Airbnb Price Prediction Documentation

We will be using the 'full_join_mayflower_top_10.csv' file from http://insideairbnb.com/get-the-data.html to perform the analysis.

I use a Jupyter Notebook for all analysis and visualization.

Step 1: Read the Dataframe

```
data = pd.read_csv('full_join_mayflower_top_10.csv')
data
```

```
data.info()
```

```
profile = ProfileReport(data)
```

Step 2: Outliers handling & SPLITS

I star with handling outliers:

```
data = data[data['adjusted_price'] < 200]
```

Then I continue with the SPLITS of data for validation, train and test:

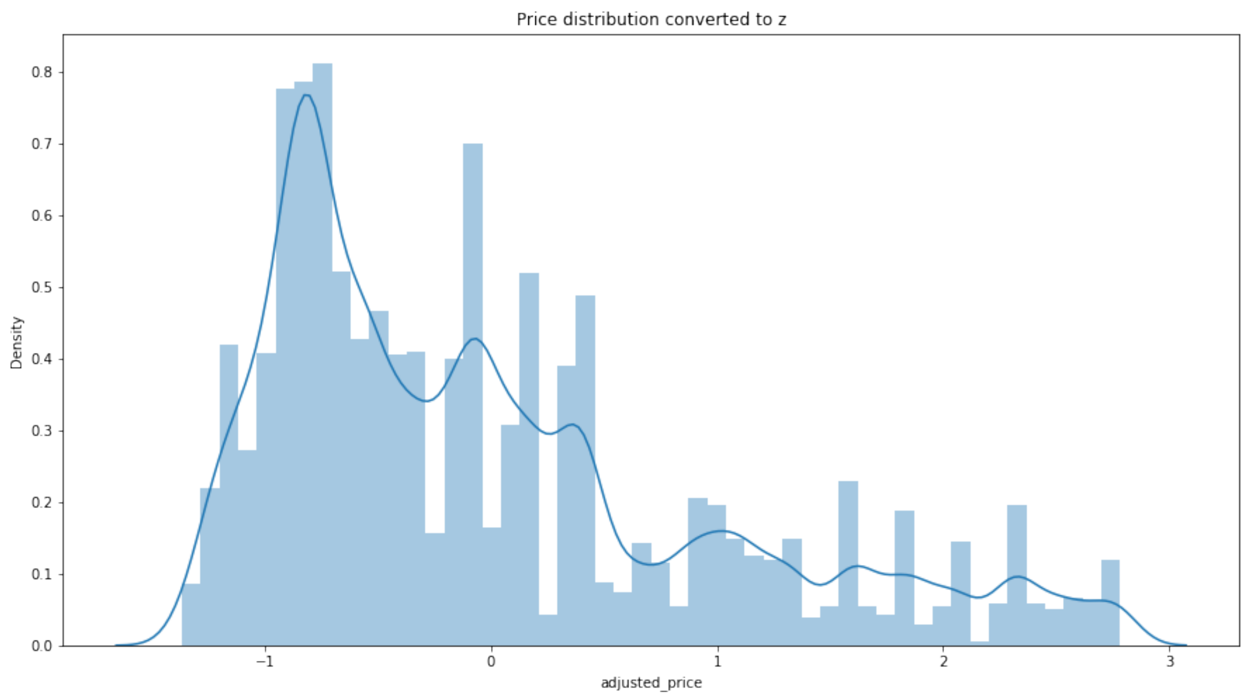
```
df, validation_df  = train_test_split(data,
                                test_size=0.25,
                                random_state = 101)
```

