# UWS Rent Prediction Analysis

**Group name: You can't six with us**

**Group members:** Berkin Akar, Lauren Lee, Vriddhi Misra, Haider Munir, Srishti Priya, Jiewan Yang

## Contents

# UWS Rent Prediction Analysis

## 1. Abstract

As six international students studying in New York City, the biggest problem we had to face was finding an apartment with reasonable rent. Rent in NYC is ever-increasing, so we decided to create a model to predict the price of an apartment depending on certain factors.

## 2. Data Extraction

In order to extract the required data for our dataset, we decided to web-scrape data from Renthop. Renthop is one of the most popular apartments listing sites, providing a huge amount of data for each apartment. For our project, we decided to focus on apartments only in the Upper West Side, near Columbia, and those that are priced in the range of $2,000 - $5,000.

The factors we thought would be most influential in determining the price of an apartment were the size of an apartment, the number of bedrooms, the number of bathrooms, distance to the closest subway station, whether the apartment building has laundry, a doorman, pool, elevator, a dishwasher, whether there is a broker fee and what the average rating of nearby restaurants is.

One of the reasons we chose Renthop as our data source is because it provided us with almost all of the information we required in a nice format, with a separate heading for "Features & Amenities" which made the data easy to extract and quantify [Appendix: Fig. 2.1.]. Another reason we preferred Renthop was because of the low amount of security it had against web scraping. Other websites had a strong security system set up against web scraping, so if someone did try to web scrape, they would just be returned with "Error 403." There are workarounds to this error, but given our time constraints, we decided it would be best to continue with Renthop. Finally, Renthop has a feature known as "Hop Score" which is a proprietary algorithm score that the site awards to each apartment listing depending on the quality of the listing, the reputation of the manager, and the freshness of the listing. So, we decided to incorporate this feature into our model as well.

Since the average ratings of nearby restaurants were not available on the website, we implemented a Google API query model to extract that information. Using the model, we got the ratings of the five closest restaurants to that apartment and calculated the average.

The biggest challenge we faced during the data extraction, apart from the "Error 403", was during the web scraping of Renthop. Initially, the code would receive an error on a random page and a random listing. We figured out that this was due to an ad or a captcha popping up at a random time. Initially, to counter this, we used a double try and except code with a while loop so the code would keep on trying to get into the page until it was successful, but this caused the whole code more than eight hours to run. Then, we implemented a sleep function from the time library, so the code would wait before going to the next page. This shortened the runtime from 8 hours to 25 minutes. Thus, we were then ready to move onto the data processing stage.

## 3. Data Preprocessing

For the Price column, we removed the $ in each row and converted the data into floats.

For all data in the Number of Beds and Number of Bathrooms columns, we removed the units and converted them into floats. We considered the number of beds in a studio, loft, and private room to all be 1, and that of flexed rooms to be the number of rooms after expansion.

To obtain the foot distance/walk time to the closest subway station, we extracted two columns from the Distance from Closest Subway column. One of those was the new distance column, where we converted mile distances into feet and kept feet distances as they were for easier analysis. The other column was times_to_subway which measured the walk time in minutes.

The original and processed data frames can be referred to in [Appendix: Fig. 3.1.].

## 4. Exploratory Data Analysis and Data Visualization

After the cleaning and preprocessing part, we decided to go ahead with the Exploratory Data Analysis. We started by visualizing how each of our independent variables, such as (the number of beds, number of bathrooms, etc.) was correlated

with our dependent variable, Price. We used a correlation matrix, to display the correlation coefficients for all the different variables, and to identify and analyze patterns in the given data. We dropped the New_distance column, which gave the distance to the nearest subway in miles, as it was heavily correlated with the times_to_subway column, to avoid any multicollinearity issues. Following this, we got the final correlation matrix between our independent variables and the dependent variable- price [Appendix: Fig. 4.1.]. We also created a Correlation Heat Map to visualize the extent of correlation between all variables better [Appendix: Fig. 4.2.]. We noticed factors such as number_of_bathrooms, elevator, hop_score, and restaurant_ratings are more highly correlated with the Price as compared to number_of_bedrooms.

