```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

        from sklearn.preprocessing import MinMaxScaler
        from sklearn.model_selection import train_test_split
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense
        from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

        import warnings
        warnings.filterwarnings('ignore')
```

```
In [ ]: df = pd.read_csv("/content/drive/MyDrive/DL/AB_NYC_2019.csv")
        df.head()
```

Out[ ]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225
2	3647	THE VILLAGE OF HARLEM....NEW YORK !	4632	Elisabeth	Manhattan	Harlem	40.80902	-73.94190	Private room	150
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80

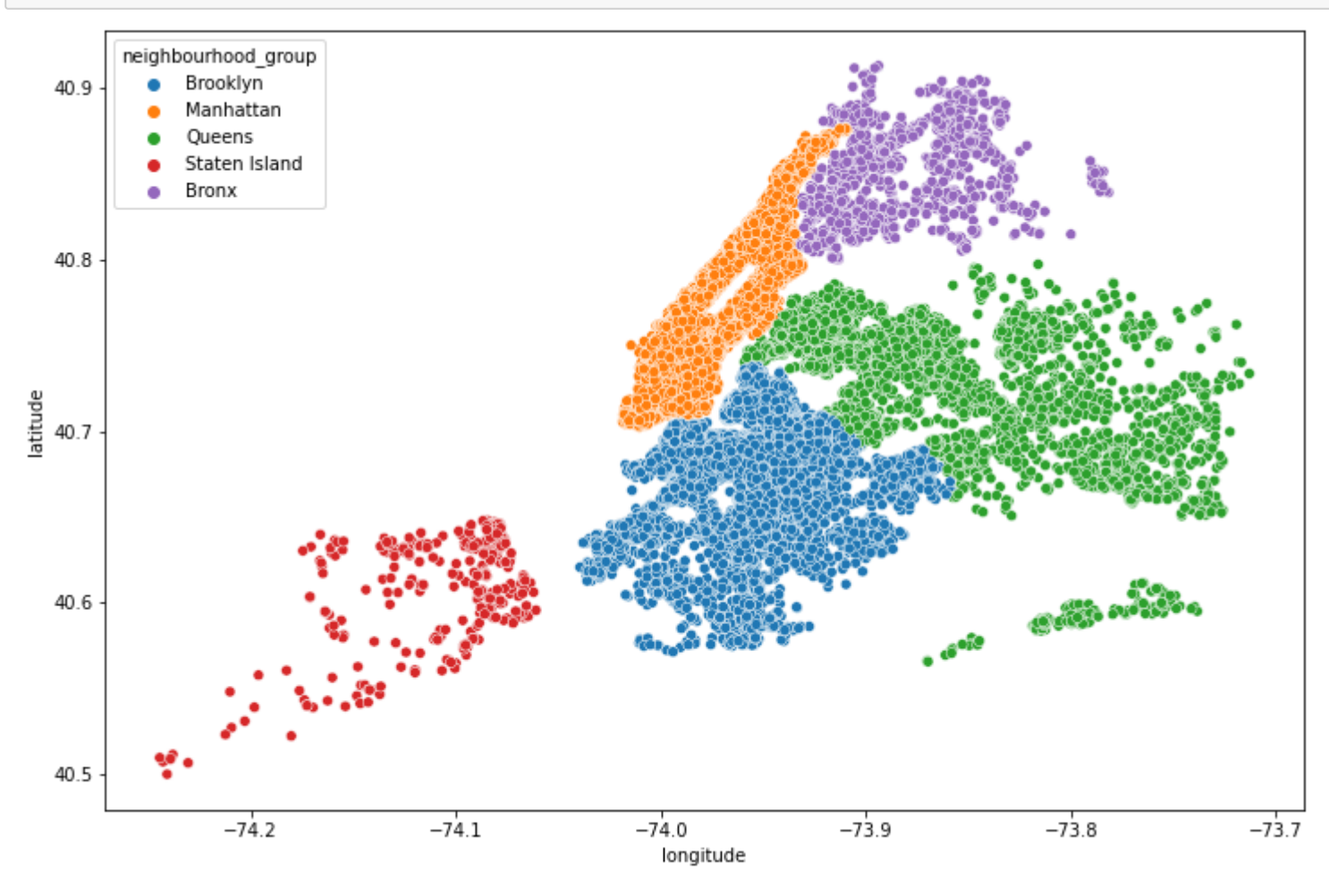
```
In [ ]: df.shape
Out[ ]: (48895, 16)
```

