New York City AirBnb Price Prediction

By

Adriana C. , Anumala T. , Jeongdae K., & Samuel O.

Project Overview

In order to test our knowledge on machine learning we used 3 different models to predict New York City Airbnb prices.

Focal points:

- ➔ Advantages of using the specific model
- ➔ Challenges of using the specific model
- ➔ Importance of features while predicting the target
- ➔ Deployment of all 3 models to app

Data

- The dataset used for this project can be downloaded online:

  https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data?select=AB_NYC_2019.csv
- It contains information about prices and locations of Airbnb listings in New York City along with information about hosts, availability, necessary metrics to make predictions and draw conclusions.
- Among the 16 columns in the dataset we chose 4 features (neighborhood_group, room_type, price, reviews_per_month, avalability_365) and 1 target (price).
- Further cleaning was done by reducing the range of the price up to 250 per night for the normal distribution of the data.
- The categorical features were converted into dummy/indicator variables and any missing data were filled appropriately.

Data Cleaning

```
In [28]: nyc = df[['neighbourhood_group', 'room_type', 'reviews_per_month', 'availability_365', 'price' ]]
         nyc.head()
```

Out[28]:

	neighbourhood_group	room_type	reviews_per_month	availability_365	price
0	Brooklyn	Private room	0.21	365	149
1	Manhattan	Entire home/apt	0.38	355	225
2	Manhattan	Private room	NaN	365	150
3	Brooklyn	Entire home/apt	4.64	194	89
4	Manhattan	Entire home/apt	0.10	0	80

```
In [29]: # #Airbnb for less than $250 per night
         nyc =nyc.loc[nyc["price"] <= 250]
```

```
In [30]: # Replacing nan values with 0
         nyc["reviews_per_month"] = nyc["reviews_per_month"].fillna(0)
```

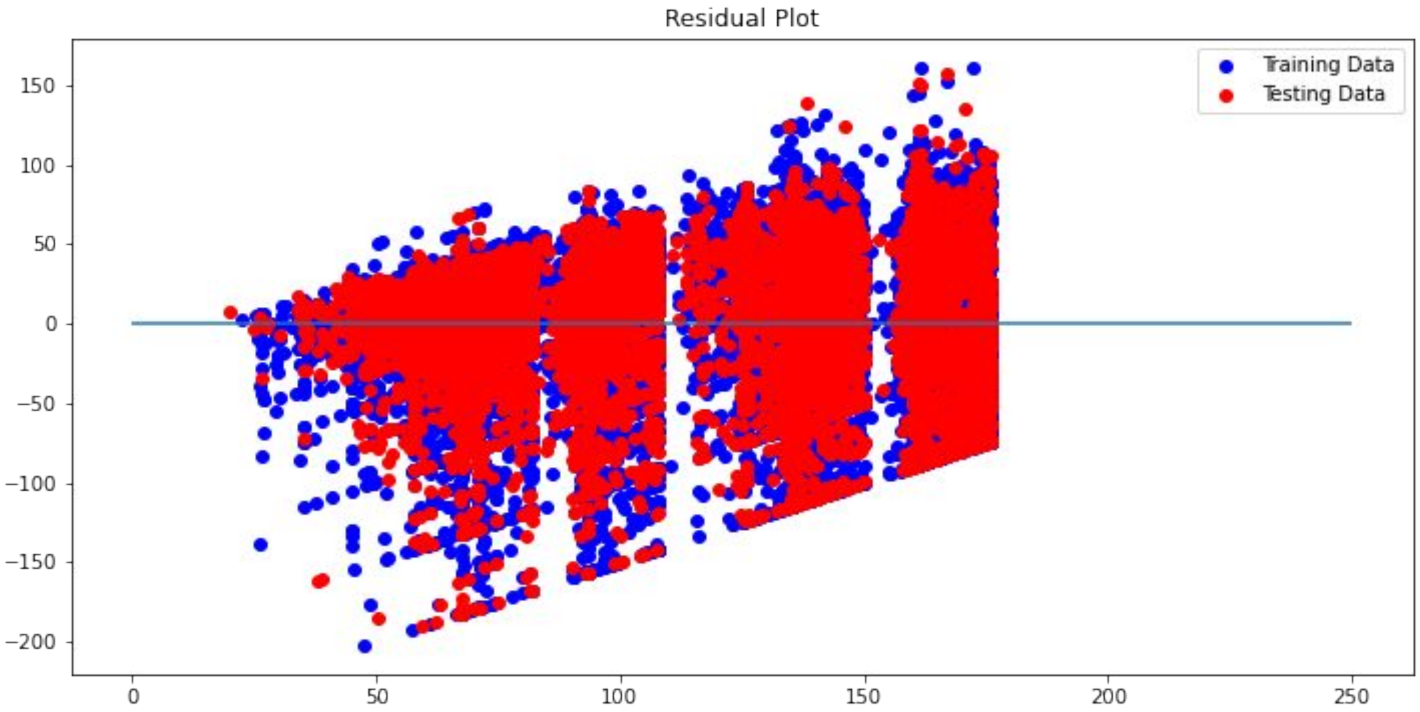
```
In [31]: # Converting categorical data using get_dummies
         X = pd.get_dummies(nyc[['neighbourhood_group', 'room_type', 'reviews_per_month', 'availability_365']])
         y = nyc['price'].values.reshape(-1, 1)
         print(X.shape, y.shape)

(43687, 10) (43687, 1)
```

