BIKE SHARING PREDICTION

Contents

1 INTRODUCTION

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

Several bike/scooter rides sharing facilities (e.g., Bird, Capital Bikeshare, Citi Bike) have started up lately especially in metropolitan cities like San Francisco, New York, Chicago and Los Angeles, and one of the most important problem from a business point of view is to predict the bike demand on any particular day. While having excess bikes results in wastage of resource (both with respect to bike maintenance and the land/bike stand required for parking and security), having fewer bikes leads to revenue loss (ranging from a short term loss due to missing out on immediate customers to potential longer term loss due to loss in future customer base), Thus, having an estimate on the demands would enable efficient functioning of these companies.

The goal of this project is to combine the historical bike usage patterns with the weather data to forecast bike rental demand. The data set consists of hourly rental data spanning two years. The training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month

2 DATA OVERVIEW

The data set is obtained from Kaggle with the following column labels

- datetime - hourly date + timestamp
- season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
- holiday - whether the day is considered a holiday
- workingday - whether the day is neither a weekend nor holiday
- weather -
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp - temperature in Celsius
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed
- casual - number of non-registered user rentals initiated
- registered - number of registered user rentals initiated
- count - number of total rentals

In this project, we use the 8 of the above columns as the feature set *['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp', 'atemp', 'humidity', 'windspeed'*] to predict the value of '*count'*. The other two columns (*casual* and *registered*) comprises of the split-up of the target column *'count'*. The provided data consists of 10886 observations with 11 column variables (excluding the datetime column - which has been used as an index)

3 EXPLORATORY DATA ANALYSIS

Before we start modeling, let us first get an idea on how the number of bike rentals depend on the various features provided to us one by one.

3.1 Weather

Below are the plots that show the average bike rental (count) for various weather conditions.

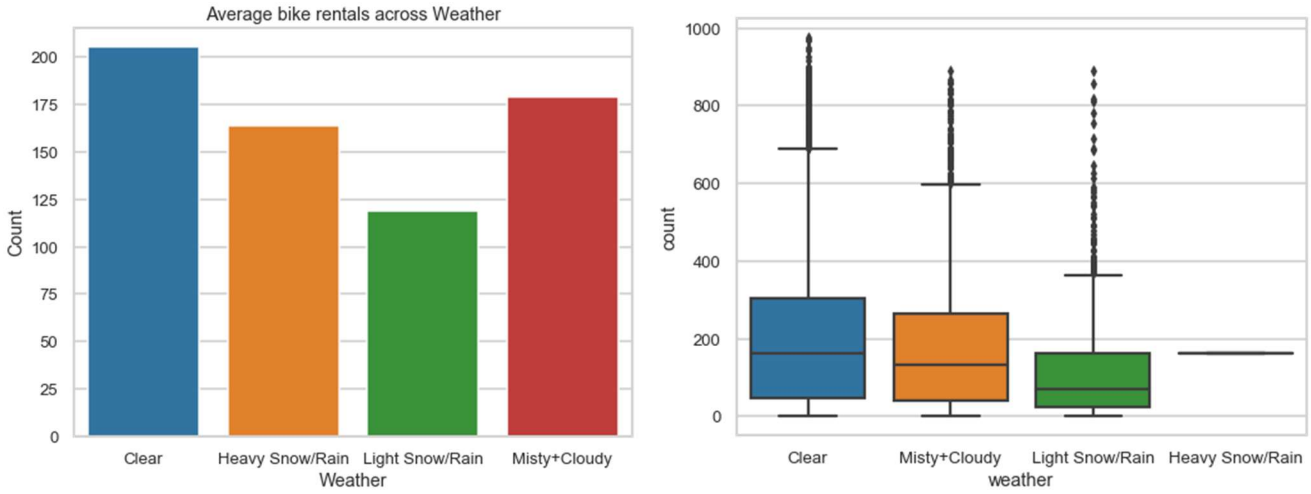


Figure 3-1: Average bike rental across Weather conditions

We observe higher bike rentals when the weather is more clear and sunny. We also notice that there is a single instance where there were rentals under heavy rain/snow condition. Few possibilities arise:

- Could be an error in logging
- Could be an outlier in the data set
- Maybe, observations were made at a time when the weather was good. But weather conditions logged sometime later in the same hour when the conditions were heavy rains/snow

We will investigate this later in the Data Cleaning section (5.2.1) to explore more.

3.2 Season .

Below are bar plots and box plots of bike rental counts across the 4 seasons.

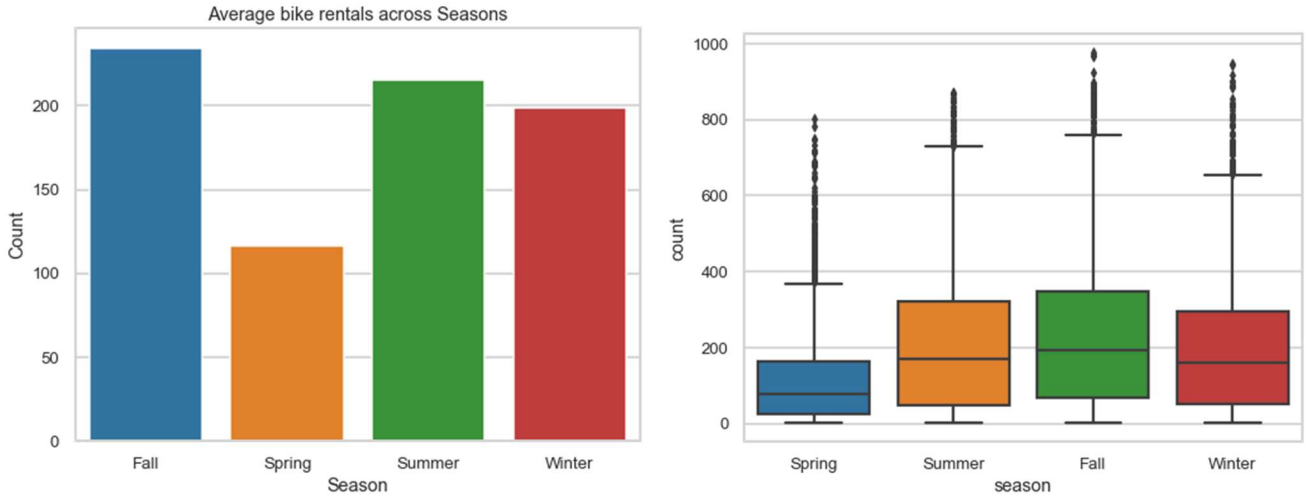


Figure 3-2: Average bike rental across different Seasons

Bike reservations are highest during the Summer (April to June) and Fall (July to September) season and least during the Spring season (January to March)

3.3 Working Day

Below are bar plots and box plots of average bike rental counts on working and non-working days.

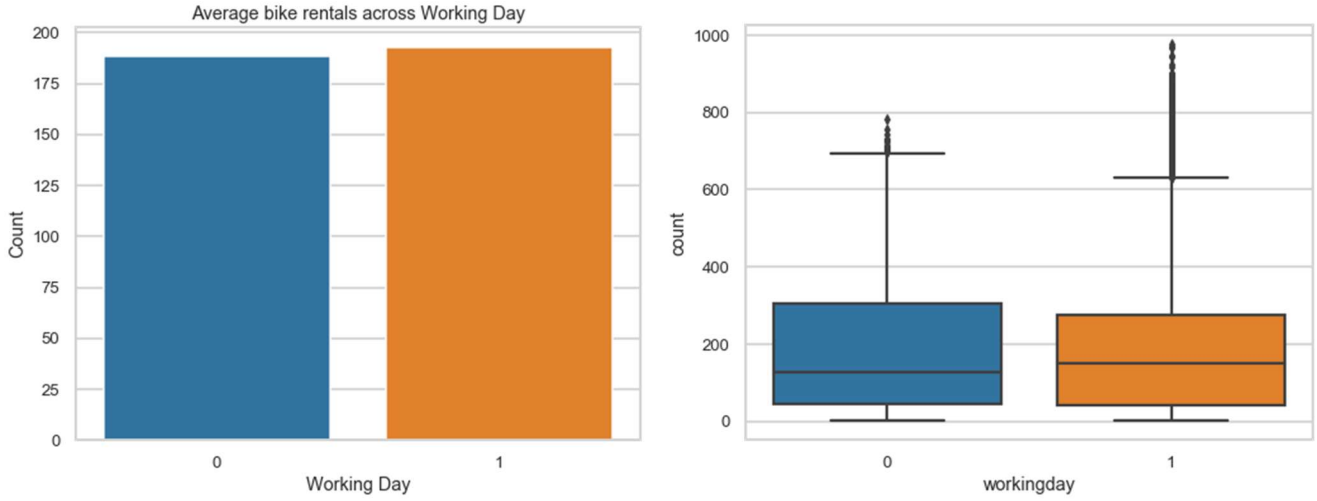


Figure 3-3: Average bike rentals on Working and Non-Working days

We can see that the mean count is similar. However, we see more outliers on working days.

3.4 Holiday .

Below are bar plots and box plots of average bike rental counts on holidays and non-holidays. Holidays correspond to non-working days excluding weekends.

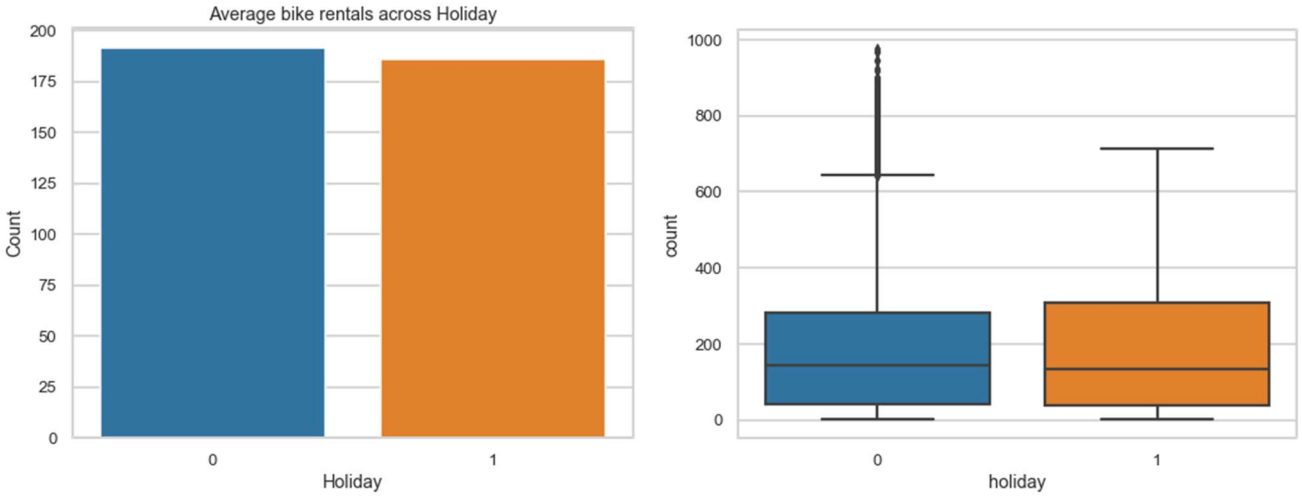


Figure 3-4:Average bike rentals on Holidays and Non-Holidays

Mean count seem to be similar with more outlier data points for non-holidays.

3.5 Temperature

Below is a histogram plot for average bike rental count given a certain range of temperature for working and non-working days.

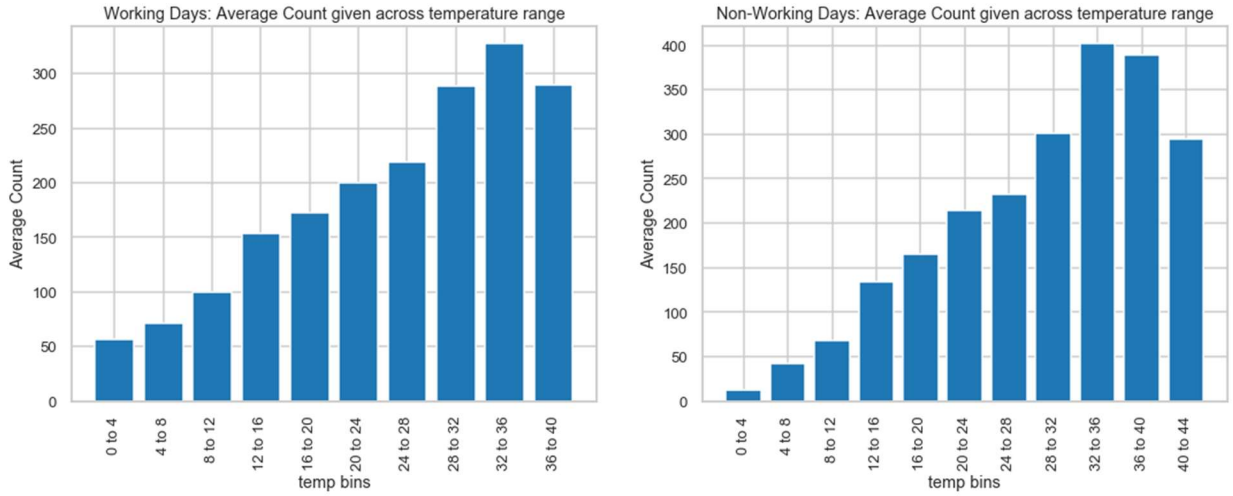


Figure 3-5: Average bike rentals across different Temperature

From the above histogram plot, we can see that there is a steady increase in the average bikes rented with temperature with a small decrease at the highest temperature bin. Temperature between 32 and 36 degrees Celsius seems to be the ideal temperature for biking.

