UNIVERSITY OF LINCOLN

UNIVERSITY OF
LINCOLN

Data Science - Item 2

Bike Rental Demand Prediction

Author:
Raymond KIRK

Student Number:
KIR14474219

May 4, 2017

Contents

1 Bike Rental Data

1.1 Data Summary

The data used in this assignment is from Capital Bikeshare, a rental bike company based in Washington DC. It's comprised of data logged every hour between 2011 and 2012, resulting in a total of 17,379 observations across 17 features, seen in figure A.1. Azure ML (Microsoft, 2017) was used to obtain some statistics from the data. The data is consists of 17 features, of which 12 are integers, 4 are floats and 1 is an object as shown in figure 1.1. A more detailed break down is given in figure 1.2, which shows the data is comprised of 10 categorical features and 7 continuous features. Also shown in figure 1.2 is that of the 17 features in the dataset, none of them have any missing values. Table 1.3 describes the data contained within each feature.

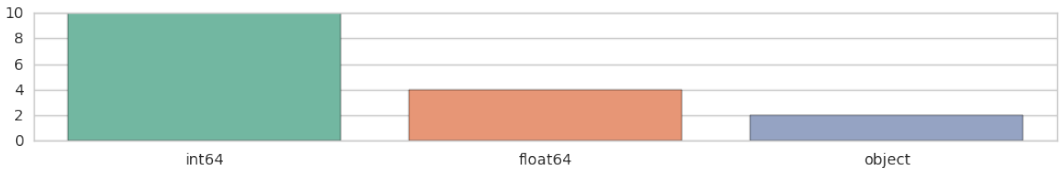

FIGURE 1.1: Bike Rental Data Type Summary

FIGURE 1.2: Bike Rental Data Summary

Feature	Count	Unique Value Count	Missing Value Count	Data Type	
				Statistical	Data
instant	17379	17379	0	Categorical	Integer
dteday	17379	731	0	Categorical	Date Time
season	17379	4	0	Categorical	Integer
yr	17379	2	0	Categorical	Integer
mnth	17379	12	0	Categorical	Integer
hr	17379	24	0	Categorical	Integer
holiday	17379	2	0	Categorical	Integer
weekday	17379	7	0	Categorical	Integer
workingday	17379	2	0	Categorical	Integer
weathersit	17379	4	0	Categorical	Integer
temp	17379	50	0	Continuous	Float
atemp	17379	65	0	Continuous	Float
hum	17379	89	0	Continuous	Float
windspeed	17379	30	0	Continuous	Float
casual	17379	322	0	Continuous	Integer
registered	17379	776	0	Continuous	Integer
cnt	17379	869	0	Continuous	Integer

FIGURE 1.3: Bike Rental Data Descriptions

Feature	Description
instant	Observation ID
dteday	Observation Capture Date
season	Season 1-4 Spring-Winter
yr	Year 0-1 2011-2012
mnth	Month 1-12 Jan-Dec
hr	Hour 0-23
holiday	Boolean for if Date is a Holiday
weekday	Boolean for if date is a weekday
workingday	Boolean for Weekday and not Holiday
weathersit	Weather Condition 1-4 Best-Worst
temp	Temperature in Celsius 0-1
atemp	Real Feel Temperature 0-1
hum	Humidity 0-1
windspeed	Wind speed 0-0.8507
casual	Number of casual rental users
registered	Number of registered rental users
cnt	Rental bike total

1.2 Preprocessing

As aforementioned, no missing values were found in the feature observations so no preprocessing had to be employed to substitute for these values. However the "instant" feature is fully unique and is descriptive of the row number consequently providing no useful information to the dataset making it redundant. For this reason the "index" was one of the columns that was dropped.

1.2.1 Feature Engineering

The "dteday" feature contains the capture date time for each observation. In the data frame however the year, month and hour are already accounted for so most of the information contained in the date time observation is useless. The only feature it contains not already in the dataset is the day of the month observation. Due to this a new feature, "day" was extracted from it and placed in the dataset, subsequently dropping the "dteday" feature. The "seasons", "weekday", "mnth" and "yr" features were also modified in-place at this stage to make the data they contain more descriptive. The changes made to there data was to replace the numeric categorisation that symbolised a value with the respective values, for example "0" in "yr" was replaced with the string "2011". To make the code base and plots more readable in this assignment some of the features were also renamed. All of the changes that occurred to the features can be seen in figure A.2. After these changes the meta data of the features listed as categorical in figure 1.2 were changed to be categorical in the Azure ML Studio. The statistics shown in figure A.3 how that "windspeed" ranged between values 0 and 0.8507, to maintain consistency with the other feature points it was min-max normalised between zero and one.

1.2.2 Removing Outliers

The statistics in figure A.3 showed that the mean for some of the features varied greatly from the minimum and maximum values of the respective features. Due to this box plots were generated with python code in Azure Studio to check for outliers in the data. Figure 1.4 shows the plotted numeric features and how many outliers they have. This erroneous data needs to be removed which was achieved by limiting the data after the 96th percentile to the maximum and data below the 1st percentile to its minimum. Figure 1.5 shows the results after this process.

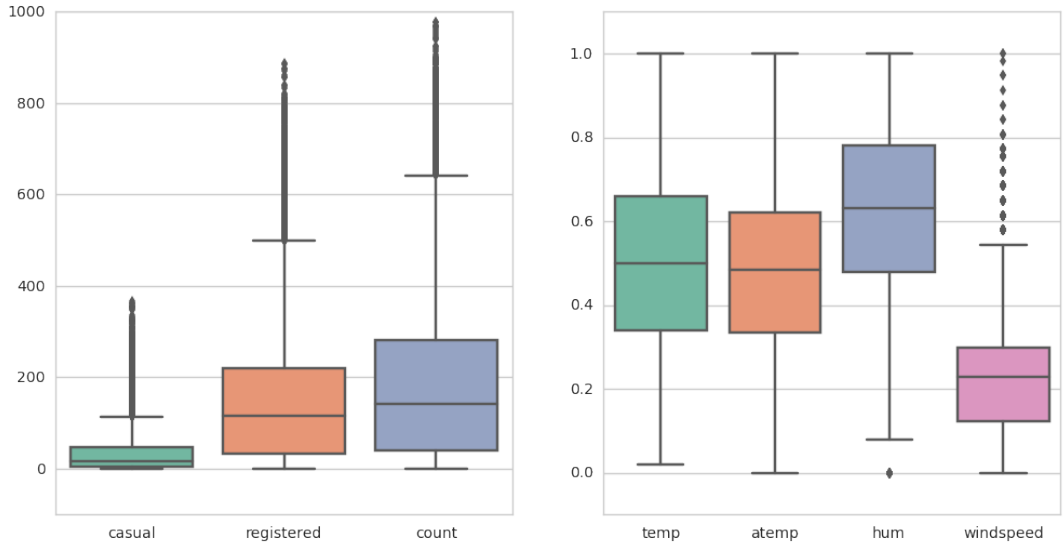

FIGURE 1.4: Feature Box Plots Showing Outliers

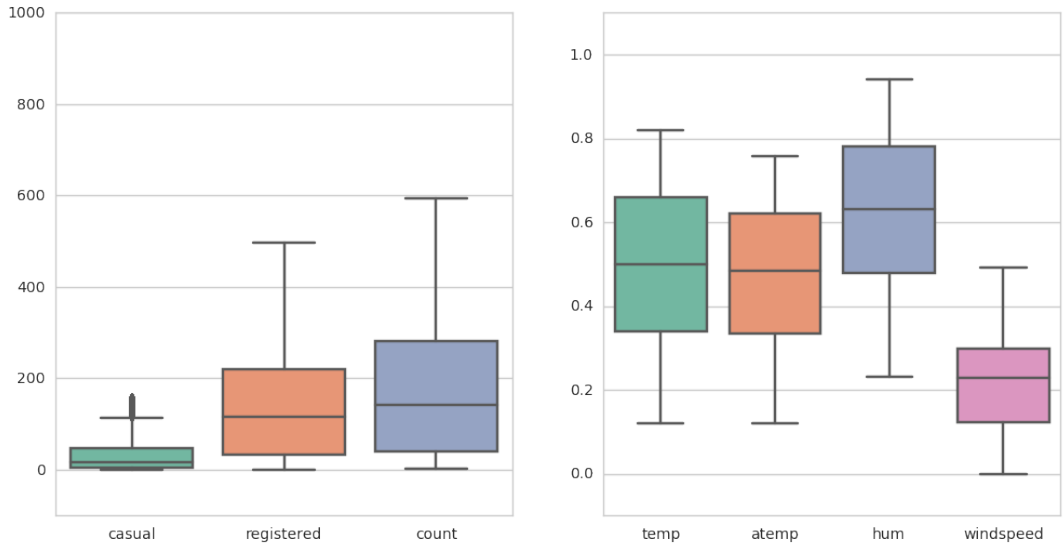

FIGURE 1.5: Clipped Feature Peaks and Sub-Peaks at 1 and 96th Percentile Respectively

1.3 Visualisation

Each variable would ideally represent a unique feature of the data. To analyse the relationships between the features many techniques can be used, one of which is a correlation matrix. Figure 1.6 shows the plotted correlation matrix of all the numeric features. Orange and blue denote high and low correlating features respectively. From this plot it is evident that the "windspeed" feature isn't going to be useful due to the low correlation value it has with the dependant variable. All of the other features share some correlation. It also becomes apparent that the "atemp" and "temp" variables have a very high similarity to each other and will need to be dropped from the dataset to produce better models and minimise collinearity between the features correlating with the dependant variable. From the description of the data and the correlation matrix it's also clear that the "casual" and "registered" features will have to be dropped when the models are trained due to the high collinearity with the dependant variable subsequently effecting the forecast "count".

