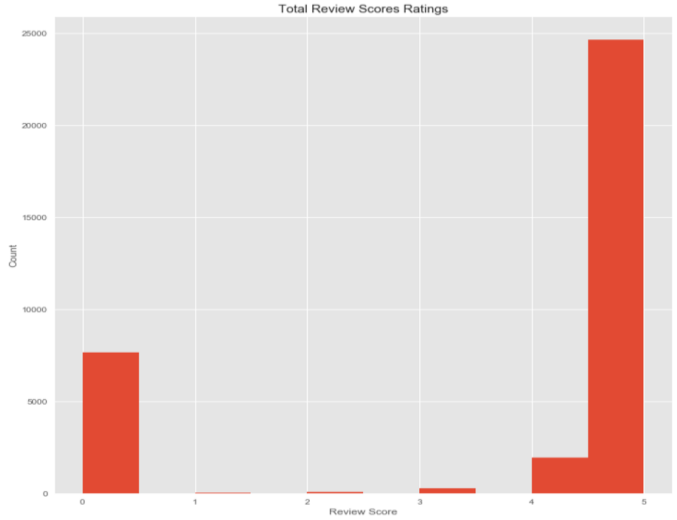


Number of beds and the count of property how many they have

And `visualize distribution of beds`

```
('Number of Uniqu
        11.,  12.
('Beds 1:', 23476
('Beds 2:', 7106)
('Beds 3:', 2312)
('Beds 4:', 941)
('Beds 5:', 357)
('Beds 6:', 184)
('Beds 7:', 63)
('Beds 8:', 41)
('Beds 9:', 15)
('Beds 10:', 27)
('Beds 11:', 7)
('Beds 12:', 6)
('Beds 13:', 1)
('Beds 14:', 2)
('Beds 15:', 1)
('Beds 16:', 7)
```

Review Scored Rating

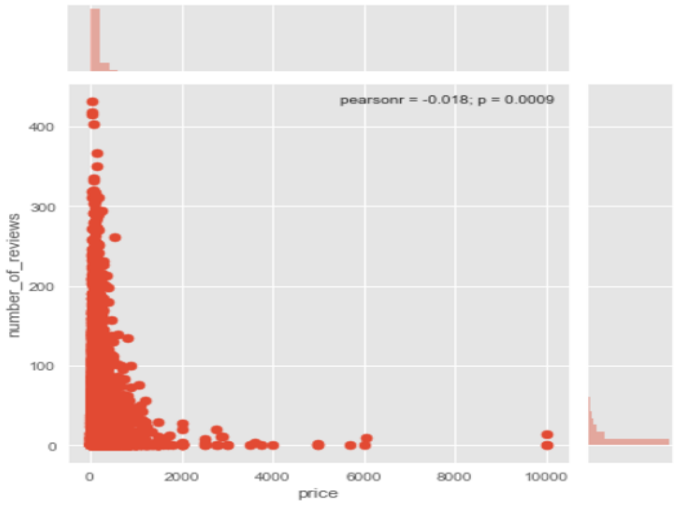


Most of the **host properties are rated 5** almost 25000 of the values of datsets

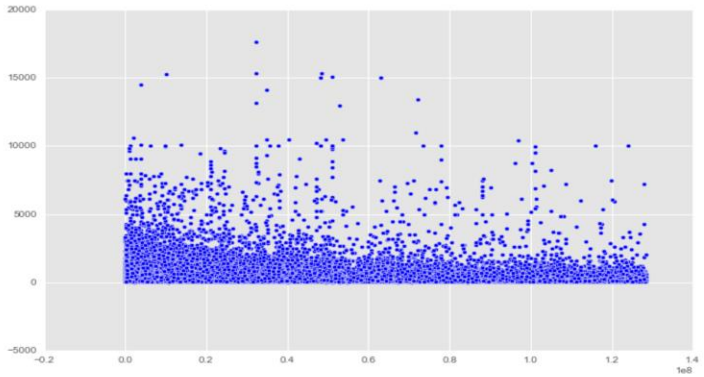
6) Here is the distribution reviews and price:

```
# number of reviews and price
reviews=edadata[['price','number_of_reviews']]
sns.jointplot('price','number_of_reviews',data=reviews)
```

```
<seaborn.axisgrid.JointGrid at 0x12b8ab00>
```



```
In [55]: # x = edadata['host_id']
         # y1 = edadata['Totalprice']
         # plot.scatter()
         plt.scatter(edadata.host_id, edadata.Totalprice)

Out[55]: <matplotlib.collections.PathCollection at 0x1492e4ee9b0>
```

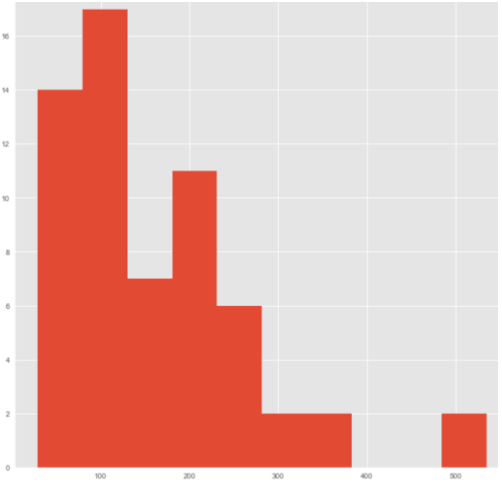


7) The rest of the Analysis is performed on the Calendar dataset:
8) No we can also analysis what kind of properties we hav

```
edadata['property_type'].value_counts()
```

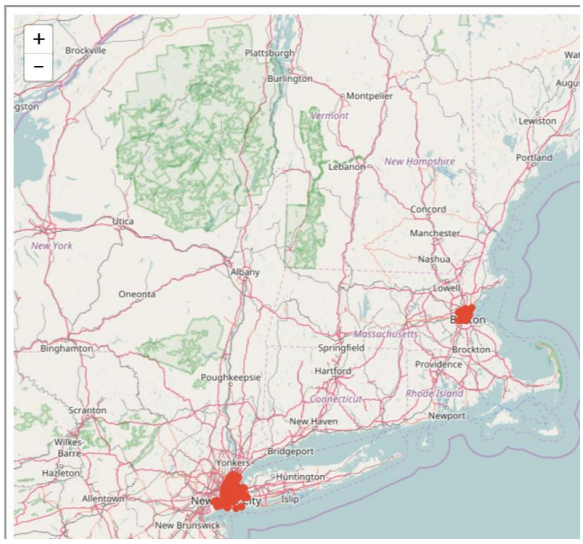
```
Apartment                29058
House                     3341
Loft                       668
Townhouse                  506
Condominium                456
Other                      191
Bed & Breakfast            184
Guesthouse                  37
Timeshare                   33
Hostel                      19
Boutique hotel              17
Villa                       17
Dorm                        13
Boat                        12
Bungalow                     9
Serviced apartment           6
Guest suite                  5
Entire Floor                 4
0                            3
Castle                       2
Hut                          1
Vacation home                1
Camper/RV                    1
Cabin                        1
Lighthouse                   1
Chalet                       1
Earth House                  1
Cave                         1
In-law                       1
Name: property_type, dtype: int64
```

9)
10) Shown below is the yearwise numbers of listings that are added to Airbnb
11) Property time versus roomt type graph



12) Now we can visialize the volume of listing in boston and newyork using mplleaf

```
import mplleaflet

sample = edadata.sample(1000)
plt.scatter(sample['longitude'], sample['latitude'])

mplleaflet.display()
```



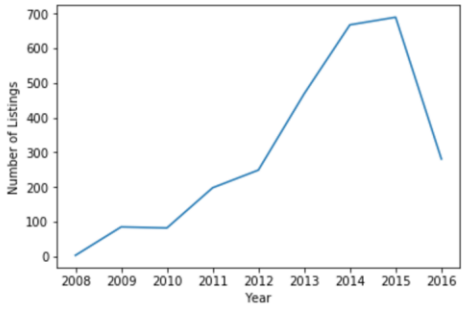
13) The below analysis shows that the number of venues added were at it's peak in 2015 and reduced during 2016. One reason may be that 2016 dataset is not complete.

Ploting a graph for the number of listings added every year to the Boston dataset

In [257]:
```
listing_analysis = listing_count.groupby(['year'])['id'].count()
print (listing_analysis)
```

```
year
2008      3
2009     85
2010     82
2011    198
2012    249
2013    469
2014    668
2015    690
2016    281
Name: id, dtype: int64
```

In [200]:
```
plt.plot(listing_analysis)
plt.ylabel('Number of Listings')
plt.xlabel('Year')
plt.show()
```



The next gragh shows us the yearwise count of hosts added to Airbnb
The same analysis goes for the number of hosts that were added to airbnb in boston over the years.

