Project Report on Predicting Bike Rental Count

Kiran C

17th Jan'19

Contents

1. Introduction

1.1 Problem Statement

The objective of the project is to develop a suitable model that predicts the count of bike rentals based environmental and seasonal settings on daily basis.

1.2 Data

Data comprises of count of bike rentals – "cnt", as dependent variable that depends on the various independent variables. Sample data and the details of the variables is given below

instant	dteday	season	yr	mnth	holiday	weekday	workingday
1	2011-01-01	1	0	1	0	6	0
2	2011-01-02	1	0	1	0	0	0
3	2011-01-03	1	0	1	0	1	1
4	2011-01-04	1	0	1	0	2	1
5	2011-01-05	1	0	1	0	3	1
6	2011-01-06	1	0	1	0	4	1
7	2011-01-07	1	0	1	0	5	1
8	2011-01-08	1	0	1	0	6	0
9	2011-01-09	1	0	1	0	0	0
10	2011-01-10	1	0	1	0	1	1

Table – 1: First 8 variables with first 10 observations as sample

weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	0.363478	0.353739	0.696087	0.248539	131	670	801
1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
1	0.2	0.212122	0.590435	0.160296	108	1454	1562
1	0.226957	0.22927	0.436957	0.1869	82	1518	1600
1	0.204348	0.233209	0.518261	0.089565	88	1518	1606
2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
2	0.165	0.162254	0.535833	0.266804	68	891	959
1	0.138333	0.116175	0.434167	0.36195	54	768	822
1	0.150833	0.150888	0.482917	0.223267	41	1280	1321

Table – 2: Rest of the variables with 10 observations as sample

1.3 Variable Explanation

-> "instant" – replicated the index (Record index)
-> "dteday" – Date in yyyy-mm-dd format
-> "season" – categories of Season (1: Spring, 2: Summer, 3: Fall, 4: Winter)
->"yr" – Year coded (0: 2011, 1: 2012)
-> "mnth" – Month coded(1 through 12)

-> “holiday” – Whether or not a day holiday (0: Not holiday, 1: Holiday) (extracted from Holiday Schedule)

-> “weekday” – Day of week coded (0 through 6)

-> “workingday” – 1, if a day is neither weekend nor holiday; 0,otherwise

-> “weathersit” – (extracted from Freemeteo) Coded as

1: Clear, Few clouds, Partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light snow, Light rain + Thunderstorm + Scattered clouds, Light rain +Scattered clouds

-> “temp” – Normalised temperature in Celsius. Values are derived via (t – t_min)/(t_max – t_min); t_min = -8 and t_max = +39 (only in hourly scale)

-> “atemp” – Normalised felling temperature in Celsius. Values are derived via (t – t_min)/(t_max – t_min); t_min = -16 and t_max = +50 (only in hourly scale)

-> “hum” – Normalised humidity. Values are divided to 100 (max)

-> “windspeed” – Normalised wind speed. Values are divided to 67 (max)

-> “casual” – Count of casual users

-> “registered” – Count of registered users

-> “cnt” – Count of total bike rentals obtained by summing up “casual” and “registered”

Among the total 16 variables, “instant” is just an index of values and doesn’t contribute to the data analysis and hence the rest 15 variables are considered for data pre-processing. The following table lists the variables (independent variables) that help to predict (so are Predictor Variables) the count of bike rentals (“casual”, “registered” and “cnt” – the dependent variables)

S.No	Predictor Variables
1	dteday
2	season
3	yr
4	mnth
5	holiday
6	weekday
7	workingday
8	weathersit
9	temp
10	atemp
11	hum
12	windspeed

Table – 3 Predictor Variables

2. Methodology

2.1 Pre-processing

Data pre-processing is the stage where we make our data ready to feed to the model. It includes various steps like – exploratory data analysis, missing values analysis, feature selection, feature engineering, feature scaling and sampling.

2.1.1 Exploratory Data Analysis

As a part of the exploratory data analysis, the data types of the variables are converted into suitable data types, so that it eases further pre-processing techniques. From Table – 3, there are 12 predictor variables and 3 target variables. The aim of the project is to predict the count of bike rentals as against various seasonal and environmental settings on daily basis. It clearly tells us that the target variable(s) must be a numeric one. Therefore "casual", "registered", "cnt" must be of numeric type. Now let us look at the each predictor variable and its type to be.

