# **Bike Renting**

# 

**Bharti Sharma**

**Table of Contents**

[1. Introduction 4](#_Toc529012078)

[1.1 Problem Statement 4](#_Toc529012079)

[1.2 Data 4](#_Toc529012080)

[2. Methodology 5](#_Toc529012081)

[2.1 Exploratory Data Analysis 5](#_Toc529012082)

[2.3 Data Preprocessing 19](#_Toc529012083)

[2.3.1 Missing Values 19](#_Toc529012084)

[2.3.2 Outlier Analysis: 25](#_Toc529012085)

[2.3.4 Feature Scaling 33](#_Toc529012086)

[2.4 Sampling 34](#_Toc529012087)

[2.5 Model Development 35](#_Toc529012088)

[2.5.1 Model Evaluation 35](#_Toc529012089)

[2.5.2 Error Measure 36](#_Toc529012090)

[2.5.3 Confusion Matrix 36](#_Toc529012091)

[2.5.4 Cross Validation 36](#_Toc529012092)

[2.6 Model Selection 37](#_Toc529012093)

[2.6.1 Linear Regression Model 37](#_Toc529012094)

[2.6.2 Decision Tree 41](#_Toc529012095)

[2.6.3 Random Forest 43](#_Toc529012096)

[2.6.4 KNN 44](#_Toc529012097)

[2.6.5 Model Selection 45](#_Toc529012098)

[2.7 Generate Output File 46](#_Toc529012099)

[3.Conclusion 46](#_Toc529012100)

3.1[Factors Affecting Count Of Bike renting](#_Toc529012102)

[3.2 Challenges Faced 50](#_Toc529012103)

[Appendix – A 51](#_Toc529012104)

[Outlier Analysis Using Boxplot Method 51](#_Toc529012105)

[Density plot and Histogram of variables 59](#_Toc529012107)

[Histogram showing Mean and skewed data of variables 62](#_Toc529012108)

[Appendix – B 68](#_Toc529012109)

[Python Code 68](#_Toc529012110)

[Appendix – C 85](#_Toc529012120)

[R-Code 85](#_Toc529012121)

# **1. Introduction**

## **1.1 Problem Statement**

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

As per the problem statement the aim of the project is to predict bike rental count on daily based on the environmental and seasonal settings.

We are applying some statistical techniques and machine learning algorithms on a public dataset that has features for analysing the predictions. We are developing an algorithm to predict bike rental count on daily based on the environmental and seasonal settings.

## **1.2 Data**

Our task is to build regression model which will predict bike rental count on daily based on the environmental and seasonal settings.

The details of data attributes in the dataset are as follows –

**Independent Variables**

instant: Record index

* dteday: Date
* season: Season (1:springer, 2:summer, 3:fall, 4:winter)
* yr: Year (0: 2011, 1:2012)
* mnth: Month (1 to 12)
* holiday: weather day is holiday or not (extracted fromHoliday Schedule)
* weekday: Day of the week
* workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
* weathersit: (extracted fromFreemeteo)
  1. 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  2. 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  3. 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  4. 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
* temp: Normalized temperature in Celsius.

The values are derived via (t-t\_min)/(t\_max-t\_min),

t\_min=-8, t\_max=+39 (only in hourly scale)

* atemp: Normalized feeling temperature in Celsius.

The values are derived via (t-t\_min)/(t\_max-t\_min),

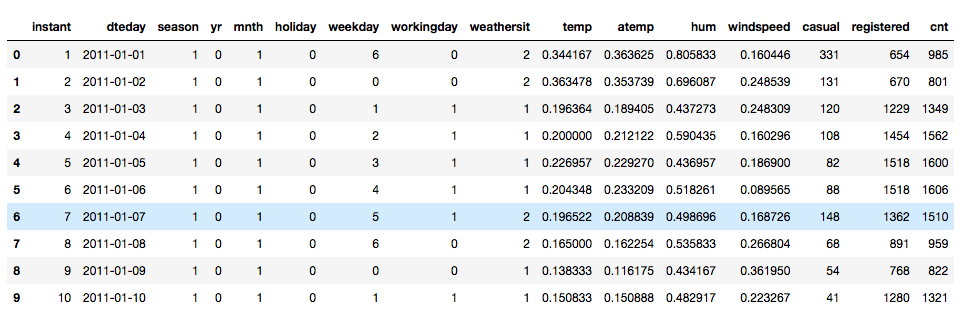
t\_min=-16, t\_max=+50 (only in hourly scale)

* hum: Normalized humidity. The values are divided to 100 (max)
* windspeed: Normalized wind speed. The values are divided to 67 (max)

**Dependent Variables**

* casual: count of casual users
* registered: count of registered users
* cnt: count of total rental bikes including both casual and registered

Given below is a sample of the dataset that we are using to predict the bike rental count.



# **2. Methodology**

## **2.1 Exploratory Data Analysis**

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

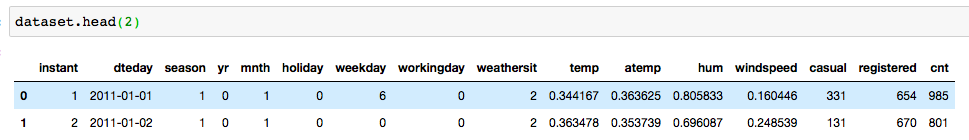
We have performed some steps for exploratory data analysis in python and R code to understand data in a better way and also to get a glimpse of dataset.

I have shared some screenshots from python code. Entire code has been shared in Appendix B and C.

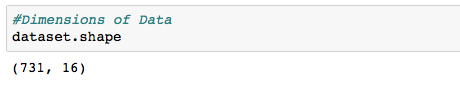
* Loaded the dataset in notebook.



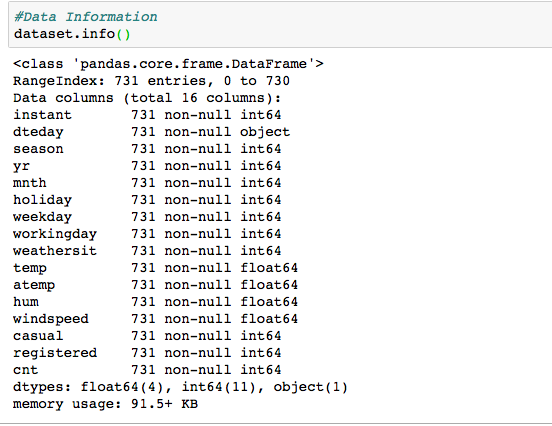
* To take a closer look at the data took help of “ .head()”function of pandas library which returns first five observations of the data set.Similarly “.tail()” returns last five observations of the data set.



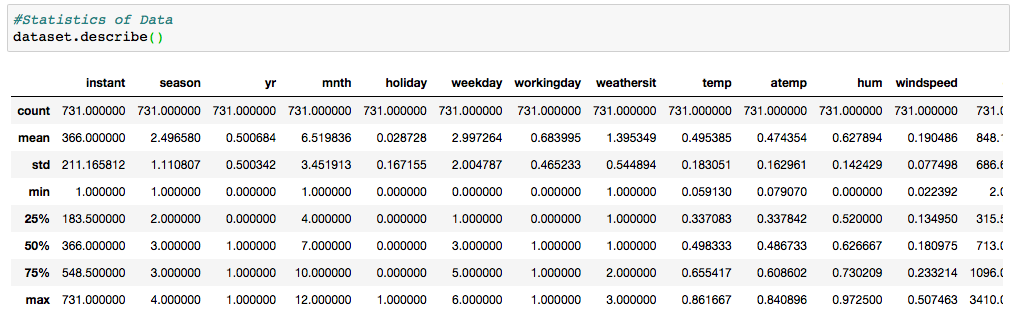
* We found out the total number of rows and columns in the data set using “.shape”.



* Dataset comprises of 731 observations and 16 characteristics. Out of which three are dependent variable and rest 13 are independent variables
* Dependent variable ‘cnt’ is sum of two dependent variables ‘registered’ and ‘casual’ as per problem statement.
* We will know the columns and their corresponding data types, along with finding whether they contain null values or not using “info()” function .



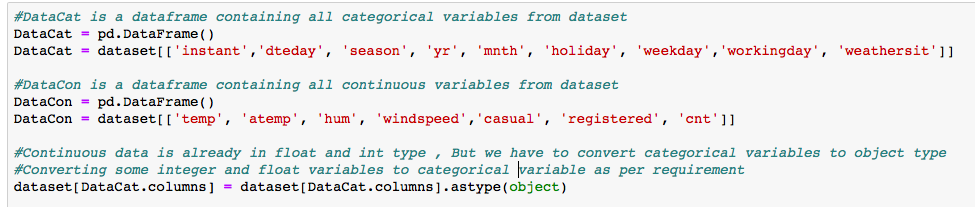
* Data has only object ,float and integer values. Since some of the variables should be categorical for example : Month, year , Seasons etc, So as per analysis we need to convert them from continuous to categorical variables. Since they are not carrying any continuous type of information whereas they are carrying a categorical type of information.
* We will use “describe()” function in pandas which helps in getting various summary statistics. This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.



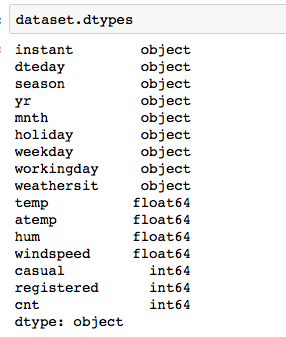
* Here as you can notice mean value is less than median value of some of column which is represented by 50%(50th percentile) in index column.There is notably a some difference between 75th %tile and max values of predictors.Thus observations suggests that there are some Outliers present in our data set.
* Now after observing the data dimensions, info and statistics. We have seen we have two type of independent variables int , object and float. And also we have to change their types as per problem requirement, So we are dividing categorical and numerical/continuous data in two groups.

1. DataCat (categorical variables)
2. DataCon (continuous variables)

* Continuous data is already in float and int type , But we have to convert categorical variables to object type
* Converted some integer and float variables to categorical variable as per requirement

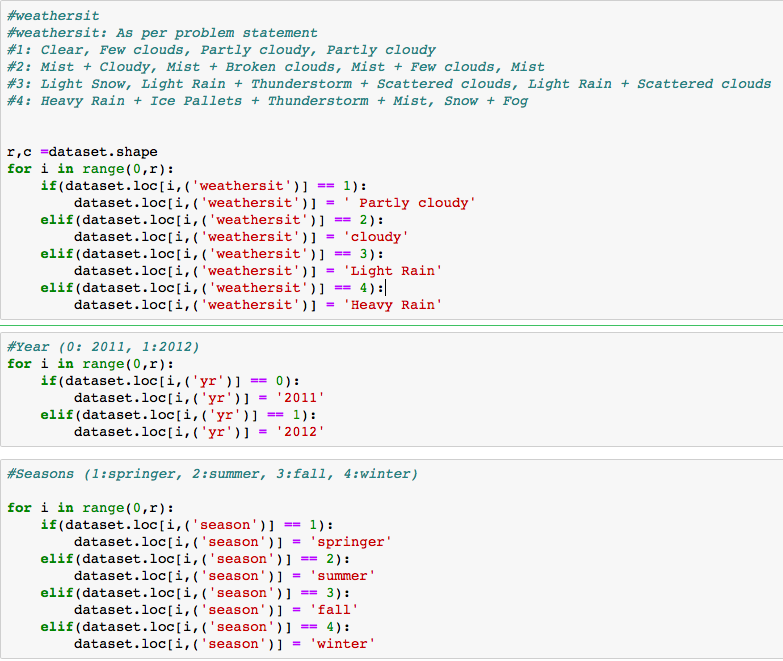


* After conversion checking datatypes for all variables in dataset.



* In the problem statement we have given the names of categories present in dataset.

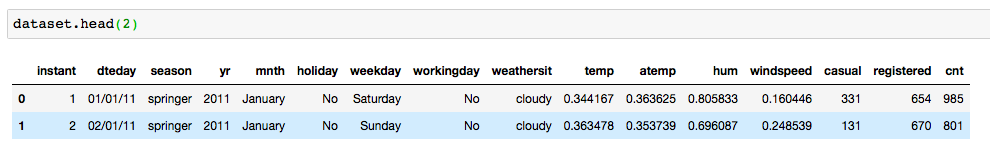
For better understanding we are assigning categories names to elements present under all categorical variables.







* After adding the categories to categorical variables of dataset . we have got our dataset as below.



* So for continuous variables we have got normalized data. With the help of which it will be difficult for us to find correlation and outliers present in the dataset for continuous variables. So we will be converting normalized form of variables to actual values by using the reverse formula which has been used to normalize data as per problem statement .
* **temp:** Normalized temperature in Celsius.

The values are derived via

(t-t\_min)/(t\_max-t\_min),t\_min=-8, t\_max=+39

After applying above formula we have added resultant values under new variable named dataset['ActualTemp'] .

* **atemp**: Normalized feeling temperature in Celsius.

The values are derived via

(t-t\_min)/(t\_max-t\_min), t\_min=-16, t\_max=+50

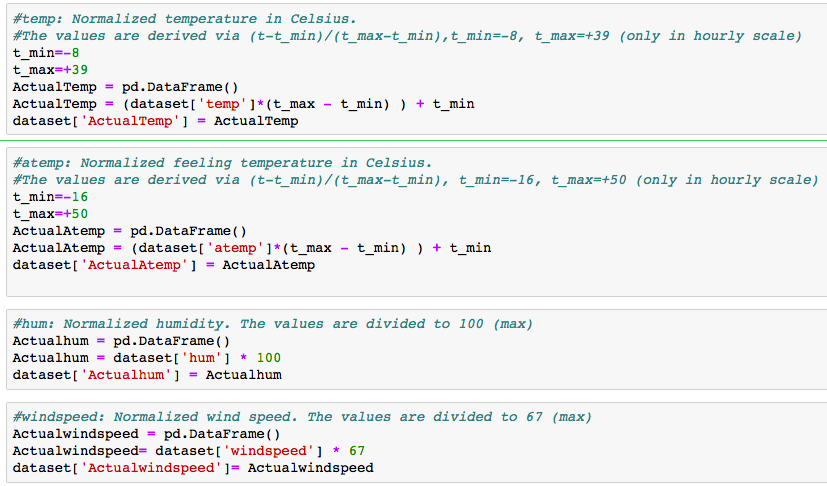
After applying above formula we have added resultant values under new variable named dataset[‘ActualAtemp’] .

* **hum**: Normalized humidity. The values are divided to 100 (max)

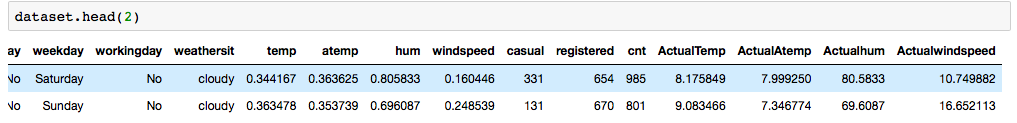
After applying above formula we have added resultant values under new variable named dataset[‘Actualhum’] .

* **windspeed**: Normalized wind speed. The values are divided to 67 (max)

After applying above formula we have added resultant values under new variable named dataset[‘Actualwindspeed’] .



* Dataset after converting the normalized values to their actual values .



We have got the data after doing manipulations on categorical variables. Python and R Code has been shared in Appendix – B,C

**2.2 Visualizations**

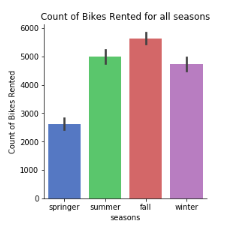
Data visualization refers to the graphical representation of information and data. By using visual elements like charts, graphs, and maps. Data visualization is an accessible way to see and understand trends, outliers, and patterns in data.

As per our problem statement, we are going to analyse relationship of target variable with categorical and continuous variables of given dataset to get an initial insights of data. So that before going for data pre-processing we have an idea about initial impact of different factors from all variables. For detailed observations we will be doing data pre-processing and model development.

Graph -1

Bar graph between Season and Count of bikes rented



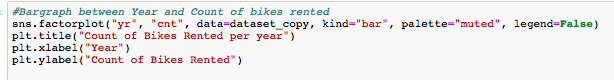


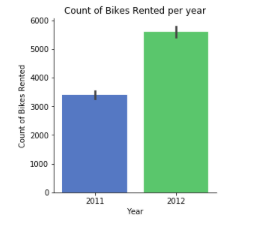
Observation:

In fall bikes have been rented more comparatively all seasons, Whereas in spring there were much less number of bikes rented.

Graph -2

Bar graph between Year and Count of bikes rented







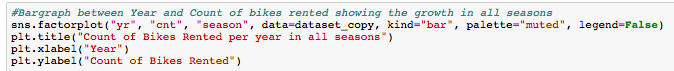
Observation:

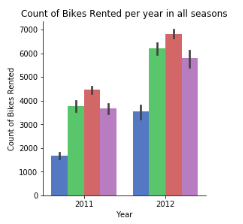
In year 2012 the growth has been good from 2011.

Growth percentage of bikes rented in 2012 as per 2011 is 64.87 %.

Graph -3

Bar graph between Year and Count of bikes rented showing ratio of growth in all seasons



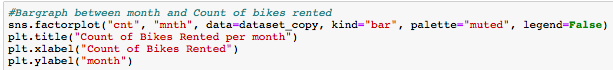


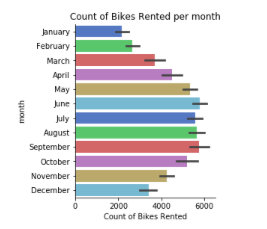
Observation:

The ratio of bike renting among all seasons of 2012 is comparatively same as of 2011.

Graph -4

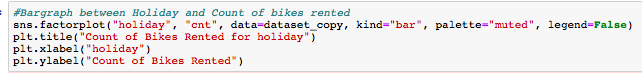
Bar graph between Month and Count of bikes rented

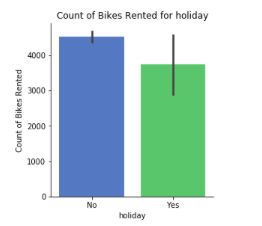




Graph -5

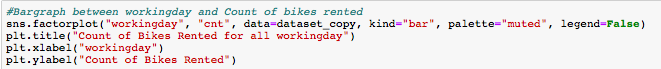
Bar graph between holiday and Count of bikes rented





Graph -6

Bar graph between workinday and Count of bikes rented



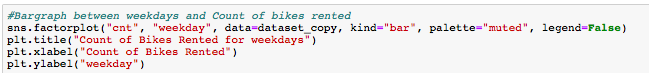


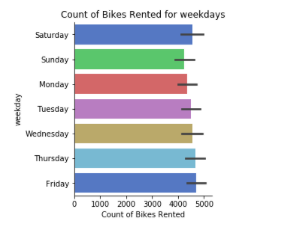
Observation:

From graph -5 and 6 , we can say that people rent comparatively more bikes on working day than on holiday.