Technologies used:

- Numpy
- Pandas
- Matplotlib
- Sklearn
- Seaborn
- Joblib
- Tableau
- Bootstraps
- Html/Css/Flask

Model 1: Linear Regression



Findings:

- Linear Regression was used to model and predict a dependent variable (price) based on given values from independent variables (neighborhood_group, room_type, reviews_per_month, and avalability_365).

- R-Squared is relevant if the primary goal is to predict the value of the dependent variable because is the measure of the error. Lower R-Squared means that the model is with more error and results that the predictions are less precise.

- The R-Squared is 0.48048851750965194.

Model 2: XGBoost

- Extreme Gradient Boosting is based on Decision Tree algorithm and uses gradient boosting framework.
- It has been gaining popularity since its development in 2016 (especially among the communities in Kaggle)
- Can be used with Scikit Learn API and it’s native XGBoost API as well
- Built in CV, custom objective functions, plot_importance, plot_trees etc.
- Regularization for avoiding overfitting, efficient handling of missing data

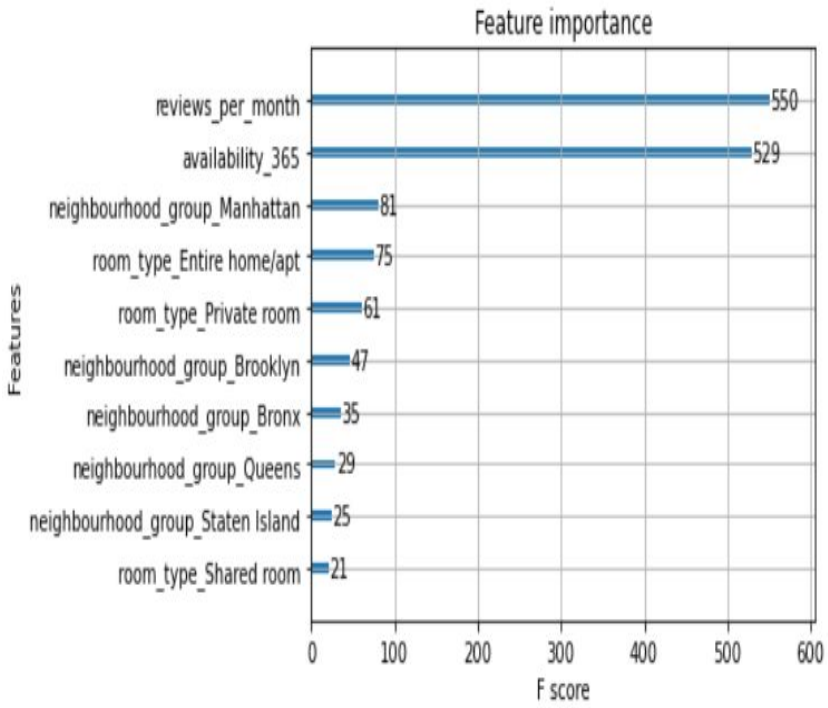
Model 2: XGBoost

- Num_boosting_rounds before tuning : 100
- Numb_boosting_rounds after : 100

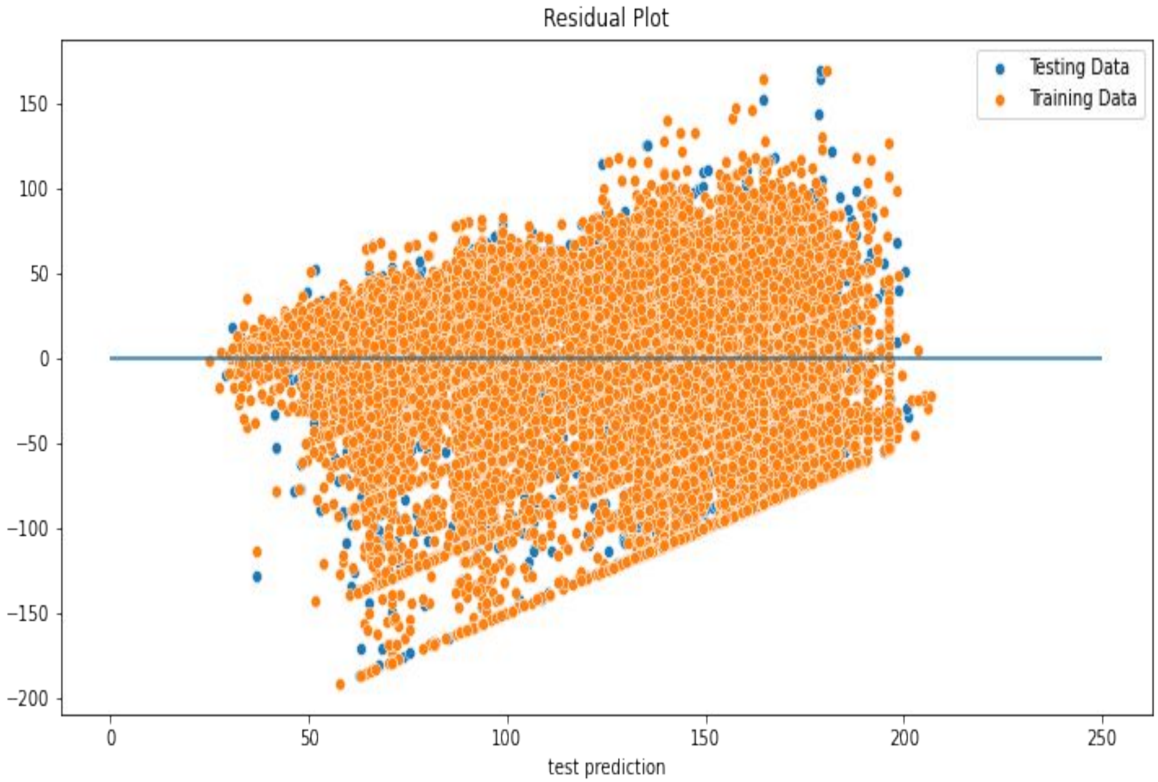
- Other Parameters used:

```
{'max_depth': 4,
 'min_child_weight': 10,
 'eta': 0.1,
 'subsample': 0.8,
 'colsample_bytree': 0.9,
 'objective': 'reg:squarederror'
 'eval_metric': 'rmse'}
```

<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>



Findings:



- Get the best model with low RMSE
  - baseline RMSE: 57.24
  - CV RMSE before: 41.37
  - CV RMSE after: 40.30
- Predict test data with the best model
- r2 score: 0.50423

Sources: [XGBoost Algorithm: Long May She Reign! | by Vishal Morde | Towards Data Science]

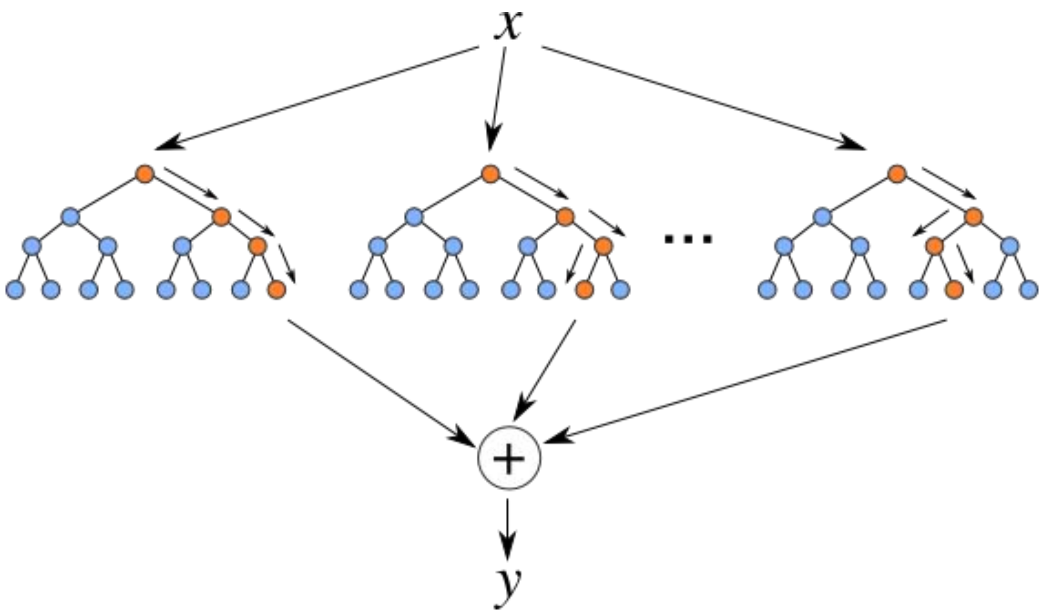
[Hyperparameter tuning in XGBoost. This tutorial is the second part of our… | by Cambridge Spark | Cambridge Spark]

[Getting started with XGBoost (cambridgespark.com)]

[(Tutorial) Learn to use XGBoost in Python - DataCamp]

[XGBoost for stock trend & prices prediction | Kaggle]

Model 3: Random Forest Model



Description:

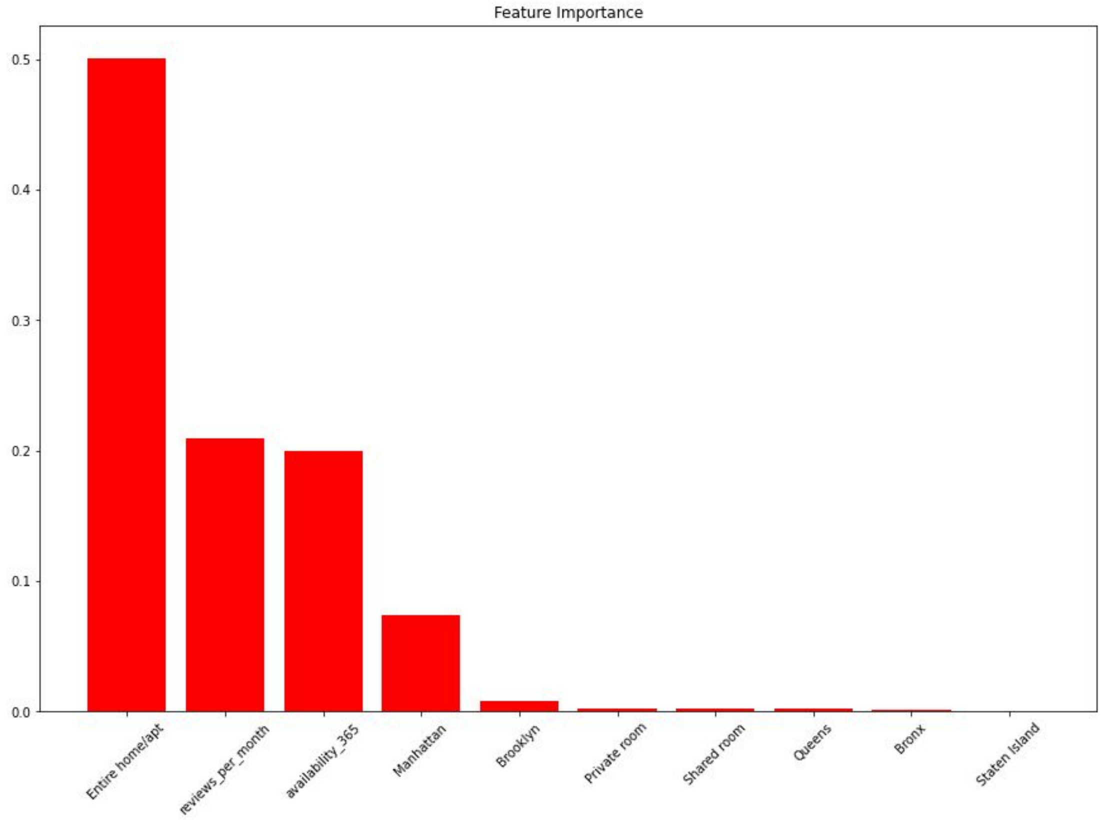
The Random Forest Model makes use of several small decision trees and aggregates them to pool the predictive ability of the model.

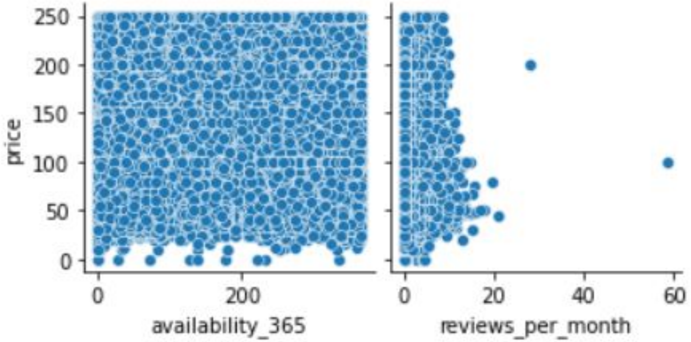
Random Forest Specs:

Number of decision Trees: 50
Model Type: Random Forest Regressor

Tested 5 other amounts from 50 up to 2000 decision trees.

