CS5661
SPRING 19
CAL STATE LA

RAVI AMIN
SHARDUL DAVE
RUTVIBEN PATEL
KRUTI SHAH

INSIDE AIRBNB NYC

CONTENTS

1. Introduction

Airbnb is a growing industry in which many people have heard of or previously used themselves. Are the hosts properly valuing their rentals or are renters paying too much or too little for their chosen rental property? Airbnb is an online marketplace and lodging service that allows people to rent their properties for short term leases or vacation rentals. The rental property cost is determined by the host and the company receives a percentage of the stay as a service fee. Airbnb has a predictive tool for hosts to utilize in order to set a competitive price for their rental property. The tool is based on a description of the property, but the host ultimately determines the price. The goal this study is to create a predictive model based off Airbnb data to better understand how hosts actually price their property rentals.

About the Dataset

The dataset contains house rental prices for New York City. It includes home rented in the past years. We will use different algorithm to predict the price

- Row: 44317
- Column: 96
- Size: 161MB
- Dataset Link: https://www.kaggle.com/peterzhou/airbnb-open-data-in-nyc
- (listings.csv): Detailed listing data, including various attributes (features) of each listing such as number of bedrooms, bathrooms, location etc.

2. Objectives

The goal of our study is to build a predictive model that explains how renters value their properties and set their rental prices in New York City. In order to reach this goal the necessary objectives include:

- Analyze and understand data for rental prices
- Identify features and labels for Airbnb rental prices
- Use different models and trained them to predict a house price RMSE
- Finding the best features amongst the all.

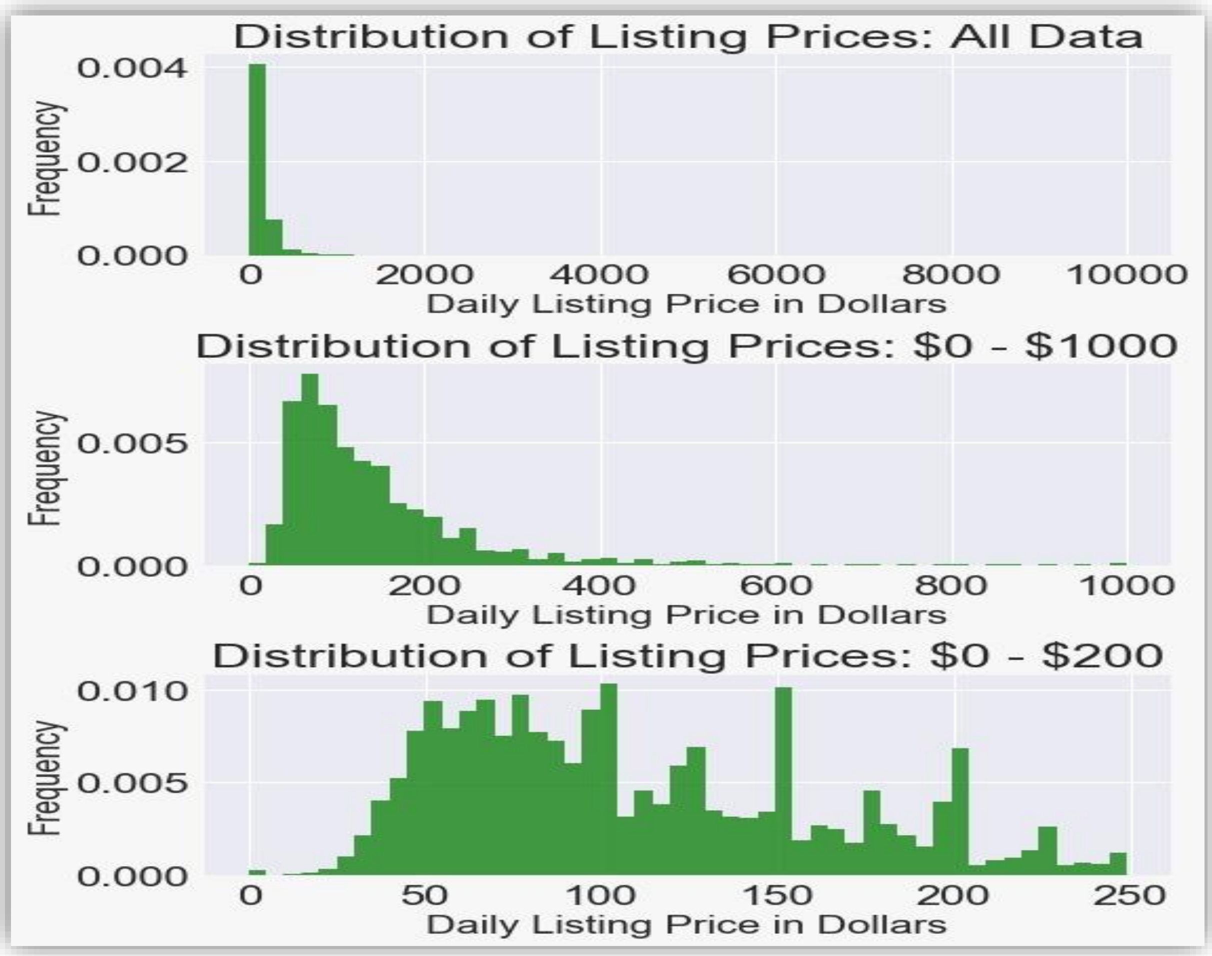
3. Price Visualization

We visualize the price table, changing them to floats and replacing the commas with a blank. We distribute price listings into 3 different category to obtain the frequency.

Get frequency of Price for listings:

1) Distribution of Listing Prices: All Data
2) Distribution of Listing Prices: $0 - $1000
3) Distribution of Listing Prices: $0 - $200

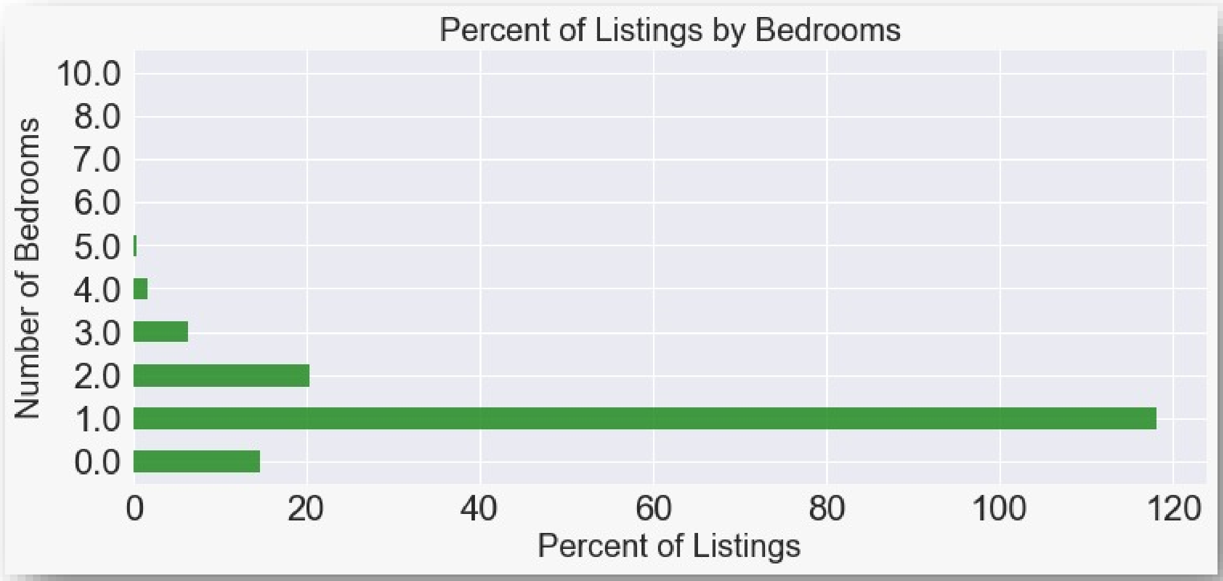
Most house prices are between $0 and $200 per day in NYC.