FIGURE 1.6: Correlation Matrix of the dataset

The categorical variables were plotted against the count variable in order to determine variables that could be used in prediction. The first on that was plotted was "weathersit", which describes the condition of the weather on a best to worst scale. So naturally the intuition would be that the count variable would decrease over the spread of the "weathersit" variable. As see in figure 1.7 this hypothesis was true and shows that progressively worse weather impacts the number of bike rentals. This was then referenced against the season categorical variable to see if there was a relationship between time of year and decline due to weather. The plot of this can be seen in figure A.4 and they seem to be independent with only slight variance due to weather in each season.

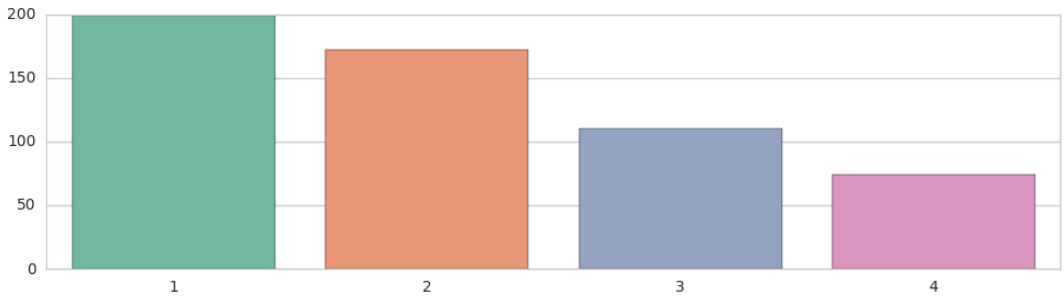

FIGURE 1.7: Count against Weather Situation

Figure 1.8 shows the number of bike rentals per month which infers that the warmer months of the year being May to October in Washington DC corresponds to higher rentals within the company. This is further validated with the seasons plot shown in figure 1.9.

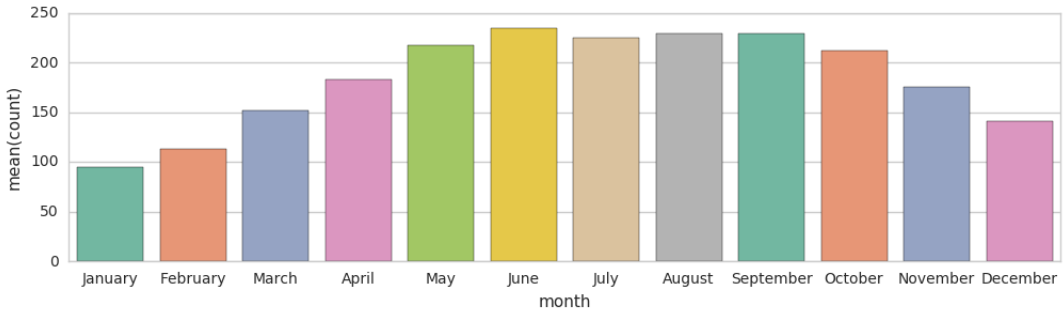

FIGURE 1.8: Count against Month

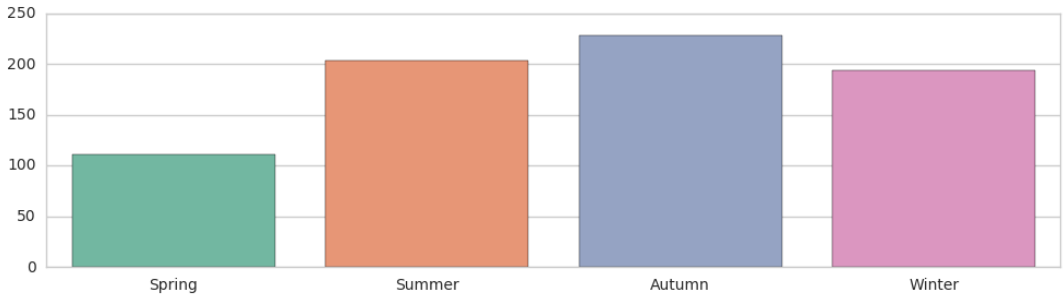

FIGURE 1.9: Count against Seasons

Figure 1.10 shows the distribution of bike rentals over the hours of the day. The graph clearly shows a trend between the hours of the day and the number of bike rentals. The bars have a distribution first peaking at 8am and then again at 5pm. To further inspect this trend the hour distribution is split by weekdays and by type of rental to determine the justification for this pattern. Figure 1.11 shows clearly that these peaks correspond to the work week which would imply the business was from some form or repetitive cycle such that of a commuter using the company to facilitate their travel to and from employment each weekday. The trend of the data split of the weekends implies that the weekend midday is more popular from an assumption that the weather is nicer, this is validated by the more normal distribution of figure 1.12 (right) showing rentals for hours in the day by weather situation.

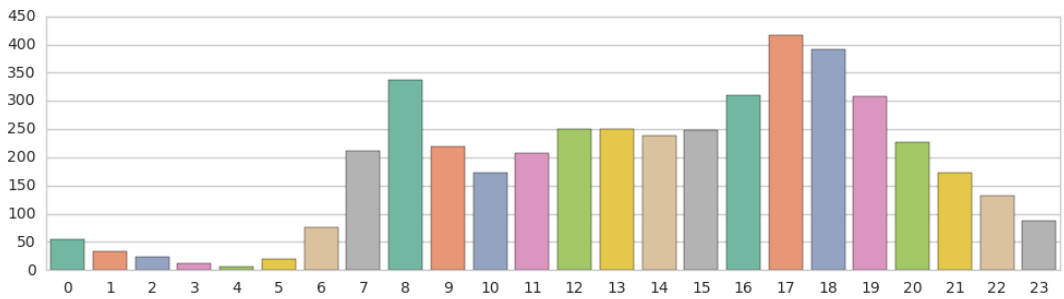

FIGURE 1.10: Count against Hour

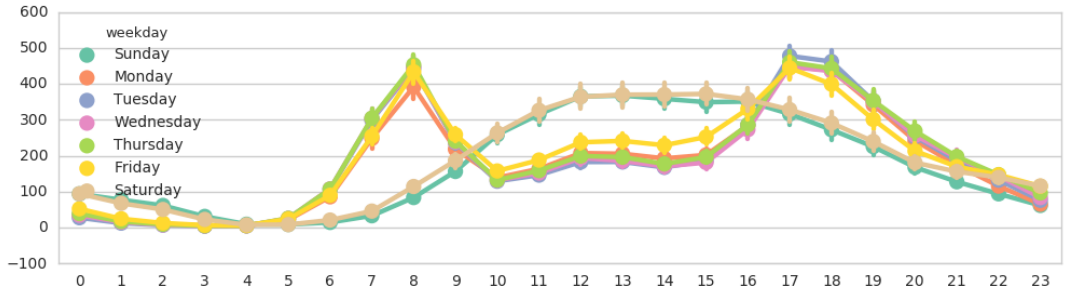

FIGURE 1.11: Count against Hours Split by Weekday

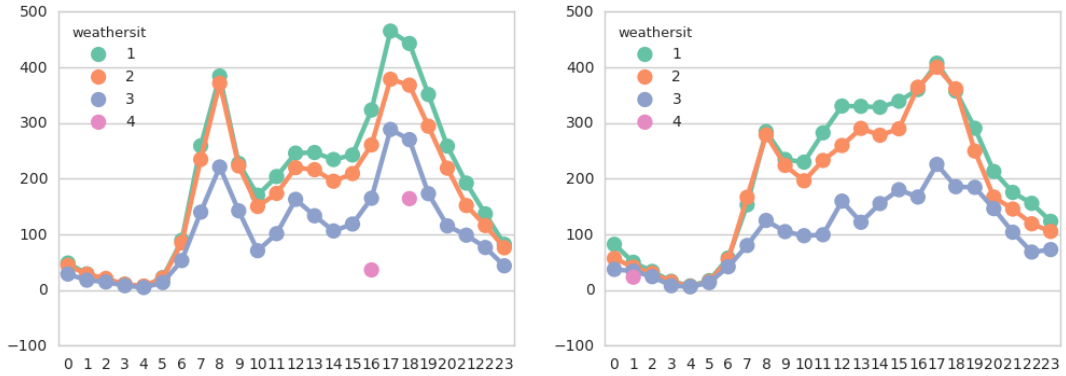

FIGURE 1.12: Count against Hours Split by Weather on Weekdays (Left) Weekends (Right)

1.3.1 Data Distribution

Normal distribution of data is ideal for most machine learning models or statistical tests (Hill *et al.*, 2006). However on evaluation of the data it's apparent that the distribution of the dependant variable is skewed to the left as seen in figure 1.13. In the figure two methods of transforming the data to a more normal distribution were executed; "sqrt" and "log" of the data respectively. It's clear from the figure that the "log" (right) of the data normalised the distribution more so that the original and "sqrt" function. The full code for the data summary, preprocessing and visualisation can be seen in figure A.5.

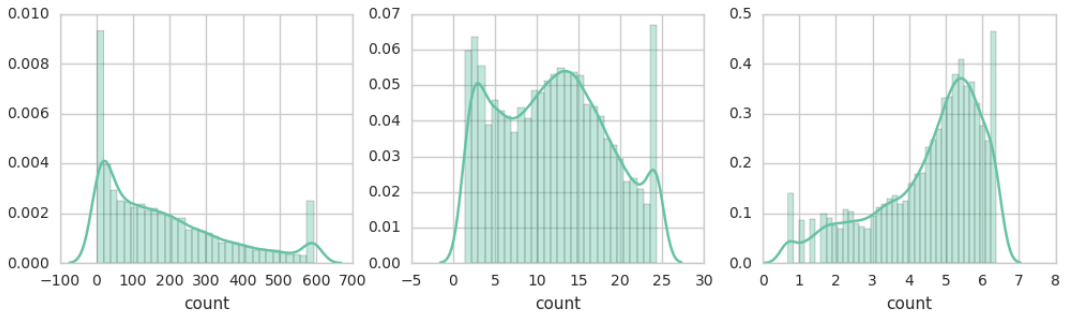

FIGURE 1.13: Distribution of the data (left) Sqrt distribution (middle) Log distribution (right)

2 Algorithm Comparison

2.1 Mean Absolute Error

Estimating the effectiveness and accuracy of trained models can be done multiple ways. A simple and straightforward way that is commonly used, is to calculate the error of the predicted values (Hyndman and Koehler, 2006). The metric that is used extensively in model evaluation throughout this assignment is MAE (Mean Absolute Error). MAE is a statistical quantity that outlines the distance between predicted values of the trained model and recorded data. One benefit of using this statistical quantity is that it's a scale dependant accuracy measure, which means the MAE value is bounded to the same scale as the original data. However this means the MAE cannot be used to evaluate different time series with different scales. MAE is calculated as stated in the equation 2.1 where n is the length of predicted values vector y and x is the external observed data.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{2.1}$$

2.2 Linear Regression

A regression model is used to predict the "count" variable since the statistical data type is continuous. The data was split into two sets for training and testing the regression models. The data is split into 75% training and 25% testing. The training set is used when training the model and the testing set is used to validate the predictions the trained model will forecast. Figure 2.1 shows the result of the linear regression models predictions against the original test data.