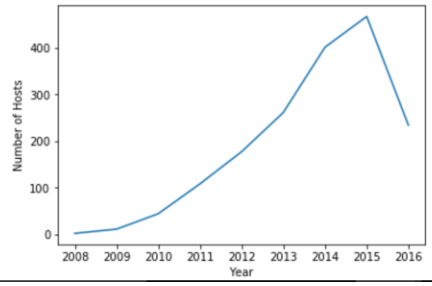
14)

Ploting a graph for the number of hosts added every year to the Boston dataset

In [201]:
```
host_analysis = listing_count.groupby(['year']).agg({"host_id": lambda x: x.nunique()})
print (host_analysis)
```

```
      host_id
year
2008        2
2009       11
2010       44
2011      108
2012      177
2013      261
2014      401
2015      467
2016      234
```

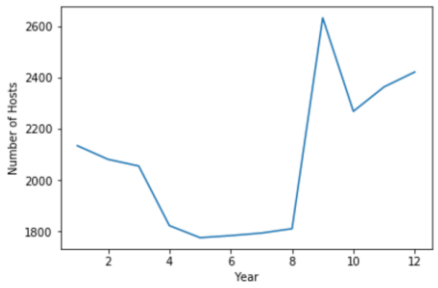
In [202]:
```
plt.plot(host_analysis)
plt.ylabel('Number of Hosts')
plt.xlabel('Year')
plt.show()
```



The below graphs show that during October and November most number of listings are available while from April to August the availability is very less

Graph to show the availability of rooms/Listings in Boston for a single year fro september 2016 to September 2017

In [210]:
```
calendar_listing_price_average = Airbnb_calendar_df[["month", "listing_id", "price"]]

availability_analysis=calendar_listing_price_average.groupby(['month']).agg({"listing_id": lambda x: x.nunique()})


plt.plot(availability_analysis)
plt.ylabel('Number of Hosts')
plt.xlabel('Year')
plt.show()
print (availability_analysis)
print (availability_analysis.shape)
print (availability_analysis.dtypes)
```

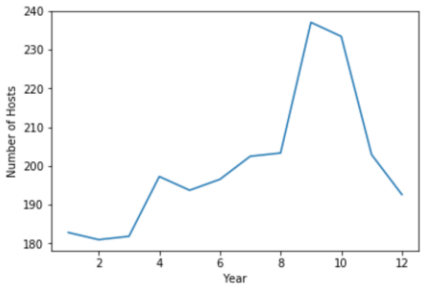



```
       listing_id
month
1            2134
2            2081
3            2055
4            1823
5            1776
6            1784
7            1794
8            1811
9            2632
10           2268
11           2363
12           2421
(12, 1)
listing_id    int64
dtype: object
```

15) Since the season has more listings the price also is high during the same time as shown below :
16)

Average Price throughout the year

```
In [212]: calendar_listing_price_average = Airbnb_calendar_df[["month", "listing_id", "price"]]

          average_monthly_price = calendar_listing_price_average.groupby(['month']).agg({"price": np.mean})


          plt.plot(average_monthly_price)
          plt.ylabel('Number of Hosts')
          plt.xlabel('Year')
          plt.show()
          print (average_monthly_price)
          print (average_monthly_price.shape)
          print (average_monthly_price.dtypes)
```



17)

18)

month	price
1	182.799671
2	180.961028
3	181.818742
4	197.252890
5	193.712295
6	196.535302
7	202.486309
8	203.330142
9	237.055387
10	233.425228
11	202.915712
12	192.600118

```
(12, 1)
price    float64
dtype: object
```

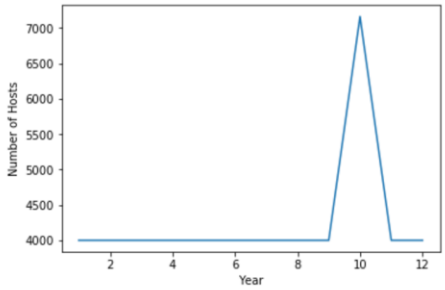
19)

20) The prices, even the maximum prices are skyrocketing in the month of October which makes it ideal time for hosts but not for guests in Boston.

29

Maximum price throughout the year in Boston

In [213]:
```
calendar_listing_price_average = Airbnb_calendar_df[["month", "listing_id", "price"]]

max_monthly_price = calendar_listing_price_average.groupby(['month']).agg({"price": np.max})


plt.plot(max_monthly_price)
plt.ylabel('Number of Hosts')
plt.xlabel('Year')
plt.show()
print (max_monthly_price)
print (max_monthly_price.shape)
print (max_monthly_price.dtypes)
```



21)

```
         price
month
1       4000.0
2       4000.0
3       4000.0
4       4000.0
5       4000.0
6       4000.0
7       4000.0
8       4000.0
9       4000.0
10      7163.0
11      4000.0
12      4000.0
(12, 1)
price    float64
dtype: object
```

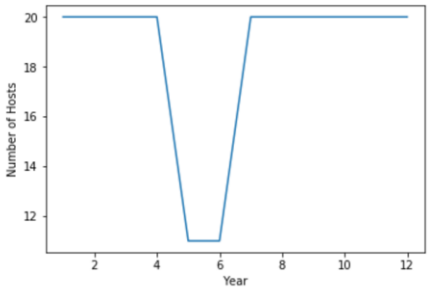
22)

23) Also according to the graph below, it is clear that the best time to have a cheap holiday in Boston is between April and July .

24)

Maximum price throughout the year in Boston

In [214]:
```
calendar_listing_price_average = Airbnb_calendar_df[["month", "listing_id", "price"]]

min_monthly_price = calendar_listing_price_average.groupby(['month']).agg({"price": np.min})


plt.plot(min_monthly_price)
plt.ylabel('Number of Hosts')
plt.xlabel('Year')
plt.show()
print (min_monthly_price)
print (min_monthly_price.shape)
print (min_monthly_price.dtypes)
```



25)

```
       price
month
1       20.0
2       20.0
3       20.0
4       20.0
5       11.0
6       11.0
7       20.0
8       20.0
9       20.0
10      20.0
11      20.0
12      20.0
(12, 1)
price    float64
dtype: object
```

5. Feature Selection for Model

For preprocessing the dataset we chect most relevant columns and value of correlation, and which columns affects the value of price for prediction.
So we use the feature selection method this data has lot of linearity in the columns
We used the exhaustive search meathod for feature selection here:

```
(Intercept)                1.770e+02  2.857e+01   6.194 6.09e-10 ***
host_response_rate        -3.209e-02  4.051e-02  -0.792 0.428199
host_total_listings_count  2.709e-02  1.922e-02   1.409 0.158751
accommodates               1.691e+01  1.514e+00  11.166  < 2e-16 ***
bathrooms                  2.935e+01  3.586e+00   8.184 3.07e-16 ***
beds                      -1.115e+00  2.379e+00  -0.469 0.639156
security_deposit           2.734e-02  6.334e-03   4.317 1.60e-05 ***
cleaning_fee               3.546e-01  4.296e-02   8.254  < 2e-16 ***
number_of_reviews         -5.040e-02  5.539e-02  -0.910 0.362829
review_scores_rating      -5.178e+00  3.956e+00  -1.309 0.190683
reviews_per_month         -1.583e+00  1.131e+00  -1.400 0.161515
ScoredRating               1.803e-01  1.910e+01   0.009 0.992466
moderatePolicy            -1.609e+02  2.679e+01  -6.003 2.00e-09 ***
flexiblePolicy            -1.546e+02  2.686e+01  -5.756 8.86e-09 ***
strictPolicy              -1.612e+02  2.664e+01  -6.052 1.49e-09 ***
Apartment                  6.719e+01  9.560e+00   7.028 2.23e-12 ***
Privateroom                1.519e+01  9.173e+00   1.656 0.097800 .
Notinstant_bookable       -1.295e+01  3.613e+00  -3.585 0.000338 ***
superhost                  8.451e+00  4.841e+00   1.746 0.080886 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 147.9 on 9981 degrees of freedom
```


```
ScoredRating               1.803e-01  1.910e+01   0.009 0.992466
moderatePolicy            -1.609e+02  2.679e+01  -6.003 2.00e-09 ***
flexiblePolicy            -1.546e+02  2.686e+01  -5.756 8.86e-09 ***
strictPolicy              -1.612e+02  2.664e+01  -6.052 1.49e-09 ***
Apartment                  6.719e+01  9.560e+00   7.028 2.23e-12 ***
Privateroom                1.519e+01  9.173e+00   1.656 0.097800 .
Notinstant_bookable       -1.295e+01  3.613e+00  -3.585 0.000338 ***
superhost                  8.451e+00  4.841e+00   1.746 0.080886 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 147.9 on 9981 degrees of freedom
Multiple R-squared:  0.1719,    Adjusted R-squared:  0.1704
F-statistic: 115.1 on 18 and 9981 DF,  p-value: < 2.2e-16
```


