Predictive Analytics for Bike Sharing Dataset

Emmanuel Cocom, Kristen Marenco, Imelda Flores

I. Introduction

Modern day bike sharing systems allow both registered and casual (‘walk in’) users to travel across cities, counties, and even to more remote destinations. Data from theses users is constantly being collected for analytics purposes. A large interest in collecting this data is to help prepare for a change in the demand of bike rental from their users. To help bring greater insight into this task, we have taken on a challenge on Kaggle to forecast bike rental demand in Washington, D.C.

II. Data

Details

The data provided on Kaggle spans over a time period of two years. Only the first 19 days of each month are provided. The test data set starts from the 20th and continues until the end of the month. However, since the test data does not include ‘count’ (which is the data being predicted), this means the predictions cannot be verified. To address this shortcoming, the training dataset was split into test and training data.

The actual data provided consists of 11 Features:

- Features:
    - ‘datetime’: day/month/year hour in a string format
    - ‘season’:
        1 - Spring
        2 - Summer

3 - Fall

4 - Winter

- ‘holiday’:

  0 - Non-Holiday

  1- Holiday

- ‘workingday’:

  0 - Non-Working Day

  1 - Working Day

- ‘weather’:

  1 - Clear, Partly cloudy, Partly cloudy

  2 - Mist + Cloudy, Mist + Broken clouds, Mist + Few Clouds, Mist

  3 - Light Snow, Light Rain + Thunderstorm + Scattered Clouds, Light Rain

  4 - Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog

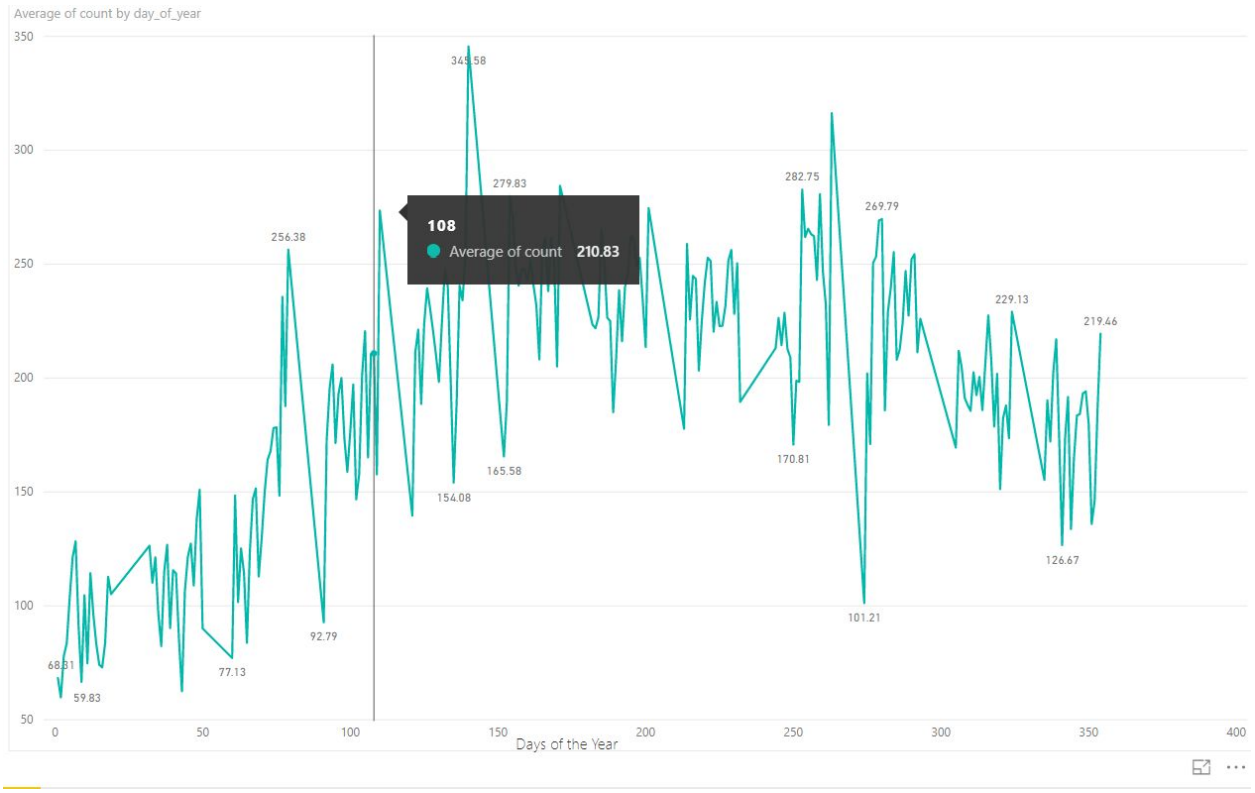
- ‘temp’: temperature in Celsius
- ‘atemp’: what the temp “feels like” in Celsius
- ‘humidity’: relative humidity
- ‘windspeed’: wind speed
- ‘casual’: number of non-registered user rentals initiated
- ‘registered’: number of registered user rentals initiated

- Label:
  - ‘count’: number of total rentals

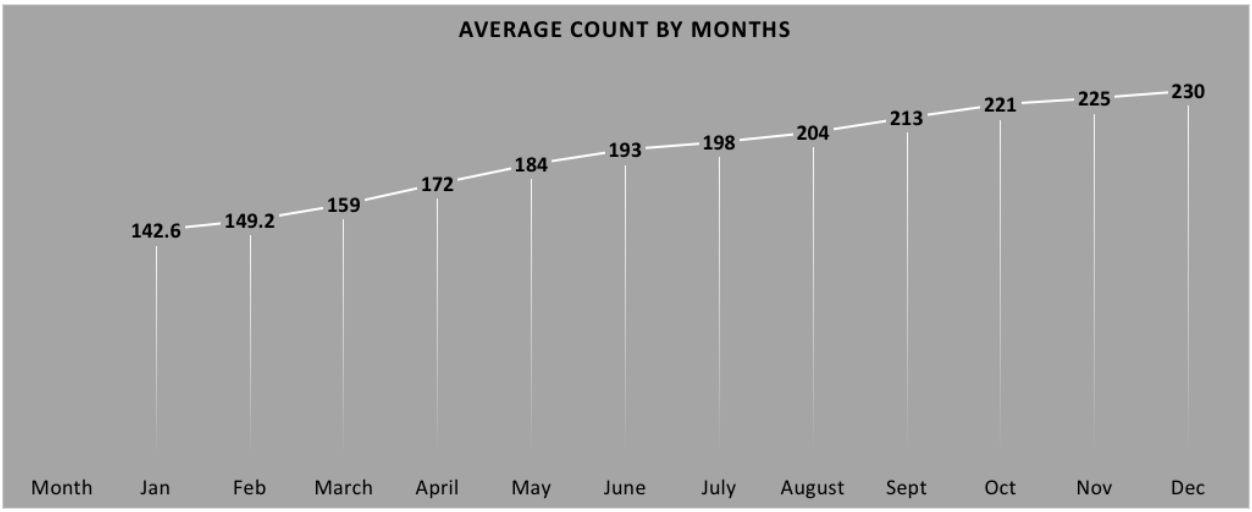
Goals:

To predict the total count of bikes rented using the given data.

Visualization



(Visual. Raw Data - Day Average)



(Visual. Raw Data - Month Average)

Observations

The ‘count’ average from day to day varies greatly.

The ‘count’ average from month to month is steadily rising.

III. Methods

Methods, Tools, and Algorithms

A. Dropping Leakage Variables

The dataset contained two ‘leakage variables’ ‘casual’ and ‘registered.’ This is to say they are not in features but our label split into two columns together. ‘Casual’ Walk-in rentals and ‘registered’ rentals together sum our label the total ‘count’ of rentals. Both these columns were dropped.

B. Extracting Data

Our dataset contained a feature called ‘datetime’ that had a format of ‘day/month/year hour:00’. We extracted several columns of information including day of the year, hour, months, and year. The following are the extracted features and the values possible for each feature.

- “hour” :

  0 - 1 am , 11- 12 pm , 23 - 12am

- “weekday” :

  0 - Sun, 1 - Mon, 2 - Tues, 3 - Wed, 4 - Thurs, 5 - Friday, 6 - Sat

- “Months”:

  1 - Jan, 2- Feb, 3-March, 4- April, 5-May, 6-June, 7-July, 8- August, 9-September, 10 - October, 11- November, 12 - December

- “Year”: 2011, 2012

C. Correcting Corrupt Data

Test were performed on each feature column to ensure data that was not missing and/or corrupt. No missing data was found, but ‘humidity’ had values of 0 in several rows which is impossible. These were replaced with the average of the column. All devised test can be seen in section D of the Methods section. The following are snippets of using the tests on a feature column.

```
#weather:: clear
bycle_df.weather.apply(lambda x: check_if_empty_ints(x))#results show NO missing values
bycle_df.weather.apply(lambda x: check_if_less_than_x_greater_than_y(x, 1, 4))#results show NO corrupt values
bycle_df.weather.unique()

#atemp :: clear
bycle_df.atemp.apply(lambda x: check_if_empty_ints(x))#results show NO missing values
bycle_df.atemp.apply(lambda x: check_if_less_than(x, 0.1))#results show NO corrupt values
bycle_df.atemp.unique()

#-humidity :: NOT CLEAR! --> NO MISSING DATA: HOWEVER ---> CORRUPT DATA: YES
bycle_df.humidity.apply(lambda x: check_if_empty_ints(x)) #results show NO missing values
bycle_df.humidity.unique() #results show YES Corrupt Data, as it's impossible to have 0 humidity on earth
```

D. Custom Functions

Test Functions

```
#HELPER FUNCTIONS, TO TEST FOR MISSING OR CORRUPT DATA IN EACH COLUMN/FEATURE

#helper for string data
def check_if_missing_strings(value):
    if value == None or value.strip() == '' or value.lower() == 'nan' or value == np.nan:
        print(value, j, ' is the missing value')
        return np.nan
    else:
        return value



#Helper for numerical data
def check_if_empty_ints(value):
    if value == '' or value == None or value == 'nan' or value == np.nan:
        print('missing value')
    else:
        pass

def check_if_less_one(value):
    if value <1:
        print('less than one, check it out!')

def check_if_greater_than(value, limit):
    if  value > limit:
        print('less than one, check it out!')
def check_if_less_than(value, limit):
    if value < limit:
        print('less than one, check it out!')
def check_if_less_than_x_greater_than_y(value, x_limit, y_limit):
    if value < x_limit:
        print(value)
        print(' is dangerous data and less than ', x_limit)
    if value > y_limit:
        print(value)
        print(' is dangerous data and greater than ', y_limit)
```

Fix Corrupt Data:

```
#verify corrupt data
bycle_df['humidity'].unique()

#changes zero corrupt data to numpy.nan value
random_var = bycle_df.humidity.apply(lambda x: nan_if_zero(x))

#finds mean of column, ignores nan values by default
mean_humid = random_var.mean()


#apply mean to all zero corrupt data values in original df column humidity
bycle_df['humidity'] = bycle_df.humidity.apply(lambda x: mean_if_nan(int(x), int(mean_humid)))

#DATA IS CLEARED NOW --> NO CORRUPTION IN HUMIDITY COLUMN
bycle_df['humidity'].unique()
```

Hours/Years to Bins:

```
def year_to_bin(value):
    if value == '2011':
        return 0
    return 1

bycle_df['year']= bycle_df.year.apply(lambda x : year_to_bin(x))
```

```
#HELPER FUNCTION TO PUT HOUR VALUES INTO BINS
def four_hour_bins(hour):
    hour = int(hour)
    if hour <=5:
        return 1
    elif hour <=11:
        return 2
    elif hour <=17:
        return 3
    else:
        return 4
```

E. One Hot Encoding

Our data set also included several categorical features which were label encoded. This is to say that they had a number assigned to each category representation. For

example Spring is represented by a value of 1 and summer by 2. But even though 1 + 1 == 2 two springs do not equal a summer. The provided formatted suggest such relationships. Therefore, we put each categorical non-binary feature column through a OneHotEncoder Process. This kept a number representation for each categorical feature without retaining those unintended numerical relationships.

Furthermore, the ‘hours’ feature was put into 4 bins and label encoded prior to undergoing the One Hot Encoding process. The ‘years’ column was label encoded. Months was put into 12 bins and then label encoded prior to undergoing the One Hot Encoding process.

