ANALYSIS OF AIRBNB RENTALS IN NYC

Ryan Chung

AGENDA

- BUSINESS PROBLEM
- DATA
- METHODS
- RESULTS
- CONCLUSION

BUSINESS PROBLEM

A real estate agency is looking to discover which model would be best for predicting the selling price of an Airbnb rental in NYC from a provided data set. The agency is looking to then use the base model in order to improve their ability to determine what the selling point of their own rentals should look like in the area.

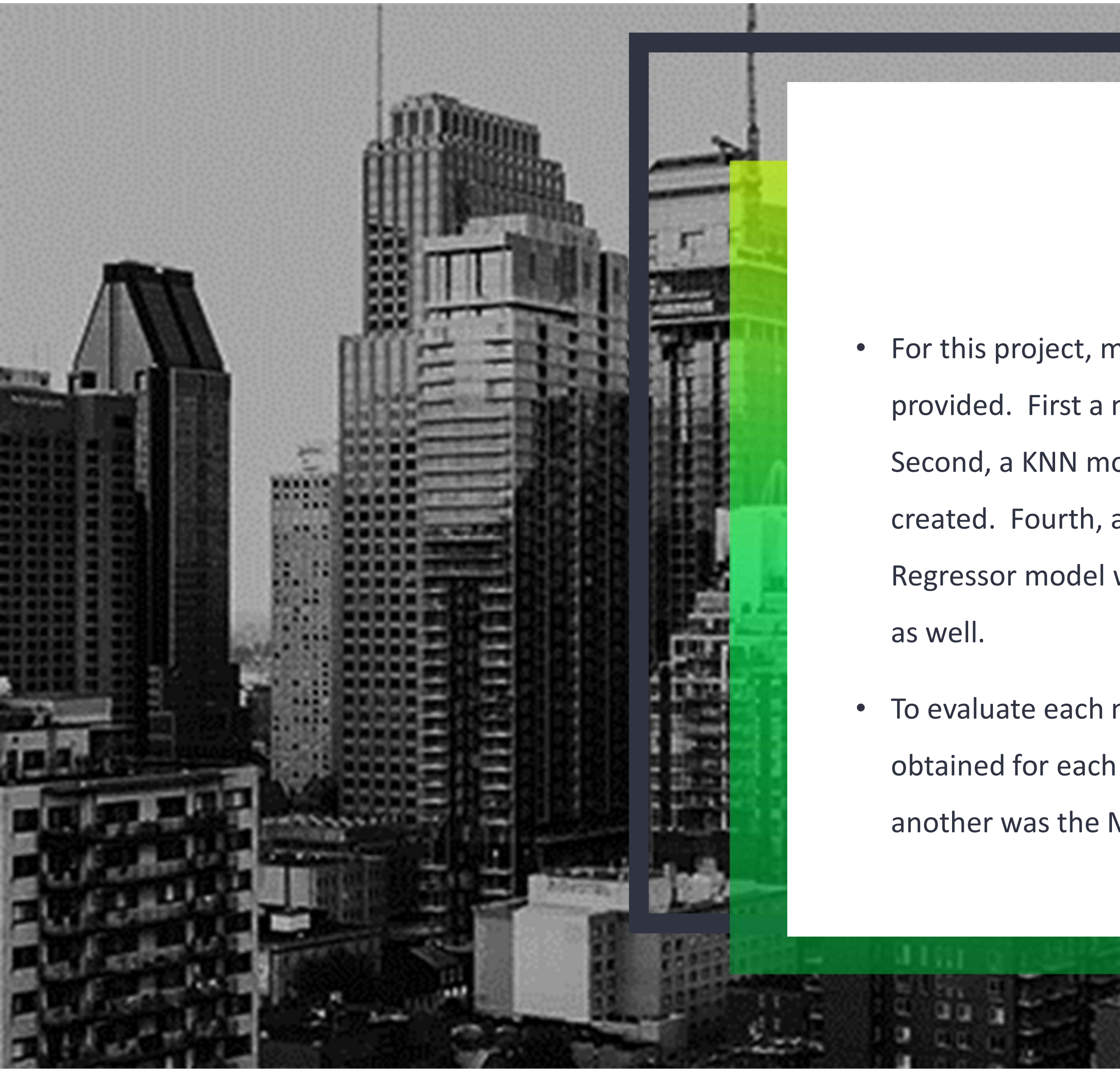
DATA

For this project the data set that was provided by the real estate agency is one found on Kaggle. It contains information on Airbnb rentals for the year of 2019 for NYC. Several features of rentals are included such as a rentals neighborhood and review count.