3.6 Hour .

Now let us examine how the average bike count varies across the day (vs. hour) for the below three categories

- Working day and Non-Working day in Figure 3-6
- Casual and Registered Users in Figure 3-7
- Different days of the week in Figure 3-8

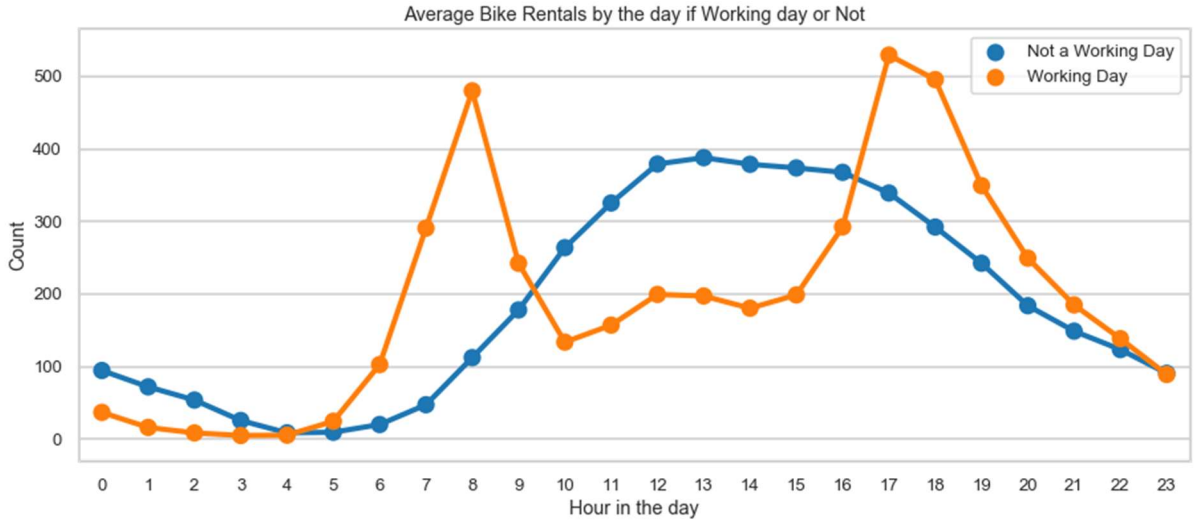


Figure 3-6: Hourly average bike rentals for Working day or Non-Working day

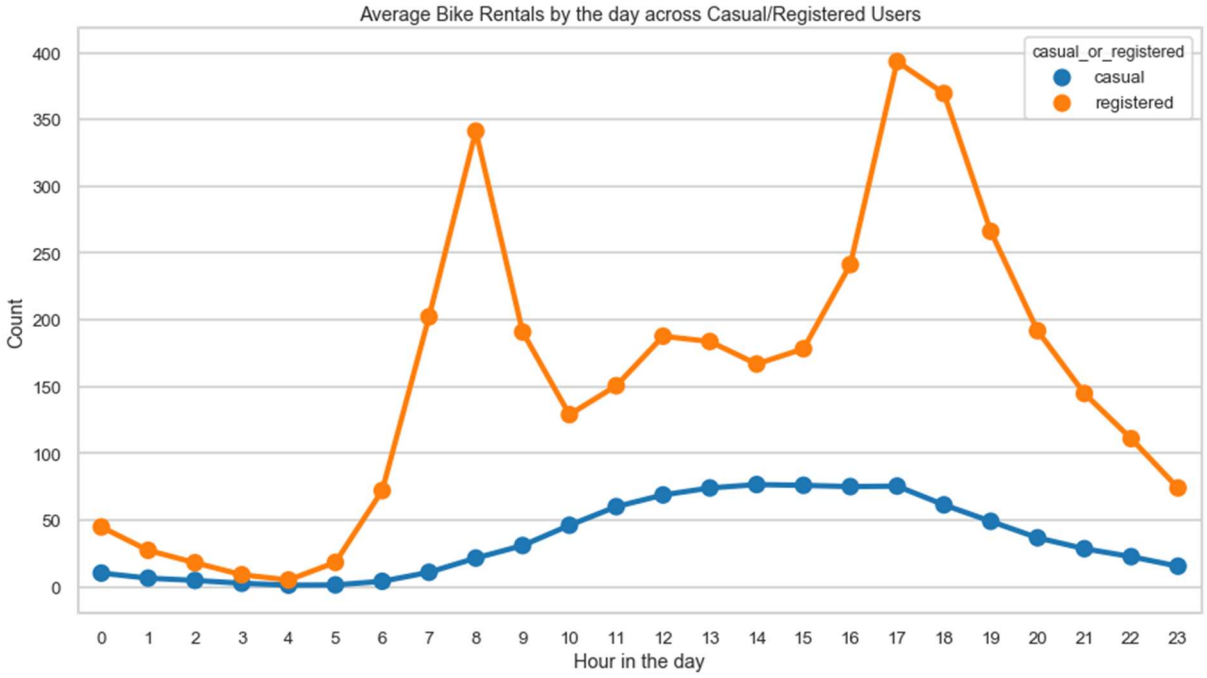


Figure 3-7: Hourly average bike rentals for Casual and Registered Users

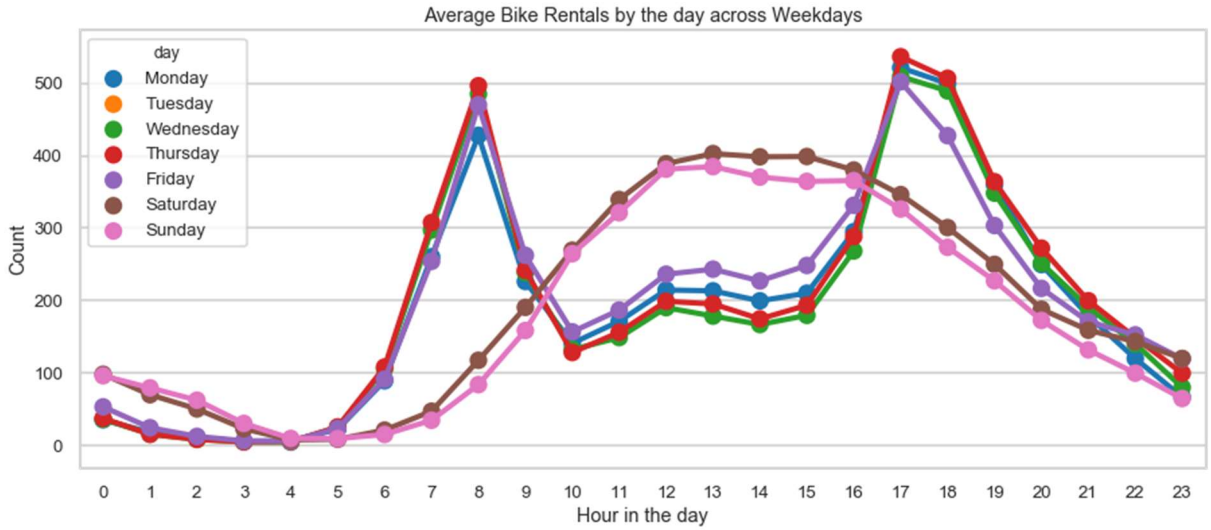


Figure 3-8: Hourly average bike rentals for different days of the week

There are few interesting observations that we can make from the above 3 figures

- Firstly, we see that there are 2 patterns in the bike rentals during the day
    - Working Day: First pattern where there is a peak in the rentals at around 8am and another at around 5pm. These correspond to working local bikers who typically are *registered* and go to work on working day which are *Monday* to *Friday*
    - Non-Working Day: Second pattern where there is a steady increase in bike rental during the day reaching the peak at around noon and a gradual decrease in the count in the evening after 4pm. These correspond to probably tourists who typically are *casual* users who rent/drop off bikes uniformly during the day and tour the city of Washington on non-working days which typically are *Saturday* and *Sunday*
- The average number of the casual users are in general lower than the average number for non-working days and weekends. This seem to indicate that there are several registered users too, who follow the same non-working day rental trend during weekends and non-working days

3.7 Month .

Below plot contains the average bike count over each month of a calendar year.

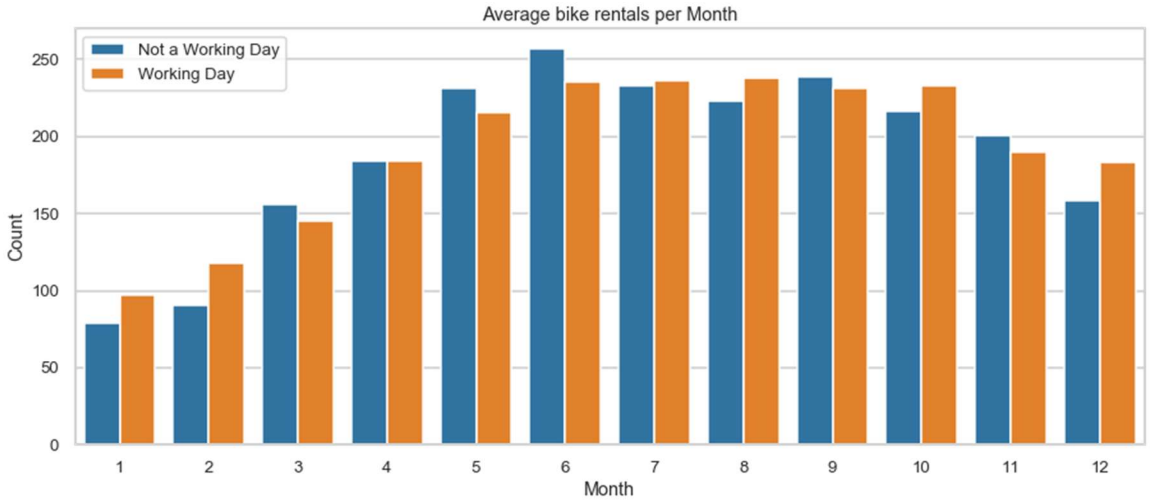


Figure 3-9:Average Monthly bike rental count

The above figure is highly correlated with the seasons bar plot since seasons plot effectively is the average count for 3 of these months. We can see that the most rentals are in the months of June and May while least are on January and February.

4 CORRELATION ANALYSIS

4.1 Heatmap

Below is a heatmap plot of the correlation between all the numerical columns

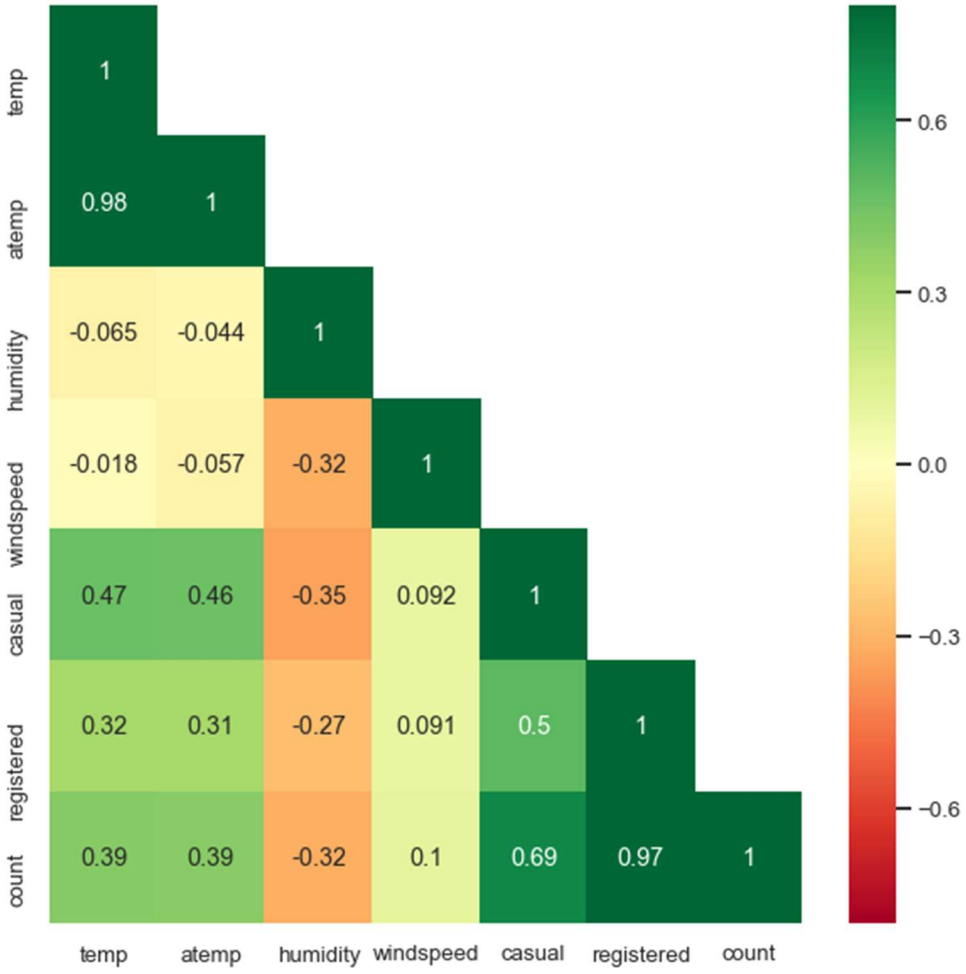


Figure 4-1: Heatmap of the correlation between all the numerical features

We can infer the following from the above heatmap

- *temp* (true temperature) and *atemp* (feels like temperature) are highly correlated, as one would expect
- *count* is highly correlated with *casual* and *registered* as expected since count = casual + registered

- We see a positive correlation between *count* and *temperature*. Note that this is only true for the range of temperatures provided
- We see a negative correlation between *count* and *humidity*. The more the humidity, the less people prefer to bike
- *Count* has a weak dependence on *windspeed*

4.2 Regression Plots

Below are regression plots of the bike rental count vs. Temperature, Humidity and Windspeed, respectively.

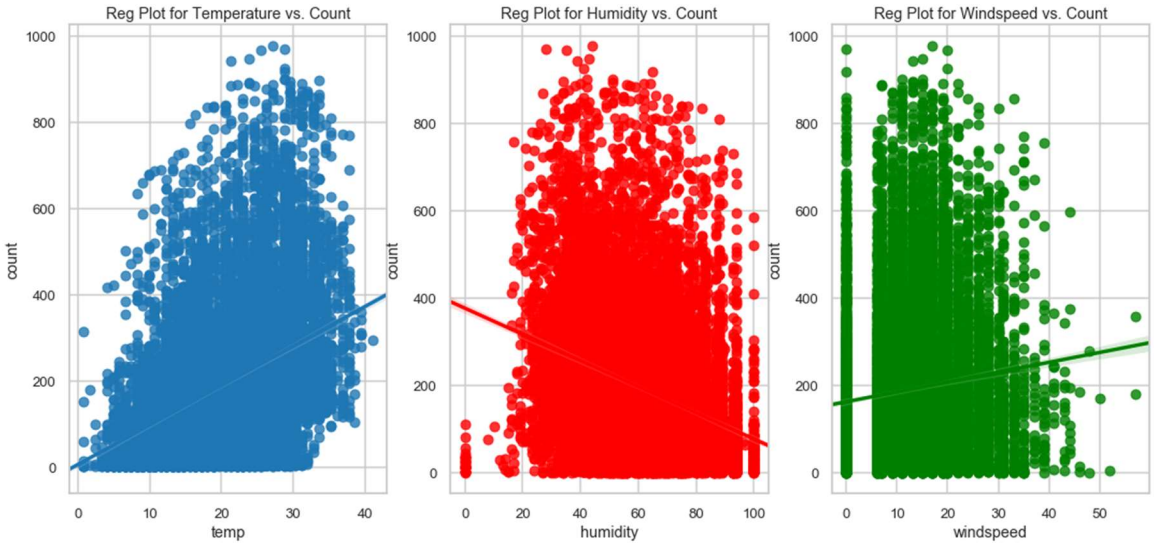


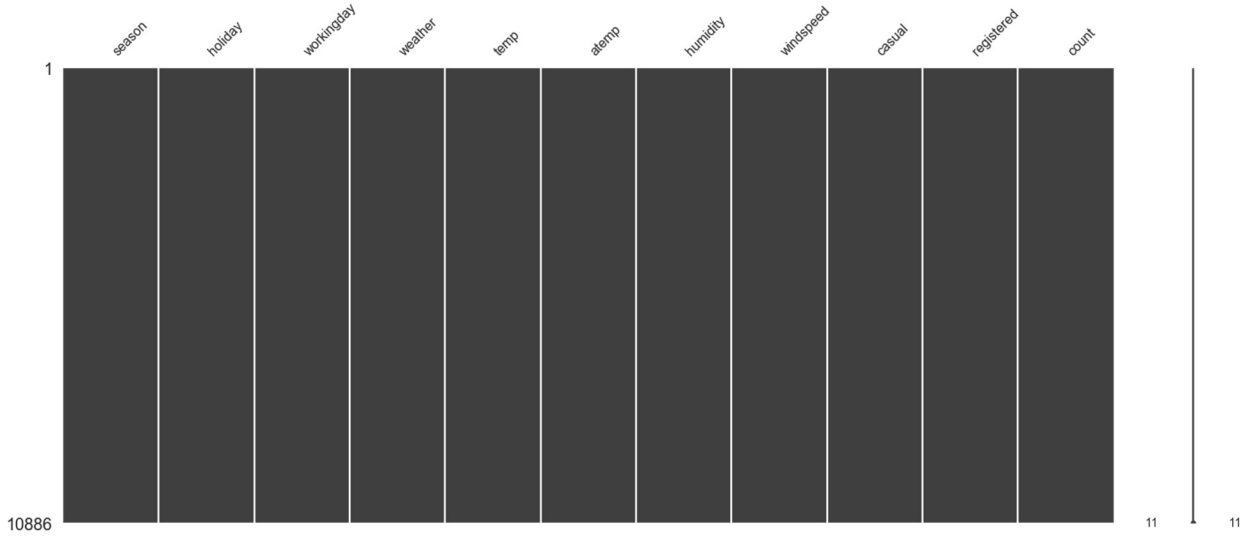
Figure 4-2: Regression Plots of 'Count' vs. a) Temperature, b) Humidity and, c) Windspeed

The above regression plots indicate a strong positive correlation of *count* with *temperature*, a weak positive correlation with *windspeed*, and a strong negative correlation with *humidity* (as we saw in the heatmap plots too). We can also see several instances with *windspeed* = 0. These are probably missing values. Since *windspeed* has a very low dependence and has several missing (or erroneous) data points, let us exclude this from our model

5 DATA CLEANING

5.1 Missing Data Fields

As seen from the below figure, the provided data set has no missing data fields for any of the features.



