BIKE RENTING

Dhanush U

26 November 2019

Contents

Chapter 1

Introduction

1.1 Problem Statement

On a daily basis, different kind and different number of users rent bikes. The number of users depend upon various factors such as environmental, seasonal and even holiday factors. The aim of this project is to predict the total number of users who are expected to rent bikes based on these different factors so that the required number of bikes can be made ready daily and this can increase the cost efficiency by reducing the wear and tear and servicing cost of the company and give us an idea about how many bikes should be serviced and kept ready beforehand to meet the demand.

1.2 Data

In this project, our task is to predict the total number of bikes rented on a particular day based on the environmental and seasonal factors. The sample data given below is a sample from the whole population which is used to predict the bike rental count:

Table 1.1 Bike Renting Sample Data (Columns1-8)

instant	dteday	season	yr	mnth	holiday	weekday	workingday
1	01-01-2011	1	0	1	0	6	0
2	02-01-2011	1	0	1	0	0	0
3	03-01-2011	1	0	1	0	1	1
4	04-01-2011	1	0	1	0	2	1
5	05-01-2011	1	0	1	0	3	1

Table 1.2 Bike Renting Sample Data (Columns 9-16)

weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	0.363478	0.353739	0.696087	0.248539	131	670	801
1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
1	0.2	0.212122	0.590435	0.160296	108	1454	1562
1	0.226957	0.22927	0.436957	0.1869	82	1518	1600

From the given data, we have to mainly predict the variable, **'cnt'**, which is the total number of users, which is the sum of casual and registered users on a daily basis. **'Casual'** is the total number of casual users and **'registered'** is the total number of registered users on a particular day. The complete structure of the data is as follows:

Table 1.3 Bike Rental Dataset Structure

Variable	Description	Predictor/Response
Instant	Index of the record	-
Dteday	Date in format (DD/MM/YYYY)	Predictor
Season	Season (1:spring, 2:summer, 3:fall, 4:winter)	Predictor
Yr	Year (0:2011,1:2012)	Predictor
Mnth	Month (1 to 12)	Predictor
Holiday	In weekdays, whether a day is holiday or not (1: holiday, 0: working day)	Predictor
Weekday	Day of the week (0:Sunday::6:Saturday)	Predictor
Workingday	Whether a day is working day or not (1: Working day, 0: Holiday)	Predictor
Weathersit	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	Predictor
Temp	Normalized temperature in Celsius.	Predictor
Atemp	Normalized feeling temperature in Celsius.	Predictor
Hum	Normalized humidity	Predictor
Windspeed	Normalised windspeed	Predictor
Casual	Count of casual users	Response (Predictor for **cnt**)
Registered	Count of Registered users	Response (Predictor for **cnt**)
cnt	Total Count of users on a daily basis	Response

Chapter 2

Methodology

2.1 Pre Processing

Before entering into the modelling phase, initially we have to analyse the data which we have. We need to clean the data, select the required variables for modelling and analyse the data to impute the missing values; detect and handle the outliers; normalise the data for bringing all the data under the same scale for giving equal weightage for all variables. For performing these functions, we can even visualise the data for easy processing using plots and graphs. This process is termed as **Exploratory Data Analysis (EDA).** In this project, response (dependent) variable is a numerical variable and so this is a **regression** problem. For regression, the data must usually be normally distributed. In the following pages, we will look more about the various pre processing techniques which we carry out in this project.

2.1.1 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models. On analysing the data, we find that, there are no missing values in any variables of the given data. It has been represented in a tabulated format as follows:

Table 2.1 Missing Value Analysis Findings (Python and R Code in Appendix B)

S.NO	Variables	Missing values
1	dteday	0
2	season	0
3	yr	0
4	mnth	0
5	holiday	0
6	weekday	0
7	workingday	0
8	weathersit	0
9	temp	0
10	atemp	0
11	hum	0
12	windspeed	0
13	casual	0
14	registered	0

But, incase, if we find any missing values, we can impute those values using any of the central tendency methods (Mean, Median or Mode) or using KNN Imputation method depending upon the data which we handle.

2.1.2 Outlier Analysis

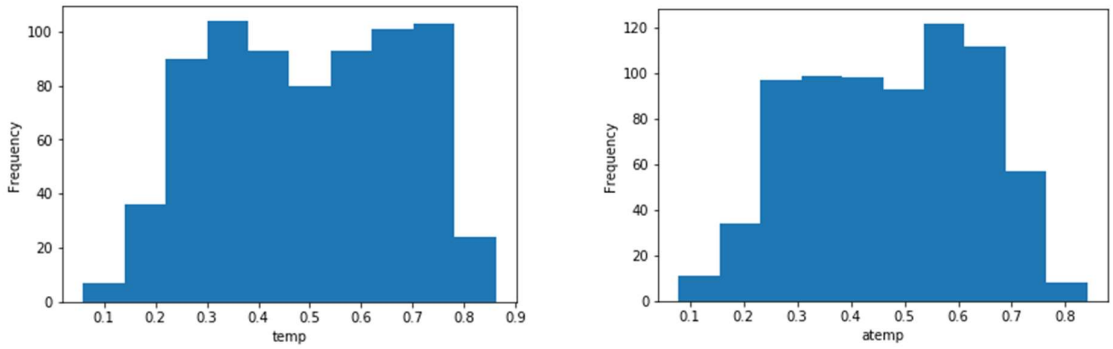
The next pre-processing technique that we usually carry out is the **Outlier Analysis**. Now, our data is free of missing values. Now, Outlier can be defined as a data point that is unusually different when compared to the remaining data of the variable. These Outliers need to be handled before the modelling phase because, these may make the model biased to a particular variable and alter the output.

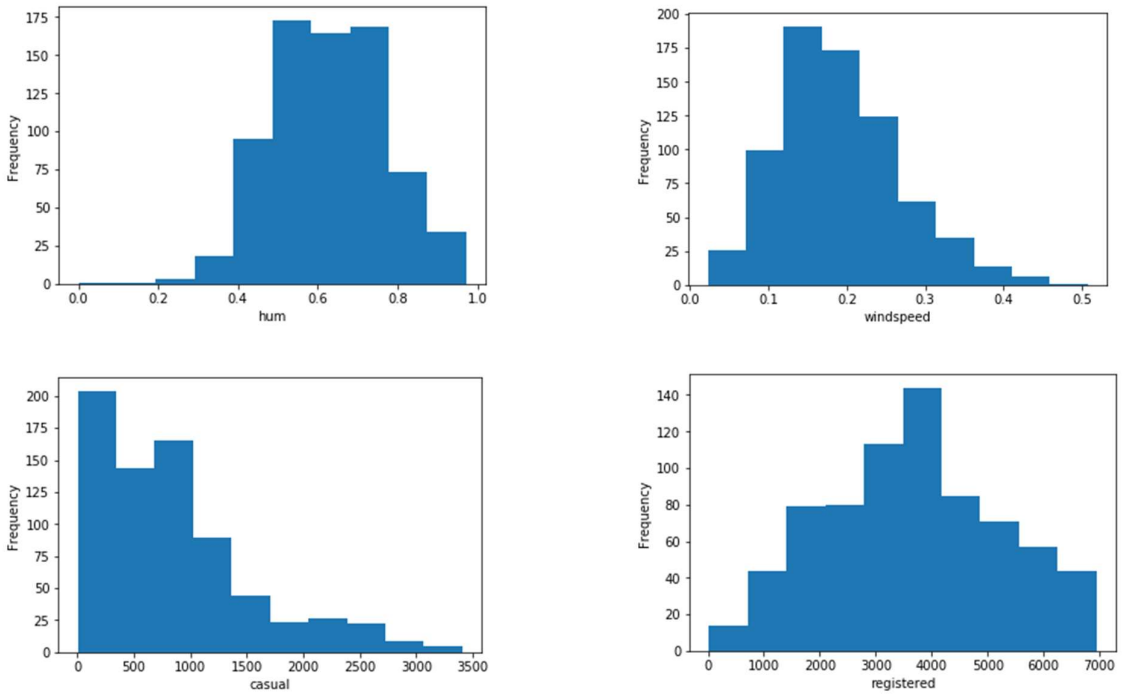
Here, we are going to handle the outliers based on the following steps:

- create the histograms to detect the variables that are skewed and not normally distributed.
- create boxplots to check for outliers.
- check the correlation between the independent and dependent variables before and after the removal of outliers and handle them accordingly.

Note: Outlier Analysis is done only to the **'temp', 'atemp', 'hum', 'windspeed', 'casual' and 'registered'** variables as the remaining are just time period data and seasonal data.

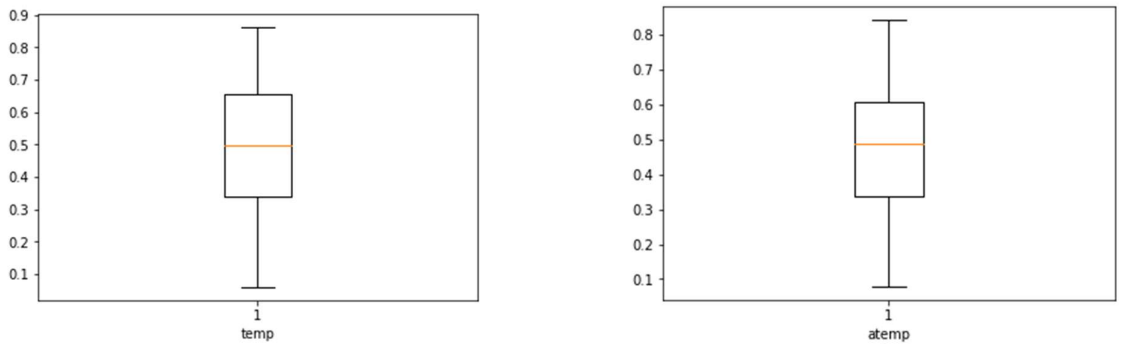
Fig 2.1 Outlier Analysis – Histograms (Python and R Code in Appendix B)

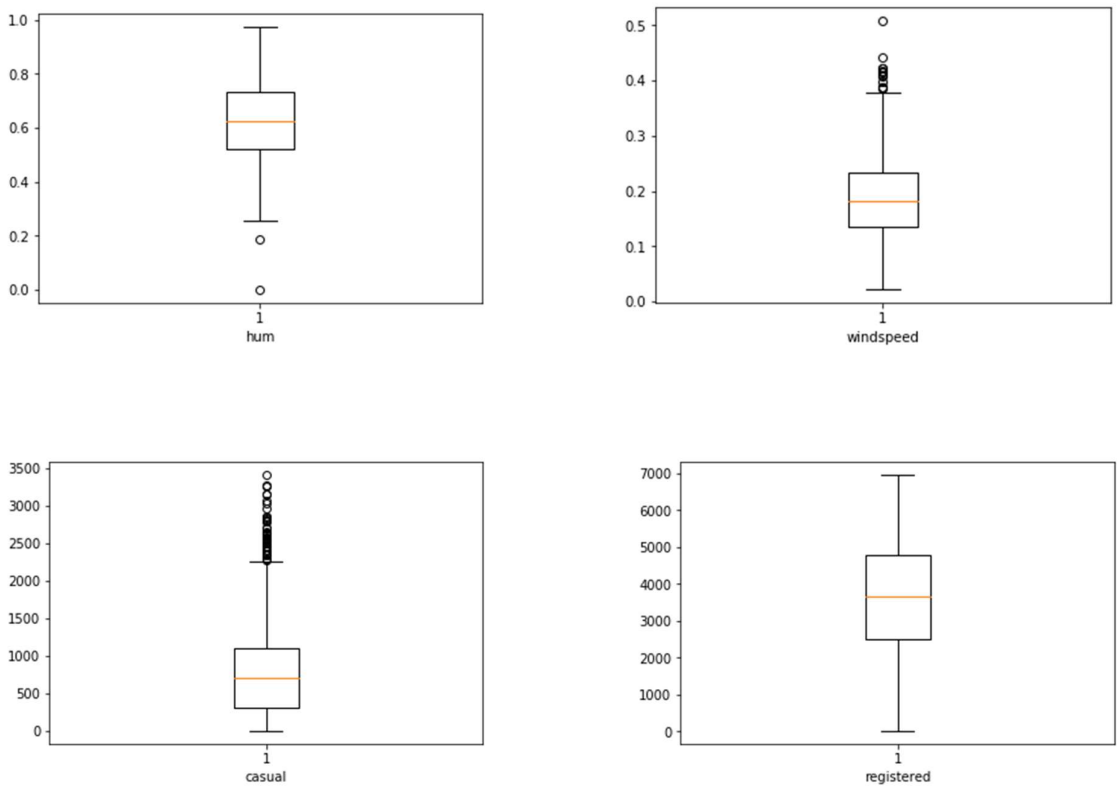




From figure 2.1, we can find that, the histograms of variables ‘hum’, ‘windspeed’ are lightly skewed and ‘casual’ is heavily skewed. So from these histograms, there is a possibility of outliers in these variables. Now let us look upon the **box and whisker plot** for these variables.