Exploratory Data Analysis (EDA) and Preprocessing

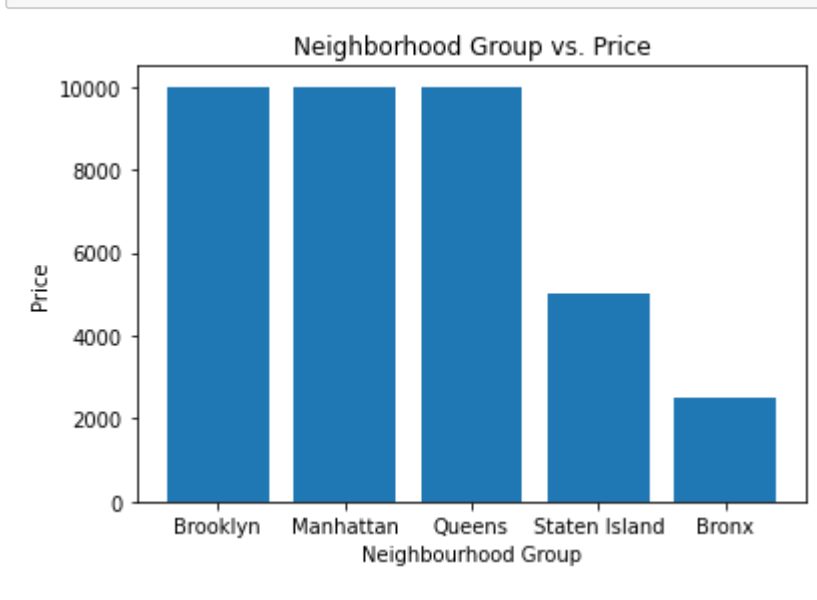
```
In [ ]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  object
 5   neighbourhood                   48895 non-null  object
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

```
In [ ]: # Since more than enough data is missing we drop review columns
        df.drop(["last_review", "reviews_per_month"], axis=1, inplace=True)
```

```
In [ ]: #scatterplot to show on which latitudes and longitudes each neighbourhood is
        plot_dims=(12,8)
        plt.figure(figsize=plot_dims)
        sns.scatterplot(df.longitude,df.latitude,hue=df.neighbourhood_group)
        plt.show()
```

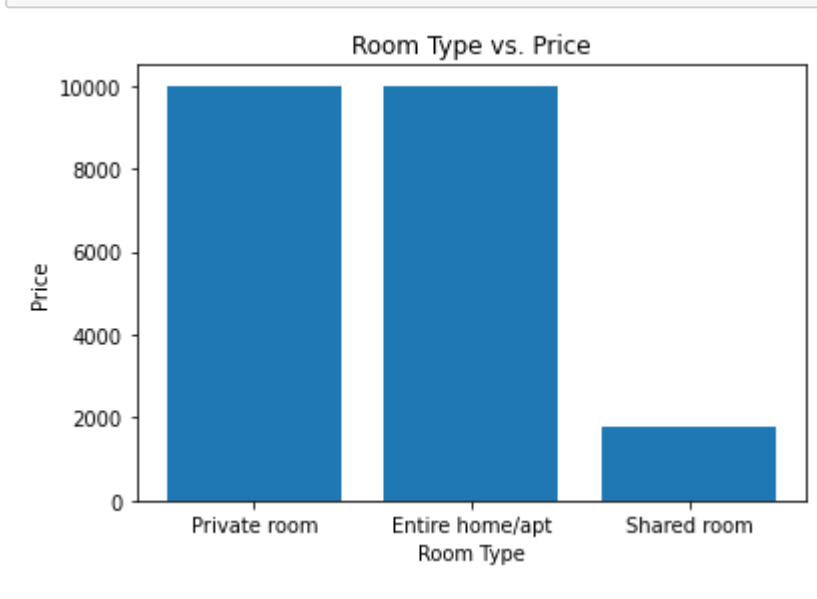



```
In [ ]: plt.figure()
        plt.bar(df.neighbourhood_group, df.price)
        plt.xlabel("Neighbourhood Group")
        plt.ylabel("Price")
        plt.title("Neighborhood Group vs. Price")
        plt.show()
```



From above plot it is very clear that, there is change in price with neighbourhood group. Brooklyn, Manhattan, Queens have maximum price. Maximum price shown here is 10K

```
In [ ]: plt.figure()
        plt.bar(df.room_type, df.price)
        plt.xlabel("Room Type")
        plt.ylabel("Price")
        plt.title("Room Type vs. Price")
        plt.show()
```



Above graph explains how price varies with room type

```
In [ ]: # Removing unwanted columns
        df = df.drop(['id', 'name', 'host_id', 'host_name'], axis=1)
```

Longitude and latitude cannot be normalised like other data. This is a special method used to normalize coordinates

