Project 1 In-depth Analysis

After checking which variables are statistically significant to our target variable, `num_of_cancellations`, we proceed to choose which machine learning model best suites our project.

In this study, we try out six different models:

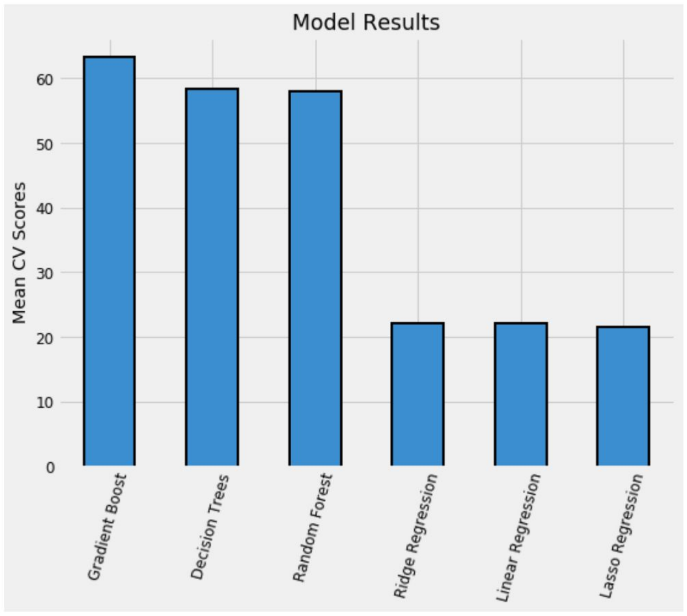
1. Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Decision Trees
5. Gradient Boosting
6. Random Forest

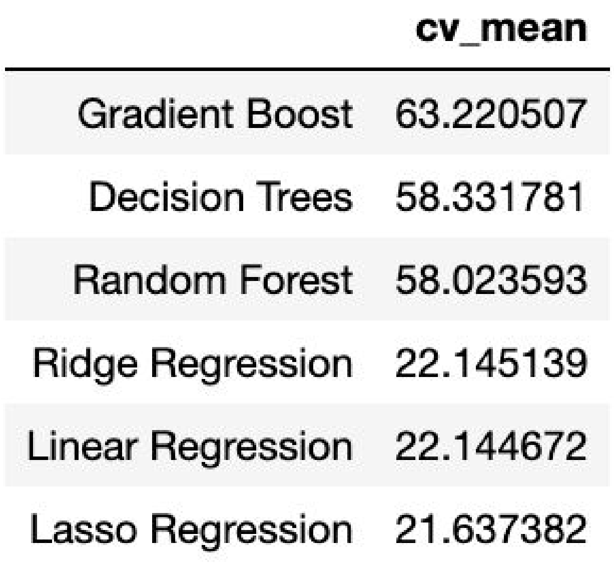
Dealing with Categorical Features
We have two categorical variables in our data frame: `room_type` and `neighbourhood`. Since Scikit Learn rejects categorical features by default, we make sure to create dummies for these two variables.

Training Set and Test Set
We are using 70% of the data as the training set and the remaining 30% as the test set.

Scoring
In this project, we are measuring accuracy by getting the average of 5-fold cross validation scores of each model. Since this is a regression problem, I use R^2 as the scoring metric. The R^2 is the indication of the goodness of fit of a set of predictions to the actual value.