Fig 2.2 Outlier Analysis - Box and Whisker Plots (Python and R Code in Appendix B)





Figures 2.1 and 2.2 are the Python plots using matplotlib library. The R plots (Figure A.1, A.2), created by using ggplot2 library, are given in the Appenidix A.

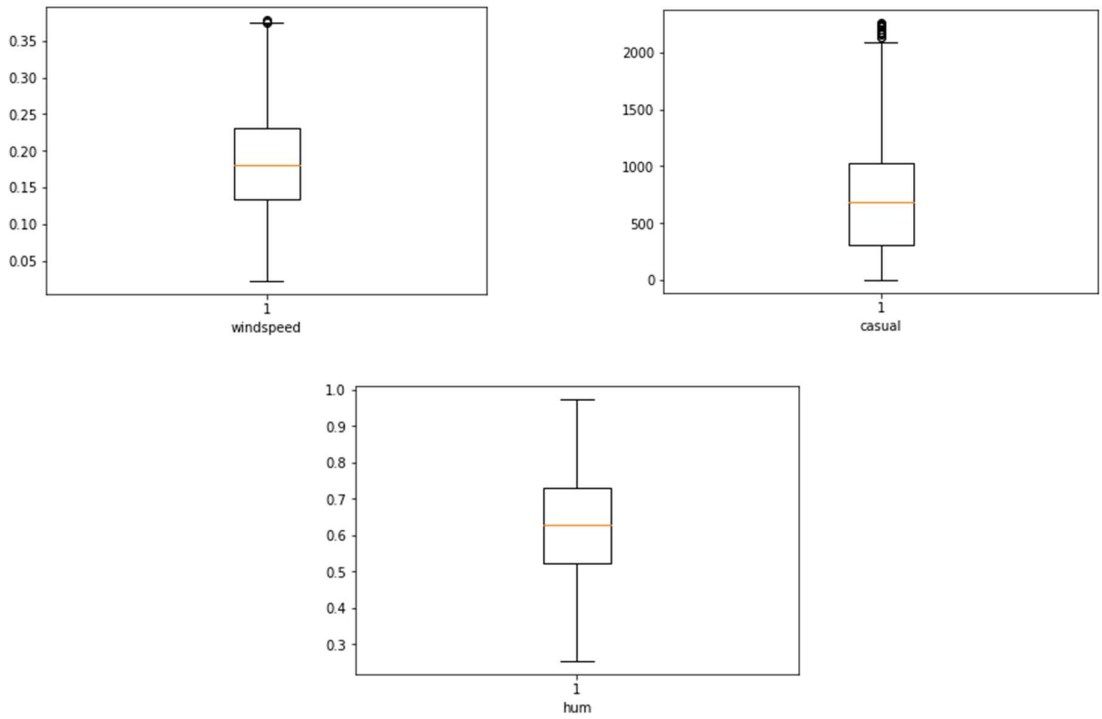
Even the box and whisker plots are stating the same findings as in the histograms. The variables **'hum', 'windspeed' and 'casual'** are having the outliers, while the other three variables are free from outliers.

For these three variables, '**casual'**, **'hum'**, and **'windspeed'**, there are only few values and so they are exceptional when compared to the rest of the data, and so need to be handled, else they may bias the model. After treatment of 'casual' variables, the variables of **'cnt'** are also needed to be changed, as 'cnt' is the sum of 'casual' and 'registered' users.

First, the outliers in variables 'hum' and 'windspeed' are handled and then the 'casual' variable is handled acordingly. Here we are going to delete the outliers and then impute those values using **KNN Imputation** method. KNN stands for **K Nearest Neighbours.** So, based on the neighbouring observations, the values for these outlier points will be imputed by this algorithm.

After imputation of outlier points, we can take the box nd whisker plot of the resulting data and analyse it.

Fig 2.3 Box and Whisker Plot(After Outlier analysis) (Python and R Code in Appendix B)



Figures 2.3 is the Python plot using matplotlib library. The R plot (Figure A.3), created by using ggplot2 library, is given in the Appenidix A

Here, we can find that, the outliers in 'hum' had been removed and outliers in 'windspeed' and 'casual' have been considerably reduced and brought close to the whiskers of the plot. Hence these outliers are not again treated and they are considered as part of data as they wont bias the model. Now, the variable **'cnt'** is also changed based on the **sum of 'casual' and 'registered'**.

Now, as the outliers are handled, we can move on to the next pre-processing technique of **Feature Selection.**

2.1.3 Feature Selection

Machine learning models work on a simple rule – if we put garbage in, we will only get garbage to come out. Here, by garbage, I mean noise in data. This becomes even more important when the number of variables is very large. We need to use only the required number of variables for creating an algorithm, or else, it might lead to **'overfitting'**. **Overfitting** is a modelling error which occurs when a function is too closely fit to a limited set of data points. At times, when building models, less is better.

We should consider the selection of feature for model based on the following criteria

- The correlation between two independent variables should be less and
- The correlation between Independent and Target variables should be high.

Initially, we will form the correlation matrix and then check the correlation between variables using the heatmap as follows.

Fig 2.4 HeatMap (Python and R Code in Appendix B)

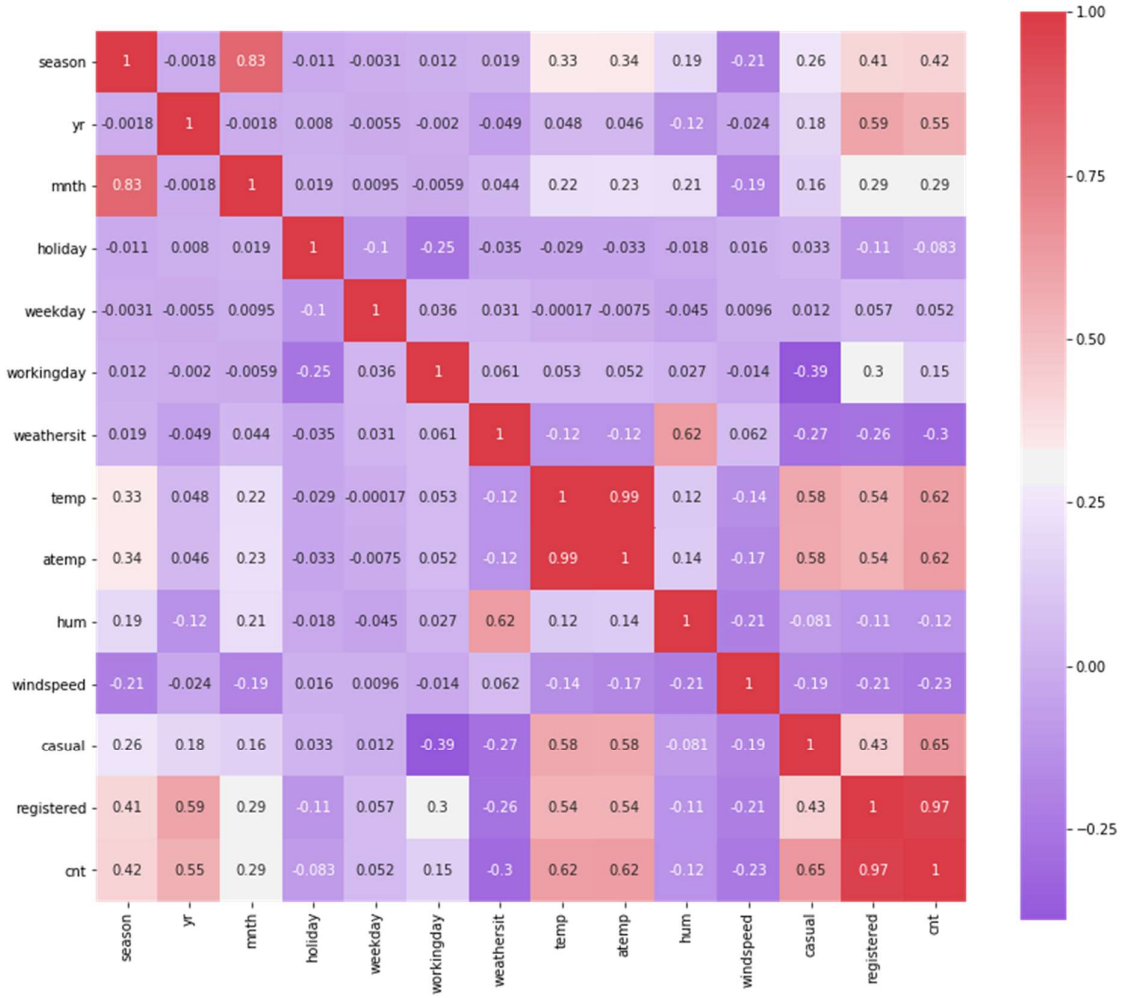


Figure 2.4 is the Python plot using **seaborn** library. The R plot (Figure A.4), created by using ggplot2 library, is given in the Appenidix A

From the heatmap, we can get the inference of the following:

- We know that the correlation between Independent and Target variables should be high (i.e., away from zero). Let us keep a cutoff region of (-0.25 to 0.25). The variables having a correlation with 'casual', 'registered' and 'cnt' variables within this region can be removed while proceeding further while modelling as they describe the target variable very less and so they are less important in prediction of target variable. Under this criteria, we find that the variables **'holiday', 'weekday', 'hum'** and **'windspeed'** fall under this category and they must be removed while modelling.

- Additionally, we know that the correlation between two Independent variables should be low (i.e., close to zero). Let us keep a cutoff region (>0.5 and <-0.5). The independent variables having a correlation in this region should be removed while going into modelling phase, as one among the two variables is sufficient enough to predict the target variable. This will reduce **overfitting.** Under this criteria, we find that the variable pairs ('season' – 'mnth'), ('temp' – 'atemp'), ('weathersit' – 'hum') are highly correlated to each other. We must remove one variable from these pairs, which are comparatively less correlated to the target variable. So, the variables, **'mnth', 'hum'** and **'atemp'** will be removed before going into the modelling phase.