Graph -7

Bar graph between weekdays and Count of bikes rented

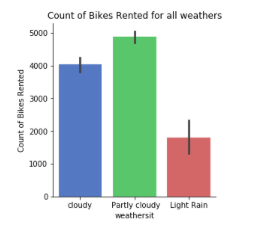




Graph -8

Bar graph between weather and Count of bikes rented





Observation:

In partly cloudy weather bikes have been rented more comparatively all weathers, Whereas in Light Rainy weather there were much less number of bikes rented.

Graph -9

Voilin plot of Count of bikes rented

../../Screen%20Shot%202018-11-26%20at%206.03.50%20PM.png



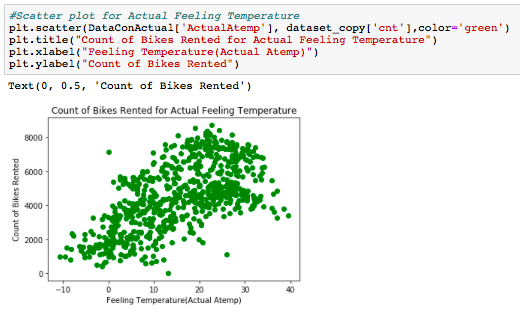
Observation:

The central dot representing the median average count of bikes rented per day over both the years is between 4000 to 4500.

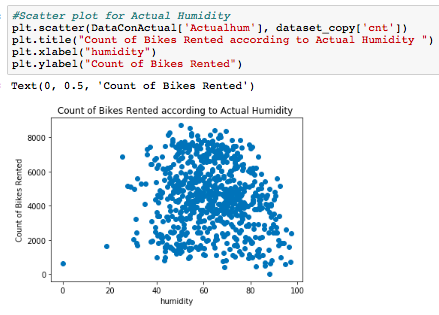
Graph -10



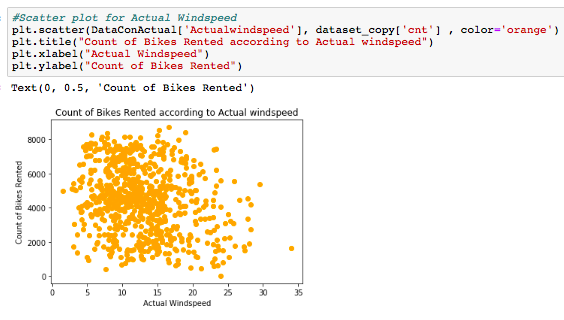
Graph -11



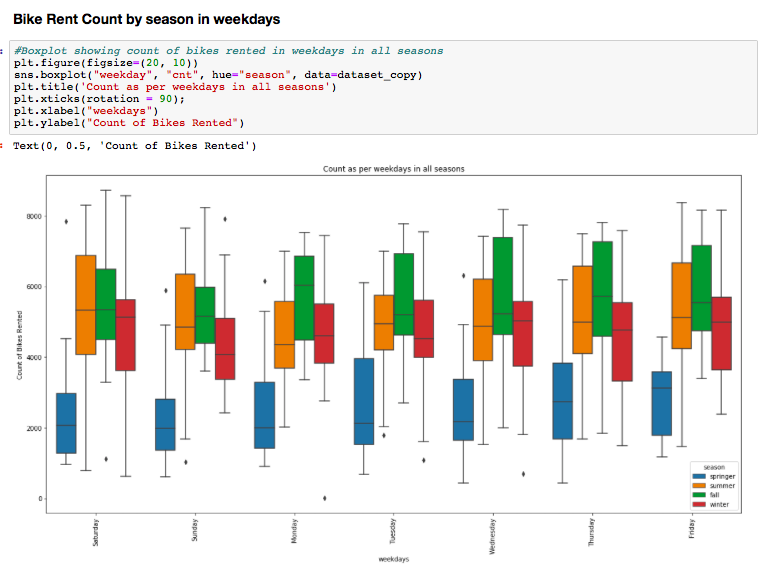
Graph -12



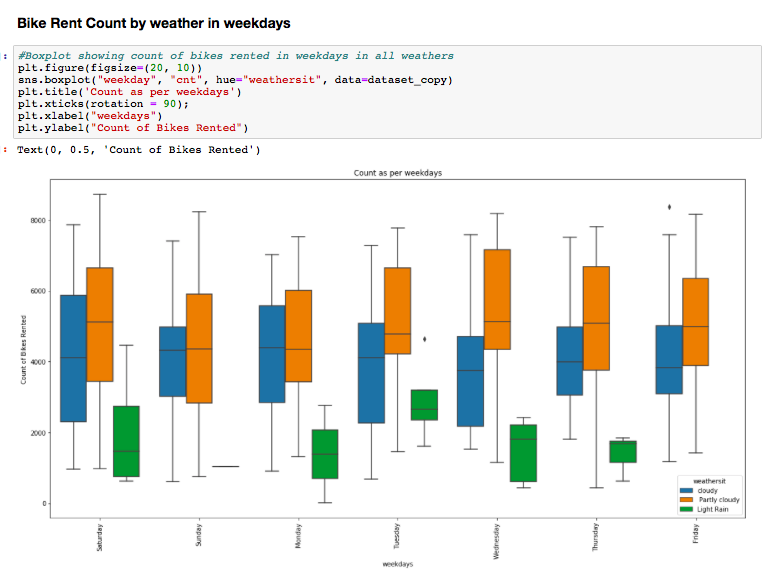
Graph -13



Graph -14



Graph -15



**Box Plot**

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution.

* The box plot (a.k.a. box and whisker diagram) is a standardized way of displaying the distribution of data based on the five number summary:

Minimum

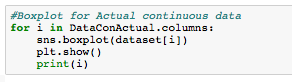
First quartile

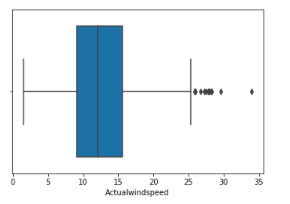
Median

Third quartile

Maximum.

In the simplest box plot the central rectangle spans the first quartile to the third quartile (the interquartile range or IQR).





A segment inside the rectangle shows the median and “whiskers” above and below the box show the locations of the minimum and maximum.

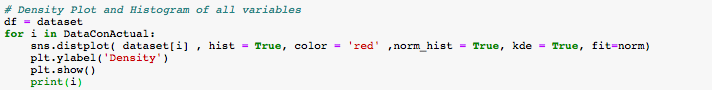
The above figure shows boxplot for Actual windspeed. ,there are outlier present for this. We can also observe the particular outlier here.

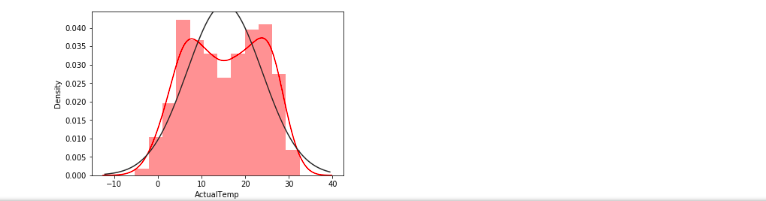
Similarly we have plotted boxplot for all variables. Code and figures for them has been shared in Appendix- A.

**Distribution Plot**

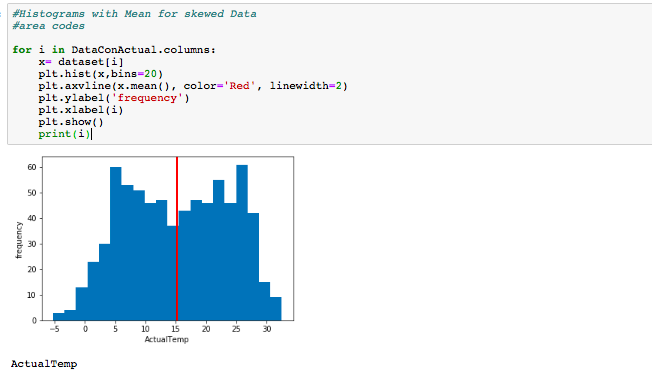
We will look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

we have plotted the probability density Kernel density estimate (kde) is a quite useful tool for plotting the shape of a distribution. functions of all the continuous variable properties we have available in the data . The red lines indicate Kernel Density Estimations (KDE)1 of the variable. The black lines represent the normal distribution. So as you can see in the ﬁgure most variables either very closely, or somewhat imitate the normal distribution. Those plots can be viewed in the Appendix – A.





**Histogram for Skewed data with Mean**



Histogram for calculated actual variables have been plotted. All the plots have been shared in Appendix – A.

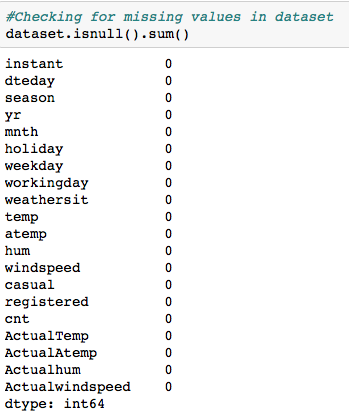
**2.3 Data Preprocessing**

Data preprocessing is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data-gathering methods are often loosely controlled, resulting in out-of-range values (e.g., Income: −100), impossible data combinations (e.g., Sex: Male, Pregnant: Yes), missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Often, data preprocessing is the most important phase of a machine learning project.

**2.3.1 Missing Values**

Missing Value treatment becomes important since the data insights or the performance of your predictive model could be impacted if the missing values are not appropriately handled

For our data first of all we will check for Missing values and then we will impute those values by checking suitable method.



We have observed that missing value are not present in our dataset.

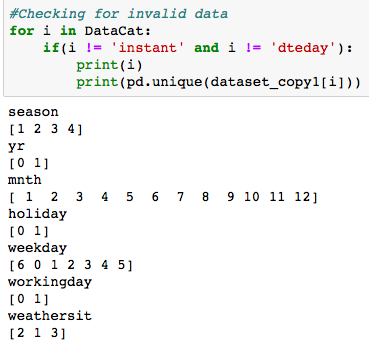
**Invalid Data**

Invalid data can be found in dataset sometimes due to incorrect user input / some invalid enteries ex. In month variables if we have got 0 entered ,whereas the range follow from (1-12) . So it will be invalid entry .

Now we are checking for invalid entries in some variables by using unique() function

From pandas.

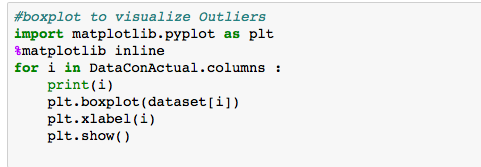
As we have observed that for all variables the values are as per the given range in problem statement. So we can say that we do not have any invalid data in our dataset.

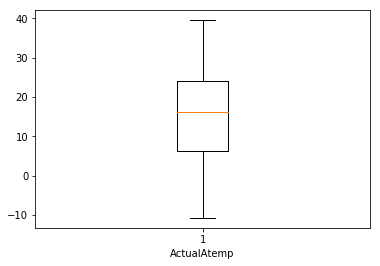
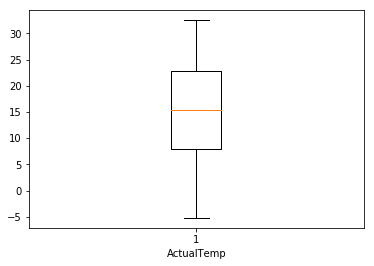


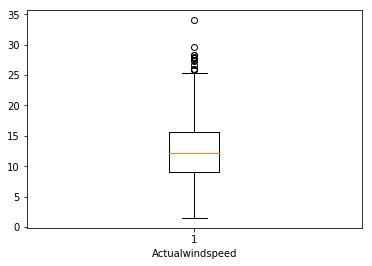
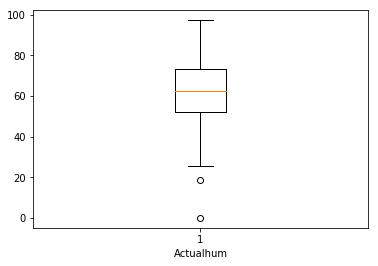
**2.3.2 Outlier Analysis:**

Now we have to check if there are outliers present in dataset for continuous variables. As we have already divided dataset in two categories one is a data frame ’DataCat’ for categorical variables and another is ‘DataCon’ for continuous variables. Here we will check outliers present in continuous dataset. Since categorical data do not have outliers present.

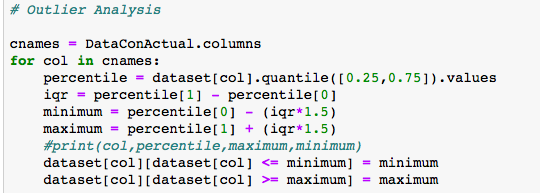
I have used for loop to plot boxplot for all continuous variables.



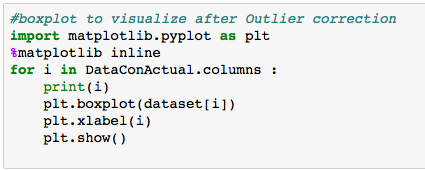


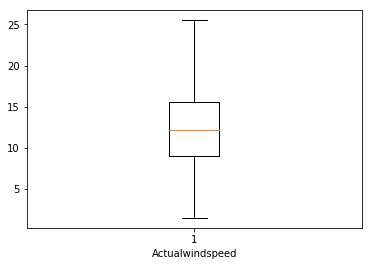
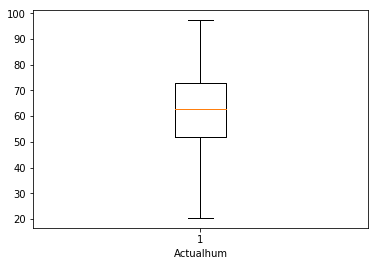


As we can see there are no outliers present for ActualTemp and ActualAtemp. But for Actualhum and ActualWindspred there are outliers present.So we will correct outliers present. Here we are replacing values which are greater than maximum by maximum value and minimum by minimum value as per given in below code.



Now we can check again if we have replaced outliers correctly by plotting boxplots again.





As per this there are no outliers present now in dataset.

**2.3.3 Feature Selection**

Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. There are several methods of doing that. Below we have used some techniques to perform features selection.

* For Continuous we will use correlation analysis
* For Categorical we will use Chi square test

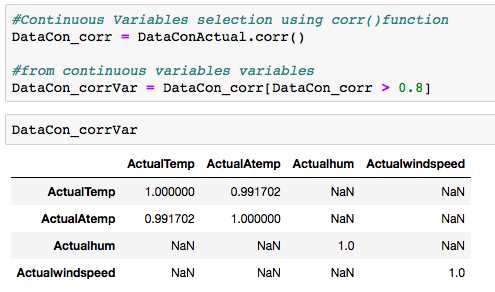
As we know there should be no dependency between independent variables. Also we have separated continuous and Categorical Variables in two groups as Continuous Data (DataCon) , Categorical Data (DataCat). So that we can apply individual suitable techniques for both the groups For Continuous we will use correlation analysis and For Categorical we will use Chi square test.

**->Feature Selection From Continuous Variables**

* Python has a visualization library ,[Seaborn](https://seaborn.pydata.org/) which build on top of matplotlib. It provides very attractive statistical graphs in order to perform both [Univariate](http://www.statisticshowto.com/univariate/) and [Multivariate analysis](http://www.camo.com/multivariate_analysis.html).
* To use linear regression for modelling,its necessary to remove correlated variables to improve your model.One can find correlations using pandas “.corr()” function and can visualize the correlation matrix using a heatmap in seaborn.
* We have heatmap plot to have a vision about overall dataset, We will explore correlation in feature selection after dividing categorical and continuous variables in detail.

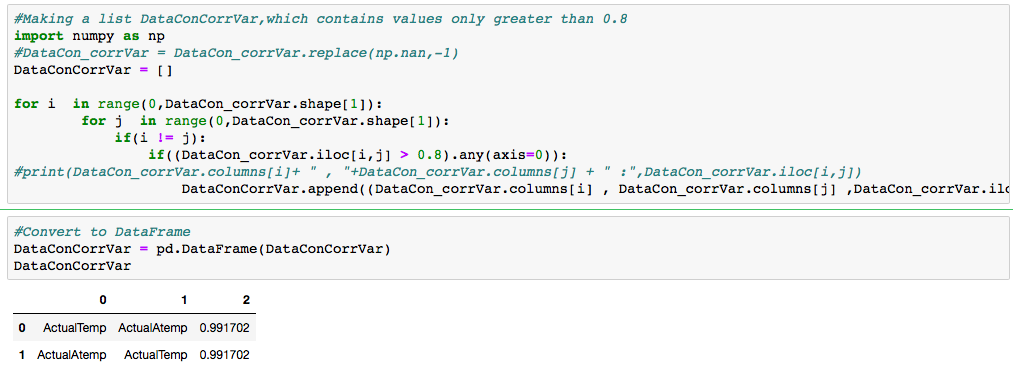


For continuous data we are using corr() function here to check the correlation among the continuous variables.



As we have got some nan values So we are removing them to find a clear dataframe of variables which are highly correlated. Which we can see in dataframe.

DataCon is representing continuous data here. If correlation is more than 0.8 than we have kept those variables in DataConcorrVar .



As we have seen that ActualAtemp and ActualTemp are highly correlated. So we can drop one of them . Here we are dropping atemp from dataset which is normalized form of ActualAtemp(which we have converted to actual values in EDA)

../Screen%20Shot%202018-11-24%20at%201.46.28%20PM.png

**-> Feature Selection From Categorical Variables**

For Categorical we will use Chi square test. As we know there should be no dependency between independent variables. The chi square test tests the null hypothesis that the categorical data has the given frequencies.

The Chi-Squared test is a statistical hypothesis test that assumes (the null hypothesis) that the observed frequencies for a categorical variable match the expected frequencies for the categorical variable. The test calculates a statistic that has a chi-squared distribution, named for the Greek capital letter Chi (X) pronounced “ki” as in kite.

As we have multiple independent categorical variables and our target variable is continuous. So we are not able to find out dependency between independent and dependent variables with the help of chi square test.

Here we are going to find dependency among independent categorical variables

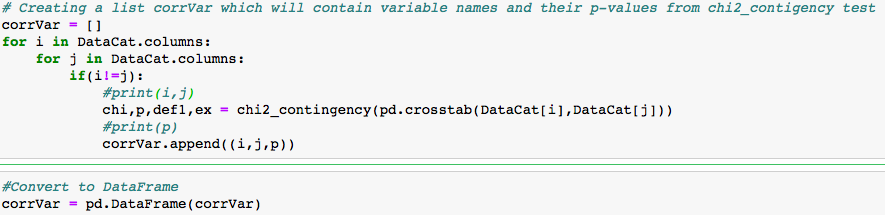
With the help of chi square test.

* If p-value < 0.05 : significant result, reject null hypothesis (H0), dependent.
* If p-value > 0.05 : not significant result, fail to reject null hypothesis (H0), independent.