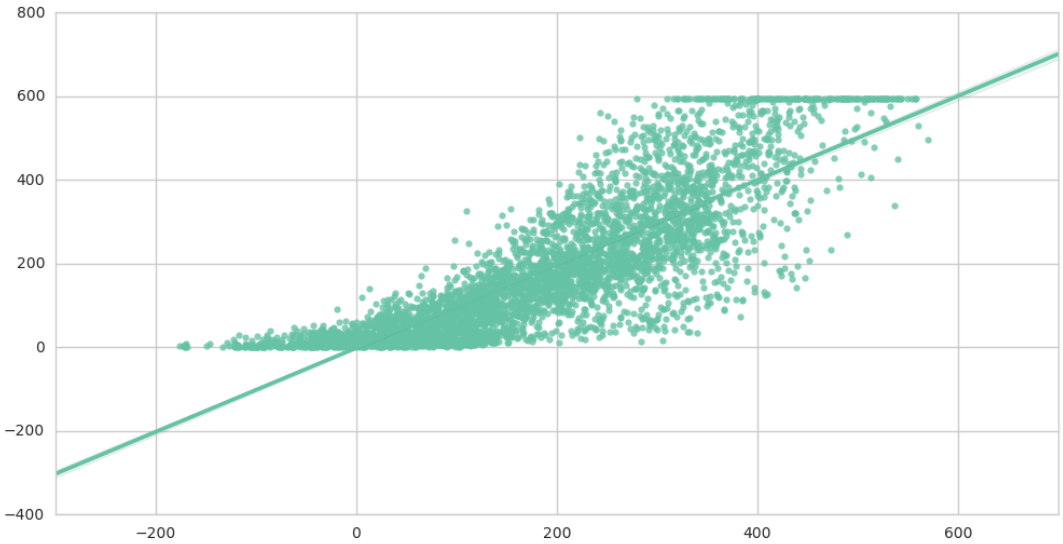

FIGURE 2.1: Linear Regression Model Scatter Plot

The disparity of the data points plotted in figure 2.1 show that the linear regression model can be used to predict the dependant variables within a margin of error from the plotted regression line. No feature selection processes were utilised at this stage to determine the features the model would be trained on. The selected features were purely based off the data from the feature visualisations, which showed that the "atemp", "casual" and "registered" all had a high collinearity to other variables and were extensions of the dependant variable so were consequently dropped. Figure 2.2 shows the results from the linear regression model, these will be used as the baseline to compare all future optimisations and regression models.

FIGURE 2.2: Linear Regression Evaluation Results

Metrics	Values
Mean Absolute Error	69.23730
Root Mean Squared Error	91.02040
Relative Absolute Error	0.506015
Relative Squared Error	0.295238
Coefficient of Determination	0.704762

2.2.1 Polynomial Feature Expansion

Polynomial feature expansion is the creation of new features from numerical data by generating all of the polynomial combinations up to some degree value. In the experiments on Azure ML Studio only direct feature polynomial expansions were created. In example, a degree $n = 2$ and column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ would create the polynomial permutations $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{x^2}$, $\boldsymbol{y^2}$ and $\boldsymbol{xy}$, however only the $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{x^2}$ and $\boldsymbol{y^2}$ permutations are considered in this assignment. The evaluation metrics for second order polynomial expansion of the "temp", "hum" and "windspeed" variables are given in figure 2.3. Comparing the results in this figure to the values obtained with the normal linear regression model, it is evident that the expansion did not increase the accuracy of the predicted data. Due to the expansion of the features in the data, higher values of degrees will cause the feature count to increase rapidly which will subsequently result in over-fitted model predictions. The results plotted in the table are an indication of the insignificance of the generated features, the introduction of an feature raised to a degree that is insignificant makes the original feature also insignificant dropping the determinability of the dependant variable using the feature. Results were obtained expanded feature sets to the second and third degree before arriving at this conclusion due to the possibility of over-fitting.

FIGURE 2.3: Second Order Polynomial Feature Expansion of "temp", "hum" and "windspeed"

Metrics	Values
Mean Absolute Error	69.11705
Root Mean Squared Error	90.80035
Relative Absolute Error	0.505136
Relative Squared Error	0.293812
Coefficient of Determination	0.706188

2.3 Boosted Decision Tree

The second model used was Boosted Decision Tree Regression, which works by creating a collection of regression trees that utilise boosting for increasing the accuracy of predictions by fitting data of the prior regression trees. The Boosted Decision Tree Regression model was trained using identically split data as the linear regression model. This ensures the prediction metrics are conclusive and can be used to draw conclusions on the effectiveness of the models. The predictions made using this model can be seen in 2.4. This figure shows that the predictions have a much lower disparity on average to that of the linear model predictions shown in figure 2.1, suggesting that the model is more accurate.

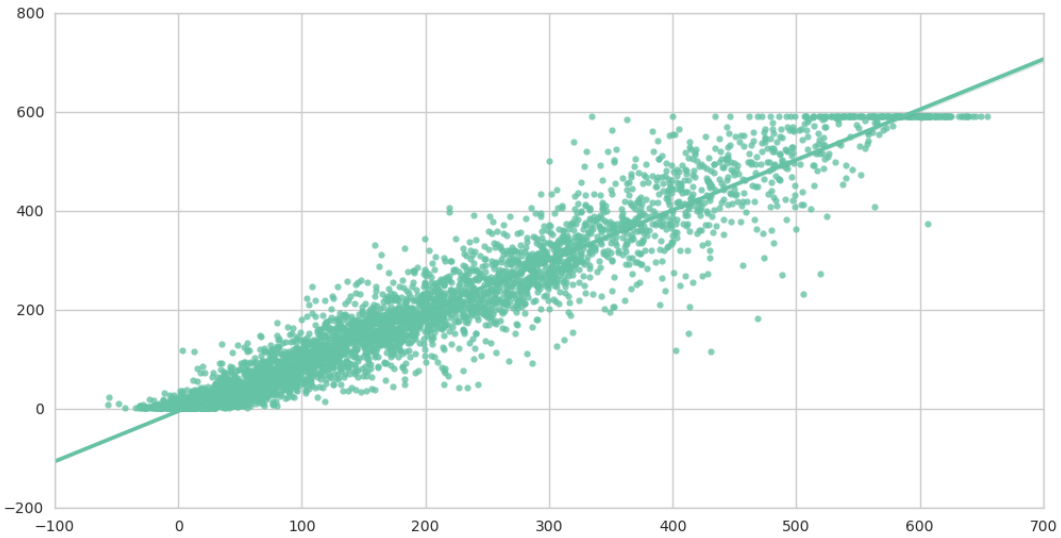

FIGURE 2.4: Boosted Tree Regression Model Scatter Plot

The accuracy of the model can be seen in figure 2.5, where the accuracy increase over the linear regression model results is clear. The mean absolute error value is 2.22 times smaller than the linear regression MAE metric validating the effectiveness of this model.

FIGURE 2.5: Boosted Decision Regression Tree Evaluation Metrics

Metrics	Values
Mean Absolute Error	31.18420
Root Mean Squared Error	45.00733
Relative Absolute Error	0.227907
Relative Squared Error	0.072187
Coefficient of Determination	0.927813

2.4 Decision Forest

Dissimilar to many regression techniques, the Decision Forest is non-parametric meaning that there is no requirement that the data should fit a normal distribution. Decision trees are traversed and simple logic is computed at each node (leaf) until a decision is reached. Some benefits of this model are that it's resource efficient in

memory and computation time in both training and tree traversal. Feature selection and classification is integrated making the effect of outliers minimal on the decision boundaries, this is also a reason decision forests are effective at recursive feature elimination.

Figure 2.6 shows the predicted values of this model. The shape of the plotted points is similar to the Boosted Decision Tree model which is expected. This shows the accuracy of this model is the same as or better than both the previous models. A point to note is that the predicted points in this model are within the bounds of the original data which is a good feature of this model for prediction with this single dataset however in terms of generalisation it may provide less accurate results for future test features outside the bounds of the original data. The disparity of predictions against the regression line is low in this plot and is indicative of the high accuracy shown in figure 2.7.