5.2 Handling Outliers

5.2.1 Weather = 'Heavy Snow/Rain' outlier

The provided data set has just one instance of 'Heavy Snow/Rain' observation and wouldn't be very helpful in training any of our model. Hence, we replace that instance with 'Light Snow/Rain'

5.2.2 Zscore outliers

We prune out all observations with zscore > 4, i.e., data with more than 4 standard deviation away from the mean. These correspond to observations which have probability of ~$6x10^{-5}$. There were 15 such observations and all these outliers occur mostly early in the morning or late at night. These could be due to some late-night shows or events. We prune these outliers out and use the remaining 10871 observations to train our model

6 FEATURE ENGINEERING

The provided data in its raw form wasn't directly used as an input to the model. Several feature engineering was carried out where few features were modified, few were dropped, and few were added. Below is a summary of the feature engineering carried out with the provided data set

1. The *datetime* column which contained the date-time stamp in 'yyyy-mm-dd hh:mm:ss' format was split into individual *['month', 'date', 'day', 'hour']* categorical columns.
2. Drop *season* column: This is because *month* column has a direct mapping with *season* (Winter: January to March, Summer: April to June, Fall: July to September and Spring: October to December). Hence, we retain *month* column due to its higher cardinality.
3. Drop *holiday* and *day* columns: The workingday column had information about holiday embedded in it. *workingday* = weekday and not a holiday. Since we noticed that there were two kinds of bike rental behaviors - during working days and not a working day, we will retain only the *workingday* column and drop *'day'* and *'holiday'* column.
4. Drop *date* column: Intuitively, there is should be no dependency on date. Hence drop this column.
5. Drop *casual* and *registered* columns: These are individual components of the 'to be predicted' column (*count*). These are not provided in the Kaggle Test Data too. Hence drop these columns.
6. Drop *windspeed* column: Very poorly correlated with *count* and has several missing/erroneous data. Hence drop this column.
7. Drop *atemp* column: *temp* and *atemp* are very highly correlated and essentially indicate the same thing. Hence retain only the *temp* column
8. OneHotEncoding of categorical feature set
    a. *Weather*: Split weather column to *weather_1*, *weather_2* and *weather 3* (recall that we had relabeled all the weather = 4 data points to weather = 3 due to its sparseness). Drop *weather_3* and *weather* columns since it is a function of the rest of the retained weather columns
    b. *Month*: Split month column to *month_1, month_2, ..., month_12*. Drop *month_12* and *month* columns since they are a function of the rest of the retained month columns
    c. *Hour*: Split hour column to *hour_0, hour_1, ..., hour_23*. Drop *hour_23* and *hour* columns since they are a function of the rest of the retained hour columns

7 MODELLING

7.1 Modeling Overview

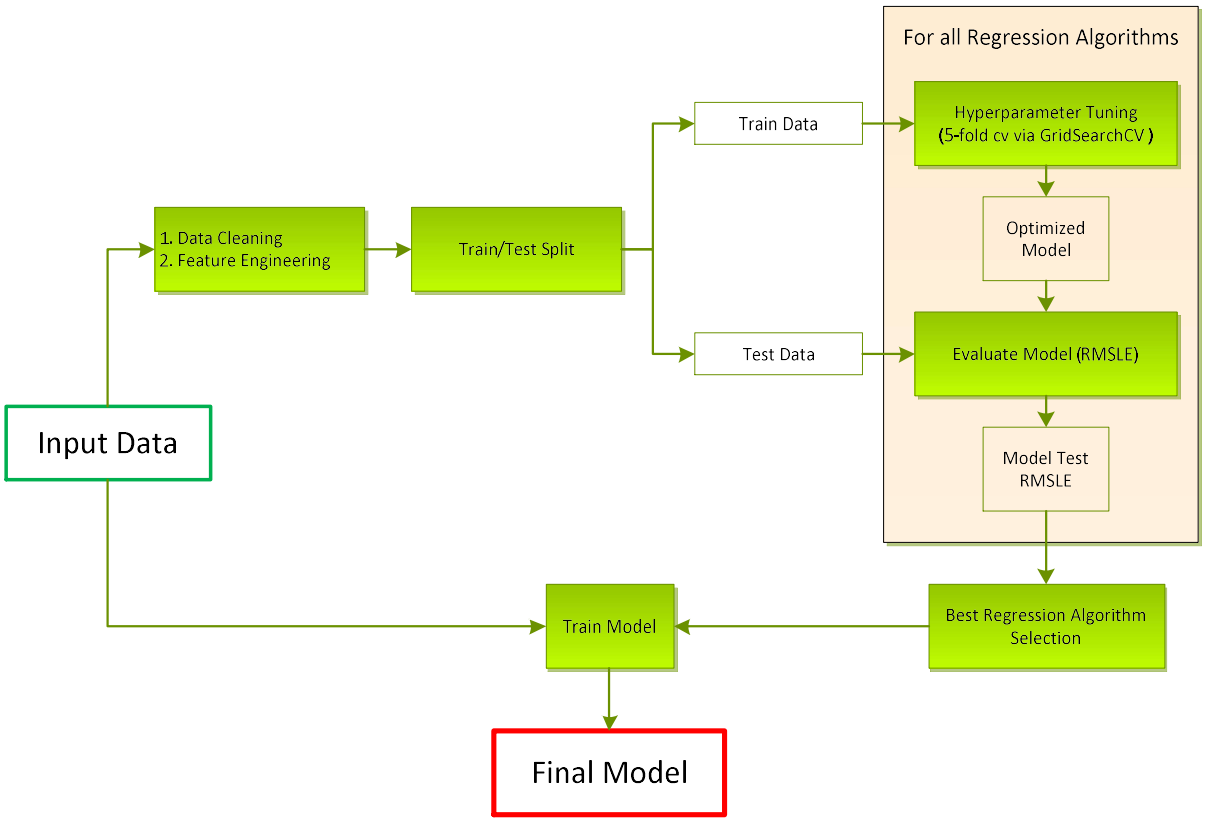


Figure 7-1: Overview of the Modelling procedure

Figure 7-1 depicts the overall procedure followed to obtain the Final Model. The provided data is first cleaned and transformed using Feature Engineering. We then split the data into Train set (for Hyperparameter tuning) and Test set (for Model Evaluation). Using RMSLE as our evaluation metric, we compare various models and select the regression algorithm based on the lowest RMSLE on the Test data. The final model used for submission is then obtained by again training the selected Regression Algorithm on the entire Input Data set.

7.2 Train/Test Split

Below figure summarizes the method to split the provided data into training and testing data set. Kaggle has held out data from 20th to the end of the month (for every month) as test set (no labels, i.e., count values

have been provided for those data). We follow a similar approach to split the provided labelled data set (consisting of 1st to 19th of every month) into two sets

1. Training set
    a. This contains data of from the 1st to 15th of every month
    b. This set is used to train various model and obtain the best set of hyperparameters for these models. We use GridSearchCV to tune the hyperparameters using this training set
2. Test set
    a. This contains data of from the 16th to 19th of every month
    b. This is used to evaluate all our models. The model with the best test score is finally chosen for submission

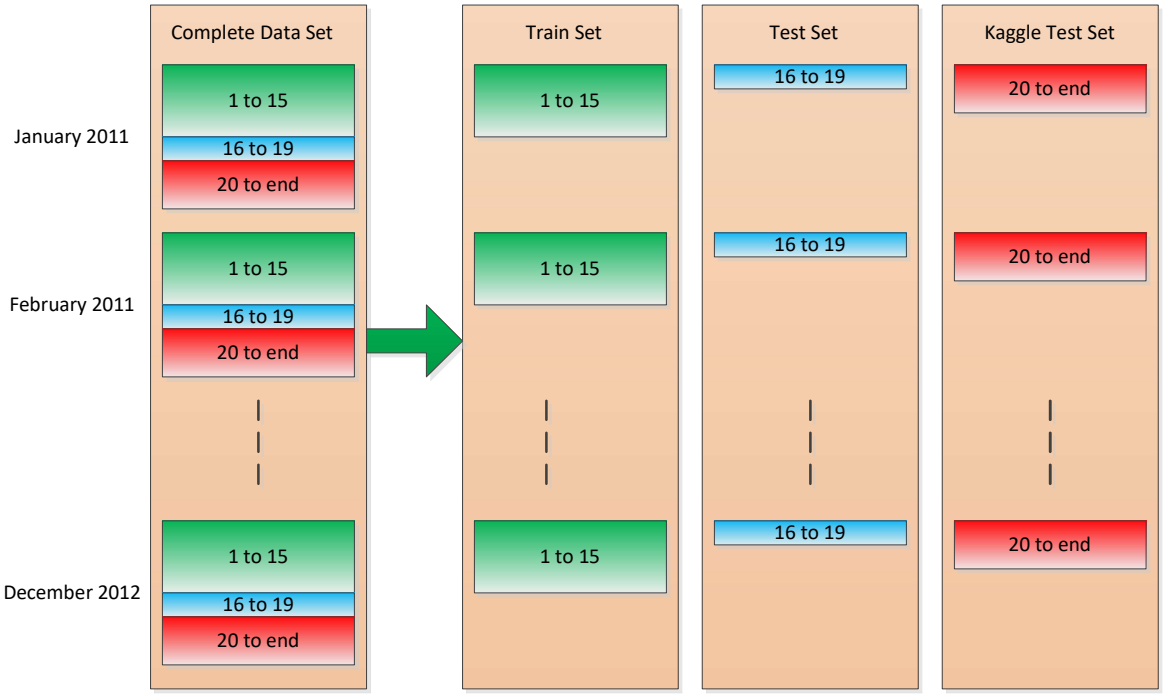


Figure 7-2: Splitting the provided data into training and testing set

7.3 Regression Algorithms Summary

Below is list of models tried out and a brief description of the algorithm.

Category	Modeling Algorithm	Details
Linear	Linear Regression	Two separate models for Working and Non-Working days. OneHotEncoded Features

Linear	Ridge Regression	Two separate models for Working and Non-Working days. OneHotEncoded Features
Linear	Lasso Regression	Two separate models for Working and Non-Working days. OneHotEncoded Features
Ensemble	Random Forest-1	Single Model for Working and Non-Working days. No OneHotEncoding
Ensemble	Random Forest-2	Two separate models for Working and Non-Working days. OneHotEncoded Features
Ensemble	Random Forest-3	Two separate models for Working and Non-Working days No OneHotEncoding
Ensemble	Gradient Boost-1	Two separate models for Working and Non-Working days. OneHotEncoded Features
Ensemble	Gradient Boost-2	Two separate models for Working and Non-Working days No OneHotEncoding
Ensemble	Ada Boost	Two separate models for Working and Non-Working days No OneHotEncoding
Stacking	Linear Regression	Use all the Linear+Ensemble model prediction as the Feature set to make final predictions
Stacking	Random Forest	Use all the Linear+Ensemble model prediction as the Feature set to make final predictions
Stacking	Gradient Boost	Use all the Linear+Ensemble model prediction as the Feature set to make final predictions

7.4 Stacking Model Details

Stacking is an ensemble learning technique that uses predictions from multiple models (for example Linear Regression, Random Forest, Gradient Boost) to build a new model. This model is used for making predictions on the test set. Below is a step-wise explanation for a simple stacked ensemble:

1. The train set is split into 5 parts – data from a) (1^{st} to 3^{rd}) of every month, b) (4^{th} to 6^{th}) of every month, c) (7^{th} to 9^{th}) of every month, d) (10^{th} to 12^{th}) of every month and e) (13^{th} to 15^{th}) of every month
2. A base model (E.g. Linear Regression) is fitted on 4 parts and predictions are made for the 5^{th} part. In Figure 7-3, LR-Model 1 corresponds to Linear Regression Model obtained by training all except the 1^{st} part. This is done for each part of the train set (LR-Model 1 to LR-Model 5)
3. The base model (Linear Regression Model) is then fitted on the whole train data set (LR-Model)

4. Using this model (LR-Model), predictions are made on the Test Set (containing 16th to 19th of every month)
5. Steps 2 to 4 are repeated for another base model (say Random Forest) resulting in another set of predictions for the train set and test set.

   Figure 7-3 depicts Steps 1 through 5.
6. All the predictions from train and test set are compiled into a new table as shown in Figure 7-4. The predictions from the train set are used as features to build a new model
7. This model is used to make final predictions on the test prediction set