-> "dteday" – represents a calendar date in yyyy-mm-dd format. Originally it is of factor type. We let it so until any date part requires to be extracted in the next pre-processing techniques.

-> "season", "yr", "mnth", "holiday", "weekday", "workingday", "weathersit" are originally of int64 type. All of these are coded as 0, 1, 2 etc and represent a particular category corresponding to the respective variables, hence they are categorical variables. They are converted to factor type. Save all these variables including "dteday" in "Cat_Var".

-> "temp", "atemp", "hum", "windspeed", are of float64 type and "casual", "registered", "cnt" are of int64 type. We keep them as they are. Save all of these variables in Con_Var.

2.1.2 Missing Value Analysis

Next step in the data pre-processing stage is to check if there are any missing values in any of the continuous variables, as these values would influence the inferences we draw from the variables. In our present dataset there are no missing values in any of the "Con_Var".

2.1.3 Outlier Analysis

Outliers are the observations that are distant from other observations. They cause the data analysis biased and incorrect model. If they are not treated well, every effort in data analysis is a mere waste. To trace out the outliers, we check box plot. Box plots of Con_Var are depicted in grid form as shown in Fig – 1, Fig – 2 & Fig – 3. From the figures, it is evident that outliers are present in "hum", "windspeed" and "casual". Different ways of dealing with outliers are –

i) Removing them from the dataset
ii) Replacing them by NAs and imputing them either by their respective mean or median or KNN.

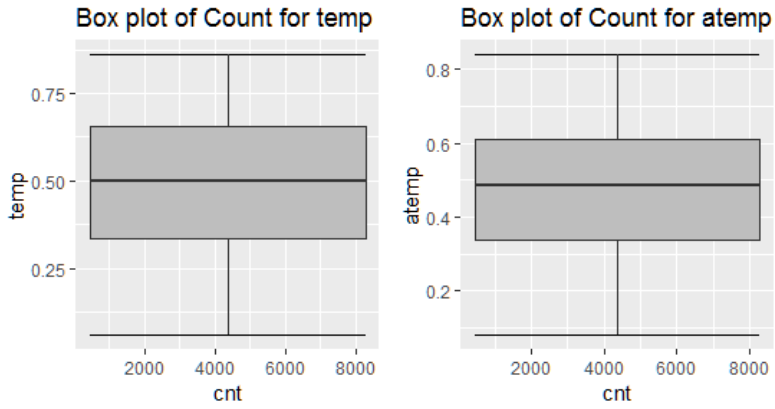

Fig – 1: Box plot grid of “temp” and “atemp” against “cnt”

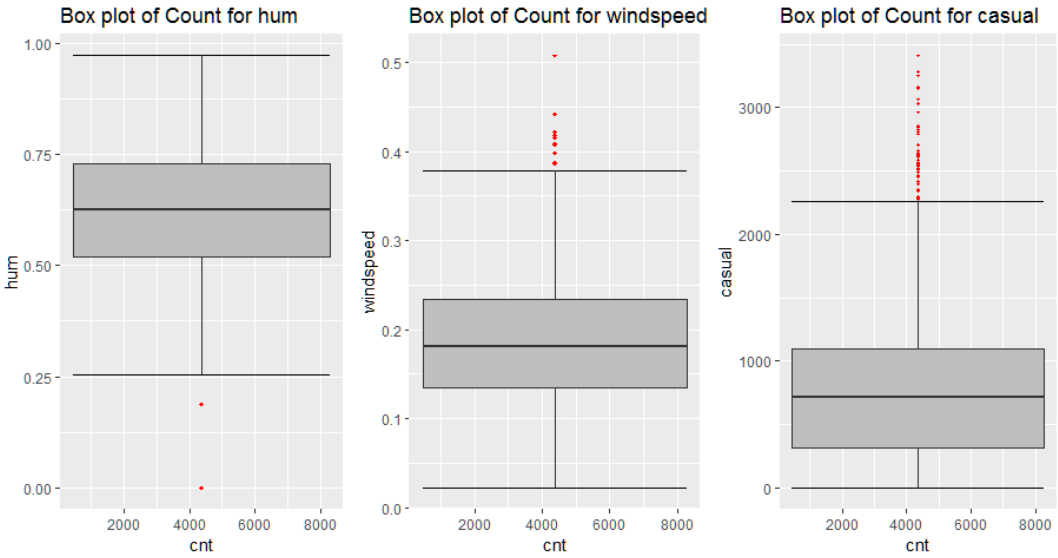

Fig – 2: Box plot grid of “hum” and “windspeed” and “casual” against “cnt”