DATA AFTER CLEAN

```
df.head()
```

	neighbourhood_group	neighbourhood	room_type	price	number_of_reviews	reviews_per_month
0	1	1.0	1.0	149.0	9.0	0.21
1	2	2.0	2.0	225.0	45.0	0.38
3	1	3.0	2.0	89.0	270.0	4.64
4	2	5.0	2.0	80.0	9.0	0.10
5	2	4.0	2.0	200.0	74.0	0.59



METHODS

- For this project, multiple models were used to analyze the data provided. First a multiple linear regression model was created. Second, a KNN model was created. Third, a Decision Tree model was created. Fourth, a Random Forest model was created. Fifth, a XGBoost Regressor model was created. Lastly, a deep neural network was used as well.
- To evaluate each model, the R squared, MAE, MSE, and RMSE were obtained for each one. The metric used to compare one model to another was the MSE.

MLR

Multiple Linear Regression

```
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_LR)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_LR)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_LR))
```

```
print('R squared: {:.2f}'.format(LR.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 6.14
Mean Absolute Error: 65.76500421880998
Mean Square Error: 36915.32169292407
Root Mean Square Error: 192.13360375770833
```

```
X = df.drop(columns = ["price"], axis =1)
y = df["price"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
LR = LinearRegression()
```

```
LR.fit(X_train, y_train)
```

```
LinearRegression()
```

```
y_pred_LR= LR.predict(X_test)
```

```
LR_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_LR})
LR_diff.head()
```

	Actual value	Predicted value
5792	225.0	198.247324
43334	150.0	131.433455
30638	105.0	101.399811
26976	67.0	138.673313
24171	150.0	189.743681

KNN

K Nearest Neighbor

```
meanAbErr = metrics.mean_absolute_error(y_test, test_preds)
meanSqErr = metrics.mean_squared_error(y_test, test_preds)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, test_preds))
```

```
print('R squared: {:.2f}'.format(model.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 11.61
Mean Absolute Error: 69.0433906861735
Mean Square Error: 37891.3992404183
Root Mean Square Error: 194.65713251873999
```

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]}

knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

```
{'n_neighbors': 20}
```

```
test_preds = model.predict(X_test)
```

```
print(model.score(X_test, y_test))
```

```
0.032959885194633554
```

DTR

Decision Tree Regressor

```
tuned_model = DecisionTreeRegressor(max_depth=5,max_features='auto',max_leaf_
```

```
tuned_model.fit(X_train, y_train)
```

```
DecisionTreeRegressor(max_depth=5, max_features='auto',
                      min_weight_fraction_leaf=0.1)
```

```
tuned_pred=tuned_model.predict(X_test)
```

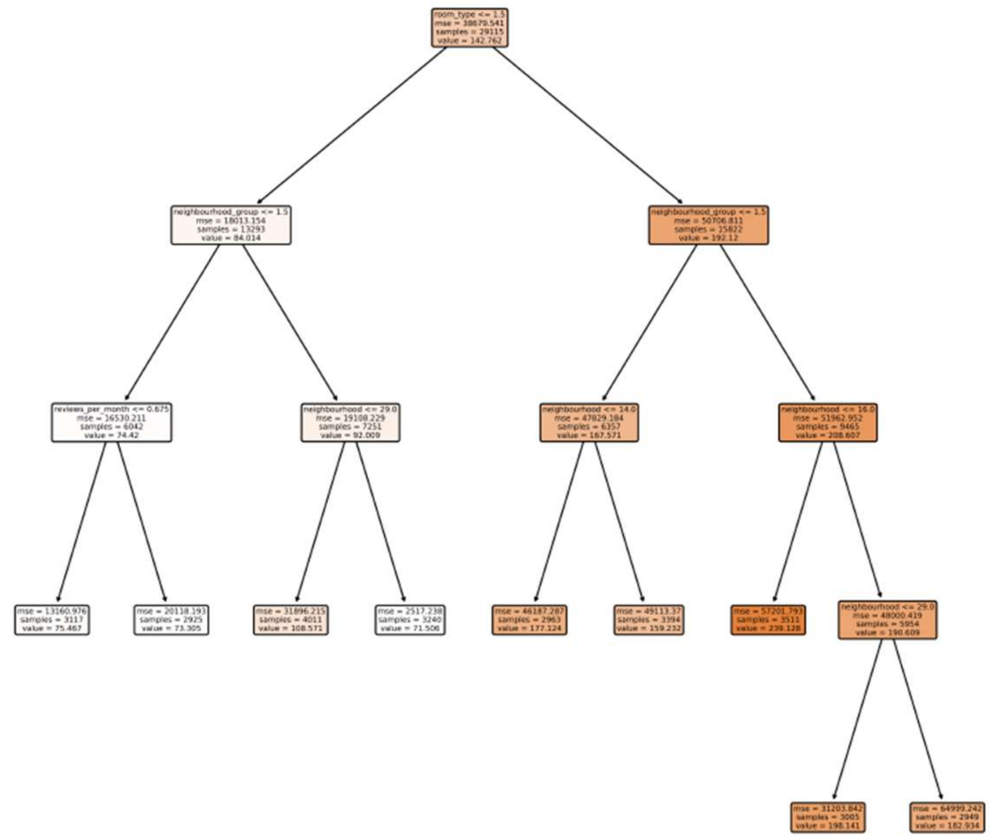
```
print(tuned_model.score(X_test, y_test))
```

```
0.0809398109436571
```

```
meanAbErr = metrics.mean_absolute_error(y_test, tuned_pred)
meanSqErr = metrics.mean_squared_error(y_test, tuned_pred)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, tuned_pred))
```

```
print('R squared: {:.2f}'.format(tuned_model.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 8.71
Mean Absolute Error: 62.65934621355715
Mean Square Error: 36011.40843729863
Root Mean Square Error: 189.76672110066778
```