Next, we wanted to observe the price range of the apartment data that we had acquired. We decided to build a histogram using the matplotlib library and plotted the mean and standard deviation of the prices of the apartments [Appendix: Fig. 4.3.]. We wanted to explain and judge how apartments in our dataset were characterized by the variables we had chosen. We plot histograms to see how many apartments can be 'grouped by' a particular independent variable [Appendix: Fig. 4.4.]. To understand the relationship between how each variable independently affects the price, we then performed linear regression. Before that, we wanted to find the best coefficients to fit the line of linear regression in order to minimize the root mean sum of squared errors or the RMSE [Appendix: Fig. 4.5.]. We also wanted to visualize and identify whether noise is affecting our results and continue with feature selection for our prediction models.

We then plotted the line against a scatter plot for each variable to understand how our data points are spread around the line of univariate regression [Appendix: Fig. 4.6.]. We were able to hypothesize that each variable affects the price positively. But since we know that correlation is not causation, we needed to perform a multivariate regression analysis to see how when all the variables are combinedly analyzed; the price varies holistically based on the independent variables and on whether or not, they are statistically significant to be considered for further analysis and ultimately, prediction.

## 5. Data Analysis

After dropping some missing values, we had (ones except the Size column) and the correlated features, we were left with 1,106 rows and 15 features to tackle our prediction task.

### 5.1. Size Prediction

Unfortunately, at Renthop.com, size values were missing for most of the listings. To be exact, we just had 269 listings where we knew the size in square feet (sqft). However, since we believed size is a major determinant for the price of a listing, we concluded on not removing the column as a whole but making predictions on the missing data. Rather than making a simple prediction like putting the mean or median values of other rows, we built a prediction model for the size variable using the 13 independent variables in the dataset we had (all features except the Price variable). We thought this would give a more accurate representation of the size of an apartment since our features included variables like the number of bedrooms and the number of bathrooms. Initially, we divided the dataset into 2, with one having size values (df_size) and the other one having missing values for the size column. Then on the df_size, we used an 80:20 train-test split for the prediction task since we already had limited data and wanted to use as much information as we could for the training task. After our analysis, we found Random Forest gave the best predictions since it gave an out-of-sample (test) $R^2$ score of 0.66. Then we ran our random forest model for the rest of the 837 entries without a size value and made predictions on them. Finally, we combined the two datasets to use them for our main prediction task then.

### 5.2. Price Prediction

For our main prediction task, we used four different models: linear regression, ridge regression, random forest regressor, and k-neighbors regressor. We used sckit-learn library in python to conduct these analyses. To have consistency across models, especially when we are comparing them, we used the same train-test split of 70:30, the same seed, and the same features when running each model.

Initially, we took a look at linear regression. However, we got a 0.30 $R^2$ in the test set [Appendix: Fig. 5.1.], which indicated our problem was probably not linear and couldn't be handled with linear regression. However, when looking at the coefficients of the model, we can still make some comments about the significant variables. We saw most of the variables (ones except 'Doorman', 'laundry_in_building', and 'elevator') to be significant [Appendix: Fig. 5.2.]. However, we can't say anything about the values of the coefficients since the variables weren't scaled, but it is still something we can compare with the values we'll get from other models.

We conducted a ridge regression, with scaled variables, to see if multicollinearity was a problem in our problem. But even when we took into account it, our test $R^2$ just increased to 0.31, so with such a minimal difference, we understood,

multicollinearity was not an issue for us, which was in line with our expectations since we removed highly correlated independent variables.

Then we ran a random forest model and saw the test $R^2$ increase to 0.58, which was the best we had seen until that point. We tried to optimize the hyperparameters for this algorithm to get the optimal parameters and finalized with 100 estimators and a max_depth of 5. When looking at the feature importance [Appendix: Fig. 5.3.], we saw that the main variable of interest was the size, which had a major impact on price prediction. It was followed by restraunt_ratings and hop_score. This was against our intuition since one was an externally calculated variable by us, which we thought would have some impact but not as much as it had here, and the hop score was a score calculated by Renthop on the quality of the listing. This indicates that there are other characteristics of a listing of an apartment that could be influential on its price more than its physical characteristics.

Finally, we ran a k-nearest neighbors regressor model, where we checked all possible k values between 1 to 15 to find the optimal k [Appendix: Fig. 5.4.]. We concluded with a k=3, where we reached the highest test $R^2$ of 0.52.

Based on the calculations, we recommended making predictions on a random forest model, which we believe understands the underlying effects of our variables on price better than other models.