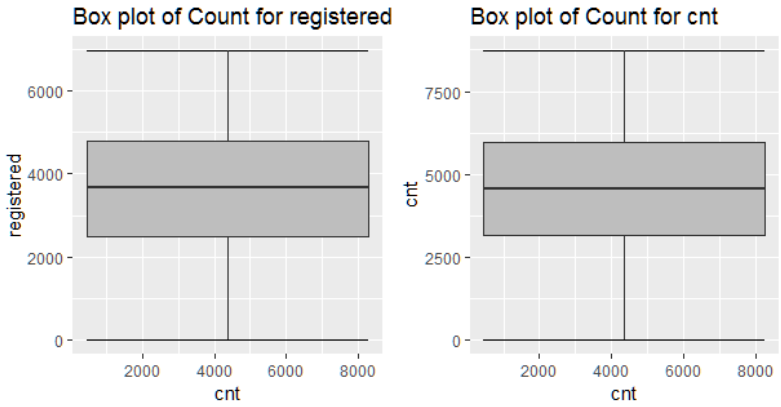

Fig – 3: Box plot grid of “cnt” and “cnt” against “cnt”

As the number of observations is low, it is better not to remove the outliers for a good predictor model. When trying to impute the outliers by the above methods, KNN has given the best results. The results are as depicted in the Fig – 4 (Only grid with box plots of "hum", "windspeed", "casual" is shown as they are the once with outliers).

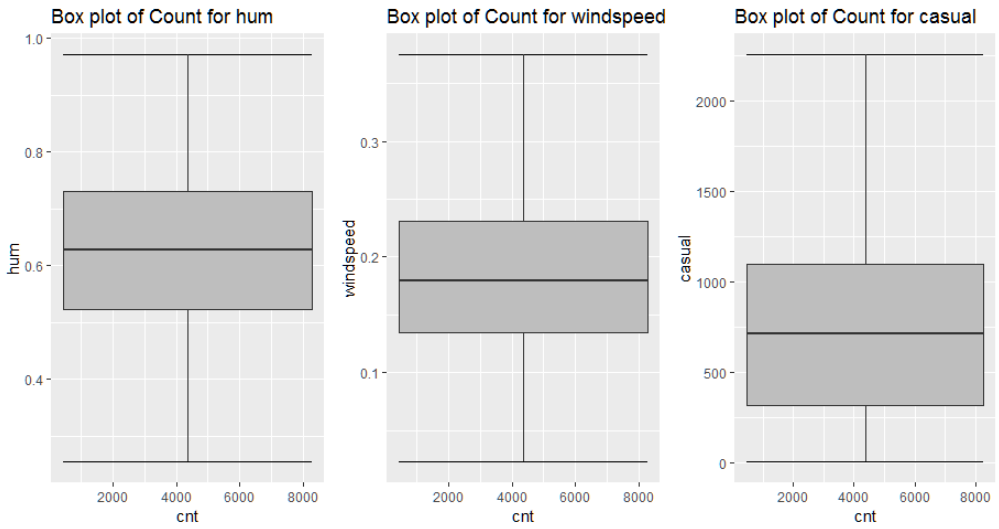

Fig – 4: Outliers after KNN Imputation

2.1.4 Feature Engineering

A) When we look at the variables - "holiday" and "workingday", we can interpret it in the following way:

i) If "holiday" == 0 & "workingday" == 0 => day is "weekend", which in turn include "weekday" coded as 0 and 6 in the dataset. This means we can combine rows with "holiday" == 0 and "workingday" == 0, and create a new value "weekend" in place, by discarding rows of "weekday" with values 0 and 6.

ii) If "holiday" == 0 & "workingday" == 1 => day is "workingday", which in turn includes "weekday" coded as 1 to 5 in the dataset. This means that we can combine rows with "holiday" == 0 and "workingday" == 1, and create a new value "workingday" in place, by discarding rows of "weekday" with values 1 to 5.

iii) If "holiday == 1" => "workingday" = 0 => day is "holiday". In this case whatever the value in "weekday" may hold, it can be discarded.