As part of **Dimensionality Reduction**, the above mentioned variables are removed and the dataset is proceeded with the variables **'season', 'yr', 'workingday', 'weathersit', 'temp', 'casual', 'registered', 'cnt'** to the next preprocessing section.

2.1.4 Feature Scaling

Feature Scaling typically means to bring all the variable data under a common scale so that none of the variables overweigh during modelling phase. **For example:** If a column value ranges between 0 to 5 and another column value ranges between 1 million to 10 million, then the second variable will overweigh the model during modelling phase. So reduce this effect, **Feature Scaling** is carried out. Generally Normalization and Standardization are the two types of Feature Scaling methods.

Normalization typically means that the range of values are **normalized** to be from 0.0 to 1.0.

Standardization typically means that the range of values are **standardized** to measure how many standard deviations the value is from its mean.

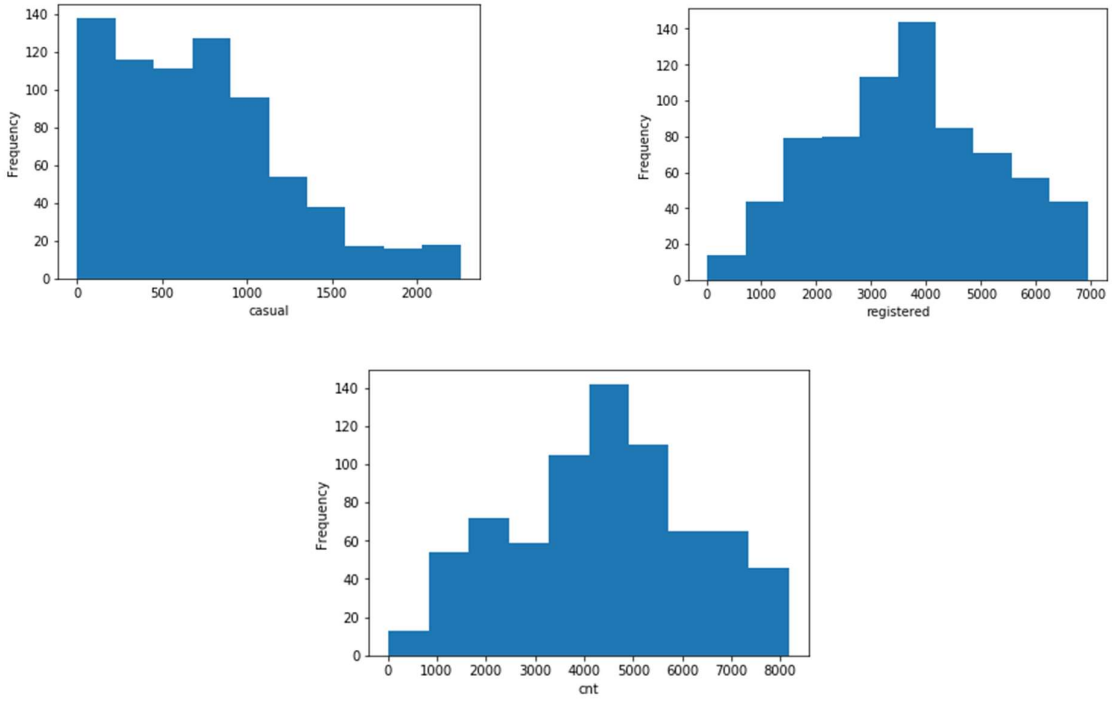
Normalization: $X_{changed} = X - X_{min} / (X_{max} - X_{min})$
Standardization: $X_{changed} = X - \mu / \sigma$

Normally, when the given **data set is uniformly distributed** (i.e., it forms a bell-shaped curve when plotted), we use **standardization**, as it will be easier for us to measure, how many standard deviations away, the particular value is located.

But, when the data set is left skewed or right skewed or **ununiformly distributed** when plotted, we use **Normalization** method, as standardization wont be so efficient as Normalization here due to accumulation of huge data at a particular area.

Here, after Feature Selection; the numerical variable, **temp** is already Normalized and the variables **'casual', 'registered'** and **'cnt'** are to be Normalized as it will be easier for evaluation of models during Modelling phase as all the metrics will be brought under common scale. As the data is not distributed perfectly in uniform, we choose Normalization. The corresponding histograms are again given for reference.

Fig 2.5(A) Histograms – Feature Scaling (Python and R Code in Appendix B)



After normalization, the whole of the data will be scaled according to their correspondig ranges. A sample of it is as follows:

Fig 2.5(B) Normalised Data

	Season	Yr	Workingday	Weathersit	Temp	Casual	Registered	cnt
0	1.0	0.0	0.0	2.0	0.344167	0.145833	0.091539	0.118145
1	1.0	0.0	0.0	2.0	0.363478	0.057181	0.093849	0.095571
2	1.0	0.0	1.0	1.0	0.196364	0.052305	0.174560	0.162802
3	1.0	0.0	1.0	1.0	0.200000	0.046986	0.207046	0.188934
4	1.0	0.0	1.0	1.0	0.200000	0.035461	0.216286	0.193596

At this moment, we have undergone the data into various pre-processing techniques such as **Missing Value Analysis, Outlier Analysis, Feature Selection** and **Feature Scaling.** Now, from the raw data, which we got from the dataset, we have come to a point where we have the data made ready to be fed into a machine learning model and train it to make predictions in the future using the same model.

These pre-processing techniques have made it possible to increase the accuracy and reduce the inefficiency of model due to noise (errors) in the data. Now, we can move on to the next phase of Modelling where we have to build a model based on the data and make predictions if data is fed into it in the future.

2.2 Modeling

The next phase our project is the Modeling phase. Till now, our data is pre-processed and it is made ready to train the model to make predictions. But initially we must be specific on which model to be used based on our dataset.

2.2.1 Model Selection

Based on our dataset, we have to decide which model have to be selected for our dataset. For model development, dependent variable may fall under one of the below categories

- Nominal
- Ordinal
- Interval
- Ratio

For our data, dependent variable falls under **interval** category and so, the predictive analysis that we can perform is **Regression** Analysis. There are various algorithms and statistical models for such regression problems. Here we will use the following models for predicting bike rental count.

- Decision Tree
- Random Forest
- Linear regression

2.2.2 Model Evaluation

Generally, for classification problems, we will be having the dependent variable as ‘Yes’ or ‘No’ or say, only two outcomes. In such cases, we can classify and evaluate the model based on the outcomes by forming the confusion matrix.

But, in regression problems, the dependent variable will be continuous and confusion matrix can't be formed in such cases. So we have to employ some other error metrics to evaluate the model performance. Some of the common error metrics we use in Regression Problems are **Mean absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Mean Square Error (MSE), Root Mean Square Error (RMSE).** Here, we will use two of these Error Metrics, **MAPE** and **RMSE** for evaluating our models.

Mean Absolute Percentage Error (MAPE) is commonly used as a loss function for regression problems and in model evaluation, because of its very intuitive interpretation in terms of relative error. The MAPE measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error, as shown in the example below:

$$\left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right) * 100$$

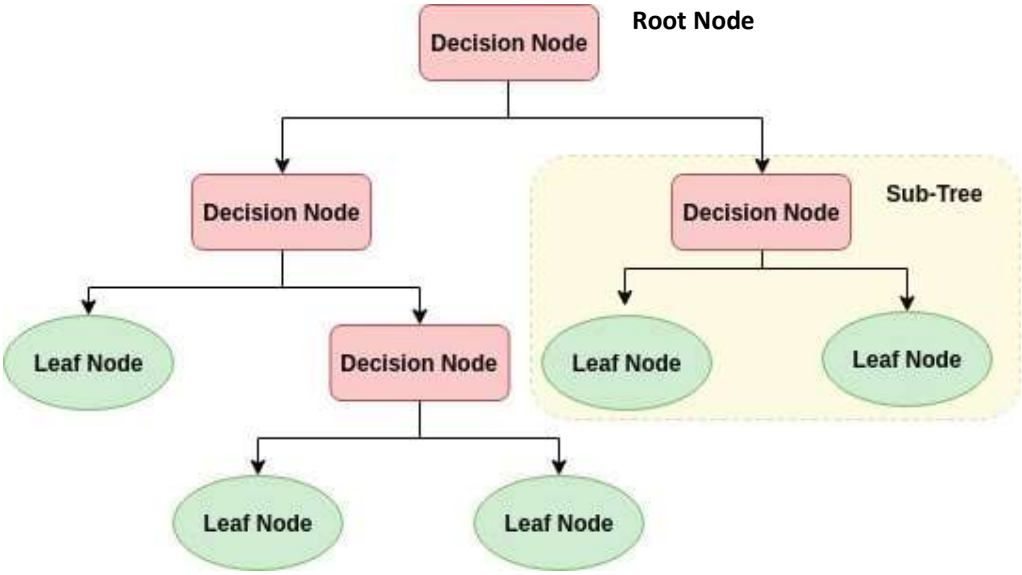
Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells us how concentrated the data is around the line of best fit.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{act} - X_{forecast})^2}{n}}$$

2.2.3 Decision Tree

The first ML algorithm, that we are going to use here is the **Decision Tree Algorithm.** Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set 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. **Leaf node** represents a classification or decision. 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.

Fig 2.6 Structure of a Decision Tree



So here we can use the Decision Tree for Regression and predict the Bike Rental count based on environmental and seasonal settings.

2.2.3.1 Implementation

Here we are going to use the **DecisionTreeRegressor** for building the model in Python and **RPart** for building model in R. They use **CART Algorithm for formation of trees** and using test_train_split method, we can sample the data into 80% for trining and 20% for testing in Python

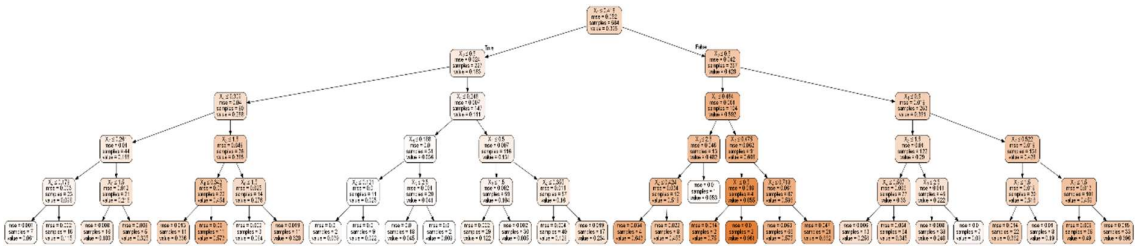
There are various sampling methods such as Simple Random Sampling, Stratified Sampling which can also be used to generate sampled data. The test_train_split method wil randomly shuffle the data and pick the data accordingly.

After the model is built, using the independent variables of test data, we will predict the dependent variables and then evaluate the error metrics for this model.

Here, we are going to build models for casual and registered user count and even the total count and check the model performance in predictig all these variables from the weather and season data.

Here we can limit the depth of the tree to keep constraints in developing the model, otherwise the tree will develop until it reaches exact leaf node for all branches. When we tried with different depths for the decision tree, a **tree depth of 5** was optimum for this dataset as after that if the tree depth is increased, the output change was not phenominal. Below, we can see the **sample decision tree generated for the casual model**

Fig 2.7 Decision Tree For Casual model (Python Code in Appendix B)



We sampled the data and built the model and the model made predictions and we evaluated the model based on the predictions made. The results are as follows:

```
MAPE
Casual - 48.81357644542093
Registered - 22.42690593083405
Cnt - 20.312685852589873
RMSE
Casual - 0.14689939052881815
Registered - 0.12027895252184015
Cnt - 0.11229041865121038
```

Here, we can see, the MAPE and RMSE shows that the model's accuracy is good but not up to the mark. But before deciding on the performance, we can look into the other two models and their performance on this dataset.