## 6. Conclusion & Evaluation

Based on our analysis above, we can summarize the following points. First, in the data visualization part, we noticed factors such as number_of_bathrooms, elevator, hop_score, and restaurant_ratings are more highly correlated with the Price as compared to number_of_bedrooms. This fact observed in this section set the stage for our subsequent analysis. Next, in terms of the missing size data, we used a random forest model for prediction, and we found Random Forest gave the best predictions since it gave an out-of-sample (test) $R^2$ score of 0.66; in the face of a large number of missing values, we believe that this model gives a reasonable prediction. The facts observed in this section set the stage for our subsequent analysis. Then, in our main prediction part, we used four different models, linear regression, ridge regression, random forest regressor, and kneighbors regressor. After different trials, we think the random forest model with $R^2$ increased to 0.58 is the best model for our prediction. The results tell us that size,restraunt_ratings, and hop_score are significant predictors of the price of apartments. Finally, we ran a k-nearest neighbors regressor model, and we found 3 is the best k which gave the highest test $R^2$ of 0.52.

Baes on all the results, we decided to use a random forest model for our final prediction. For our entire forecasting analysis process, we believe that there are two possible biases. The first is that due to a large number of missing key forecast indicator “size”, the forecasted size data we used can cause some errors in the final forecast results. Second, we believe that there is some error in the significant predictors we obtained because restraunt_ratings and hop_score are manually counted, and these indicators are closely related to some variables in the apartment itself, but we did not find these more intuitive predictors.

# Appendix

## 2. Data Extraction:

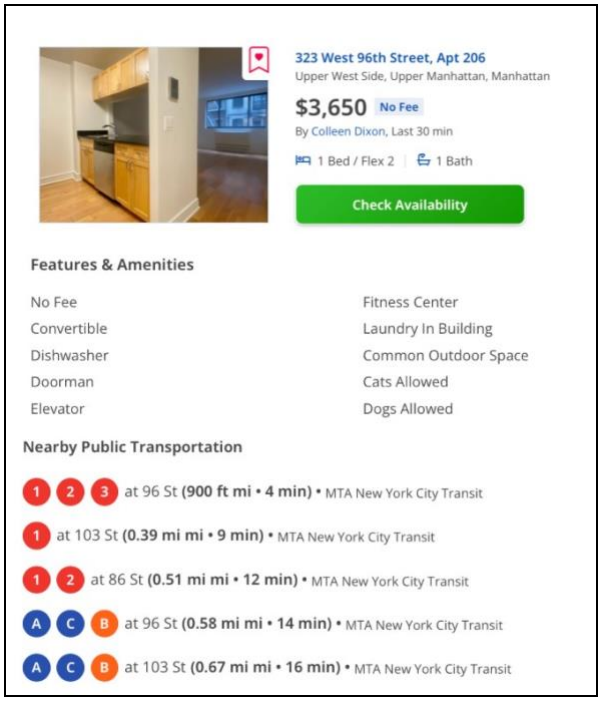



Fig. 2.1. Renthop Apartment Page for Data Extraction

## 3. Data Preprocessing:

| Address | Price | Number of Beds | Number of Bathrooms | Size (sqft) | Distance from Closest Subway | Hop Score | Laundry In Building | Doorman | No Fee | Pool | Dishwasher | Elevator |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 808 Columbus Avenue, Apt 10E | $4,913 | 1 Bed | 1 Bath | 652.0 | (0.23 mi mi • 5 min) • | 100.0 | 0 | 0 | 1 | 0 | 0 | 1 |
| Amsterdam Avenue | $2,600 | Studio | 1 Bath | NaN | (680 ft mi • 3 min) • | 97.5 | 1 | 1 | 1 | 0 | 0 | 1 |
| 965 Amsterdam Avenue | $2,075 | Private Room | 1 Bath | 97.0 | (0.21 mi mi • 5 min) • | 95.4 | 0 | 0 | 1 | 0 | 0 | 0 |
| west 68 st | $2,200 | Loft | 1 Bath | NaN | (840 ft mi • 4 min) • | 73.2 | 0 | 0 | 1 | 0 | 0 | 1 |
| West 93 St | $4,850 | 2 Bed Flex 3 | 1.5 Bath | NaN | (0.22 mi mi • 5 min) • | 68.6 | 1 | 1 | 1 | 0 | 1 | 1 |