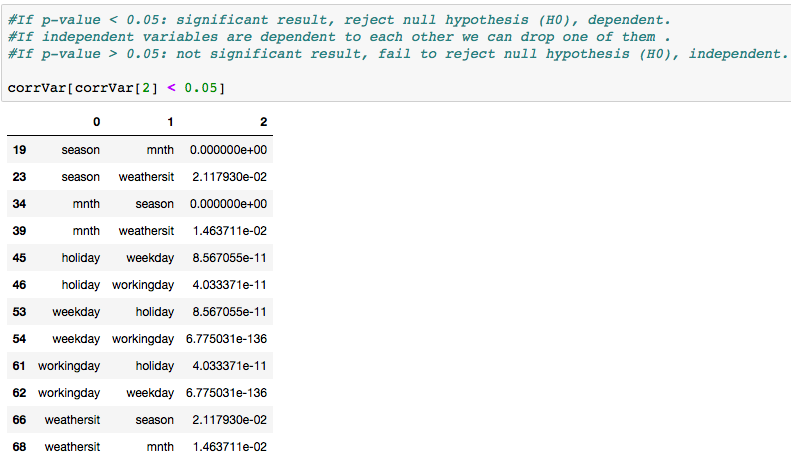
So if we have p-value < 0.05, we can drop one of the compared variables , to reduce collinearity in model.

As in the below screenshot we have performed chi square test on two variables in which p-value is less than 0.05 ,So we can drop one variable of them.Since they are carrying same kind of data.Similarly we are going to check for all categorical variables. Untill we get all independent variables.

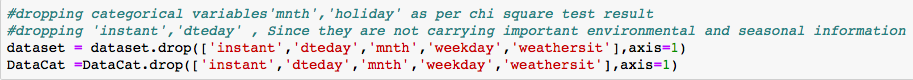
* Creating a list corrVar which will contain variable names and their p-values from chi2\_contigency test.



* After chi2\_contigency test we are selecting variables with p-values less than 0.05.So that we can drop correlated variables and prevent our model from multicollinearity effect.

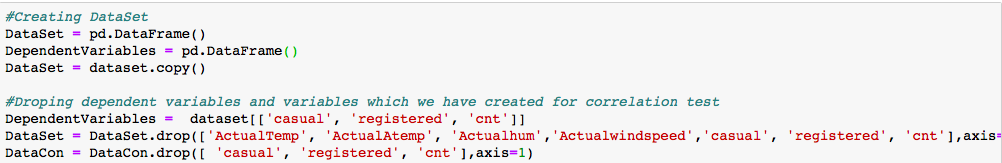


* dropping categorical variables'mnth','holiday' as per chi square test result
* dropping 'instant','dteday' , Since they are not carrying important environmental and seasonal information to the model as per problem statement.

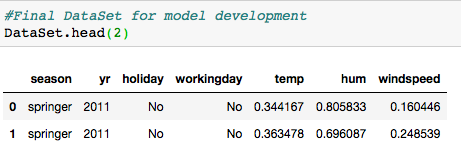


* Copying dataset to DataSet (new dataframe) for model implementation with required variables.

Dependent variables have been copied to new dataframe named ‘DependentVariables’



We have got below DataSet (containing all independent variables) after applying feature selection techniques for dimension reduction.



### 2.3.4 Feature Scaling

Feature scaling is the method to limit the range of the data. So that they can be compared on common ground.It can be performed only on continuous variables.

There are two methods of feature scaling:

* Normalization
* Standardization / Z-score

Normalization is the process of scaling individual samples to have unit norm. This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.

Standardization works well if the data is uniformly distributed. As dataset for problem is not uniformly distributed as we have already seen in graphs that we have skewed data in our dataset. So as per requirement we have applied normalization on dataset to do feature scaling.

Since we already have continuous data given in normalised form .There is no need for normalization.

We are passing the same continuous data to our model.



We are plotting histogram for normality check of the data .

### ../Screen%20Shot%202018-11-24%20at%202.06.47%20PM.png

### ../Screen%20Shot%202018-11-24%20at%202.07.43%20PM.png

### All plots have been shared in appendix-A.

### 2.4 Sampling

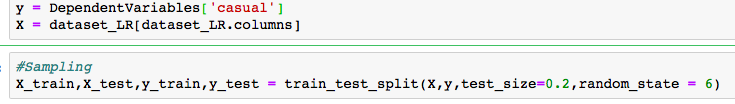
Sample is the subset of the population. The process of selecting a sample is known as sampling. Number of elements in the sample is the sample size.

There are different sampling techniques available like Simple Random Sampling, Stratified sampling, Systematic sampling, Cluster Sampling.

Here we are using simple random sampling method by using ‘train\_test\_split’

Function , which helps to divide dataset into two half’s, one is trained data and rest is test data.

We are not using stratify sampling method , Since stratified sampling method needs to have categorical dependent/target variable. Which results into a classification problem. But here we have continuous dependent variable. So it is a regression problem.That is why we using simple random Sampling method.



Here we have done sampling for ‘casual’ dependent variables.Similarly we have used for ‘registered’ also.

### 2.5 Model Development

In our early stages of analysis during pre-processing we have processed our dataset. So that we can get most accurate predictions from our regression model. Now going ahead both dependent and independent variables are in shape for our model production. Hence, we need to analyse the data sets and generate model on top of processed dataset.

Dependent variables, in our dataset are ‘cnt’ , ‘casual’, ‘registered’,which are continuous variables.

**casual**: count of casual users

**registered**: count of registered users

**cnt**: count of total rental bikes including both casual and registered

As per problem statement we have to predict count of total rental bikes.

Predictive analysis that we can perform is regression,Here we are using regression model to solve the problem.

* We have to predict count of total rental bikes(both casual and registered).
* We have two independent variables casual and registered to predict.
* After predicting both the variables we can sum up their predictions and get predictions of the count of total rental bikes(cnt).

Firstly we will use statistical model ‘Linear Regression’ on our dataset, which is a regression model, to predict the target variables casual and registered.

Afterwards we will use decision tree regressor, random forest regressor, KNN. And then one by one we can apply several models and check the error rate using different techniques. And on basis of error rate we can select one model which will be more suitable for our dataset.

### 2.5.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance

2. Interpretability

3. Computational Efficiency

In our model Interpretability and Computation Efficiency, do not hold much significance. Therefore we will use Predictive performance as the criteria to compare and evaluate models.

Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure and confusion matrix.

### 2.5.2 Error Measure

There are three primary metrics used to evaluate linear models. These are: Mean absolute error (MAE), Mean squared error (MSE), or Root mean squared error (RMSE), Root Mean Squared Logarithmic Error (RMSLE).

**MAE**: The easiest to understand. Represents average error

**MSE**: Similar to MAE but noise is exaggerated and larger errors are “punished”. It is harder to interpret than MAE as it’s not in base units, however, it is generally more popular.

**RMSE**: Most popular metric, similar to MSE, however, the result is square rooted to make it more interpretable as it’s in base units. It is recommended that RMSE be used as the primary metric to interpret your model.

**RMSLE**: In case of RMSLE, we take the log of the predictions and actual values. So basically, what changes is the variance that we are measuring. RMSLE is usually used when we don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

We are using RMSE and RMSLE for our model evaluation.

We our considering RMSE over MAPE because this is a regression model. And RMSE is more suitable for regression models.We are using RMSLE becacause predicted and actual values are huge numbers.

### 2.5.3 Confusion Matrix

In case of classification models ,Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Here our model is regression model so for accuracy evaluation we can not use confusion matrix. So we have only used error matrix.

### 2.5.4 Cross Validation

The validation set approach

Cross Validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it.

In this approach, we reserve 50% of the dataset for validation and the remaining 50% for model training.

### 2.6 Model Selection

### 2.6.1 Linear Regression Model

linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

We have multiple independent variables explaining our dependent variable .So in this case we are doing multiple linear regression.In linear regression, we are attempting to build a model that allows us to predict the value of new data, given the training data used to train our model.

../Screen%20Shot%202018-11-02%20at%2012.41.38%20PM.png

Here B0 is intercept, B1, B2…Bp are coefficients/weights of independent variables x1,x2….xp , where p is the number of independent variables and y is target/dependent variable.

We have two approaches for linear regression one is api from statsmodel and another is using linear\_model from scikit learn library.Here we are using statsmodel Since we get summary matrix in this.So we do not have to call score,coef or intercept individually.whereas we can get reasonable information about model with the help of summary matrix.

In statsmodel we are using OLS method , OLS stands for ordinary least square and the method ‘Least Square’ means that we are trying to fit a regression line that would minimise the squares of distance from the regression line.

**Summary Statistics Description**

Summary statistics helps us to evaluate performance of our model based on some parameters.

From summary statistics matrix we can conclude some observations explain basic information regarding model .

p-Value – x to put into model. P-value should me greater than mode of t.

p-value should be less than 0.05 , Since if p-value is more than 0.05 then it shows the multicollinearity effect.

R-squared – It shows how much amount of variance it is giving for target variables.

Adj. R-squared – It will penalise the effect of extra variables in the model.

Adj. R-squared should be less than R-squared.

F-statistics – It helps us to measure how significant the fit is. It is difference between actual values and predicted values. It should be as low as possible.

Which explains less deviation between actual and predictive values.

Prob(F-statistics) – It is probability of F- statistics of model. It gives us overall performance of model, If probability is less than 0.05 then we can say model is reasonably good.

AIC – Alkalined information criterion , It adjusts the log likelihood based on the number of obsevations and complexity of model.

BIC - It should be higher than AIC.

Coefficients – Coefficients tell us the amount of information, each variable is carrying weight/explaining about target variables.

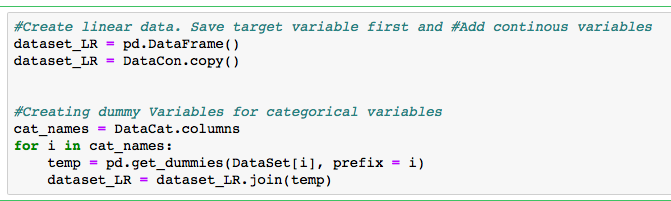
Std-err – Standard error / deviation tells us the error of the estimate of coefficients.

t – t statistics measures how statistically significant the coefficient is because based on t- value we calculate p-value from t-table.

Here we are using statsmodels.api.OLS for regression problem. For model development we are following some steps:

Firstly we have created a dataset for Linear Regression model.

* Create linear dataset.
* Added continuous variables
* Created dummy Variables for categorical variables and then added to dataset



So Now we have a dataset prepared for model development. As we are building regression model.

As per problem statement given cnt = casual + Registered

cnt: count of total rental bikes including both casual and registered

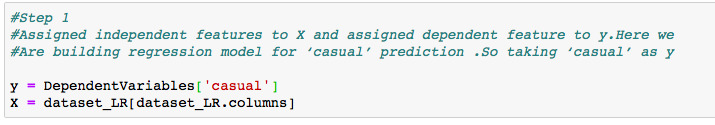
We are calculating predicted cnt with the sum of predicted values of registered and casual variables. So we will build regression model for casual first and then for registered variables to get the final predicted count.

**LR Model for Target Variable : Casual**

**Step 1**

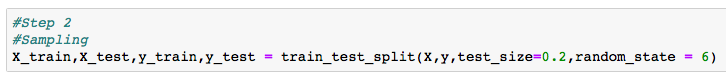
Assigned independent features to X and assigned dependent feature to y.Here we

Are building regression model for ‘casual’ prediction .So taking ‘casual’ as y.



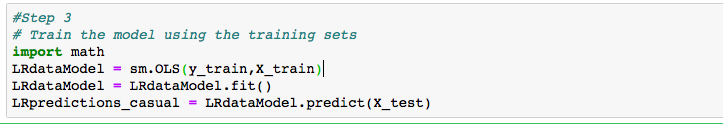
**Step 2**

Sampling



**Step 3**

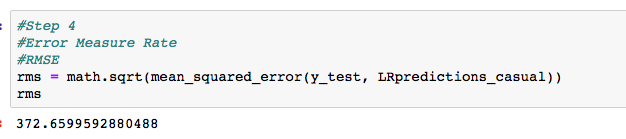
Training the model using the training sets



**Step 4**

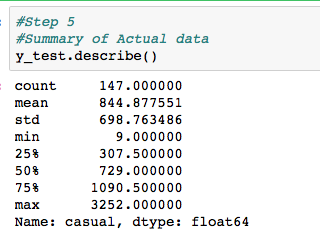
Error Measure Rate

RMSE



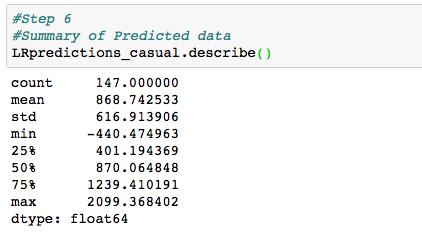
**Step 5**

Summary of Actual data



**Step 6**

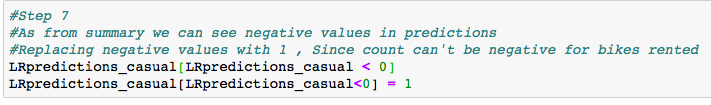
Summary of Predicted data



**Step 7**

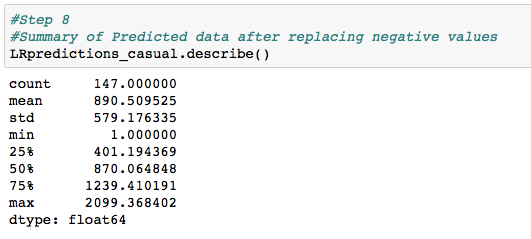
As from summary we can see negative values in predictions

Replacing negative values with 1 , Since count can't be negative for bikes rented



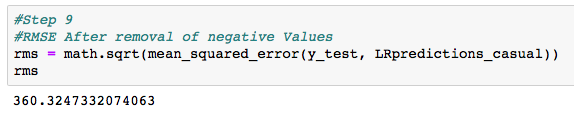
**Step 8**

Summary of Predicted data after replacing negative values



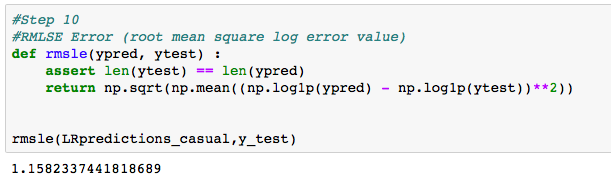
**Step 9**

RMSE After removal of negative Values



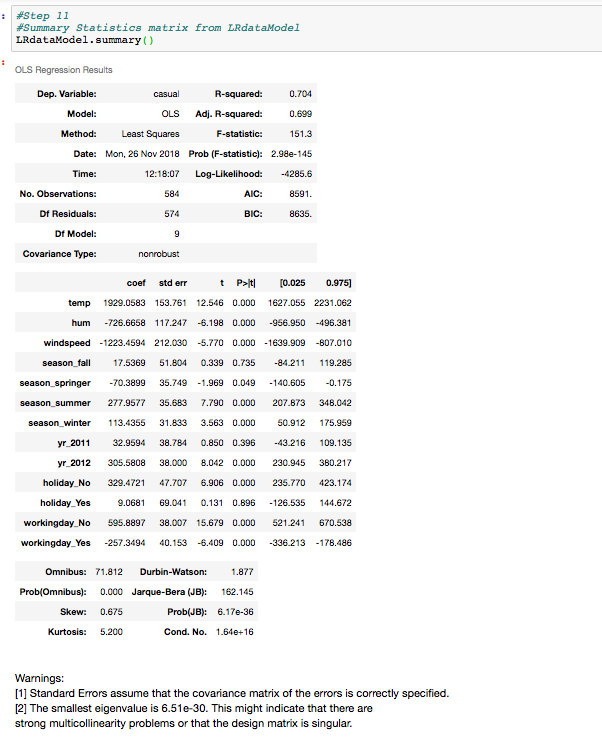
**Step 10**

RMLSE Error (root mean square log error value)



**Step 11**

Summary Statistics matrix from LRdataModel

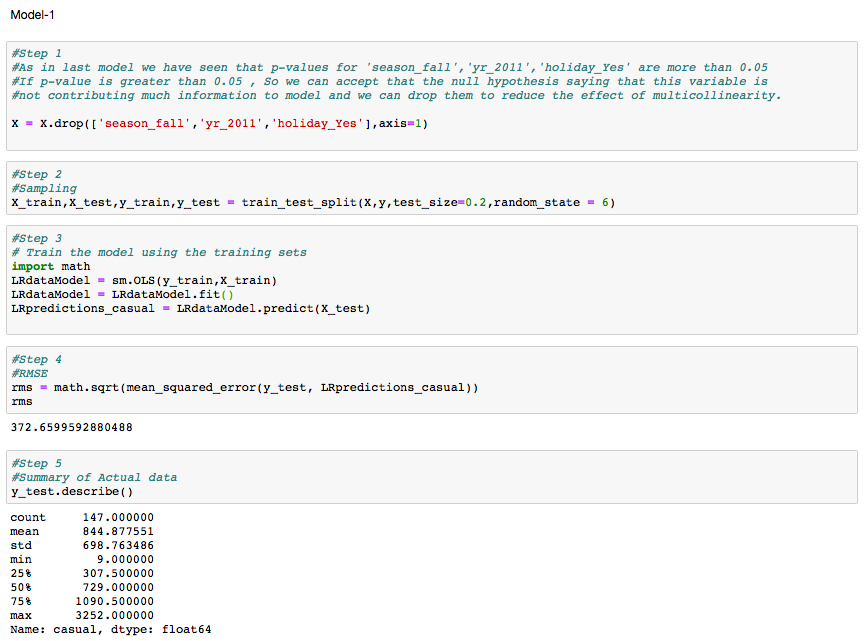


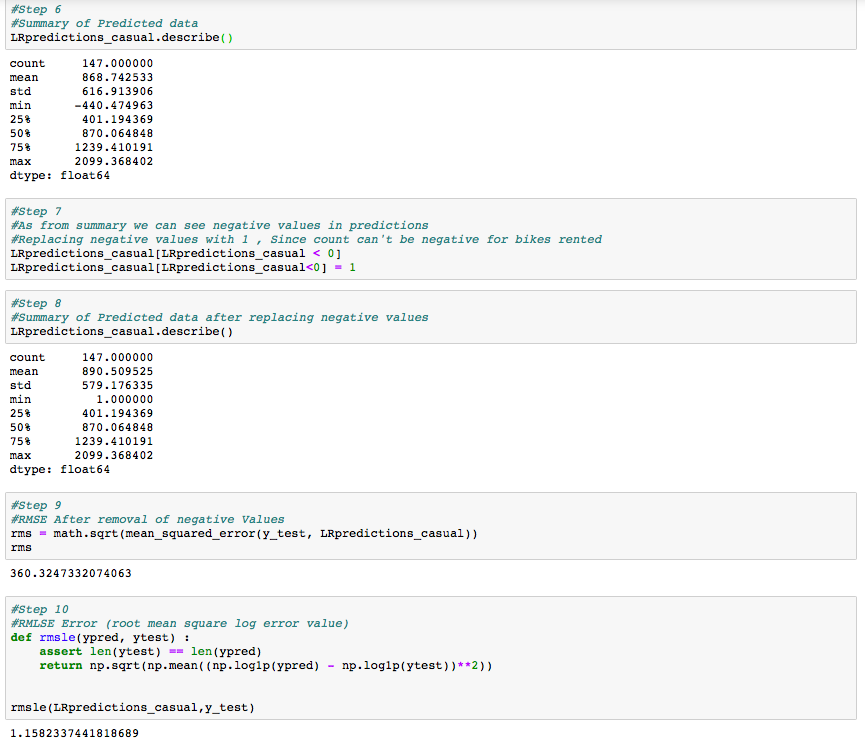
**Iterative Modelling**

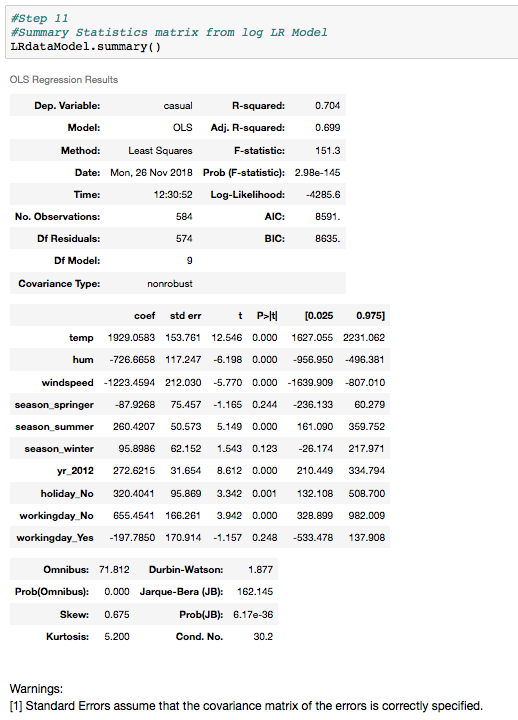
As we have observed from above summary statistics that we have multicollinearity problem in our model. We can do iterative modelling to improve our model performance by removing the effect of multicollinearity.