RFR

Random Forest Regressor

```
print('R squared: {:.2f}'.format(grid_search_rf.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 18.28
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35055.01124393906
Root Mean Square Error: 187.2298353466644
```

```
grid_search_rf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    2.0s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:   23.8s
[Parallel(n_jobs=-1)]: Done 333 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 616 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed:  3.2min finished
GridSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [10, 20, 30, 40],
                         'max_features': ['sqrt'],
                         'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [1, 2, 3, 4],
                         'n_estimators': [100, 200, 300, 400, 500]},
             verbose=2)
```

```
grid_search_rf.best_params_
```

```
{'bootstrap': True,
 'max_depth': 10,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 4,
 'n_estimators': 200}
```

```
test_preds = grid_search_rf.predict(X_test)
```

```
print(grid_search_rf.score(X_test, y_test))
```

```
0.10534836988331808
```

XGBOOST

XGBoost Regressor

```
xgb1 = XGBRegressor()

parameters = {'nthread':[4],
              'objective':['reg:squarederror'],
              'learning_rate': [.03, 0.05, .07,],
              'max_depth': [4, 5, 6],
              'min_child_weight': [4],
              'silent': [1],
              'subsample': [0.5, 0.7, 0.9],
              'colsample_bytree': [0.5, 0.7, 0.9],
```

```
xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_grid.fit(X_train, y_train)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

```
Fitting 2 folds for each of 81 candidates, totalling 162 fits
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   11.7s
[Parallel(n_jobs=5)]: Done 162 out of 162 | elapsed:   46.8s finished
[11:54:30] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


0.104377735435551332
{'colsample_bytree': 0.5, 'learning_rate': 0.03, 'max_depth': 4, 'min_child_weight': 4, 'n_estimators': 400, 'nth
read': 4, 'objective': 'reg:squarederror', 'silent': 1, 'subsample': 0.7}
```

```
print('R squared: {:.2f}'.format(xgb_grid.score(X,y)*100))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
R squared: 15.26
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35147.01263402663
Root Mean Square Error: 187.47536540576905
```

NN

Deep Neural Network

```
Mean Absolute Error: 58.07161729550634
Mean Square Error: 35872.17842360018
Root Mean Square Error: 189.39952065303697
```

```python
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(128, kernel_initializer='normal',input_dim = X_train.shape[1], activation='relu'))

# The Hidden Layers :
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))
NN_model.add(Dense(256, kernel_initializer='normal',activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal',activation='linear'))
```

```python
# Compile the network :
NN_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
NN_model.summary()

history = NN_model.fit(X_train,
                       y_train,
                       epochs=40,
                       batch_size=32,
                       validation_data=(X_test, y_test))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               768
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 165,633
Trainable params: 165,633
Non-trainable params: 0
_________________________________________________________________
```

- Taking a look at the results to the right, we can see that the model with the lowest Mean Squared Error was the tuned Random Forest Regressor. (MSE of 35048).

- The model with the second lowest Mean Squared Error was the XGBoost Regressor. (MSE of 35147).

RESULTS

```
# MLR Results:

# R squared: 6.14
# Mean Absolute Error: 65.76500421880998
# Mean Square Error: 36915.32169292407
# Root Mean Square Error: 192.13360375770833
```

```
# KNN Results:

# R squared: 11.61
# Mean Absolute Error: 69.0433906861735
# Mean Square Error: 37891.3992404183
# Root Mean Square Error: 194.65713251873999
```

```
# Tuned Decision Tree Results:

# R squared: 8.71
# Mean Absolute Error: 62.65934621355715
# Mean Square Error: 36011.40843729863
# Root Mean Square Error: 189.76672110066778
```

```
# Tuned Random Forest Results:

# R squared: 18.48
# Mean Absolute Error: 58.07161729550634
# Mean Square Error: 35048.80209410964
# Root Mean Square Error: 187.21325298736102
```

```
# Tuned XGBoost Results:

# R squared: 15.26
# Mean Absolute Error: 58.07161729550634
# Mean Square Error: 35147.01263402663
# Root Mean Square Error: 187.47536540576905
```

```
# Neural Network Results:

# Mean Absolute Error: 58.07161729550634
# Mean Square Error: 35872.17842360018
# Root Mean Square Error: 189.39952065303697
```

CONCLUSION

For this project, we were able to confirm the Random Forest Regressor Model had the lowest Mean Squared Error compared to the other models created.

Given the nature of the data set provided it is recommended to further develop each model.

THANK YOU