| Address | Price | Number of Beds | Number of Bathrooms | Size (sqft) | Hop Score | Laundry In Building | Doorman | No Fee | Pool | Dishwasher | Elevator | times_to_subway | new_distance |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 808 Columbus Avenue, Apt 10E | 4,913 | 1 | 1 | 652.0 | 100.0 | 0 | 0 | 1 | 0 | 0 | 1 | 5 | 1214.4 |
| Amsterdam Avenue | 2,600 | 1 | 1 | Nan | 97.5 | 1 | 1 | 1 | 0 | 0 | 1 | 3 | 680 |
| 965 Amsterdam Avenue | 2,075 | 1 | 1 | 97.0 | 95.4 | 0 | 0 | 1 | 0 | 0 | 0 | 5 | 1108.8 |
| west 68 st | 2,200 | 1 | 1 | Nan | 73.2 | 0 | 0 | 1 | 0 | 0 | 1 | 4 | 840 |
| West 93 St | 4,850 | 3 | 1.5 | Nan | 68.6 | 1 | 1 | 1 | 0 | 1 | 1 | 5 | 1161.6 |

Fig. 3.1. Partial Data Frames from Before and After Data Cleaning

## 4. Exploratory Data Analysis and Data Visualization Results:

```
df.corr().style.background_gradient(cmap='coolwarm')
```

| | Price | number_of_beds | number_of_bathrooms | hop_score | laundry_in_building | Doorman | no_fee | Pool | Dishwasher | Elevator |
|---|---|---|---|---|---|---|---|---|---|---|
| Price | 1.000000 | 0.336396 | 0.207680 | -0.040490 | 0.152346 | 0.168073 | 0.033723 | 0.132670 | 0.270071 | 0.085284 |
| number_of_beds | 0.336396 | 1.000000 | 0.416787 | -0.162586 | -0.005872 | -0.025082 | 0.205122 | -0.051954 | 0.211982 | -0.082682 |
| number_of_bathrooms | 0.207680 | 0.416787 | 1.000000 | -0.001863 | -0.093640 | -0.165073 | 0.009595 | -0.039996 | 0.012648 | -0.241031 |
| hop_score | -0.040490 | -0.162586 | -0.001863 | 1.000000 | -0.061906 | -0.029718 | -0.021899 | 0.097056 | -0.124209 | -0.048351 |
| laundry_in_building | 0.152346 | -0.005872 | -0.093640 | -0.061906 | 1.000000 | 0.519224 | 0.146682 | 0.098886 | 0.312555 | 0.541191 |
| Doorman | 0.168073 | -0.025082 | -0.165073 | -0.029718 | 0.519224 | 1.000000 | 0.195910 | 0.147666 | 0.265721 | 0.603563 |
| no_fee | 0.033723 | 0.205122 | 0.009595 | -0.021899 | 0.146682 | 0.195910 | 1.000000 | -0.018280 | 0.215002 | 0.145559 |
| Pool | 0.132670 | -0.051954 | -0.039996 | 0.097056 | 0.098886 | 0.147666 | -0.018280 | 1.000000 | 0.027579 | 0.095185 |
| Dishwasher | 0.270071 | 0.211982 | 0.012648 | -0.124209 | 0.312555 | 0.265721 | 0.215002 | 0.027579 | 1.000000 | 0.190740 |
| Elevator | 0.085284 | -0.082682 | -0.241031 | -0.048351 | 0.541191 | 0.603563 | 0.145559 | 0.095185 | 0.190740 | 1.000000 |
| times_to_subway | 0.163402 | -0.009243 | 0.027403 | 0.048420 | 0.050726 | 0.106040 | 0.109411 | 0.205248 | 0.127552 | 0.034205 |
| restaraunt_ratings | 0.069483 | 0.066389 | 0.073466 | -0.034801 | -0.023244 | -0.023785 | 0.055547 | 0.012862 | 0.108335 | -0.041969 |