The above logical combination of variables has a logical connection with another variable "weekday". In the given dataset "weekday" is coded as 0: Sunday through 6: Saturday. It can be interpreted as whether a day is a week day (Monday through Friday) or weekend (Saturday & Sunday). And our above logic does exactly the same. Therefore we can create a variable "day" that is an implication of three variables.

From Fig – 5 shows the box-plot of "day" vs. the target variable "cnt". Important observations from it are –

-> "weekend" and "workingday" have almost same means and higher than that of "holiday".

-> The inter-quartile range of “holiday” is higher when compared to both which indicates dispersion of “cnt” values is more for “holiday”

-> Upper quartiles (Q_3) of three categories are same while lower quartiles (Q_1) are different

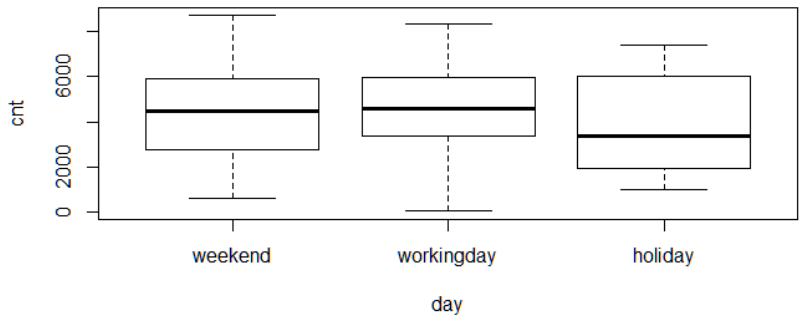

Fig – 5: “day” vs. “cnt” box-plot

B) In predictive model development, the best way to deal with date field in the dataset is to make feature engineering over it. In our dataset, we have “dteday” in yyyy-mm-dd format. On the other hand, we also have fields representing year and month as “yr” and “mnth” respectively. It will be meaningless if we carry out analysis either by letting the “dteday” column as it is or extract date from it, because extracted date would be just a series of repeating numbers from 1 to 28/29/30/31. The better alternative is to extract dates and group them week-wise for all months, which means first seven days of each month will be categorised “First Week” and so on. Later these categories are labelled 1, 2, 3, 4, 5 under “week_in_month” variable. With this step, all the parts of “dteday” will come into the dataset, so we can discard it.

Fig – 6 shows box-plot of “cnt” vs. “week_in_month”. Important observations from it are –

-> Inter-quartile ranges of first three weeks is same; fourth week has greater range while fifth has intermediate range

-> Means of all the weeks are almost same.

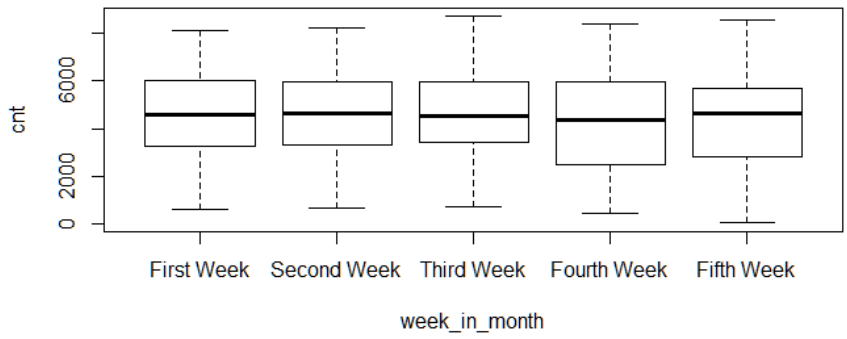

Fig – 6: “week_in_month” vs. “cnt” box-plot

2.1.5 Feature Selection

Feature selection aka Variable selection is a very important step in any machine learning model development as we decide how and which features (variables) in the given dataset would affect the predictive model that we are going to adopt. After feature engineering, we have obtained new categorical variables. In this section, we verify, which independent variables are considered to be part of model development.
There are several methods to do feature selection. Here we follow correlation analysis to check dependency among continuous variables (features) and ANOVA for to select categorical variables w.r.t target variable "cnt".