Label Encoded:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count	hour	day_of_year	weekday
0	1/1/11 0:00	1	0	0	1	9.84	14.395	81	0.0	16	0	1	6
1	1/1/11 1:00	1	0	0	1	9.02	13.635	80	0.0	40	1	1	6
2	1/1/11 2:00	1	0	0	1	9.02	13.635	80	0.0	32	2	1	6
3	1/1/11 3:00	1	0	0	1	9.84	14.395	75	0.0	13	3	1	6
4	1/1/11 4:00	1	0	0	1	9.84	14.395	75	0.0	1	4	1	6

One Hot Encoded

```
OneHotEncoded DF:

      spring,  summer  fall  winter   clear  mist  light  heavy_rain  \
0         1.0     0.0   0.0     0.0     1.0   0.0    0.0         0.0
50        1.0     0.0   0.0     0.0     1.0   0.0    0.0         0.0
100       1.0     0.0   0.0     0.0     1.0   0.0    0.0         0.0
150       1.0     0.0   0.0     0.0     0.0   1.0    0.0         0.0
200       1.0     0.0   0.0     0.0     1.0   0.0    0.0         0.0
250       1.0     0.0   0.0     0.0     0.0   0.0    1.0         0.0

      EarlyMorning  Morning ...   March  April  May  June  July  August  Sept  \
0              1.0      0.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0
50             1.0      0.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0
100            1.0      0.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0
150            0.0      1.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0
200            1.0      0.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0
250            0.0      0.0 ...     0.0    0.0  0.0   0.0   0.0     0.0   0.0

     Oct  Nov  Dec
0    0.0  0.0  0.0
50   0.0  0.0  0.0
100  0.0  0.0  0.0
150  0.0  0.0  0.0
200  0.0  0.0  0.0
250  0.0  0.0  0.0

[6 rows x 24 columns]
```

F. New Feature After One Hot Encoding

a. 4 Season Columns: “Spring”, “Summer”, “Fall”, “Winter”

b. 4 Weather Columns:

“ Clear, Few Clouds, Cloudy/Broken/Few clouds”,

“ Light Snow + Light Rain + Thunderstorm/Scattered Clouds/Light Rain/Scattered clouds”,

“Heavy Rain/Ice Pellets/Thunderstorm/Mist/Snow + Fog"

c. 12 Month Columns:

1 - Jan, 2- Feb, 3-March, 4- April, 5-May, 6-June, 7-July, 8- August, 9-September, 10 - October, 11- November, 12 - December

G. Columns Discarded

H. Columns Discarded:

After OneHotEncoder

a. “Season”

b. “Weather”

c. “Year” (applies only to approach 3)

d. “Months” (applies only to approach 3)

After Datetime stamp data extraction

e. “Datetime”

I. Total Number of Features

a. Approach 1: KNN Regression, XGBoost, Decision Tree: 20 features

b. Approach 2: 31 features (includes months of the year, and year columns)

J. Normalizing Data

To ensure that features whose values are high numbers are not weighed more than features whose value ranges are low all data was normalized. This was done using the sklearn preprocessing method. Below the code for this process can be seen.

```
from sklearn import preprocessing

#normalize data
for x in range(len(df_splits)):
    #scale it -> d type changes to numpy array
    scaled_feature_matrix_month_numpyarray = preprocessing.scale(df_splits[x][0])

    #change back to df
    df_month_scaled = pd.DataFrame(scaled_feature_matrix_month_numpyarray, columns = df_splits[x

    #store back the scaled data back into list.
    df_splits[x][0] = df_month_scaled

print('sample of list of stored monthly dataframes and label\n\n')
print(df_splits[0][0].head())
print(df_splits[0][1][:5:])
```

```
sample of list of stored monthly dataframes and label


    holiday  workingday          temp     atemp  humidity  windspeed  \
0 -0.295518   -1.283241 -4.389351e-16  0.479102  1.436178    -1.7049
1 -0.295518   -1.283241 -2.026208e-01  0.321980  1.378933    -1.7049
2 -0.295518   -1.283241 -2.026208e-01  0.321980  1.378933    -1.7049
3 -0.295518   -1.283241 -4.389351e-16  0.479102  1.092707    -1.7049
4 -0.295518   -1.283241 -4.389351e-16  0.479102  1.092707    -1.7049

   day_of_year   weekday      year   spring, ...   March  April  May  June  \
0    -1.640371  1.600362 -1.025204        0.0 ...     0.0    0.0  0.0   0.0
1    -1.640371  1.600362 -1.025204        0.0 ...     0.0    0.0  0.0   0.0
2    -1.640371  1.600362 -1.025204        0.0 ...     0.0    0.0  0.0   0.0
3    -1.640371  1.600362 -1.025204        0.0 ...     0.0    0.0  0.0   0.0
4    -1.640371  1.600362 -1.025204        0.0 ...     0.0    0.0  0.0   0.0
```

K. Other Methods:

K Fold Cross - Validation:

10 Fold Cross - Validation is a resampling procedure used to evaluate the algorithms on the data sample. The procedure consists of a parameter called *k*. *K* refers to the number of groups the data is split into. In this case, the date will consist of ten groups. First, the data will be shuffled. Second, the data will be split into 10 groups. Third, for each set of groups cross validation will take a group that will be used as test data. The remaining nine groups will be used as training data. It will continue to repeat until all ten

groups have been used as test data. The results are summarized with the mean of the scores achieved with the algorithm.

Root Mean Square Error (RMSE)

RMSE measures the average magnitude of the error. This is the square root of the average of differences between prediction and actual results. The errors are squared before are averaged.

Root Mean Squared Logarithmic Error (RMSLE)

RMSLE is the RMSE of the log-transformed predicted and target values. Using this measurement with a wide range is useful in the target variable. It is effective when the percentage errors are more important than absolute value of errors.

Adaptive Boosting (ADA boost)

ADA Boost mainly focuses on converting weak learning algorithms become a strong learning algorithm. ADA boost will boost the performance of the algorithms to achieve better results. It finds the weighted prediction of the algorithm. Each instance in the training set is weighted. The weight of each trained algorithm depends on the best results achieved.

Principal Component Analysis (PCA)

PCA's primary component is to reduce the dimensionality of the data set. A data set that consists of many variables that correlate with one another. It reduces the data while retaining the variation in the data set. With PCA the variation will be emphasized and bring out strong patterns in the data set.

L. Tools

Programs: Python, Jupyter Notebook, Excel, Power Bi, Google, and Docx.

Libraries: Pandas, Numpy, Datetime, Matplotlib.

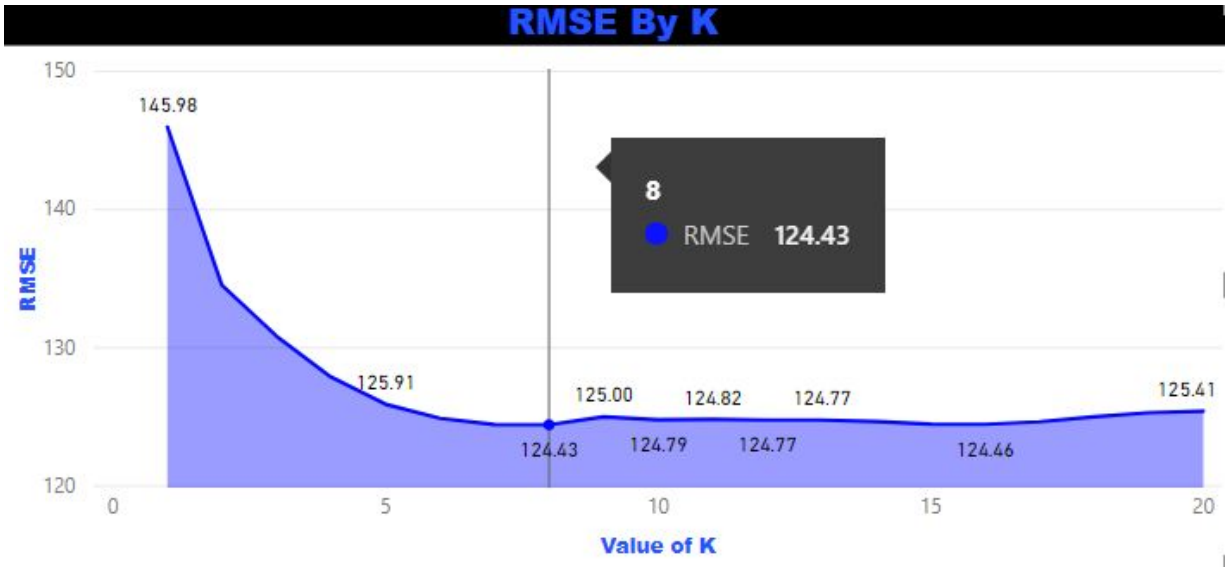
M. Algorithms

KNN Regressor

Distance functions

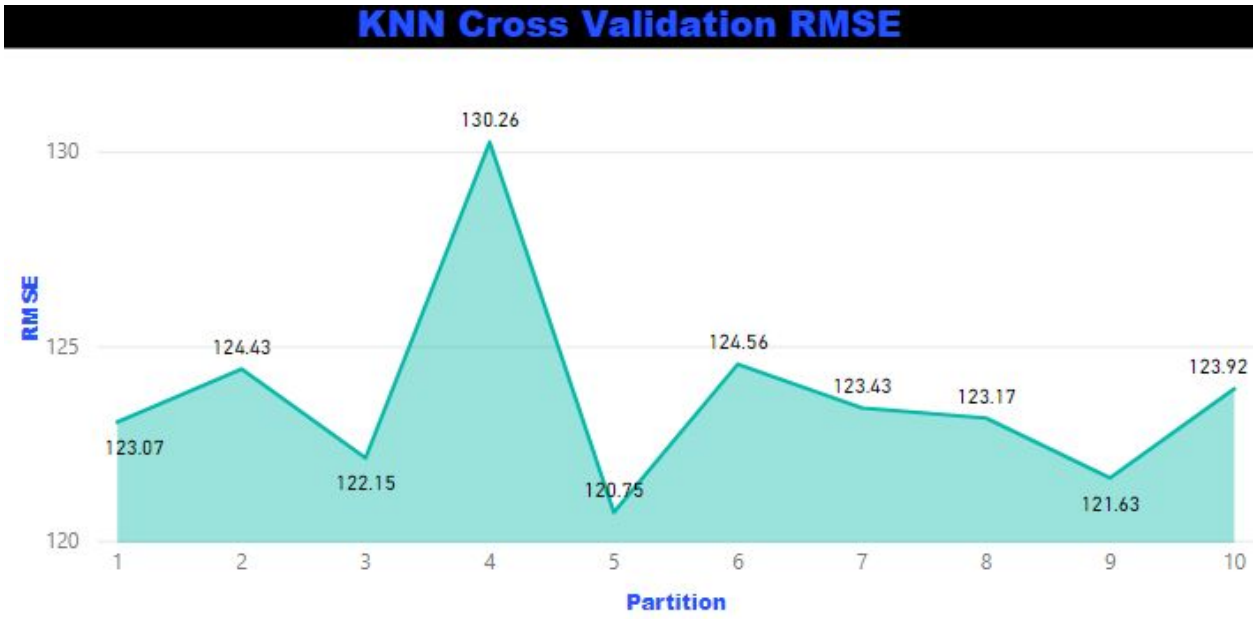
Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

The K nearest neighbor regression algorithm does predictions based on the majority of K closest training samples in the feature space, much similar to KNN classifier algorithm. The KNN regressor algorithm uses the euclidean distance function to determine the closest training samples depending on the value of K. There were 20 features used in the training and testing set. These 20 features were the common features listed earlier in the data section of this report. The dataset was split into two parts, 25% became the testing set and 75% became the training set.



When applying the KNN regression algorithm to the dataset the results found that with a range of K from 1-20, the lowest rmse found was, 124.42 when K was set to 8. Kaggle recommended using rmsle (root mean squared logarithmic error) as another form of calculating error for this problem. When K = 8 , the RMSLE was, 0.78. The lower the value for error, the better the algorithm.

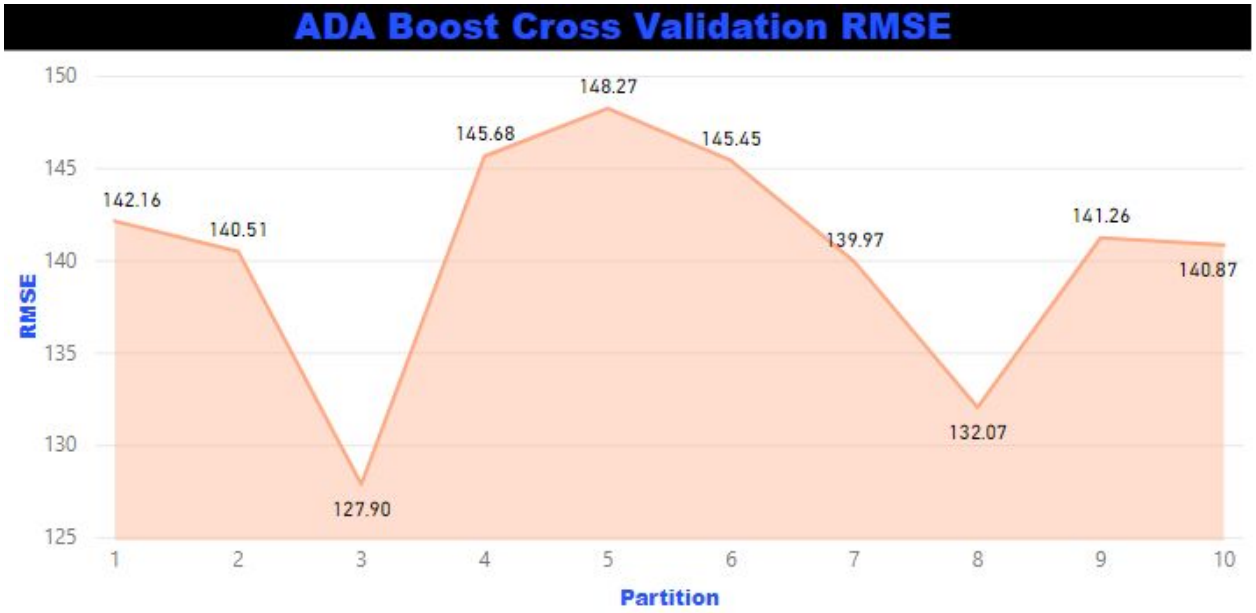
After finding the best value for K, 10-fold cross validation was applied in order to perform fair evaluation. The results are as follows:



The mean of the RMSE values was 123.7682680498514, which is actually lower than the RMSE value originally found when only testing on one portion of the dataset.