Fig. 4.1. Correlation Matrix

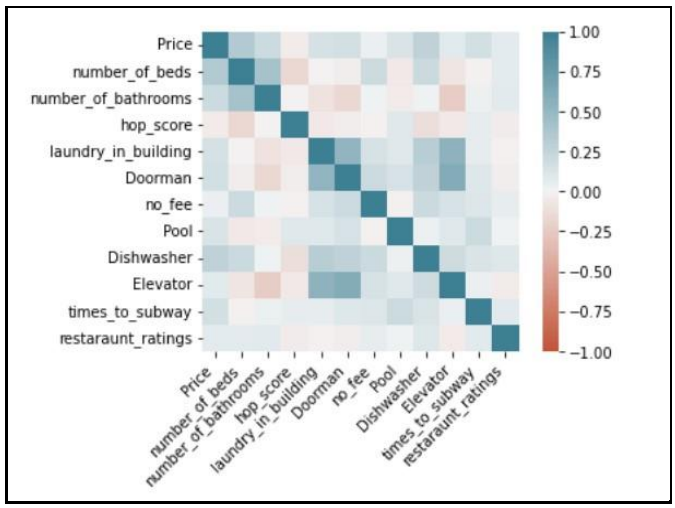



Fig. 4.2. Correlation Heat Map

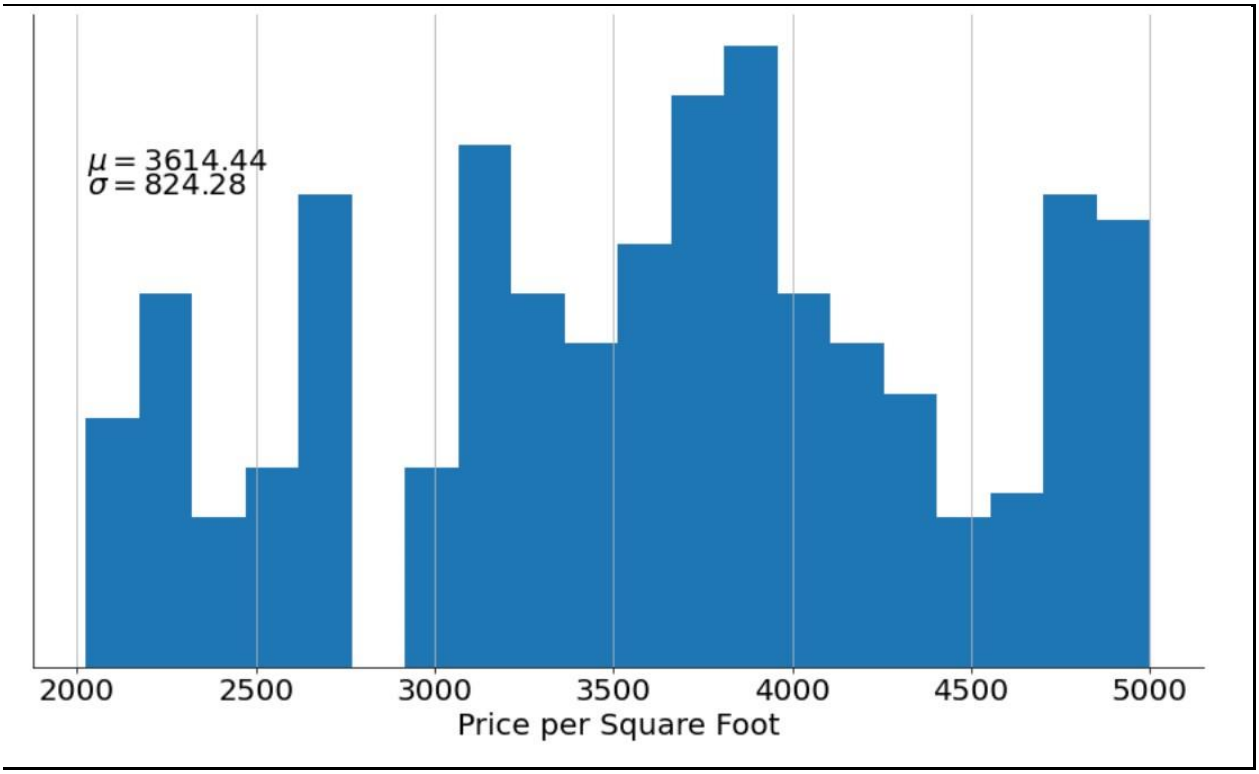
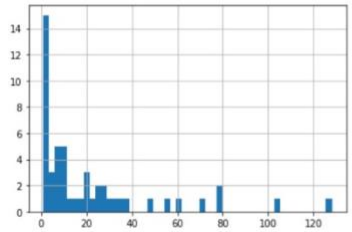