With feature selection we get the only few columns for the machine learning model for prediction and classification

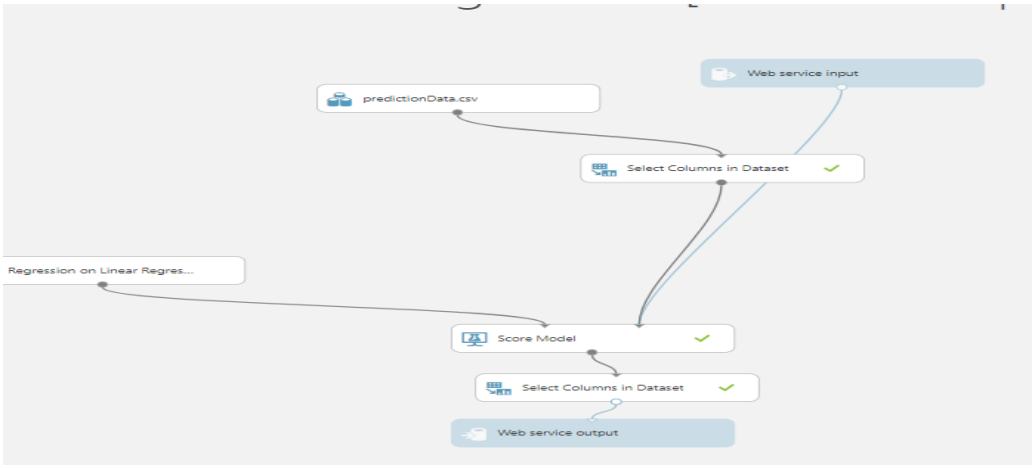
Selected columns are for predicting price of a listing :

Hostresponse rate, accommodates, host listing count, bathrooms, security deoposit, cleaning fee, Policy, instantbookable, superhost, scored rating etc

6. Machine learning models for prediction:

We used machine learning models in azure for price prediction and training and testing dataset

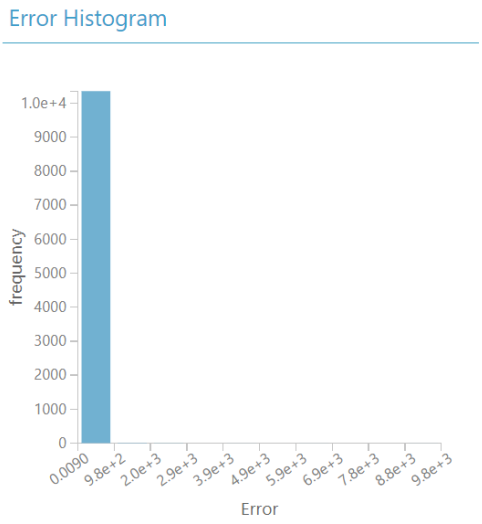
Create the evaluation model and predictive model using the linear regression for price prediction:



We also created Random forest but linear regression had lesser RMSE value ,
So we selected Linear regression as price predictive model
Here are the evaluated values for the model And the error

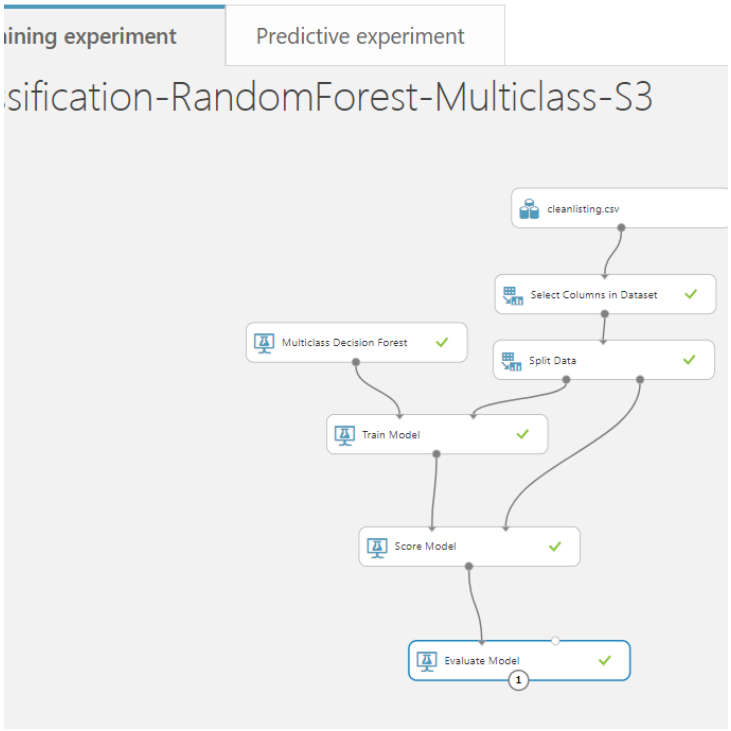
Metrics

Mean Absolute Error	55.709729
Root Mean Squared Error	159.699679
Relative Absolute Error	0.736052
Relative Squared Error	0.810999
Coefficient of Determination	0.189001



8. Machine learning models Host Rating: Using Multi class classification

For predicting whearther the host will be have what rating Negative=0, Positive=2 or Neutral=1 ratings by using multi-class classification machine learning model
For evaluation model these will be the scored probabilities



rows 10377 columns 18

Air Conditioning	ScoredRatingmulticlass	Scored Probabilities for Class "0"	Scored Probabilities for Class "1"	Scored Probabilities for Class "2"	Scored Labels
1	2	0.5	0	0.5	0
1	2	0	0	1	2
1	0	1	0	0	0
1	2	0	0	1	2

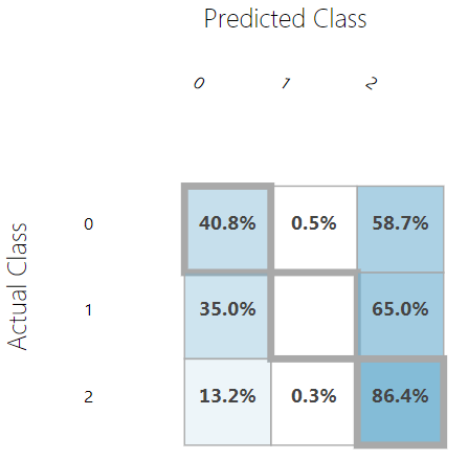
Classification-RandomForest-Multiclass-S3 ›

Metrics

Overall accuracy	0.755035
Average accuracy	0.83669
Micro-averaged precision	0.755035
Macro-averaged precision	0.430299
Micro-averaged recall	0.755035
Macro-averaged recall	0.424095