* As in last model we have seen that p-values for 'season\_fall','yr\_2011','holiday\_Yes' are more than 0.05.
* If p-value is greater than 0.05 , So we can accept that the null hypothesis saying that this variable is not contributing much information to model and we can drop them to reduce the effect of multicollinearity.

So after dropping these variables we will place the regression model again on remaining features to predict the ‘casual’ variable.



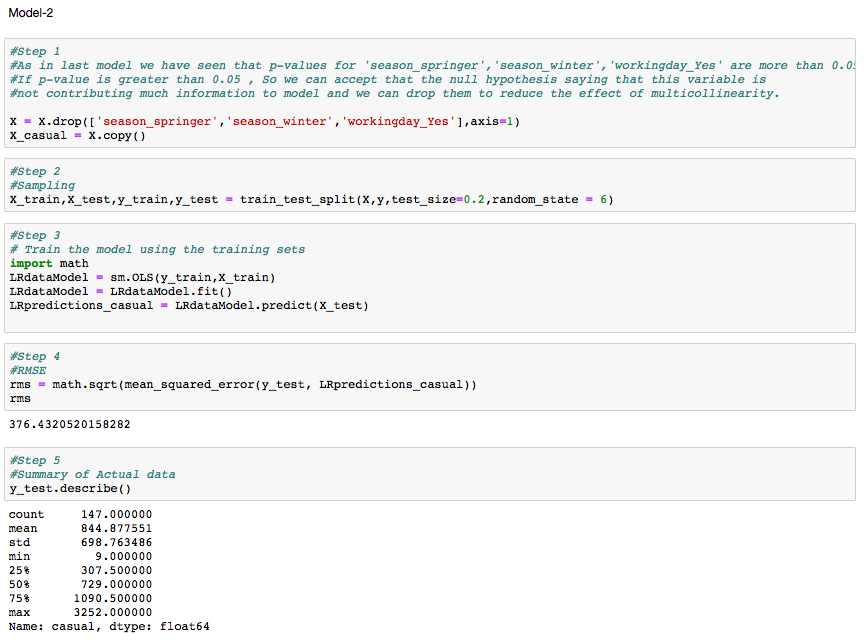


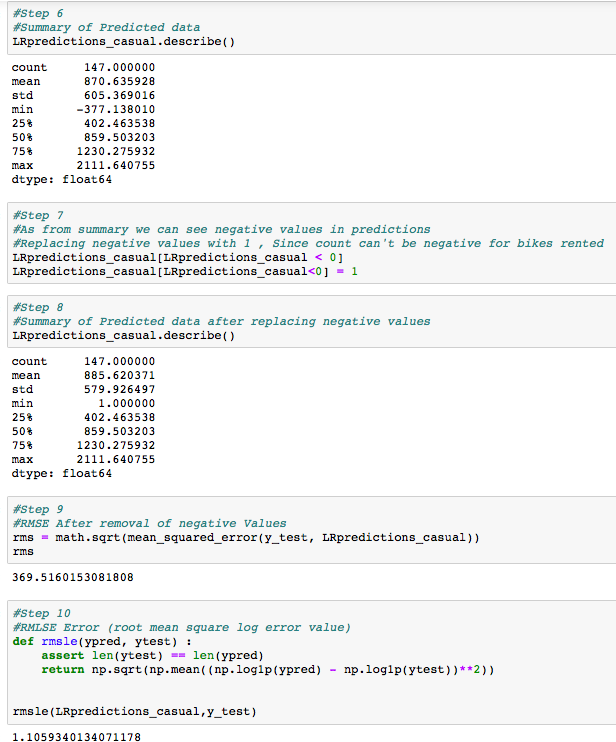


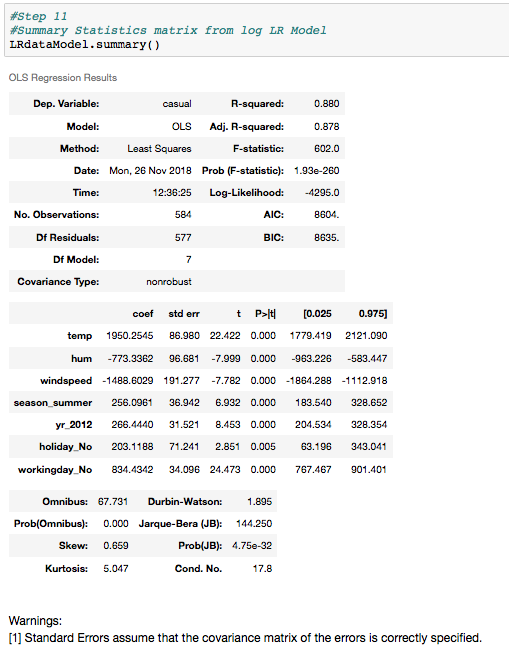
Now we can see that there is no multicollinearity effect in our model

But we have seen in summary statistics that p-values for 'season\_springer','season\_winter','workingday\_Yes' are more than 0.05.

If p-value is greater than 0.05 , So we can accept that the null hypothesis saying that this variable is not contributing much information to model and we can drop them to improve the model performance.







**LR Model for Target Variable : Casual [Log]**

As we have got negative values in prediction from last model, Which we have replaced by ‘1’ and after that we have calculated errors measures for the regression model.

Since we got negative predicted values, we can do log transformation and run

regression model again to get more accurate predictions.

Here we are taking log on target variable first and then we are passing log of target

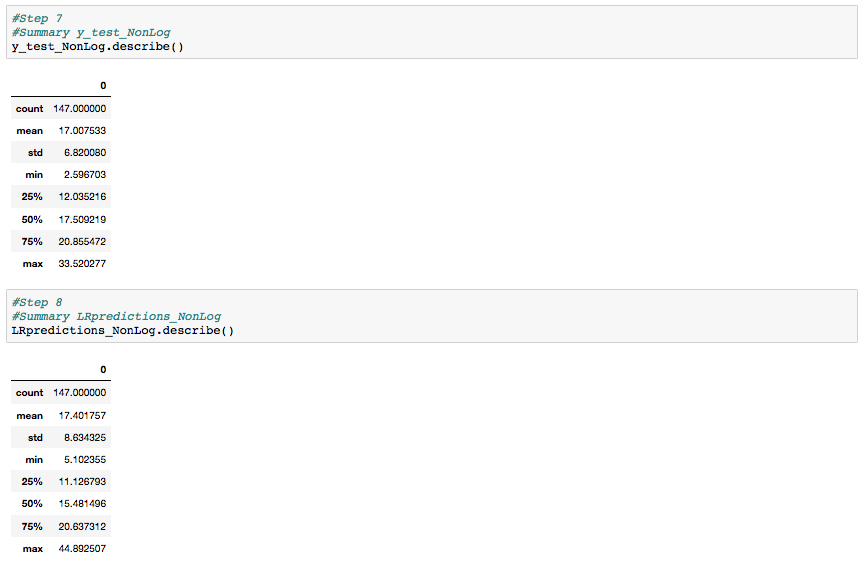
variable to the model. So that we can get all positive predictions.

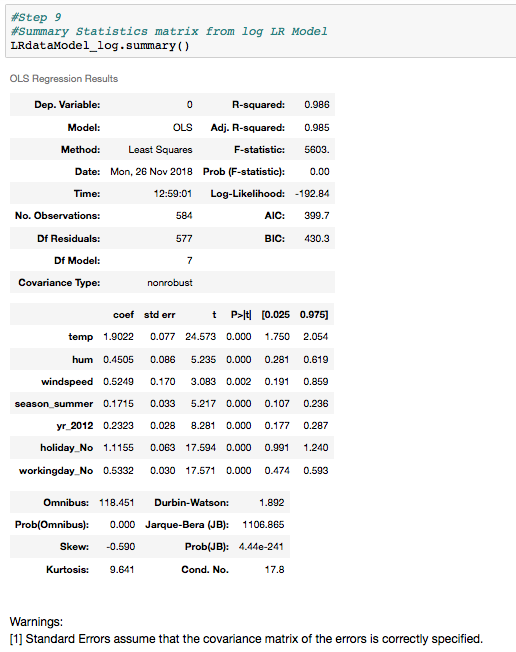
For this we will get predictions also in log format. So we will convert prediction values to exponential and reverse log both. Since we need to calculate error measures with the help of exponential values and actual count of bikes with reverse log.

As the predicted values are in log format, use exponential(exp) to convert from log to non-log values to calculate error measures.

After calculating error measures. We will convert log predictions by reversing log. So that we can get actual predictions by regression model. And we can get the ‘casual’ count of bikes







From summary statistics we can see that performance of our model has improved .in last model R-squared was 0.88 , Now R-squared is 0.98 . And also we do not have multicollinearity effect also.

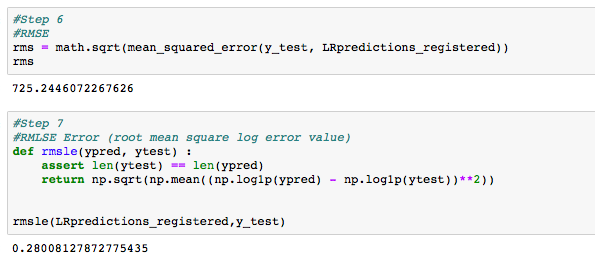
* We got below predictions after reversing the log from predictions.

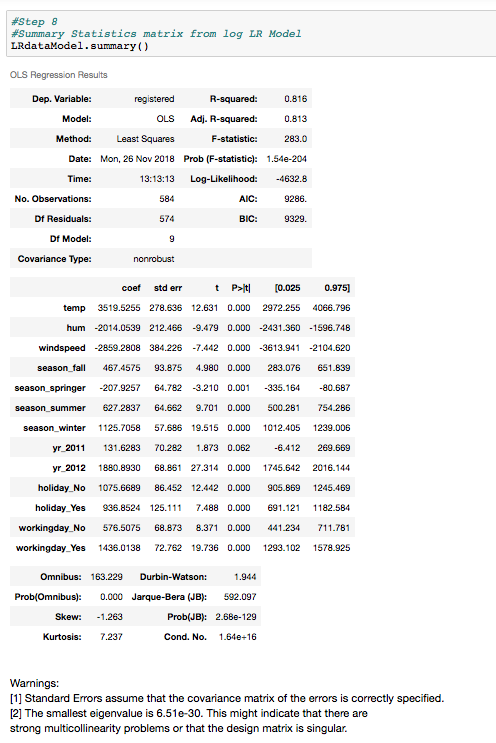


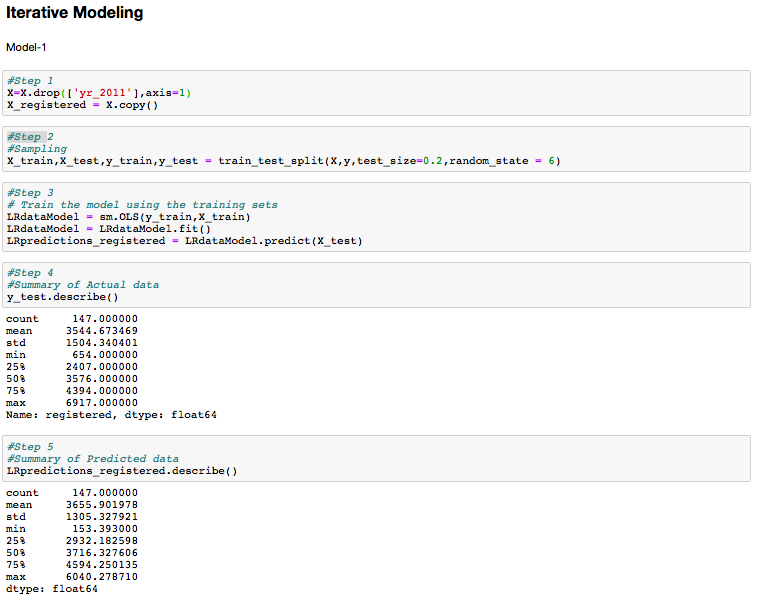
**LR Model for Target Variable : Registered**

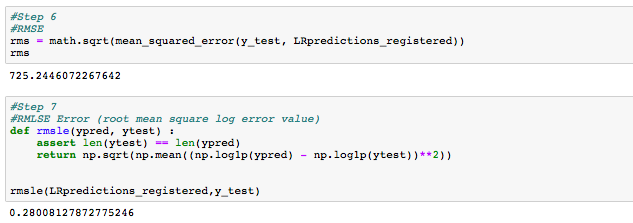
We have applied the same approach for ‘registered’ which we have followed to predict ‘casual’.

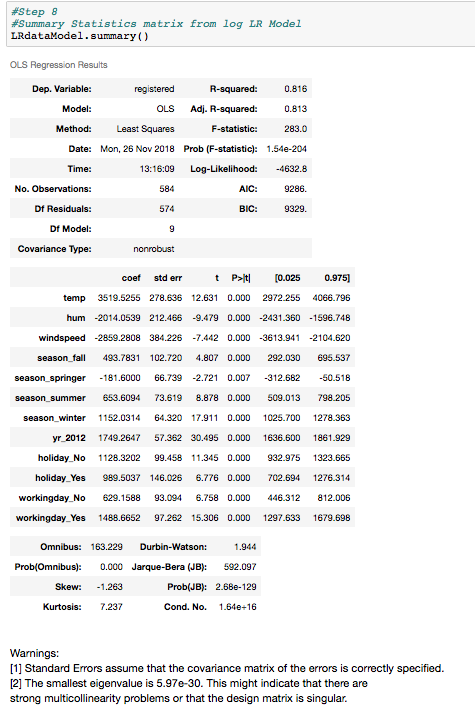


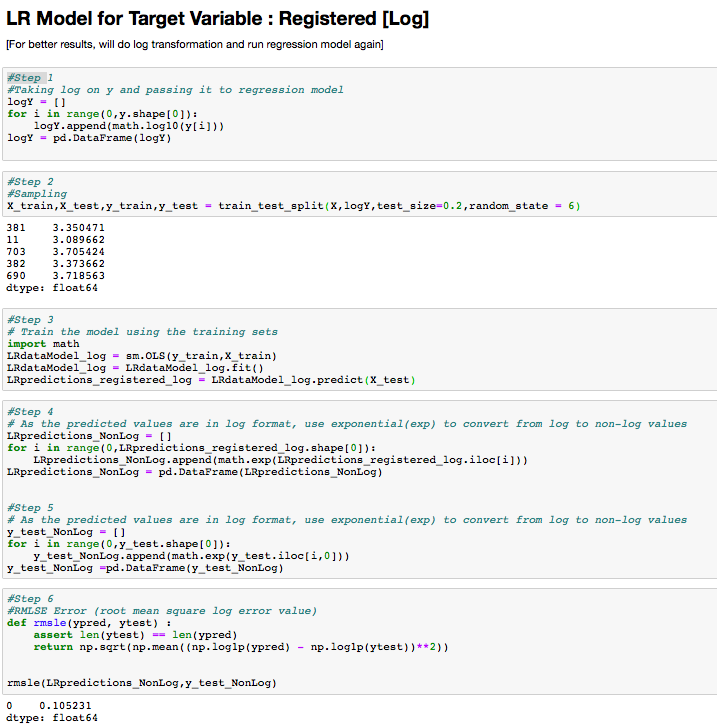




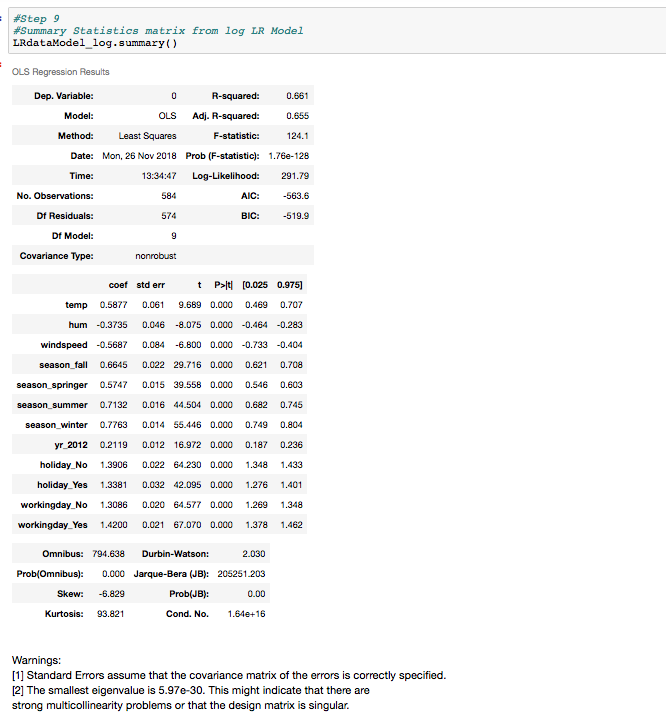














### 2.6.2 Decision Tree

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Here we are using Decision Tree Regressor for regression problem. For decision tree regressor model development we are following some steps:



digraph Tree {

node [shape=box] ;

0 [label="temp <= 0.418\nmse = 467242.021\nsamples = 584\nvalue = 849.007"] ;