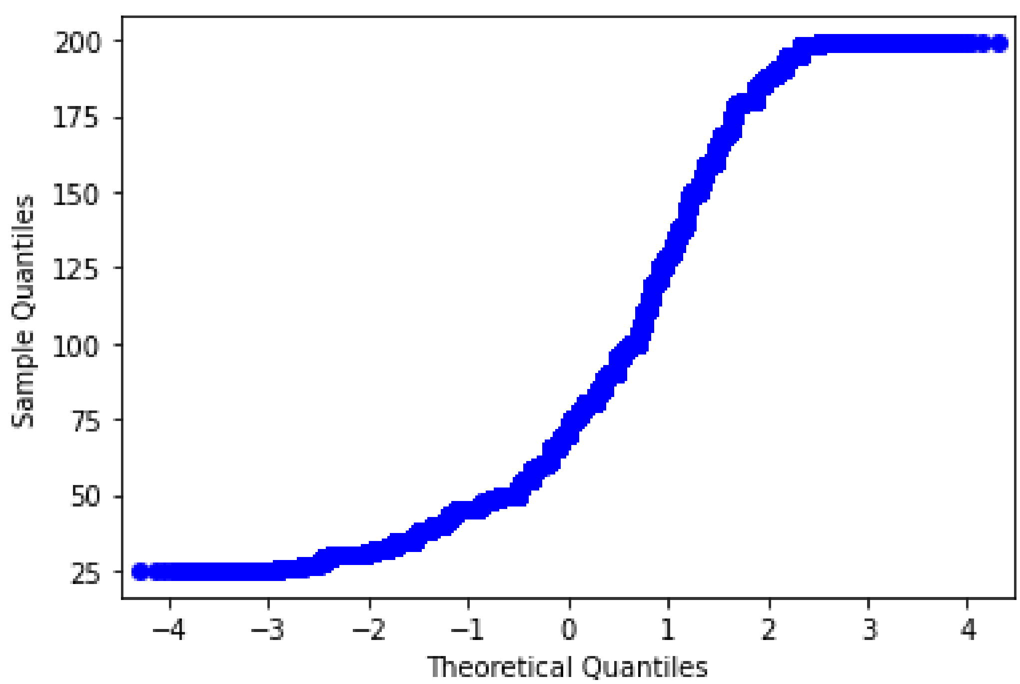
In [52]:

```
df_train, df_test  = train_test_split(df,
                                test_size=0.25,
                                random_state = 101)
```

Step 3: EDA

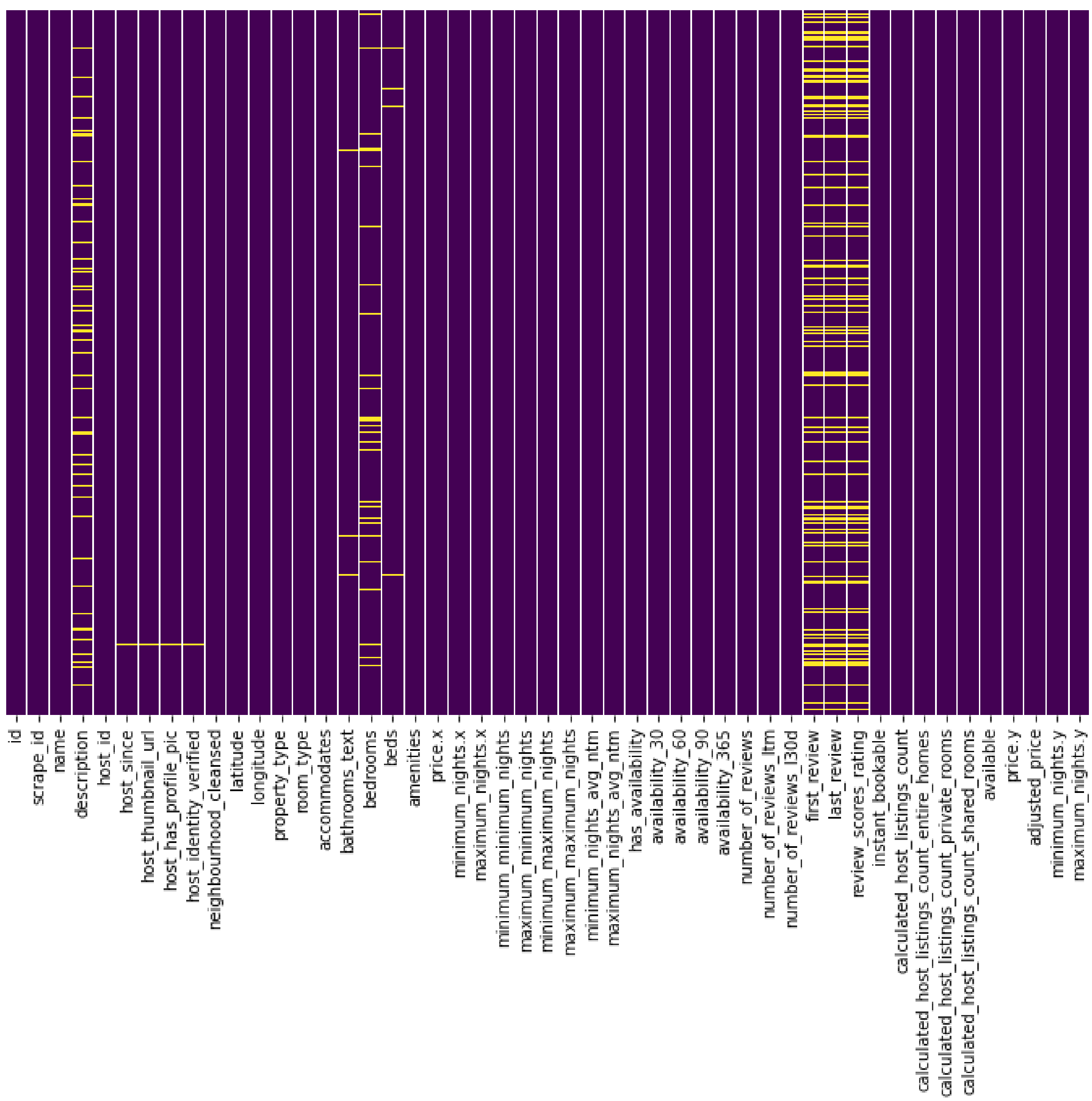
I plot Density vs Adjusted_price and also qqplot of the adjusted_price.