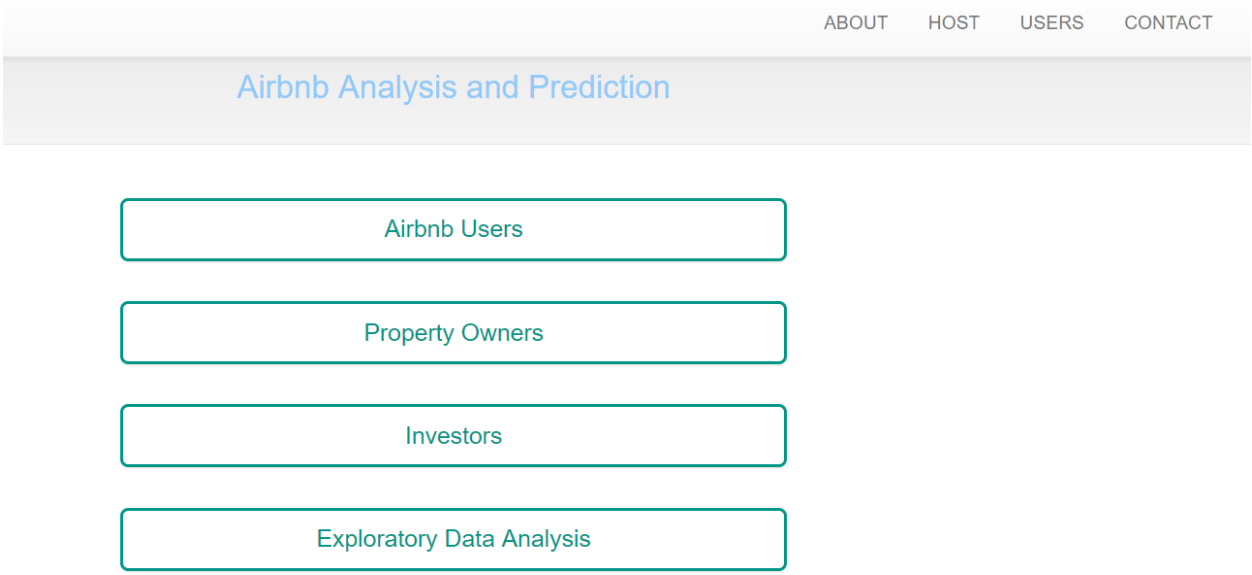
The confusion matrix here shows is

Confusion Matrix

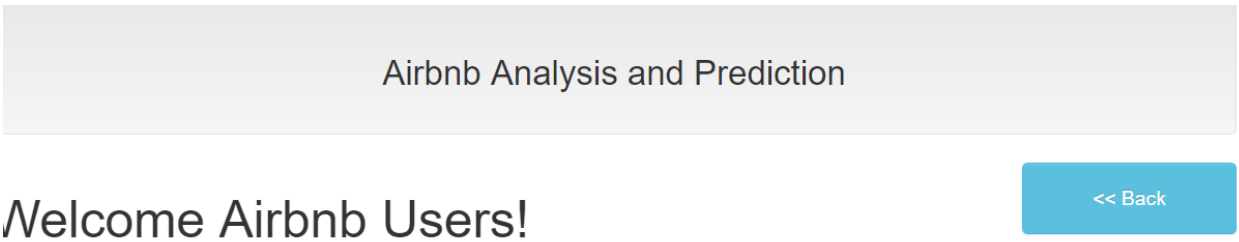


Here the in the confusion matrix the diagonal parameters or fine so model is working fine but may be improved, what we get here is atlreat around 40% of actual positives With prediction accuracy of 75%

7. Creating the Web App for deploying the same using Flask



We created the user, Owners, Investors, and EDA analysis on app
Persona: User:



For checking the deals in city can be available by analysis the data within the budhet

Airbnb Analysis and Prediction

<< Back

The Best Deals in the City !

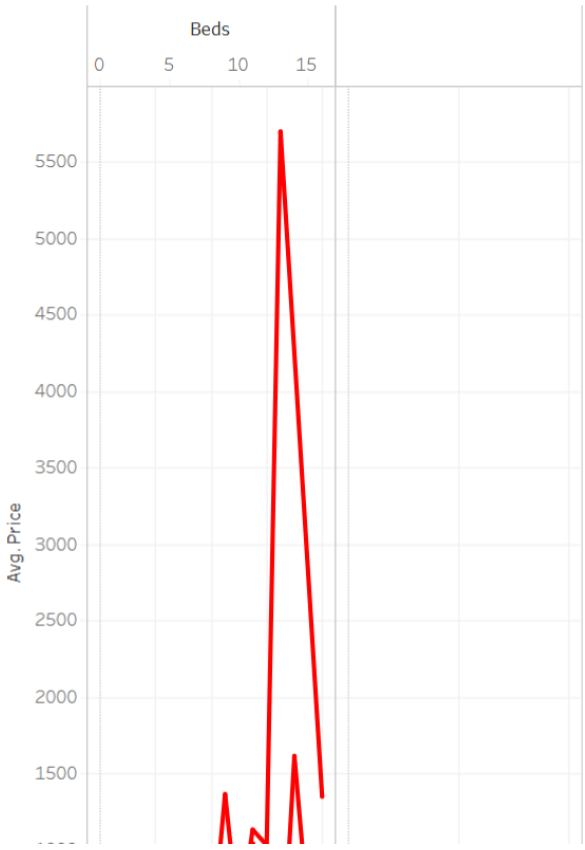
Country: US

City: Boston

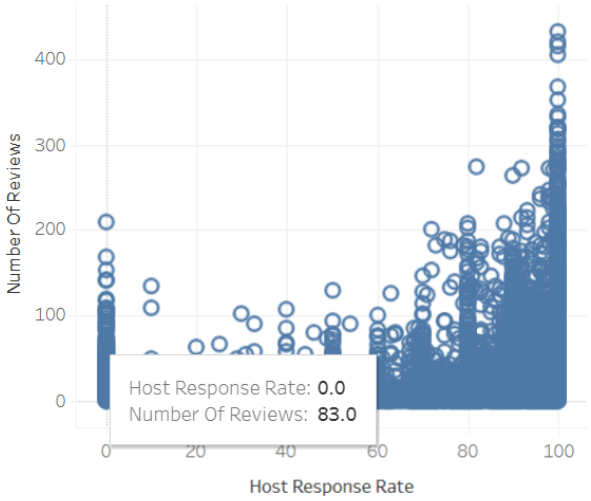
Price Range (Per day basis):