2.1.5.1 Correlation Analysis

Fig – 7 depicts correlation plot of Con_Var of the dataset. From this plot, it is clearly seen that – "temp" & "atemp"; "registered" & "cnt" are highly correlated. Because highly correlated variables cause multicolinearity problem in the model, one variable from each of the above set should be discarded from the main dataset. Between "temp" & "atemp", we discard "atemp" as it is comparatively more correlated to "cnt". And between "registered" & "cnt", we discard "registered", because it is highly correlated with "cnt". After "registered" is being discarded, "casual" alone does not contribute much to the analysis because "registered" & "casual" are summed up to "cnt". Therefore in our final dataset, continuous variables that we will have are - "temp", "hum", "windpseed", "cnt"

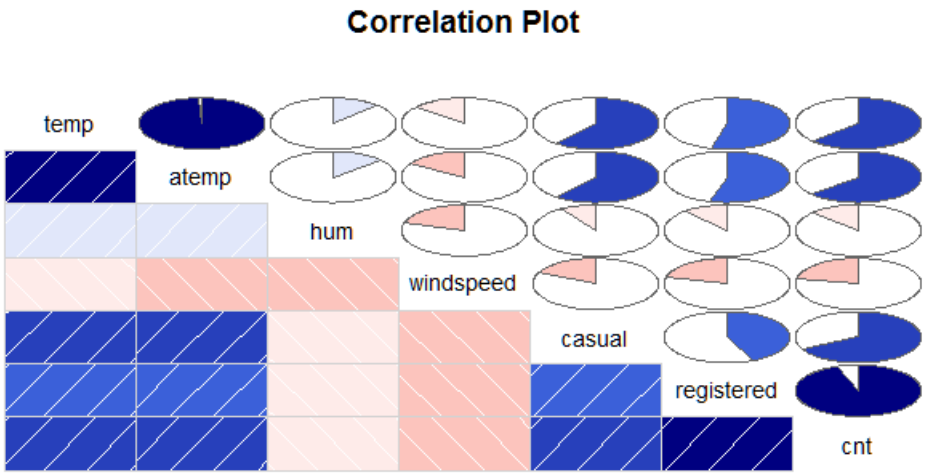

Fig – 7: Correlation plot of Con_Var

2.1.5.2 ANOVA Test

Now, let's look at the results of the ANOVA test of Cat_Var against "cnt", in Fig – 8. Here we are conducting n-way ANOVA in a simple way with the functions available in R & Python. The hypotheses of ANOVA are –
Null hypothesis (H_0): the mean of dependent variable is same for all the groups (levels)

Alternative hypothesis (H_1): the mean of dependent variable is not same for all the groups (levels)

The ANOVA test produces F value which is used to calculate Pr value. The Pr value tells whether or not to accept null hypothesis. If p<0.05, we reject null hypothesis, which means, mean is not same for all groups of independent variable. In Fig – 8, we find Df (degrees of freedom) which is equal to number of levels-1; Sum squared value; Mean squared value; F value and Pr of each independent variable and Significant codes at the end represent the ranges of Pr value that determine the association of each independent variable with predictor variable.