We use following code for price visualization:

```
def plot_hist(n, titles, ranges):

   fig, ax = plt.subplots(n, figsize = (8, 10.5))

   for i in range(n):

      d, bins, patches = ax[i].hist(ranges[i], 50, normed = 1, color= BNB_GREEN, alpha = 0.85)

      ax[i].set_title(titles[i])

      ax[i].set_xlabel("Daily Listing Price in Dollars", fontsize=20)

      ax[i].set_ylabel("Frequency", fontsize=20)

   plt.tight_layout()

   plt.show()#Visualize price table, changing them to floats and replacing the commas with
a blank

prices = df['price'].apply(lambda x:float(str(x).replace(',','').replace('$','')))

plot_hist(3, ['Distribution of Listing Prices: All Data', 'Distribution of Listing Prices: \$0 -
\$1000',

           'Distribution of Listing Prices: \$0 - \$200'], [prices, prices[prices <= 1000],
prices[prices < 250]])
```

Get frequency of bedroom number for listings



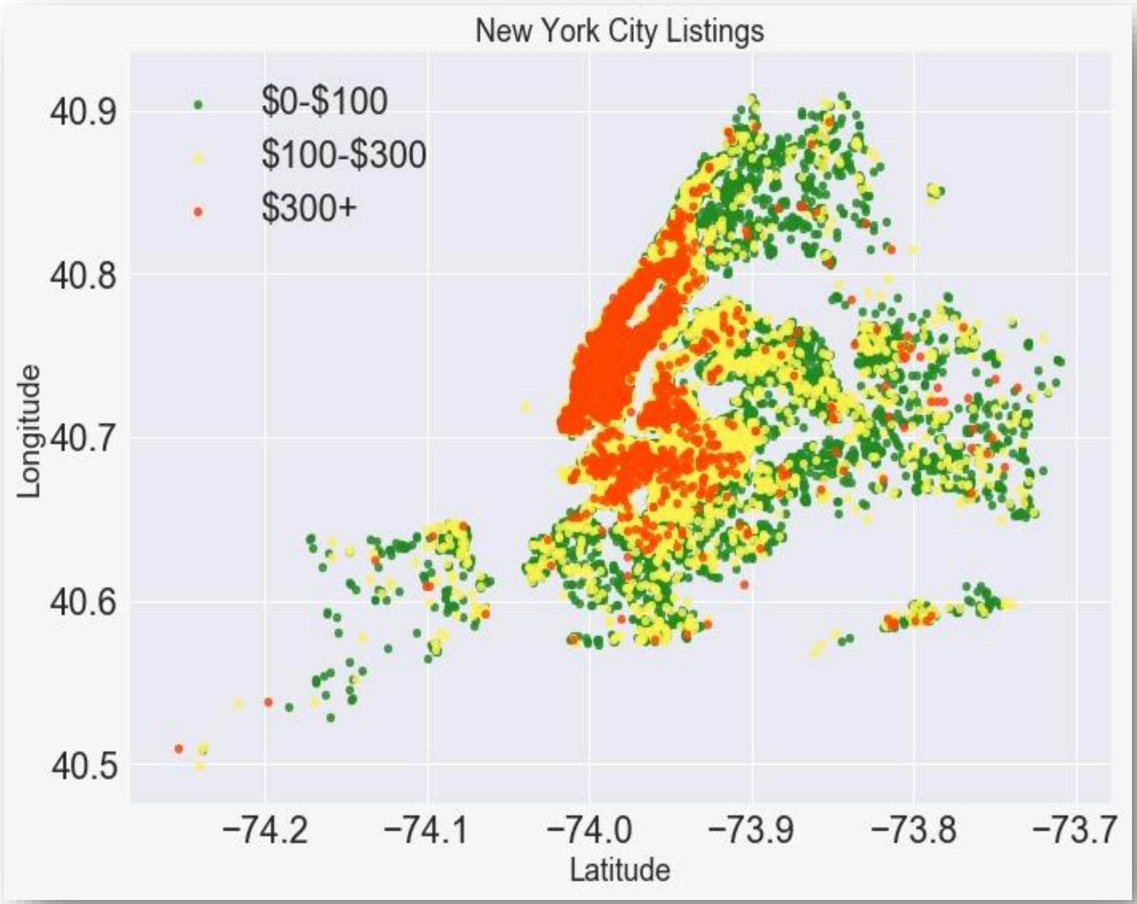
We use following code to get percent of listings using bedrooms

```
# Get frequency of bedroom number for listings

bedrooms_counts = Counter(df.bedrooms)

tdf = pd.DataFrame.from_dict(bedrooms_counts, orient = 'index').sort_values(by = 0)

tdf = (tdf.iloc[-10:, :] / 27392) * 100

# Sort bedroom dataframe by number

tdf.sort_index(axis = 0, ascending = True, inplace = True)

# Plot percent of listings by bedroom number

ax = tdf.plot(kind = 'barh', figsize = (12, 5), color = BNB_GREEN, alpha = 0.85)

ax.set_xlabel("Percent of Listings", fontsize=20)

ax.set_ylabel("Number of Bedrooms", fontsize=20)

ax.set_title('Percent of Listings by Bedrooms', fontsize=20)

ax.legend_.remove()

plt.show()

print ("Percent of 1 Bedroom Listings: %{0:.2f}".format(tdf[0][1]))
```

Plot listings on scatterplot by using Latitude and Longitude

For this we divide price into 3 different category:

1) Distribution of Listing Prices: $0 - $100
2) Distribution of Listing Prices: $100 - $300
3) Distribution of Listing Prices: $300 +



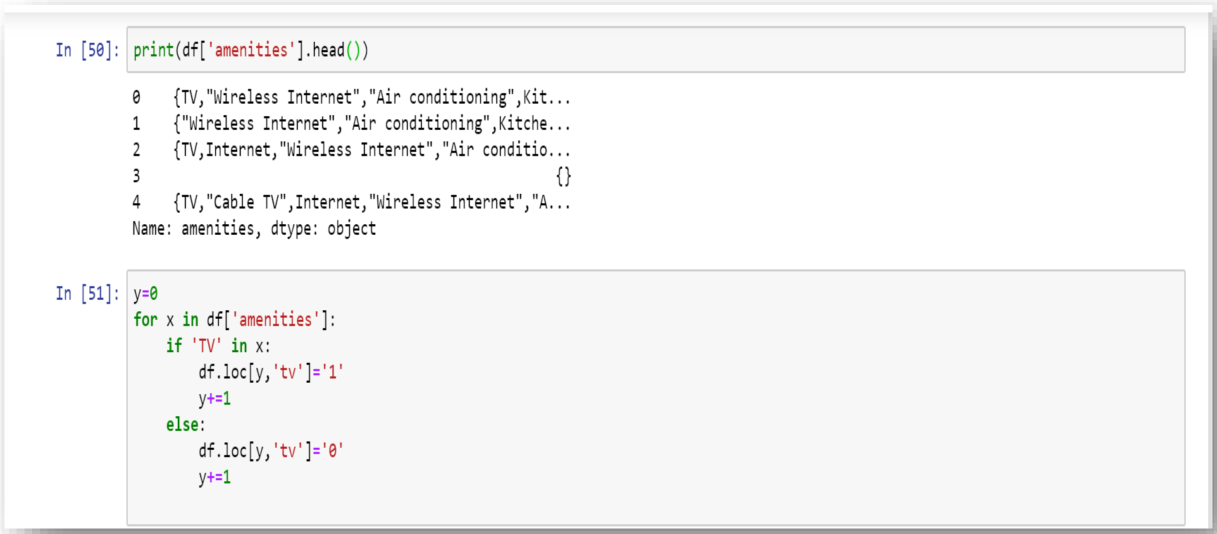
We use following code to plot listings on scatterplot

```
# Outline price buckets

intervals = [0,100,300, 10000]

leg_labels = []

# Get Labels for legend

for i in range(0,len(intervals) - 1):

    if i == len(intervals) - 2:

        leg_labels.append('\${}+'.format(intervals[i]))

    else:

        leg_labels.append("\${}-\${}".format(intervals[i], intervals[i+1]))

buckets = []

for i in range(0, len(intervals) - 1):

    buckets.append(df[(prices > intervals[i]) & (prices < intervals[i+1])])

colors = [BNB_GREEN, BNB_LEMON, BNB_ORANGE]

alphas = [0.85, 0.85, 0.85]

# Plot listings on scatterplot

plt.figure(figsize=(11,8))

for i in range(0, len(buckets)):

     plt.scatter(buckets[i]['longitude'], buckets[i]['latitude'], alpha = alphas[i], c=colors[i],
s=25)

    plt.title('New York City Listings', fontsize=18)

plt.xlabel('Latitude', fontsize=18)

plt.ylabel('Longitude', fontsize=18)

plt.legend(labels=leg_labels, loc = 'best')

plt.show()
```