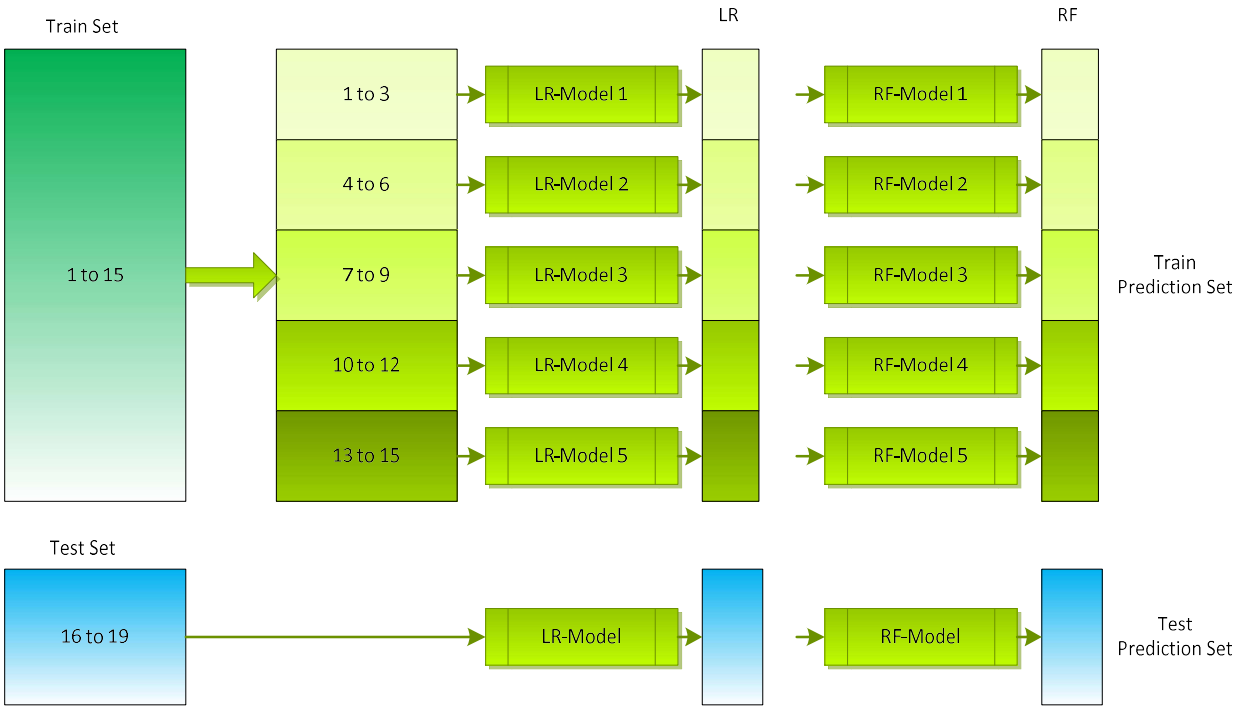


Figure 7-3: Step 1 to Step 5 in the Stacking algorithm

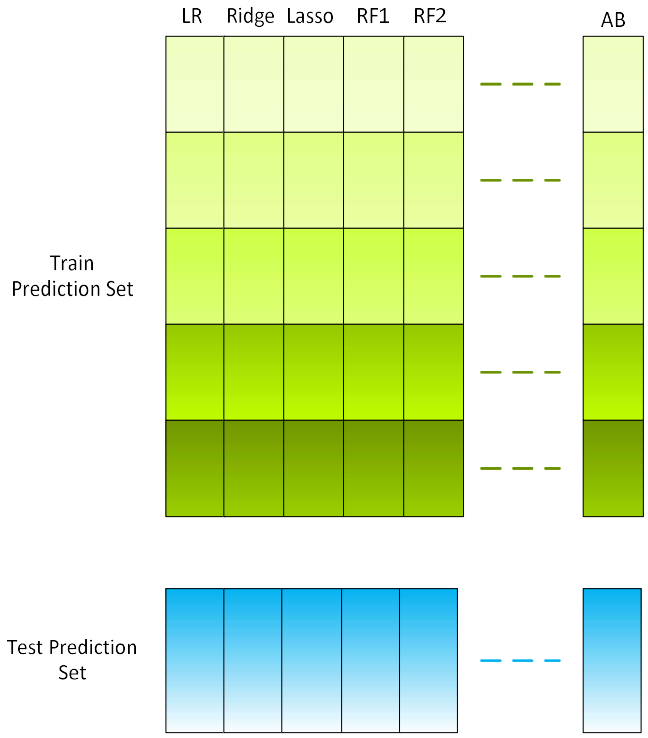


Figure 7-4: Predictions from Individual Base models used as features to build a new model

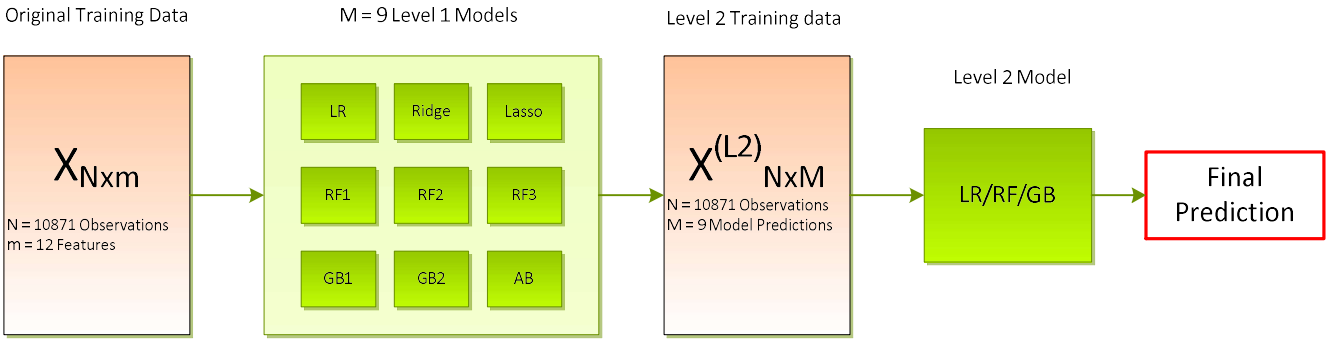


Figure 7-5: Modelling via Stacking – Overall Summary

The overall summary of the Model that we developed via Stacking is shown in Figure 7-5. Predictions from 9 Individual Models are used as features for the Level 2 Model (for which Linear Regression, Random Forest and Gradient Boost were tried out) to obtain the final prediction.

7.5 Evaluation Metric – RMSLE

Root Mean Square Log Error (RMSLE) is chosen as the evaluation metric by Kaggle and is used in our modelling too. The RMSLE is calculated as

$$\sqrt{\frac{1}{n}\sum_{i}^{n}(\log\ (p_i+1)-\log(a_i+1))^2}$$

Where

- n is the number of hours in the test set
- p_i is the predicted count
- a_i is the actual count
- $\log(x)$ is the natural logarithm

As a result, each of our models are trained using $\log(\text{count})$ as the target columns (instead of count). This would result in the predicted values being returned in the log domain. Thus, we further transform the predicted output via $\exp(y_{pred})-1$.

For each model, we will evaluate the following metrics for 3 portions of data: {Working Days, Non-Working Days, All combined}

RMSLE Metric	Model Training Data Set	Model Testing Data Set
Train	Train Set	Train Set
CV Test	(4/5) th of the Train Set	The remaining (1/5) th of the Train Set
Test	Train Set	Test Set

CV Test RMSLE is obtained as a by-product of the above Stacking Model building. It is essentially the RMSLE computed on the Train prediction and Test prediction set for each of the models.

7.6 Linear Regression

For Linear Regression, we obtain two separate models – one for working and the other for non-working days. We also use the entire set of features obtained via OneHotEncoding.

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.418155	0.432817	**0.422797**
CV Test	0.430658	0.474808	**0.444944**
Test	0.390881	0.495181	**0.428666**
Total Train + Test Time			**0.197 sec**

Table 7-1: RMSLE Summary for Linear Regression

RMSLE on the test data = 0.43 and training data = 0.42 are almost the same. Linear Regression model is definitely not an overfit model.

Below plots the true *count* value vs. the model predicted value for a few days in the test data set. We can see that the predicted *count* closely tracks the true value.

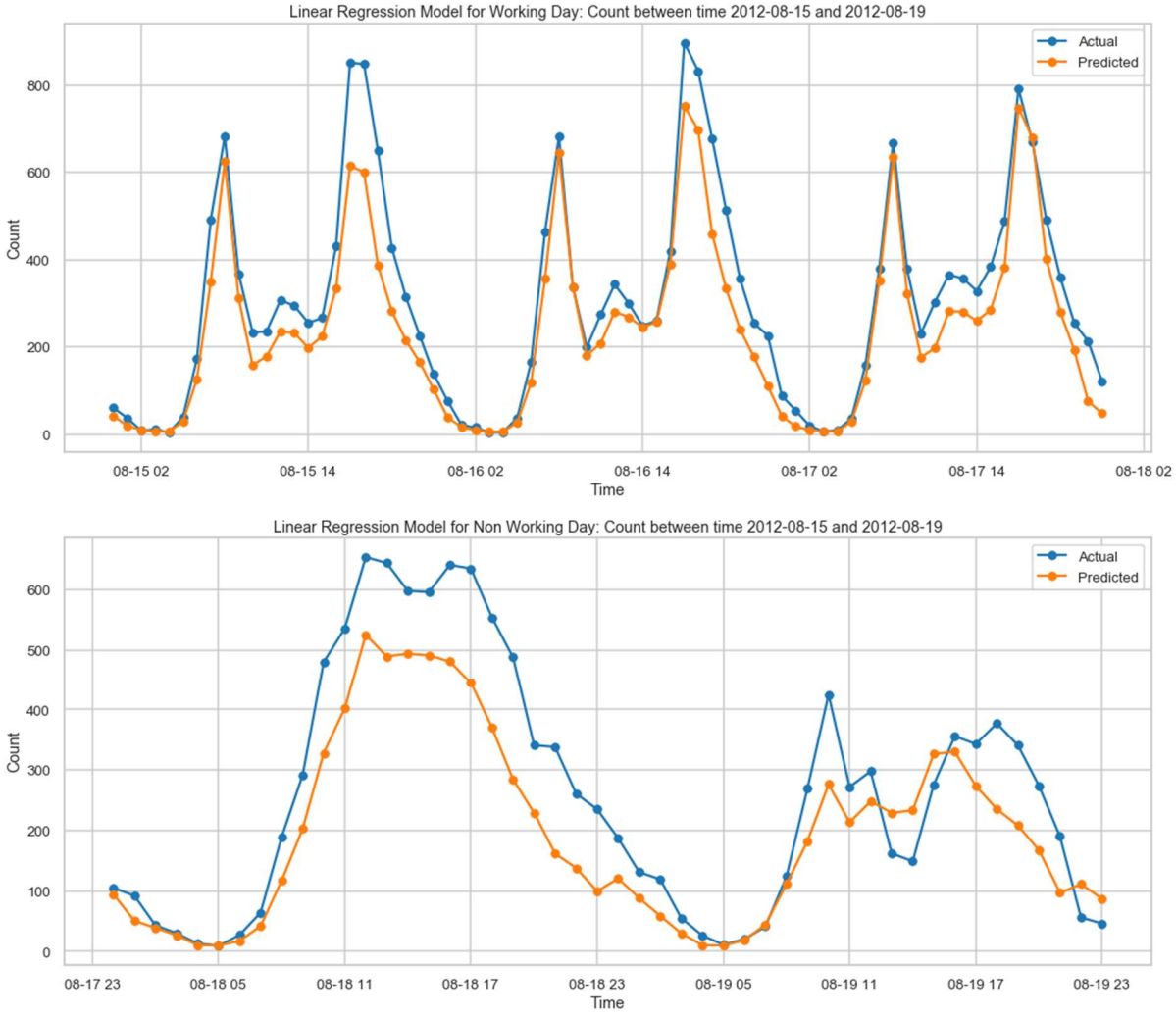


Figure 7-6: Linear Regression: Actual vs. Predicted Bike rental count for between 2012-8-15 and 2012-8-19

7.7 Ridge Regression

For Ridge Regression too, we obtain two separate models – one for working and the other for non-working days. We also use the entire set of features obtained via OneHotEncoding.

We tune the hyperparameter using the Train Set using GridSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters for Ridge Regression

Hyperparameters	Working Day Model	Non-Working Day Model

α	10	10

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.423189	0.448141	**0.431152**
CV Test	0.438216	0.492361	**0.45585**
Test	0.39423	0.516847	**0.439148**
Total Train + Test Time			**1.3 sec**

Table 7-2: RMSLE Summary for Ridge Regression

Ridge Regression Model has a higher RMSLE compared to the Linear Regression Model.

Below plot the actual '*count'* value vs. the model predicted value for a few days in the test data set

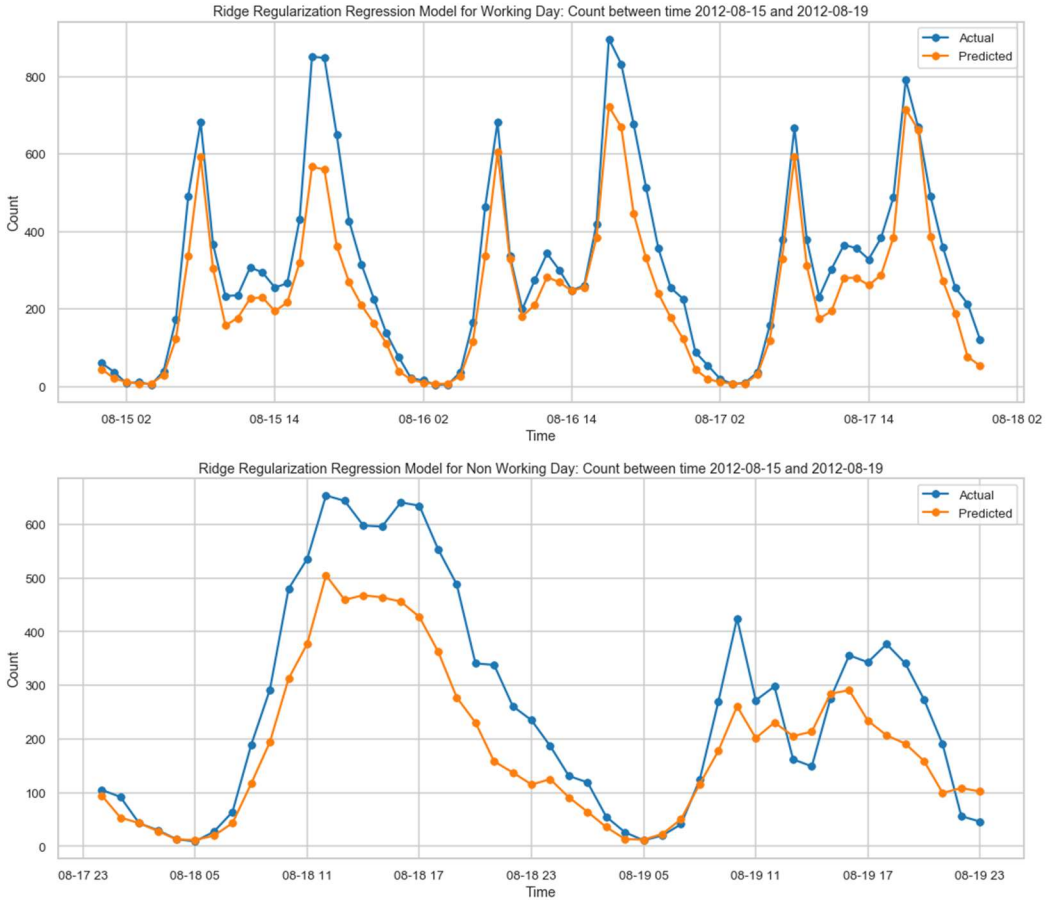


Figure 7-7: Ridge Regression: Actual vs. Predicted Bike rental count for between 2012-8-15 and 2012-8-19

7.8 Lasso Regression

As with the previous two Linear models, we obtain two separate models – one for working and the other for non-working days and use the entire set of features obtained via OneHotEncoding for Lasso Regression Model too

We tune the hyperparameter using the Train Set using GridSearchCV with 5-fold cross validation. Below table summarizes the optimal hyperparameters for Lasso Regression

Hyperparameters	Working Day Model	Non-Working Day Model
α	0.1	0.5

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	1.313353	1.065252	**1.241062**
CV Test	1.31459	1.075364	**1.244685**
Test	1.285346	1.117959	**1.231794**
Total Train + Test Time			**1.3 sec**

Table 7-3: RMSLE Summary for Lasso Regression

Clearly, the Lasso Regression Model performs very bad. Lasso Model has a RMSLE of 1.23 for the test data compared to 0.43 for Linear Regression

Below plot the actual 'count' value vs. the model predicted value for a few days in the test data set. We can see that the predicted value is way off the true value.