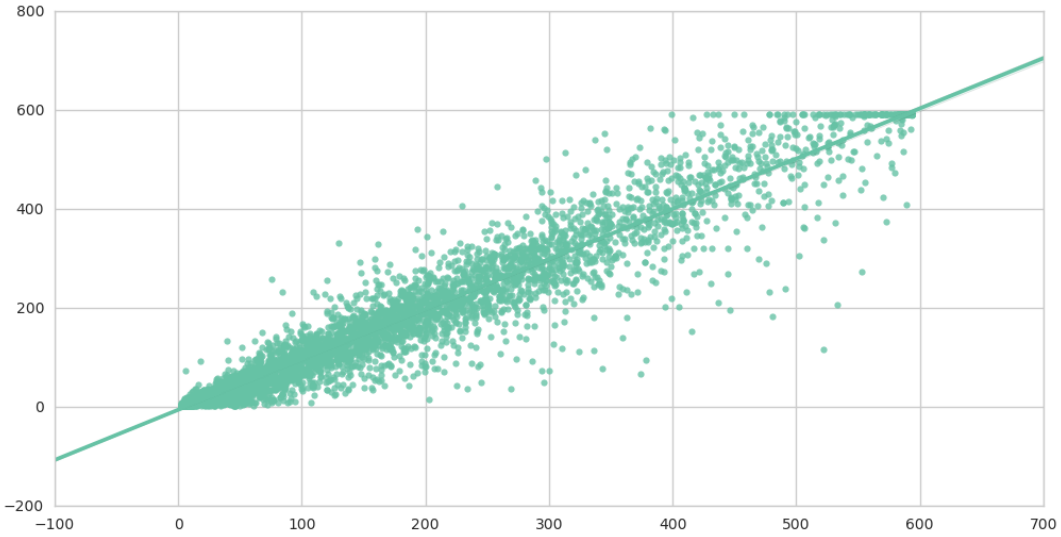

FIGURE 2.6: Decision Tree Regression Model Scatter Plot

The results portrayed below show the lowest mean error out of the three models evaluated in this section. The coefficient of determination also suggests that future recorded data of the dependent variable can be predicted with the independent variables used. The value of 0.91 shows that 91% of the variance in "count" is predictable from the independent features. As with the Boosted Decision Tree this model has a 2.27 times lower mean absolute error in the predicted data, which is a vast improvement. The full Azure ML code used in the evaluation of the three models mentioned in this section is given in B.1.

FIGURE 2.7: Boosted Decision Regression Tree Evaluation Metrics

Metrics	**Values**
Mean Absolute Error	30.50673
Root Mean Squared Error	47.40714
Relative Absolute Error	0.222956
Relative Squared Error	0.080091
Coefficient of Determination	0.919909

3 Model Selection

3.1 Hyper Parameter Tuning

The Boosted Decision Tree model has several configurable parameters than can optimise the accuracy of the predicted data. These parameters will be optimised in this section to achieve an optimal accuracy in the current data configuration, with considerations given to over and under fitting of the data. The four tunable parameters of the Boosted Decision Tree Model are "Maximum number of leaves per tree", "Maximum number of samples per leaf node", "Learning rate" and the "Number of constructed trees". The first parameter that will be optimised will be the "Maximum number of samples per leaf node" parameter. This parameter decides the minimum number of cases required when the terminal nodes are constructed in a single regression tree. The number of cases determine the threshold for creating the decision boundaries (rules), for a value of 10 for example at least 10 cases would have to meet the same conditions in the training data. For the purposes of tuning this parameter the configuration of the other parameters respectively for "Maximum number of leaves per tree", "Learning rate" and the "Number of constructed trees" are 32, 0.4 and 100. To evaluate the optimal number of samples per leaf node a range of parameters is trained on and the MAE metric is used to evaluate the effectiveness of the changes; the range of parameters used is 1,2,3,4,6,8,10,15,20,40 and 80. The results of these tests are shown in figure 3.1 and in table 3.2.

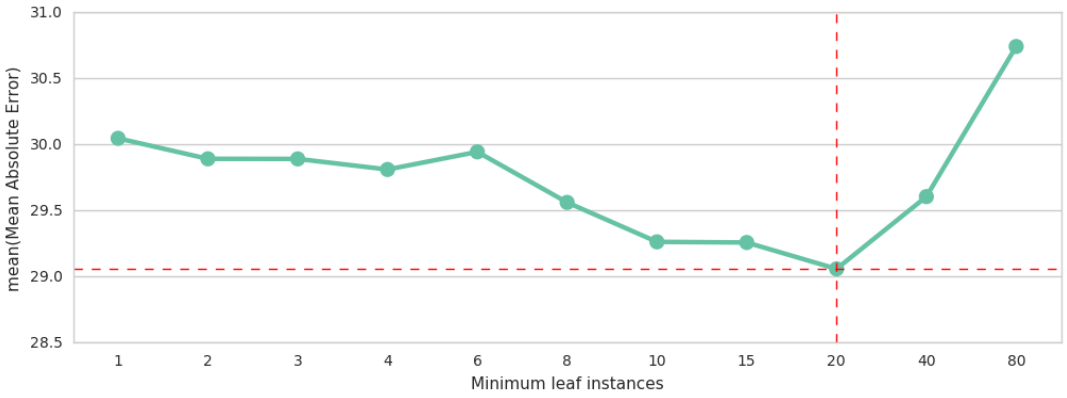

FIGURE 3.1: Maximum Number of Samples Per Leaf Node Parameter Tuning with Parameter Grid of 1, 2, 3, 4, 6, 8, 10, 15, 20, 40 and 80

Both figure 3.1 and table 3.2 show clearly that the optimal number of terminal nodes in the regression trees in this configuration is 20 leaf instances. Figure 3.1 also shows the performance increase and decrease over the number of samples. The mean absolute error was decreased by 1.68 at the graph minima with the parameter value 20, showing that this is the optimal number of leaf instances. Decision trees are susceptible to over fitting with their low bias and high variance, due to this the next section evaluates the effectiveness of the parameter tuning using test set validation and k-fold cross validation.

FIGURE 3.2: Maximum Number of Samples Per Leaf Node Parameter Tuning Accuracy Results

Number of leaves	Minimum leaf instances	Learning rate	Number of trees	Mean Absolute	Root Mean Squared Error	Relative Absolute Error
32	80	0.4	100	30.741998	43.476800	0.224675
32	1	0.4	100	30.046830	44.379461	0.219595
32	6	0.4	100	29.944394	43.660085	0.218846
32	2	0.4	100	29.891666	44.193692	0.218461
32	3	0.4	100	29.891070	43.496961	0.218456
32	4	0.4	100	29.811025	43.231019	0.217871
32	40	0.4	100	29.605630	42.382219	0.216370
32	8	0.4	100	29.561898	43.379929	0.216051
32	10	0.4	100	29.263159	42.615922	0.213867
32	15	0.4	100	29.258370	42.249984	0.213832
32	20	0.4	100	29.058704	42.137893	0.212373

3.2 Parameter Evaluation

In figure 3.1 its clear that plotted points greater than 20 leaf instances show some form of over-fitting due to the almost linear decrease in performance. Under fitting would be visualised on the graph by poor performance usually aggregating on the left side of the figure due to low model complexity not fully capturing the trends of the data. Variance in the predicted MAE accuracy would be an indication of an over fitted model also, which can be visualised by partitioning and sampling the data into folds and cross validating the generated models, an example of which is shown in figure 3.3. If the data is partitioned into n number of sections and each section n has a largely different MAE value, this would indicate the model is fitted to the data.

FIGURE 3.3: Cross Validation Evaluation by Fold of Optimal Leaf Instances Parameter Tuning in 3.1

Fold Number	Fold Observations	Mean Absolute Error	RMSE	RAE
0	1738	29.1165960	42.563350	0.2111462
1	1738	28.6542739	42.507841	0.2116424
2	1738	28.2443855	41.494675	0.2102019
3	1738	29.4021917	42.005415	0.2086671
4	1738	29.9535265	44.840326	0.2285309
5	1738	29.0454337	43.100412	0.2128783
6	1738	28.7775609	41.948336	0.2055955
7	1737	30.4580595	44.753853	0.2227353
8	1738	29.5981352	42.059156	0.2193178
9	1738	28.6592953	40.508165	0.2065966

Figure 3.3 shows the result of 10 fold Cross Validation of the Boosted Decision Tree model using the optimal parameter for leaf instances found in table 3.2. The variance between partitioned sets $n_0 ... n_9$ over 10 cross validation model folds is low

for the mean absolute error indicating no signs of over fitting, with a standard deviation of only 0.67, which consequently provides validation for the optimal parameter selection chosen. Cross validation using partitioned data provides a more in depth analysis of the accuracy of the model over the test set data alone, also providing greater metric spread for assessment of the future predictability the model provides.

In regards to 10 fold cross validation for the entire parameter grid, the folds are averaged into one error metric. As shown in figure 3.4 this changes the optimal leaf instances value to 10. This is more reflective of the data and provides lower error than the MAE selected using the split data alone. For this reason the rest of the parameter tuning in the following section will use k-folds cross validation to tune and find an optimal set of parameters. In contrast to the tuned model results given in figure 3.2 using the test data split of the original data, the results for the tuned cross validation models are given in 3.5.

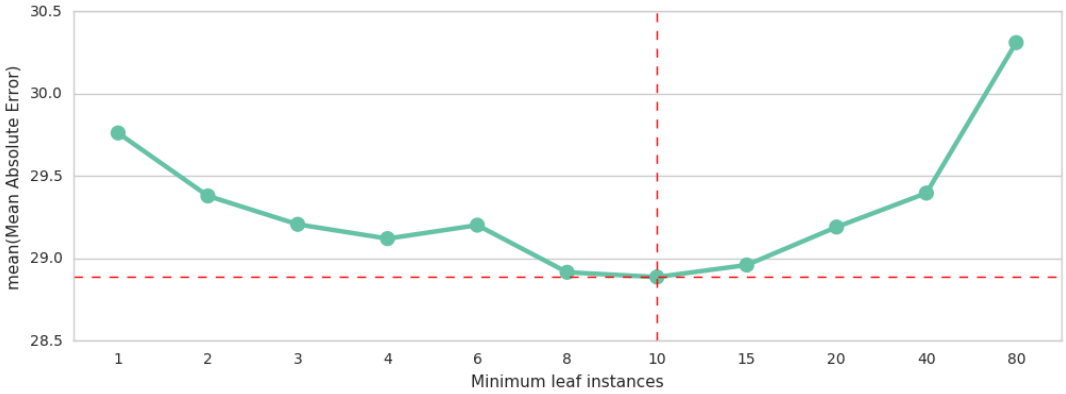

FIGURE 3.4: Maximum Number of Samples Per Leaf Node Parameter Tuning with Parameter Grid and 10-Fold Cross Validation