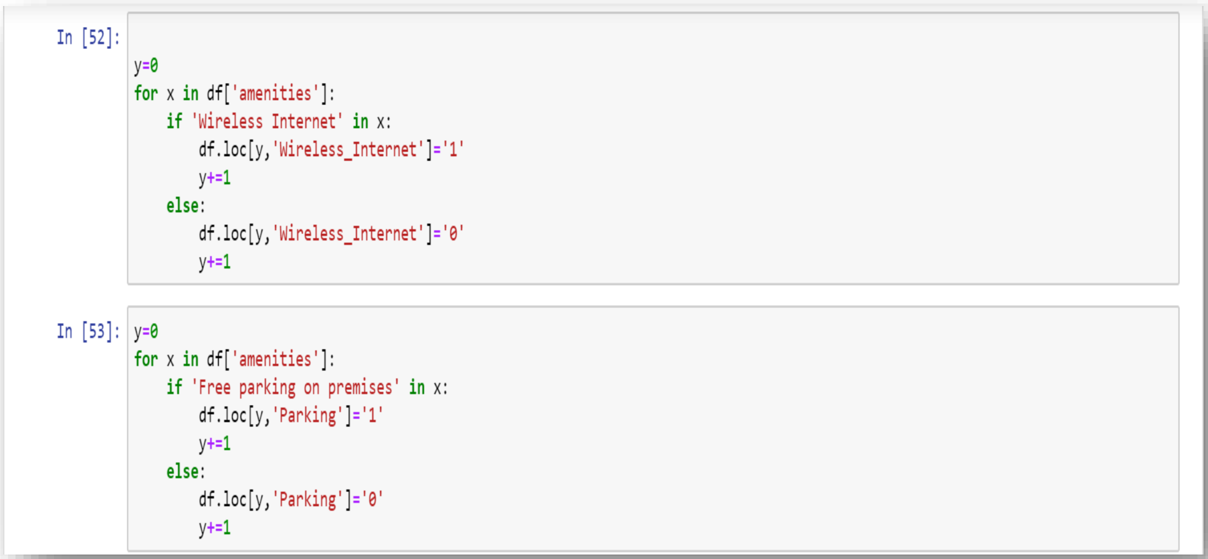
4. Define A Function To Convert Categorical Feature Into Numerical Feature

We use the following two methods in order to convert strings to floats in pandas Data Frame:

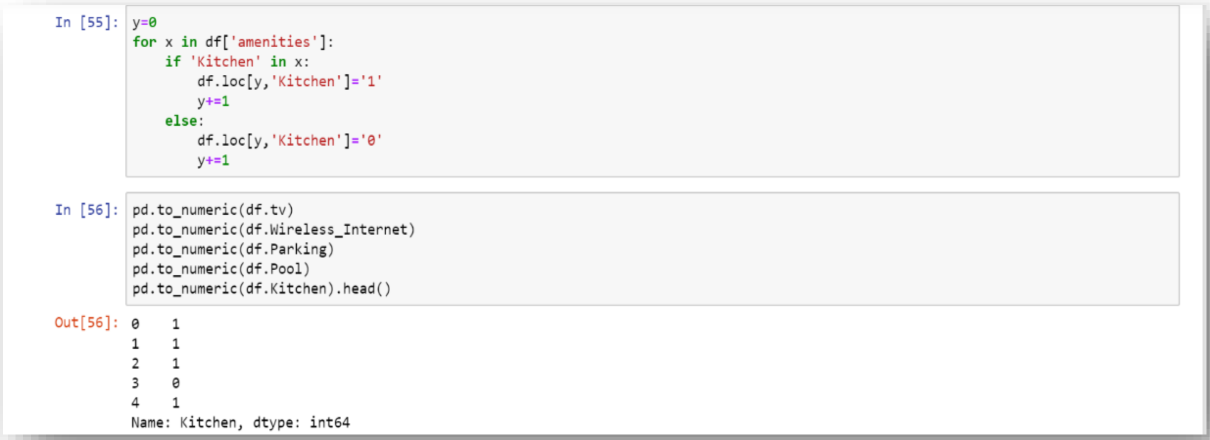
A. astype(float) method
B. to_numeric method



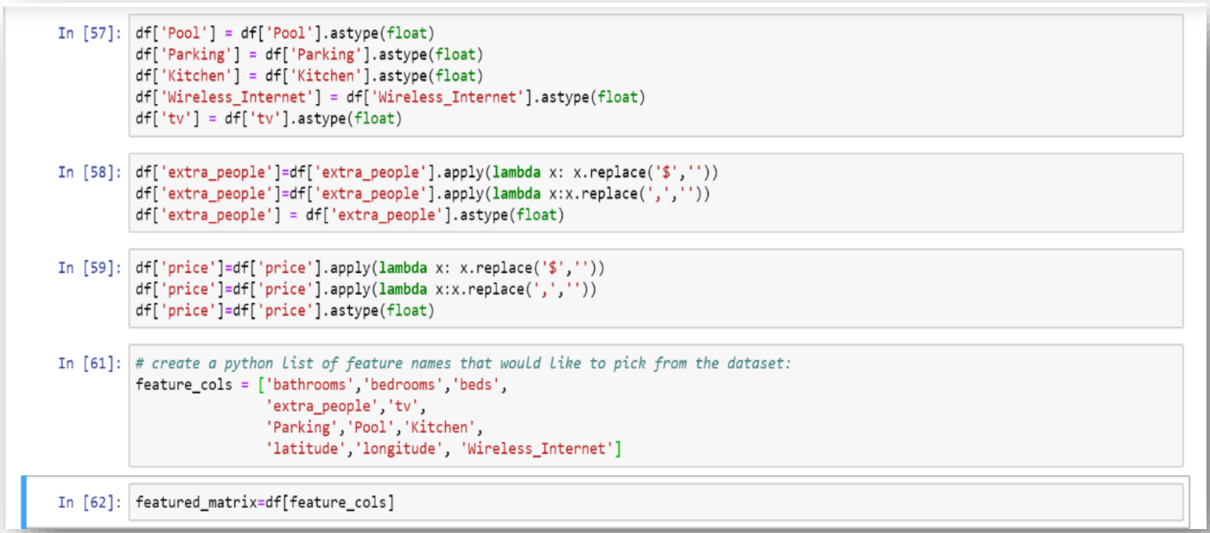
```
In [50]: print(df['amenities'].head())

0    {TV,"Wireless Internet","Air conditioning",Kit...
1    {"Wireless Internet","Air conditioning",Kitche...
2    {TV,Internet,"Wireless Internet","Air conditio...
3                                                   {}
4    {TV,"Cable TV",Internet,"Wireless Internet","A...
Name: amenities, dtype: object

In [51]: y=0
         for x in df['amenities']:
             if 'TV' in x:
                 df.loc[y,'tv']='1'
                 y+=1
             else:
                 df.loc[y,'tv']='0'
                 y+=1
```