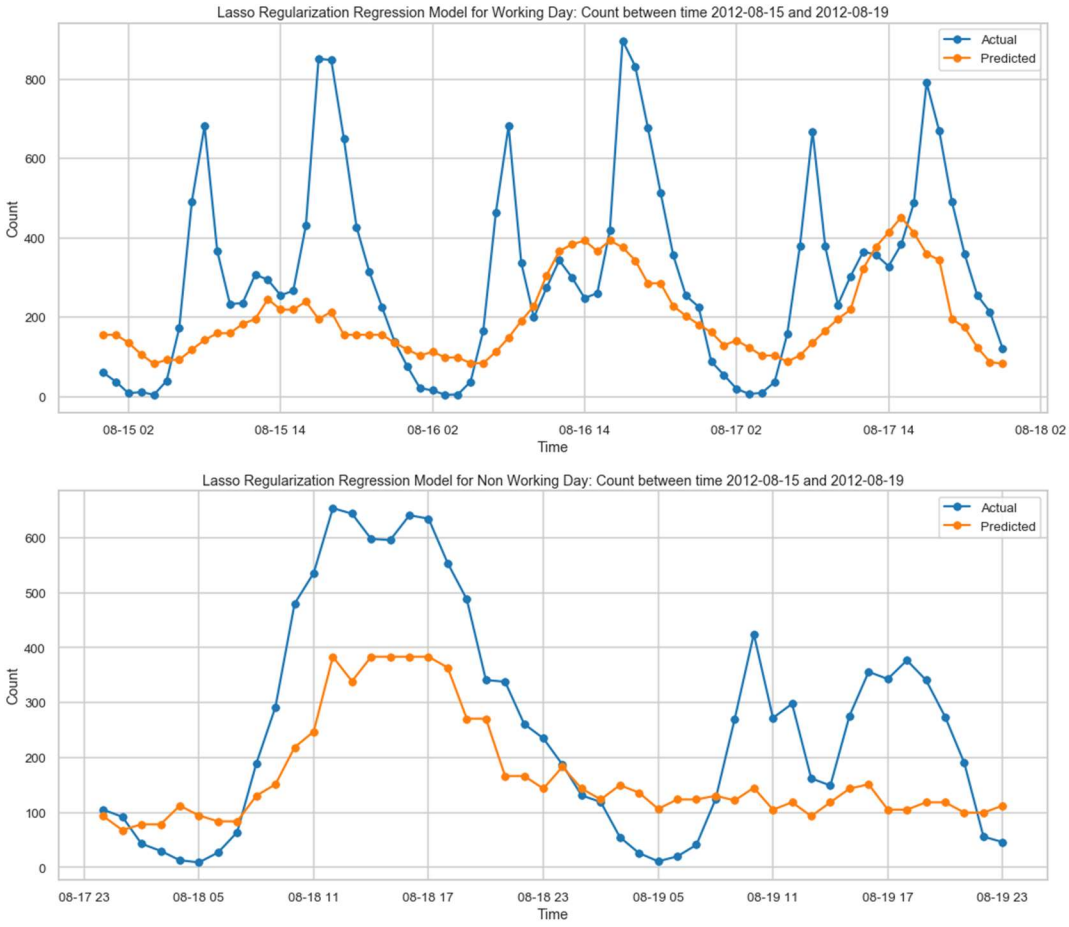


Figure 7-8: Lasso Regression: Actual vs. Predicted Bike rental count for between 2012-8-15 and 2012-8-19

Feature Importance – Linear Models

Feature importance for the Linear Models are obtained from the Coefficients assigned to each of the feature. Higher the magnitude, more is the importance. Figure 7-9 compares the feature coefficient for each of the 3 Linear models.

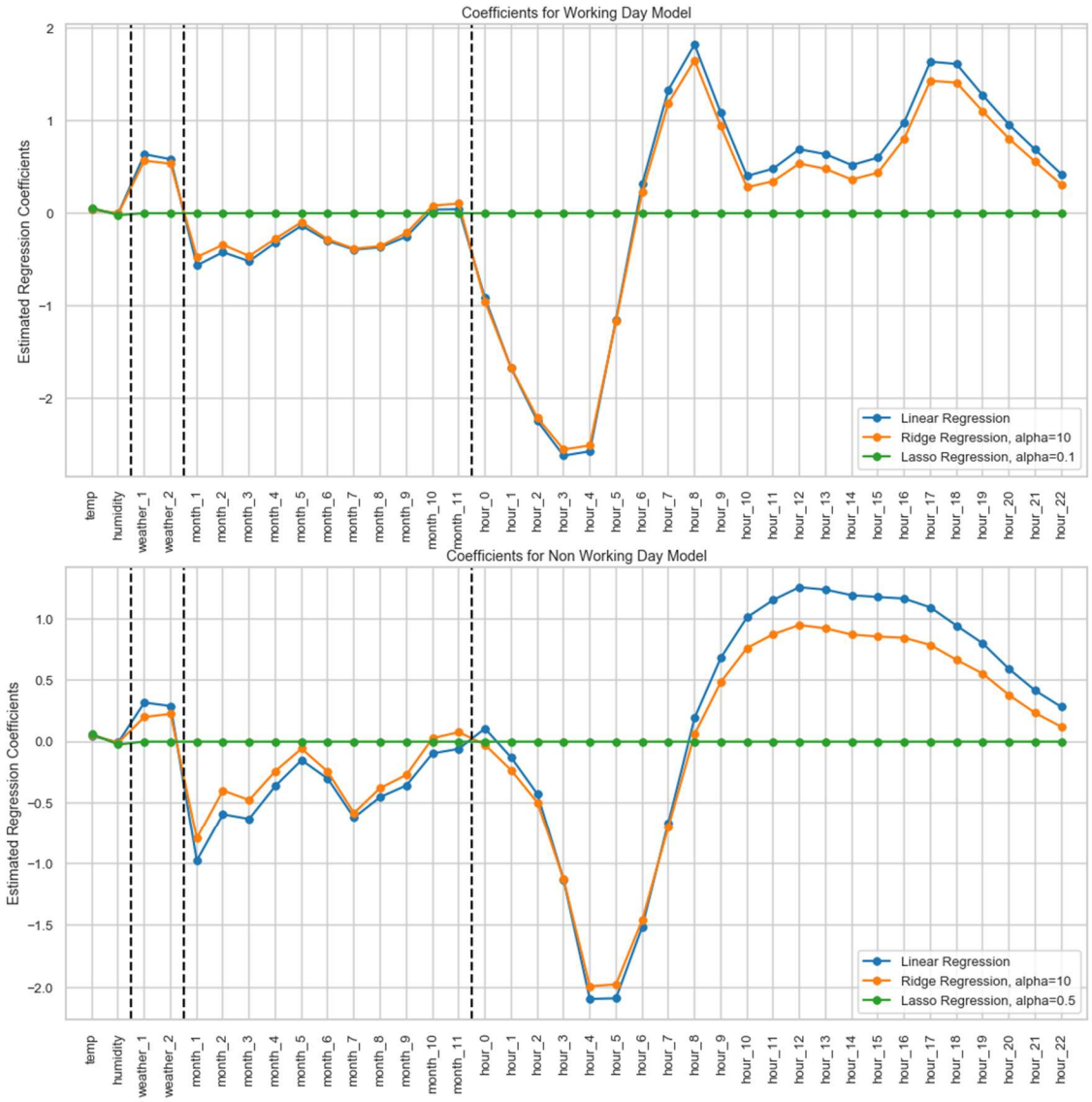


Figure 7-9: Feature Coefficient Comparison for the Linear Models (Linear Regression, Ridge and Lasso)

Below are few observations from the

- Linear Regression coefficients are most aggressive (highest magnitude). Ridge Regression coefficients closely follows Linear Regression. And Lasso nullifies most of the coefficients.
- Lasso model provides a non-zero value only for temp and humidity for working day model
- From the Linear and Ridge coefficient plots, we can see that maximum dependency on the bike rental count are the hour in the day (highest coefficient magnitudes). Negative values of coefficient for early morning hours and positive values with greater magnitude during peak hours of the respective models.
- Working day coefficient has highest positive coefficient values at 8am and 5pm

- Non-working day coefficients has a single peak across hours ~ 12 noon (as observed from our earlier plots)
- Weather_1 and weather_2 have positive coefficient value, indicating a negative bias for weather_3 (light snow/rain). Note that if weather=3, then weather_1 and weather_2 = 0. Hence effective weather_3 coefficient = 0
- Though the coefficient value for temp and humidity is low compared to the others, the range of these values are higher too.
- The absolute value of the coefficients is higher for months 5, 6, 10, 11, 12 indicating higher bike rentals during that month.

7.9 Random Forest (RF1)

Since Random Forest Regression predicts based on decision tree, we expect it to handle categorical features including hours, months, working and non-working day gracefully. In RF1, we use a single model for both Working and Non-Working days and do not use OneHotEncoding to transform the categorical data.

We tune the hyperparameter using the Train Set using GridSearchCV with 5-fold cross validation. Below is the procedure adopted to tune hyperparameter for Random Forest Model

1. First obtain n_estimators using default values of the remaining parameters
2. Tune for the max_features using the best n_estimators
3. Tune for min_samples_leaf using the best n_estimators and max_features
4. Tune for max_depth using the best n_estimators, max_features and min_samples_leaf
5. Tune for min_samples_split using the best n_estimators, max_features, min_samples_leaf and max_depth

Below figures plot the variation of the cv_score (which reflects CV Test score)

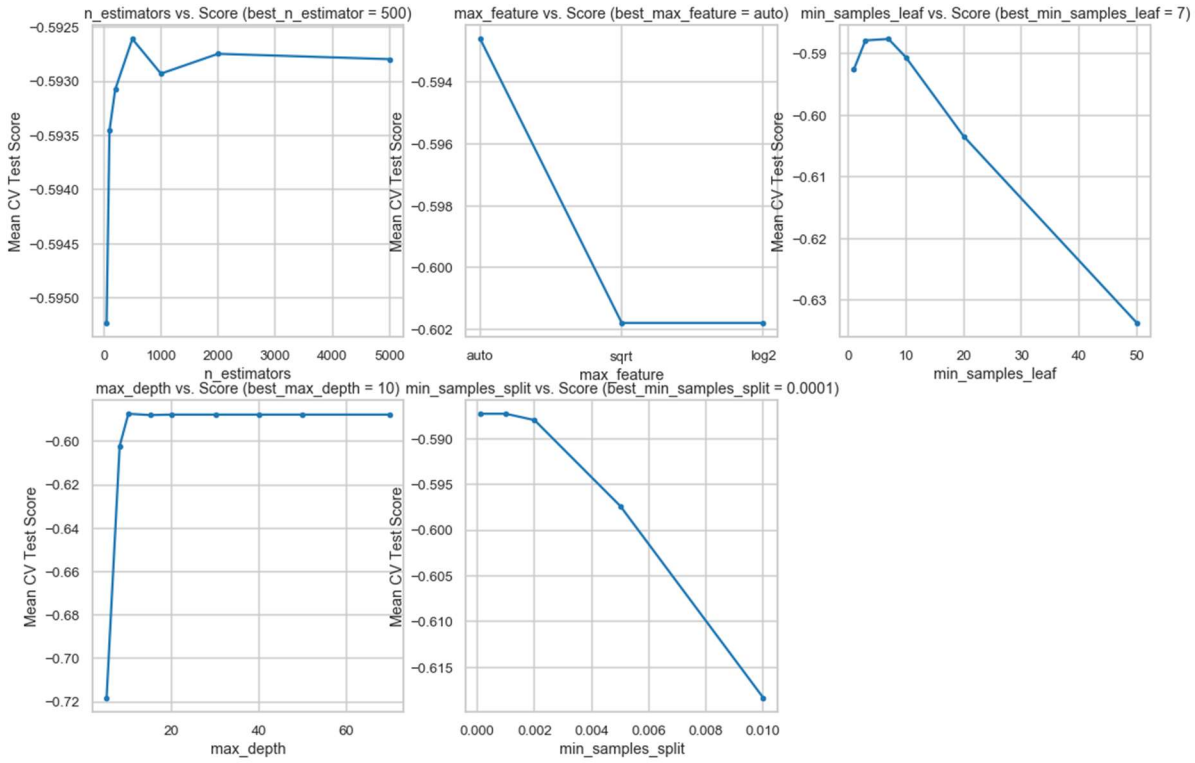


Figure 7-10: Hyperparameter tuning (one parameter at a time) for Random Forest (n_estimators, max_features, min_samples_leaf, max_depth and min_samples_split)

The below table summarizes the optimal hyperparameters for RF1 model

Hyperparameters	Optimal Value
n_estimators	500
max_features	'auto'
min_samples_leaf	7
max_depth	10
min_samples_split	1

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.353823	0.387612	**0.364732**
CV Test	0.425339	0.482957	**0.444173**
Test	0.393007	0.48925	**0.427676**

Total Train + Test Time	5.48 sec

Table 7-4: RMSLE Summary for Random Forest Regression (RF1)

RF1 model performs on par with Linear Regression Model. Also, the train and test RMSLE are similar, which indicates that it is not overfit.

Below plot the actual 'count' value vs. the model predicted value for a few days in the test data set

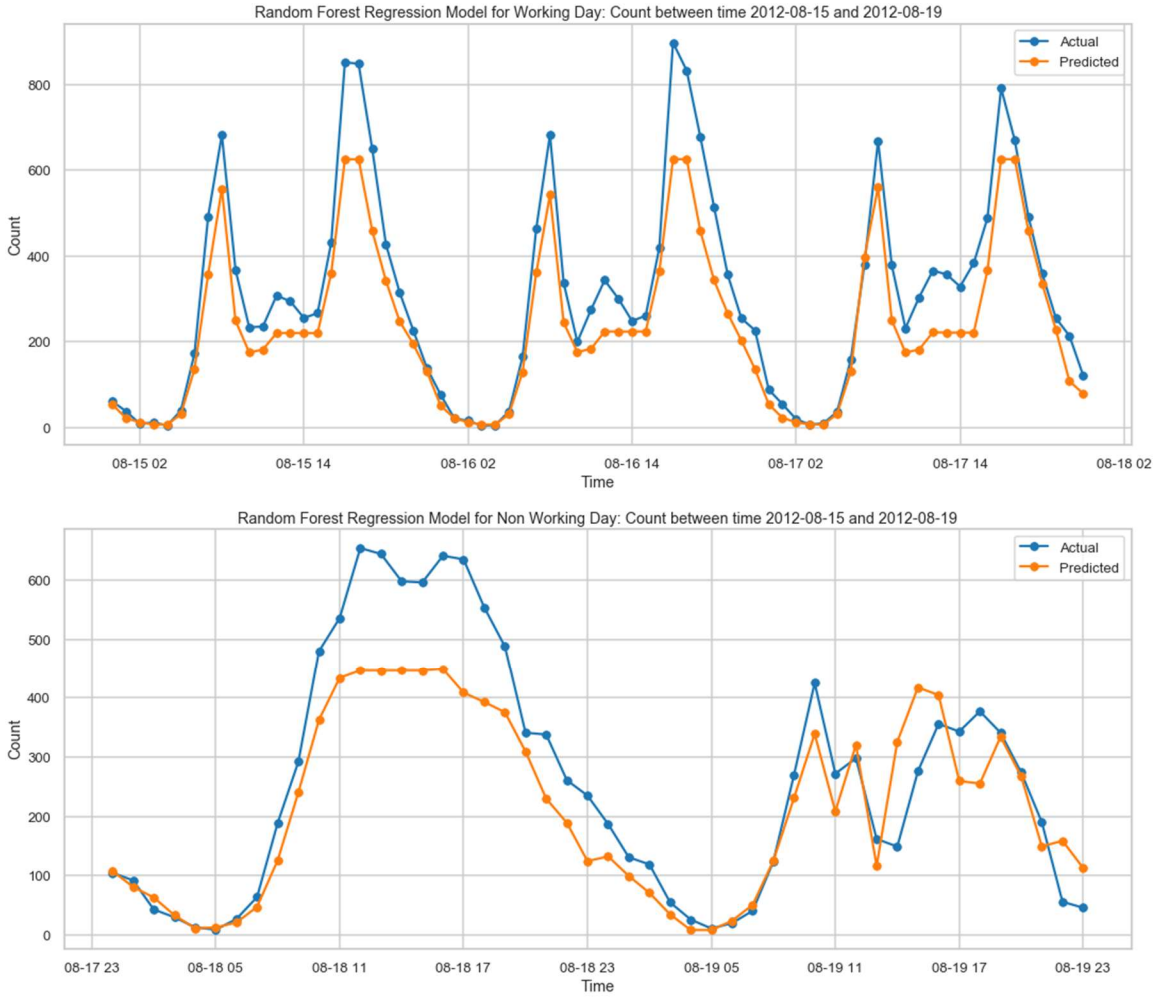


Figure 7-11: Random Forest (RF1): Actual vs. Predicted Bike rental count for between 2012-8-15 and 2012-8-19