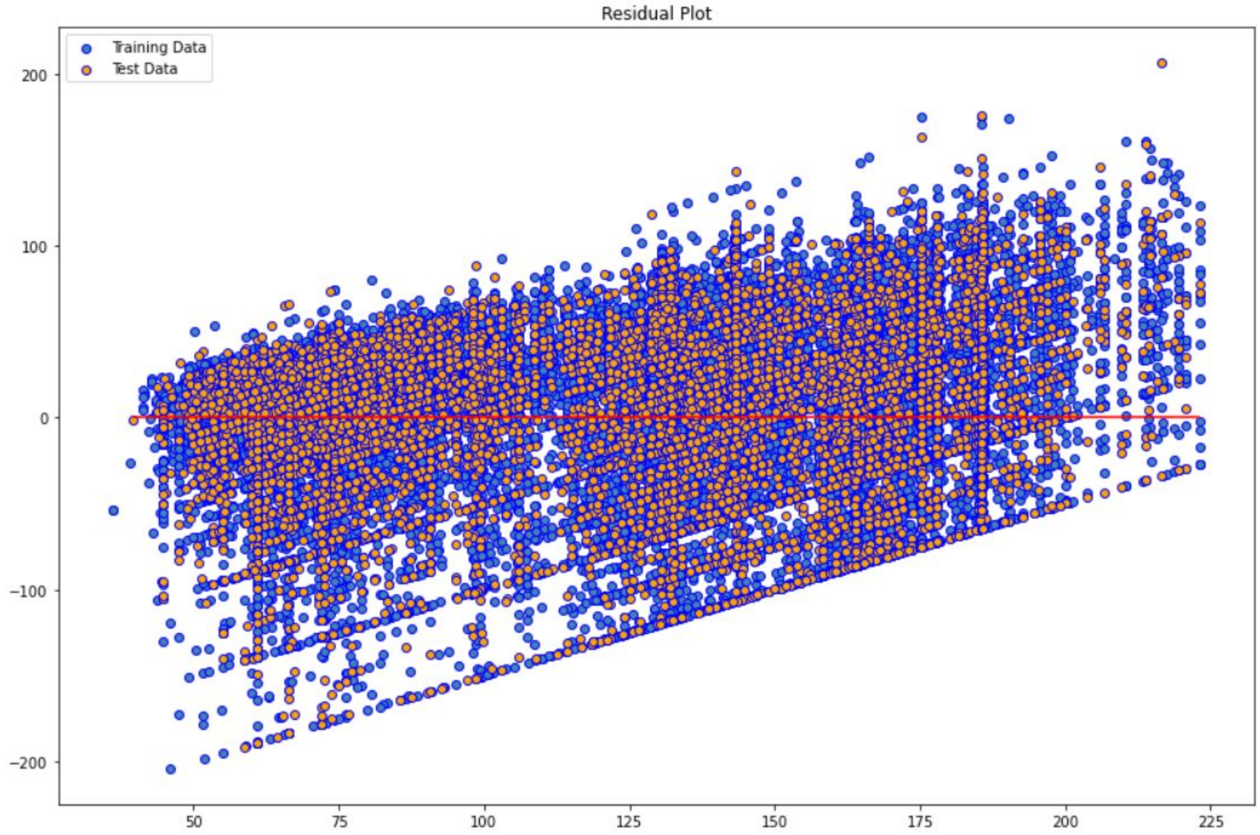
Feature Importance





[(0.5009596557916712, 'Entire home/apt'),
(0.20919920813037393, 'reviews_per_month'),
(0.19972746842749894, 'availability_365'),
(0.07373497253542877, 'Manhattan'),
(0.00796912013459814, 'Brooklyn'),
(0.002208009237272677, 'Private room'),
(0.002130355689854842, 'Shared room'),
(0.002084768184867461, 'Queens'),
(0.001190917127093299, 'Bronx'),
(0.0007955247413407038, 'Staten Island')]

Findings



Final Metrics:

Decision Trees: 50

Mean Absolute Error: 35.0
Mean Squared Error: 2052.01
R-squared scores: 0.3709942287778123

Testing:

Additional number of decision trees tested
100 Trees r2 = 0.3711418228196197
200 Trees r2 = 0.3711418228196197
500 Trees r2 =0.3711418228196197
1000 Trees r2 =0.3711418228196197
2000 Trees r2 = 0.3711418228196197

We found that adding more decision trees did not have a significant improvement on the model's predicting ability.

Challenges of Random Forest Model

- File size rf.h5 2/2/2021 9:22 PM H5 File 156,325 KB
- Unable to push to GitHub without Git LFS.
- Not enough space to host in GitHub free tier.
- Distribution of H5 File as most tools will have trouble transmitting file.
- Unable to deploy to Heroku.

Flask + Heroku App

- Connecting the 3 models into app.py
- Application gives result of a possible price range and a possible average price based on the 3 model results.
- App uses **radio** for *borough* and *room types* inputs, and uses **range** for *reviews per month* and *availability in a year* inputs.
- It is also mobile friendly!!