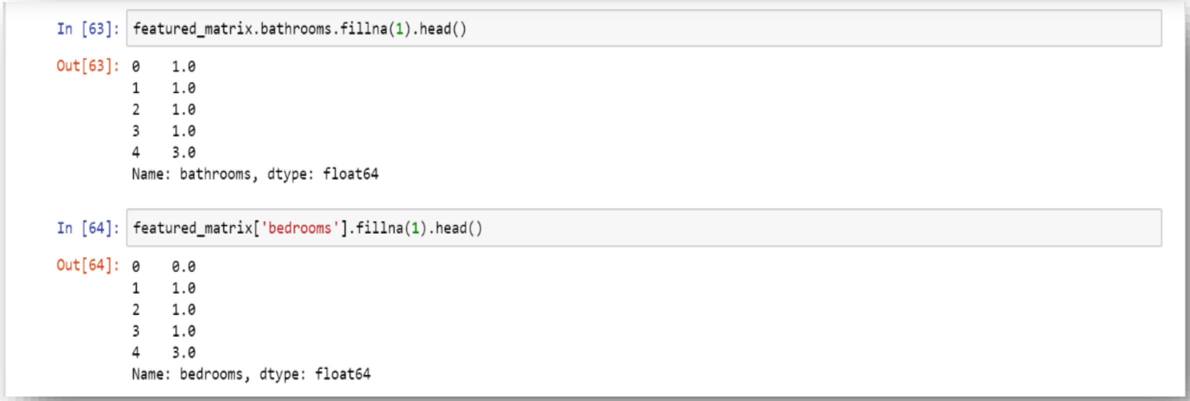
```
In [52]: 
         y=0
         for x in df['amenities']:
             if 'Wireless Internet' in x:
                 df.loc[y,'Wireless_Internet']='1'
                 y+=1
             else:
                 df.loc[y,'Wireless_Internet']='0'
                 y+=1

In [53]: y=0
         for x in df['amenities']:
             if 'Free parking on premises' in x:
                 df.loc[y,'Parking']='1'
                 y+=1
             else:
                 df.loc[y,'Parking']='0'
                 y+=1
```



```
In [55]: y=0
         for x in df['amenities']:
             if 'Kitchen' in x:
                 df.loc[y,'Kitchen']='1'
                 y+=1
             else:
                 df.loc[y,'Kitchen']='0'
                 y+=1

In [56]: pd.to_numeric(df.tv)
         pd.to_numeric(df.Wireless_Internet)
         pd.to_numeric(df.Parking)
         pd.to_numeric(df.Pool)
         pd.to_numeric(df.Kitchen).head()

Out[56]: 0    1
         1    1
         2    1
         3    0
         4    1
         Name: Kitchen, dtype: int64
```



```
In [57]: df['Pool'] = df['Pool'].astype(float)
         df['Parking'] = df['Parking'].astype(float)
         df['Kitchen'] = df['Kitchen'].astype(float)
         df['Wireless_Internet'] = df['Wireless_Internet'].astype(float)
         df['tv'] = df['tv'].astype(float)

In [58]: df['extra_people']=df['extra_people'].apply(lambda x: x.replace('$',''))
         df['extra_people']=df['extra_people'].apply(lambda x:x.replace(',',''))
         df['extra_people'] = df['extra_people'].astype(float)

In [59]: df['price']=df['price'].apply(lambda x: x.replace('$',''))
         df['price']=df['price'].apply(lambda x:x.replace(',',''))
         df['price']=df['price'].astype(float)

In [61]: # create a python list of feature names that would like to pick from the dataset:
         feature_cols = ['bathrooms','bedrooms','beds',
                         'extra_people','tv',
                         'Parking','Pool','Kitchen',
                         'latitude','longitude', 'Wireless_Internet']

In [62]: featured_matrix=df[feature_cols]
```

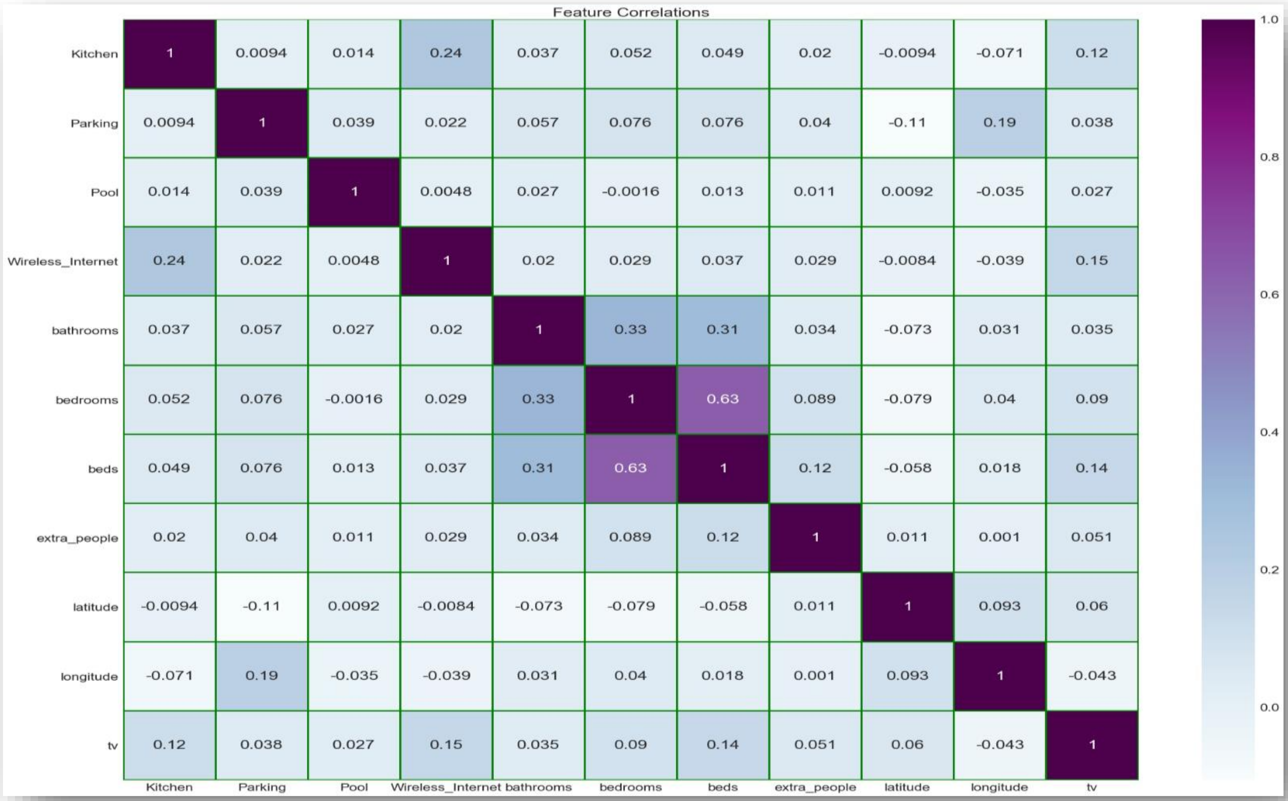


```
In [63]: featured_matrix.bathrooms.fillna(1).head()

Out[63]: 0    1.0
         1    1.0
         2    1.0
         3    1.0
         4    3.0
         Name: bathrooms, dtype: float64

In [64]: featured_matrix['bedrooms'].fillna(1).head()

Out[64]: 0    0.0
         1    1.0
         2    1.0
         3    1.0
         4    3.0
         Name: bedrooms, dtype: float64
```

5. Correlation Between Features

We then run the correlation heatmap on the dataset to see if any of the features are particularly correlated. We evaluating the correlation between all the features, the The “corr()” method includes the correlation of each feature with itself, which is always 1, the “corr()” is very easy to use and very powerful for the early stages of data analysis (data preparation), by doing a graph of its results using matplotlib, we get a better idea of the data so you can make decisions for the next steps of data preparation and data analysis.

The correlation of each feature with itself, which is always 1, so that is why this type of graph always has the magenta diagonal from the upper left to the lower right. Other than the diagonal, the rest of the squares show correlation between different features, making it really easy to find that “bedrooms” and “bed” are highly correlated have a correlation of about 0.63, “bedrooms” and “bathrooms” are highly correlated, “bathrooms” and “beds” have a correlation of about 0.31.