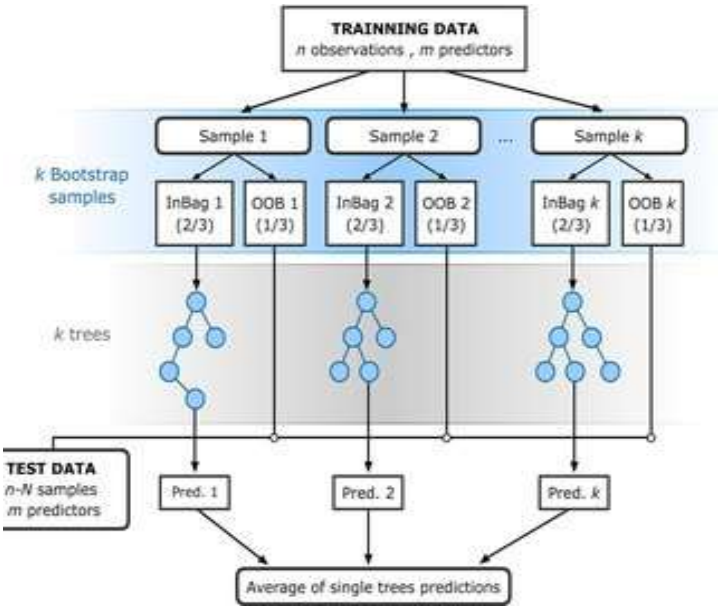
2.2.4 Random Forest

The next regression model that we are going to use to predict the target variables is the Random Forest model. Random Forest is a supervised learning algorithm. From it's name, we can sense that it creates a forest and makes it somehow random. The forest it builds, is an **ensemble of Decision Trees**, most of the time trained with the "**bagging**" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random Forest basically is a collection of many decision trees and it combines the result of all the decision trees to get more accurate prediction of dependent variable.

Random Forest algorithm can be used for both Classification and Regression problems and it is a great advantage for this algorithm. Here, we can use this algorithm to predict the dependent variable by regression.

Fig 2.8 Structure of a Random Forest



Here we can use the Random Forest for Regression and predict the Bike Rental count based on environmental and seasonal settings

2.2.4.1 Implementation

Here we are going to use the **RandomForestRegressor** for building the model in Python and **RPart** for building model in R. We can use the same sampled data we used for decision trees, here too for making predictions.

As said previously, Random Forest is an ensemble of Decision Trees. We can specify and decide on the number of trees to be used in the algorithm. On trying out different number of trees for this dataset, **10 trees** is decided for building this model as above 10, the result of the predictions didn't change much.

Usually, ensemble models perform better when compared to the normal ML algorithms. But it doesn't guarantee that ensemble models are always the better ones.

We sampled the data and built the model and the model made predictions and we evaluated the model based on the predictions made. The results are as follows:

```
MAPE
Casual - 45.15864398383852
Registered - 17.980290446016873
Cnt - 17.84105905024535
RMSE
Casual - 0.1494970815324995
Registered - 0.09587161057236272
Cnt - 0.09496679391120676
```

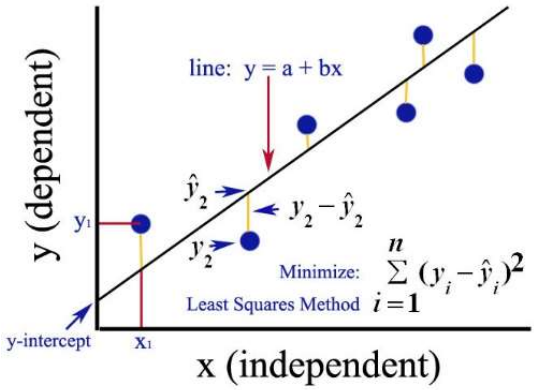
Here, we can see, the MAPE and RMSE shows that the model's accuracy has increased when compared to the decision tree. So for this data, the ensemble model performs better. But before deciding on the performance, we can look into the linear regression model and its performance on this dataset.

2.2.5 Linear regression

Linear regression is used for finding linear relationship between target and one or more predictors. There are two types of linear regression- Simple and Multiple. Simple Linear Regression means there will be one predictor and one response variable. Multiple Linear Regression is the current scenario which we face where there will be many predictors and a single target variable.

Linear Regression employs various methodologies to formulate the linear relationship and one among them is **the Least Squares Method**, where, there will be a linear line formed, which has the least squared error from the actual values, when compared to any other line formed for the data. A linear regression looks something similar to the below:

Fig 2.9 Structure of Linear Regression based on Least Squares



2.2.5.1 Implementation

Here we use **OLS(Optimum Least Square)** method in Python and R to develop the model. This method formulates the line with least squared error.

We sampled the data and built the model and the model made predictions. We can visualise the summary of the model using model method. For example, here we can see the model summary of cnt_model:

Fig 2.10 (A) cnt_model Summary

OLS Regression Results

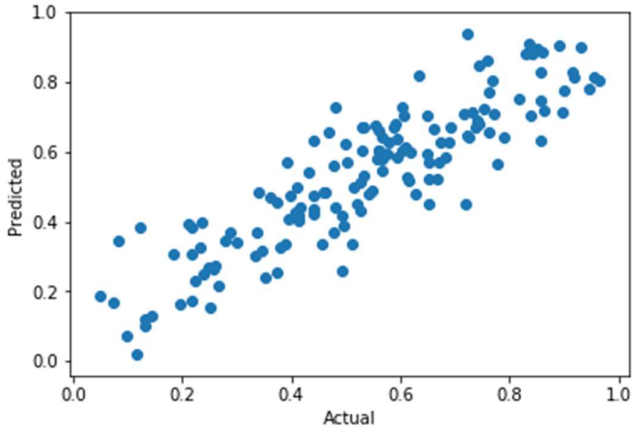
Dep. Variable:	cnt	**R-squared:**	0.966
Model:	OLS	**Adj. R-squared:**	0.965
Method:	Least Squares	**F-statistic:**	3253.
Date:	Wed, 29 May 2019	**Prob (F-statistic):**	0.00
Time:	22:30:36	**Log-Likelihood:**	471.29
No. Observations:	584	**AIC:**	-932.6
Df Residuals:	579	**BIC:**	-910.7
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
season	0.0588	0.004	14.260	0.000	0.051	0.067
yr	0.2417	0.009	27.440	0.000	0.224	0.259
workingday	0.0784	0.009	8.280	0.000	0.060	0.097
weathersit	-0.0766	0.007	-11.561	0.000	-0.090	-0.064
temp	0.6418	0.023	28.270	0.000	0.597	0.686

Omnibus:	100.282	**Durbin-Watson:**	1.896
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	224.385
Skew:	-0.921	**Prob(JB):**	1.89e-49
Kurtosis:	5.415	**Cond. No.**	16.5

From the model summary, we can see that the **R-squared and Adjusted R-squared** values are close to 1. This shows that the model developed can determine 95-96% of the predicted values. Similarly the column **'coef'**explains 1 unit of a variable brings about how many units of change in the predicted variable. Based on this, the highest change (i.e., more correlation) is brought about by the **'temp'** column. The **p-value** column is also negligible for all variables and so none of the variables are unnecessary in predicting the target variable. The scatter plot between 'actual' and 'predicted' values of cnt_model is shown below:

Fig 2.10 (B) Actual vs Predicted (cnt_model)



The Python and R code for figures 2.10 (A) & (B) is given in Appendix B

The linear relationship in the scatter plot shows that the predicted values are close to the actual values. If more data is fed in, the model performance can be improved. We evaluated the model based on the predictions made. The results are as follows:

```
MAPE
Casual - 65.02613561697862
Registered - 21.23683334237258
Cnt - 22.720081671830826
RMSE
Casual - 0.15812334399529426
Registered - 0.08875489435017742
Cnt - 0.10418189613032994
```

Here, we can see, the MAPE and RMSE shows that the model's performance is not up to the mark when compared when compared to the random forest. So for this data, the ensemble model performs better.

Chapter 3

Conclusion

3.1 Model Selection

Here initially we performed various pre-processing techniques on the data as part of dta cleaning and we made the data ready for Modeling phase. Then we build models using three different algorithms to decide which odel is better in predicting for this data. So finally we have developed the models and we have evaluated all the three models.

Based on the error metrics, though Decision Tree and Linear regression algorithms are able to predict the target variable, they are not better when compared to **Random Forest model** in predicting the target variable for this dataset as the error metrics show that there is less error in Random Forest when compared to the other two models.

Even for the R-code, the error metrics are as follows when the models are run:

Decision Tree:

```
#RMSE
#"casual -  0.120012777101718"
#"registered -  0.106531368734166"
#"cnt -  0.0964989527097329"

#MAPE
#"casual -  47.6220988759675"
#"registered -  21.4695856369951"
#"cnt -  18.4708929146792"
```

Random Forest:

```
#RMSE
#"casual -  0.14501263201758"
#"registered -  0.096473865409887"
#"cnt -  0.0885746302777854"

#MAPE
#"casual -  43.5856369951654"
#"registered -  18.2914679285633"
#"cnt -  17.4439937438519"
```

Linear Regression:

```
#RMSE
#"casual -  0.128242142735652"
#"registered -  0.10249082905932"
#"cnt -  0.107447951947048"

#MAPE
#"casual -  54.2798720007824"
#"registered -  18.8597522591259"
#"cnt -  18.1615320907755"
```

So, for this dataset, we use **Random Forest model** to predict the target variable.

3.2 Output

Using Random Forest Model in R and Python, let us predict the bike rental count for a particular environmental and seasonal settings.

For sample input, we are fetching the 100^{th} , 200^{th} , 300^{th} , 400^{th} and 500^{th} rows. Using these observations, we are predicting the dependent variables. Here we have already normalized the variables and so the predicted values will also be normalised values.

The output in Python is stored in **Python Sample Output.csv** file and it is as follows:

Fig 3.1 (A) Python Output (Python Code in Appendix B)

	season	yr	workingday	weathersit	temp	casual	registered	cnt	casual_pd	registered_pd	cnt_pd
0	2.0	0.0	0.0	2.0	0.426667	0.525709	0.243575	0.352472	0.474158	0.259500	0.329444
1	3.0	0.0	1.0	1.0	0.776667	0.332447	0.544181	0.554411	0.341829	0.541597	0.554417
2	4.0	0.0	1.0	2.0	0.470000	0.105496	0.346376	0.323519	0.144947	0.399653	0.390222
3	1.0	1.0	0.0	2.0	0.264167	0.169326	0.350563	0.344743	0.168440	0.342478	0.327862
4	2.0	1.0	1.0	2.0	0.573333	0.150709	0.358215	0.346093	0.239184	0.427346	0.465734

The output in R is stored in **R Sample Output.csv** file and it is as follows:

Fig 3.1 (B) R Output (R Code in Appendix B)

	season	yr	workingday	weathersit	temp	casual	registered	cnt	casual_pd	registered_pd	cnt_pd
100	2	0	0	2	0.426667	0.5257092	0.2435749	0.3524721	0.4898520	0.3303471	0.3987152
200	3	0	1	1	0.776667	0.3324468	0.5441813	0.5544105	0.3358633	0.5672666	0.5711866
300	4	0	1	2	0.470000	0.1054965	0.3463760	0.3235186	0.2564771	0.4953984	0.4614861
400	1	1	0	2	0.264167	0.1693262	0.3505631	0.3447430	0.3004618	0.3502649	0.3155985
500	2	1	1	2	0.573333	0.1507092	0.3582154	0.3460925	0.3517649	0.6535249	0.7005580

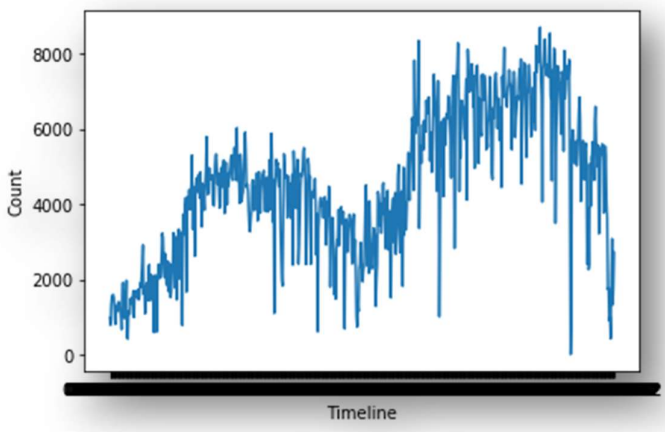
Thus the model is generated based on the data and the output is predicted by the model and stored in the csv files.