If 0 < Pr < 0.001, it is three starred – meaning strong association

0.001 < Pr < 0.01, it is two starred – meaning good association

0.01 < Pr < 0.05, it is one starred – meaning little association

0.05 < Pr < 0.1, it is a dot – meaning inconsiderable association

0.1 < Pr < 1, it is a single space – meaning no association

From the Pr values shown, all the variables have considerable association with "cnt". So we take all of them into our analysis.

```
> summary(ANO_Test)
              Df    Sum Sq   Mean Sq  F value  Pr(>F)
yr             1 879828893 879828893 1208.526 < 2e-16 ***
season         3 954775238 318258413  437.157 < 2e-16 ***
mnth          11 187311622  17028329   23.390 < 2e-16 ***
week_in_month  4  11098183   2774546    3.811 0.00449 **
day            2   6681627   3340814    4.589 0.01047 *
weathersit     2 185130981  92565491  127.147 < 2e-16 ***
Residuals    707 514708848    728018
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fig – 8: Summary of ANOVA test with significant codes

2.1.6 Dimensionality Reduction

After Correlation and ANOVA test analysis, the variables we are going to consider in our final dataset to feed to a model are –

Dependent Variable – "cnt"

Predictor Variables – "yr", "season", "mnth", "week_in_month", "day", "weathersit", "temp", "hum", "windspeed"

The discarded variables and reason associated –

-> "dteday" – year and month part this are given as separate columns in the original dataset and the date part is feature engineered to "week_in_month".

-> "weekday", "workingday", "holiday" – together feature engineered to "day".

-> "atemp" – highly correlated with "temp" and compared to "temp", it is more correlated to "cnt"

-> "casual", "registered" – these two sum up to main target variable "cnt" and "registered" is highly correlated with "cnt".

2.1.7 Feature Scaling

This is another important step in making the dataset uniform w.r.t scale. The model predictions will go wrong if the variables are of different scales. All other variables except "cnt" have come normalised along with the data. To make the scale of "cnt" on par with other variables, it should also be normalised. Following table gives the first few values of "cnt" before and after normalisation:

Index	"cnt" before normalisation	"cnt" after normalisation
1	985	0.11079153
2	801	0.08962264
3	1349	0.15266912
4	1562	0.17717441
5	1600	0.18154625
6	1606	0.18223654
7	1510	0.17119190

Table – 4 "cnt" before and after normalisation

2.2 Modelling

Our target variable is a continuous variable, so our predictive model should be a regression model. The following 4 regression methodologies are chosen for investigation so as to decide which is best suitable for our dataset and problem statement –

-> Decision Tree Regression
-> Random Forest Regression
-> Multiple – Linear Regression
-> KNN Regression

For training the regression model and testing the prediction values, we will divide our dataset into "train" and "test" in 8 to 2 ratio. We do this division by "simple random sampling" method. To evaluate the models, we will calculate error metrics that will signify which model is best suitable for our dataset and business problem.

2.2.1 Model Development

2.2.1.1 Decision Tree Regression (DTR) Model

In R, we use "*rpart*" function from "*rpart*" library to build DTR model on training data by choosing "cnt" as the target variable. This generates a set of rules based on which nodes are divided and tree is built. We use these set of rules to predict the test cases. Later we compare the predicted test case with the original values and see how close the values are. On the other hand we also evaluate the model by various error metrics. We use "*anova*" as the method of

regression in *rpart* as ours is a regression model. Fig – 9 shows the rules created by rpart function.

```
> DT_Reg
n= 584

node), split, n, deviance, yval
      * denotes terminal node

 1) root 584 29.25482000 0.5210292
   2) temp< 0.4327175 240  6.94519900 0.3540641
     4) yr=0 125  1.65124700 0.2569793
       8) season=1,2 87  0.37858930 0.1991976 *
       9) season=4 38  0.31716650 0.3892690 *
     5) yr=1 115  2.83513800 0.4595910
      10) temp< 0.2804165 33  0.32619130 0.3091069 *
      11) temp>=0.2804165 82  1.46090300 0.5201517
        22) season=1 34  0.14341450 0.4438598 *
        23) season=2,4 48  0.97941770 0.5741918
          46) hum>=0.7525 8  0.23137050 0.3850236 *
          47) hum< 0.7525 40  0.40451500 0.6120254 *
   3) temp>=0.4327175 344 10.95122000 0.6375165
     6) yr=0 160  1.52282400 0.4833051
      12) weathersit=3 7  0.01252417 0.2599435 *
      13) weathersit=1,2 153  1.14508900 0.4935242 *
     7) yr=1 184  2.31472200 0.7716135
      14) hum>=0.738125 34  0.53422340 0.6639405 *
      15) hum< 0.738125 150  1.29697400 0.7960193 *
```

Fig – 9: Rules created by "*rpart*"

In Python, we use *DecisionTreeRegressor* function imported from *sklearn.treeI.* We have specified *max_depth = 2* as an argument and given the training data to fit into the model. It takes on the default values for the arguments if we don't specify any. We let the default criterion be "mse" for regression. Fig – 10 below shows the function and its default argument values.