We use following code to define correlation between different features

```
import seaborn as sns

import matplotlib.pyplot as plt

sns.set(font_scale=2.2)

str_list = []

for colname, colvalue in featured_matrix.iteritems():

    if type(colvalue)==str:

        str_list.append(colname)

num_list = featured_matrix.columns.difference(str_list)

house_num = featured_matrix[num_list]

f,ax = plt.subplots(figsize = (35,30))

plt.title('Feature Correlations', fontsize = 25)

sns.heatmap(house_num.astype(float).corr(),linewidths=2.0,vmax=1.0, square = False,

            cmap = 'BuPu',linecolor = 'G', annot = True)

plt.show()
```

6. Models:

A. Variable Selection

We build the feature matrix and label vector. Feature matrix includes bathrooms, bedrooms, beds, extra_people, tv, Parking, Pool, Kitchen, latitude, longitude and Wireless_Internet. We consider 'Price' as a Label vector. These variables were then used as a reference to compare the outputs of the four models conducted.

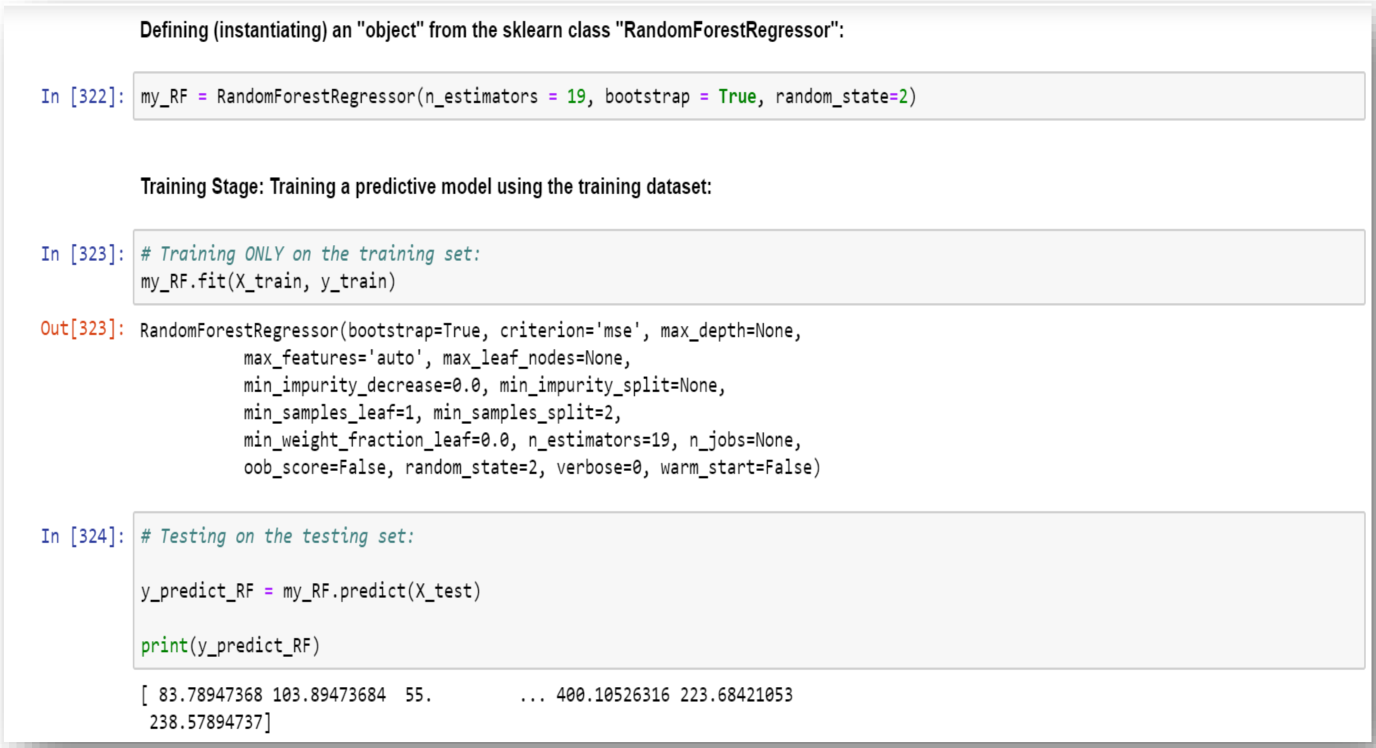
We use sklearn functions to split the dataset into testing and training sets with the following parameters: test_size=0.2, random_state=3. we randomly split the training dataset into 0.8-0.2 train-test ratio, and then run the following method:

```
X_train, X_test, y_train, y_test = train_test_split(featured_matrix,
label_vector, test_size=0.2, random_state=3)
```

In this project, we import 4 different regression models from Sklearn library — Random forest, Linear regression, Polynomial regressor and Logistic regression. We also want to use the ANN Regressor to be a secondary prediction method, so that we don't just use the models from one library. To verify our prediction, we use RMSE (Root Mean Square Error) as our indicator for how good our model works.

B. Model 1: Random Forest

We use following method, We use following method, to predict the result and calculate RMSE, and compare the RMSE between testing set and the training set, and compare amongst different regression models.

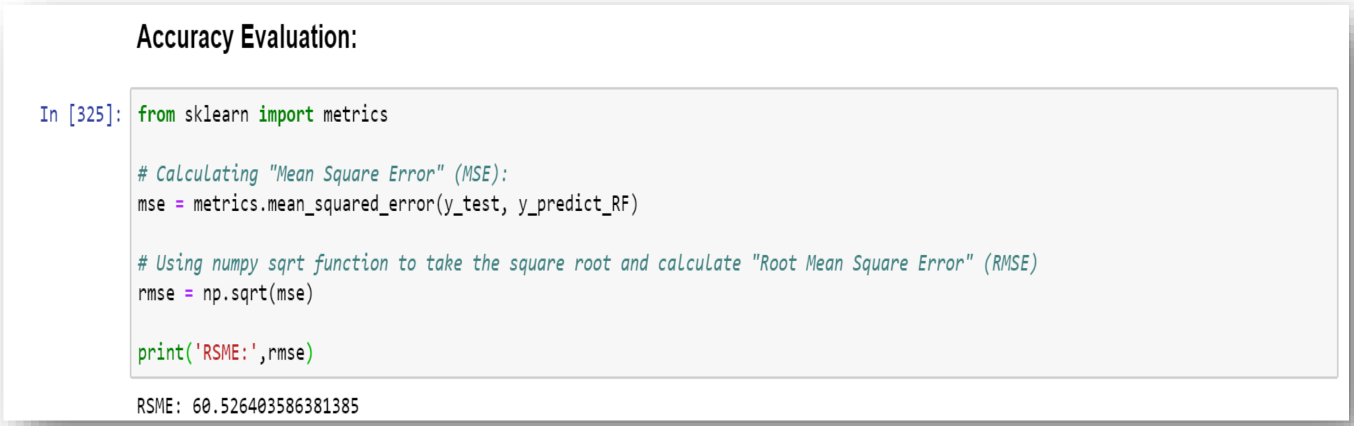


Defining (instantiating) an "object" from the sklearn class "RandomForestRegressor":

```
In [322]: my_RF = RandomForestRegressor(n_estimators = 19, bootstrap = True, random_state=2)
```

Training Stage: Training a predictive model using the training dataset:

```
In [323]: # Training ONLY on the training set:
          my_RF.fit(X_train, y_train)

Out[323]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=19, n_jobs=None,
                     oob_score=False, random_state=2, verbose=0, warm_start=False)

In [324]: # Testing on the testing set:

          y_predict_RF = my_RF.predict(X_test)

          print(y_predict_RF)

          [ 83.78947368 103.89473684  55.         ... 400.10526316 223.68421053
           238.57894737]
```