Fig. 4.3. Frequency of Prices with Mean and Standard Deviation

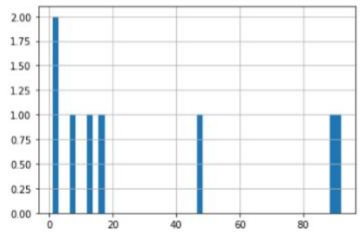
```
df_final.groupby('restaraunt_ratings').size().sort_values(ascending=False).hist(bins=50)
```

```
<AxesSubplot:>
```



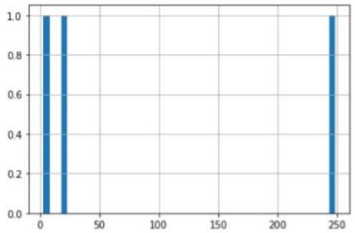
```
df_size.groupby('times_to_subway').size().sort_values(ascending=False).hist(bins=50)
```

```
<AxesSubplot:>
```



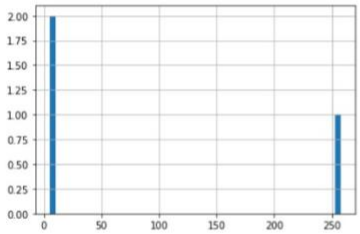
```
df_size.groupby('number_of_beds').size().sort_values(ascending=False).hist(bins=50)
```

```
<AxesSubplot:>
```



```
df_size.groupby('number_of_bathrooms').size().sort_values(ascending=False).hist(bins=50)
```

```
<AxesSubplot:>
```



```
df_size.groupby('hop_score').size().sort_values(ascending=False).hist(bins=50)
```

```
<AxesSubplot:>
```

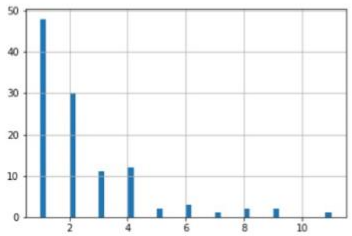


Fig. 4.4. Grouping Apartments by Attributes

```python
def create_plot(var):
    '''
    This function takes the name of a variable in df_cleaned, and
    produces a plot of the variable against the price
    '''

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

    plt.plot(df_size[var], df_size.Price, linewidth=0, marker='x')

    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)

    plt.xlabel(var, fontsize=20)
    plt.ylabel('Price', fontsize=20)

    sns.despine()
```

```python
def find_beta(var):
    df_size[var]=df_size[var].astype(float)
    return df_size.Price.corr(df_size[var])*df_size.Price.std()/df_size[var].std()

def find_alpha(var):
    df_size[var]=df_size[var].astype(float)
    return df_size.Price.mean() - find_beta(var)*df_size[var].mean()

beta = find_beta('number_of_beds')
alpha = find_alpha('number_of_beds')
print(f'$y = {round(alpha,2)} + {round(beta,2)}x$')
```

```
$y = 2966.22 + 595.12x$
```