Comparing Model Performance

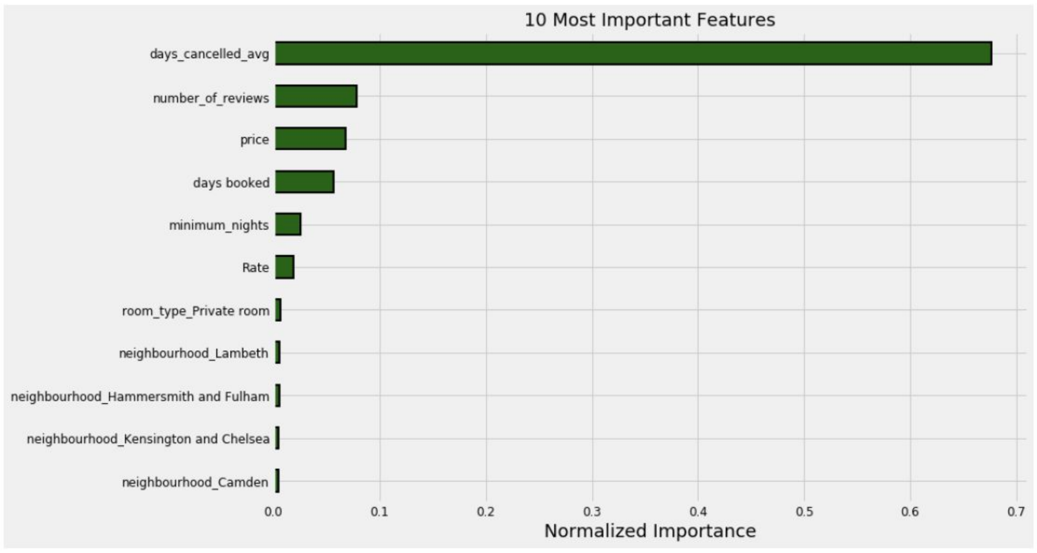




	cv_mean
Gradient Boost	63.220507
Decision Trees	58.331781
Random Forest	58.023593
Ridge Regression	22.145139
Linear Regression	22.144672
Lasso Regression	21.637382

The Gradient Boosting Regressor performed best while Lasso Regression performed worse. To further optimize the performance of the Gradient Boosting Regressor, we go back into feature selection by determining which features create ‘noise’. Doing so will help us prevent overfitting.

Feature Selection



We see that the non-categorical variable that has the least importance is the crime rate of each neighborhood. We simply drop this from our data frame and fit it to our Gradient Boosting Regressor again to see if the average CV score improves.

After running this, we see that average CV score improves from 63.22% to 63.36%. This indicates that the feature only added noise to our model.

Hyperparameter Tuning

To further optimize our model’s performance, we try to tune some of the parameters of the `GradientBoostingRegressor()` function. In this project, I try to find the best values of the following:

- max_depth
- n_estimators
- learning_rate

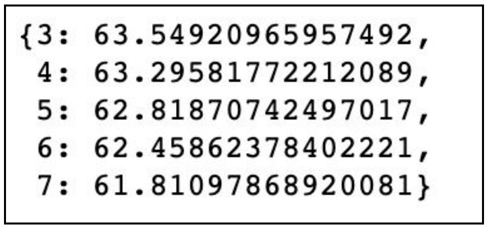
To do so, we create a list of possible values for each of the parameters for the forloop to search over. We then take the parameter that performs best.

The results are the following:
`n_estimators`

```
{50: 63.54920965957492,
 100: 63.356236092445364,
 200: 62.96948529810927,
 300: 62.38900703879856,
 400: 61.95043209337545,
 500: 61.59295400799691}
```

At an average CV score of 63.55%, our best `n_estimator` is 50.

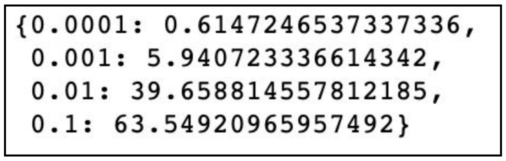
`max_depth`



```
{3: 63.54920965957492,
 4: 63.29581772212089,
 5: 62.81870742497017,
 6: 62.45862378402221,
 7: 61.81097868920081}
```

At an average CV score of 63.55%, our best `max_depth` is 3.

`learning_rate`



```
{0.0001: 0.6147246537337336,
 0.001: 5.940723336614342,
 0.01: 39.658814557812185,
 0.1: 63.54920965957492}
```

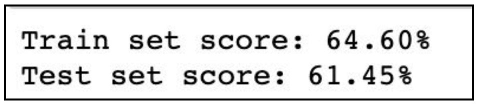
At an average CV score of 63.55%, our best `learning_rate` is 0.1.

After getting the best `n_estimator` of 50, our best CV score has remained at 63.55% when looking for best `max_depth` and `learning_rate` because the best values we found for `max_depth` and `learning_rate` are the GradientBoostingRegressor()'s default values for the parameters.

Despite the feature selection and hyperparameter tuning, the score only improves by a few percentage points. This tells us that we need more features for better prediction.

Results

After selecting the best performing model, and further optimizing it through feature selection and hyperparameter tuning, our final results are:



```
Train set score: 64.60%
Test set score: 61.45%
```

As can be expected, the test set score is less than the train set score. Although, the difference is not much, which indicates that overfitting was not a problem.