FIGURE 3.5: Maximum Number of Samples Per Leaf Node Parameter Tuning Accuracy Results using Cross Validation Models from Partitioned Data

Number of leaves	Minimum leaf instances	Learning rate	Number of trees	Mean Absolute	Root Mean Squared Error	Relative Absolute Error
32	80	0.4	100	30.310429	43.56807	0.221779
32	1	0.4	100	29.761778	44.21245	0.217765
32	40	0.4	100	29.396838	42.89142	0.215094
32	2	0.4	100	29.379578	43.59388	0.214968
32	3	0.4	100	29.207112	43.36706	0.213706
32	6	0.4	100	29.202212	42.88607	0.213670
32	20	0.4	100	29.190872	42.59749	0.213587
32	4	0.4	100	29.120549	43.01422	0.213073
32	15	0.4	100	28.960249	42.29167	0.211900
32	8	0.4	100	28.916116	42.65781	0.211577
32	10	0.4	100	28.887243	42.34455	0.211366

3.2.1 Further Parameter Tuning

The tuning that occurs in this section will used fixed parameters either optimised from previous sections or using an arbitrary value. The only value that will have a range of parameters will be the parameter to be optimised. and one parameter with a range. Another parameter that can be tuned to find the optimal model is "Maximum number of leaves per tree". This value specifies the maximum number of terminal leaves that can be created for each individual regression tree. This variable will directly result in an increase or decrease in size of the decision trees in the Boosted Decision Model which can increase the time it takes to train the model. The parameter grid that was searched to find the optimal value for this parameter was 1, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. Figure 3.6 shows the performance of the parameter tuning results over the number of leaves per regression tree. This trend of the line is as expected, due to the low complexity of the decision trees at the lower end of the grid search they do not capture the essence or trends of the data. The data is under fitted at the start of of this chart but then stabilises for the rest of the parameter grid showing that 70 is the optimal parameter for this data.

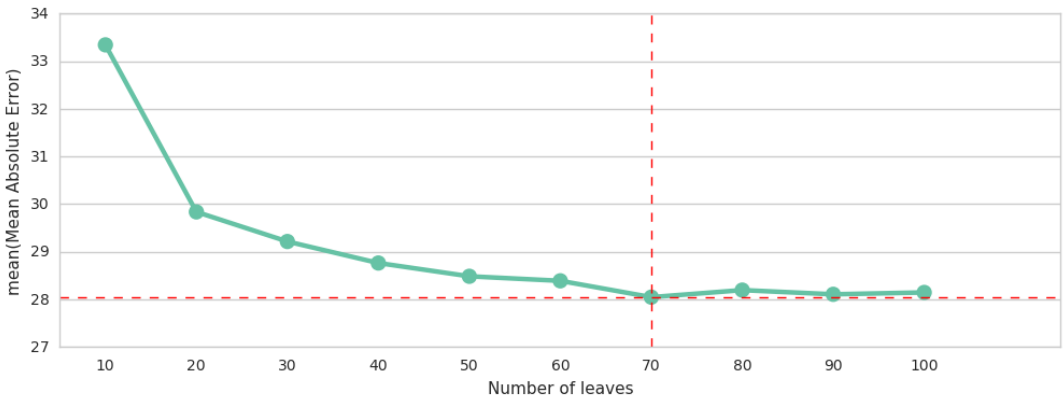

FIGURE 3.6: Maximum Number of Leaves Parameter Tuning with Parameter Grid

The next two parameters to optimise are "Learning Rate" and "Maximum Number of Leaves per Tree". The learning rate defines the step size while the model is being trained. The lower the learning rate, the longer the model will take to converge on an optimal solution. The learning rate graph shown in figure 3.7, clearly shows a relationship between the mean absolute error and the learning rate, the lower the learning rate is the better the predictions of the model become. The last parameter that is optimised is the number of trees constructed in the model. The variable decides the number of trees that are constructed in the model, the more trees that are created will relate to longer model training times. Figure 3.8 shows the results of the cross validation model tuning performed on the data with a parameter range of 1, 50, 100, 200, 300, 400, 500, 600 and 700 trees. It's evident from the plot that due to he nature of this model the number of trees must be greater than one to allow for accurate predictions of forecast observations. Figure 3.9 shows a clearer view of the important parameter ranges in the search grid since the high error of one search tree makes it hard to see the differences. From this plot it's evident that the optimal number of tree's is around 400. This concludes the parameter optimisation steps with end parameters of 70, 10, 0.2 and 400 for 'Maximum number of leaves per tree", "Maximum number of samples per leaf node", "Learning rate" and the "Number of constructed trees" respectively.

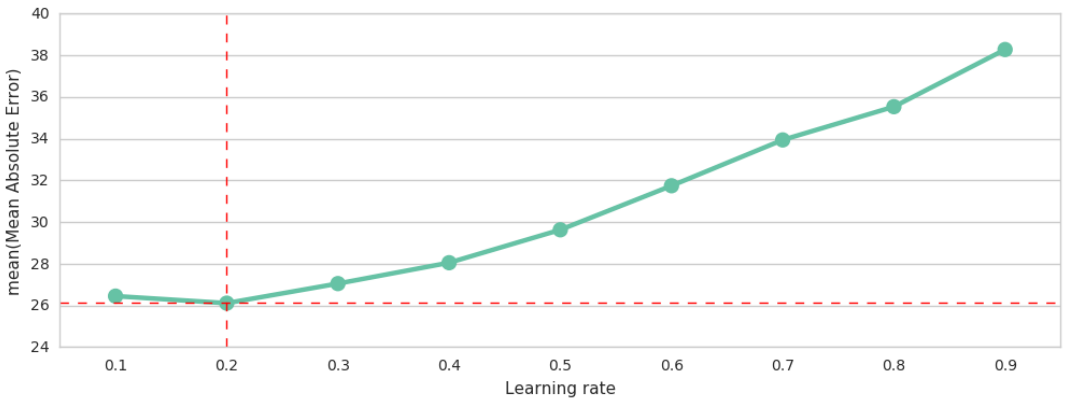

FIGURE 3.7: Learning Rate Tuning with Parameter Grid

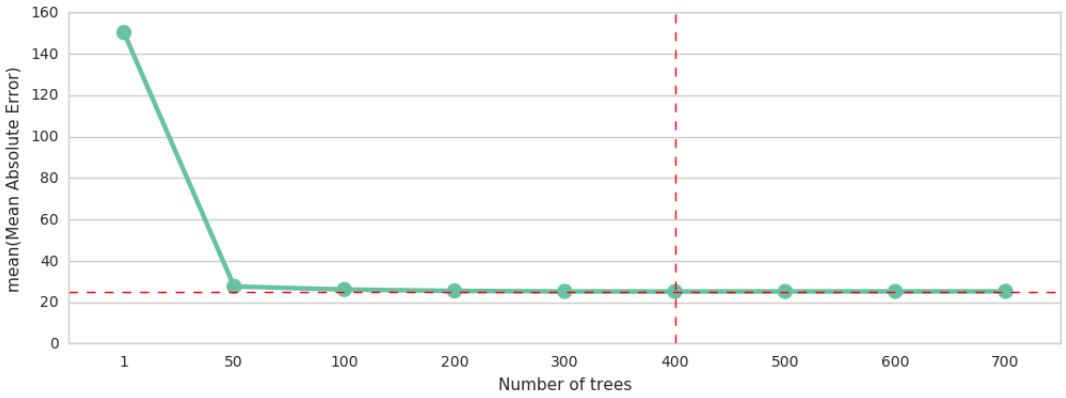

FIGURE 3.8: Maximum Number of Trees Tuning with Parameter Grid

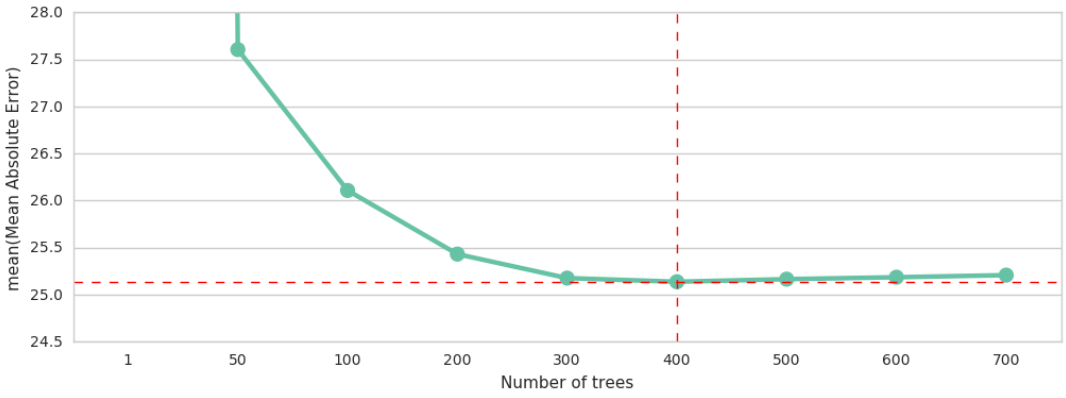

FIGURE 3.9: Zoomed Maximum Number of Trees Tuning with Parameter Grid

4 Time Series Modelling

4.1 Splitting Data by Year

To test the predictability of the models on future years the dataset was split by the year into two different datasets. This was done using the split module of Azure ML Studio as seen in the complete code in figure D.1. The year column now contains only a single value so this was removed from both of the new datasets. The new partitioning allows the model to be trained on one year to attempt to predict the number of bike rentals the next year. Figure 4.1 shows that the results are dramatically worse than the results obtained in the previous section.