```
In [ ]: df['x'] = np.cos(df['latitude'], dtype=np.float64) * np.cos(df['longitude'], dtype=np.float6
        4)
        df['y'] = np.cos(df['latitude'], dtype=np.float64) * np.sin(df['longitude'], dtype=np.float6
        4)
        df['z'] = np.sin(df['latitude'], dtype=np.float64)
```

```
In [ ]: df=df.drop(['latitude','longitude'],axis=1)
```

```
In [ ]: df.head()
```

Out[ ]:

	neighbourhood_group	neighbourhood	room_type	price	minimum_nights	number_of_reviews	calculated_host_listings_coun
0	Brooklyn	Kensington	Private room	149	1	9	
1	Manhattan	Midtown	Entire home/apt	225	1	45	
2	Manhattan	Harlem	Private room	150	3	0	
3	Brooklyn	Clinton Hill	Entire home/apt	89	1	270	
4	Manhattan	East Harlem	Entire home/apt	80	10	9	

```
In [ ]: df_cat = df.select_dtypes(object)
        df_num = df.select_dtypes(["float64","int64"])
        #  Label encoding

        from sklearn.preprocessing import LabelEncoder

        for col in df_cat:
            le = LabelEncoder()
            df_cat[col] = le.fit_transform(df_cat[col])
```

```
In [ ]: df_cat
```

Out[ ]:

	neighbourhood_group	neighbourhood	room_type
0	1	108	1
1	2	127	0
2	2	94	1
3	1	41	0
4	2	61	0
...	...	...	...
48890	1	13	1
48891	1	28	1
48892	2	94	0
48893	2	95	2
48894	2	95	1

48895 rows × 3 columns

```
In [ ]: df_new=pd.concat([df_num,df_cat],axis=1)
```

```
In [ ]: X = df_new.drop("price",axis=1)
```

```
In [ ]: #Now applying the log transformation to the price.
        y = np.log1p(df_new.price.values)
```

```
In [ ]: standard_scaler = MinMaxScaler()
        x = X[['minimum_nights', 'number_of_reviews', 'calculated_host_listings_count', 'availabilit
        y_365','neighbourhood_group','neighbourhood','room_type']]
        x_scaled = standard_scaler.fit_transform(x)
        newX = pd.DataFrame(x_scaled)
```

```
In [ ]: X[['minimum_nights', 'number_of_reviews', 'calculated_host_listings_count', 'availability_36
        5','neighbourhood_group','neighbourhood','room_type']] = newX
        X.head()
```

Out[ ]:

	minimum_nights	number_of_reviews	calculated_host_listings_count	availability_365	x	y	z	neighb
0	0.000000	0.014308	0.015337	1.000000	-0.141748	-0.971101	0.192015	
1	0.000000	0.071542	0.003067	0.972603	-0.155116	-0.984060	0.086974	
2	0.001601	0.000000	0.000000	1.000000	-0.114165	-0.992957	0.031679	
3	0.000000	0.429253	0.000000	0.531507	-0.130353	-0.979287	0.154938	
4	0.007206	0.014308	0.000000	0.000000	-0.116195	-0.992330	0.042182	

Dividing into training and testing data

```
In [ ]: xtrain,xtest,ytrain,ytest=train_test_split(X,y,random_state=1,test_size=0.3)
```

```
In [ ]: xtrain.shape
Out[ ]: (34226, 10)
```

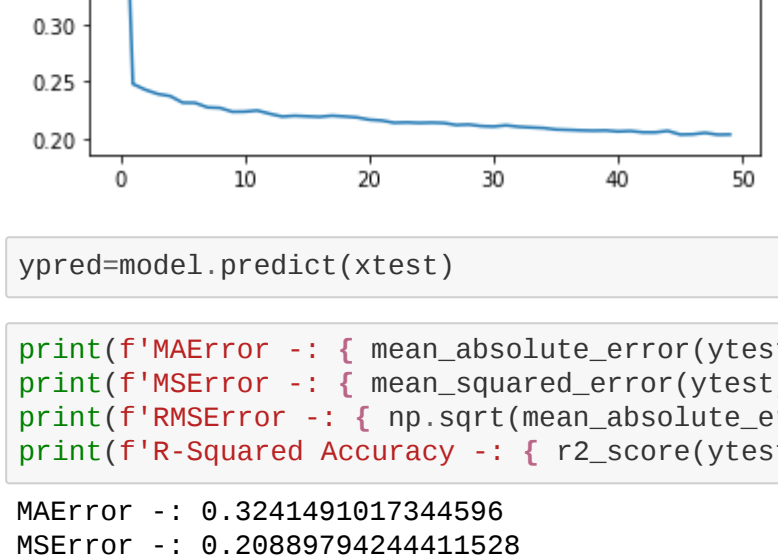
```
In [ ]: model=Sequential()
        model.add(Dense(64,input_dim=10,activation='relu'))
        model.add(Dense(64,activation='relu'))
        model.add(Dense(64,activation='relu'))
        model.add(Dense(64,activation='relu'))
        model.add(Dense(64,activation='relu'))
        model.add(Dense(1))
```