Principal Component Analysis was then applied to the dataset. PCA reduces the dimensionality of a dataset by taking variables that are redundant and transforming them to a new set of variables referred to as principal components. The PCA technique is usually used for data with high dimensions such as, facial recognition, and image compression. After applying the

PCA technique to the dataset, ADA Boost was then applied with cross validation to test if the RMSE would improve. The results are as follows:



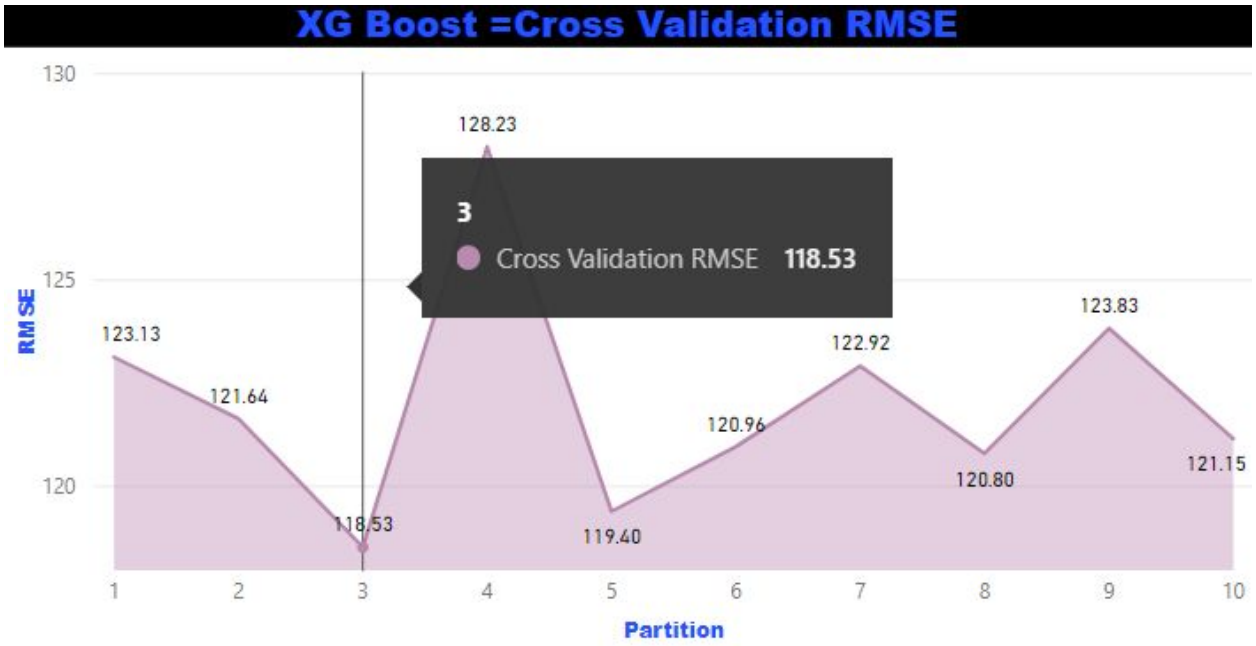
The mean of the RMSE values was 140.41378325969782, which is much higher than the original value found using KNN Regressor on one portion of the dataset.

XGBoost Regressor

Boosting is an approach that converts a set of weak learners into strong learners. To convert a weak learner into a strong learner, a boosting algorithm takes a set of weak learners, then combines them and uses voting. The set of weak learners must have low correlation between them to reduce error. XGBoost, or Extreme Gradient Boosting is a much more effective version of the Gradient Boosting algorithm. Gradient Boosting creates a loss function and then tries to optimise it in order to reduce loss. However, XGBoost goes a step further. The difference between these two algorithms are the mathematical functions they implement. While Gradient Boosting uses the Gradient Descent method taught in class, XGBoost implements the

Newton-Raphson method which provides a more direct route to the minima compared to gradient descent.

XGBoost with 10-fold cross validation was applied to the dataset. The results are as follows:



The mean of the RMSE values was 122.05911203438066, which is actually lower than the original value found using KNN Regressor on one portion of the dataset.

Decision Tree Regressor

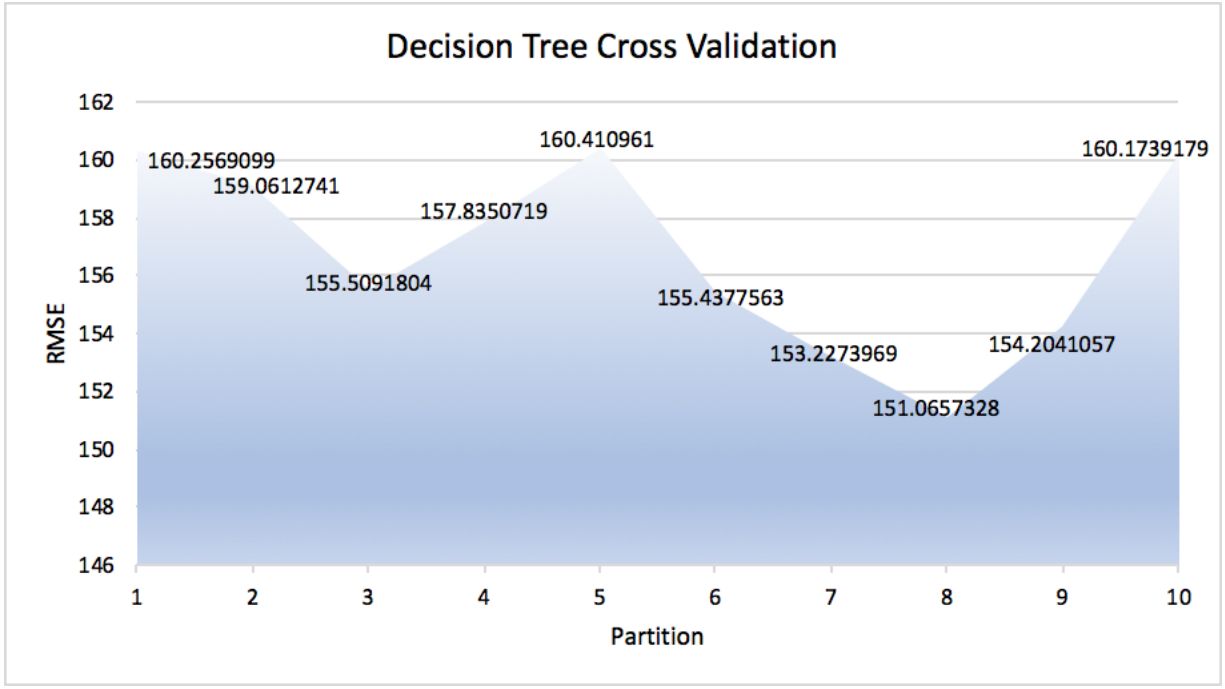
A Decision Tree Regressor is a tool based on a tree like model of decisions and possible consequences, which include outcomes, resource costs, and utility. The tree contains leaves, each leaf is labeled with a probability of an event occurring. The Decision Tree advantages is that it implicitly performs feature selection and handles nonlinear data. It is easily interpretable by human. It handles both numerical and categorical data. The tree ‘learns’ from the data by

splitting the data set into subsets. The information gained is based on the 'entropy' calculated after the data is split. The entropy formula is the following:

$$H(X) = -\sum_{k=1}^{K} p(X = x_k) \log_2 p(X = x_k)$$

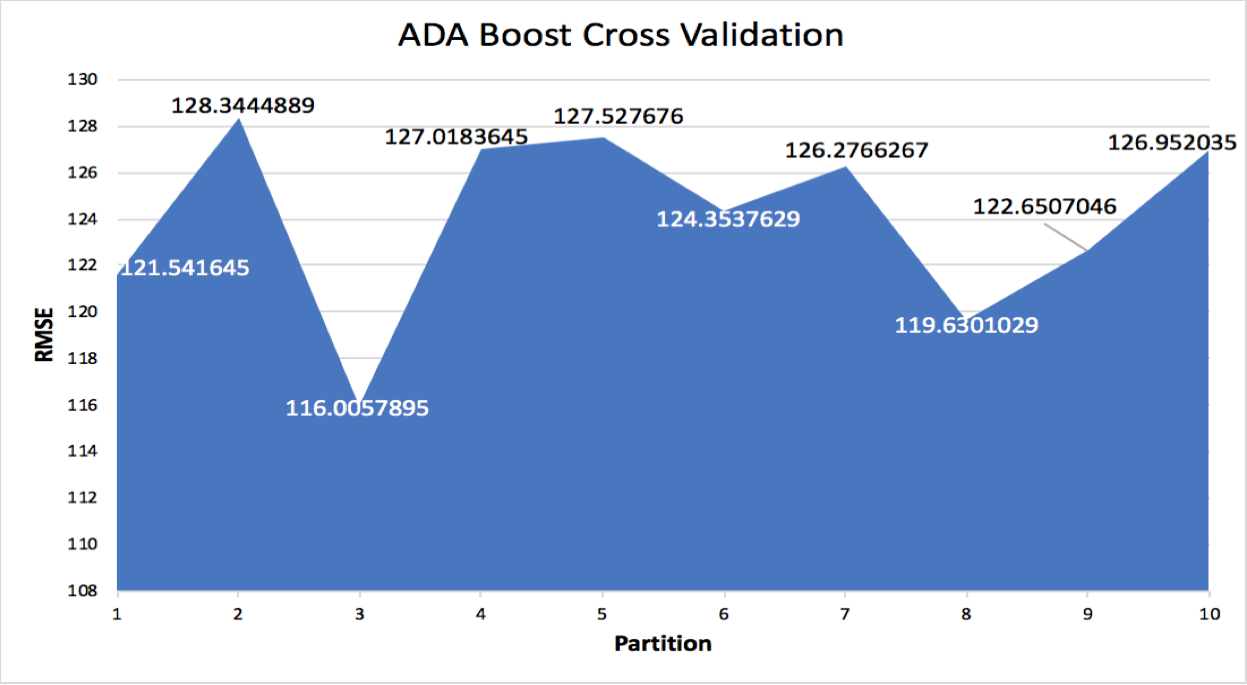
where *p* is the probability of an event occurring. The number of features used when training and testing were 20 features. The features are listed in the 'Data' section. The data set was split into a test set and train set. The test set was 25% of the data. The training set was 75% of the data. The random state was set to 3 when Decision Tree was applied to our data set.

After setting the test size, train size, and random state, 10 - fold cross validation was applied to achieve better results. The RMSE results were: [160.25690988, 159.06127409, 155.50918042, 157.83507194, 160.41096098, 155.43775626, 153.22739685, 151.06573281, 154.2041057, 160.17391785]. The RMSE mean was 155.42495591453672.



(DT Figure 1: Cross Validation)

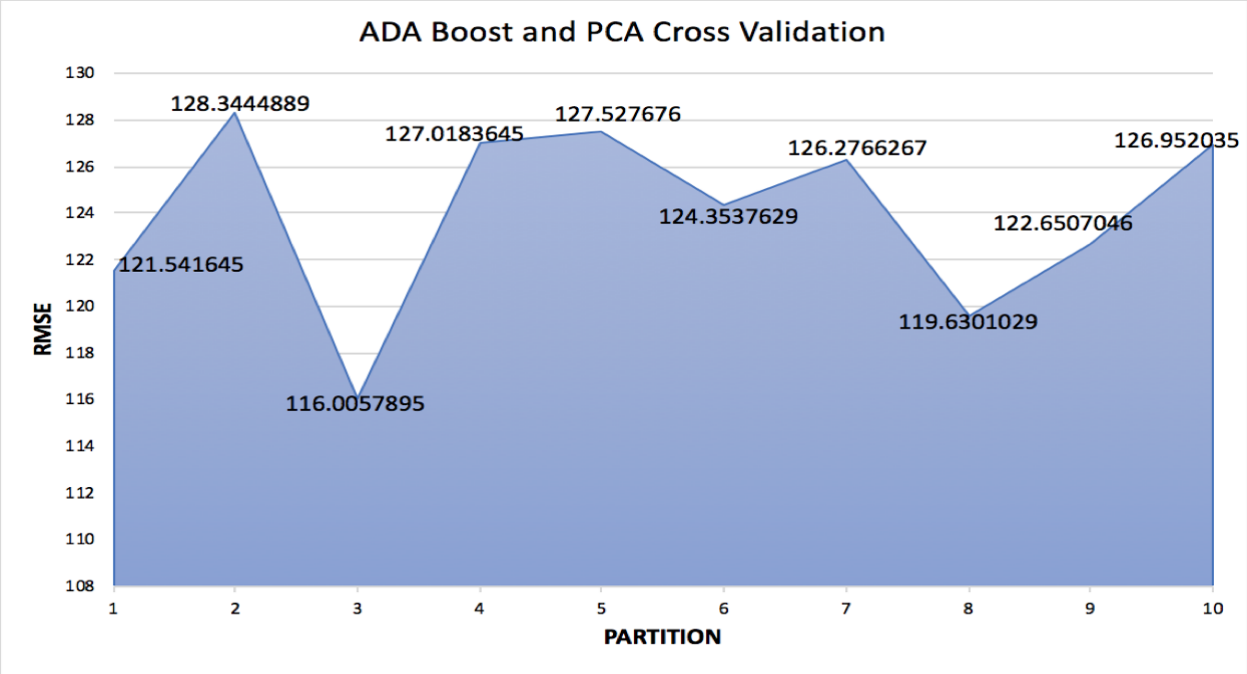
To improve the performance of Decision Tree, ADA Boost was applied. ADA Boost was set n_estimators to 300 and a random state set to 3. The same splitted data was used with 10 - fold cross validation. The RMSE results were: [121.54164498, 128.34448891, 116.00578948, 127.01836446, 127.52767603, 124.35376289, 126.2766267, 119.63010286, 122.65070464, 126.95203501]. The RMSE mean was: 124.03011959584452. The RMSE value decreased when ADA Boost was used with Decision Tree. The lower the RMSE is the better the results are.