Check the prices evaluations

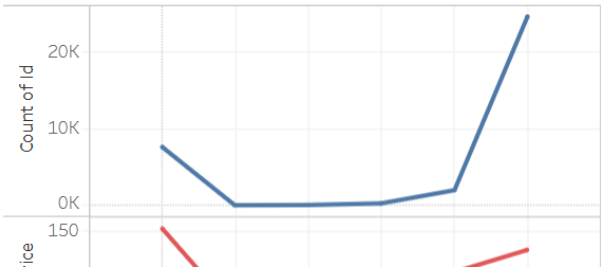
Bath beds host response rate

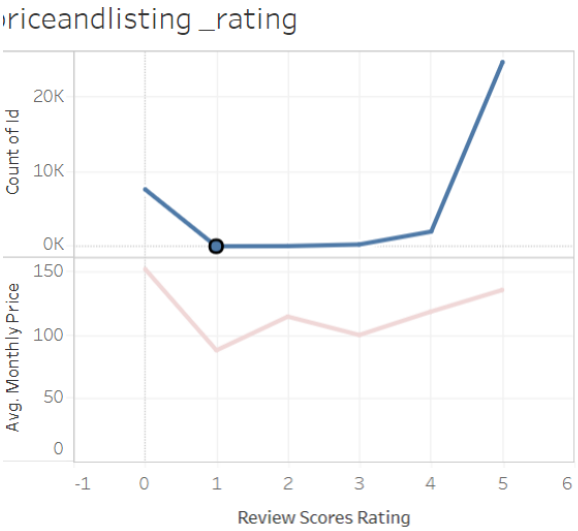


Responserate_Reviews count

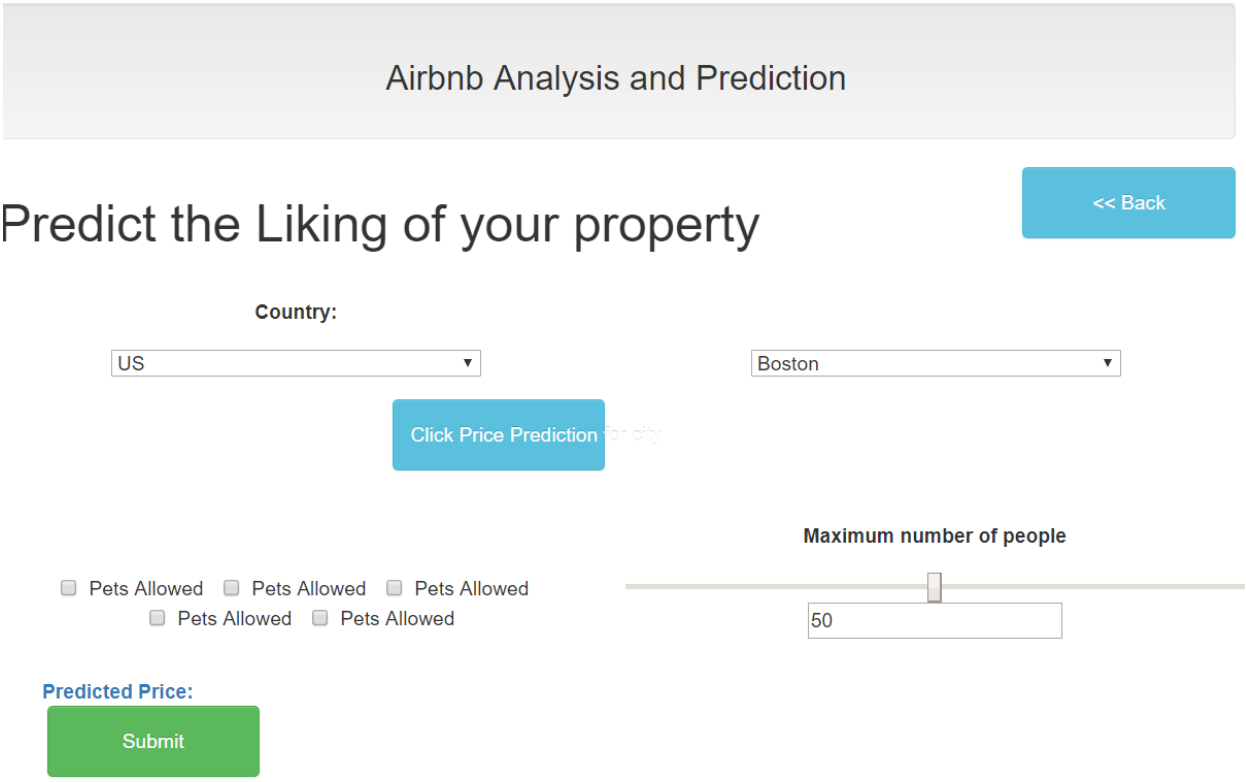


priceandlisting _rating



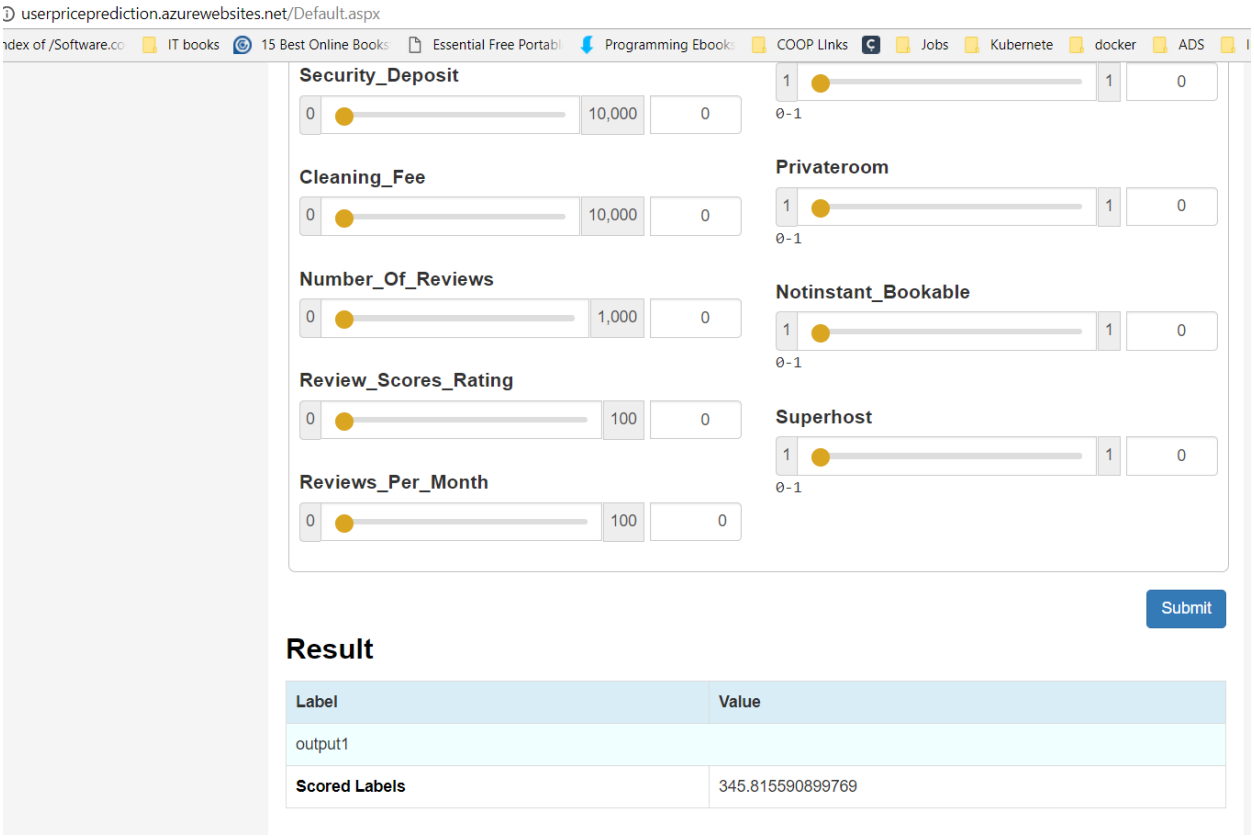


User can have analysis of prices and listing available

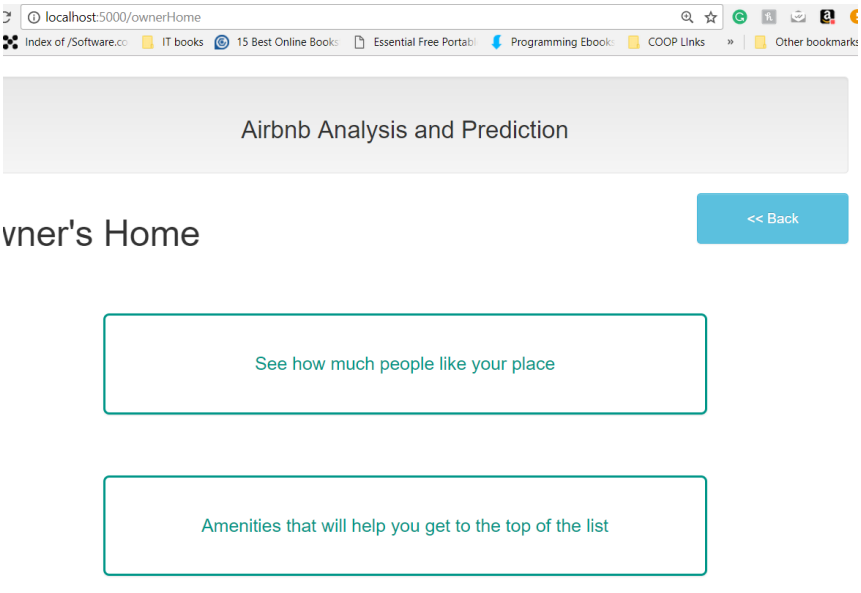


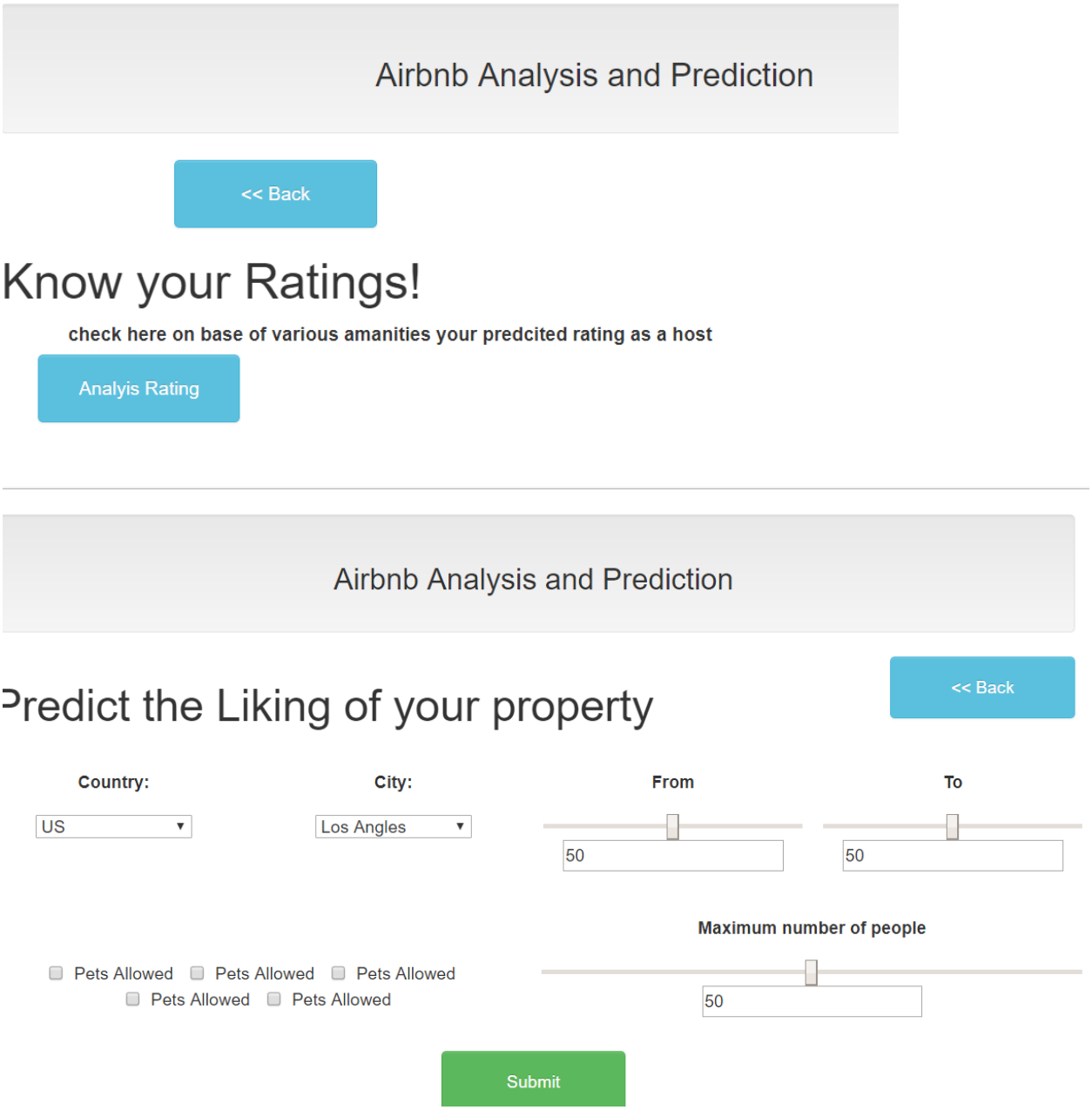
For predicting the prices of lisitng

Prediction on price:

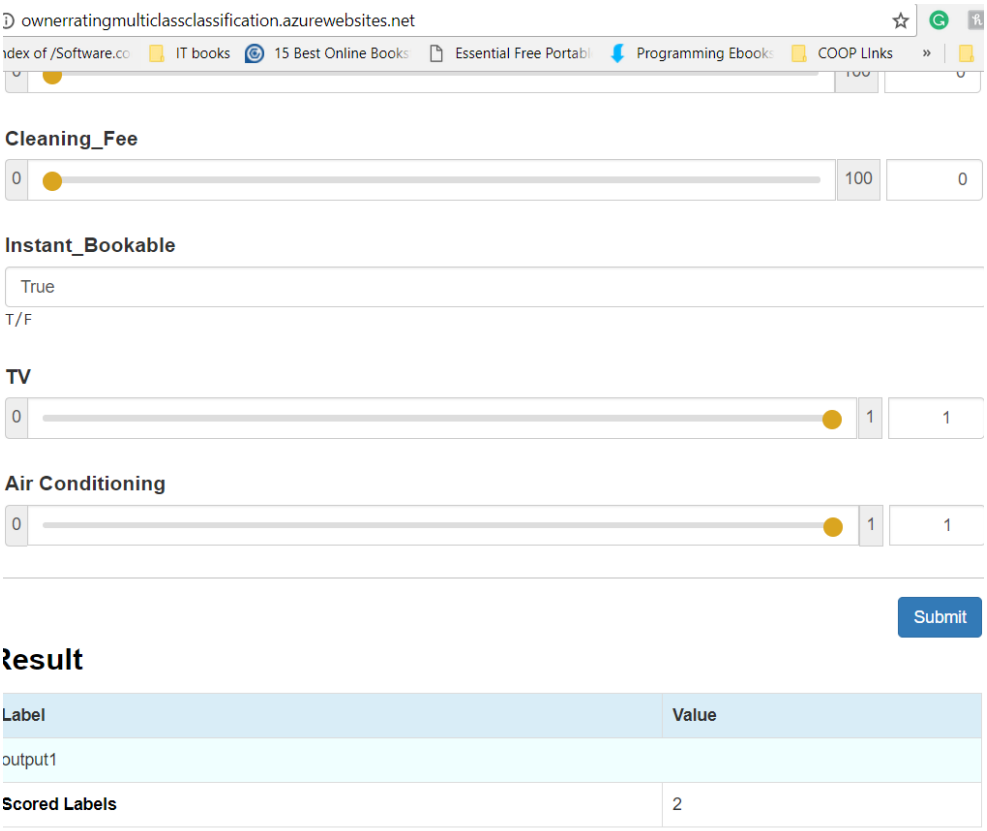


Property Owner:





This will get the owner Rating

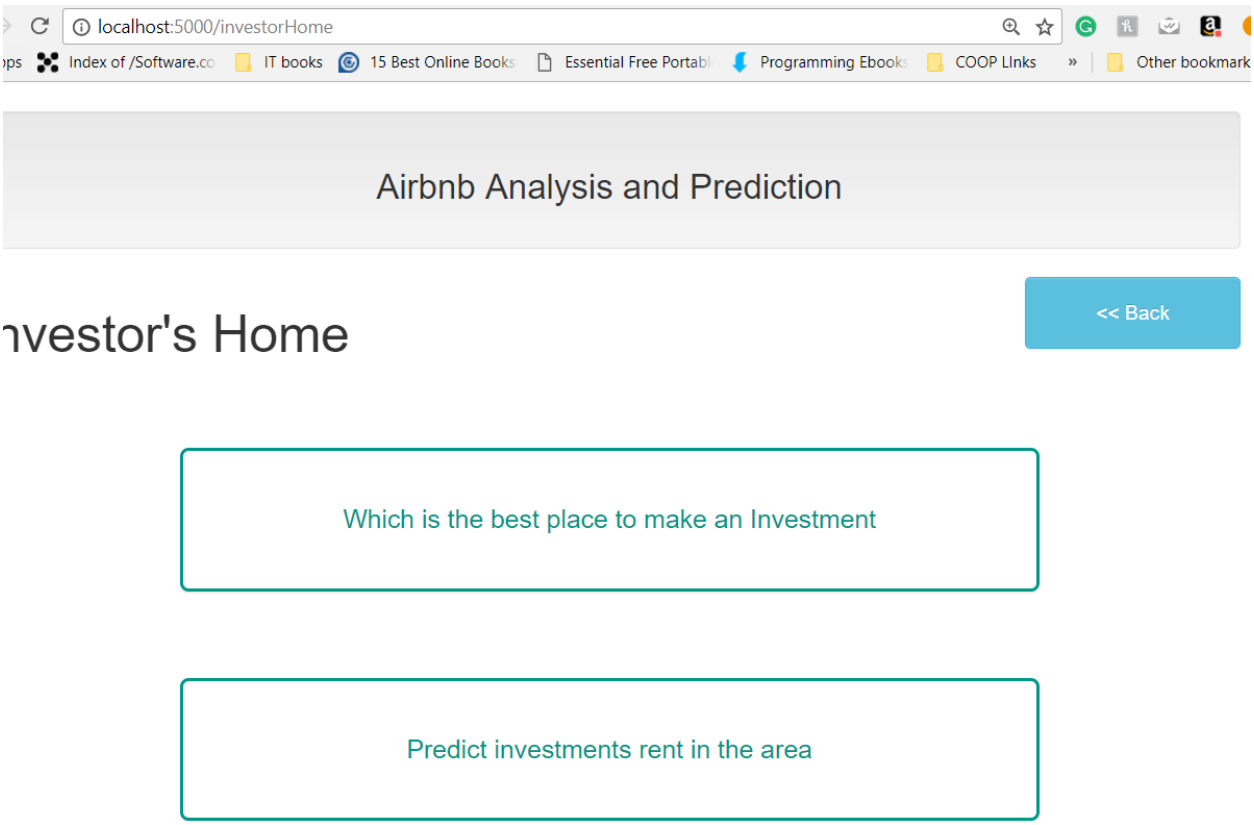


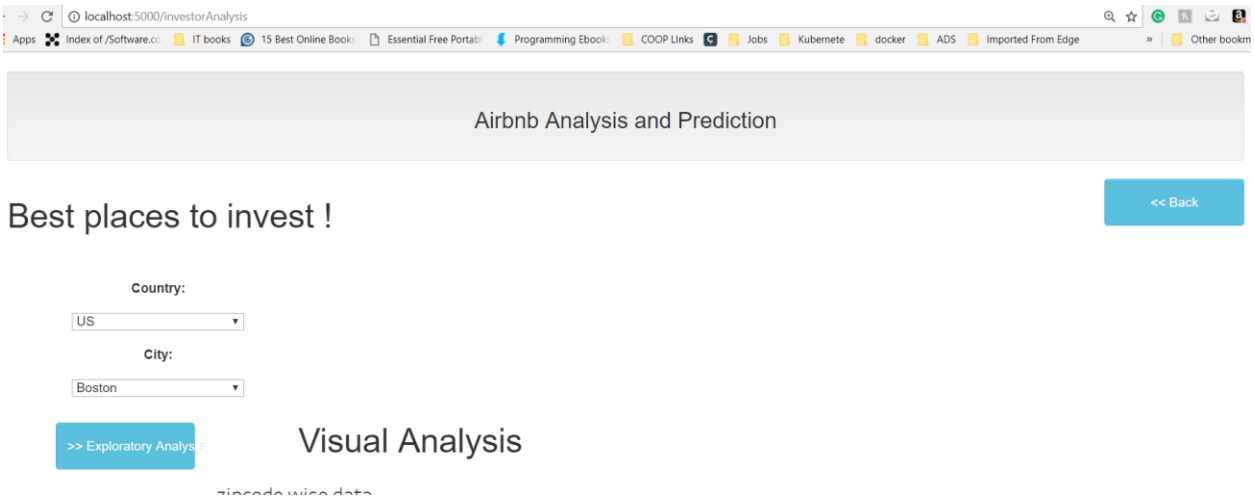
Result

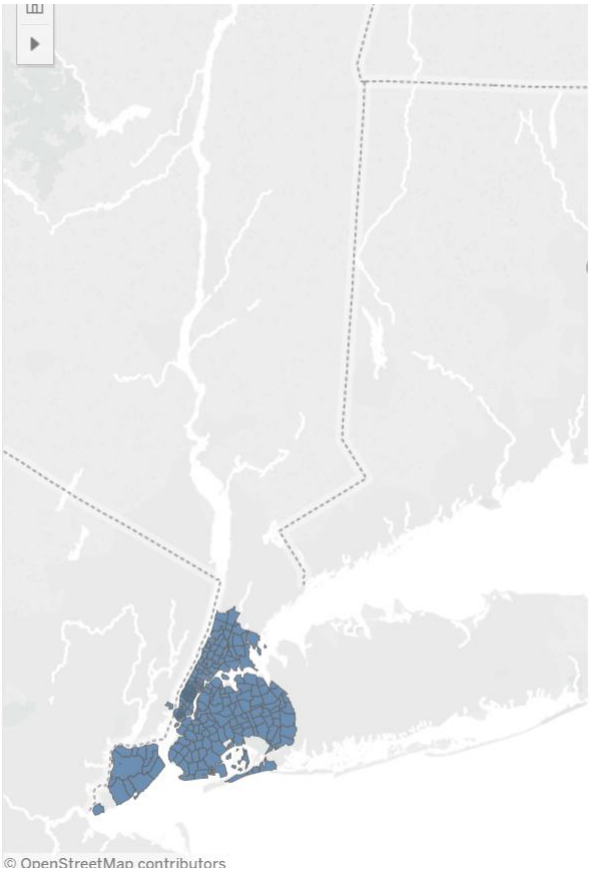
Label	Value
output1	
Scored Labels	2

Here 2 is the Positive Rating

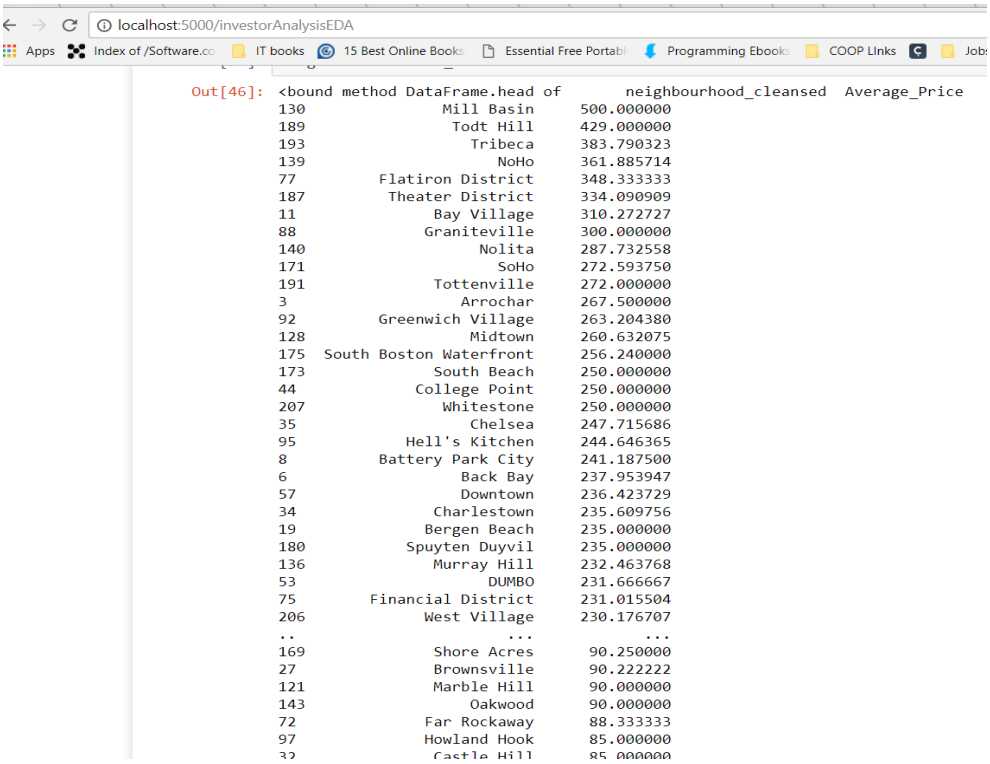
For Investor

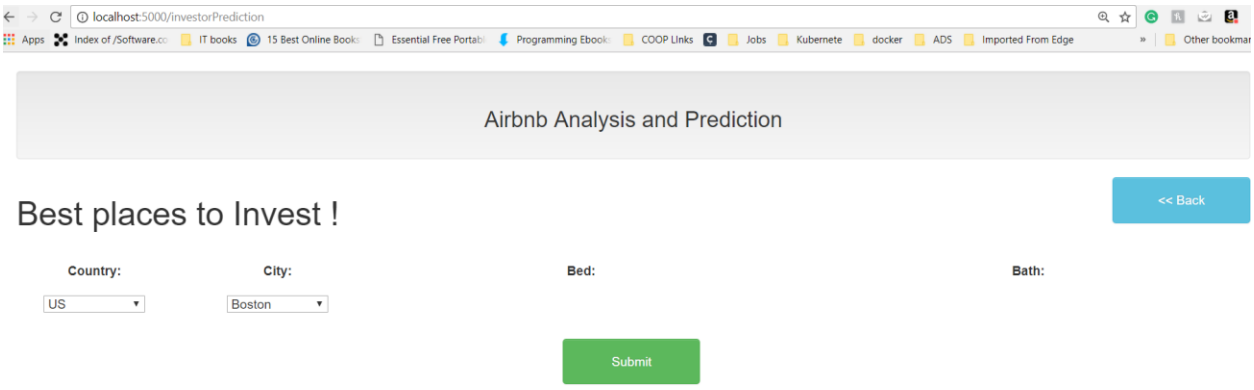

localhost:5000/investorHome
Airbnb Analysis and Prediction
nvestor's Home
<< Back
Which is the best place to make an Investment
Predict investments rent in the area


localhost:5000/investorAnalysis
Airbnb Analysis and Prediction
Best places to invest !
<< Back
Country:
US
City:
Boston
>> Exploratory Analys
Visual Analysis