```
In [ ]: model.compile(optimizer='adam',loss='mse')
```

```
In [ ]: trained_model=model.fit(xtrain,ytrain,epochs=50)

Epoch 1/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.5509
Epoch 2/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2477
Epoch 3/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2426
Epoch 4/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2388
Epoch 5/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2370
Epoch 6/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2312
Epoch 7/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2311
Epoch 8/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2271
Epoch 9/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2266
Epoch 10/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2232
Epoch 11/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2234
Epoch 12/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2243
Epoch 13/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2215
Epoch 14/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2190
Epoch 15/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2197
Epoch 16/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2192
Epoch 17/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2188
Epoch 18/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2198
Epoch 19/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2191
Epoch 20/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2183
Epoch 21/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2163
Epoch 22/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2155
Epoch 23/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2136
Epoch 24/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2139
Epoch 25/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2134
Epoch 26/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2137
Epoch 27/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2134
Epoch 28/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2116
Epoch 29/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2120
Epoch 30/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2107
Epoch 31/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2103
Epoch 32/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2114
Epoch 33/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2101
Epoch 34/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2095
Epoch 35/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2090
Epoch 36/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2077
Epoch 37/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2073
Epoch 38/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2069
Epoch 39/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2066
Epoch 40/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2068
Epoch 41/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2060
Epoch 42/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2064
Epoch 43/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2052
Epoch 44/50
1070/1070 [==============================] - 2s 2ms/step - loss: 0.2052
Epoch 45/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2065
Epoch 46/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2032
Epoch 47/50
1070/1070 [==============================] - 3s 2ms/step - loss: 0.2035
Epoch 48/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2047
Epoch 49/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2031
Epoch 50/50
1070/1070 [==============================] - 3s 3ms/step - loss: 0.2033
```

```
In [ ]: trained_model.history
Out[ ]: {'loss': [0.5509178638458252,
  0.2476518154144287,
  0.2425631880760193,
  0.23884886503219604,
  0.23702363669872284,
  0.23122753202915192,
  0.2310522347688675,
  0.22708866000175476,
  0.2266114503145218,
  0.22321200370788574,
  0.22335757315158844,
  0.22431504726409912,
  0.22148428857326508,
  0.2190050184726715,
  0.21972103416919708,
  0.21924316883087158,
  0.21877732872962952,
  0.21979157626628876,
  0.21910491585731506,
  0.21829961240291595,
  0.21626824140548706,
  0.21554960310459137,
  0.2135658860206604,
  0.21389423310756683,
  0.21344135701656342,
  0.213713601231575,
  0.21337008476257324,
  0.21161223948001862,
  0.21204574406147003,
  0.21067583560943604,
  0.21025912463665009,
  0.21135178208351135,
  0.21006323397159576,
  0.20946641266345978,
  0.2089657485485077,
  0.20768381655216217,
  0.2073175311088562,
  0.20688697695732117,
  0.20661379396915436,
  0.20682232081890106,
  0.20595304667949677,
  0.20637518167495728,
  0.20518697798252106,
  0.20516973733901978,
  0.20646366477012634,
  0.20321600139141083,
  0.20347006618976593,
  0.20474886894226074,
  0.2031114548444748,
  0.2033006250858307]}
```

```
In [ ]: plt.plot(trained_model.history['loss'])
Out[ ]: [<matplotlib.lines.Line2D at 0x7f014848d090>]
```




```
In [ ]: ypred=model.predict(xtest)
```

```
In [ ]: print(f'MAError -: { mean_absolute_error(ytest, ypred) }')
        print(f'MSError -: { mean_squared_error(ytest, ypred) }')
        print(f'RMSError -: { np.sqrt(mean_absolute_error(ytest, ypred)) }')
        print(f'R-Squared Accuracy -: { r2_score(ytest, ypred) }')

MAError -: 0.3241491017344596
MSError -: 0.20889794244411528
RMSError -: 0.5693409362890215
R-Squared Accuracy -: 0.5610300989281758
```