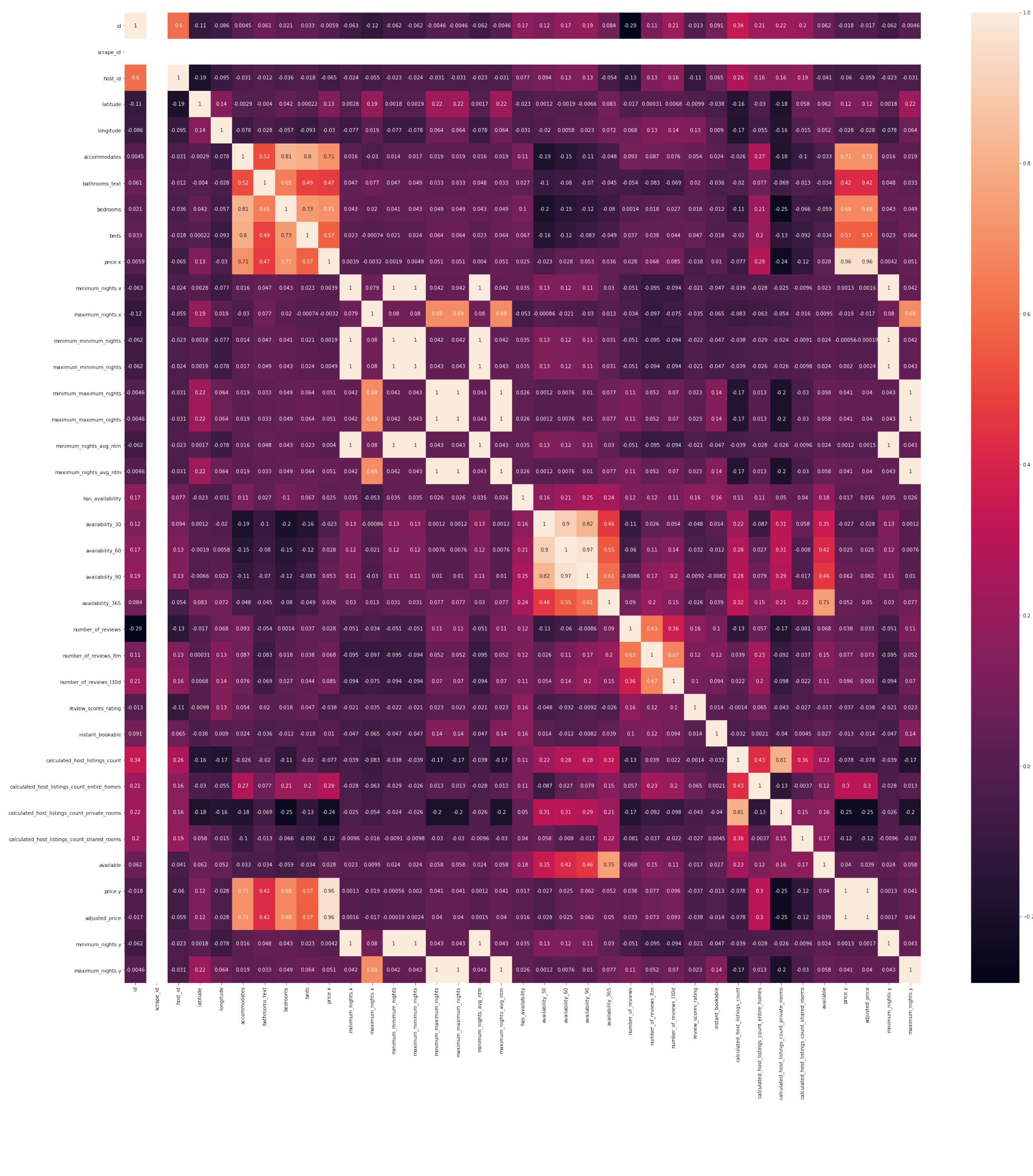
Step 4: Columns with a lot of missing data

I identify them easily with this heatmap to eliminate the unnecessary columns:



Correlations

I also use the correlations heatmap to explore the data:



Step 5: Building the Model

Finally, we can build the sentiment analysis model!