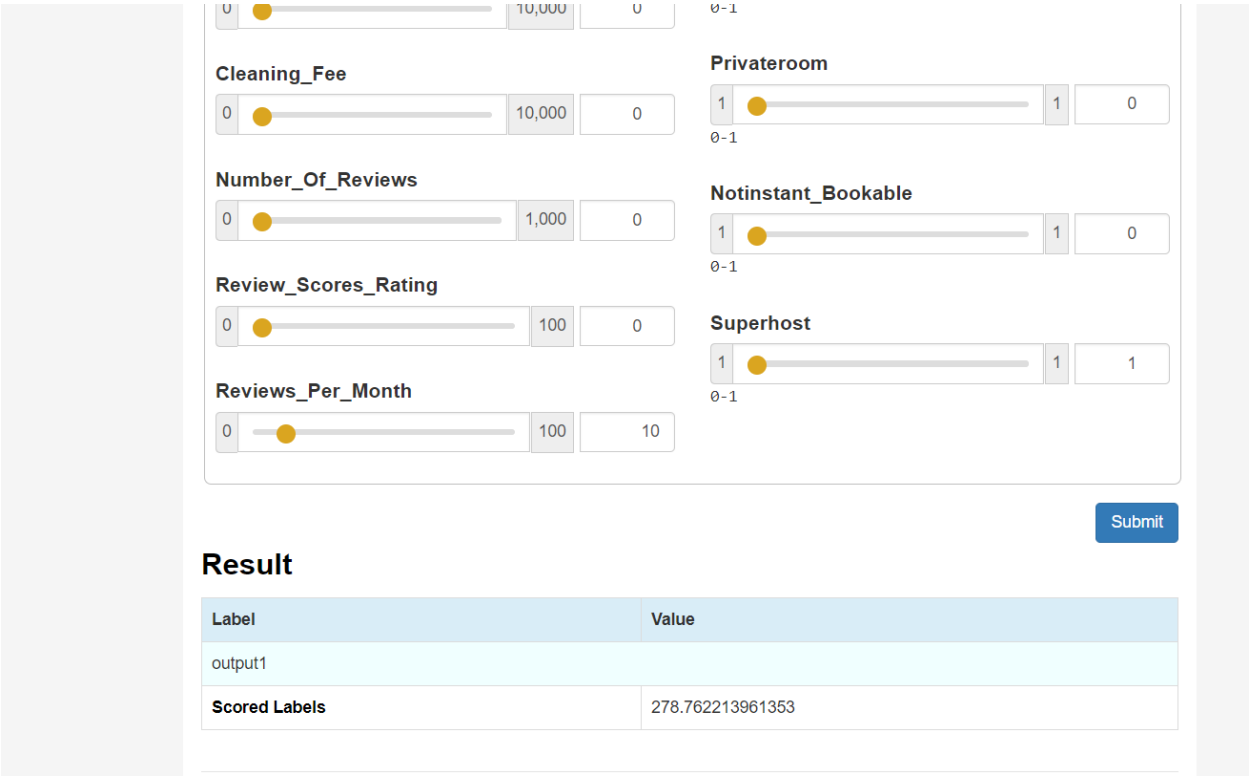


Here we get the analusis of the price ranges listing in the area most costy and leat costly

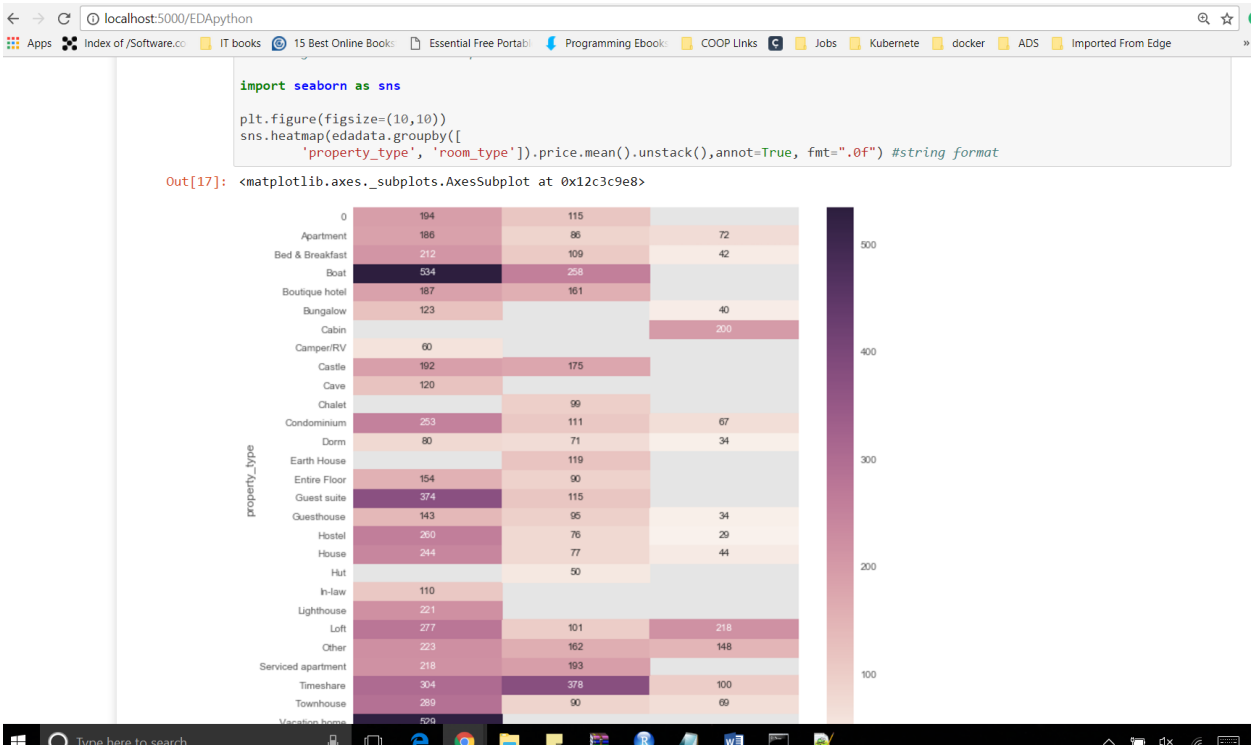




Price for the selected areas



Last [age is the EDA analsysi avaulbe for the current status For boston properties:



Improvement

Improving Reuslt using python scit-learn
Here we find the dataset has many categorical values due to which the RMSE values were bad
So we tried to remove those

```
gorical_feature in ['neighbourhood_cleansed', 'property_type','cancellation_policy',
                    'room_type','instant_bookable','host_is_superhost']:
ures = pd.concat([features, pd.get_dummies(regressiondf[categorical_feature])], axis=1)

.head()
```

dRating	host_total_listings_count	accommodates	bathrooms	beds	price	number_of_reviews	review_scores_rating	host_is_supe
	1.0	4.0	1.5	3.0	250.0	0.0	0	False
	1.0	2.0	1.0	1.0	65.0	36.0	5	False
	1.0	2.0	1.0	1.0	65.0	41.0	5	True
	1.0	2.0	1.5	2.0	79.0	29.0	5	True
	2.0	2.0	1.0	1.0	75.0	8.0	5	True

```
In [71]: #regression
         clf = LinearRegression()
         y = fitters['price']
         clf.fit(fitters.drop('price', axis='columns'), y)

Out[71]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [72]: y_pred = clf.predict(fitters.drop('price', axis='columns'))
         import sklearn.metrics
```

MSE is the square of the average error in each term, while root MSE is its absolute value.

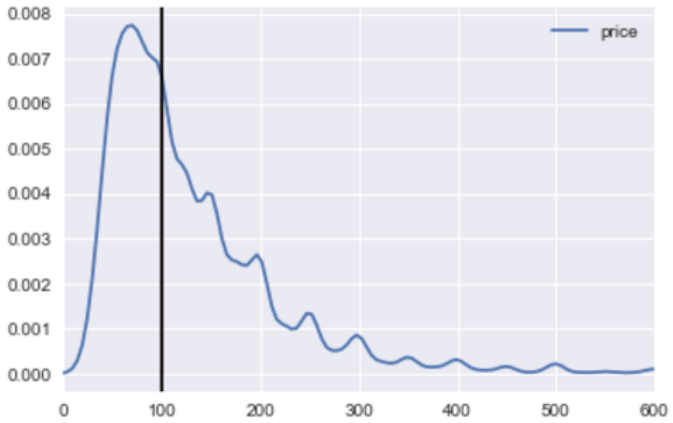
```
In [73]: mse = sklearn.metrics.mean_squared_error(y, y_pred)
         mse

Out[73]: 3195.2695320106332

In [74]: root_mse = mse**(1/2)
         root_mse

Out[74]: 1.0
```

Our RMSE is 1.0 dollars, : meaning that our classifier is wrong by that much on average. So its less HEnce out model is fine

How significant is this with respect to the range of prices we are seeing?

To see that let's plot RMSE as a boundary around the median price.

```
Out[77]: <matplotlib.patches.Rectangle at 0x4234ffd0>
```




```
In [78]: r_squared = sklearn.metrics.r2_score(y, y_pred)
         r_squared

Out[78]: 0.58005177660573826
```

With improved model we get almost 1% of deviation from the mean priced value so it's a better improvement in prediction model with these features.

Summary

1. Here We have performed multiclass classification for rating of Host and we get around 79% of Accuracy in prediction
2. With analysis we found the room type and the host rating are two major factor affects the price of the listing
3. Also for price prediction using linear regression we found the RMSE less but accuracy better than other alorithms so we performed scit-learn using linear regression and imoreved te model a little and get the rmse lesser and more accuracy
4. With analysis we have more propertied in newtyork than boston
5. The most cost efficent houses are the room type than the apartmemt type, it is significant that the cost/bedrooms of the 1 rooms are little more than the double of the Apartment, so it would be interesting to search houses with apartment type rooms instead of houses with a single room.
6. In Boston property analysis we found most of the investment in roomttupe property is valueable and can have better return.
7. Also Boston listing for low budget then he should avoid visiting during September and October as these are the times when average price of listings are fairly high as compared to the other months.
8. If the taveller is on low budget then he should avoid visiting Boston on Weekends as prices of listings on weekend are higher than that of weekdays.