Fig. 4.5. Finding the Best Coefficients Minimizing the RMSE

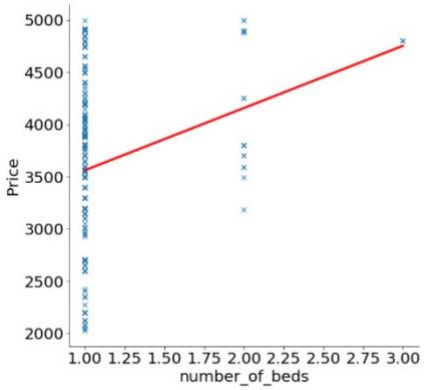



Fig. 4.6.a. Price vs. Number of Beds

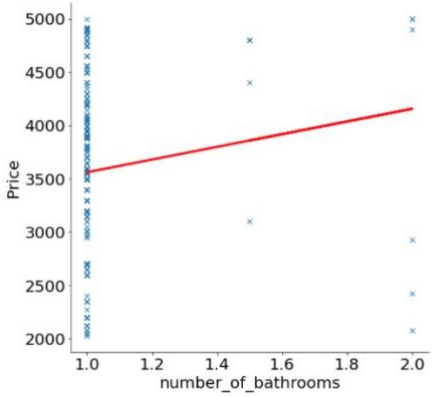



Fig. 4.6.b. Price vs. Number of Bathrooms

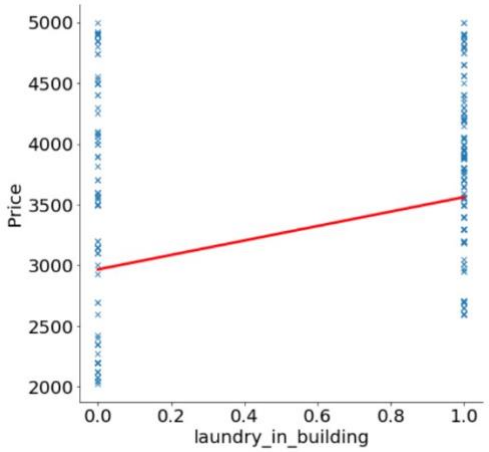



Fig. 4.6.c. Price vs. In-Building Laundry

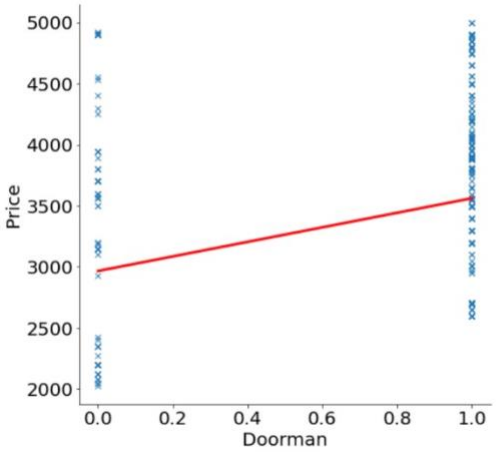



Fig. 4.6.d. Price vs. Doorman

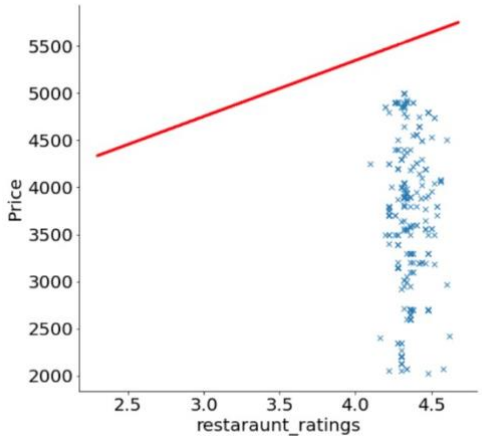



Fig. 4.6.e. Price vs. Average Restaurant Ratings

## 5. Data Analysis

| Model | Parameters | R^2 Score on Test Set |
|---|---|---|
| **Linear Regression** | | 0.30 |
| **Ridge Regression** | scaled variables | 0.31 |
| **Random Forest Regressor** | n_estimators = 100<br>max_depth = 5 | 0.58 |
| **K-neighbours Regressor** | k = 3 -> optimal | 0.52 |