1 [label="workingday <= 0.5\nmse = 124813.488\nsamples = 224\nvalue = 374.871"] ;

0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;

2 [label="mse = 209391.325\nsamples = 76\nvalue = 601.066"] ;

1 -> 2 ;

3 [label="mse = 41616.244\nsamples = 148\nvalue = 258.716"] ;

1 -> 3 ;

4 [label="workingday <= 0.5\nmse = 453394.135\nsamples = 360\nvalue = 1144.025"] ;

0 -> 4 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;

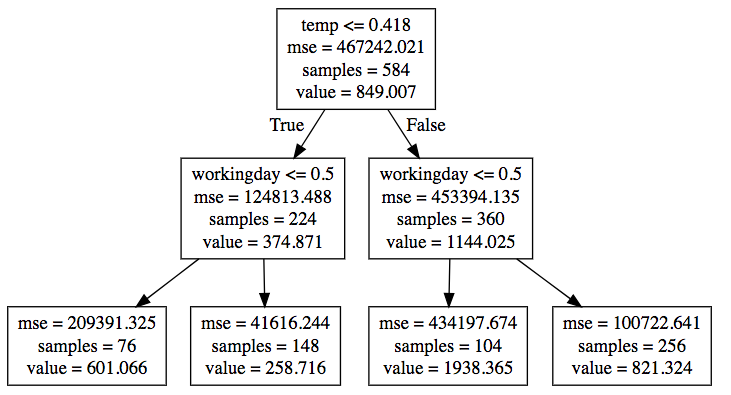
5 [label="mse = 434197.674\nsamples = 104\nvalue = 1938.365"] ;

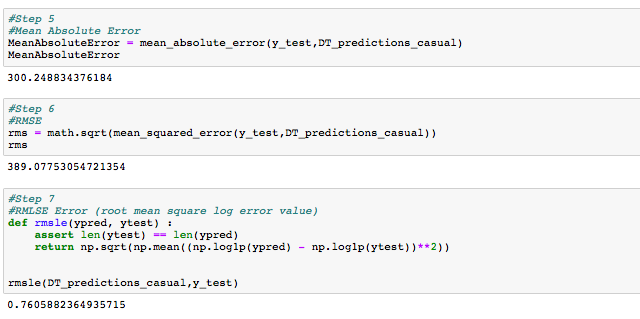
4 -> 5 ;

6 [label="mse = 100722.641\nsamples = 256\nvalue = 821.324"] ;

4 -> 6 ;

}







digraph Tree {

node [shape=box] ;

0 [label="yr <= 0.5\nmse = 2473322.972\nsamples = 584\nvalue = 3684.238"] ;

1 [label="temp <= 0.458\nmse = 1113556.752\nsamples = 295\nvalue = 2716.105"] ;

0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;

2 [label="mse = 768017.017\nsamples = 139\nvalue = 1941.302"] ;

1 -> 2 ;

3 [label="mse = 409931.647\nsamples = 156\nvalue = 3406.474"] ;

1 -> 3 ;

4 [label="temp <= 0.43\nmse = 1927975.516\nsamples = 289\nvalue = 4672.471"] ;

0 -> 4 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;

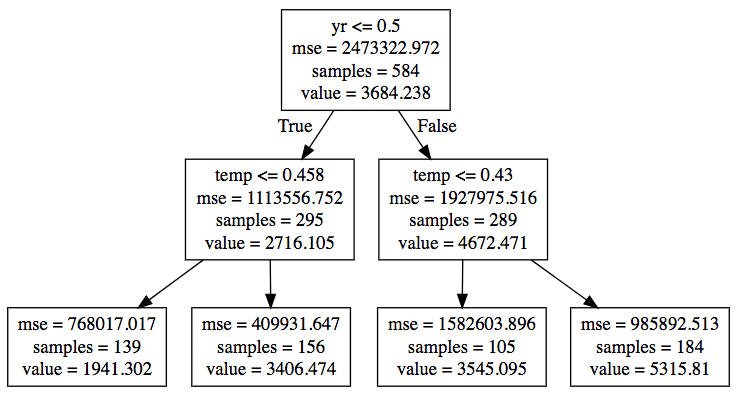
5 [label="mse = 1582603.896\nsamples = 105\nvalue = 3545.095"] ;

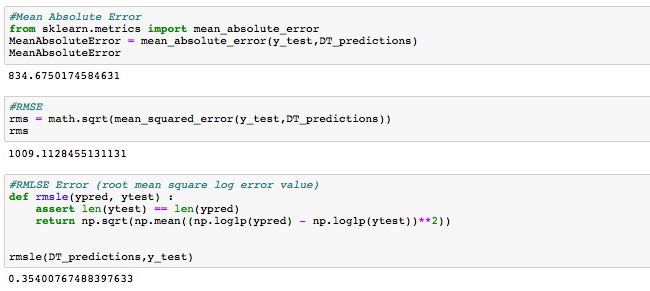
4 -> 5 ;

6 [label="mse = 985892.513\nsamples = 184\nvalue = 5315.81"] ;

4 -> 6 ;

}





### 

### 2.6.3 Random Forest

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models.

This broad technique of using multiple models to obtain better predictive performance is called model ensembling. In random forests, all the base models are constructed independently using a different subsample of the data.

The random forest model is very good at handling tabular data with numerical features, or categorical features with fewer than hundreds of categories. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target.

Here we are using Decision Forest Regressor for regression problem. For model development we are following some steps:





### 2.6.4 KNN

KNN can be used for both classification and regression problems. The algorithm uses ‘feature similarity’ to predict values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

Here we are using KNeighbors Regressor for regression problem. For model development we are following some steps:





### 2.6.5 Model Selection

You always start your model building most suitable model .Therefore we use statistical model first which is Linear Regression model. And then one by one we can apply several models and check the accuracy score and error measures.

Since we are solving regression problem our main focus is on error rate.

So on basis of error rate , we can select one model which will be more suitable for target variable prediction.

We have applied these four models on dataset, And we found error measures also from them .

* Linear Regression Model

Regression Model for casual

RMSLE : 0.304147

Regression Model for Registered

RMSLE : 0.105231

* Decision Tree Regression Model

Regression Model for casual

RMSLE : 0.760588

Regression Model for Registered

RMSLE : 0.354007

* Random Forest Regression Model

Regression Model for casual

RMSLE : 0.640292

Regression Model for Registered

RMSLE : 0.282603

* KNN Regression Model

Regression Model for casual

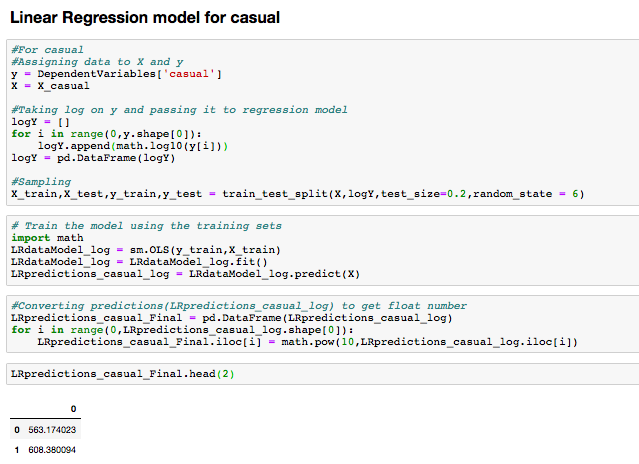
RMSLE : 0.551801

Regression Model for Registered

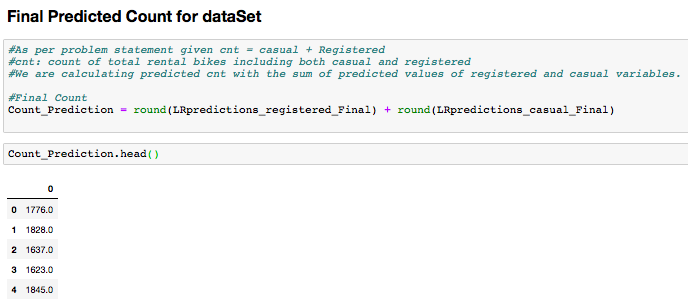
RMSLE : 0.244016

**Final Model**

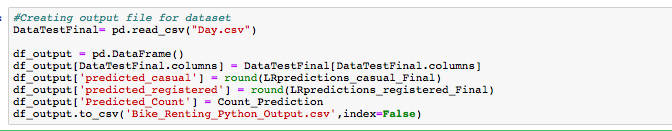
So we can conclude here from above results that error measure rates for Linear Regression model are low as compared to other Models. Linear regression model is most suitable model for the problem. And we can freeze Linear regression model for the dataset.

****

****



### 2.7 Generate Output File



# **3.Conclusion**

## **3.1 Factors affecting Count of Bike Renting**

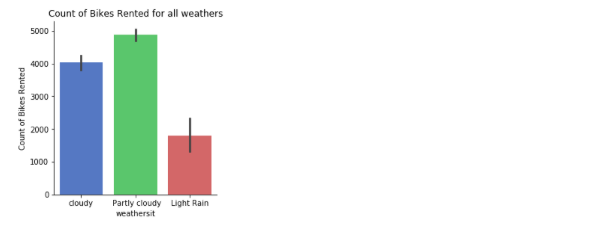
Problem statement says that

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

We are concluding here by answering the question which are also got explained by our regression model.

As we have predicted the bike rental count on daily basis using given predictors as per problem statement using Linear Regression Model on both ‘registered’ and ‘casual’ dependent variables and then with their sum we have got bike rental count on daily basis.

And with some graphs we can conclude that count of bike rented get affected by weather and seasons.





* As in partly cloudy or clear weather the count of bikes rented were high comparatively other weathers.
* In lightly rainy weather the count of bikes rented were much low comparatively other weathers
* In fall season the count of bikes rented were high comparatively other seasons.
* In spring season the count of bikes rented were much low comparatively other seasons

We can see growth from 2011 to 2012 in count of bikes rented in following graph



* In year 2012 the growth has been good from 2011.
* Growth percentage rate of bikes rented in 2012 as per 2011 was 64.87 % higher than 2011.

The below Violin plot is showing dependent variable ‘cnt’ (count of bikes)



* Here the central dot representing the median average count of bikes rented per day over both the years is between 4000 to 4500.

## **3.2 Challenges Faced**

There are a number of important challenges that have been faced during development:

* Figuring out what assumptions can be safely made about the data and the underlying system.
* The data needed Exploratory Data Analysis and data preprocessing Since Dataset has outliers present in dataset and also we had subcategories explained in problem statement for some variables.
* Under Exploratory data analysis we have converted normalized data to actual data and plotted scatter plots for actual data to get the insights of actual values of temperature, feeling temperature, humidity and wind speed.
* Outliers present in dataset and selection of features were handled under data pre processing.
* Feature selection has to be done individually for categorical and continuous data. And keeping only related variables which explain the dependent variable and are not collinear.
* For preventing multicollinearity. We have used correlation analysis, Chi-square test and iterative modelling also in linear regression models .
* We had three dependent variables casual , registered and cnt

cnt: count of total rental bikes including both casual and registered

* For predicting count of bikes rented we have predicted casual and registered count individually and then by their sum we got our desired output which was ‘cnt’

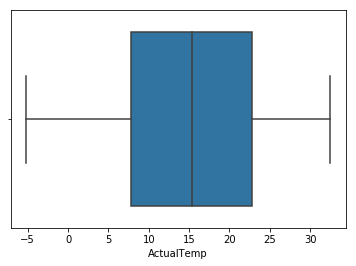
Count of bikes rented.

* For casual by using iterative modelling we have corrected the effect of multicollinearity.
* For Linear regression we were getting negative predictions. Which can not be correct in this case. Since count of bikes cannot be negative in number. So to get the predictions in positive numbers. We have used Linear regression model by taking Log of target variable.
* We have first used RMSE , Since we have huge predicted values and the error rate was higher. And it was not justifying the error because of huge values So to get the justified error rate we have used RMSLE(Root Mean Square Log Error).

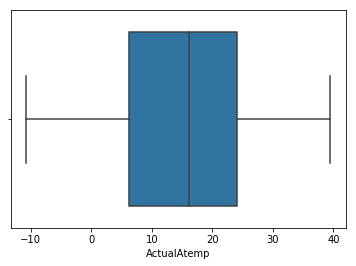
# **Appendix – A**

# **Boxplots for Calculated Actual Continuous variables**

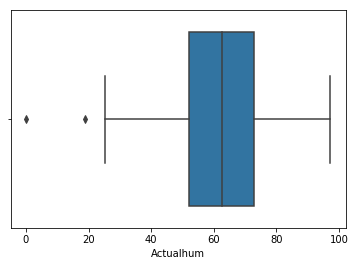




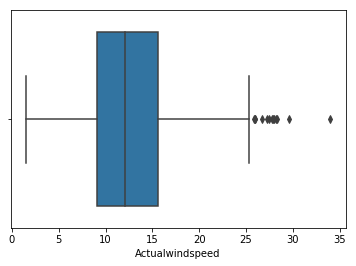
ActualTemp



ActualAtemp

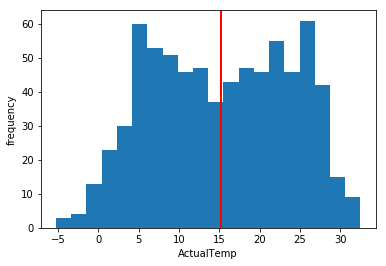


Actualhum

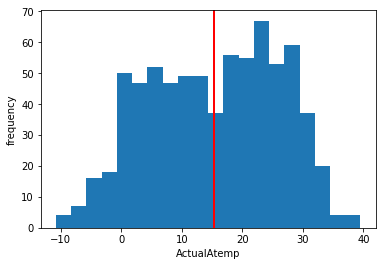


Actualwindspeed

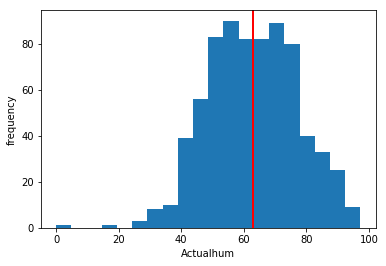
**Histograms with Mean for skewed Data**



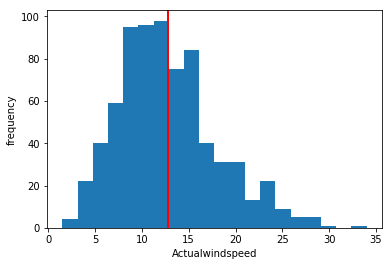
ActualTemp



ActualAtemp

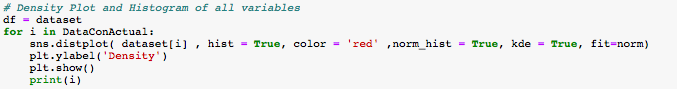


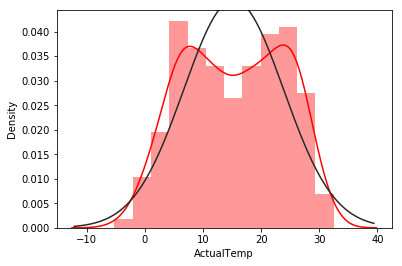
Actualhum



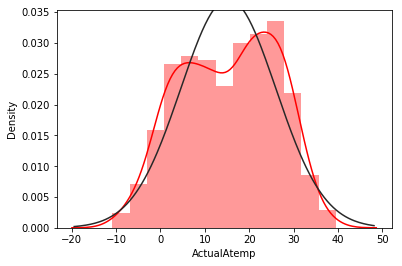
Actualwindspeed

**Density Plot and Histogram of all variables**

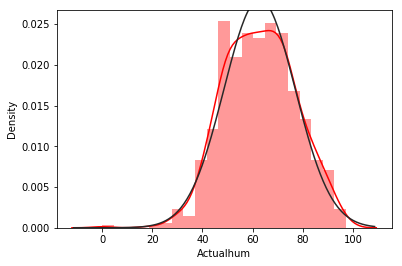
****



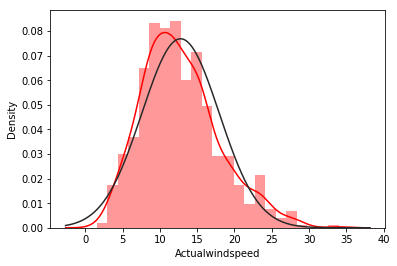
ActualTemp



ActualAtemp



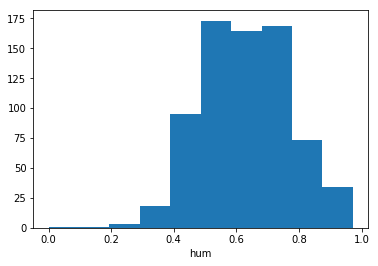
Actualhum

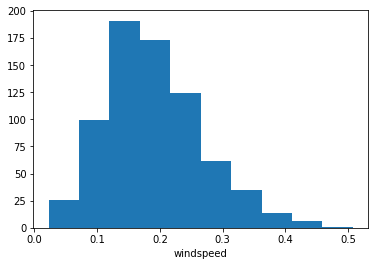


Actualwindspeed

**Plot Histogram for Normality check**







In [ ]:

1

​

**Appendix – B**

# **Python Code**

# Problem : Bike Renting

In [ ]:

*#Load Libraries*

**import** **os**

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sns**

**import** **math**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy.stats** **import** norm

**from** **scipy.stats** **import** chi2\_contingency

**from** **sklearn.model\_selection** **import** train\_test\_split

**import** **statsmodels.api** **as** **sm**

**from** **sklearn.ensemble** **import** RandomForestRegressor

**from** **sklearn.tree** **import** DecisionTreeRegressor

**from** **sklearn** **import** tree

**from** **sklearn.neighbors** **import** KNeighborsRegressor

**from** **sklearn.metrics** **import** mean\_absolute\_error

**from** **sklearn.metrics** **import** explained\_variance\_score

**from** **sklearn.metrics** **import** mean\_squared\_error

In [ ]:

*#Loading dataSet*

os.chdir("/Users/bhartisharma/Desktop/Bike Renting")

dataset = pd.read\_csv("Day.csv")

dataset\_copy = dataset.copy()

dataset\_copy1=dataset.copy()

# Data

In [ ]:

dataset.head(2)

In [ ]:

*#Dimensions of Data*

dataset.shape

In [ ]:

*#Data Information*

dataset.info()

In [ ]:

*#Statistics of Data*

dataset.describe()

# Exploratory Data Analysis

In [ ]:

*#DataCat is a dataframe containing all categorical variables from dataset*

DataCat = pd.DataFrame()

DataCat = dataset[['instant','dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday','workingday', 'weathersit']]

*#DataCon is a dataframe containing all continuous variables from dataset*

DataCon = pd.DataFrame()

DataCon = dataset[['temp', 'atemp', 'hum', 'windspeed','casual', 'registered', 'cnt']]

*#Continuous data is already in float and int type , But we have to convert categorical variables to object type*