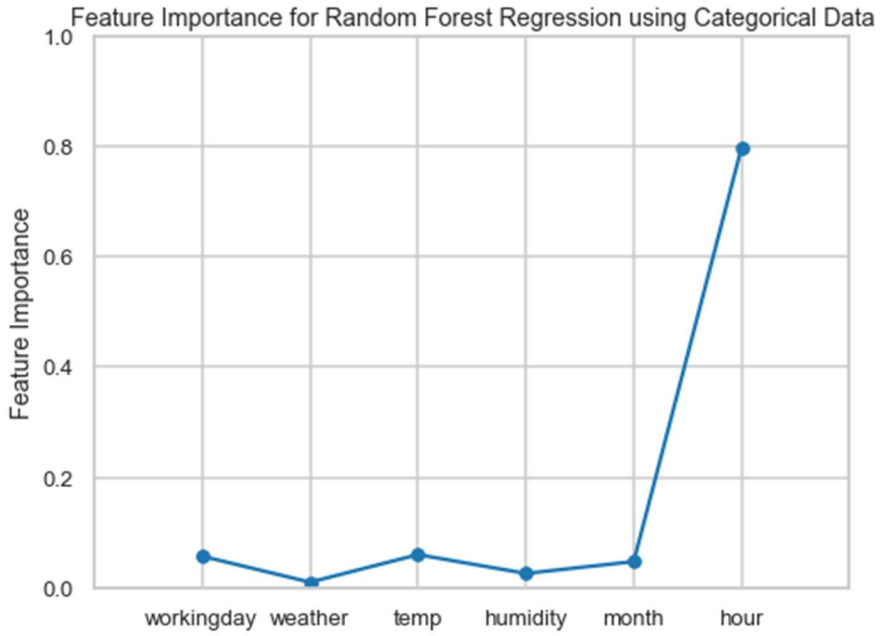


Figure 7-12: Feature Importance for Random Forest Model, RF1

As expected 'hour' feature holds maximum importance. We saw spikes and dips in count value depending on the hour of the day. We also notice that *workingday* feature has marginal importance. In our next model, we will use two separate models for Working and Non-Working days.

7.10 Random Forest (RF2)

In this Random Forest model (RF2), we use all the features transformed via OneHotEncoding and obtain two separate models for Working and Non-Working days.

Hyperparameter tuning is done as explained in RF1. Figure 7-13 shows the cv_score obtained during hyperparameter tuning via GridSearchCV.

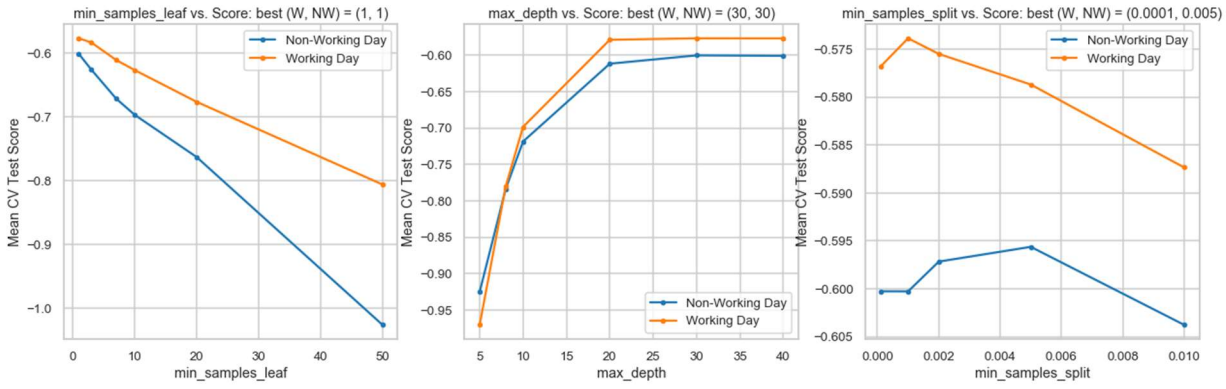


Figure 7-13: Hyperparameter tuning for Random Forest (min_samples_leaf, max_depth and min_samples_split)

The below table summarizes the optimal hyperparameters for the RF2 model

Hyperparameters	Working Day Model	Non-Working Day Model
n_estimators	2000	200
max_features	'sqrt'	'log2'
min_samples_leaf	1	1
max_depth	30	30
min_samples_split	0.0001	0.005

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.181652	0.378901	**0.259992**
CV Test	0.437254	0.53261	**0.46918**
Test	0.40938	0.541029	**0.457731**
Total Train + Test Time			**22.4 sec**

Table 7-5: RMSLE Summary for Random Forest Regression (RF2)

From the table, we see a dip in prediction relative to our earlier Random Forest model (which used Categorical Features and single model for working and non-working days). Average Test RMSLE increases from 0.43 to 0.46. Also note that the Train RMSLE is reasonably lower than Test RMSLE. This possibly is a result of on over-fit model

Below is a feature importance plot for RF2 model.

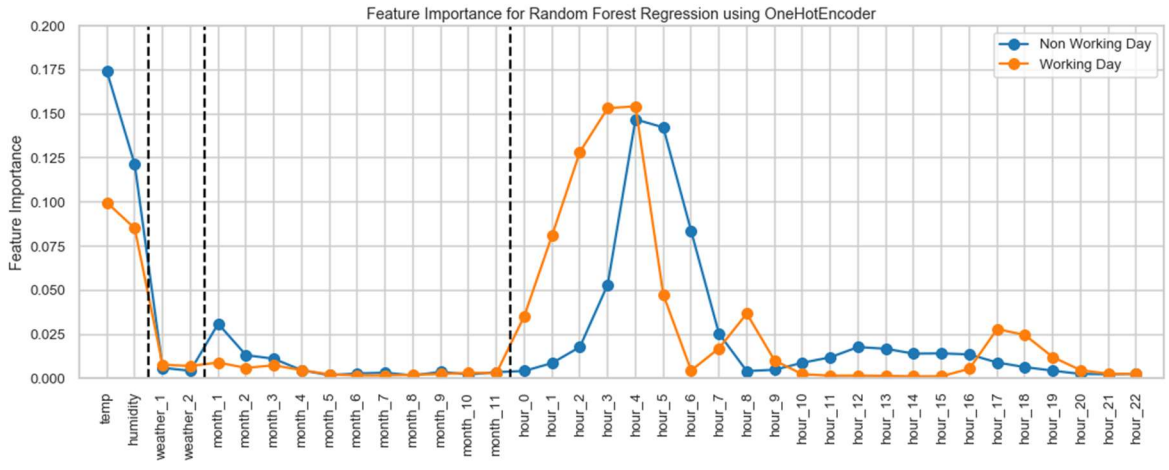


Figure 7-14: Feature Importance plot for Random Forest Model, RF2

From the feature importance plot Figure 7-14 we can see that the features for which the average *count* is significantly different from the generic mean *count* (late night or peak hours), tends to hold higher weightage. For example

- Non-working day hour_x feature set: *hour_3* to *hour_6* have high importance as the average count at these hours are very low compared to the daily average. Not splitting based on these features correctly would result in higher MSE. The peak hours for non-working days are at 1pm and 2pm; correspondingly, we do see relatively higher importance for *hour_12* and *hour_13*
- Working day hour_x feature set: *hour_1* to *hour_4* have high importance for similar reasons explained in the previous bullet. *hour_8, hour_17*, *hour_18* too hold reasonably high importance since they are the peak hours and tend to have higher than usual count values

7.11 Random Forest (RF3)

In this model, we will not use OneHotEncoding to transform the categorical features but have two separate models to make independent predictions for Working and Non-Working days.

Below are hyperparameter tuning plots obtained via GridSearchCV with cv=5

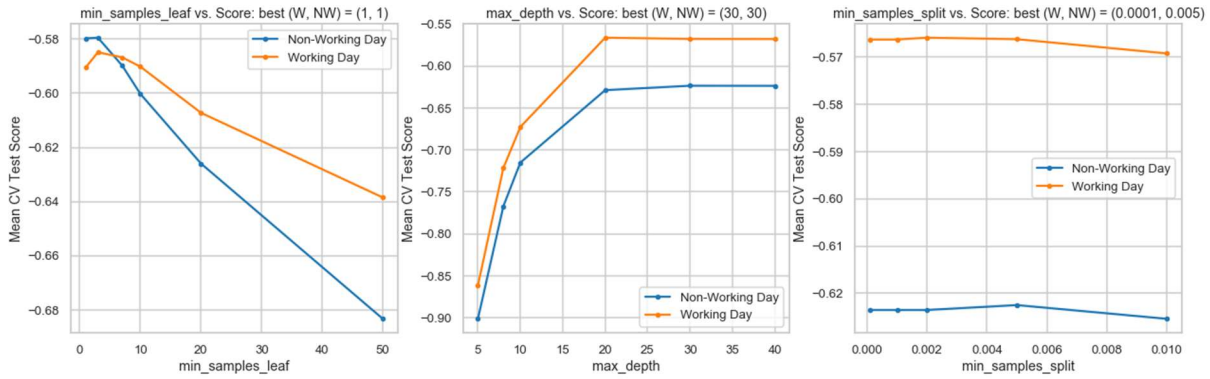


Figure 7-15: Hyperparameter tuning for Random Forest (min_samples_leaf, max_depth and min_samples_split)

The below table summarizes the optimal hyperparameters for RF3 model

Hyperparameters	Working Day Model	Non-Working Day Model
n_estimators	500	500
max_features	'auto'	'sqrt'
min_samples_leaf	3	3
max_depth	20	30
min_samples_split	0.002	0.005

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.290989	0.346424	0.309405
CV Test	0.418383	0.496303	0.444237
Test	0.391217	0.524161	0.44026
Total Train + Test Time			6.54 sec

Table 7-6: RMSLE Summary for Random Forest Regression (RF3)

Average Test RMSLE = 0.44 which is still a bit higher than RF1 Random Forest model. Additionally, the average training RMSLE = 0.31 indicates that this model too might be a bit overfit

Below is the feature importance plot for RF3 model. As RF1, *temp* and *hour* feature hold maximum importance in the regression

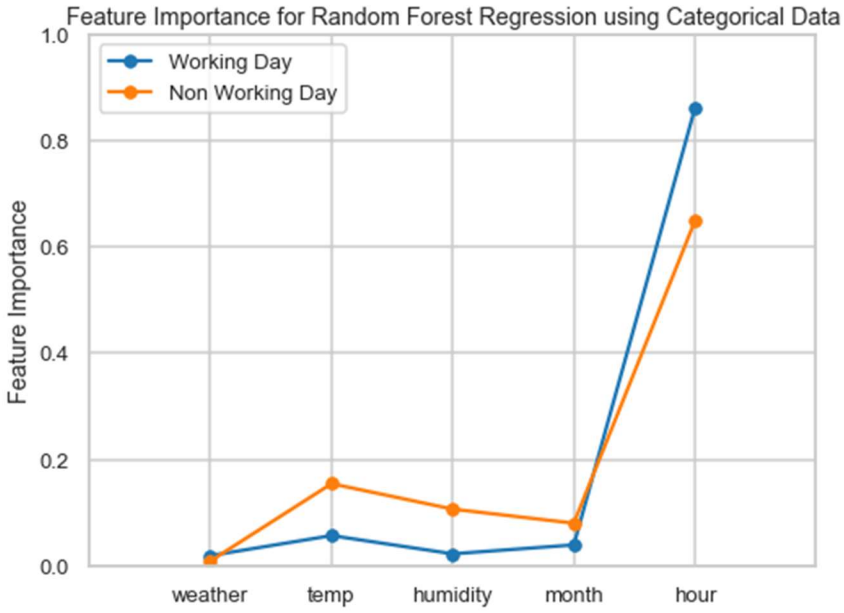


Figure 7-16: Feature Importance plot for Random Forest Model, RF3

Random Forest uses an ensemble of 500 decision trees in our earlier model. Let us visualize one of them and see if it makes sense. The entire tree would have several leaf nodes and would be difficult to analyze. Instead, let us limit the max_depth to 3 and plot one decision tree (as shown in Figure 7-17)

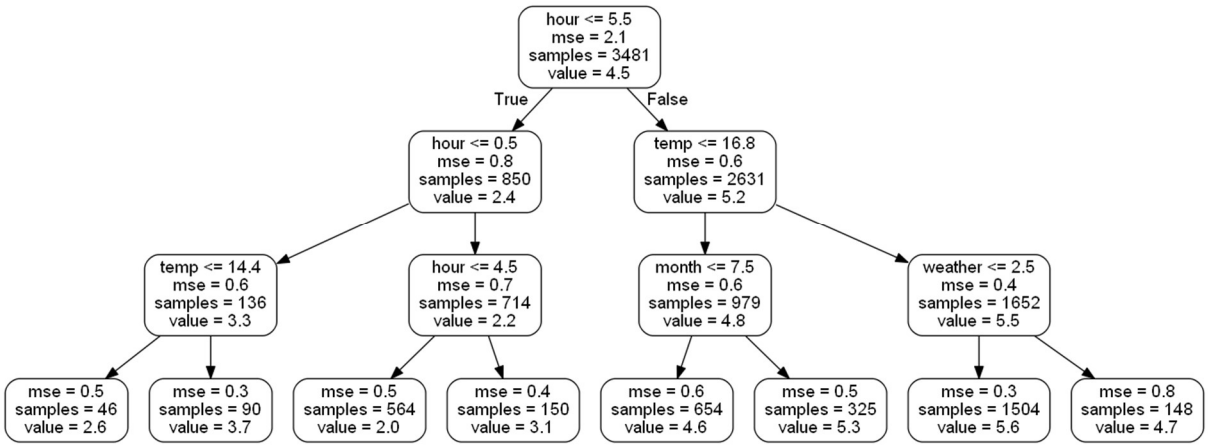


Figure 7-17: One of the Decision Trees used in our Random Forest Models

The first split is made based on hour feature indicating that it is very important (since that first split at 5.5 hours minimizes the MSE). Note that the value indicated at the leaf node corresponds to the predicted value (which is log(1+count)).

7.12 Gradient Boost (GB1)

Gradient Boosting is another powerful regression technique which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. Here, we use a two Gradient Boost models to predict the bike rental count for Working and Non-Working days. We will also use the feature set obtained via OneHotEncoder.

We adopt the below procedure to tune hyperparameter for the Gradient Boost Model

1. Pick a large n_estimator = 3000
2. Tune max_depth, learning_rate, min_samples_leaf, and max_features via grid search.
3. Increase n_estimators even more (5000) and tune learning_rate again holding the other parameters fixed.