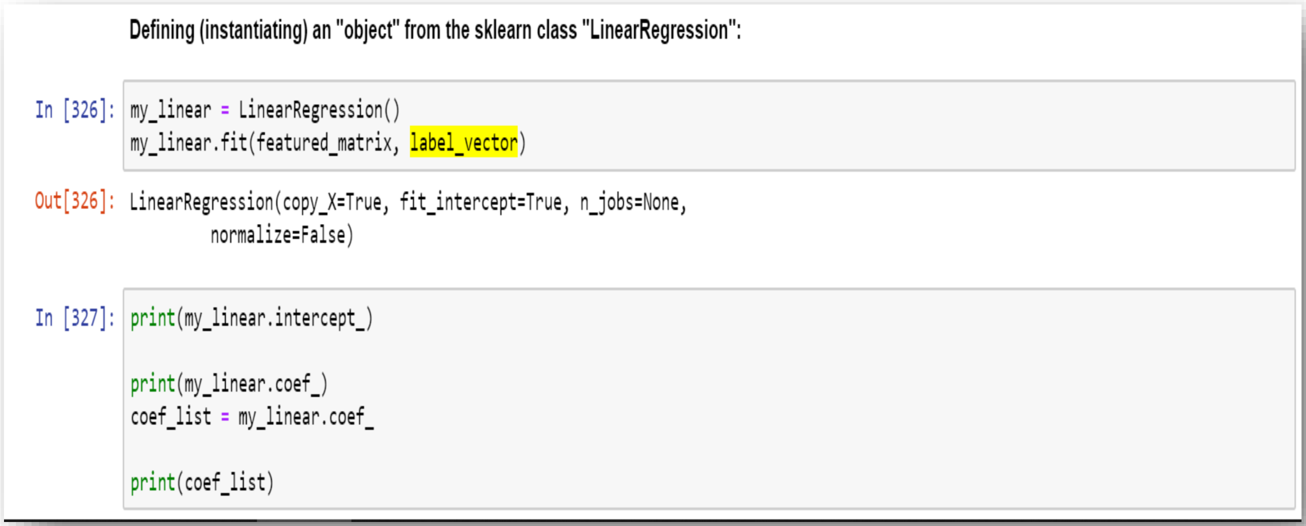


Accuracy Evaluation:

```
In [325]: from sklearn import metrics

          # Calculating "Mean Square Error" (MSE):
          mse = metrics.mean_squared_error(y_test, y_predict_RF)

          # Using numpy sqrt function to take the square root and calculate "Root Mean Square Error" (RMSE)
          rmse = np.sqrt(mse)

          print('RSME:',rmse)

          RSME: 60.526403586381385
```

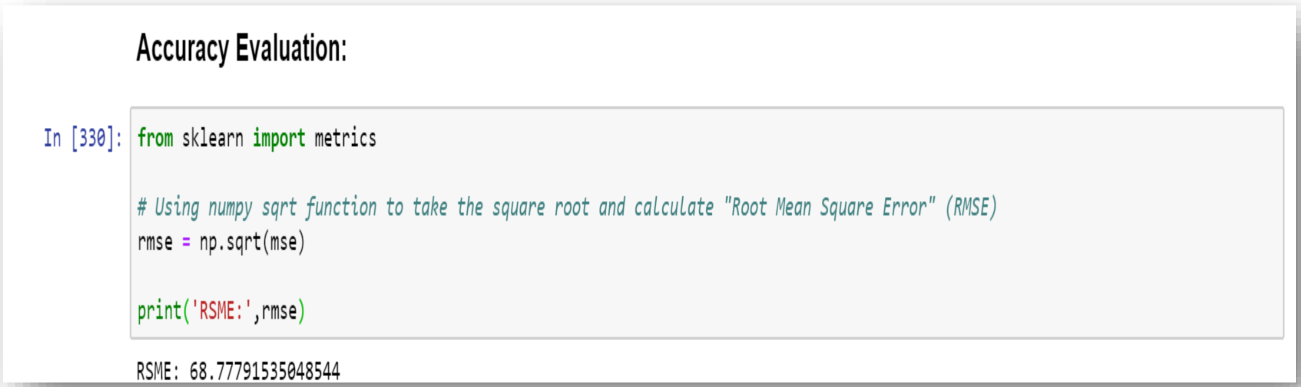
C. Model 2: Linear Regression

We use following method, to predict the result and calculate RMSE.



Defining (instantiating) an "object" from the sklearn class "LinearRegression":

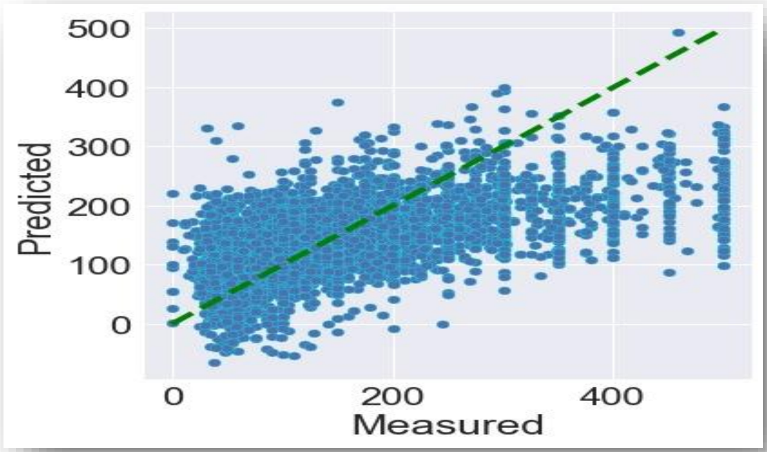
```
In [326]: my_linear = LinearRegression()
          my_linear.fit(featured_matrix, label_vector)

Out[326]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                   normalize=False)

In [327]: print(my_linear.intercept_)

          print(my_linear.coef_)
          coef_list = my_linear.coef_

          print(coef_list)
```



Accuracy Evaluation:

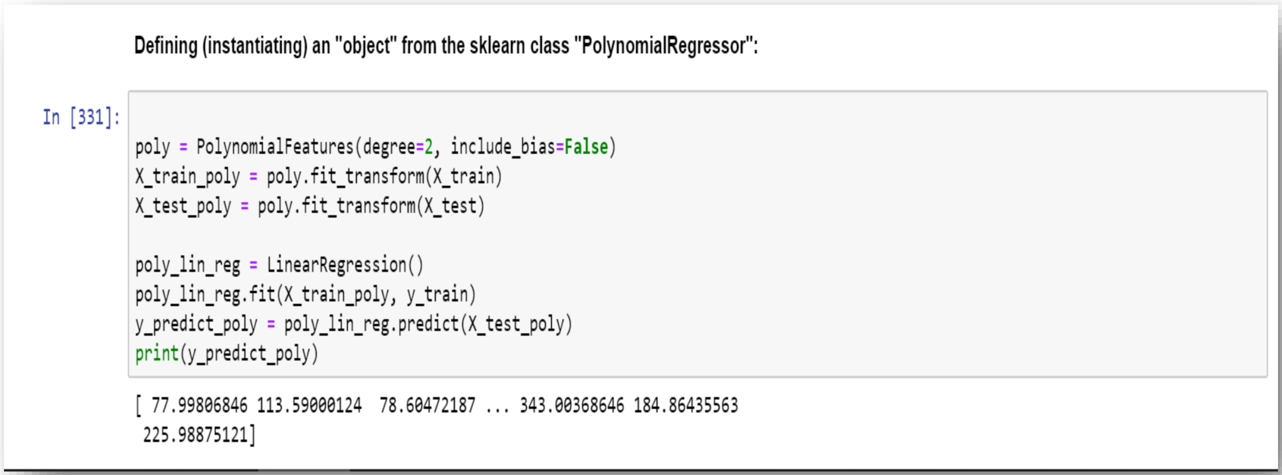
```
In [330]: from sklearn import metrics

          # Using numpy sqrt function to take the square root and calculate "Root Mean Square Error" (RMSE)
          rmse = np.sqrt(mse)

          print('RSME:',rmse)

          RSME: 68.77791535048544
```

Linear plot was then conducted to measure for model assumptions.