(DT Figure 2: RMSE of ADA Boost Cross Validation)

PCA was applied to improve results. The data was normalized using standard scaler. The PCA was applied on the training data. ADA Boost was then applied to PCA. The RMSE results with both algorithms were: [121.54164498, 128.34448891, 116.00578948 127.01836446, 127.52767603, 124.35376289, 126.2766267, 119.63010286, 122.65070464, 126.95203501].The RMSE results did not change from using ADA Boost alone. The RMSE mean was:

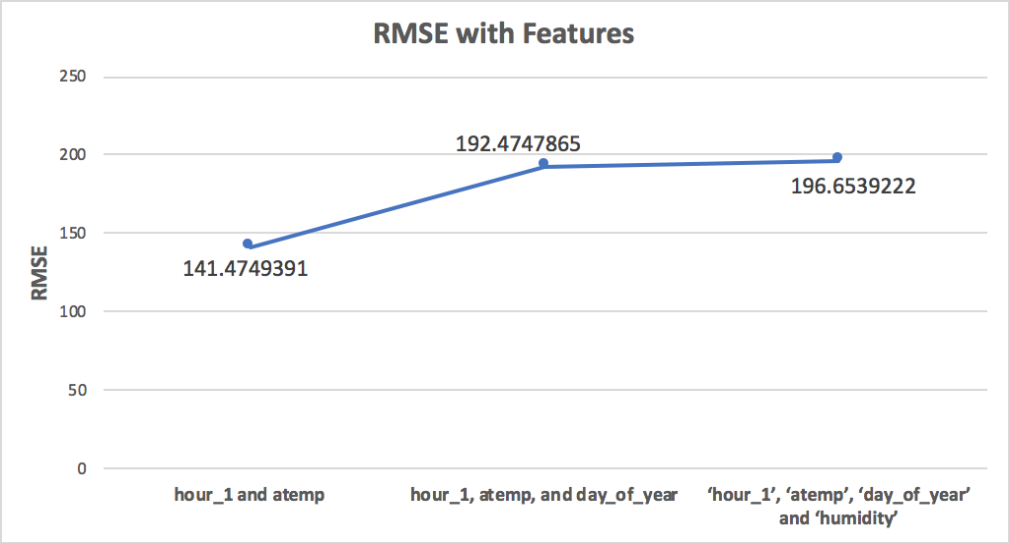
124.03011959584452. The RMSE with PCA and ADA Boost did not change compared to ADA Boost alone.



(DT Figure 3: RMSE of ADA Boost and PCA Cross Validation)

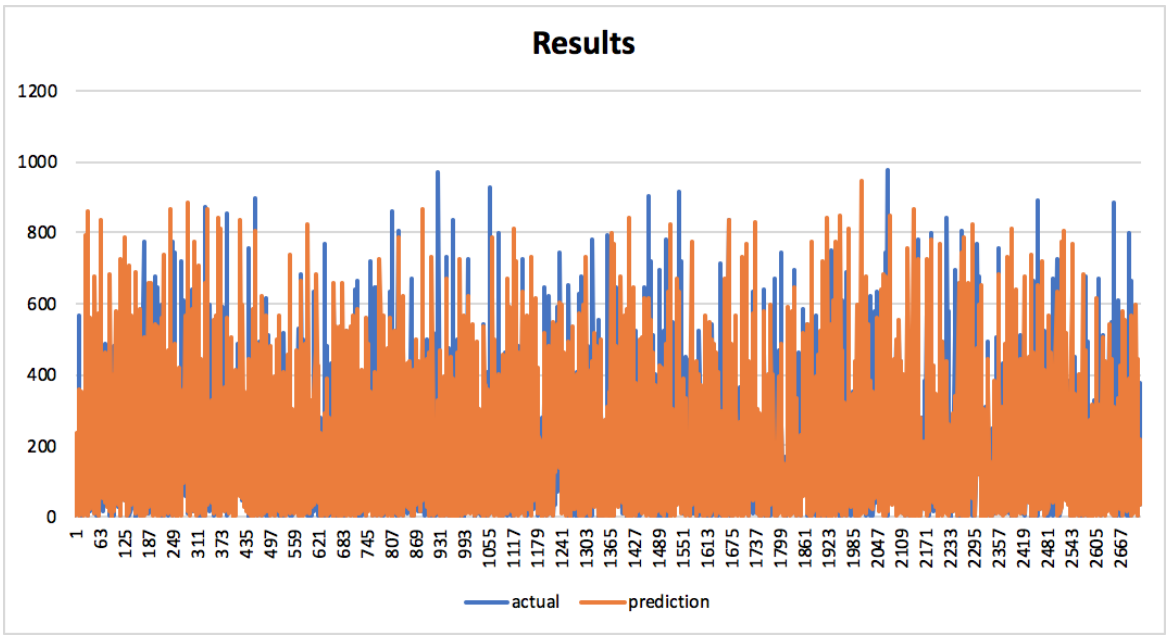
The recommendation for our data set was to use root mean squared logarithmic error (RMSLE). The lower the RMSLE is the better the results are. The RMSLE for Decision Tree was 0.89.

To improve RMSE the most important features were used. The two most important features used were, ‘hour_1’ and ‘atemp’. The calculated RMSE mean with Decision Tree and cross validation was: 141.47493911985347. The best three features were, ‘hour_1’, ‘atemp’ and ‘day_of_year’. The RMSE mean was: 192.4747865330747. The best four features were, ‘hour_1’, ‘atemp’, ‘day_of_year’ and ‘humidity’. The RMSE mean was: 196.65392217510822. The least number of features the better the RMSE mean is.



(DT Figure 4: RMSE of best features)

The actual results compared to the prediction without using cross validation are different than with cross validation. The data was split in the sizes mentioned above and the Decision Tree was fitted with the training data. The actual value is the column 'count' in test data. The results are shown in the graph below:



(DT Figure 5: Actual and Prediction Results of single run

Random Forest Regressor

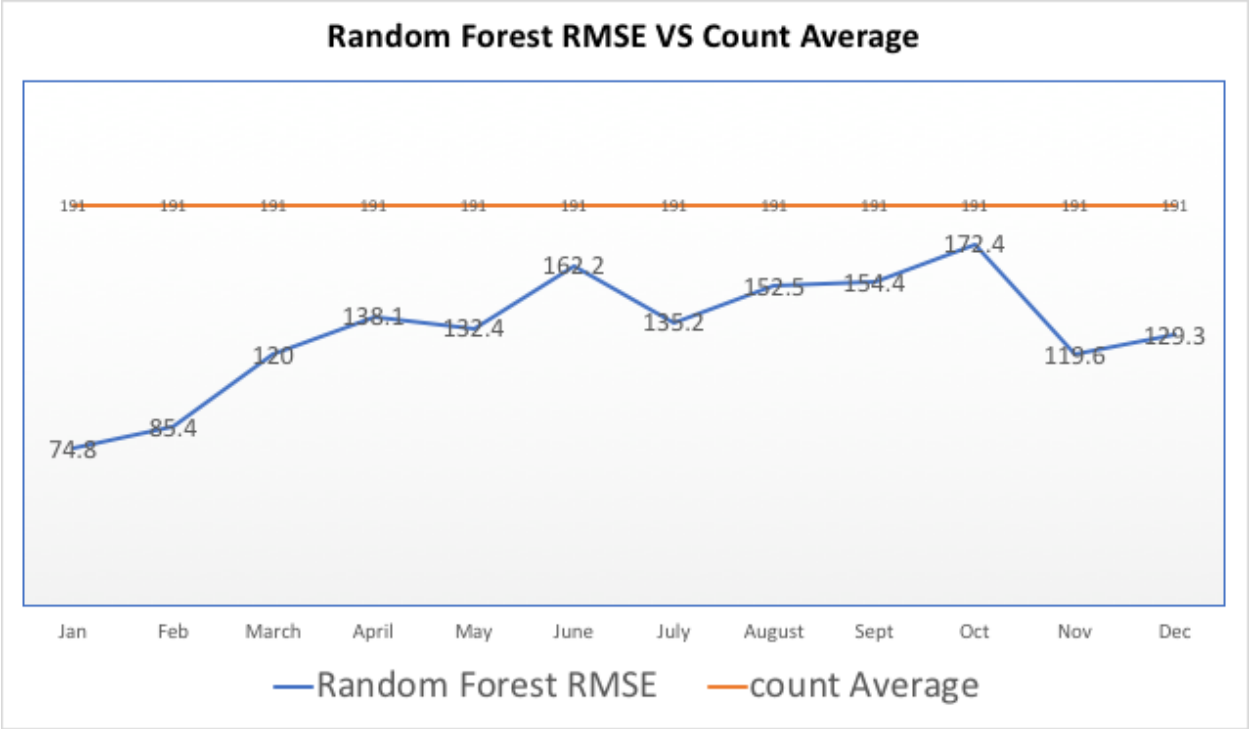
Unlike most other algorithms, 12 models were built from Random Forest Regressor. Each model represented a month of the year. Each month model was trained and made predictions for data occurring in the same month.The January Model, for example, was trained exclusively using bike rental data generated in the month of January. It also only predicted a count for data occurring in January. This was in part done because the average of 'count' varied by large numbers from month to month. This was an attempt to generate a smaller RMSE.The lowest rmse previous models had generated was 124.

The algorithm itself uses bootstrapping and ensemble learning for training and prediction purposes. It creates and uses many weaker models in place of one. Using random sampling to train each model it then relies on some models becoming an expert at specific conditions and then through voting it expects outliers to be corrected by the rest of the trees. In total 5 approaches were attempted below using random forest. Factors that affect the algorithm such as the random state and the number of trees can see be seen below in Table. RF - Condition.

Random Forest Approaches	RF Individual Run	RF RMSE CV Run	RF Feature Reduction	RF ADA CV2	RF PCA ADA
Random State	1	42	42	3	6
N Trees	200	100	100	100	500

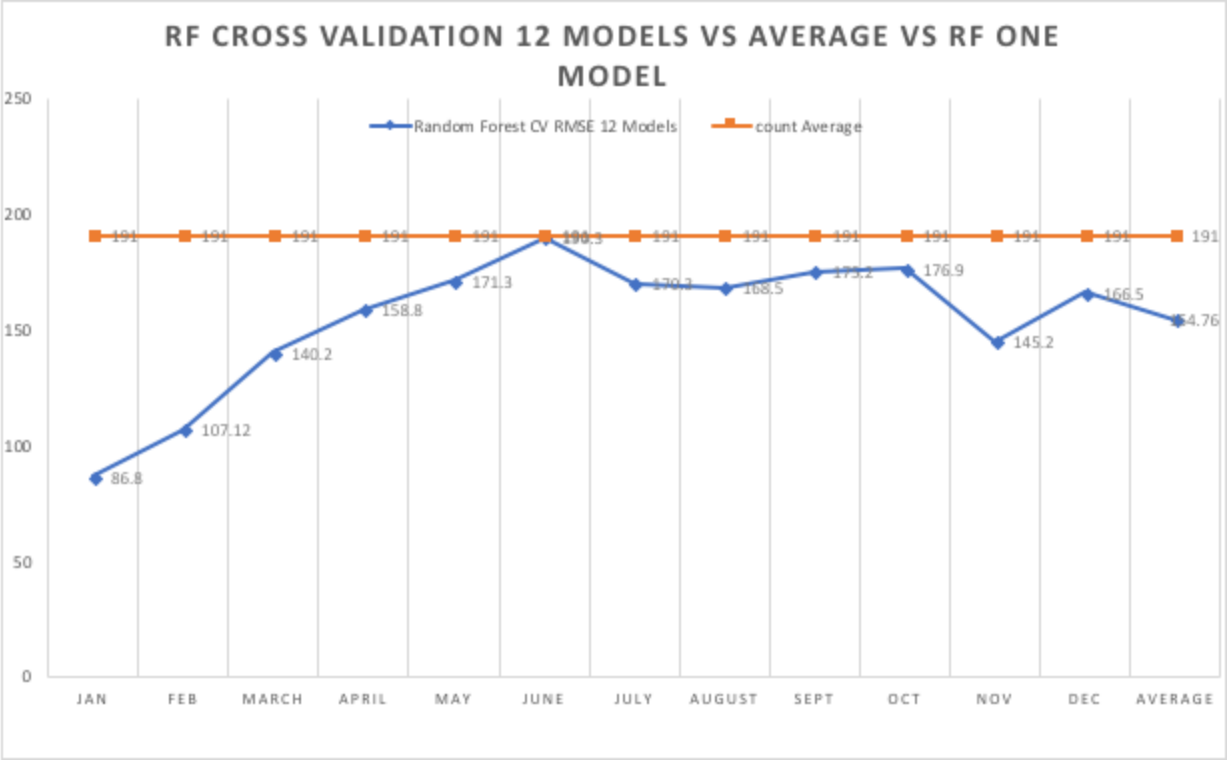
(Table RF - Conditions)

The results of the RF Individual Run can be see below in Visual. RF-Individual Run.



(Visual. RF-Individual Run)

To get more fair results cross validation was performed yielding the results below in Visual. RF-2.