*#Converting some integer and float variables to categorical variable as per requirement*

dataset[DataCat.columns] = dataset[DataCat.columns].astype(object)

In [ ]:

dataset.dtypes

In [ ]:

*#weathersit*

*#weathersit: As per problem statement*

*#1: Clear, Few clouds, Partly cloudy, Partly cloudy*

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

*#3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds*

*#4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog*

r,c =dataset.shape

**for** i **in** range(0,r):

**if**(dataset.loc[i,('weathersit')] == 1):

dataset.loc[i,('weathersit')] = ' Partly cloudy'

**elif**(dataset.loc[i,('weathersit')] == 2):

dataset.loc[i,('weathersit')] = 'cloudy'

**elif**(dataset.loc[i,('weathersit')] == 3):

dataset.loc[i,('weathersit')] = 'Light Rain'

**elif**(dataset.loc[i,('weathersit')] == 4):

dataset.loc[i,('weathersit')] = 'Heavy Rain'

In [ ]:

*#Year (0: 2011, 1:2012)*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('yr')] == 0):

dataset.loc[i,('yr')] = '2011'

**elif**(dataset.loc[i,('yr')] == 1):

dataset.loc[i,('yr')] = '2012'

In [ ]:

*#Seasons (1:springer, 2:summer, 3:fall, 4:winter)*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('season')] == 1):

dataset.loc[i,('season')] = 'springer'

**elif**(dataset.loc[i,('season')] == 2):

dataset.loc[i,('season')] = 'summer'

**elif**(dataset.loc[i,('season')] == 3):

dataset.loc[i,('season')] = 'fall'

**elif**(dataset.loc[i,('season')] == 4):

dataset.loc[i,('season')] = 'winter'

In [ ]:

*#mnth: Month (1 to 12)*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('mnth')] == 1):

dataset.loc[i,('mnth')] = 'January'

**elif**(dataset.loc[i,('mnth')] == 2):

dataset.loc[i,('mnth')] = 'February'

**elif**(dataset.loc[i,('mnth')] == 3):

dataset.loc[i,('mnth')] = 'March'

**elif**(dataset.loc[i,('mnth')] == 4):

dataset.loc[i,('mnth')] = 'April'

**elif**(dataset.loc[i,('mnth')] == 5):

dataset.loc[i,('mnth')] = 'May'

**elif**(dataset.loc[i,('mnth')] == 6):

dataset.loc[i,('mnth')] = 'June'

**elif**(dataset.loc[i,('mnth')] == 7):

dataset.loc[i,('mnth')] = 'July'

**elif**(dataset.loc[i,('mnth')] == 8):

dataset.loc[i,('mnth')] = 'August'

**elif**(dataset.loc[i,('mnth')] == 9):

dataset.loc[i,('mnth')] = 'September'

**elif**(dataset.loc[i,('mnth')] == 10):

dataset.loc[i,('mnth')] = 'October'

**elif**(dataset.loc[i,('mnth')] == 11):

dataset.loc[i,('mnth')] = 'November'

**elif**(dataset.loc[i,('mnth')] == 12):

dataset.loc[i,('mnth')] = 'December'

In [ ]:

*#holiday: weather day is holiday or not (extracted fromHoliday Schedule)*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('holiday')] == 1):

dataset.loc[i,('holiday')] = 'Yes'

**elif**(dataset.loc[i,('holiday')] == 0):

dataset.loc[i,('holiday')] = 'No'

In [ ]:

*#weekday: Day of the week*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('weekday')] == 1):

dataset.loc[i,('weekday')] = 'Monday'

**elif**(dataset.loc[i,('weekday')] == 2):

dataset.loc[i,('weekday')] = 'Tuesday'

**elif**(dataset.loc[i,('weekday')] == 3):

dataset.loc[i,('weekday')] = 'Wednesday'

**elif**(dataset.loc[i,('weekday')] == 4):

dataset.loc[i,('weekday')] = 'Thursday'

**elif**(dataset.loc[i,('weekday')] == 5):

dataset.loc[i,('weekday')] = 'Friday'

**elif**(dataset.loc[i,('weekday')] == 6):

dataset.loc[i,('weekday')] = 'Saturday'

**elif**(dataset.loc[i,('weekday')] == 0):

dataset.loc[i,('weekday')] = 'Sunday'

In [ ]:

*#workingday: If day is neither weekend nor holiday is 1, otherwise is 0.*

**for** i **in** range(0,r):

**if**(dataset.loc[i,('workingday')] == 1):

dataset.loc[i,('workingday')] = 'Yes'

**elif**(dataset.loc[i,('workingday')] == 0):

dataset.loc[i,('workingday')] = 'No'

In [ ]:

dataset\_copy = dataset.copy()

In [ ]:

dataset.head(2)

In [ ]:

*#temp: Normalized temperature in Celsius.*

*#The values are derived via (t-t\_min)/(t\_max-t\_min),t\_min=-8, t\_max=+39 (only in hourly scale)*

t\_min=-8

t\_max=+39

ActualTemp = pd.DataFrame()

ActualTemp = (dataset['temp']\*(t\_max - t\_min) ) + t\_min

dataset['ActualTemp'] = ActualTemp

In [ ]:

*#atemp: Normalized feeling temperature in Celsius.*

*#The values are derived via (t-t\_min)/(t\_max-t\_min), t\_min=-16, t\_max=+50 (only in hourly scale)*

t\_min=-16

t\_max=+50

ActualAtemp = pd.DataFrame()

ActualAtemp = (dataset['atemp']\*(t\_max - t\_min) ) + t\_min

dataset['ActualAtemp'] = ActualAtemp

In [ ]:

*#hum: Normalized humidity. The values are divided to 100 (max)*

Actualhum = pd.DataFrame()

Actualhum = dataset['hum'] \* 100

dataset['Actualhum'] = Actualhum

In [ ]:

*#windspeed: Normalized wind speed. The values are divided to 67 (max)*

Actualwindspeed = pd.DataFrame()

Actualwindspeed= dataset['windspeed'] \* 67

dataset['Actualwindspeed']= Actualwindspeed

In [ ]:

dataset.head(2)

In [ ]:

*#DataCon is a dataframe containing all continuous variables from dataset*

*#Here we are adding actual values for ActualTemp and ActualAtemp which we have calculated using given formula in problem statement*

*#Since we will use actual values for these variable in feature analysis and outier analysis to get accurate results*

DataConActual = pd.DataFrame()

DataConActual['ActualTemp'] = dataset['ActualTemp']

DataConActual['ActualAtemp'] = dataset['ActualAtemp']

DataConActual['Actualhum'] = dataset['Actualhum']

DataConActual['Actualwindspeed'] = dataset['Actualwindspeed']

DataConActual.head(2)

# Visualisations

In [ ]:

*#Bargraph between Season and Count of bikes rented*

sns.factorplot("season", "cnt", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented for all seasons")

plt.xlabel("seasons")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Bargraph between Year and Count of bikes rented*

sns.factorplot("yr", "cnt", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented per year")

plt.xlabel("Year")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Bargraph between Year and Count of bikes rented showing the growth in all seasons*

sns.factorplot("yr", "cnt", "season", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented per year in all seasons")

plt.xlabel("Year")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#In yr 2012 the bike renting count has incresed*

*#growth in Bike Renting*

growth = dataset['cnt'][dataset['yr'] == '2012' ].sum() - dataset['cnt'][dataset['yr'] == '2011' ].sum()

growth\_percentage = growth\*100/dataset['cnt'][dataset['yr'] == '2011' ].sum()

growth\_percentage

In [ ]:

*#Bargraph between month and Count of bikes rented*

sns.factorplot("cnt", "mnth", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented per month")

plt.xlabel("Count of Bikes Rented")

plt.ylabel("month")

In [ ]:

*#Bargraph between Holiday and Count of bikes rented*

sns.factorplot("holiday", "cnt", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented for holiday")

plt.xlabel("holiday")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Bargraph between workingday and Count of bikes rented*

sns.factorplot("workingday", "cnt", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented for all workingday")

plt.xlabel("workingday")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Bargraph between weekdays and Count of bikes rented*

sns.factorplot("cnt", "weekday", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented for weekdays")

plt.xlabel("Count of Bikes Rented")

plt.ylabel("weekday")

In [ ]:

*#Bargraph between weather and Count of bikes rented*

sns.factorplot("weathersit", "cnt", data=dataset\_copy, kind="bar", palette="muted", legend=**False**)

plt.title("Count of Bikes Rented for all weathers")

plt.xlabel("weathersit")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Scatter plot for Actual Temperature*

plt.scatter(DataConActual['ActualTemp'], dataset\_copy['cnt'],color='gray')

plt.title("Count of Bikes Rented for Actual Temperature")

plt.xlabel("Actual Temperature")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Scatter plot for Actual Feeling Temperature*

plt.scatter(DataConActual['ActualAtemp'], dataset\_copy['cnt'],color='green')

plt.title("Count of Bikes Rented for Actual Feeling Temperature")

plt.xlabel("Feeling Temperature(Actual Atemp)")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Scatter plot for Actual Humidity*

plt.scatter(DataConActual['Actualhum'], dataset\_copy['cnt'])

plt.title("Count of Bikes Rented according to Actual Humidity ")

plt.xlabel("humidity")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Scatter plot for Actual Windspeed*

plt.scatter(DataConActual['Actualwindspeed'], dataset\_copy['cnt'] , color='orange')

plt.title("Count of Bikes Rented according to Actual windspeed")

plt.xlabel("Actual Windspeed")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Violinplot showing data of dependent variable 'cnt'*

*#So average count of bikes rented per day over the year is between 4000 to 5000*

sns.violinplot(x = "cnt", data=dataset\_copy)

plt.xlabel("Count of Bikes Rented")

### Bike Rent Count by season in weekdays

In [ ]:

*#Boxplot showing count of bikes rented in weekdays in all seasons*

plt.figure(figsize=(20, 10))

sns.boxplot("weekday", "cnt", hue="season", data=dataset\_copy)

plt.title('Count as per weekdays in all seasons')

plt.xticks(rotation = 90);

plt.xlabel("weekdays")

plt.ylabel("Count of Bikes Rented")

### Bike Rent Count by weather in weekdays

In [ ]:

*#Boxplot showing count of bikes rented in weekdays in all weathers*

plt.figure(figsize=(20, 10))

sns.boxplot("weekday", "cnt", hue="weathersit", data=dataset\_copy)

plt.title('Count as per weekdays')

plt.xticks(rotation = 90);

plt.xlabel("weekdays")

plt.ylabel("Count of Bikes Rented")

In [ ]:

*#Boxplot for Actual continuous data*

**for** i **in** DataConActual.columns:

sns.boxplot(dataset[i])

plt.show()

print(i)

In [ ]:

*#Histograms with Mean for skewed Data*

*#area codes*

**for** i **in** DataConActual.columns:

x= dataset[i]

plt.hist(x,bins=20)

plt.axvline(x.mean(), color='Red', linewidth=2)

plt.ylabel('frequency')

plt.xlabel(i)

plt.show()

print(i)

In [ ]:

*# Density Plot and Histogram of all variables*

df = dataset

**for** i **in** DataConActual:

sns.distplot( dataset[i] , hist = **True**, color = 'red' ,norm\_hist = **True**, kde = **True**, fit=norm)

plt.ylabel('Density')

plt.show()

print(i)

# Data Preprocessing

# Missing Values

In [ ]:

*#Checking for missing values in dataset*

dataset.isnull().sum()

### Invalid Data

In [ ]:

*#Checking for invalid data*

**for** i **in** DataCat:

**if**(i != 'instant' **and** i != 'dteday'):

print(i)

print(pd.unique(dataset\_copy1[i]))

# Outlier Analysis

In [ ]:

*#boxplot to visualize Outliers*

**import** **matplotlib.pyplot** **as** **plt**

%**matplotlib** inline

**for** i **in** DataConActual.columns :

print(i)

plt.boxplot(dataset[i])

plt.xlabel(i)

plt.show()

In [ ]:

*# Outlier Analysis*

cnames = DataConActual.columns

**for** col **in** cnames:

percentile = dataset[col].quantile([0.25,0.75]).values

iqr = percentile[1] - percentile[0]

minimum = percentile[0] - (iqr\*1.5)

maximum = percentile[1] + (iqr\*1.5)

*#print(col,percentile,maximum,minimum)*

dataset[col][dataset[col] <= minimum] = minimum

dataset[col][dataset[col] >= maximum] = maximum

In [ ]:

*#boxplot to visualize after Outlier correction*

**import** **matplotlib.pyplot** **as** **plt**

%**matplotlib** inline

**for** i **in** DataConActual.columns :

print(i)

plt.boxplot(dataset[i])

plt.xlabel(i)

plt.show()

# Feature Selection

Feature Selection for continuos data

In [ ]:

*#Correlation plot for Continuous variables*

cnames = DataConActual.columns

df\_corr = DataConActual.loc[:,cnames]

*#Set the width and hieght of the plot*

f, ax = plt.subplots(figsize=(7, 5))

*#Generate correlation matrix*

corr = df\_corr.corr()

*#Plot using seaborn library*

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=**True**),

square=**True**, ax=ax)

In [ ]:

*#Continuous Variables selection using corr()function*

DataCon\_corr = DataConActual.corr()

*#from continuous variables variables*

DataCon\_corrVar = DataCon\_corr[DataCon\_corr > 0.8]

In [ ]:

DataCon\_corrVar

In [ ]:

*#Making a list DataConCorrVar,which contains values only greater than 0.8*

**import** **numpy** **as** **np**

*#DataCon\_corrVar = DataCon\_corrVar.replace(np.nan,-1)*

DataConCorrVar = []

**for** i **in** range(0,DataCon\_corrVar.shape[1]):

**for** j **in** range(0,DataCon\_corrVar.shape[1]):

**if**(i != j):

**if**((DataCon\_corrVar.iloc[i,j] > 0.8).any(axis=0)):

*#print(DataCon\_corrVar.columns[i]+ " , "+DataCon\_corrVar.columns[j] + " :",DataCon\_corrVar.iloc[i,j])*

DataConCorrVar.append((DataCon\_corrVar.columns[i] , DataCon\_corrVar.columns[j] ,DataCon\_corrVar.iloc[i,j] ))

In [ ]:

*#Convert to DataFrame*

DataConCorrVar = pd.DataFrame(DataConCorrVar)

DataConCorrVar

In [ ]:

dataset = dataset.drop(['atemp'],axis=1)

DataCon = DataCon.drop(['atemp'],axis=1)

In [ ]:

dataset.columns

Feature Selection for categorical data

In [ ]:

*# Creating a list corrVar which will contain variable names and their p-values from chi2\_contigency test*

corrVar = []

**for** i **in** DataCat.columns:

**for** j **in** DataCat.columns:

**if**(i!=j):

*#print(i,j)*

chi,p,def1,ex = chi2\_contingency(pd.crosstab(DataCat[i],DataCat[j]))

*#print(p)*

corrVar.append((i,j,p))

In [ ]:

*#Convert to DataFrame*

corrVar = pd.DataFrame(corrVar)

*#corrVar[2]*

In [ ]:

*#If p-value < 0.05: significant result, reject null hypothesis (H0), dependent.*

*#If independent variables are dependent to each other we can drop one of them .*

*#If p-value > 0.05: not significant result, fail to reject null hypothesis (H0), independent.*

corrVar[corrVar[2] < 0.05]

In [ ]:

*#dropping categorical variables'mnth','holiday' as per chi square test result*

*#dropping 'instant','dteday' , Since they are not carrying important environmental and seasonal information to the model as per problem statement*

dataset = dataset.drop(['instant','dteday','mnth','weekday','weathersit'],axis=1)

DataCat =DataCat.drop(['instant','dteday','mnth','weekday','weathersit'],axis=1)

In [ ]:

*#Creating DataSet*

DataSet = pd.DataFrame()

DependentVariables = pd.DataFrame()

DataSet = dataset.copy()

*#Droping dependent variables and variables which we have created for correlation test*

DependentVariables = dataset[['casual', 'registered', 'cnt']]

DataSet = DataSet.drop(['ActualTemp', 'ActualAtemp', 'Actualhum','Actualwindspeed','casual', 'registered', 'cnt'],axis=1)

DataCon = DataCon.drop([ 'casual', 'registered', 'cnt'],axis=1)

In [ ]:

*#Final DataSet for model development*

DataSet.head(2)

# Feature Scaling

In [ ]:

*#Since we already have continuous data given in normalised form .*

*#We are passing the same data to our model.*

In [ ]:

DataSet.head(2)

In [ ]:

*#Plot Histogram for Normality check*

cnames = DataCon.columns

**for** i **in** cnames:

plt.hist(dataset[i])

plt.xlabel(i)

plt.show()

# Model Development

# Linear Regression Model

In [ ]:

*#Create linear data. Save target variable first and #Add continous variables*

dataset\_LR = pd.DataFrame()

dataset\_LR = DataCon.copy()

*#Creating dummy Variables for categorical variables*

cat\_names = DataCat.columns

**for** i **in** cat\_names:

temp = pd.get\_dummies(DataSet[i], prefix = i)

dataset\_LR = dataset\_LR.join(temp)

In [ ]:

dataset\_LR.head(2)

# LR Model for Target Variable : Casual

In [ ]:

*#Step 1*

*#Assigned independent features to X and assigned dependent feature to y.Here we*

*#Are building regression model for ‘casual’ prediction .So taking ‘casual’ as y*

y = DependentVariables['casual']

X = dataset\_LR[dataset\_LR.columns]

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

**import** **math**

LRdataModel = sm.OLS(y\_train,X\_train)

LRdataModel = LRdataModel.fit()

LRpredictions\_casual = LRdataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#Error Measure Rate*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 5*

*#Summary of Actual data*

y\_test.describe()

In [ ]:

*#Step 6*

*#Summary of Predicted data*

LRpredictions\_casual.describe()

In [ ]:

*#Step 7*

*#As from summary we can see negative values in predictions*

*#Replacing negative values with 1 , Since count can't be negative for bikes rented*

LRpredictions\_casual[LRpredictions\_casual < 0]

LRpredictions\_casual[LRpredictions\_casual<0] = 1

In [ ]:

*#Step 8*

*#Summary of Predicted data after replacing negative values*

LRpredictions\_casual.describe()

In [ ]:

*#Step 9*

*#RMSE After removal of negative Values*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 10*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_casual,y\_test)

In [ ]:

*#Step 11*

*#Summary Statistics matrix from LRdataModel*

LRdataModel.summary()

## Iterative Modeling

Model-1

In [ ]:

*#Step 1*

*#As in last model we have seen that p-values for 'season\_fall','yr\_2011','holiday\_Yes' are more than 0.05*

*#If p-value is greater than 0.05 , So we can accept that the null hypothesis saying that this variable is*

*#not contributing much information to model and we can drop them to reduce the effect of multicollinearity.*

X = X.drop(['season\_fall','yr\_2011','holiday\_Yes'],axis=1)

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

**import** **math**

LRdataModel = sm.OLS(y\_train,X\_train)

LRdataModel = LRdataModel.fit()