3.3 Interesting Patterns

Adding to the model which we have developed, let us analyse few interesting patterns from the dataset.

3.3.1 Relationship between dteday and cnt

Fig 3.2 Relationship between dteday and cnt (Python Code in Appendix B)

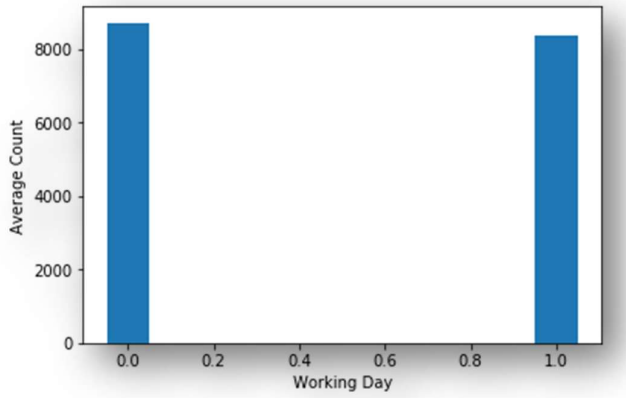


The plot between timeline and count shows that during the mid-years i.e., (Apr-Sep) the count of rental bikes increase and in the beginning and end of the years, they decrease.

So, we can increase the availability of bike count during mid-years and reduce the availability at the beginning and end of years. This is done to efficiently manage the demand and reduce the maintenance cost.

3.3.2 Relationship between workingday and cnt

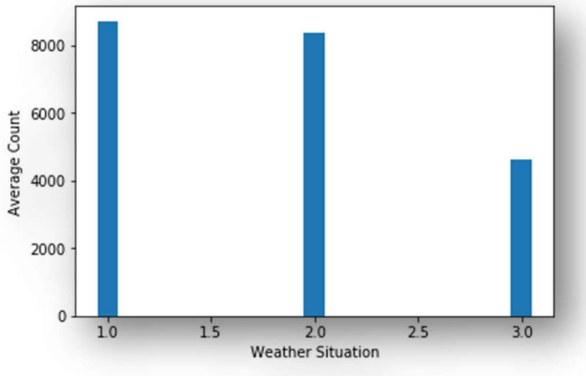
Fig 3.3 Relationship between workingday and cnt (Python Code in Appendix B)



Based on this Bar Chart, there is no significant difference in total rental bikes used by customers on working days and weekends. So during weekends, there is no need of increasing the availability of rental bikes.

3.3.3 Relationship between weathersit and cnt

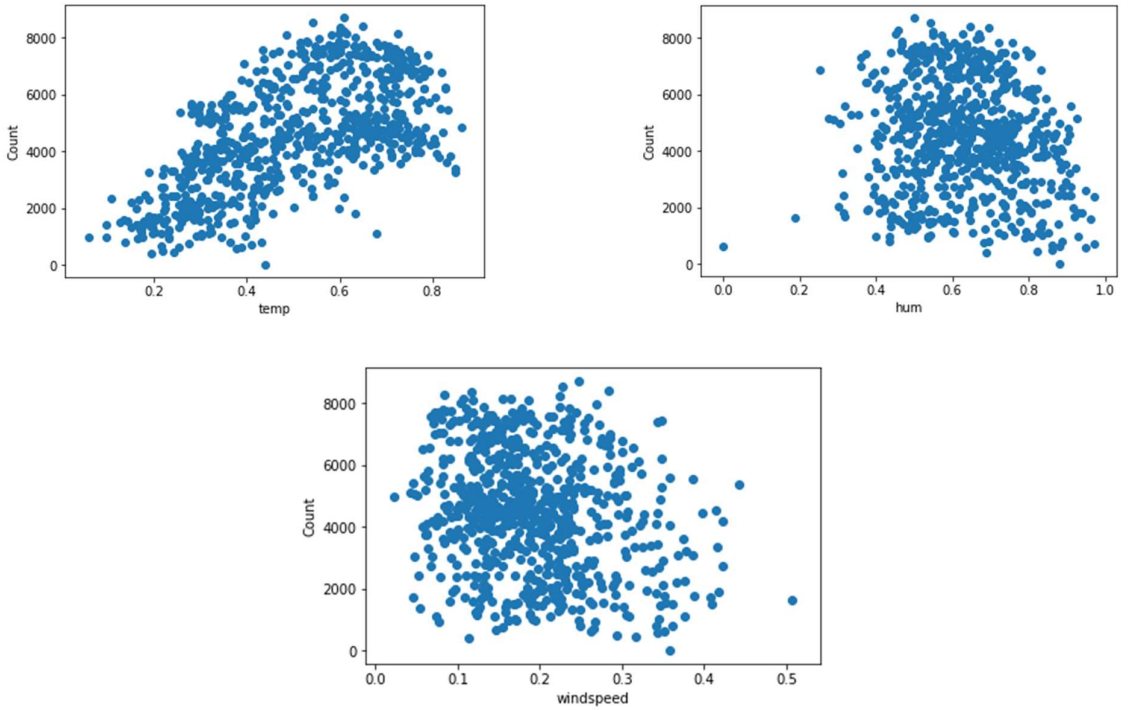
Fig 3.4 Relationship between weathersit and cnt (Python Code in Appendix B)



This Barchart shows that during weather situation 3 (Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds), the total rented bikes has become nearly half when compared to the normal days. So during rainy days, the count of bikes rented will be reduced. So, accordingly during rainy days, we can plan to utilise those days for maintenance purposes.

3.3.4 Relationship between (temp/hum/windspeed) and cnt

Fig 3.5 Relationship between (temp/hum/windspeed) and cnt(Python Code in Appendix B)



The relationship between Temperature and Count shows that during hotter days, the count of rented bikes is higher when compared to colder days. But the other two scatter plots show that the count of rental bikes is not related to 'hum' and 'windspeed' variables as data is randomly scattered in these plots.

Thus we have trained a model which can predict the cost of rental bikes based on environmental and seasonal settings.

Appendix A - (Extra figures)

Fig A.1 Outlier Analysis – Histogram (R Plot)

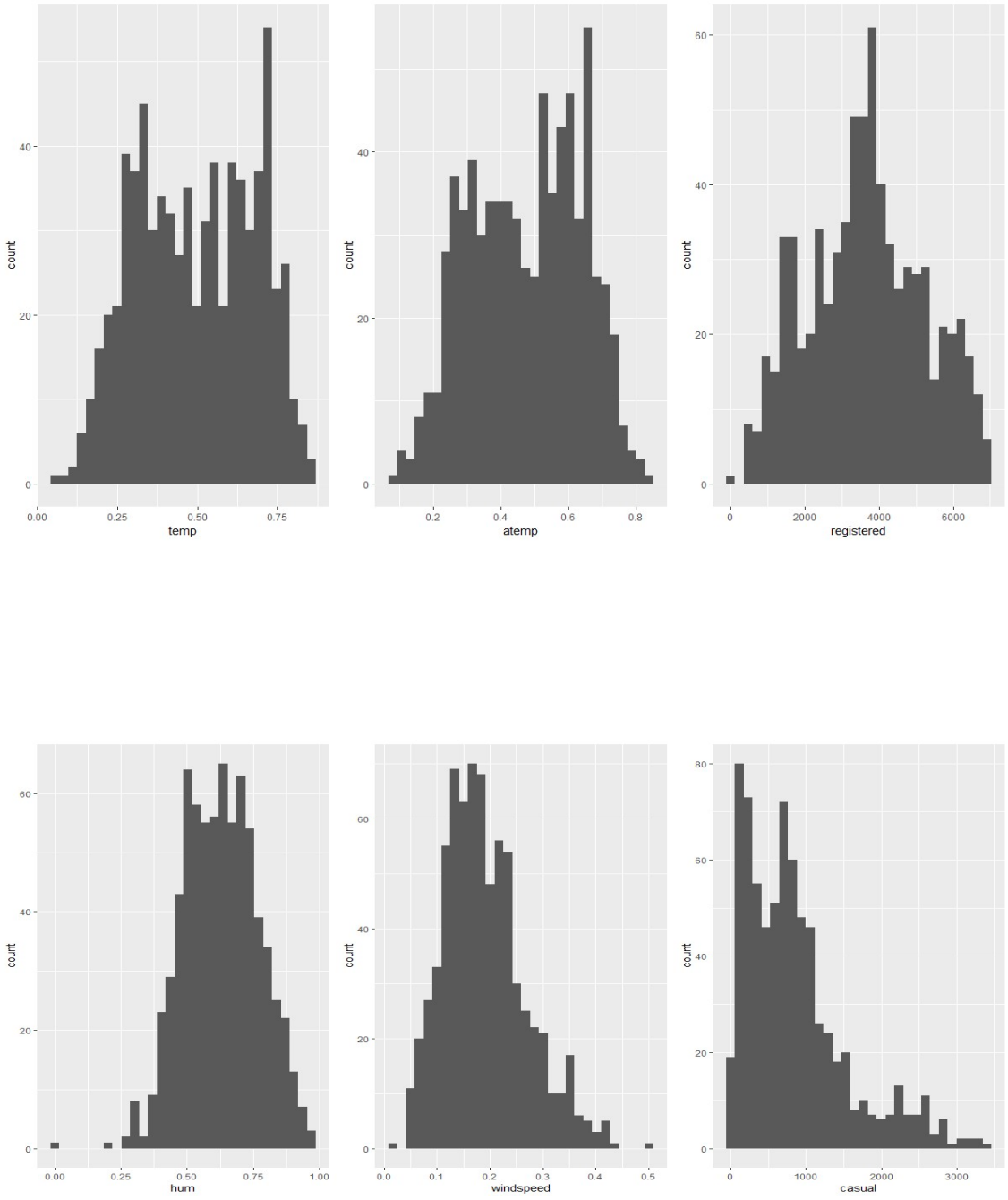


Fig A.2 - Outlier Analysis - Box and Whisker Plots (R Plot)