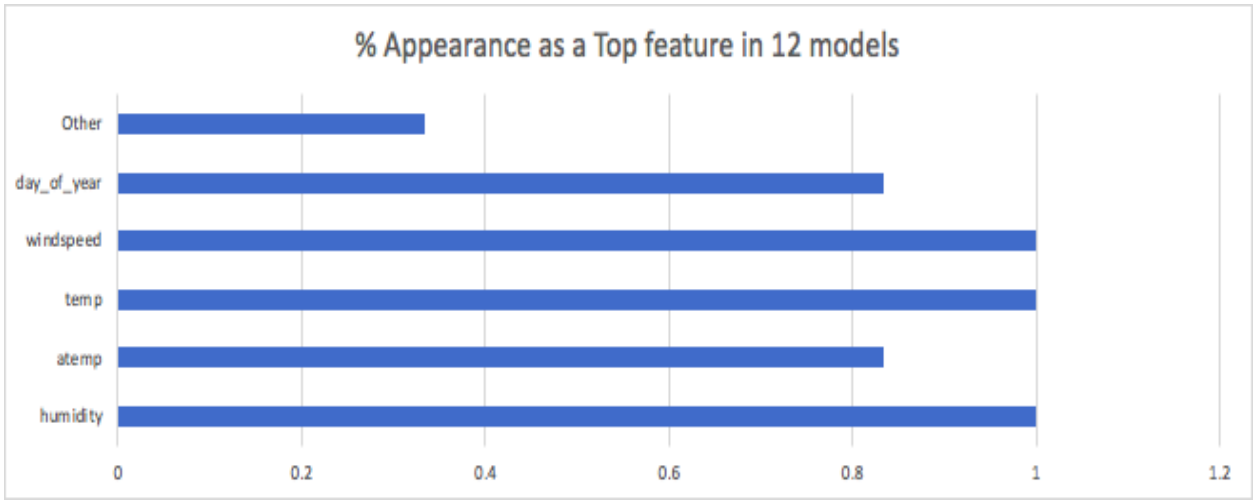
(Visual. RF-2)

From this cross validated model, we saw which features contributed most to each model below in Visual. RF-2 Table. RF Feature Importance.

Month	#1 Feature	#2 Feature	#3 Feature	#4 Feature	#5 Feature
Jan	humidity	atemp	temp	windspeed	year
Feb	humidity	temp	windspeed	atemp	day_of_year
March	temp	atemp	humidity	windspeed	year
April	humidity	windspeed	temp	atemp	day_of_year
May	atempt	humidity	windspeed	day_of_year	temp
June	humidity	windspeed	year	day_of_year	temp
July	temp	humidity	day_of_year	windspeed	atemp
August	temp	humidity	windspeed	day_of_year	year
Sept	humidity	atemp	windspeed	temp	day_of_year
Oct	humidity	windspeed	atemp	temp	day_of_year
Nov	humidity	temp	windspeed	day_of_year	temp
Dec	humidity	temp	windspeed	day_of_year	atemp

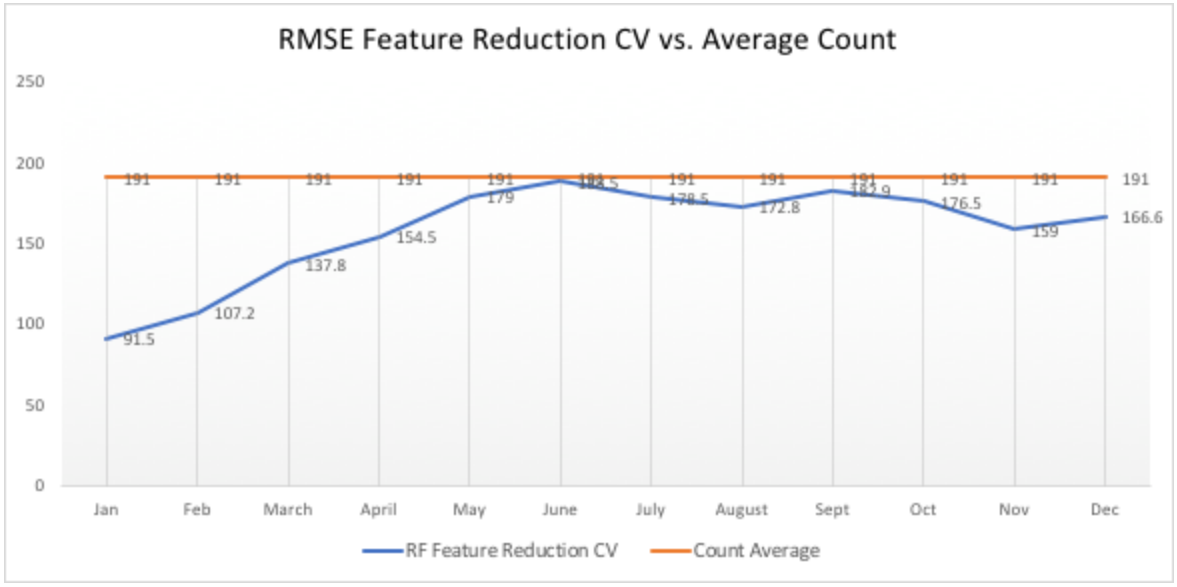
(Table. RF Feature Importance)

From Visual. RF-Feature Importance we see the importance each feature across the all models.



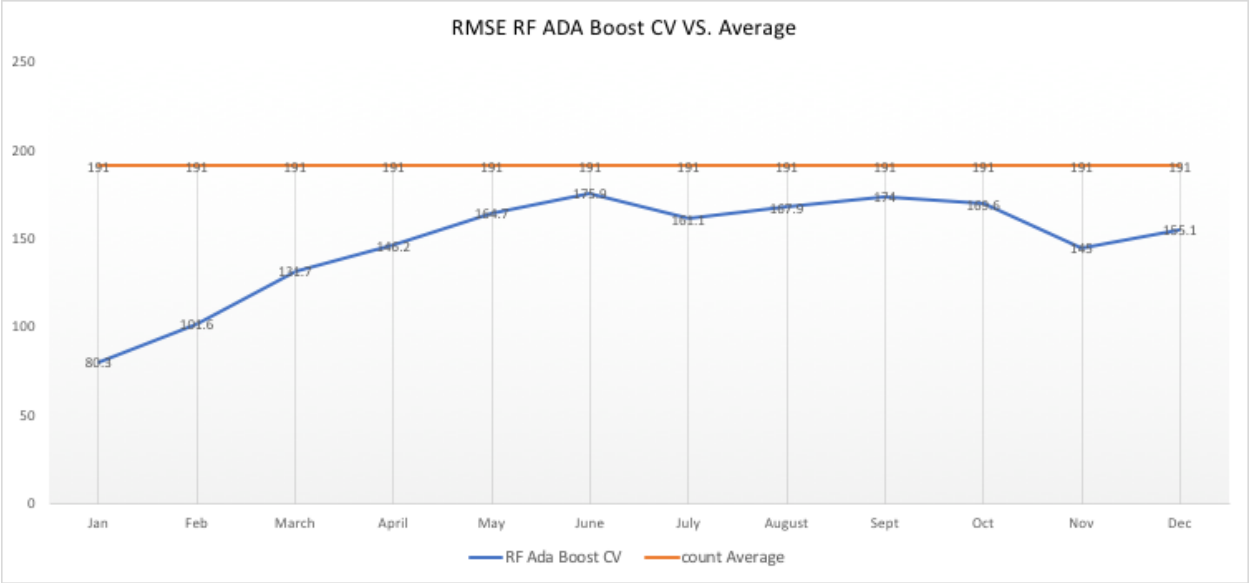
(Visual. RF-Feature Importance)

To try and improve our rmse, manual feature reduction only using the best 5 features. The results can be seen below in Visual. RF-Feature Reduction.



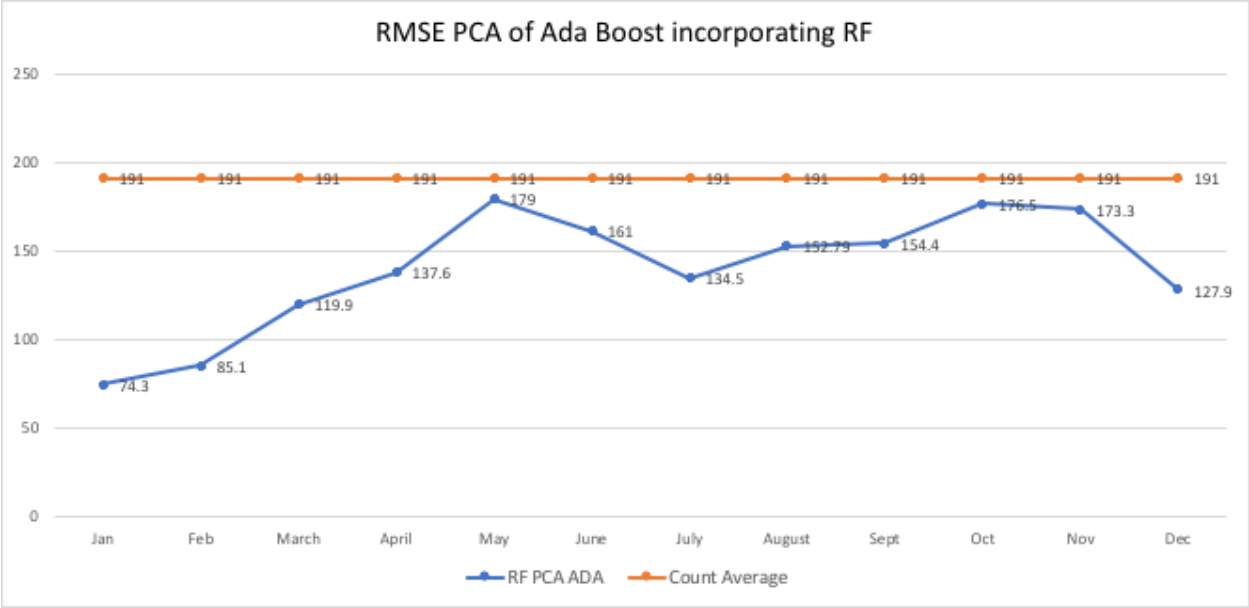
(Visual. RF-Feature Reduction)

To further improve accuracy ADA Boost was applied to Random Forest. Cross validation was again used. The results can be seen below in Visual. RF-ADA.



(Visual. RF-ADA)

PCA was then applied to our ADA Boost model that itself incorporated Random Forest.The results can be seen below in Visual RF-PCA.



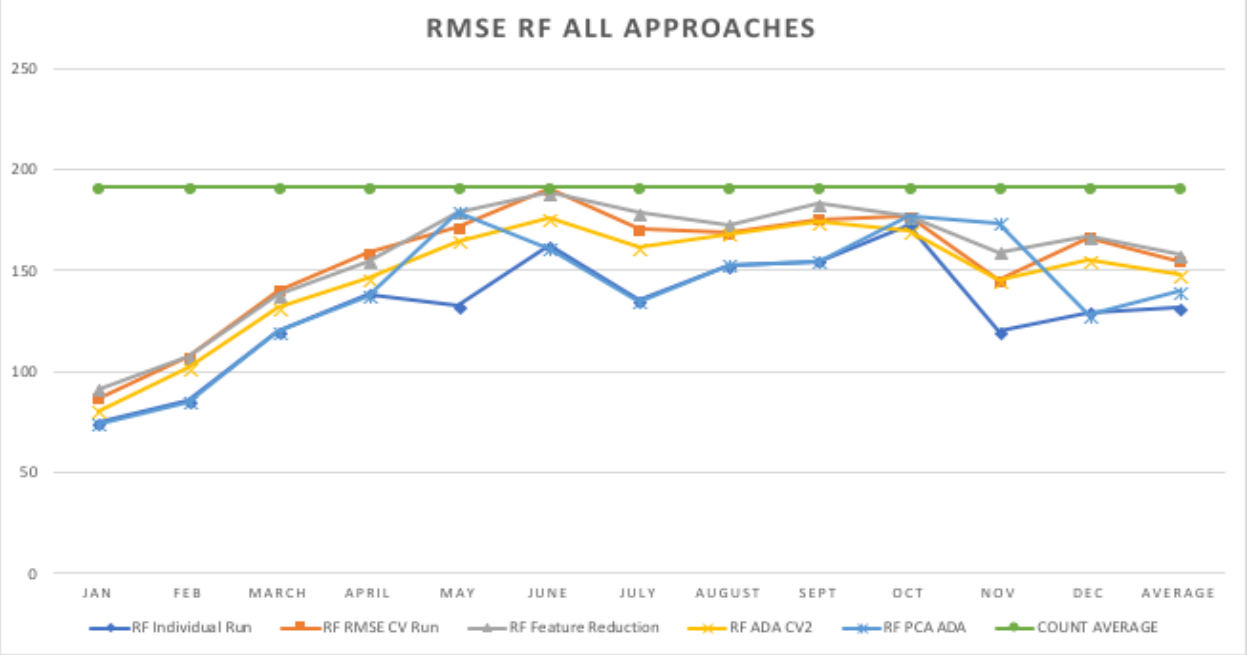
(Visual RF-PCA)

All of these results can be seen in the Table RF-Comparison below:

Months	RF Individual Run	RF RMSE CV Run	RF Feature Reduction	RF ADA CV2	RF PCA ADA	COUNT AVERAGE
Jan	74.8	86.8	91.5	80.3	74.3	191
Feb	85.4	107.12	107.2	101.6	85.1	191
March	120	140.2	137.8	131.7	119.9	191
April	138.1	158.8	154.5	146.2	137.6	191
May	132.4	171.3	179	164.7	179	191
June	162.2	190.3	188.5	175.9	161	191
July	135.2	170.3	178.5	161.1	134.5	191
August	152.5	168.5	172.8	167.9	152.79	191
Sept	154.4	175.2	182.9	174	154.4	191
Oct	172.4	176.9	176.5	169.6	176.5	191
Nov	119.6	145.2	159	145	173.3	191
Dec	129.3	166.5	166.6	155.1	127.9	191
Average	131.3583333	154.76	157.9	147.7583333	139.6908333	191

(Table. RF-Comparison)