LRpredictions\_casual = LRdataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 5*

*#Summary of Actual data*

y\_test.describe()

In [ ]:

*#Step 6*

*#Summary of Predicted data*

LRpredictions\_casual.describe()

In [ ]:

*#Step 7*

*#As from summary we can see negative values in predictions*

*#Replacing negative values with 1 , Since count can't be negative for bikes rented*

LRpredictions\_casual[LRpredictions\_casual < 0]

LRpredictions\_casual[LRpredictions\_casual<0] = 1

In [ ]:

*#Step 8*

*#Summary of Predicted data after replacing negative values*

LRpredictions\_casual.describe()

In [ ]:

*#Step 9*

*#RMSE After removal of negative Values*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 10*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_casual,y\_test)

In [ ]:

*#Step 11*

*#Summary Statistics matrix from log LR Model*

LRdataModel.summary()

Model-2

In [ ]:

*#Step 1*

*#As in last model we have seen that p-values for 'season\_springer','season\_winter','workingday\_Yes' are more than 0.05*

*#If p-value is greater than 0.05 , So we can accept that the null hypothesis saying that this variable is*

*#not contributing much information to model and we can drop them to reduce the effect of multicollinearity.*

X = X.drop(['season\_springer','season\_winter','workingday\_Yes'],axis=1)

X\_casual = X.copy()

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

**import** **math**

LRdataModel = sm.OLS(y\_train,X\_train)

LRdataModel = LRdataModel.fit()

LRpredictions\_casual = LRdataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 5*

*#Summary of Actual data*

y\_test.describe()

In [ ]:

*#Step 6*

*#Summary of Predicted data*

LRpredictions\_casual.describe()

In [ ]:

*#Step 7*

*#As from summary we can see negative values in predictions*

*#Replacing negative values with 1 , Since count can't be negative for bikes rented*

LRpredictions\_casual[LRpredictions\_casual < 0]

LRpredictions\_casual[LRpredictions\_casual<0] = 1

In [ ]:

*#Step 8*

*#Summary of Predicted data after replacing negative values*

LRpredictions\_casual.describe()

In [ ]:

*#Step 9*

*#RMSE After removal of negative Values*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_casual))

rms

In [ ]:

*#Step 10*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_casual,y\_test)

In [ ]:

*#Step 11*

*#Summary Statistics matrix from log LR Model*

LRdataModel.summary()

# LR Model for Target Variable : Casual [Log]

[Since we got negative predicted values, we will do log transformation and run regression model again]

In [ ]:

*#Step 1*

*#Taking log on y for passing it to regression model*

logY = []

**for** i **in** range(0,y.shape[0]):

logY.append(math.log10(y[i]))

logY = pd.DataFrame(logY)

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,logY,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

**import** **math**

LRdataModel\_log = sm.OLS(y\_train,X\_train)

LRdataModel\_log = LRdataModel\_log.fit()

LRpredictions\_casual\_log = LRdataModel\_log.predict(X\_test)

In [ ]:

*#Step 4*

*# As the predicted values are in log format, use exponential(exp) to convert from log to non-log values*

LRpredictions\_NonLog = []

**for** i **in** range(0,LRpredictions\_casual\_log.shape[0]):

LRpredictions\_NonLog.append(math.exp(LRpredictions\_casual\_log.iloc[i]))

LRpredictions\_NonLog = pd.DataFrame(LRpredictions\_NonLog)

In [ ]:

*#Step 5*

*# As the target variable passed is in log format, use exponential(exp) to convert from log to non-log values*

y\_test\_NonLog = []

**for** i **in** range(0,y\_test.shape[0]):

y\_test\_NonLog.append(math.exp(y\_test.iloc[i,0]))

y\_test\_NonLog =pd.DataFrame(y\_test\_NonLog)

In [ ]:

*#Step 6*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_NonLog,y\_test\_NonLog)

In [ ]:

*#Step 7*

*#Summary y\_test\_NonLog*

y\_test\_NonLog.describe()

In [ ]:

*#Step 8*

*#Summary LRpredictions\_NonLog*

LRpredictions\_NonLog.describe()

In [ ]:

*#Step 9*

*#Summary Statistics matrix from log LR Model*

LRdataModel\_log.summary()

## Final Output For Casual

In [ ]:

*#Converting predictions(LRpredictions\_casual\_log) to get float number*

LRpredictions\_casual\_Final = pd.DataFrame(LRpredictions\_casual\_log)

**for** i **in** range(0,LRpredictions\_casual\_log.shape[0]):

LRpredictions\_casual\_Final.iloc[i] = math.pow(10,LRpredictions\_casual\_log.iloc[i])

LRpredictions\_casual\_Final.head(10)

# LR Model for Target Variable : Registered

In [ ]:

*#Step 1*

*#Split dataset*

*#Applying Model for 2nd dependent variable 'registered'*

y = DependentVariables['registered']

X = dataset\_LR[dataset\_LR.columns]

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

LRdataModel = sm.OLS(y\_train,X\_train)

LRdataModel = LRdataModel.fit()

LRpredictions\_registered = LRdataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#Summary of Actual data*

y\_test.describe()

In [ ]:

*#Step 5*

*#Summary of Predicted data*

LRpredictions\_registered.describe()

In [ ]:

*#Step 6*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_registered))

rms

In [ ]:

*#Step 7*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_registered,y\_test)

In [ ]:

*#Step 8*

*#Summary Statistics matrix from log LR Model*

LRdataModel.summary()

## Iterative Modeling

Model-1

In [ ]:

*#Step 1*

X=X.drop(['yr\_2011'],axis=1)

X\_registered = X.copy()

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

LRdataModel = sm.OLS(y\_train,X\_train)

LRdataModel = LRdataModel.fit()

LRpredictions\_registered = LRdataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#Summary of Actual data*

y\_test.describe()

In [ ]:

*#Step 5*

*#Summary of Predicted data*

LRpredictions\_registered.describe()

In [ ]:

*#Step 6*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test, LRpredictions\_registered))

rms

In [ ]:

*#Step 7*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_registered,y\_test)

In [ ]:

*#Step 8*

*#Summary Statistics matrix from log LR Model*

LRdataModel.summary()

# LR Model for Target Variable : Registered [Log]

[For better results, will do log transformation and run regression model again]

In [ ]:

*#Step 1*

*#Taking log on y and passing it to regression model*

logY = []

**for** i **in** range(0,y.shape[0]):

logY.append(math.log10(y[i]))

logY = pd.DataFrame(logY)

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,logY,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*# Train the model using the training sets*

**import** **math**

LRdataModel\_log = sm.OLS(y\_train,X\_train)

LRdataModel\_log = LRdataModel\_log.fit()

LRpredictions\_registered\_log = LRdataModel\_log.predict(X\_test)

In [ ]:

*#Step 4*

*# As the predicted values are in log format, use exponential(exp) to convert from log to non-log values*

LRpredictions\_NonLog = []

**for** i **in** range(0,LRpredictions\_registered\_log.shape[0]):

LRpredictions\_NonLog.append(math.exp(LRpredictions\_registered\_log.iloc[i]))

LRpredictions\_NonLog = pd.DataFrame(LRpredictions\_NonLog)

*#Step 5*

*# As the predicted values are in log format, use exponential(exp) to convert from log to non-log values*

y\_test\_NonLog = []

**for** i **in** range(0,y\_test.shape[0]):

y\_test\_NonLog.append(math.exp(y\_test.iloc[i,0]))

y\_test\_NonLog =pd.DataFrame(y\_test\_NonLog)

In [ ]:

*#Step 6*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(LRpredictions\_NonLog,y\_test\_NonLog)

In [ ]:

*#Step 7*

*#Summary y\_test\_NonLog*

y\_test\_NonLog.describe()

In [ ]:

*#Step 8*

*#Summary LRpredictions\_NonLog*

LRpredictions\_NonLog.describe()

In [ ]:

*#Step 9*

*#Summary Statistics matrix from log LR Model*

LRdataModel\_log.summary()

## Final Output For Registered

In [ ]:

*#Converting predictions(LRpredictions\_casual\_log) to get float number*

LRpredictions\_registered\_Final = pd.DataFrame(LRpredictions\_registered\_log)

**for** i **in** range(0,LRpredictions\_registered\_log.shape[0]):

LRpredictions\_registered\_Final.iloc[i] = math.pow(10,LRpredictions\_registered\_log.iloc[i])

LRpredictions\_registered\_Final.head(10)

## Final Output for Testdata For 'cnt' variable

In [ ]:

*#As per problem statement given cnt = casual + Registered*

*#cnt: count of total rental bikes including both casual and registered*

*#We are calculating predicted cnt with the sum of predicted values of registered and casual variables.*

*#Final Count*

Count\_Prediction = LRpredictions\_registered\_Final + LRpredictions\_casual\_Final

Count\_Prediction = round(Count\_Prediction)

Count\_Prediction.head(10)

In [ ]:

Count\_Prediction.head(10)

# Decision Tree

In [ ]:

*#convert categorical variables to codes, So that we can easily compute and impute easily*

**for** i **in** range(0,DataSet.shape[1]):

**if**(DataSet.iloc[:,i].dtypes == 'object'):

DataSet.iloc[:,i] = pd.Categorical(DataSet.iloc[:,i])

DataSet.iloc[:,i] = DataSet.iloc[:,i].cat.codes

In [ ]:

DataSet.head(2)

## Decision Tree for Casual

In [ ]:

*#Step 1*

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['casual']

In [ ]:

*#Step 2*

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Step 3*

*#Applying Model*

DataModel = DecisionTreeRegressor(random\_state =6,max\_depth=2)

DataModel = DataModel.fit(X\_train,y\_train)

DT\_predictions\_casual = DataModel.predict(X\_test)

In [ ]:

*#Step 4*

*#Create dot file to visualise tree #http://webgraphviz.com/*

df = tree.export\_graphviz(DataModel, out\_file='tree\_casual.dot' ,feature\_names = X.columns)

In [ ]:

*#Step 5*

*#Mean Absolute Error*

MeanAbsoluteError = mean\_absolute\_error(y\_test,DT\_predictions\_casual)

MeanAbsoluteError

In [ ]:

*#Step 6*

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,DT\_predictions\_casual))

rms

In [ ]:

*#Step 7*

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(DT\_predictions\_casual,y\_test)

## Decision Tree for registered

In [ ]:

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['registered']

In [ ]:

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Applying Model*

DataModel = DecisionTreeRegressor(random\_state =6,max\_depth=2)

DataModel = DataModel.fit(X\_train,y\_train)

DT\_predictions = DataModel.predict(X\_test)

In [ ]:

*#Create dot file to visualise tree #http://webgraphviz.com/*

df = tree.export\_graphviz(DataModel, out\_file='tree\_registered.dot' ,feature\_names = X.columns)

In [ ]:

*#Mean Absolute Error*

**from** **sklearn.metrics** **import** mean\_absolute\_error

MeanAbsoluteError = mean\_absolute\_error(y\_test,DT\_predictions)

MeanAbsoluteError

In [ ]:

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,DT\_predictions))

rms

In [ ]:

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(DT\_predictions,y\_test)

# Random Forest

## Random Forest for casual

In [ ]:

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['casual']

In [ ]:

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Apply Model*

DataModel\_RF = RandomForestRegressor(random\_state=7, max\_depth=3, n\_estimators=125)

DataModel\_RF = DataModel\_RF.fit(X\_train,y\_train)

RF\_Predictions\_casual = DataModel\_RF.predict(X\_test)

In [ ]:

*#Mean Absolute Error*

MeanAbsoluteError = mean\_absolute\_error(y\_test,RF\_Predictions\_casual)

MeanAbsoluteError

In [ ]:

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,RF\_Predictions\_casual))

rms

In [ ]:

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(RF\_Predictions\_casual,y\_test)

## Random Forest for registered

In [ ]:

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['registered']

In [ ]:

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#Apply Model*

DataModel\_RF = RandomForestRegressor(random\_state=7, max\_depth=3, n\_estimators=125)

DataModel\_RF = DataModel\_RF.fit(X\_train,y\_train)

RF\_Predictions\_registered = DataModel\_RF.predict(X\_test)

In [ ]:

*#Mean Absolute Error*

MeanAbsoluteError = mean\_absolute\_error(y\_test,RF\_Predictions\_registered)

MeanAbsoluteError

In [ ]:

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,RF\_Predictions\_registered))

rms

In [ ]:

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(RF\_Predictions\_registered,y\_test)

# KNN

## KNN for casual

In [ ]:

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['casual']

In [ ]:

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#4 KNN implementation*

KNN\_Model = KNeighborsRegressor()

KNN\_Model = KNN\_Model.fit(X\_train,y\_train)

KNNpredictions\_casual = KNN\_Model.predict(X\_test)

In [ ]:

*#Mean Absolute Error*

MeanAbsoluteError = mean\_absolute\_error(y\_test,KNNpredictions\_casual)

MeanAbsoluteError

In [ ]:

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,KNNpredictions\_casual))

rms

In [ ]:

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(KNNpredictions\_casual,y\_test)

## KNN for registered

In [ ]:

*#Split dataset*

X = DataSet[DataSet.columns]

y = DependentVariables['registered']

In [ ]:

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state = 6)

In [ ]:

*#4 KNN implementation*

KNN\_Model = KNeighborsRegressor()

KNN\_Model = KNN\_Model.fit(X\_train,y\_train)

KNNpredictions\_registered = KNN\_Model.predict(X\_test)

In [ ]:

*#Mean Absolute Error*

MeanAbsoluteError = mean\_absolute\_error(y\_test,KNNpredictions\_registered)

MeanAbsoluteError

In [ ]:

*#RMSE*

rms = math.sqrt(mean\_squared\_error(y\_test,KNNpredictions\_registered))

rms

In [ ]:

*#RMLSE Error (root mean square log error value)*

**def** rmsle(ypred, ytest) :

**assert** len(ytest) == len(ypred)

**return** np.sqrt(np.mean((np.log1p(ypred) - np.log1p(ytest))\*\*2))

rmsle(KNNpredictions\_registered,y\_test)

# Final Model

In [ ]:

*#We are freezing Linear Regression model*

*#As we have observed ERROR MESEARE rates for Linear Regression model are low as compared to other Models.*

## Linear Regression model for casual

In [ ]:

*#For casual*

*#Assigning data to X and y*

y = DependentVariables['casual']

X = X\_casual

*#Taking log on y and passing it to regression model*

logY = []

**for** i **in** range(0,y.shape[0]):

logY.append(math.log10(y[i]))

logY = pd.DataFrame(logY)

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,logY,test\_size=0.2,random\_state = 6)

In [ ]:

*# Train the model using the training sets*

**import** **math**

LRdataModel\_log = sm.OLS(y\_train,X\_train)

LRdataModel\_log = LRdataModel\_log.fit()

LRpredictions\_casual\_log = LRdataModel\_log.predict(X)

In [ ]:

*#Converting predictions(LRpredictions\_casual\_log) to get float number*

LRpredictions\_casual\_Final = pd.DataFrame(LRpredictions\_casual\_log)

**for** i **in** range(0,LRpredictions\_casual\_log.shape[0]):

LRpredictions\_casual\_Final.iloc[i] = math.pow(10,LRpredictions\_casual\_log.iloc[i])

In [ ]:

LRpredictions\_casual\_Final.head(2)

## Linear Regression model for registered

In [ ]:

*#Assigning data to X and y*

y = DependentVariables['registered']

X = X\_registered

*#Taking log on y and passing it to regression model*

logY = []

**for** i **in** range(0,y.shape[0]):

logY.append(math.log10(y[i]))

logY = pd.DataFrame(logY)

*#Sampling*

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,logY,test\_size=0.2,random\_state = 6)

In [ ]:

*# Train the model using the training sets*

**import** **math**

LRdataModel\_log = sm.OLS(y\_train,X\_train)

LRdataModel\_log = LRdataModel\_log.fit()

LRpredictions\_registered\_log = LRdataModel\_log.predict(X)

In [ ]:

*#Converting predictions(LRpredictions\_registered\_log) to get float number*

LRpredictions\_registered\_Final = pd.DataFrame(LRpredictions\_registered\_log)

**for** i **in** range(0,LRpredictions\_registered\_log.shape[0]):

LRpredictions\_registered\_Final.iloc[i] = math.pow(10,LRpredictions\_registered\_log.iloc[i])

In [ ]:

LRpredictions\_registered\_Final.head(2)

## Final Predicted Count for dataSet

In [ ]:

*#As per problem statement given cnt = casual + Registered*

*#cnt: count of total rental bikes including both casual and registered*

*#We are calculating predicted cnt with the sum of predicted values of registered and casual variables.*

*#Final Count*

Count\_Prediction = round(LRpredictions\_registered\_Final) + round(LRpredictions\_casual\_Final)

In [ ]:

Count\_Prediction.head()

In [ ]:

*#Creating output file for dataset*

DataTestFinal= pd.read\_csv("Day.csv")

df\_output = pd.DataFrame()

df\_output[DataTestFinal.columns] = DataTestFinal[DataTestFinal.columns]

df\_output['predicted\_casual'] = round(LRpredictions\_casual\_Final)

df\_output['predicted\_registered'] = round(LRpredictions\_registered\_Final)

df\_output['Predicted\_Count'] = Count\_Prediction

df\_output.to\_csv('Bike\_Renting\_Python\_Output.csv',index=**False**)

# **Appendix – C**

# **R-Code**

# #---------------------------------------------------Problem - Bike Renting--------------------------------------------------------

# # Load Libraries --

# rm(list=ls(all=T))

# setwd("/Users/bhartisharma/Desktop/Bike Renting")

# # Load Libraries --

# x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information",

# "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

# #install.packages(x)

# lapply(x, require, character.only = TRUE)

# rm(x)

# #--------------------------------------Data

# #Load Data

# dataset = read.csv("day.csv")

# #Converting dataset into a dataframe

# dataset<-as.data.frame(dataset)

# View(dataset)

# #Checking the default datatypes of dataset

# str(dataset)

# #statistics of dataset

# summary(dataset)

# #--------------------------------Exploratory data analysis

# #Creating two different dataframes DataCat and DataCon which contains categorical and continuos variables respectively as explained in the report

# #DataCat is a dataframe containing all categorical variables from dataset

# DataCat = data.frame()

# DataCat = subset(dataset, select = c('instant','dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday','workingday', 'weathersit'))

# View(DataCat)

# #DataCon is a dataframe containing all continuous variables from dataset

# DataCon = data.frame()

# DataCon = subset(dataset , select = c('temp', 'atemp', 'hum', 'windspeed','casual', 'registered', 'cnt') )

# View(DataCat)

# #weathersit

# #weathersit: As per problem statement

# #1: Clear, Few clouds, Partly cloudy, Partly cloudy

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

# #3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

# #4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('weathersit')] == 1)

# {

# dataset[i,('weathersit')] = 'Partly cloudy'

# }

# if(dataset[i,('weathersit')] == 2)

# {

# dataset[i,('weathersit')] = 'cloudy'

# }

# if(dataset[i,('weathersit')] == 3)

# {

# dataset[i,('weathersit')] = 'Light Rain'

# }

# if(dataset[i,('weathersit')] == 4)

# {

# dataset[i,('weathersit')] = 'Heavy Rain'

# }

# 

# }

# #Year (0: 2011, 1:2012)

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('yr')] == 0)