Fig. 5.1. Test Set $R^2$ Results for All Prediction Models

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Price | **R-squared:** | 0.305 |
| **Model:** | OLS | **Adj. R-squared:** | 0.298 |
| **Method:** | Least Squares | **F-statistic:** | 40.03 |
| **Date:** | Sun, 11 Dec 2022 | **Prob (F-statistic):** | 3.88e-78 |
| **Time:** | 13:46:05 | **Log-Likelihood:** | -8785.9 |
| **No. Observations:** | 1106 | **AIC:** | 1.760e+04 |
| **Df Residuals:** | 1093 | **BIC:** | 1.766e+04 |
| **Df Model:** | 12 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **Intercept** | -402.8294 | 611.440 | -0.659 | 0.510 | -1602.558 | 796.899 |
| **number_of_beds** | 158.4843 | 44.995 | 3.522 | 0.000 | 70.198 | 246.771 |
| **number_of_bathrooms** | 492.5252 | 114.326 | 4.308 | 0.000 | 268.202 | 716.848 |
| **size_sqft** | 1.8859 | 0.164 | 11.500 | 0.000 | 1.564 | 2.208 |
| **hop_score** | 7.9684 | 2.648 | 3.009 | 0.003 | 2.772 | 13.165 |
| **laundry_in_building** | 62.8160 | 54.241 | 1.158 | 0.247 | -43.611 | 169.243 |
| **Doorman** | 7.7508 | 57.374 | 0.135 | 0.893 | -104.826 | 120.327 |
| **no_fee** | -206.4291 | 47.292 | -4.365 | 0.000 | -299.222 | -113.636 |
| **Pool** | 535.2970 | 138.818 | 3.856 | 0.000 | 262.918 | 807.676 |
| **Dishwasher** | 233.2021 | 46.119 | 5.057 | 0.000 | 142.711 | 323.693 |
| **Elevator** | 77.7104 | 63.518 | 1.223 | 0.221 | -46.921 | 202.342 |
| **times_to_subway** | 32.9831 | 13.680 | 2.411 | 0.016 | 6.141 | 59.825 |
| **restaraunt_ratings** | 301.7701 | 123.497 | 2.444 | 0.015 | 59.453 | 544.087 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 38.794 | **Durbin-Watson:** | 1.935 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 25.873 |
| **Skew:** | 0.252 | **Prob(JB):** | 2.41e-06 |
| **Kurtosis:** | 2.445 | **Cond. No.** | 1.85e+04 |

Fig. 5.2. Linear Regression Summary

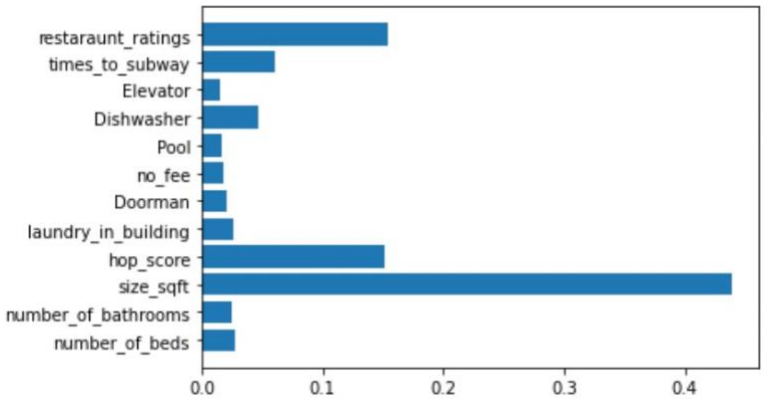



Fig. 5.3. Feature Importance for Random Forest model

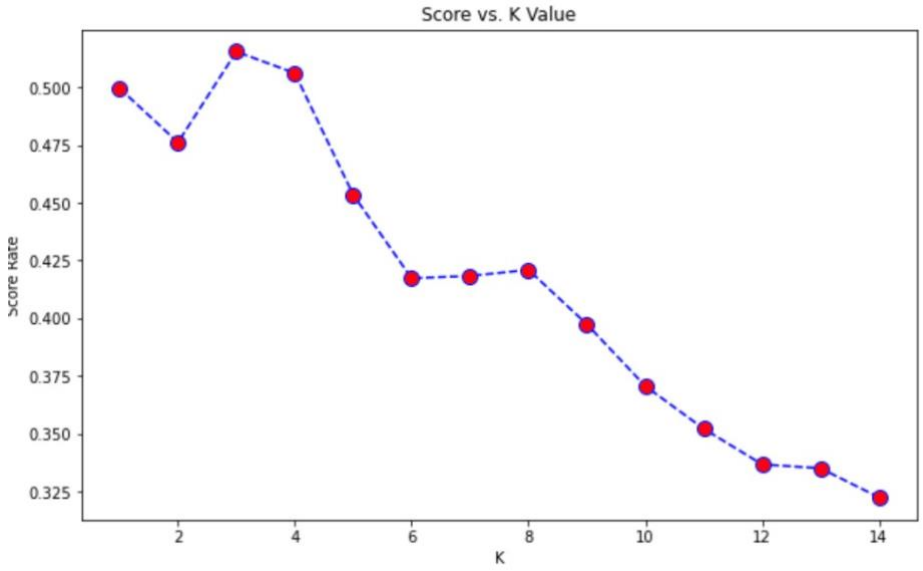



Fig. 5.4. Finding Optimal K for K-neighbors Regressor Model