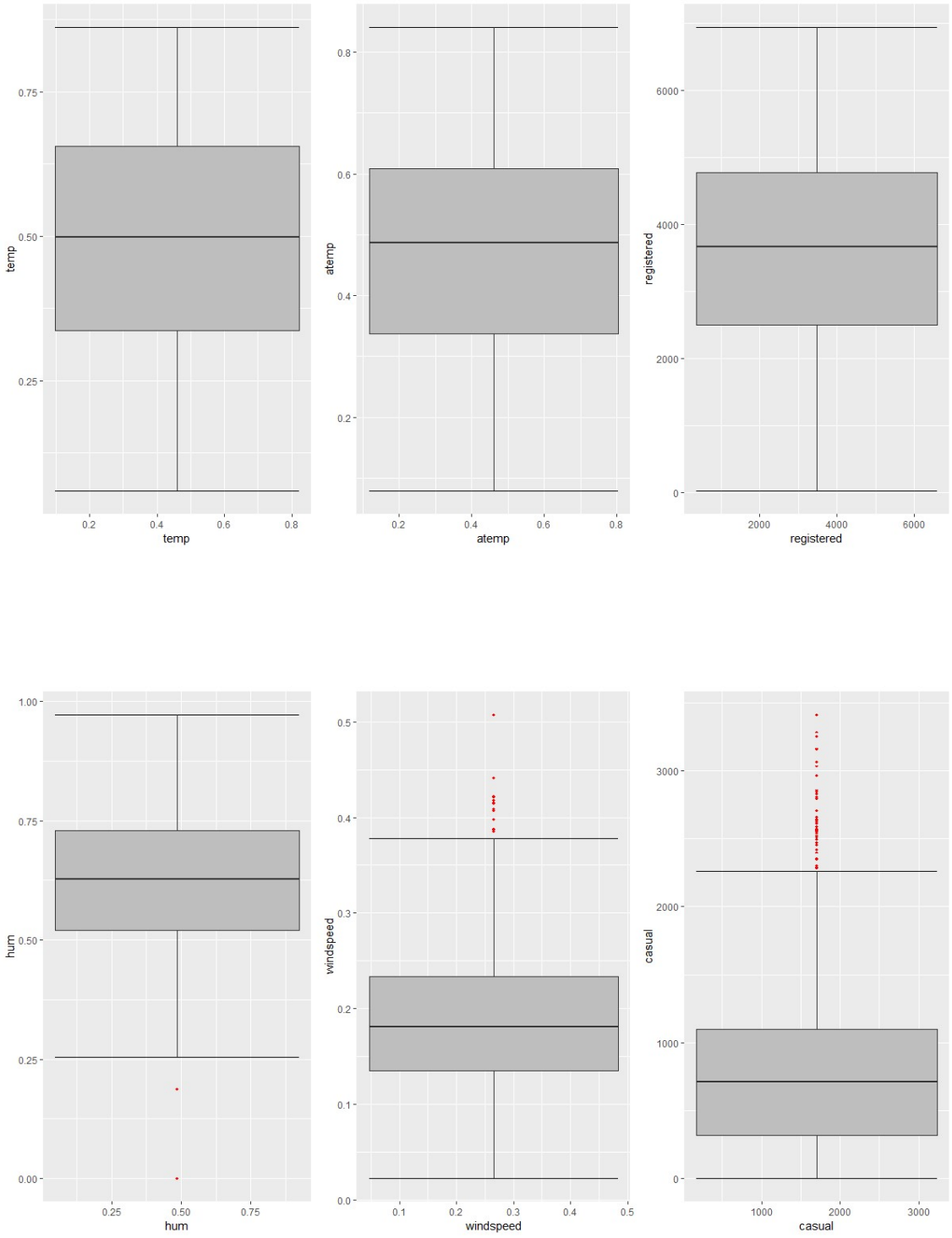


Fig A.3 - Box and Whisker Plot(After Outlier analysis) (R Plot)

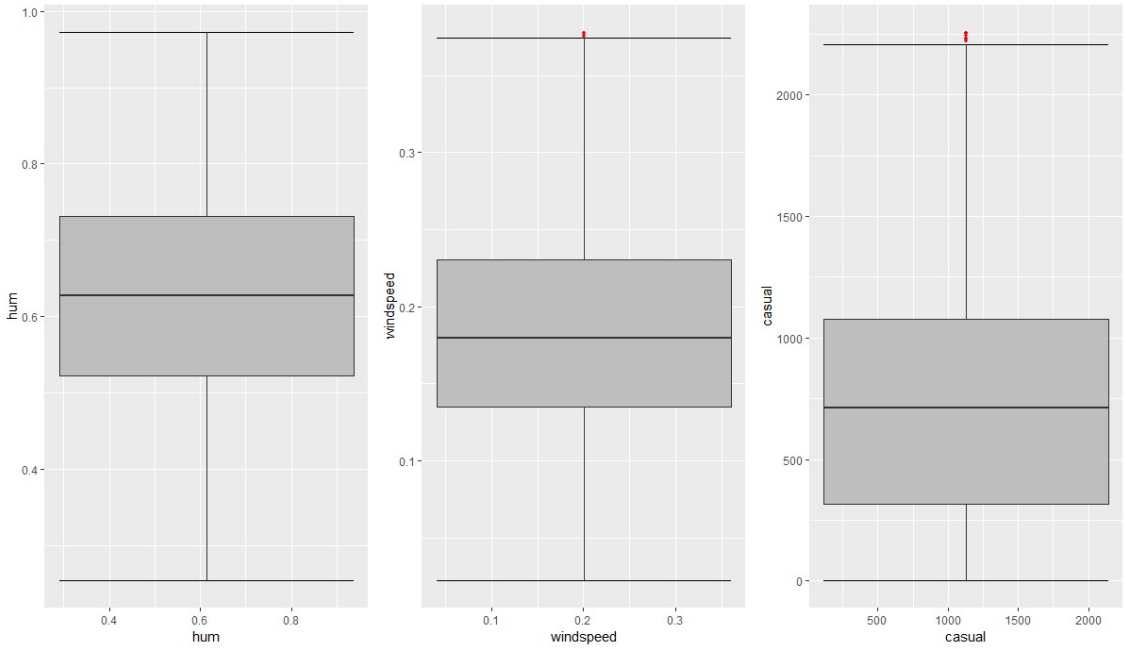
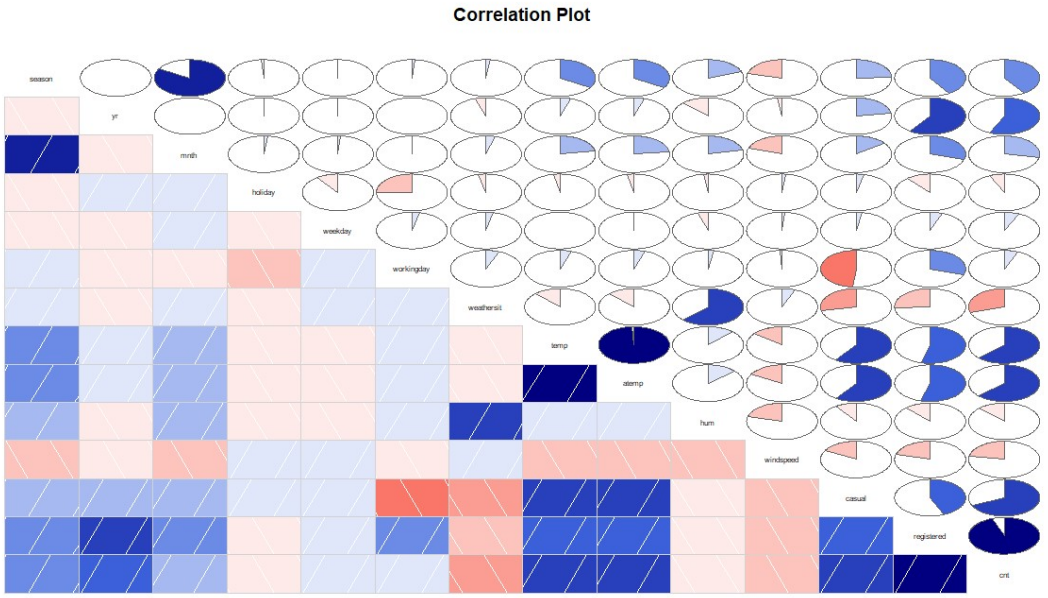


Fig A.4 HeatMap (R Plot)



Appendix B – (Code)

Table 2.1 Missing Value Analysis Findings

Python Code

```
### Missing Value Analysis

In [13]: missing_val = pd.DataFrame(rental_data.iloc[:,1:15].isnull().sum())

In [14]: missing_val

Out[14]:
```

	0
dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
casual	0
registered	0

R Code

```
############################## Missing Value Analysis ##################################

missing_val = data.frame(apply(rental_data[,2:15],2,function(x){sum(is.na(x))}))
colnames(missing_val)="Total Missing Values"
```

	Total Missing Values
dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
casual	0
registered	0

Fig 2.1 Outlier Analysis – Histograms (Python Code)

```
### Outlier Analysis

#Creating Histogram to check the data distribution
for i in range(9,15):
    plt.hist(rental_data.iloc[:,i])
    plt.xlabel(rental_data.columns[i])
    plt.ylabel('Frequency')
    plt.show()
```

Fig A.1 Outlier Analysis – Histograms (R Code)

```
#Creating Histograms to check the data distribution
for (i in 10:15)
{
  assign(colnames(rental_data)[i],ggplot(rental_data,aes_string(x=rental_data[,i]))+
           labs(x=colnames(rental_data)[i])+geom_histogram())
}

gridExtra::grid.arrange(temp, atemp, registered, ncol=3)
gridExtra::grid.arrange(hum, windspeed, casual, ncol=3)
```

Fig 2.2 Outlier Analysis – Box and Whisker Plots (Python Code)

```
#Creating Box and Whisker plots to check for outliers
for i in range(9,15):
    plt.boxplot(rental_data.iloc[:,i])
    plt.xlabel(rental_data.columns[i])
    plt.show()
```

Fig A.2 Outlier Analysis – Box and Whisker Plots (R Code)

```
#Creating Box and Whisker plots to check for outliers
for (i in 10:15)
{
  assign(colnames(rental_data)[i],
         ggplot(rental_data,aes_string(x=rental_data[,i],y=colnames(rental_data)[i]))+
           labs(x=colnames(rental_data)[i])+
           stat_boxplot(geom = "errorbar", width = 0.5) +
geom_boxplot(outlier.colour="red",fill = "grey" ,outlier.shape=18, outlier.size=1, notch=F))
}
```

Fig 2.3 Box and Whisker Plot(After Outlier analysis) (Python Code)

```
#Creating Box and Whisker plots to check for outliers after outliers are removed
for i in cnames:
    plt.boxplot(df.loc[:,i])
    plt.xlabel(i)
    plt.show()
```

Fig A.3 Box and Whisker Plot(After Outlier analysis) (R Code)

```
#Box and whisker plot to visualise after outlier analysis
for (i in cnames){
  assign(i,ggplot(df,aes_string(x=df[,i],y=i))+
  labs(x=i)+stat_boxplot(geom = "errorbar", width = 0.5) +
  geom_boxplot(outlier.colour="red",fill = "grey" ,outlier.shape=18, outlier.size=1, notch=F))
}

gridExtra::grid.arrange(hum, windspeed, casual, ncol=3)
```

Fig 2.4 Heat Map (Python Code)

Feature Selection

Correlation analysis for forming the heatmap based on correlation matrix

```
#Formation of Correlation Matrix

cor = rental_data.iloc[:,2:].corr()
cor
```

...

```
#Fixing the figure size and generating heatmap

f, ax = plt.subplots(figsize=(14, 12))
sns.heatmap(cor,square=True,annot=True,cmap=sns.diverging_palette(1000, 10, as_cmap=True), ax=ax)
```