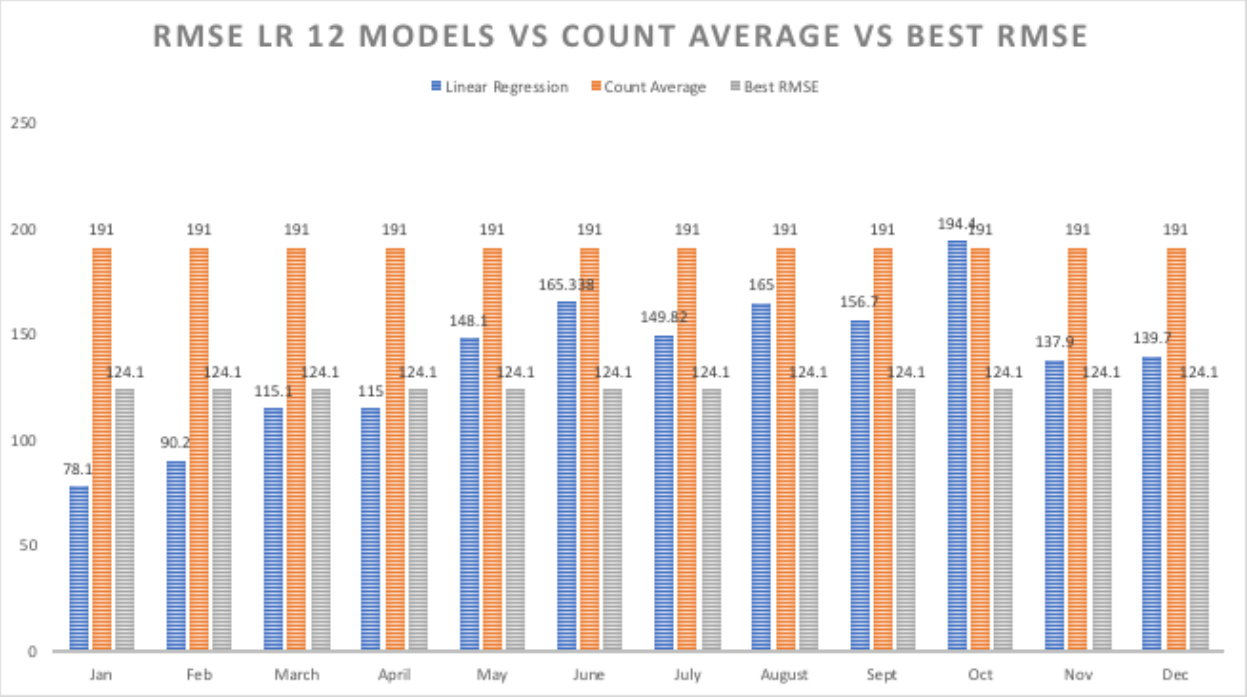
Visual results are below in Visual RF- All Approaches.



(Visual RF- All Approaches)

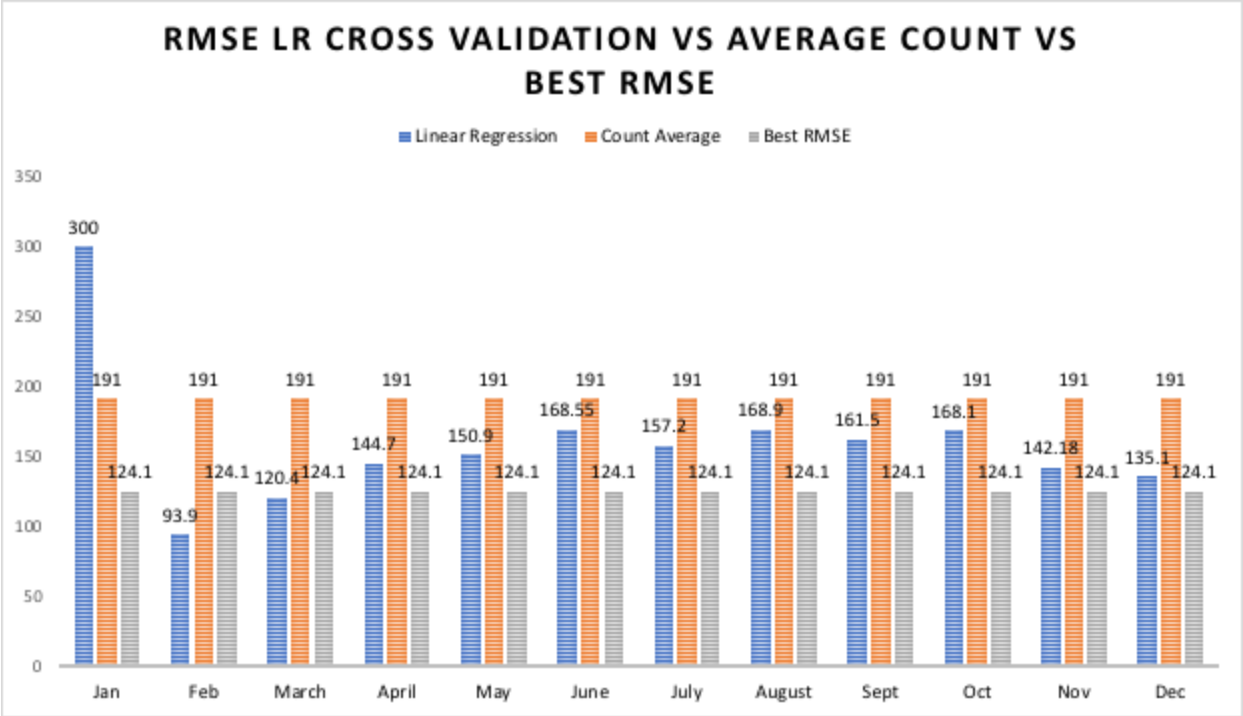
Linear Regressor

Similar to Random Forest, 12 models were built from Linear Regression. Again, Each model represented a month of the year and was trained and made predictions for data occurring in the same month. The reason for pursuing this 12 model approach was to gain a better insight as to the results provided by Random Forest. The results for an single run can be seen below in (Visual LR-Individual Run):



(Visual LR-Individual Run)

To get more fair results cross validation was used yielding the results below in (Visual LR-Cross Validation).



(Visual LR-Cross Validation)

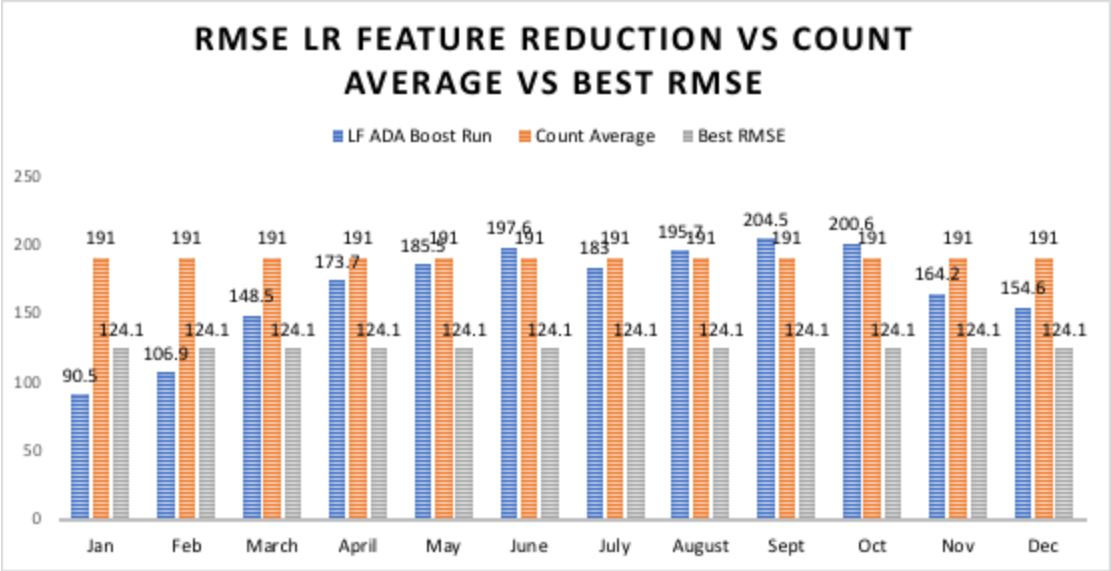
This allowed us to see the most Most Important Features for each model as can be seen in Table.

LR Feature Importance.

Month	#1 Feature	#2 Feature	#3 Feature	#4 Feature	#5 Feature
Jan	Fall	Spring	Summer	Clear	Winter
Feb	Spring	Fall	Winter	Clear	Morining
March	Clear	Early Morning	Morning	Evening	Fall
April	Winter	Morning	EarlyMorning	Summer	Mist
May	Early Morning	Fall	Light	Summer	Winter
June	Spring	Mist	Clear	Summer	Winter
July	Winter	Clear	Fall	EarlyMorning	Evening
August	Early Morning	Mist	Light	Morning	Clear
Sept	Early Morning	Mist	Light	Morning	Clear
Oct	Summer	Morning	Light	Clear	Early Morning
Nov	Spring	Winter	Light	Clear	Morning
Dec	clear	mist	Spring	EarlyMorning	Morining

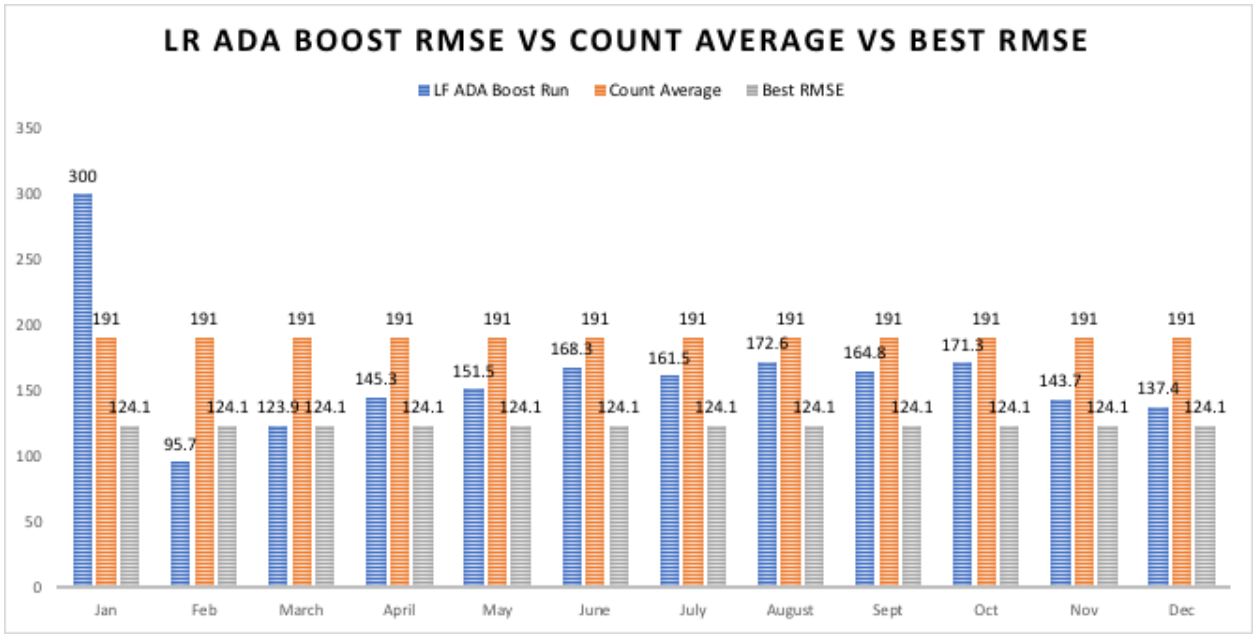
(Table. LR Feature Importance)

Feature reduction was applied and only the features in Table. LR Feature Importance were used for each model. The results can be seen below in Visual LR-Feature Reduction.



(Visual LR-Feature Reduction)

ADA boost was then applied to our model. Results can be seen below in Visual LR-Individual Run.

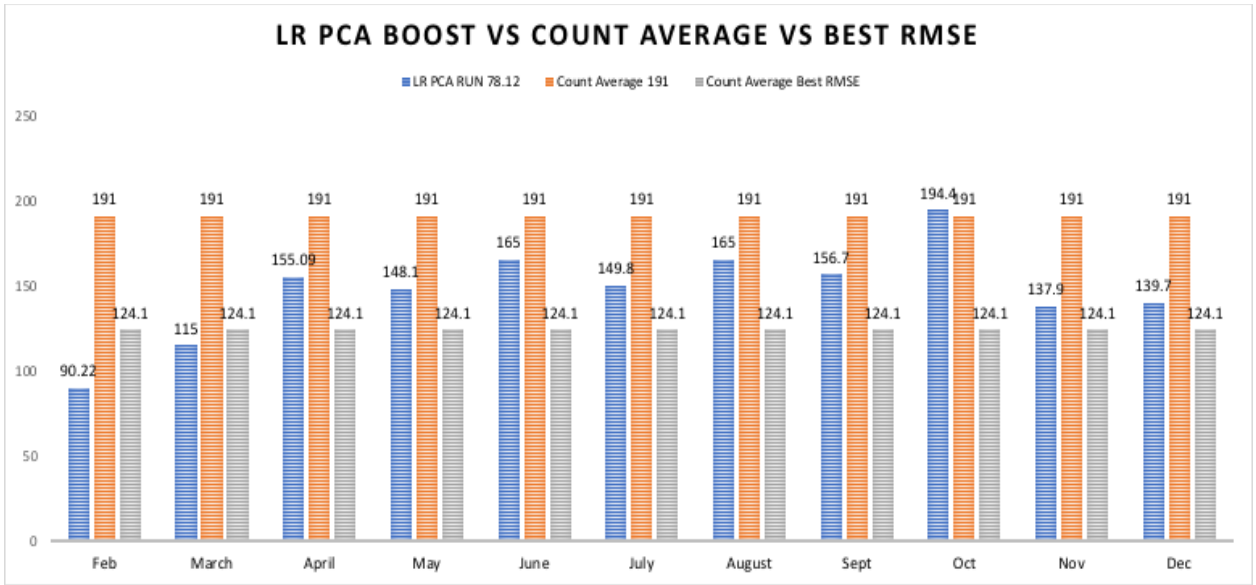


(Visual LR-Individual Run)