The below table summarizes the optimal hyperparameters for GB1 model

Hyperparameters	Working Day Model	Non-Working Day Model
n_estimators	3000	3000
learning_rate	0.01	0.005
max_features	1	1
min_samples_leaf	5	5
max_depth	6	6

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.272817	0.299252	**0.281356**
CV Test	0.425073	0.559165	**0.471153**
Test	0.407705	0.550037	**0.460327**
Total Train + Test Time			**42.9 sec**

Table 7-7: RMSLE Summary for Gradient Boost (GB1)

GB1 has a higher Test RMSLE compared to RF1 model. Also, there is a significant difference between the Train (0.3) and Test RMSLE (0.55), indicating the possibility of an overfit.

Below is the feature importance plot for GB1 model. Unlike the Random Forest model, all the hour_x features are given almost equal weightage.

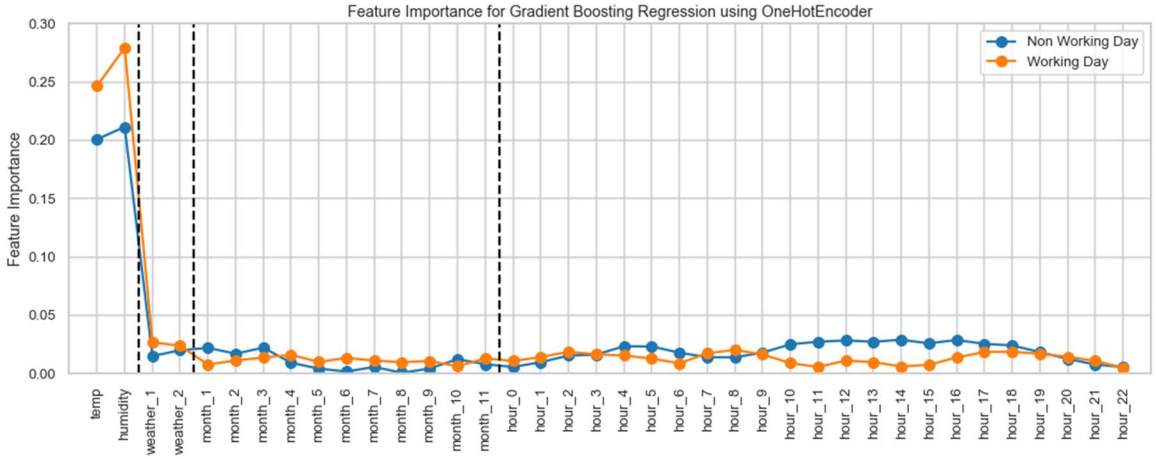


Figure 7-18: Feature Importance plot for Gradient Boost Model, GB1

7.13 Gradient Boost (GB2)

Let us now try out another variation of the Gradient Boost model. For GB2, we will have 2 models – one for Working days and another for Non-Working days. Also, we will not use OneHotEncoding on categorical features.

The below table summarizes the optimal hyperparameters for GB2 model

Hyperparameters	Working Day Model	Non-Working Day Model
n_estimators	3000	3000
learning_rate	0.01	0.005

max_features	4	4
min_samples_leaf	1	1
max_depth	21	21

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.312417	0.334973	**0.319646**
CV Test	0.404077	0.501422	**0.436876**
Test	0.387938	0.493577	**0.426262**
Total Train + Test Time			**10.1 sec**

Table 7-8: RMSLE Summary for Gradient Boost (GB2)

GB2 results in a low test RMSLE (0.43) but a significantly lower train RMSLE (0.32) seem to suggest that this model too might be overfit

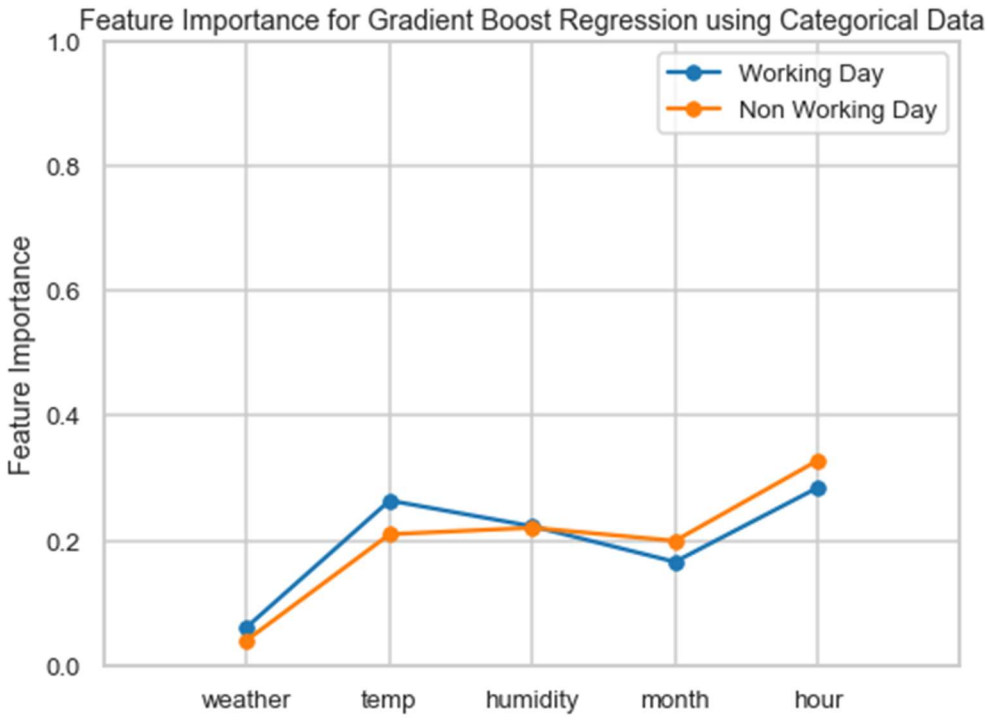


Figure 7-19: Feature Importance plot for Gradient Boost Model, GB2

7.14 Adaboost

Adaboost is another popular boosting algorithm where we convert the predictions of several weak learning algorithms into a strong one by weighted combining. Let us u

We have two hyperparameters for Adaboost and the below table summarizes the optimal hyperparameters obtained via GridSearchCV with cv=5

Hyperparameters	Working Day Model	Non-Working Day Model
n_estimators	5000	5000
learning_rate	0.001	0.001

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.604791	0.558883	**0.590809**
CV Test	0.621749	0.5989	**0.614691**
Test	0.579941	0.651514	**0.604868**
Total Train + Test Time			**59.2**

Table 7-9: RMSLE Summary for Adaboost (AB)

AdaBoost model has higher RMSLE and doesn't perform as well as the other 2 Ensemble methods (RF and GB). Below is a feature importance plot for the Adaboost model which prioritizes like other ensemble models (RF and GB)

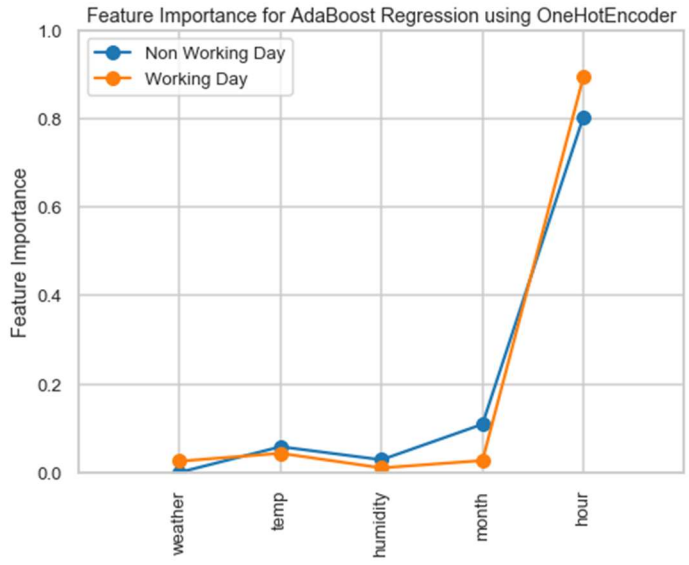


Figure 7-20: Feature Importance plot for Adaboost Model

Below, we plot the estimator errors and the weights given to each of the individual 5000 estimators used for our Adaboost model.

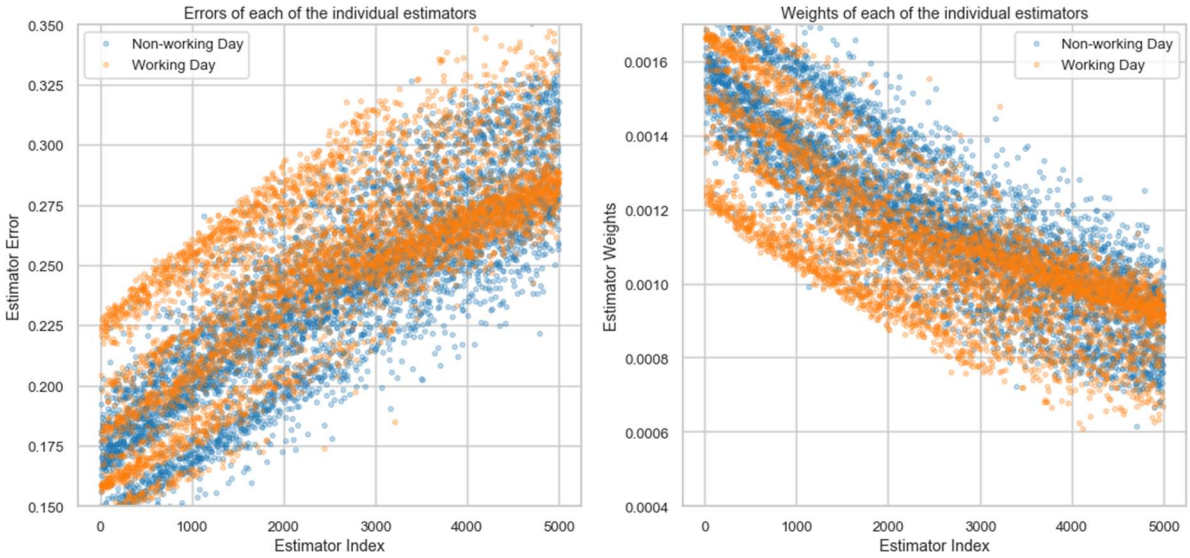


Figure 7-21: Estimator Error and Estimator Weights for each of the 5000 individual estimators for Working and Non-Working day Adaboost models

From the above plot, it AdaBoost might not be a good model for this problem. Every subsequent estimator seems to result in an increase in the error (and as a result, we give it a lower weight for our final combined estimator model)

7.15 Stacking via Linear Regression

Model stacking is an efficient ensemble method in which the predictions, generated by using various machine learning algorithms, are used as inputs in a second-layer learning algorithm. This second-layer algorithm is trained to optimally combine the model predictions to form a new set of predictions. Next few models will apply the stacking concept by using the predictions obtained from above individual models (7.6 to 7.14) as meta-features to build a new ensemble model.

First let us use Linear Regression as the second layer model - this would estimate the weight for each of the individual model predictions by minimizing the least square errors.

The RMSLE and the model train+test time summary for this stacked Linear Regression model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.436751	0.499553	**0.457331**
Test	0.402243	0.48662	**0.432355**
Total Train + Test Time			**584 secs**

Table 7-10: RMSLE Summary for Stacking via Linear Regression

Stacking using Linear Regression results in poorer test performance (0.46 RMSLE) compared to few ensemble individual models (e.g. RF1 with 0.43 RMSLE).

Also note that the total train+test time for stacking models are usually very high. This is because, we must train+test every individual model first, to obtain the predicted value which then becomes the feature set for the stacked model.

Below we plot the Coefficients obtained for our Level 2 Linear Regression model.

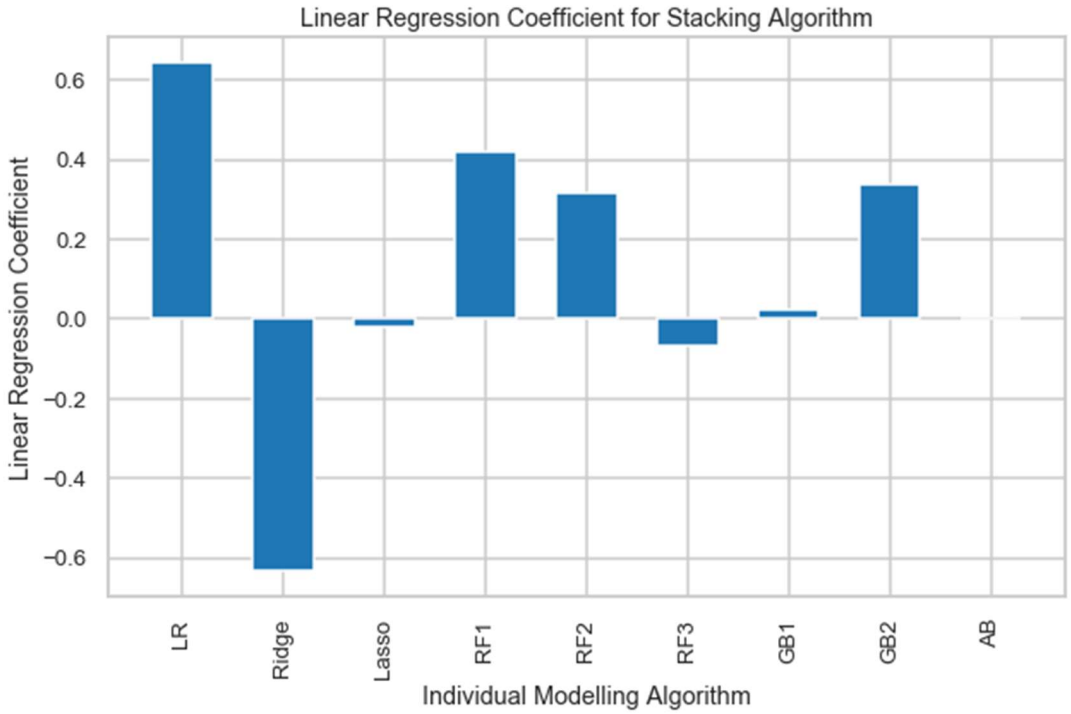


Figure 7-22: Linear Regression Coefficients for Stacking Algorithm

- We notice that the Lasso and GB1 have been given very low values indicating that those models were probably not very accurate.
- LR and Ridge have equal opposite signed coefficients. Since these two are highly correlated (as we saw by their coefficients in Figure 7-9: Feature Coefficient Comparison for the Linear Models (Linear Regression, Ridge and Lasso)
- So, to sum up, this Linear Regression seem to primarily use the predicted values from RF1, RF2 and GB2.

7.16 Stacking via Random Forest

Let us use Random Forest to model our stacking model. The below table summarizes the optimal hyperparameters for this stacked Random Forest model

Hyperparameters	Optimal Value
n_estimators	1000
max_features	'sqrt'
min_samples_leaf	10
max_depth	5

The RMSLE and the model train+test time summary for the model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.422611	0.472015	**0.438669**
Test	0.3899	0.494256	**0.427714**
Total Train + Test Time			**593 secs**

Table 7-11: RMSLE Summary for Stacking via Random Forest

Stacking using Linear Regression results in poorer test performance (0.46 RMSLE) compared to few ensemble individual models (e.g. Random Forest 3.5.1 with 0.43 RMSLE)

Below is feature importance plot for our stacked Random Forest model.