Fig A.4 Heat Map (R Code)

```
################################### Feature Selection ###################################

#Formation of Correlation matrix and heatmap based on the correlation between variables
corrgram(df[,3:16], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
```

Fig 2.5(A)&(B) Feature Scaling

```
### Feature Scaling

#Variables casual and registered are normalised as the histograms show that data is not uniformly distributed
norm_var = ["casual","registered"]
for i in norm_var:
    plt.hist(rental_data[i])
    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.show()
    rental_data[i] = (rental_data[i] - min(rental_data[i]))/(max(rental_data[i]) - min(rental_data[i]))
print(rental_data.head(5))
```

Fig 2.7 Decision Tree For Casual model (Python Code)

```
#To visualise a decision tree
dot_data = StringIO()
export_graphviz(casual_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pdp.graph_from_dot_data(dot_data.getvalue())
graph.write_png("DT.png") #The decision tree is exported to the directory as png file
Image(graph.create_png())
```

Fig 2.10 (A) & (B) cnt_model Characteristics and Scatter Plot of Predicted Vs Actual

```
#Model Summary of cnt model
cnt_model.summary()
```

```
#Scatter plot to visualise cnt variable predictions
plt.scatter(test_data.iloc[:,7],cnt_predictions)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

Fig 3.1 (A) Python Output

```
#Sample input of 5 rows is fetched from the dataset
sample_input = rental_data.iloc[[99,199,299,399,499],:]

#Random Forest model is built using the complete dataset
casual_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,5])
registered_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,6])
cnt_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,7])

#Using these models the dependent variables are predicted for the sample_input
casual_predictions = casual_model.predict(sample_input.iloc[:,0:5])
registered_predictions = registered_model.predict(sample_input.iloc[:,0:5])
cnt_predictions = cnt_model.predict(sample_input.iloc[:,0:5])
```

```
#Output dataframe is created and it is stored in csv file and exported
sample_output = pd.concat([sample_input.reset_index(drop=True), \
    pd.DataFrame({'casual_pd':casual_predictions,'registered_pd':registered_predictions,'cnt_pd':cnt_predictions})],axis = 1)
sample_output.to_csv("Python Sample Output.csv",index=False)
sample_output
```

Fig 3.1 (B) R Output

```
#Sample input of 5 rows is fetched from the dataset
sample_input = df[c(100,200,300,400,500),]

#Random Forest model is built using the complete dataset
casual_model = randomForest(casual ~., data = df[,1:6], ntree = 10)
registered_model = randomForest(registered ~., data = df[,c(1:5,7)], ntree = 10)
cnt_model = randomForest(cnt ~., data = df[,c(1:5,8)], ntree = 10)

#Using these models the dependent variables are predicted for the sample_input
casual_prediction = predict(casual_model, sample_input[,1:5])
registered_prediction = predict(registered_model, sample_input[,1:5])
cnt_prediction = predict(cnt_model, sample_input[,1:5])

#Output dataframe is created and it is stored in csv file and exported
sample_output = data.frame(sample_input,casual_pd=casual_prediction, +
                           registered_pd=registered_prediction,cnt_pd=cnt_prediction)
write.csv(sample_output,file = "R Sample Output.csv", row.names = F)
```

Fig 3.2 Relationship between dteday and cnt

```
plt.plot(rental_data["dteday"],rental_data["cnt"])
plt.xlabel("Timeline")
plt.ylabel("Count")
plt.show()
```

Fig 3.3 Relationship between workingday and cnt

```
plt.bar(rental_data["workingday"],rental_data["cnt"],width=0.1)
plt.xlabel("Working Day")
plt.ylabel("Average Count")
plt.show()
```

Fig 3.4 Relationship between weathersit and cnt

```
plt.bar(rental_data["weathersit"],rental_data["cnt"], width=0.1)
plt.xlabel("Weather Situation")
plt.ylabel("Average Count")
plt.show()
```

Fig 3.5 Relationship between (temp/hum/windspeed) and cnt

```
for i in ["temp","hum","windspeed"]:
    plt.scatter(rental_data[i],rental_data["cnt"])
    plt.xlabel(i)
    plt.ylabel("Count")
    plt.show()
```

Complete Python Code

Python Code for Bike-Rental Project

The following libraries are imported to be used in this project.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from fancyimpute import KNN
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import pydotplus as pdp
from sklearn.tree import export_graphviz
from IPython.display import Image
from sklearn.externals.six import StringIO
import statsmodels.api as sm
```

To set the working directory, the following statement is used.

```
os.chdir("C:\\Users\\admin\\Desktop\\Aravi\\Data Science\\Important Notes\\EdWisor\\Assignments\\Project 1 - Bike Rental")
```

Using the following statement, the data can be loaded from the working directory.

```
rental_data = pd.read_csv("Day.csv")
```

```
#To check the dimensions of the data
rental_data.shape
```

Missing Value Analysis

```
#Selected all the variables except instant and cnt and checked for the number of missing values in each variable
missing_val = pd.DataFrame(rental_data.iloc[:,1:15].isnull().sum())
missing_val.columns = ["Missing Values"]
```

```
missing_val
```

As there are no missing values, we can move on to the next section.

Outlier Analysis

```
#Creating Histogram to check the data distribution
for i in range(9,15):
    plt.hist(rental_data.iloc[:,i])
    plt.xlabel(rental_data.columns[i])
    plt.ylabel('Frequency')
    plt.show()
```

```
#Creating Box and Whisker plots to check for outliers
for i in range(9,15):
    plt.boxplot(rental_data.iloc[:,i])
    plt.xlabel(rental_data.columns[i])
    plt.show()
```

Now. we will remove the outliers from 'hum' , 'windspeed' and 'casual' variables and replace them with imputed data using knn imputation method.

```
#Handling Outliers using KNN Imputation method

#Taking copy of data and initiating the variables to handle

df=rental_data.copy()
cnames = ["hum","windspeed","casual"]

#Detect outliers and replace with NA

for i in cnames:
    q75, q25 = np.percentile(df.loc[:,i], [75 ,25])
    iqr = q75 - q25
    low = q25 - (iqr*1.5)
    high = q75 + (iqr*1.5)
    df.loc[df[i] < low,i] = np.nan
    df.loc[df[i] > high,i] = np.nan


missing_val = pd.DataFrame(df.loc[:,cnames].isnull().sum())
missing_val.columns = ["Missing Values"]
missing_val    #displays number of outliers in the three variables
```

```
#Imputing the outliers with KNN
df = pd.DataFrame(KNN(k = 3).fit_transform(df.iloc[:,2:15]), columns = df.iloc[:,2:15].columns)
```

```
#Checking the missing values after KNN imputation
missing_val = pd.DataFrame(df.loc[:,cnames].isnull().sum())
missing_val.columns = ["Missing Values"]
missing_val    #displays number of outliers in the three variables
```

```
#Creating Box and Whisker plots to check for outliers after outliers are removed
for i in cnames:
    plt.boxplot(df.loc[:,i])
    plt.xlabel(i)
    plt.show()
```

Here, we can find that, the outliers in 'hum' had been removed and outliers in 'windspeed' and 'casual' have been considerably reduced and brought close to the whiskers of the plot. Hence these outliers are not again treated and they are considered as part of data as they wont bias the model. Now, the variable 'cnt' is also changed based on values of 'casual' and 'registered'.

```
df['cnt']=df['casual']+df['registered']
rental_data = pd.concat([rental_data.iloc[:,0:2].reset_index(drop=True),df],axis = 1) #Rental data dataset is again formed
```

Now, outliers have been efficiently handled and we can move on to the next preprocessing technique

Feature Selection

Correlation analysis for forming the heatmap based on correlation matrix

```
#Formation of Correlation Matrix

cor = rental_data.iloc[:,2:].corr()
cor
```

```
#Fixing the figure size and generating heatmap

f, ax = plt.subplots(figsize=(14, 12))
sns.heatmap(cor,square=True,annot=True,cmap=sns.diverging_palette(1000, 10, as_cmap=True), ax=ax)
```

From the heatmap, we can come to the conclusion that the variables 'holiday', 'weekday', 'hum', 'windspeed', 'mnth' and 'atemp' can be removed as part of feature selection to reduce overfitting and remove the less important variables.

```
rental_data = rental_data.drop(['instant','dteday','holiday', 'weekday', 'hum', 'windspeed', 'mnth','atemp'],axis=1)
```

Feature Scaling

```
#Dependent Variables casual, registered and cnt are also normalised as it will be esier for evaluation of model
#They are normalised as all data are not ubiformly distributed
norm_var = ["casual","registered","cnt"]
for i in norm_var:
    plt.hist(rental_data[i])
    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.show()
    rental_data[i] = (rental_data[i] - min(rental_data[i]))/(max(rental_data[i]) - min(rental_data[i]))
print(rental_data.head(5))
```

Now the data is completely Normalised and the basic pre processing is completed and the dataset is ready for the Modelling phase.

Modeling

Error Metric

In this project, we are going to use MAPE and RMSE as the error metrics to measure the performance of the model

```
#Defining MAPE and RMSE
def MAPE(actual, predicted):
    result = 100*np.mean(np.abs((actual-predicted)/actual))
    return result
```

```
def RMSE(actual, predicted):
    result = (np.mean((actual-predicted)**2))**0.5
    return result
```

Decision Tree

The first model is the decision tree

```
#Sampling the data for training and testing purposes
train_data, test_data = train_test_split(rental_data, test_size=0.2)
```

```
#Model generaion based on training data
casual_model = DecisionTreeRegressor(max_depth = 5).fit(train_data.iloc[:,0:5],train_data.iloc[:,5])
registered_model = DecisionTreeRegressor(max_depth = 5).fit(train_data.iloc[:,0:5],train_data.iloc[:,6])
cnt_model = DecisionTreeRegressor(max_depth = 5).fit(train_data.iloc[:,0:5],train_data.iloc[:,7])
```

```
#Getting the model predict the target variable based on test data
casual_predictions = casual_model.predict(test_data.iloc[:,0:5])
registered_predictions = registered_model.predict(test_data.iloc[:,0:5])
cnt_predictions = cnt_model.predict(test_data.iloc[:,0:5])
```

```
#Evaluating the model
print("MAPE")
print("Casual - " + str(MAPE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(MAPE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(MAPE(test_data.iloc[:,7],cnt_predictions)))
print("RMSE")
print("Casual - " + str(RMSE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(RMSE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(RMSE(test_data.iloc[:,7],cnt_predictions)))
```

```
#To visualise a decision tree
dot_data = StringIO()
export_graphviz(casual_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pdp.graph_from_dot_data(dot_data.getvalue())
graph.write_png("DT.png") #The decision tree is exported to the directory as png file
Image(graph.create_png())
```

Random Forest

The next model is the Random Forest.

```
#We can use the same train and test data that we used for decision trees
#Model generation
casual_model = RandomForestRegressor(n_estimators = 10).fit(train_data.iloc[:,0:5],train_data.iloc[:,5])
registered_model = RandomForestRegressor(n_estimators = 10).fit(train_data.iloc[:,0:5],train_data.iloc[:,6])
cnt_model = RandomForestRegressor(n_estimators = 10).fit(train_data.iloc[:,0:5],train_data.iloc[:,7])
```

```
#Getting the model predict the target variable based on test data
casual_predictions = casual_model.predict(test_data.iloc[:,0:5])
registered_predictions = registered_model.predict(test_data.iloc[:,0:5])
cnt_predictions = cnt_model.predict(test_data.iloc[:,0:5])
```

```
#Evaluating the model
print("MAPE")
print("Casual - " + str(MAPE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(MAPE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(MAPE(test_data.iloc[:,7],cnt_predictions)))
print("RMSE")
print("Casual - " + str(RMSE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(RMSE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(RMSE(test_data.iloc[:,7],cnt_predictions)))
```