*Please note 300 is a dummy value for the January for a really large number revealed at Table

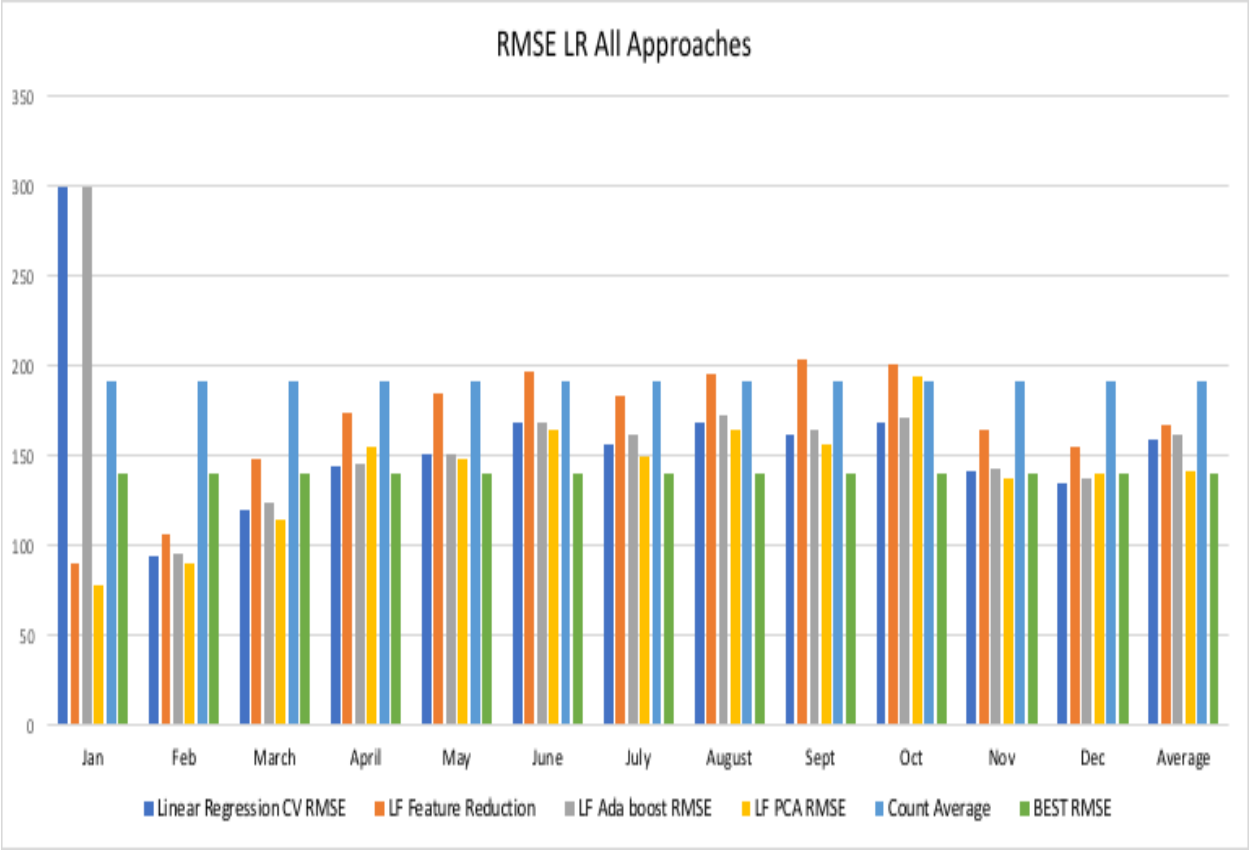
LR- All Approaches

Next we reduced the dimensionality of the dataset by applying PCA. The results can be seen below in (Visual LR-PCA).



(Visual LR-PCA)

This can all be summarized in Visual LR-All Approaches and Table LR- All Approaches below.



(Visual LR-All Approaches)

*Please note 300 is a dummy value for the January for a really large number revealed at Table LR- All Approaches

These numbers can also be seen in Table LR- All Approaches

Linear Regression	Linear Regression RMSE	Linear Regression CV RMSE	LF Feature Reduction	LF Ada boost RMSE	LF PCA RMSE	Count Average	BEST RMSE
Jan	78.1	1.31822E+12	90.5	6.88051E+13	78.12	191	141
Feb	90.2	93.9	106.9	95.7	90.22	191	141
March	115.1	120.4	148.5	123.9	115	191	141
April	115	144.7	173.7	145.3	155.09	191	141
May	148.1	150.9	185.5	151.5	148.1	191	141
June	165.338	168.55	197.6	168.3	165	191	141
July	149.82	157.2	183	161.5	149.8	191	141
August	165	168.9	195.7	172.6	165	191	141
Sept	156.7	161.5	204.5	164.8	156.7	191	141
Oct	194.4	168.1	200.6	171.3	194.4	191	141
Nov	137.9	142.18	164.2	143.7	137.9	191	141
Dec	139.7	135.1	154.6	137.4	139.7	191	141
Average	137.9465	1.09852E+11	167.1083333	5.73376E+12	141.2525	191	141

(Table LR- Comparison)

IV. Analysis

Individual Analysis

KNN Regression & XGBoost:

KNN Regression proved to be a good algorithm since it had one of the lowest RMSE values found among all algorithms which was 124 initially. Incorporating cross validation gave a clearer view of the accuracy of KNN Regressor with a mean RMSE of 123 and a range from 120 - 130. KNN uses feature similarities to predict values. When a new data point was being assigned a value, it closely resembled the points in the training set that were closest to it. Which is how this algorithm makes its predictions.

PCA paired with ADA Boost resulted in higher rmse values. This is because ADA Boost is best used to boost the performance of decision trees on binary classification problems, not for boosting KNN Regressor.

XGBoost also works well with decision tree, however it has many uses and is known for winning Kaggle competitions. XGBoost was not paired with PCA. Perhaps ADA Boost would have had a better outcome if it was not paired with PCA.

Decision Tree Regression:

Decision Tree Figure 1. Figure 1 includes the cross validation of decision tree. In the ten partitions created the best partition with the lowest RMSE value is partition 8 with the value of 151. It is lower than the RMSE mean. The worst RMSE value in the partitions are one, five, and ten. The RMSE values are 160 for 1, 5, and 10 partitions. The RMSE in this case is higher than the RMSE mean, which is 155. In between partitions one and five, the RMSE values decrease than increase at three. From partitions five to ten, the same happens. It decreases until eight, the lowest RMSE value. Then increases to 160 at partition ten.

Decision Tree Figure 2. Figure 2 is cross validation of ADA Boost. In the ten partitions, partition three provides the best RMSE value, 116. The RMSE for partition three is lower than the mean RMSE, 124. The worst RMSE value is from partition two, 128. The RMSE for partition two is higher than the RMSE mean. From partition one to three, the biggest change takes place. The RMSE value increases from one to two. Then rapidly decreases to the lowest value, 116. The rest of partitions increase and decrease constantly with no big change. The RMSE values decreased compared to Decision Tree

RMSE values. The RMSE mean also decreased. ADA Boost performed better than Decision Tree.

Decision Tree Figure 3. Figure 3 includes cross validation with both ADA Boost and PCA. The RMSE values did not change when ADA Boost was used alone. The RMSE mean remained the same. ADA Boost performed the same with PCA included.

Decision Tree Figure 4. Figure 4 includes a comparison of the features that affected the results the most. The best two features for Decision Tree were 'hour_1' and 'atemp', whose RMSE value together is 141. The RMSE for the two features is lower than when 20 features were used, 155. The best three features were 'hour_1', 'atemp' and 'day_of_year'. The RMSE value was 192. The RMSE value for the three features is higher than using 20 features. It is also higher than using two features. The best four features were 'hour_1', 'atemp', 'day_of_year' and 'humidity' whose RMSE value was 196. The value for four features is higher than using 20 features, two or three features. As the number of features increases, the RMSE value increases. However, when 20 features are used the RMSE is lower.

Decision Tree Figure 5. Figure 5 is the results from Decision Tree predictions compared to actual values in the test data. The test data column used is the 'count' column. They are from a single run, without cross validation. The graph shows that many times the prediction is higher than the actual value. Other times, the prediction is lower than the accrual value. There are times when the predicted value is close to or the same as the actual value.

Random Forest Regression:

Visual. RF-Individual Run provided very mixed results. 4 out of the 12 models had a lower rmse than the lowest rmse of 124 that was generated by all previous algorithms. The lowest RMSE for Random Forest being in January at 74.8 almost cutting the RMSE in half. As for the other 8 algorithms all were within a 50 rmse margin. The models were either under 124 or close to it.

Cross validation gave us a better picture, as can be seen in Visual. RF-2. This time only 2 models were below the 124 and the rest were within a 36 rmse margin. Still the results of Table. RF-Comparison clearly show the rmse generally increased. In fact comparing the average from the individual and cross validation run we see a jump in rmse from 131 to 154 a nearly 23 rmse jump. Still the rmse for January and February were considerably lower than the 124 threshold.

This is interesting because across the board the majority of features that contribute the most to each model are almost all the same as can be seen in Table. RF Feature Importance and Table. RF Feature Importance.

In Visual. RF-Feature Reduction, we note that we see that reducing the number of features did not improve our rmse and infant raised it by 3 units. Only 2 models were below the 124 threshold the rest having a greater rmse as can be seen in Table. RF-Comparison.

Visual. RF-ADA showed us an average rmse from all 12 models of 147 . Applying ADA boost lowered the rmse to 147 10 units below the Cross Validation run. 2 models were below 124 and 10 above it as can be seen in Table. RF-Comparison.

Applying PCA to our model lowered the rmse even more, to an average of 139 as can be seen in Table. RF-Comparison. Jan, Feb, and March models were lower than the 124 Threshold and the rest above it. This was definitely the best RMSE excluding the individual run.

Linear Regression:

Visual. LR-Individual Run provided very mixed results similar to Visual RF-Individual Run. 4 out the 12 models had a lower rmse than the lowest rmse at 124 by all previous algorithms. Table LR- Comparison shows an average of 137 of all 12 models. The highest RMSE for Oct at 194 and the lowest on January at 78.

Cross validation gave us a better picture, as can be see in Table LR- Comparison. The average rmse jumped to 1.0985 e ^ 11. This is because the January model now has a rmse of 1.31822e^12 as can be seen in Table LR- All Approaches. Aside from that 1 model is below the 124 threshold and the rest above it. Excluding that outlier the average rmse is 146.

Again across the board the majority of features that contribute the most to each model are almost all the same as can be seen in Table. LR Feature Importance.

In Visual. LF-Feature Reduction, we note that January model is now at 90.5. Also 2 models are below the 124 threshold. As Table LR- Comparison shows the average of all 12 models is 167 with the highest rmse being October at 200.6. Reducing the number of features using this approach did not improve our rmse and in fact raised it.

Visual. LR-ADA shows us an average of 5.73336 e^12 from all 12 models of 147 .This is because the January model has a rmse of 6.88051e^13. In Table. LR-Comparison. We notice the Jan model is an outlier and the other 11 models have an average rmse of 148. 2 models are below the 124 threshold and the rest are higher.

Applying PCA to our model lowered the average rmse of all 12 models to 141 as is shown in Table. LR-Comparison. We note the January model has an rmse of 78. This is definitely the best model for Linear Regression excluding the individual run. The best RMSE being on January at 78 and the highest on October at 194.

V. Results

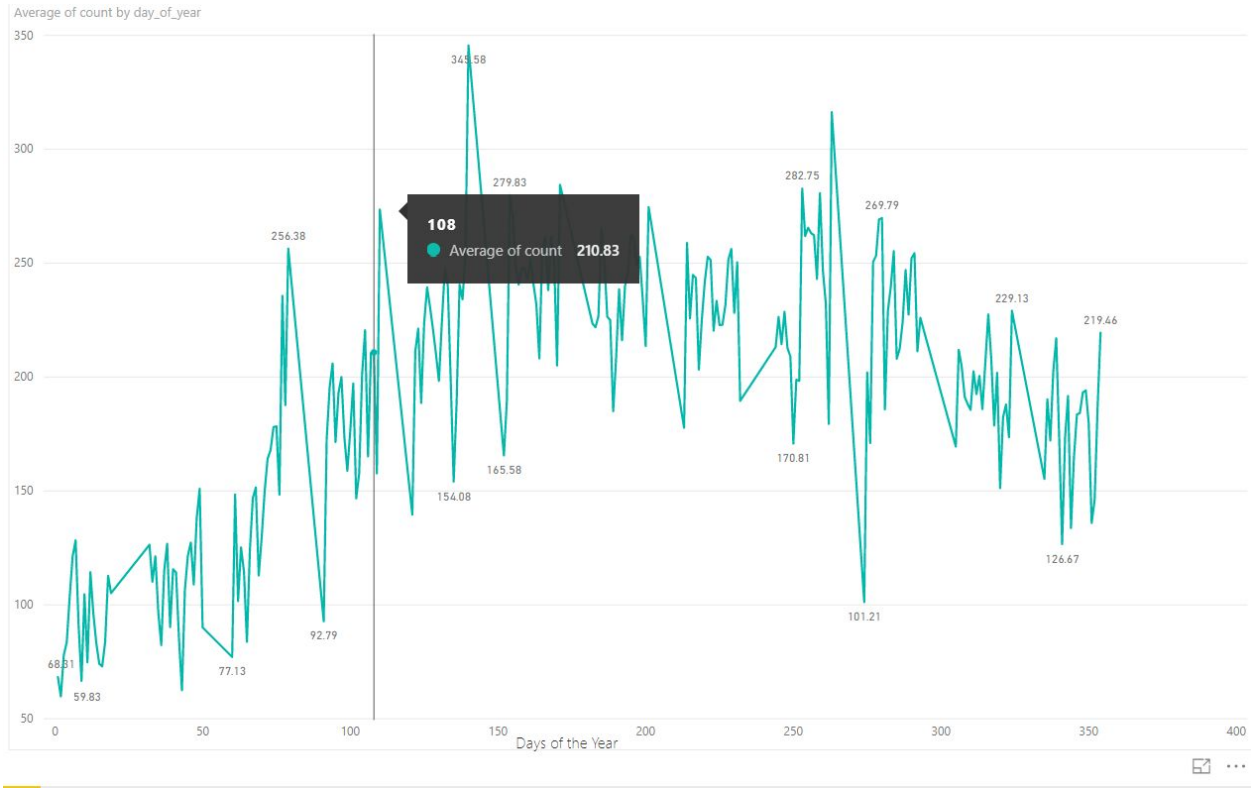
Individual Results

Linear Regression & Random Forest

Both Linear Regression and Random Forest provided mixed results with models that were definitely below the 122 and 124 threshold and at the same time models that were well above that error margin.