# {

# dataset[i,('yr')] = '2011'

# }

# else{

# if(dataset[i,('yr')] == 1)

# {

# dataset[i,('yr')] = '2012'

# }}

# }

# #Seasons (1:springer, 2:summer, 3:fall, 4:winter)

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('season')] == 1)

# {

# dataset[i,('season')] = "springer"

# }

# if(dataset[i,('season')] == 2)

# {

# dataset[i,('season')] = 'summer'

# }

# if(dataset[i,('season')] == 3)

# {

# dataset[i,('season')] = 'fall'

# }

# if(dataset[i,('season')] == 4)

# {

# dataset[i,('season')] = 'winter'

# }

# 

# }

# #mnth: Month (1 to 12)

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('mnth')] == 1)

# {

# dataset[i,('mnth')] = 'January'

# }

# if(dataset[i,('mnth')] == 2)

# {

# dataset[i,('mnth')] = 'February'

# }

# if(dataset[i,('mnth')] == 3)

# {

# dataset[i,('mnth')] = 'March'

# }

# if(dataset[i,('mnth')] == 4)

# {

# dataset[i,('mnth')] = 'April'

# }

# if(dataset[i,('mnth')] == 5)

# {

# dataset[i,('mnth')] = 'May'

# }

# if(dataset[i,('mnth')] == 6)

# {

# dataset[i,('mnth')] = 'June'

# }

# if(dataset[i,('mnth')] == 7)

# {

# dataset[i,('mnth')] = 'July'

# }

# if(dataset[i,('mnth')] == 8)

# {

# dataset[i,('mnth')] = 'August'

# }

# if(dataset[i,('mnth')] == 9)

# {

# dataset[i,('mnth')] = 'September'

# }

# if(dataset[i,('mnth')] == 10)

# {

# dataset[i,('mnth')] = 'October'

# }

# if(dataset[i,('mnth')] == 11)

# {

# dataset[i,('mnth')] = 'November'

# }

# if(dataset[i,('mnth')] == 12)

# {

# dataset[i,('mnth')] = 'December'

# }

# }

# 

# #holiday: weather day is holiday or not (extracted fromHoliday Schedule)

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('holiday')] == 0)

# {

# dataset[i,('holiday')] = 'No'

# }

# else{

# if(dataset[i,('holiday')] == 1)

# {

# dataset[i,('holiday')] = 'Yes'

# }}

# }

# #weekday: Day of the week

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('weekday')] == 1)

# {

# dataset[i,('weekday')] = 'Monday'

# }

# if(dataset[i,('weekday')] == 2)

# {

# dataset[i,('weekday')] = 'Tuesday'

# }

# if(dataset[i,('weekday')] == 3)

# {

# dataset[i,('weekday')] = 'Wednesday'

# }

# if(dataset[i,('weekday')] == 4)

# {

# dataset[i,('weekday')] = 'Thursday'

# }

# if(dataset[i,('weekday')] == 5)

# {

# dataset[i,('weekday')] = 'Friday'

# }

# if(dataset[i,('weekday')] == 6)

# {

# dataset[i,('weekday')] = 'Saturday'

# }

# if(dataset[i,('weekday')] == 0)

# {

# dataset[i,('weekday')] = 'Sunday'

# }

# }

# #workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

# for (i in c(1,2:nrow(dataset)))

# {

# if(dataset[i,('workingday')] == 0)

# {

# dataset[i,('workingday')] = 'No'

# }

# else{

# if(dataset[i,('workingday')] == 1)

# {

# dataset[i,('workingday')] = 'Yes'

# }}

# }

# #Continuous data is already in float and int type , But we have to convert categorical variables to object type

# #Converting some integer and float variables to categorical variable as per requirement

# for (i in colnames(DataCat))

# { print(i)

# dataset[,i] = as.factor(dataset[,i])

# }

# #Checking the converted datatypes of dataset

# str(dataset)

# View(dataset)

# #temp: Normalized temperature in Celsius.

# #The values are derived via (t-t\_min)/(t\_max-t\_min),t\_min=-8, t\_max=+39 (only in hourly scale)

# t\_min=-8

# t\_max=+39

# ActualTemp = data.frame()

# ActualTemp = (dataset['temp']\*(t\_max - t\_min) ) + t\_min

# dataset$ActualTemp = ActualTemp

# #atemp: Normalized feeling temperature in Celsius.

# #The values are derived via (t-t\_min)/(t\_max-t\_min), t\_min=-16, t\_max=+50 (only in hourly scale)

# t\_min=-16

# t\_max=+50

# ActualAtemp =data.frame()

# ActualAtemp = (dataset['atemp']\*(t\_max - t\_min) ) + t\_min

# dataset$ActualAtemp = ActualAtemp

# #hum: Normalized humidity. The values are divided to 100 (max)

# Actualhum = data.frame()

# Actualhum = dataset$hum\* 100

# dataset$Actualhum = Actualhum

# #windspeed: Normalized wind speed. The values are divided to 67 (max)

# Actualwindspeed = data.frame()

# Actualwindspeed= dataset$windspeed \* 67

# dataset$Actualwindspeed = Actualwindspeed

# #After data exploratory analysis View dataset

# View(dataset)

# #DataCon is a dataframe containing all continuous variables from dataset

# #Here we are adding actual values for ActualTemp and ActualAtemp which we have calculated using given formula in problem statement

# #Since we will use actual values for these variable in feature analysis and outier analysis to get accurate results

# DataConActual = data.frame(dataset['atemp'])

# DataConActual$ActualTemp = dataset$ActualTemp

# DataConActual$ActualAtemp = dataset$ActualAtemp

# DataConActual$Actualhum = dataset$Actualhum

# DataConActual$Actualwindspeed = dataset$Actualwindspeed

# DataConActual['atemp'] = NULL

# View(DataConActual)

# #-------------------------------------------------------------Data PreProcessing

# #---------------Missing Values

# missing\_val = data.frame(apply(dataset,2,function(x){sum(is.na(x))}))

# View(missing\_val)

# #----------------Outlier Analysis

# #BoxPlots - Distribution and Outlier Check

# cnames = colnames(DataConActual)

# for (i in 1:length(cnames))

# {

# assign(paste0("gn",i), ggplot(aes\_string(y = (cnames[i]), x = "cnt"), data = subset(dataset))+

# stat\_boxplot(geom = "errorbar", width = 0.5) +

# geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

# outlier.size=1, notch=FALSE) +

# theme(legend.position="bottom")+

# labs(y=cnames[i],x="cnt")+

# ggtitle(paste("Box plot for",cnames[i])))

# }

# #Viewing plots

# gn1

# gn2

# gn3

# gn4

# #Remove outliers using boxplot method

# df = dataset

# #loop to remove from all variables

# for(i in cnames){

# #print(i)

# val = dataset[,i][dataset[,i] %in% boxplot.stats(dataset[,i])$out]

# #print(length(val))

# dataset = dataset[which(!dataset[,i] %in% val),]

# }

# #Checking for outliers after outlier removal

# for (i in 1:length(cnames))

# {

# assign(paste0("gn",i), ggplot(aes\_string(y = (cnames[i]), x = "cnt"), data = subset(dataset))+

# stat\_boxplot(geom = "errorbar", width = 0.5) +

# geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

# outlier.size=1, notch=FALSE) +

# theme(legend.position="bottom")+

# labs(y=cnames[i],x="cnt")+

# ggtitle(paste("Box plot for",cnames[i])))

# }

# #Viewing plots

# gn1

# gn2

# gn3

# gn4

# #----------------------------------------Feature Selection

# #--------------for continuous data

# # Correlation Plot for continuous data

# cnames = colnames(DataConActual)

# corrgram(dataset[,cnames], order = F,

# upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

# # Dimension Reduction

# dataset = subset(dataset, select = -c(atemp))

# #-----------------For categorical

# ## Chi-square Test for categorical variables

# #1 Comparing "season" "yr" "mnth" "holiday" "weekday" "workingday" "weathersit"

# p1 = chisq.test(table(DataCat$season,DataCat$weathersit))

# p2 = chisq.test(table(DataCat$season,DataCat$yr))

# p3 = chisq.test(table(DataCat$season,DataCat$mnth))

# p4 = chisq.test(table(DataCat$season,DataCat$holiday))

# p5 = chisq.test(table(DataCat$season,DataCat$weekday))

# p6 = chisq.test(table(DataCat$season,DataCat$workingday))

# ##If p-value < 0.05: significant result, reject null hypothesis (H0), dependent.

# #If independent variables are dependent to each other we can drop one of them .

# #If p-value > 0.05: not significant result, fail to reject null hypothesis (H0), independent.

# p1$p.value < 0.05

# p2$p.value < 0.05

# p3$p.value < 0.05

# p4$p.value < 0.05

# p5$p.value < 0.05

# p6$p.value < 0.05

# #Here we got p1 and p3 , which are less than 0.05 .If independent variables are dependent to each other we can drop one of them .

# dataset$weathersit = NULL

# dataset$mnth = NULL

# #2 As we have compared 'season' with all variables,So we will not compare again

# #Comparing "yr" "holiday" "weekday" "workingday"

# p1 = chisq.test(table(DataCat$yr,DataCat$holiday))

# p2 = chisq.test(table(DataCat$yr,DataCat$weekday))

# p3 = chisq.test(table(DataCat$yr,DataCat$workingday))

# ##If p-value < 0.05: significant result, reject null hypothesis (H0), dependent.

# #If independent variables are dependent to each other we can drop one of them .

# #If p-value > 0.05: not significant result, fail to reject null hypothesis (H0), independent.

# p1$p.value < 0.05

# p2$p.value < 0.05

# p3$p.value < 0.05

# #3 As we have compared 'yr' with all variables,So we will not compare again

# #Comparing "holiday" "weekday" "workingday"

# p1 = chisq.test(table(DataCat$workingday,DataCat$weekday))

# p2 = chisq.test(table(DataCat$holiday,DataCat$weekday))

# p3= chisq.test(table(DataCat$holiday,DataCat$workingday))

# ##If p-value < 0.05: significant result, reject null hypothesis (H0), dependent.

# #If independent variables are dependent to each other we can drop one of them .

# #If p-value > 0.05: not significant result, fail to reject null hypothesis (H0), independent.

# p1$p.value < 0.05

# p2$p.value < 0.05

# p3$p.value < 0.05

# dataset$weekday = NULL

# #Dropping extra variables which have been created for transforming normalised data

# dataset$ActualAtemp= NULL

# dataset$ActualTemp= NULL

# dataset$Actualhum= NULL

# dataset$Actualwindspeed= NULL

# dataset$instant = NULL

# dataset$dteday = NULL

# #Dataset after feature Selection

# View(dataset)

# #-------------------------------Feature Scaling

# #Since we already have continuous data given in normalised form .

# #We are passing the same data to our model.

# #------------------------------------------------------------Model Development----------------------------------------------

# #---------------------------------------Linear Regression Model for 'Casual'

# #Sampling

# #Divide the data into train and test

# #y = DependentVariables$dataset.casual

# dataset\_casual = dataset

# dataset\_casual$registered = NULL

# dataset\_casual$cnt = NULL

# View(dataset\_casual)

# TrainData\_index = sample(1:nrow(dataset\_casual), 0.8 \* nrow(dataset\_casual))

# TrainData = dataset\_casual[TrainData\_index,]

# TestData = dataset\_casual[-TrainData\_index,]

# #run regression model

# lm\_model = lm(casual ~., data = TrainData)

# #Summary of the model

# summary(lm\_model)

# #Predict

# predictions\_LR\_casual = predict(lm\_model, TestData[,-8])

# #predictions\_LR\_casual

# #As from summary we can see negative values in predictions

# #Replacing negative predictions with 1 ,Since count can't be negative for bikes rented

# predictions\_LR\_casual[predictions\_LR\_casual < 0] = 1

# predictions\_LR\_casual[predictions\_LR\_casual < 0]

# #Calculate MAE

# MAE( predictions\_LR\_casual,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_LR\_casual,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$casual

# y.pred <- predictions\_LR\_casual

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y +1))^2)))

# }

# rmlse(lm\_model)

# #-------------------------------------------Linear Regression Model for 'registered'

# #Sampling

# #Divide the data into train and test

# #y = DependentVariables$dataset.casual

# dataset\_registered = dataset

# dataset\_registered$casual = NULL

# dataset\_registered$cnt = NULL

# View(dataset\_registered)

# TrainData\_index = sample(1:nrow(dataset\_registered), 0.8 \* nrow(dataset\_registered))

# TrainData = dataset\_registered[TrainData\_index,]

# TestData = dataset\_registered[-TrainData\_index,]

# #run regression model

# lm\_model = lm(registered ~., data = TrainData)

# #Summary of the model

# summary(lm\_model)

# #Predict

# predictions\_LR\_registered = predict(lm\_model, TestData[,-8])

# #predictions\_LR\_casual

# #As from summary we can see negative values in predictions

# #Replacing negative predictions with 1 ,Since count can't be negative for bikes rented

# predictions\_LR\_registered[predictions\_LR\_registered < 0] = 1

# #Calculate MAE

# MAE( predictions\_LR\_registered,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_LR\_registered,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$registered

# y.pred <- predictions\_LR\_registered

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y +1))^2)))

# }

# rmlse(lm\_model)

# #------------------------------------------Decision Tree Regression Model for 'casual'

# #Sampling

# #Divide the data into train and test

# TrainData\_index = sample(1:nrow(dataset\_casual), 0.8 \* nrow(dataset\_casual))

# TrainData = dataset\_casual[TrainData\_index,]

# TestData = dataset\_casual[-TrainData\_index,]

# #Load Libraries

# library(rpart)

# #rpart for regression

# Model\_DT\_casual = rpart(casual ~ ., data = TrainData, method = "anova")

# #Predict for new test cases

# predictions\_DT\_casual = predict(Model\_DT\_casual, TestData[,-8])

# #calculate MAE

# MAE(predictions\_DT\_casual,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_DT\_casual,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$casual

# y.pred <- predictions\_DT\_casual

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y +1))^2)))

# }

# rmlse(Model\_DT\_casual)

# #--------------------------------------------Decision Tree Regression Model for 'registered'

# #Sampling

# #Divide the data into train and test

# TrainData\_index = sample(1:nrow(dataset\_registered), 0.8 \* nrow(dataset\_registered))

# TrainData = dataset\_registered[TrainData\_index,]

# TestData = dataset\_registered[-TrainData\_index,]

# #Load Libraries

# library(rpart)

# #rpart for regression

# Model\_DT\_registered = rpart(registered ~ ., data = TrainData, method = "anova")

# #Predict for new test cases

# predictions\_DT\_registered = predict(Model\_DT\_registered, TestData[,-8])

# #calculate MAE

# MAE(predictions\_DT\_registered,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_DT\_registered,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$registered

# y.pred <- predictions\_DT\_registered

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y+1))^2)))

# }

# rmlse(Model\_DT\_registered)

# #---------------------------------------------Random Forest Regression Model for casual

# #Sampling

# #Divide the data into train and test

# TrainData\_index = sample(1:nrow(dataset\_casual), 0.8 \* nrow(dataset\_casual))

# TrainData = dataset\_casual[TrainData\_index,]

# TestData = dataset\_casual[-TrainData\_index,]

# Model\_RF\_casual = randomForest(formula = casual ~ ., data = TrainData, ntree = 45, method = "anova")

# #Predict for new test cases

# predictions\_RF\_casual = predict(Model\_RF\_casual, TestData[,-8])

# #calculate MAE

# MAE(predictions\_RF\_casual,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_RF\_casual,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$casual

# y.pred <- predictions\_RF\_casual

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y+1))^2)))

# }

# rmlse(Model\_RF\_casual)

# #-----------------------------------------------Random Forest Regression Model for registered

# #Sampling

# #Divide the data into train and test

# TrainData\_index = sample(1:nrow(dataset\_registered), 0.8 \* nrow(dataset\_registered))

# TrainData = dataset\_registered[TrainData\_index,]

# TestData = dataset\_registered[-TrainData\_index,]

# Model\_RF\_registered = randomForest(formula = registered ~ ., data = TrainData, ntree = 45, method = "anova")

# #Predict for new test cases

# predictions\_RF\_registered = predict(Model\_RF\_registered, TestData[,-8])

# #calculate MAE

# MAE(predictions\_RF\_registered,TestData[,8])

# #Calculate RMSE

# RMSE(predictions\_RF\_registered,TestData[,8])

# #Calculate RMSLE

# rmlse <- function(model) {

# y <- TestData$registered

# y.pred <- predictions\_RF\_registered

# return(sqrt(1/length(y)\*sum((log(y.pred +1)-log(y+1))^2)))

# }

# rmlse(Model\_RF\_registered)

# #------------------------------------------------------------Final Model-----------------------------------------

# #We are freezing Linear Regression model

# #As we have observed ERROR MESEARE rates for Linear Regression model are low as compared to other Models.

# #As per problem statement given cnt = casual + Registered

# #cnt: count of total rental bikes including both casual and registered

# #We are calculating predicted cnt with the sum of predicted values of registered and casual variables.

# #Applying Linear Regression on entire DataSet

# #-------------------Predicting casual

# #run regression model

# lm\_model = lm(casual ~., data = dataset\_casual)

# #Predict

# predictions\_LR\_casual = predict(lm\_model, dataset\_casual[,-8])

# predictions\_LR\_casual

# #As from summary we can see negative values in predictions

# #Replacing negative predictions with 1 ,Since count can't be negative for bikes rented

# predictions\_LR\_casual[predictions\_LR\_casual < 0] = 1

# #----------------------Predicting registered

# #run regression model

# lm\_model = lm(registered ~., data = dataset\_registered)

# #Predict

# predictions\_LR\_registered = predict(lm\_model, dataset\_registered[,-8])

# #predictions\_LR\_registered

# #As from summary we can see negative values in predictions

# #Replacing negative predictions with 1 ,Since count can't be negative for bikes rented

# predictions\_LR\_registered[predictions\_LR\_registered < 0] = 1

# #As per problem statement given cnt = casual + Registered

# #cnt: count of total rental bikes including both casual and registered

# #We are calculating predicted cnt with the sum of predicted values of registered and casual variables.

# #Final Count

# Count\_Prediction = round(predictions\_LR\_registered) + round(predictions\_LR\_casual)

# #Creating output file for dataset

# DataTestFinal= read.csv("Day.csv")

# df\_output = data.frame(DataTestFinal)

# df\_output['predicted\_casual'] = round(predictions\_LR\_casual)

# df\_output['predicted\_registered'] = round(predictions\_LR\_registered)

# df\_output['Predicted\_Count'] = Count\_Prediction

# write.csv(df\_output, "Bike\_Renting\_R\_Output.csv",row.names = F)