D. Model 3: Polynomial Regressor

We use following method, to predict the result and calculate RMSE.

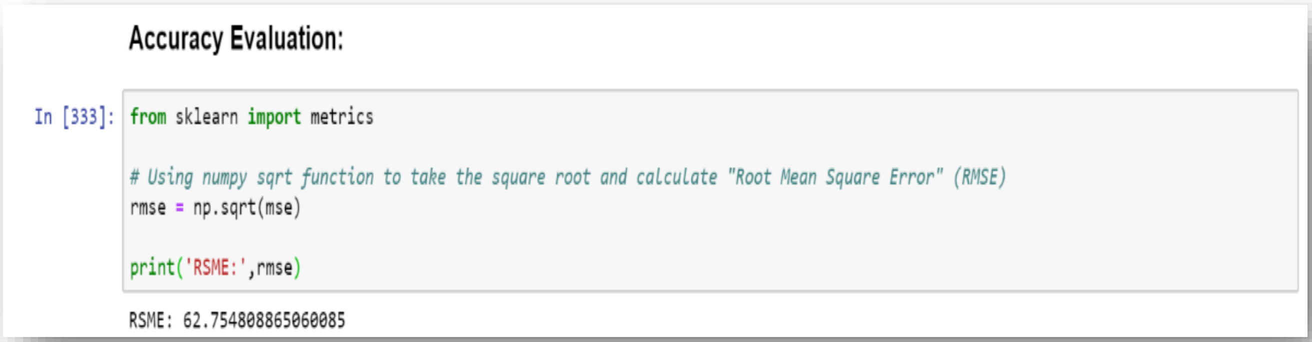


Defining (instantiating) an "object" from the sklearn class "PolynomialRegressor":

In [331]:

```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.fit_transform(X_test)

poly_lin_reg = LinearRegression()
poly_lin_reg.fit(X_train_poly, y_train)
y_predict_poly = poly_lin_reg.predict(X_test_poly)
print(y_predict_poly)
```

```
[ 77.99806846 113.59000124  78.60472187 ... 343.00368646 184.86435563
 225.98875121]
```

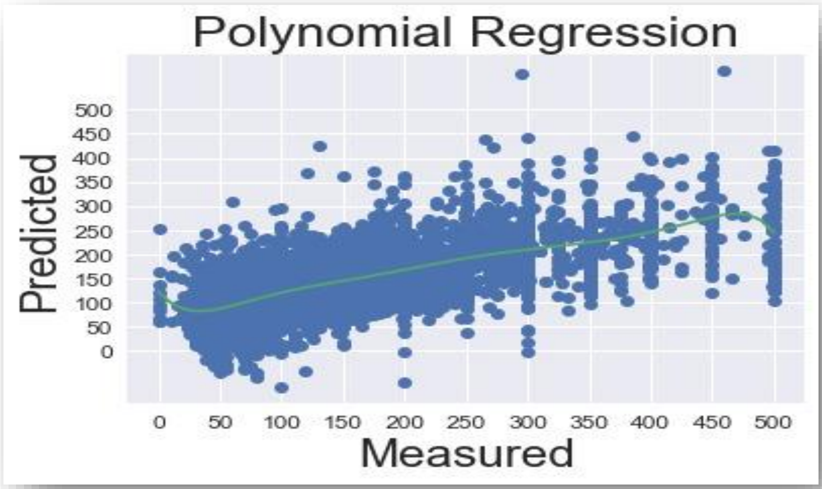


Accuracy Evaluation:

In [333]:
```python
from sklearn import metrics

# Using numpy sqrt function to take the square root and calculate "Root Mean Square Error" (RMSE)
rmse = np.sqrt(mse)

print('RSME:',rmse)
```

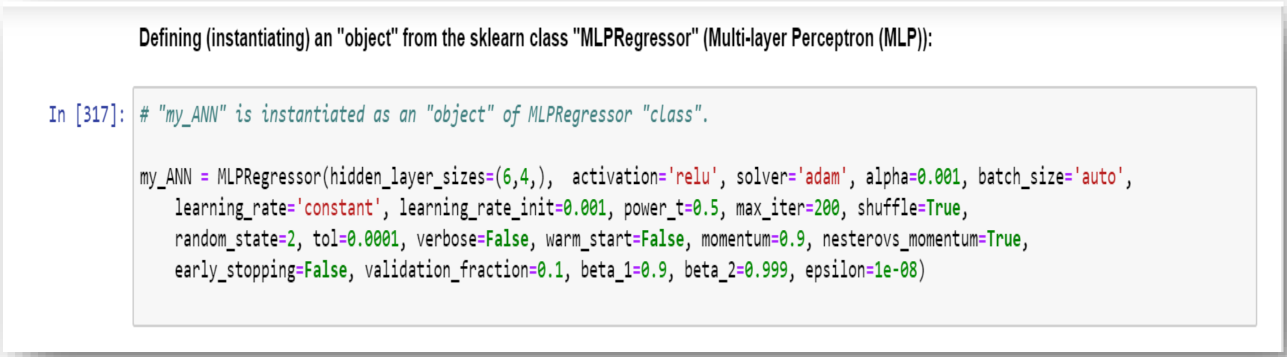
```
RSME: 62.754808865060085
```

Plot was then conducted to measure for model assumptions.



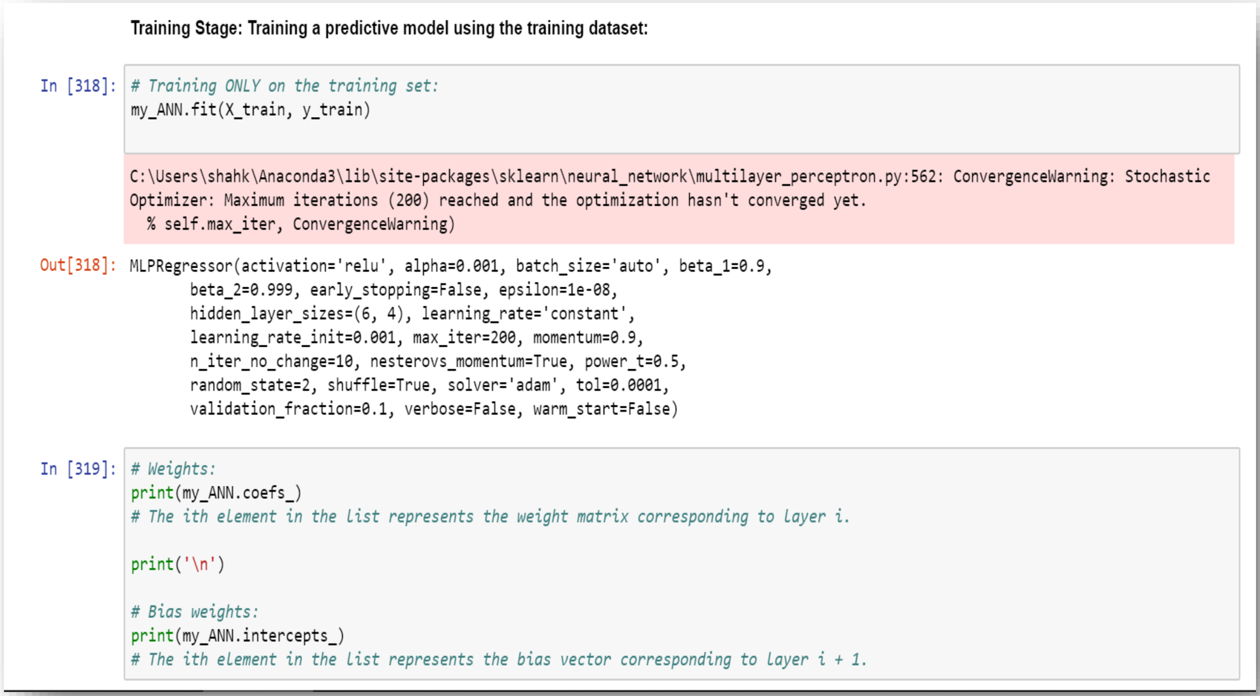
F. Model 5: ANN Regression

We use ANN Regressor to run the model and compare the result. This method provides a RMSE which is higher than some of the regression models. We got 69.64 RMSE using ANN Regressor.