Random Forest provided no rmse outliers while Linear Regression did for the January model (using the ADA boost approach and the Cross Validation approach). We cannot know for certain why this is the case. Other approaches in Linear Regression suggest a normal dataset as does the result of Random Forest. Perhaps, corrupt data exist in our dataset and through voting random forest overlooked the bias. It could also be possible that the other runs for Linear Regression did not use this corrupt data in it's training or testing. This would also suggest that our tests to prevent corruption were not sufficient.

Interestingly enough both algorithms had a select small set of features that were most important across all models. This suggest to us that the difference in rmse is probably due to the actual data itself being more scattered in parts than others. Still, the oscillation happening everyday as seen below in Visual Raw Data -1 makes it hard to predict an accurate value.



(Visual. Raw Data - 1)

Since only two years worth of information was provided, it is difficult to know if the data is inconsistency plays a large part in the high rmse score or if it is in fact other factors.

The best rmse from Random Forest was for the January model at 74. The best rmse for Linear Regression was for the January model at 78. Both using the PCA approach. This suggest to us that their was columns that were not providing much information to use but adding

complexity to our model. To get the best rmse the 12 models should be used for some months such as January and February but not all.

KNN Regression & XGB Boost

The value for K-Nearest Neighbor was set to 8 due to it having a low rmse of 124 when testing the rmse with K-values ranging from 1-20. The initial rmsle was 0.78. To perform fair evaluation, 10-fold cross validation was implemented. The results from cross validation revealed the lowest rmse was 120 and the highest was 130, with a mean of 123.

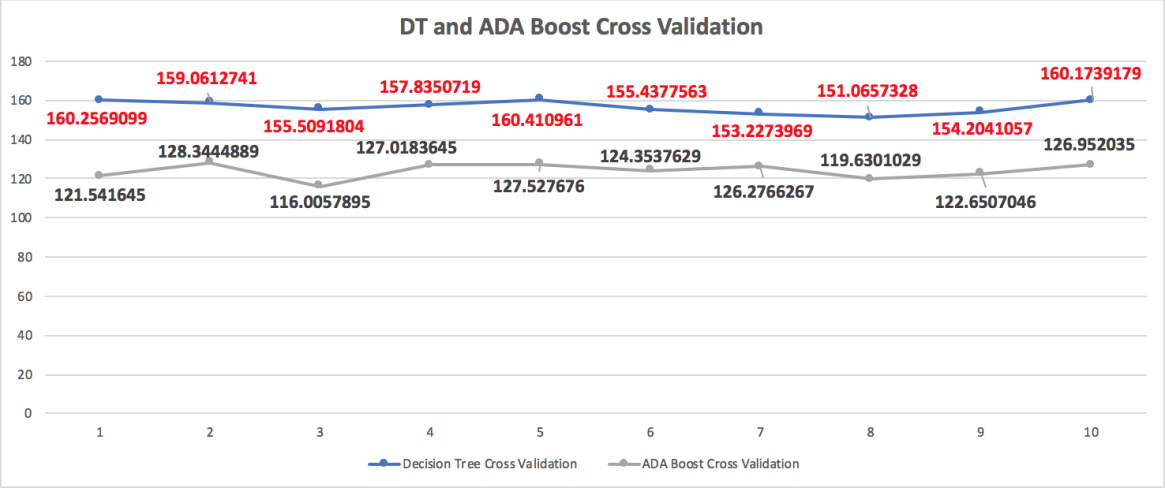
PCA was then performed on the dataset in order to compress the data. Once this was done ADA Boost with cross validation was performed as well. ADA Boost had terrible results, its lowest rmse was 127, which was much higher than the initial rmse of 124. It also had a much larger range of error with lowest rmse value of 127 and highest rmse of 148.

However, the best results came from applying XGBoost. 10-fold cross validation was performed and partition 3 had the lowest rmse of 118. XGBoost had a smaller range of error when compared to ADA Boost. Its lowest rmse was 118 and the highest was 128, with a mean rmse of 122.

Decision Tree

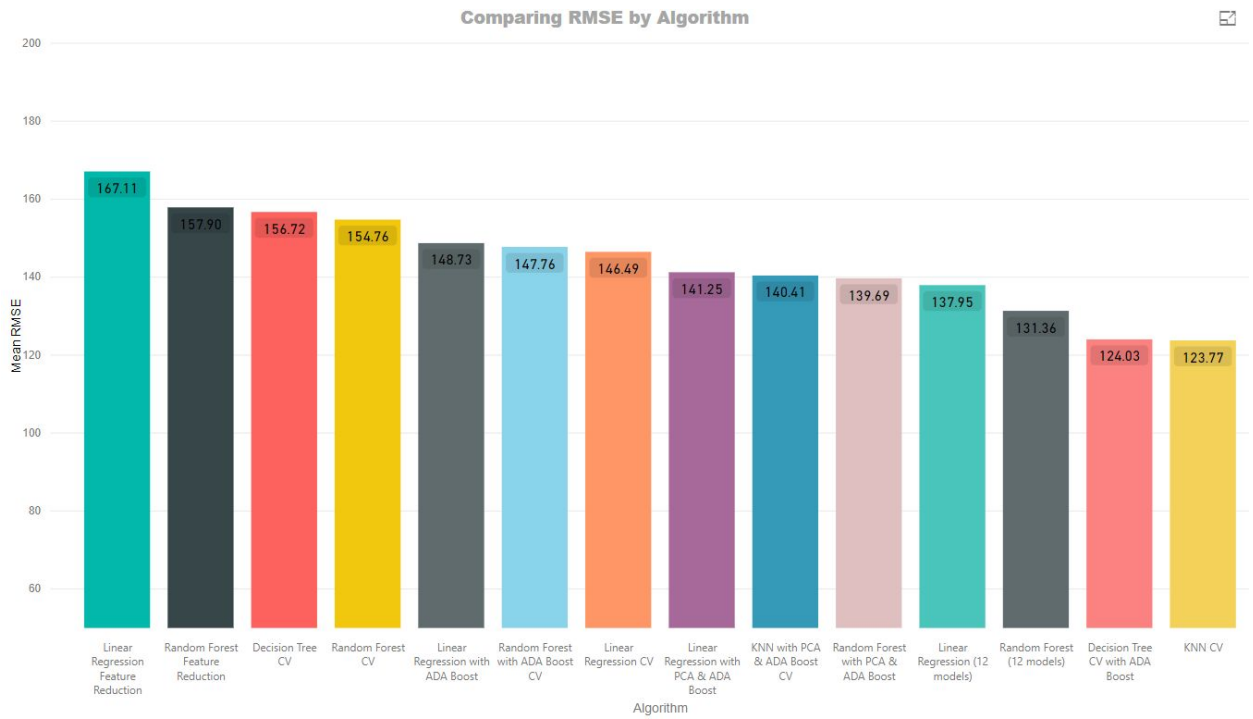
The results for Decision Tree were improved with ADA Boost. There was no change when using PCA. The RMSE values that were found using cross validation improved. However, the partitions that were worst in Decision Tree, partitions 1, 5, and 10, improved with ADA Boost. The partition with the lowest RMSE value in ADA Boost was not the same partition in Decision Tree.

The first half of the partitions (1-5) performed better in ADA Boost. The remaining five performed better in Decision Tree. This might be because ADA Boost is finding the weight prediction and averaging it. Some values that changed are consistent with both algorithms. They were not changed drastically. All RMSE values in ADA Boost algorithm are below the RMSE mean found in Decision Tree. Overall, ADA Boost improved the results of Decision Tree. PCA did not affect the results of ADA Boost.



(DT and ADA Boost Cross Validation RMSE comparison)

Group Results



(GR - Comparing RMSE by Algorithm)

As a whole not all the methods we applied improved our rmse. They certainly did not to the same degree across the board. Different algorithms were best under different conditions. PCA gave Decision Tree, Random Forest and Linear Regression the best RMSE but cross validation gave XGB boost the best rmse.

Even more interesting more than one of the algorithms we applied can arguably be said to have the best model with the best rmse. XGB's 122 rmse(with cross validation) clearly stands out although it is very close to Decision Tree 124 rmse (using PCA). For most of our data they are the best models with the smallest error margin. However, since 12 models were generated with random forest and linear regression a black and white comparison cannot be made. This is because for the months of January and February Random Forest (using PCA) generated an rmse

of 74 and 85. Furthermore, a 115 rmse was generated for March with Linear Regression (using PCA). For the rest of the months, they are either close to or much higher than 122 and 124.

From another perspective if we average the rmse for the 12 month models generated by Random Forest and Linear Regression as can be seen above in GR - Comparing RMSE by Algorithm then KNN (with cross validation) and Decision Tree (with cross validation and ADA Boost) are definitely our best algorithms with an approximate rmse of 124.

VI. Conclusion

In the end we did not manage to generate a ground breaking rmse or even a desirable one. Even PCA and the boosting algorithms only slightly lowered our otherwise large margin of error. Feature reduction only increased our rmse error. We applied all techniques we learned in class as well as techniques we researched and those recommended to us by our peers.

XGBoost with Cross Validation gave us the smallest mean error rmse of 122. Decision Tree Regression (with PCA and Cross Validation) a comparable error rmse of 124. This is not without exception as Random Forest with PCA gave us a rmse of 74 for January and 85 for February. Also Linear Regression for the month of March using PCA had a error rmse of 115. For all other months, they were worse of than the standard 122.

As we only had 2 years of data and only the first 20 days to train and test with, it is uncertain as how much an impact this had on our models. Also, our data was confined to 2 years. Our models would most likely not fare well for future use especially if the number of rentals spike.

It is interesting to note that in the 12 model approach, the most significant contributions to each of the models were coming from mostly the same features but the rmse for some months were as low as 74 and as high as 200 for others.

Perhaps, for future research in this problem we would recommend trying to use unsupervised learning algorithms before using supervised algorithms. Also we would recommend attempting different normalization techniques and different regressive algorithms.

VII. Appendix

References

[1] Kaggle Project Description: https://www.kaggle.com/c/bike-sharing-demand/

[2] Insightsbot Blog:
http://www.insightsbot.com/blog/McTKK/python-one-hot-encoding-with-scikit-learn

[3] Python Documentation:
https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior

[4] Analyticsvidhya Blog:
https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/

[5] Blopig Blog:
https://www.blopig.com/blog/2017/07/using-random-forests-in-python-with-scikit-learn/

[6] Udacity Portfolio: https://career-resource-center.udacity.com/portfolio/data-science-reports

[7] Saedsayad Article: http://www.saedsayad.com/k_nearest_neighbors_reg.htm

[8] Towards data science Article:

https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c

[9] Dezyre Article:

https://www.dezyre.com/data-science-in-python-tutorial/principal-component-analysis-tutorial

[10] Machine Learning mastery article:

https://machinelearningmastery.com/k-fold-cross-validation/

[11] Medium Article:

https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

[12] Oreilly Article:

https://www.oreilly.com/library/view/machine-learning-with/9781785889936/4317943c-9452-4013-99b9-d267a2820b23.xhtml

VIII. Responsibility of Team Members

Team Members & Responsibilities:

- *Emmanuel Cocom*:
    - Emmanuel's responsibilities included applying Random Forest Regression, Linear regression, ADA Boost and PCA to our dataset. He also used an alternative approach in attempt to lower the rmse by creating 12 models(one for each month of the year) instead of 1. He did this for both Random Forest Regression and

Linear regression. He created test functions to fix corrupt data. Completed the Introduction and data sections of the report. Completed Dropping Leakage variables, Extracting Data, Correcting Corrupt Data, Custom Functions, One Hot Encoding, New Features After One Hot Encoding, Columns Discarded, Total Number of Features, Normalizing Data, Tools subsections of the Methods section of the report. Completed Random Forest Regression and Linear Regression of the algorithms subsection of the methods section in the report. He also finished the Results and Analysis for Linear Regression and Random Forest. He also worked on creating the power point for group presentations. He also worked on the conclusion for the group.

- *Kristen Marenco*:
    - Kristen Marenco's responsibilities included applying KNN Regressor, ADA Boost, XGBoost Regressor, and PCA to our dataset. Kristen also did research on normalizing data and tested one alternative approach. She also inserted the code snippet for hourly bins. In addition, she worked on the appendix.. She also completed the ADA Boost and XGBoost subsections for the methods section. She also completed the analysis and results section for ADA boost and XGBoost. She also worked on the graph for the Results section and helped proof read the report. She also worked on the powerpoint presentation.

- *Imelda Flores* :
    - Imelda Flores responsibilities included applying Decision Tree Regressor, ADABoost Regressor, and PCA to our dataset. She also completed the other methods subsection of the methods section of the report.She also worked on the appendix of the report. She also helped proof read the report. She also worked on the decision tree subsection of the methods section as well as the analysis and results of the section of the report for Decision Tree Regression. She also helped creating the presentation slides.