FIGURE 4.1: MAE Accuracy Results on 2011 and 2012 data

	Linear	**Boosted Decision Tree**	**Decision Forest**
New MAE	100.3597	81.69601	81.61298
Old MAE	69.23730	31.18420	30.50673
Difference	1.449503	2.619789	2.675245

The poorer performance could be attributed to the model being too well fitted to the previous years data trends resulting in inaccurate predictions in the new year. This point is validated in table 4.2 where it's clear the model is over fitted to the 2011 data. To make the comparison fair, close to optimal parameters for the Decision Forest were also found. Otherwise the Boosted Decision Tree Model would have a great advantage from the aforementioned optimisations. The close to optimal parameters for the "Number of Decision Trees", "Maximum Decision Tree Depth", "Maximum Random Splits Per Node" and "Minimum Number of Samples Per Node" were 32, 128, 64 and 1 respectively. The results these optimal parameters were derived from can be seen in table D.2.

FIGURE 4.2: Testing and Training MAE Results for 2011 and 2012 Data

	MAE	**RMSE**	**RAE**	**RSE**	**COE**
Training	26.48103	40.24269	0.248788	0.089692	0.910308
Testing	83.26523	111.7301	0.531122	0.357417	0.642583

4.2 Time Series Feature Engineering

To improve the accuracy of the models, features for the "count" of total bikes rented in the past x hours were generated. Twelve of these features were generated so that each recorded row included the last twelve hours of rental history. This was done so that the dependant variable could possibly be predicted hours in the future instead

of a year. To achieve this a python script was written that would write the data to each row for x hours in the past or future. Since these new features will contain information from the past or future the first or last x rows for the dataset will have to be removed since they do not have any information to fill the features. The python script that did this can be seen in figure 4.3.

```python
import pandas as pd

def azureml_main(df = None):

    hour_range = 12
    column_name = "hour"
    dependant_name = "count"

    direction = 0 # past : 0, future : 1

    for x in range(1, hour_range + 1):
        new_name = column_name + ("+" if direction else "-")
        ↪ + str(x)
        df[new_name] = df[dependant_name].shift(x * -1 if
        ↪ direction else x)

    df.drop(df.head(hour_range).index if not direction else
    ↪ df.tail(hour_range).index, inplace=True)

    df["year"] = pd.to_numeric(df["year"])

    return df,
```

FIGURE 4.3: Python Code for Generating Last x Hour Bike Rental Features

Figure 4.4 shows the results from all three of the regression models. It's clear from these results that the newly engineered features improve the MAE score drastically, all converging around an MAE of 50. The results are from models that were trained with the 2011 data and tested on the 2012 data. Using a similar method to how the twelve past hour features were generated the past twelve days were also generated to compare against the performance achieved this far. The modified python code for this can be seen in figure 4.5 where the lines modified select the "count" dependant variable in steps of 24 hours over the last 12 days. Consequently more rows had to be trimmed from the data to accommodate for missing values.

FIGURE 4.4: MAE Accuracy Results on 2011 and 2012 data with engineered hour features

	Linear	Boosted Decision Tree	Decision Forest
New MAE	49.87806	44.98089	47.45665
Old MAE	100.3597	81.69601	81.61298

4.3 Optimal Engineered Features

The results for the models trained with the new hour features, day features and hour and day are shown in figure 4.6. The error metrics show that the most optimal parameter set is utilising both of the newly constructed feature sets so these models would be used in the final company product. The final error metrics are within a small range of each other with the Decision Forest model falling behind. The Boosting Decision Tree model reduces bias and less so, variance in the aggregation of many models (Fontama *et al.*, 2014). Whereas the Decision Forest model focuses on minimising the variance through non correlating tree creation, providing discrete predictions that can be interpolated between. Over the continuous predictions provided by the Boosted Decision Tree model. Decision Forest does not minimise bias but is less sensitive to outliers and poor parameter optimisation (Criminisi and Shotton, 2013). Boosted Decision Tree models produce more accurate results and are the final model used in this assessment of the data, however as a note, the Decision Forest is more robust and provides reliable results with minimal optimisation.

```python
import pandas as pd

def azureml_main(df = None, dataframe2 = None):

    hour_range = 12
    column_name = "day_count"
    dependant_name = "count"

    direction = 0 # past : 0, future : 1

    for x in range(1, hour_range + 1):
        new_name = column_name + ("+" if direction else "-") + str(x)
        df[new_name] = df[dependant_name].shift((x * 24) * -1 if direction else x * 24)

    df.drop(df.head(hour_range * 24).index if not direction else df.tail(hour_range * 24).index, inplace=True)

    df["year"] = pd.to_numeric(df["year"])

    return df,
```

FIGURE 4.5: Python Code for Generating Last x Days Bike Rental Features

FIGURE 4.6: Engineered Feature Mean Absolute Error Comparison

	Linear	Boosted Decision Tree	Decision Forest
Baseline	100.3597	81.69601	81.61298
Past 12 Hour Count	49.87806	44.98089	47.45665
Past 12 Day Count	62.57473	52.24416	55.04498
Past 12 Day and Hour Count	41.27679	41.01363	43.76771

5 Time Series Prediction

Using the last twelve hours to predict the next hour of the bike rental count variable is useful but to increase it's effectiveness the ability to predict further into the future would allow the company to better utilise the information gained. To assess the predictability of future rental counts multiple variables will be predicted and evaluated to examine the performance. The data that will be used to facilitate this will be the split 2011 and 2012 data, the models will attempt to predict a range of hours into the future from a single hour feature observations of the 2012 data using models trained on the 2011 data. The baseline MAE accuracy will be calculated first and then every hour up until a maximum of five hours ahead of time, leaving a good proportion of time during the work day of the company to use the information.

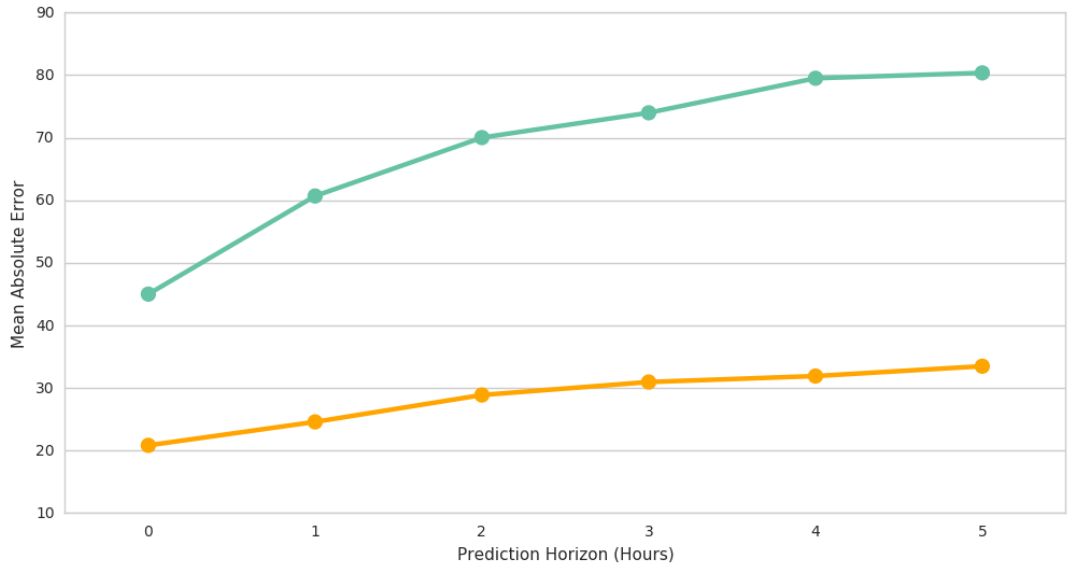

FIGURE 5.1: Prediction Horizon for 5 hours into the future of the 2012 dataset using a model trained with 2011 Data (Green) and 75% and 25% Split (Orange)

Figure 5.1 shows the performance degradation over, ahead of time predictions. Its evident from this graph that the error gain initially when predicting only one or two hours ahead is great, and then for further future points the error gain decreases and starts to plateau. To generate this plot a dataset was created with the twelve past hour counts as described in the previous section, with the addition of five new count features. The new count features are the counts of the next one to five count hours. These new variables are set as the dependant variables for five separate models that are trained, scored and evaluated. The evaluations are then joined into a dingle data set and plotted to generate the graph seen in 5.1. The code used to generate this plot can be seen in appendix D along with the script for generating the future count features. For the purpose of completion, figure 5.1 also shows the diminishing accuracy of the parameter horizon for the initial 75% and 25% data split in orange.

Bibliography

Criminisi, A. and Shotton, J. (2013) *Decision forests for computer vision and medical image analysis*. Springer Science and Business Media.

Fontama, V., Barga, R., and Tok, W. H. (2014) *Predictive Analytics with Microsoft Azure Machine Learning: Build and Deploy Actionable Solutions in Minutes*. Apress.

Hill, T., Lewicki, P., and Lewicki, P. (2006) *Statistics: methods and applications: a comprehensive reference for science, industry, and data mining*. StatSoft, Inc.

Hyndman, R. J. and Koehler, A. B. (2006) Another look at measures of forecast accuracy. *International Journal of Forecasting*. 22 (4), pp. 679–688.

Microsoft (2017). *Azure ML*.