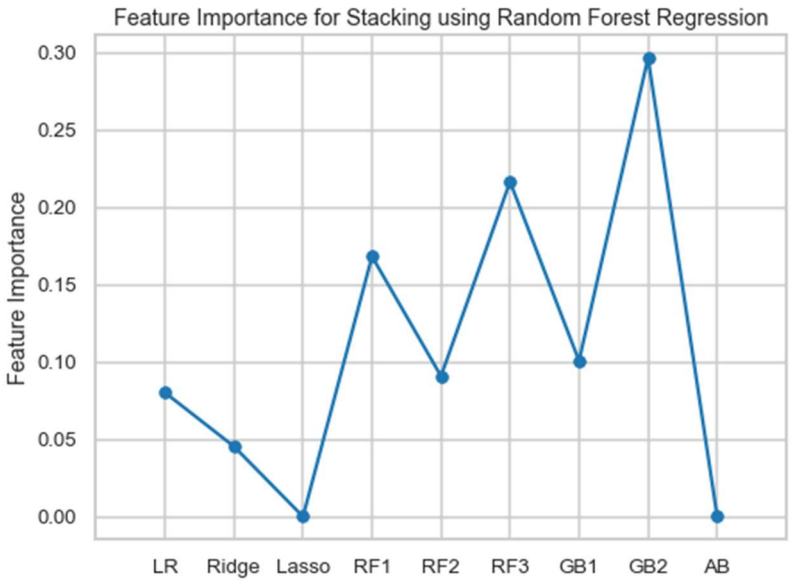


Figure 7-23: Feature importance plot for Stacking using Random Forest Regression

Lasso and Adaboost model have zero importance (which agrees with our RMSLE analysis). Predictions from GB2, RF3 and RF1 seem to get most importance

7.17 Stacking via Gradient Boost

Finally, let us use Gradient Boost to model our stacking model. The below table summarizes the optimal hyperparameters for this stacked Gradient Boost model

Hyperparameters	Optimal Value
n_estimators	3000
learning_rate	0.01
max_features	0.3
min_samples_leaf	100
max_depth	4

The RMSLE and the model train+test time summary for this stacked Gradient Boost model is given in the below table

	Working Day	Non-Working Day	All Combined
Train	0.408437	0.458788	**0.424835**
Test	0.393135	0.494927	**0.429937**
Total Train + Test Time			**592 secs**

Table 7-12: RMSLE Summary for Stacking via Gradient Boost

Below is feature importance plot for our stacked Gradient Boost model.

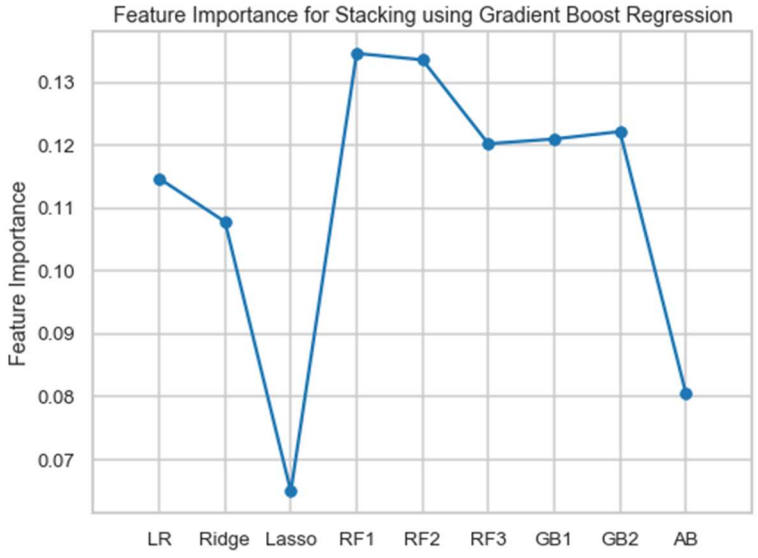


Figure 7-24: Feature importance plot for Stacking using Gradient Boost Regression

As with the Random Forest stacked model, Lasso and Adaboost gets the lowest importance. All the remaining models are given more or less equal priority here.

8 SUMMARY & CONCLUSIONS

8.1 RMSLE and Train+Test time

We summarize the average Test RMSLE and Train+Test time for every model we used in the below bar charts

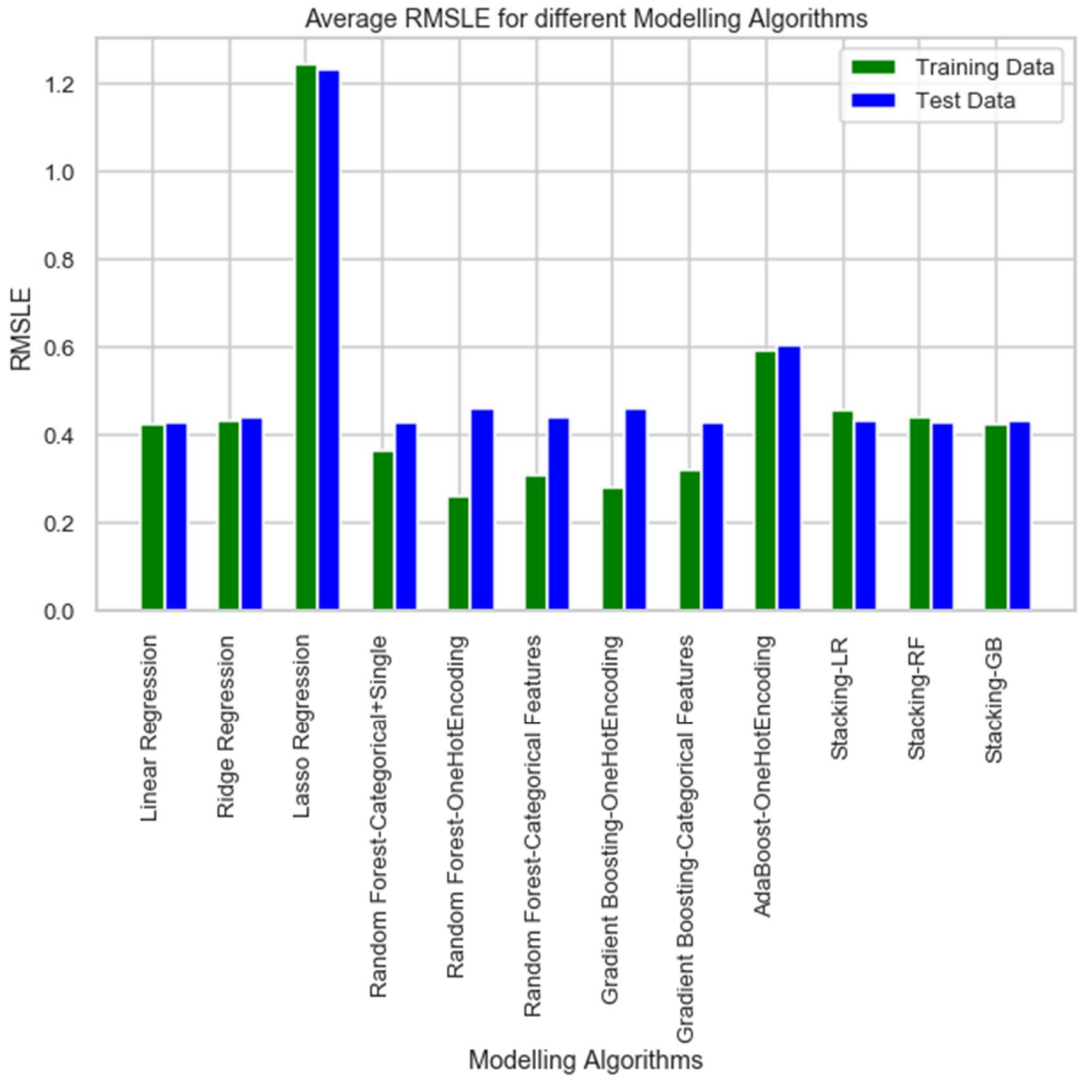


Figure 8-1: Test RMSLE for all the Modeling algorithms

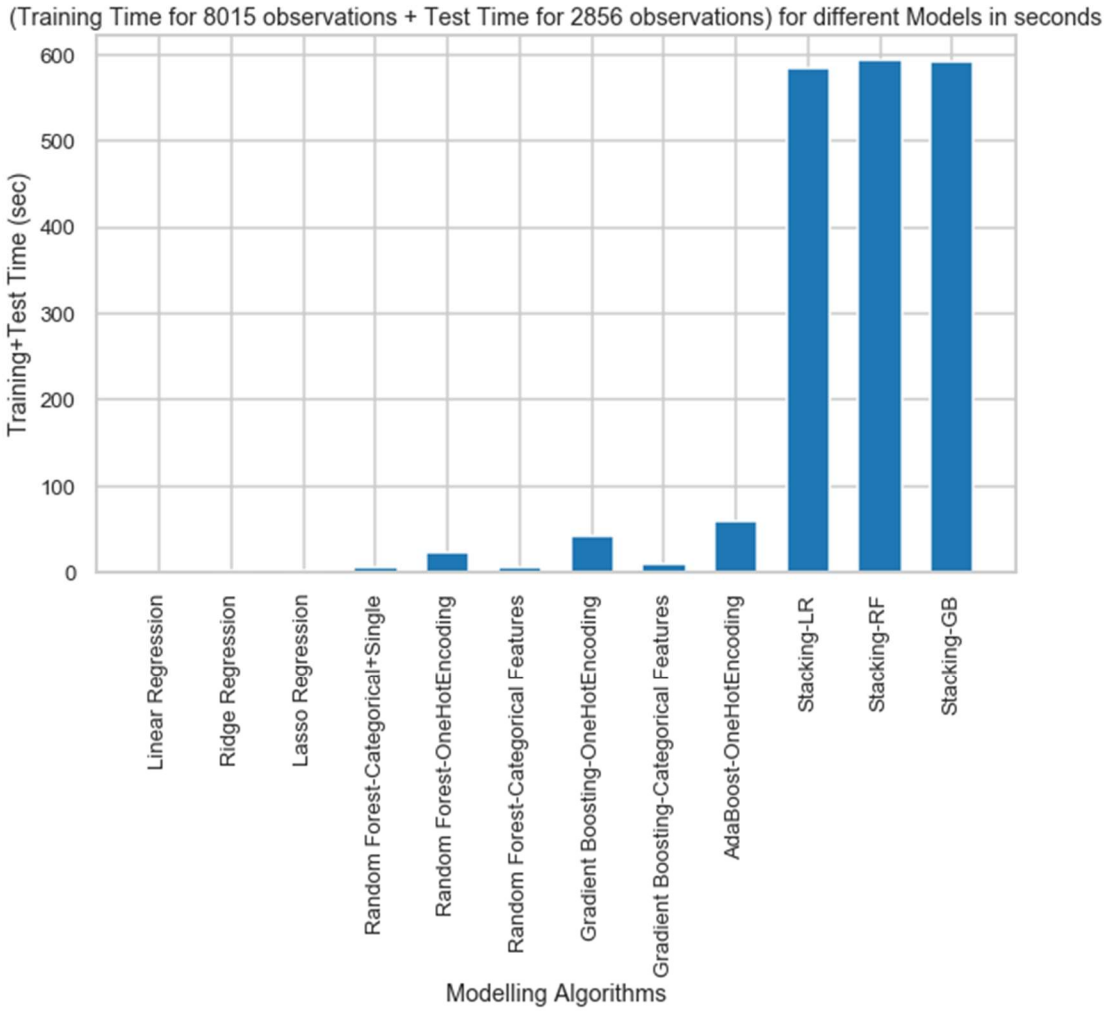


Figure 8-2: Train+Test time for all the considered Models

Based on the above results, let us use RF1, i.e., Single Model Random Forest Regressor Model (with categorical feature) based on the below couple of reasons

- One of the lowest Average Test RMSLE
- Single model which works for both working and non-working days
- Not an Overfit model (Train Average RMSLE is closer to Test Average RMSLE)
- Lesser to train due to lesser number of features

Other reasonable choice would be the Linear Regression Model.

8.2 Summary

In this report, we train a model to predict the number of bike rentals at any hour of the year given the weather conditions. The data set was obtained from the Capital Bikeshare program in Washington, D.C. which contained the historical bike usage pattern with weather data spanning two years.

First, we do Exploratory Data Analysis on the data set. We look for missing data values (none were found) and outliers and appropriately modify them. We also perform correlation analysis to extract out the important and relevant feature set and later perform feature engineering to modify few existing columns and drop out irrelevant ones.

We then look at several popular individual models from simple ones like Linear Regressor and Regularization Models (Ridge and Lasso) to more complicated ensemble ones like Random Forest, Gradient Boost and Adaboost. Additionally, few options for model formulation were tried - 1. A single unified model for working and non-working days, 2. Two separate models for working and non-working days, 3. Using OneHotEncoding to get Binary Vector representation of Categorical Features and 4. Using Categorical Features as provided. Finally, we also tried stacking algorithms where the predictions from the level 1 individual models were used as meta-features into a second level model (Linear Regressor, Random Forest and Gradient Boost) to further enhance the predicting capabilities.

The labeled data set provided comprised of the first 19 days of each month while the Kaggle Test Data comprised of rental information from 20th to the end of the month. Hyperparameters were tuned using GridSearchCV cross validation using 5 folds on part of the provided training data set (first 14 days of each month). The remaining data (15th to 19th of each month) were used as hold out set to test our model performance. Of all the methods and models, we found Random Forest Ensemble method using a Single Model and Categorical Feature set an ideal choice with train and test scores of 0.36 and 0.427, respectively. The training and test time for the chosen model are very reasonable (~23 seconds for total train+test observation size = 10871). The chosen model yields a score of 0.49 on Kaggle Test Data

8.2.1 Data Exploration Conclusions

In this project, we explored several factors that influenced bike rental count. Below is a quick summary of exploratory data analysis

- Working or Non-working Day We see 2 rental patterns across the day in bike rentals count - first for a Working Day where the rental count high at peak office hours (8am and 5pm) and the second for a Non-working day where rental count is more or less uniform across the day with a peak at around noon.
- Hour of the day: Bike rental count is mostly correlated with the time of the day. As indicated above, the count reaches a high point during peak hours on a working day and is mostly uniform during the day on a non-working day
- Casual and Registered Users: While most casual users are likely to be tourists whose rental count is high during non-working days, most registered users are most likely city natives whose rental count is high during working days

- Temperature: People generally prefer to bike at moderate to high temperatures. We see highest rental counts between 32 to 36 degrees Celsius
- Season: We see highest number bike rentals in Fall (July to September) and Summer (April to June) Seasons and the lowest in Spring (January to March) season
- Weather: As one would expect, we see highest number of bike rentals on a clear day and the lowest on a snowy or rainy day
- Humidity: With increasing humidity, we see decrease in the number of bike rental count.

8.2.2 Modeling Conclusions

We used 6 Regression Models to predict the bike rental count at any hour of the day - Linear Regression, Ridge, Lasso, Random Forest, Gradient Boost and Adaboost. Using the predictions made by these level 1 individual models as features, we trained 3 level 2 stacking algorithms (Linear Regression, Random Forest and Gradient Boost) to make more refined predictions. Below is a summary of the model performances

- Of all the models, we found a simple Random Forest Model providing the best/lowest RMSLE score.
- Stacking individual models didn't provide any improvement over the best individual model
- Having separate models for Working days and Non-working days didn't provide any improvement in prediction accuracy
- '*Hour*' of the day holds most importance among all the features for prediction

8.2.3 Limitations and Scope for Model Improvements

Below are few limitations in this analysis and ideas to improve model prediction accuracy

- Since casual + registered = total count, we just predicted the total bike rental count by ignoring the casual and registered user information. Another (possibly better) method would be to train separate models for casual and registered users and add the two
- Windspeed wasn't used due to very low correlation. This might have been due to several instances where windspeed = 0. One possible method could be to first estimate those windspeed and then use it as a feature to estimate count.
- One limitation in the provided training data set is that it lacked data with extreme weather condition data (weather = 4). Hence, we had to modify it to weather = 3