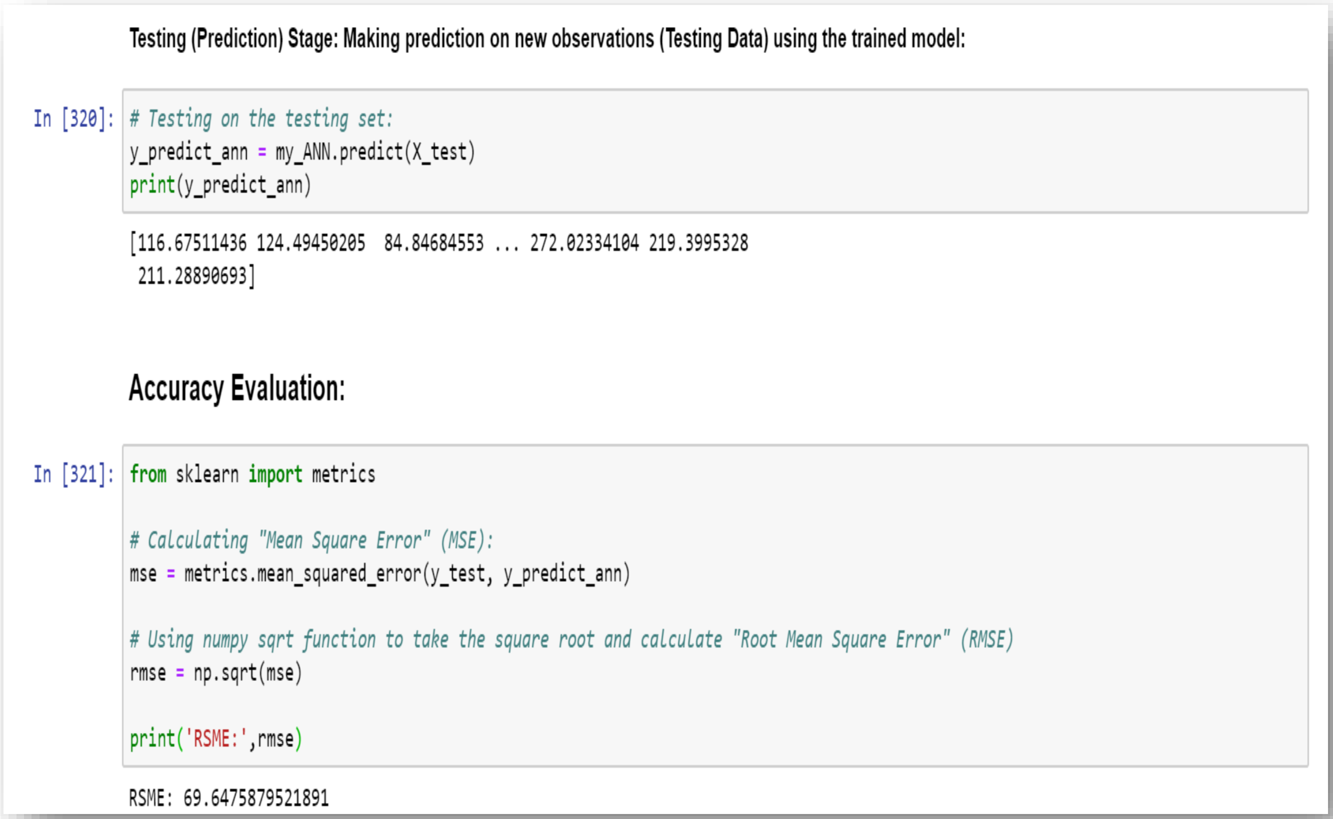
Defining (instantiating) an "object" from the sklearn class "MLPRegressor" (Multi-layer Perceptron (MLP)):

```
In [317]: # "my_ANN" is instantiated as an "object" of MLPRegressor "class".

          my_ANN = MLPRegressor(hidden_layer_sizes=(6,4,),  activation='relu', solver='adam', alpha=0.001, batch_size='auto',
              learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
              random_state=2, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
              early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```



Training Stage: Training a predictive model using the training dataset:

```
In [318]: # Training ONLY on the training set:
          my_ANN.fit(X_train, y_train)

          C:\Users\shahk\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:562: ConvergenceWarning: Stochastic
          Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
            % self.max_iter, ConvergenceWarning)

Out[318]: MLPRegressor(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
                 beta_2=0.999, early_stopping=False, epsilon=1e-08,
                 hidden_layer_sizes=(6, 4), learning_rate='constant',
                 learning_rate_init=0.001, max_iter=200, momentum=0.9,
                 n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
                 random_state=2, shuffle=True, solver='adam', tol=0.0001,
                 validation_fraction=0.1, verbose=False, warm_start=False)

In [319]: # Weights:
          print(my_ANN.coefs_)
          # The ith element in the list represents the weight matrix corresponding to layer i.

          print('\n')

          # Bias weights:
          print(my_ANN.intercepts_)
          # The ith element in the list represents the bias vector corresponding to layer i + 1.
```



Testing (Prediction) Stage: Making prediction on new observations (Testing Data) using the trained model:

```
In [320]: # Testing on the testing set:
          y_predict_ann = my_ANN.predict(X_test)
          print(y_predict_ann)

          [116.67511436 124.49450205  84.84684553 ... 272.02334104 219.3995328
           211.28890693]
```

Accuracy Evaluation:

```
In [321]: from sklearn import metrics

          # Calculating "Mean Square Error" (MSE):
          mse = metrics.mean_squared_error(y_test, y_predict_ann)

          # Using numpy sqrt function to take the square root and calculate "Root Mean Square Error" (RMSE)
          rmse = np.sqrt(mse)

          print('RSME:',rmse)

          RSME: 69.6475879521891
```

7. FINDING THE BEST FEATURE

Bedrooms and Beds are the most significant variables for the prediction of price.

We use following code to find the best feature

```
from sklearn.metrics import matthews_corrcoef

feature_cols = ['bathrooms','bedrooms','beds',

            'extra_people','tv',

            'Parking','Pool','Kitchen',

            'latitude','longitude', 'Wireless_Internet']

a = []

for f in feature_cols:

    a.append(np.abs(np.corrcoef(X[f],y)[1,0]))


a.sort()

print(a)

count = 1

for f in feature_cols:

    if np.abs(np.corrcoef(X[f],y)[1,0]):

        #print(f)

        print(count , '-' , f  , '=' ,  np.abs(np.corrcoef(X[f],y)[1,0]))

        count = count + 1
```

8. Conclusion

When we compared the test RMSE of these models, the ANN regression model has the highest one which is 69.54 and the random forest model has the lowest one, which is 60.52. Compared to other models we used, random forest model is an effective model to predict the Airbnb price here, because this model is easier to identify important predictors and easier to be improved, and it works well with high dimensional predictors and large training sets. However, it has a disadvantage that it takes too long to do the prediction when the tree numbers are large. Based on the test RMSE of each model, we selected the random forest model as our best model to predict the price as it has the smallest test RMSE. We also concluded that the most significant variables for the prediction of price are cleaning fee, bedrooms, accommodates, beds and bathrooms.

Comparison amongst different regression models.

NO	MODEL	RMSE
1	ANN REGRESSION	69.54
2	RANDOM FOREST	60.52
3	LINEAR REGRESSION	68.77
4	POLYNOMIAL REGRESSOR	62.75

9. Team Member Responsibilities

The team started with a research on data pre-processing and each team member provided ideas to pre-process the data. After that, the team split into two groups. Ravi and Kruti worked on the price visualization and regression algorithms, including linear regression, random forest. Shardul and Rutvi define a function to convert categorical feature into numerical feature, define correlation between features and performed polynomial regressor and neural network related regression. After that, we find the best feature. Ravi and Kruti made the final report. Shardul and Rutvi worked on presentation.