A Data Analysis

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

FIGURE A.1: Head of depth 5 of Bike Rental Data

```python
import pandas as pd

def azureml_main(df1 = None, df2 = None):

    seasons = ["Spring", "Summer", "Autumn", "Winter"]
    months = ["January", "February", "March", "April", "May",
    ↪ "June", "July", "August", "September", "October",
    ↪ "November", "December"]
    days = ["Monday", "Tuesday", "Wednesday", "Thursday",
    ↪ "Friday", "Saturday", "Sunday"]
    years = ["2011", "2012"]

    df1.season.replace([1, 2, 3, 4], seasons, inplace=True)
    df1.mnth.replace([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    ↪ months, inplace=True)
    df1.weekday.replace([0,1,2,3,4,5,6], days, inplace=True)
    df1.yr.replace([0, 1], years, inplace=True)

    df1.dteday = df1.dteday.dt.day

    df1 = df1.rename(columns = {
        "dteday":"day",
        "mnth":"month",
        "yr":"year",
        "hr":"hour",
        "cnt":"count"
        })
    return df1,
```

FIGURE A.2: Feature Engineering and Data Readability Python Script

FIGURE A.3: Bike Rental Data Statistics

Feature	Min	Max	Mean	Median	Mode	Standard Deviation
day	NA	NA	NA	NA	NA	NA
season	NA	NA	NA	NA	NA	NA
yr	NA	NA	NA	NA	NA	NA
mnth	NA	NA	NA	NA	NA	NA
hr	NA	NA	NA	NA	NA	NA
holiday	NA	NA	NA	NA	NA	NA
weekday	NA	NA	NA	NA	NA	NA
workingday	NA	NA	NA	NA	NA	NA
weathersit	NA	NA	NA	NA	NA	NA
temp	0.02	1	0.496987168	0.5	0.62	0.192556121
atemp	0	1	0.475775102	0.4848	0.6212	0.171850216
hum	0	1	0.627228839	0.63	0.88	0.192929834
windspeed	0	0.8507	0.190097606	0.194	0	0.122340229
casual	0	367	35.67621842	17	0	49.30503039
registered	0	886	153.7868692	115	4	151.3572859
cnt	1	977	189.4630876	142	5	181.3875991