```
DecisionTreeRegressor(criterion='mse', max_depth=2, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

Fig – 10: *DecisionTreeRegressor* function with default arguments

2.2.1.2 Random Forest Regression (RFR) Model

In R we use "*randomForest*" from *randomForest* library to build DTR model on training data by choosing "cnt" as the target variable. We use 100 as number of trees to be there in the model. "*randomForest*" generates an object shown below:

```
> RF_Reg

Call:
 randomForest(formula = cnt ~ ., data = train, ntree = 100)
               Type of random forest: regression
                     Number of trees: 100
No. of variables tried at each split: 3

          Mean of squared residuals: 0.006043536
                    % Var explained: 87.94
```

Fig – 11: object created by "*randomForest*"

We can extract rules generated by "*randomForest*" function, and through which we can obtain rule metrics. We feed the above object to predict the test cases for RFR. Then we calculate error metrics to see how good our RFR is. Fig – 12 below shows first 3 rules created.

```
> ReadRules[1:3,]
[1] "yr %in% c('0') & season %in% c('1','2') & mnth %in% c('1') & week_in_month %in% c('4','5') & temp<=
0.42375 & windspeed<=0.1185685"
[2] "yr %in% c('0') & season %in% c('1','2') & mnth %in% c('1') & week_in_month %in% c('4','5') & temp<=
0.42375 & windspeed>0.1185685"
[3] "yr %in% c('0') & season %in% c('1','2') & mnth %in% c('2','3','4') & week_in_month %in% c('4','5')
& temp<=0.42375"
```

Fig – 12: first 3 rules created in random forest regression

In Python, we use *RandomForestRegressor* function imported from *sklearn.ensemble.* We have specified *max_depth = 2* and *n_estimators = 100,* number of trees as arguments and given the training data to fit into the model. It takes on the default values for the arguments if we don't specify any. We let the default criterion be "mse" for regression. Fig – 13 below shows the function and its default argument values.

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

Fig – 13: *RandomForestRegressor* function with default arguments

2.2.1.3 Multiple-Linear Regression (MLR) Model

In R, we use a base function *lm* for linear regression. Linear regression assumes the following among the variables –

-> Linear relationship

-> Multivariate normality

-> No or little multi-collinearity

-> No auto-correlation

During the Feature Selection stage, with the help of *Correlation Analysis* and *ANOVA Test*, we tried to deal with one or more conditions specified above and discarded the variables that do not fit into the predictive modelling. Now we will look at the summary of linear regressor and check the values "Residual standard error", "Multiple R-squared", "Adjusted R-Squared", "F-statistic" and "p-value" to estimate whether our set of variables hold validity to fit in an MLR model. As shown in Fig – 14 below, the above said parameter values satisfy an MLR model's requirements for the chosen variables.

```
> summary(Lin_Reg)

Call:
lm(formula = cnt ~ ., data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-0.41677 -0.04450  0.00902  0.05553  0.26880

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     0.226477   0.032877   6.889 1.53e-11 ***
yr1             0.230845   0.007668  30.103  < 2e-16 ***
season2         0.123525   0.023837   5.182 3.07e-07 ***
season3         0.106748   0.027765   3.845 0.000135 ***
season4         0.175618   0.023254   7.552 1.76e-13 ***
mnth2           0.010035   0.019228   0.522 0.601947
mnth3           0.042589   0.022363   1.904 0.057372 .
mnth4           0.027106   0.033061   0.820 0.412638
mnth5           0.060530   0.035238   1.718 0.086396 .
mnth6           0.030150   0.036878   0.818 0.413963
mnth7          -0.012806   0.040994  -0.312 0.754869
mnth8           0.022149   0.039659   0.558 0.576732
mnth9           0.100726   0.034607   2.911 0.003752 **
mnth10          0.064088   0.031317   2.046 0.041183 *
mnth11         -0.016489   0.030418  -0.542 0.587981
mnth12         -0.006973   0.024053  -0.290 0.772008
week_in_month2  0.002898   0.010848   0.267 0.789428
week_in_month3  0.014939   0.010865   1.375 0.169685
week_in_month4 -0.018159   0.010956  -1.657 0.098013 .
week_in_month5 -0.039734   0.016286  -2.440 0.015010 *
day2            0.022438   0.008357   2.685 0.007469 **
day3           -0.062746   0.021605  -2.904 0.003827 **
weathersit2    -0.051025   0.010023  -5.091 4.88e-07 ***
weathersit3    -0.200094   0.026008  -7.693 6.54e-14 ***
temp            0.533228   0.055837   9.550  < 2e-16 ***
hum            -0.222001   0.039217  -5.661 2.41e-08 ***
windspeed      -0.351075   0.057349  -6.122 1.75e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08976 on 557 degrees of freedom
Multiple R-squared:  0.8496,    Adjusted R-squared:  0.8426
F-statistic:   121 on 26 and 557 DF,  p-value: < 2.2e-16
```