This model will take reviews in as input. It will then come up with a prediction on whether the review is positive or negative.

This is a classification task, so we will train a simple logistic regression model to do it.

For reference, take a look at the data frame again:

There are a few steps we need to take:

Linear Model

```
lm = LinearRegression(

        n_jobs = -1,

        normalize = True

)


lm.fit(df_d.drop('adjusted_price', axis = 1), df_d['adjusted_price'])


comment = ''


analysis(

    model = lm,

    X_train = df_d.drop('adjusted_price', axis = 1),

    X_test = test.drop('adjusted_price', axis = 1),

    y_train = df_d['adjusted_price'],

    y_test = test['adjusted_price']

)
```

	MAE	MSE	RMSE	RMSE_ratio_test	RMSE_ratio_train	R_2_test	R_2_train
0	3623288.23	1.666261e+13	4081985.89	49669.612	49682.244	-5.850580e+09	-5.872179e+09

RandomForestRegressor

```
rfm = RandomForestRegressor(

          max_depth = 10,

          n_jobs = -1,

          random_state = 101,

          n_estimators = 700


)

rfm.fit(df_d.drop('adjusted_price', axis = 1), df_d['adjusted_price'])


comment = ''


analysis(

    model = rfm,

    X_train = df_d.drop('adjusted_price', axis = 1),

    X_test = test.drop('adjusted_price', axis = 1),

    y_train = df_d['adjusted_price'],

    y_test = test['adjusted_price']

)
```

	MAE	MSE	RMSE	RMSE_ratio_test	RMSE_ratio_train	R_2_test	R_2_train
0	0.02	0.22	0.47	0.006	0.004	1.0	1.0

GradientBoostingRegressor

```
gbr_model = GradientBoostingRegressor(random_state = 101)

gbr_model.fit(df_d.drop('adjusted_price', axis = 1), df_d['adjusted_price'])


comment = ''


analysis(

    model = gbr_model,

    X_train = df_d.drop('adjusted_price', axis = 1),

    X_test = test.drop('adjusted_price', axis = 1),

    y_train = df_d['adjusted_price'],

    y_test = test['adjusted_price']

)
```

	MAE	MSE	RMSE	RMSE_ratio_test	RMSE_ratio_train	R_2_test	R_2_train
0	0.14	0.25	0.5	0.006	0.006	1.0	1.0

```
#scaling

scaler = MinMaxScaler()

scaler.fit(df_d.drop('adjusted_price', axis = 1))

X_train_sc = scaler.transform(df_d.drop('adjusted_price', axis = 1).values)

X_test_sc = scaler.transform(test.drop('adjusted_price', axis = 1).values)

y_train = df_d['adjusted_price'].values

y_test = test['adjusted_price'].values
```

In [94]:

```
nn_model1 = Sequential()



es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
```

```
nn_model1.add(Dense(64, activation = 'relu'))
nn_model1.add(Dropout(0.1))
nn_model1.add(Dense(1))

nn_model1.compile(
    optimizer='rmsprop',
    loss='mse'
)

nn_model1.fit(
    x = X_train_sc,
    y = y_train,
    epochs = 100,
    validation_data=(X_test_sc, y_test),
    batch_size = 128,
    callbacks=[es]
)


pd.DataFrame(nn_model1.history.history).plot()
plt.show()

analysis(model = nn_model1,
         X_train = X_train_sc,
         X_test = X_test_sc,
         y_train = y_train,
         y_test = y_test)
```

Dense neural models

```
#scaling
scaler = MinMaxScaler()
scaler.fit(df_d.drop('adjusted_price', axis = 1))
X_train_sc = scaler.transform(df_d.drop('adjusted_price', axis = 1).values)
X_test_sc = scaler.transform(test.drop('adjusted_price', axis = 1).values)
y_train = df_d['adjusted_price'].values
y_test = test['adjusted_price'].values
```

In [94]:

```
nn_model1 = Sequential()



es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)




nn_model1.add(Dense(64, activation = 'relu'))
nn_model1.add(Dropout(0.1))
nn_model1.add(Dense(1))

nn_model1.compile(
    optimizer='rmsprop',
    loss='mse'
)

nn_model1.fit(
    x = X_train_sc,
    y = y_train,
    epochs = 100,
```

```
    validation_data=(X_test_sc, y_test),
    batch_size = 128,
    callbacks=[es]
)


pd.DataFrame(nn_model1.history.history).plot()
plt.show()

analysis(model = nn_model1,
         X_train = X_train_sc,
         X_test = X_test_sc,
         y_train = y_train,
         y_test = y_test)
```

Step 6: Conclusion

This model doesn't have an advantage over more simple models. So for final submission I prefer to use just simple RFR