FIGURE A.4: Season and Weather Situation Plot

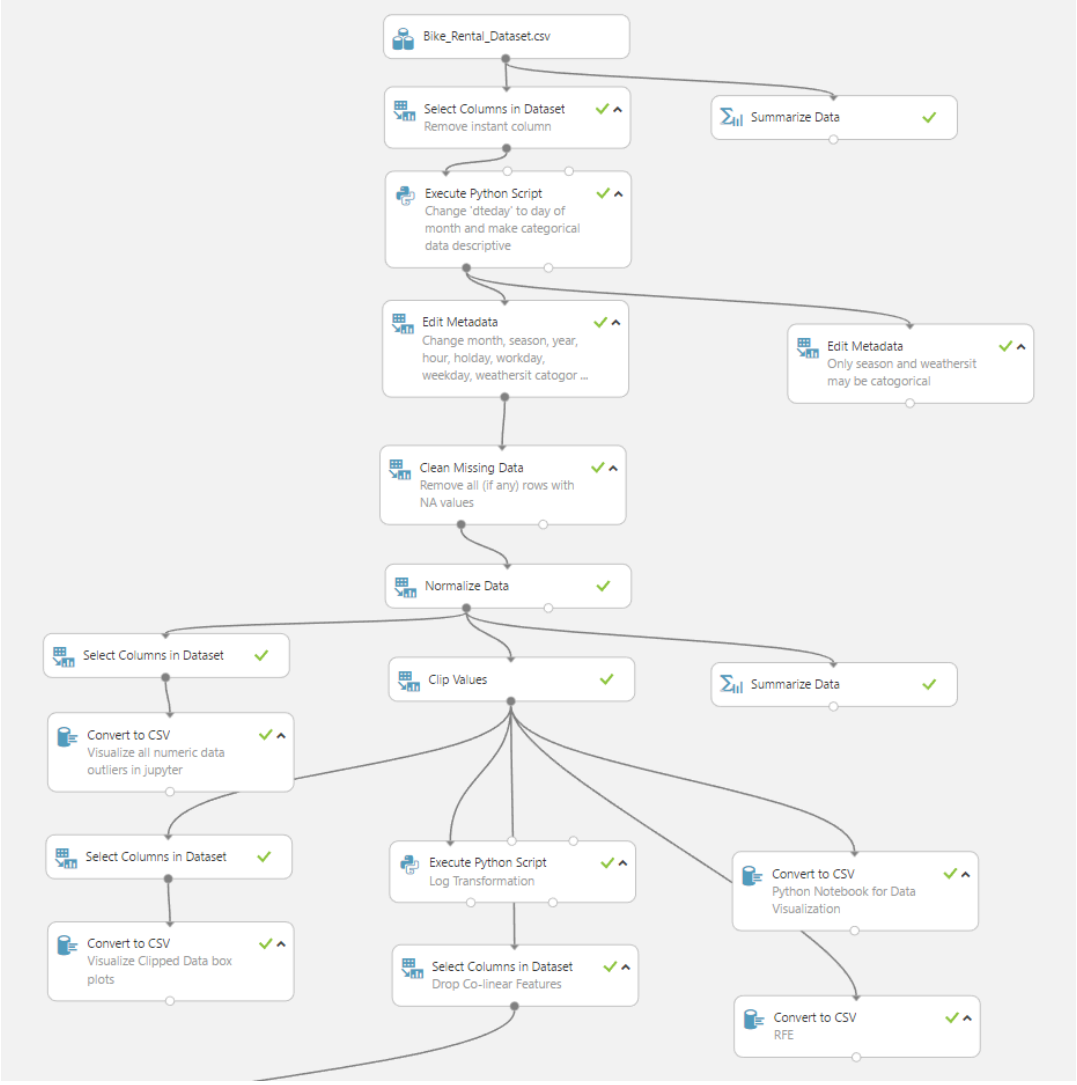

FIGURE A.5: Azure ML Preprocessing Code

B Algorithm Comparison

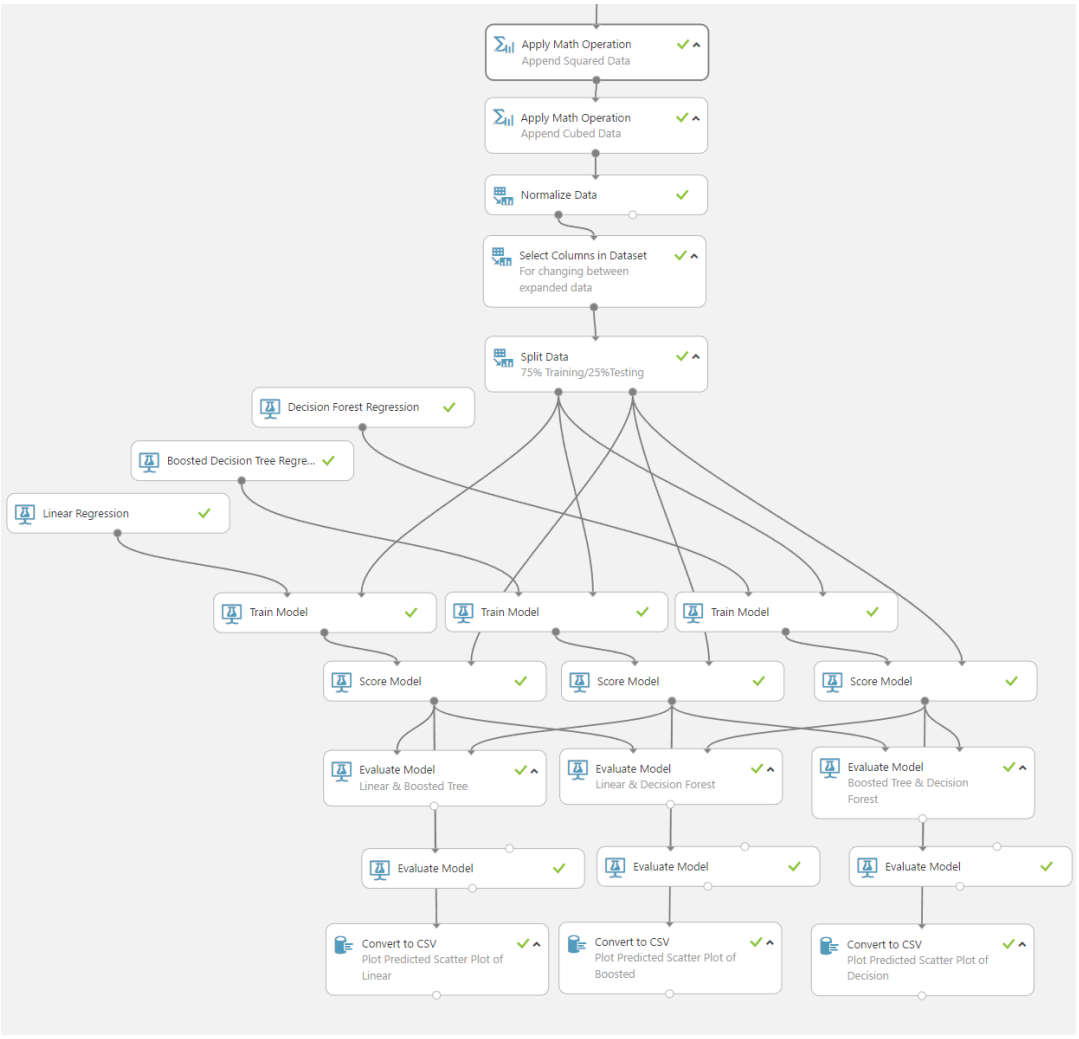

FIGURE B.1: Azure ML Model Evaluation Code

C Model Selection

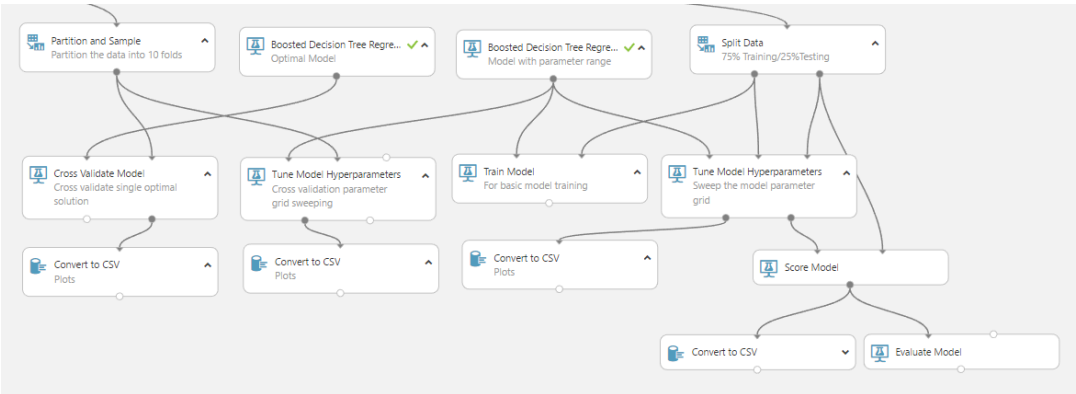

FIGURE C.1: Azure ML Model Hyper Parameter Tuning Code

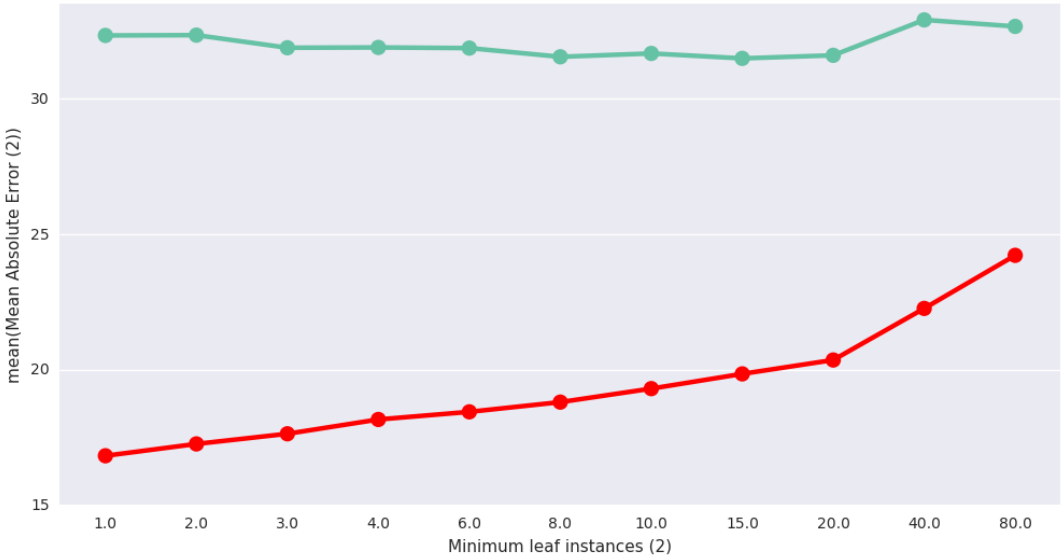

FIGURE C.2: Training (red) and Testing (green) results of Leaf Instances Parameter Tuning showing over fitting at 20

D Time Series

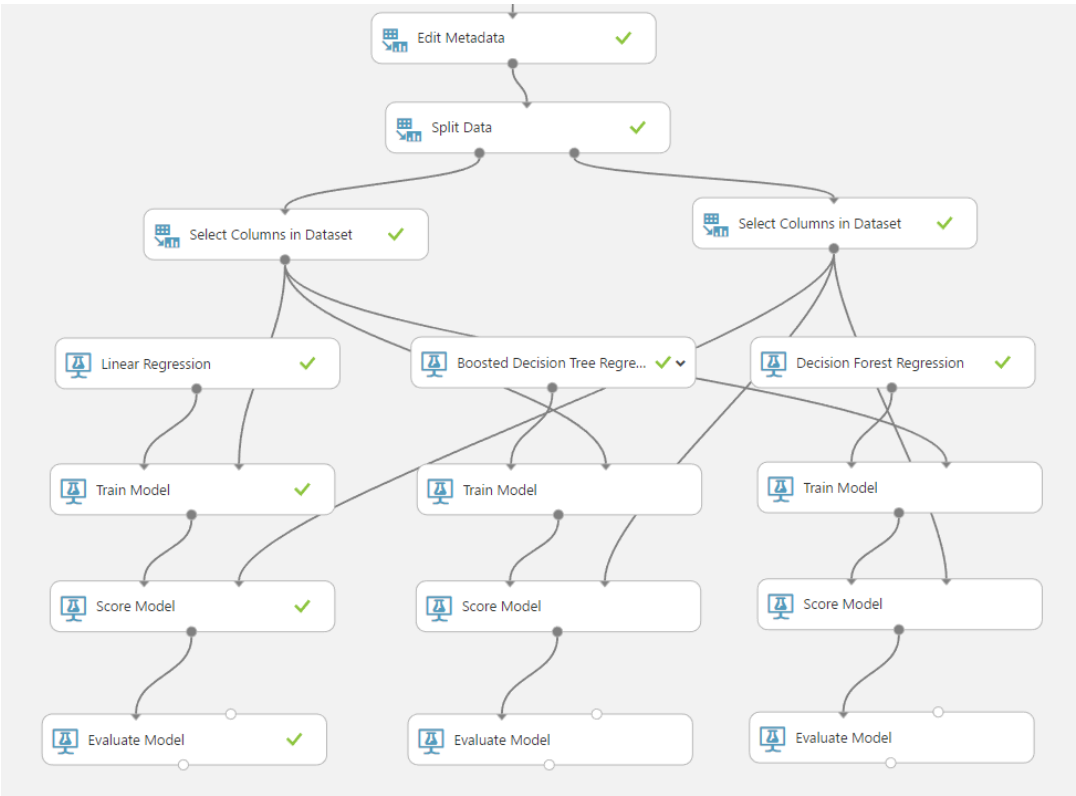

FIGURE D.1: Azure ML 2011 and 2012 Code

FIGURE D.2: Decision Forest Parameter Tuning Full Grid

Number of Samples Per Leaf	Random Splits Per Node	Decision Tree Depth	Number of trees	Mean Absolute Error	Root Mean Squared Error
1	128	64	32	28.705180	44.978445
1	1024	64	32	29.454867	46.887021
4	128	64	32	29.485198	46.088391
4	128	64	8	29.993437	47.862209
1	128	64	8	30.391164	47.813281
1	1024	64	8	31.080219	49.650462
4	1024	64	32	31.960779	49.860797
4	1024	64	8	33.855256	53.424063
4	128	64	1	38.892955	66.486497
16	128	64	32	39.398852	57.285197
16	128	64	8	39.478451	57.737610
1	128	64	1	40.275266	67.392526
1	1024	64	1	40.655186	67.774412
16	1024	64	32	41.215303	60.957451
16	1024	64	8	42.377359	62.857856
1	1024	16	32	42.867132	60.910806
4	1024	16	32	43.235165	61.428666
1	128	16	32	43.601150	62.100589
1	1024	16	8	43.955163	63.214546
4	1024	16	8	44.312902	63.059182
1	128	16	8	44.515061	63.833913
4	1024	64	1	44.550550	75.214774
4	128	16	32	44.596651	63.494647
4	128	16	8	46.112556	65.890862
16	1024	64	1	48.114570	75.957804
16	1024	16	32	48.741105	68.021774
16	128	16	32	48.745413	68.223444
16	128	64	1	49.410826	77.246600
16	128	16	8	49.542919	69.163880
16	1024	16	8	50.555675	70.429296
4	128	16	1	53.126365	81.368054
1	128	16	1	53.178401	80.799003
4	1024	16	1	54.484697	79.786444
1	1024	16	1	54.485521	80.303448
16	128	16	1	55.908023	82.639154
16	1024	16	1	59.694721	85.948247
1	1	64	8	130.90846	161.55359
1	1	16	8	130.90846	161.55359
4	1	16	8	130.92197	161.55569
4	1	64	8	130.92197	161.55569
1	1	16	1	131.35578	163.99415
1	1	64	1	131.35578	163.99415
4	1	16	1	131.35915	163.99442
4	1	64	1	131.35915	163.99442
16	1	64	1	131.36333	163.99474
16	1	16	1	131.36333	163.99474

```python
import pandas as pd

def azureml_main(df = None, dataframe2 = None):

    hour_range = 5
    column_name = "count"
    dependant_name = "count"
    direction = 1 # past : 0, future : 1

    for x in range(1, hour_range + 1):
        new_name = column_name + ("+" if direction else "-") + str(x)
        df[new_name] = df[dependant_name].shift(x * -1 if direction else x)

    df.drop(df.head(hour_range).index if not direction else df.tail(hour_range).index, inplace=True)
    df["year"] = pd.to_numeric(df["year"])
    return df,
```

FIGURE D.3: Python Code for Future Count Variable Generation

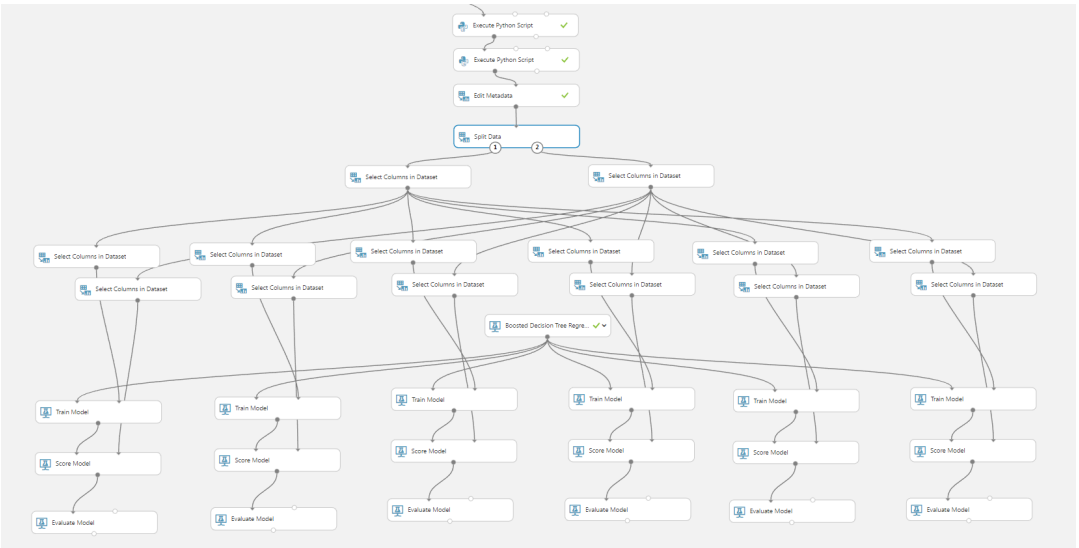

FIGURE D.4: Prediction Horizon Code