Fig – 14: summary of MLR

In Python, we use "*sm.OLS*" function imported from "*statsmodels.api*". As parts of the arguments, we simply need to the training data. Then we predict the test cases and compare them with the original test cases via various error metrics.

2.2.1.4 KNN Regression (KNN-R) Model

In R, we use "*knnreg*" function from "*caret*" library to build a KNN Regression model. We give training data and number of neighbours (k) to be considered as arguments. By trial and error, value of 7 for k has given best predictions.

In Python, we use "*KNeighborsRegressor*" function imported from *"sklearn.neighbors*". We provide *n_neighbors* as argument to this function and remaining are taken as their default and we fit this to the train data. With the help outcome of this function, we predict the test values and compare them with the original test data values to evaluate the model.

3. Conclusion

3.1 Model Evaluation

In order to evaluate a regression model, there are various error metrics. Here we will consider four important metrics for a comparative study. They are – mean absolute error (mae), mean squared error (mse), mean absolute percentage error (mape) and root mean squared error (rmse). We can take any one metric as a reference and compare the performance of each model and finally choose the best model. Table – 5 & Table – 6 gives the values of all four error metrics of all models.

Important point to note here is, these values will change each time when we run the train and test lines of code because the random samples will be changed.

In R, we have "*regr.eval*" function from *DMwR* library which calculates all the above error metrics in one shot. The syntax is

$$regr.eval(trues, preds, stats = if\big(is.null(train.y)\big)\ c(\text{"mae"}, \text{"}mse\text{"}, \text{"}rmse\text{"}, \text{"}mape\text{"})\ else\ c(\text{"mae"},\ \text{mse"}, \text{rmse"}, \text{"mape"}, \text{"nmse"}, \text{"nmae"}),\ train.y = NULL)$$

Error Metric / Model	MAE	MSE	MAPE	RMSE
DTR	0.07523593	0.01003967	0.17539468	0.10019813
RFR	0.04972769	0.00445285	0.11703345	0.06672968
MLR	0.06296960	0.00713127	0.14784436	0.08444685
KNN-R	0.06429117	0.00708266	0.15627407	0.08415858

Table – 5: All four error metrics of four models predicted using R

In Python, we have "*mean_absolute_error*" and "*mean_squared_error*" functions in "*sklearn.metrics*" library. For the remaining two metrics we define the functions as –

```
#Define MAPE
def mape(predict, true):
    mape = np.mean(np.abs((true-predict)/true))
    return mape
#Define RMSE
def rmse(predict, true):
    rmse = np.sqrt(((true - predict) ** 2).mean())
    return rmse
```

Error Metric / Model	MAE	MSE	MAPE	RMSE
DTR	0.09424492	0.01426813	0.21387388	0.11944929
RFR	0.08341095	0.01134943	0.17915757	0.10653370
MLR	0.07825860	0.01027569	0.18111119	0.10136910
KNN-R	0.07920814	0.01012059	0.16184634	0.10060115

Table – 6: All four error metrics of four models predicted using Python

3.2 Model Selection

From Table – 5 and Tables – 6, we can see values of four error metrics of the models we developed. Random Forest Regression model outperforms all other model w.r.t all error metrics in R. When it comes to Python, RFR, MLR and KNN-R are equally good across all error metrics.