Linear Regression

The next model is the Linear Regression

```
casual_model = sm.OLS(train_data.iloc[:,5],train_data.iloc[:,0:5]).fit()
registered_model = sm.OLS(train_data.iloc[:,6],train_data.iloc[:,0:5]).fit()
cnt_model = sm.OLS(train_data.iloc[:,7],train_data.iloc[:,0:5]).fit()
```

```
casual_predictions = casual_model.predict(test_data.iloc[:,0:5])
registered_predictions = registered_model.predict(test_data.iloc[:,0:5])
cnt_predictions = cnt_model.predict(test_data.iloc[:,0:5])
```

```
#Evaluating the model
print("MAPE")
print("Casual - " + str(MAPE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(MAPE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(MAPE(test_data.iloc[:,7],cnt_predictions)))
print("RMSE")
print("Casual - " + str(RMSE(test_data.iloc[:,5],casual_predictions)))
print("Registered - " + str(RMSE(test_data.iloc[:,6],registered_predictions)))
print("Cnt - " + str(RMSE(test_data.iloc[:,7],cnt_predictions)))
```

```
#Model Summary of cnt model
cnt_model.summary()
```

```
#Scatter plot to visualise cnt variable predictions
plt.scatter(test_data.iloc[:,7],cnt_predictions)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

Based on the error metrics of these three models, Random Forest is the best model for this data. Hence we will use it to predict output.

Output

Using Random Forest model, let us give in a sample input to the model to get the sample predicted output.

```
#Sample input of 5 rows is fetched from the dataset
sample_input = rental_data.iloc[[99,199,299,399,499],:]

#Random Forest model is built using the complete dataset
casual_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,5])
registered_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,6])
cnt_model = RandomForestRegressor(n_estimators = 10).fit(rental_data.iloc[:,0:5],rental_data.iloc[:,7])

#Using these models the dependent variables are predicted for the sample_input
casual_predictions = casual_model.predict(sample_input.iloc[:,0:5])
registered_predictions = registered_model.predict(sample_input.iloc[:,0:5])
cnt_predictions = cnt_model.predict(sample_input.iloc[:,0:5])
```

```
#Output dataframe is created and it is stored in csv file and exported
sample_output = pd.concat([sample_input.reset_index(drop=True), \
    pd.DataFrame({'casual_pd':casual_predictions,'registered_pd':registered_predictions,'cnt_pd':cnt_predictions})],axis = 1)
sample_output.to_csv("Python Sample Output.csv",index=False)
sample_output
```

Interesting Patterns

Additionally, let us see some interesting patterns for getting certain insights on the data

```
#The original data is fetched once again to check patterns
rental_data = pd.read_csv("Day.csv")
```

```
plt.plot(rental_data["dteday"],rental_data["cnt"])
plt.xlabel("Timeline")
plt.ylabel("Count")
plt.show()
```

The plot between timeline and count shows that during the mid-years i.e., (Apr-Sep) the count of rental bikes increase and in the beginning and end of the years, they decrease.

```
plt.bar(rental_data["workingday"],rental_data["cnt"],width=0.1)
plt.xlabel("Working Day")
plt.ylabel("Average Count")
plt.show()
```

```
Based on this Bar Chart, there is no significant difference in total rental bikes used by customers on workingdays and weekends
```

```
plt.bar(rental_data["weathersit"],rental_data["cnt"], width=0.1)
plt.xlabel("Weather Situation")
plt.ylabel("Average Count")
plt.show()
```

This Barchart shows that during weather situation 3 (Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds), the total rented bikes has become nearly half than the normal days. So during rainy days, the count of bikes rented will be reduced.

```
for i in ["temp","hum","windspeed"]:
    plt.scatter(rental_data[i],rental_data["cnt"])
    plt.xlabel(i)
    plt.ylabel("Count")
    plt.show()
```

The relationship between Temperature and Count shows that during hotter days, the count of rented bikes is higher when compared to colder days. But the other two scatter plots show that the count o rental bikes is not related to 'hum' and 'windspeed' variables as data is randomly scattered in these plots.

Thus we have analysed the data, built a model and predicted the count of rental bikes based on environmental and seasonal settings.

Complete R Code

```
#This statement is used to clear the environment
rm(list=ls(all=T))

#The libraries are loaded
x=c("ggplot2","DMwR","corrgram","rpart","randomForest")
install.packages(x)
lapply(x,require,character.only=T)
rm(x)

#The working directory is set
setwd("C:\\Users\\admin\\Desktop\\Aravi\\Data Science\\Important Notes\\EdWisor\\Assignments\\Project 1 - Bike Rental")

#Now the data is loaded
rental_data = read.csv("Day.csv", header=T, na.strings=c(" ","","NA"))

############################ Missing Value Analysis ################################################

missing_val = data.frame(apply(rental_data[,2:15],2,function(x){sum(is.na(x))}))
colnames(missing_val)="Total Missing Values"
rm(missing_val) #To free the memory allocated

#From the missing value analysis, we have found that there are no missing values in the dataset

################################## Outlier Analysis ################################################

#Creating Histograms to check the data distribution
for (i in 10:15)
{
  assign(colnames(rental_data)[i],ggplot(rental_data,aes_string(x=rental_data[,i]))+
           labs(x=colnames(rental_data)[i])+geom_histogram())
}

gridExtra::grid.arrange(temp, atemp, registered, ncol=3)
gridExtra::grid.arrange(hum, windspeed, casual, ncol=3)

#Creating Box and Whisker plots to check for outliers
for (i in 10:15)
{
  assign(colnames(rental_data)[i],
         ggplot(rental_data,aes_string(x=rental_data[,i],y=colnames(rental_data)[i]))+
           labs(x=colnames(rental_data)[i])+
           stat_boxplot(geom = "errorbar", width = 0.5) +
geom_boxplot(outlier.colour="red",fill = "grey" ,outlier.shape=18, outlier.size=1, notch=F))
}

gridExtra::grid.arrange(temp, atemp, registered, ncol=3)
gridExtra::grid.arrange(hum, windspeed, casual, ncol=3)

#Handling Outliers using KNN Imputation method

#Taking a copy of the data and initiating columns for outlier handling
df = rental_data
cnames = c("hum","windspeed","casual")

#Replace outliers with NA
for(i in cnames){
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(val)) #To visualise th total number of outliers
  df[,i][df[,i] %in% val] = NA
  }

#Impute missing values using KNN
df = knnImputation(df, k = 3)
df["cnt"] = df["casual"]+df["registered"]

#Box and whisker plot to visualise after outlier analysis
for (i in cnames){
  assign(i,ggplot(df,aes_string(x=df[,i],y=i))+
  labs(x=i)+stat_boxplot(geom = "errorbar", width = 0.5) +
  geom_boxplot(outlier.colour="red",fill = "grey" ,outlier.shape=18, outlier.size=1, notch=F))
}

gridExtra::grid.arrange(hum, windspeed, casual, ncol=3)
```

```
#From the box and whisker plots, we can find that the outliers have considerably been reduced
#So the model impact due to bias will be reduced.

################################### Feature Selection ########################################

#Formation of Correlation matrix and heatmap based on the correlation between variables
corrgram(df[,3:16], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

df = subset(df, select = -c(instant,dteday,holiday,weekday,hum,windspeed,atemp,mnth))

#The variables of least importance and the variables which may lead to overfitting are removed and proceeded to the next phase.

################################### Feature Scaling ########################################

#The column temp is already normalised and the columns 'casual', 'registered' and 'temp' are to be normalised
norm_var = c("casual","registered","cnt")
for (i in norm_var){
  hist(df[,i])
  df[,i] = (df[,i] - min(df[,i]))/
    (max(df[,i] - min(df[,i])))
}
print(head(df))

################################### Decision Tree ########################################
#Error Metrics
#MAPE
MAPE = function(act,pred){
  mean(abs(act-pred)/act)*100
}

#RMSE
RMSE = function(act, pred){
  (mean((act-pred)**2))**0.5
}
#The first model that we are using is the decision tree
train_index = sample(1:nrow(df),0.8*nrow(df)) #80% of the whole index is sampled for selecting the train data
#Train and test data are sampled
train_data = df[train_index,]
test_data = df[-train_index,]

#Now the decision tree model is built using rpart
casual_model = rpart(casual ~., data = train_data[,1:6], method = "anova")
registered_model = rpart(registered ~., data = train_data[,c(1:5,7)], method = "anova")
cnt_model = rpart(cnt ~., data = train_data[,c(1:5,8)], method = "anova")

#Using the model, the dependent variable is predicted
casual_prediction = predict(casual_model, test_data[,1:5])
registered_prediction = predict(registered_model, test_data[,1:5])
cnt_prediction = predict(cnt_model, test_data[,1:5])

#Model Evaluation

print("RMSE")
print(paste("casual - ",(RMSE(test_data[,6],casual_prediction))))
print(paste("registered - ",(RMSE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(RMSE(test_data[,8],cnt_prediction))))

print("MAPE")
print(paste("casual - ",(MAPE(test_data[,6],casual_prediction))))
print(paste("registered - ",(MAPE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(MAPE(test_data[,8],cnt_prediction))))

#The outputs obtained are given as comments

#RMSE
#"casual -  0.120012777101718"
#"registered -  0.106531368734166"
#"cnt -  0.0964989527097329"

#MAPE
#"casual -  47.6220988759675"
#"registered -  21.4695856369951"
#"cnt -  18.4708929146792"

#Now we can measure the error metrics for the Random forest in the below section

################################### Random Forest ########################################

#Now the random forest model is built using randomForest library
casual_model = randomForest(casual ~., data = train_data[,1:6], ntree = 10)
registered_model = randomForest(registered ~., data = train_data[,c(1:5,7)], ntree = 10)
cnt_model = randomForest(cnt ~., data = train_data[,c(1:5,8)], ntree = 10)

#Using the model, the dependent variable is predicted
casual_prediction = predict(casual_model, test_data[,1:5])
registered_prediction = predict(registered_model, test_data[,1:5])
cnt_prediction = predict(cnt_model, test_data[,1:5])

#Model Evaluation

print("RMSE")
print(paste("casual - ",(RMSE(test_data[,6],casual_prediction))))
print(paste("registered - ",(RMSE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(RMSE(test_data[,8],cnt_prediction))))
```

```
print("MAPE")
print(paste("casual - ",(MAPE(test_data[,6],casual_prediction))))
print(paste("registered - ",(MAPE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(MAPE(test_data[,8],cnt_prediction))))

#The outputs obtained are given as comments

#RMSE
#"casual -  0.14501263201758"
#"registered -  0.096473865409887"
#"cnt -  0.0885746302777854"

#MAPE
#"casual -  43.5856369951654"
#"registered -  18.2914679285633"
#"cnt -  17.4439937438519"

############################## Linear Regression ########################################

#The same train test data is used for linear regression too
#Model generation
casual_model = lm(casual ~., data = train_data[,1:6])
registered_model = lm(registered ~., data = train_data[,c(1:5,7)])
cnt_model = lm(cnt ~., data = train_data[,c(1:5,8)])

#Using the model, the dependent variable is predicted
casual_prediction = predict(casual_model, test_data[,1:5])
registered_prediction = predict(registered_model, test_data[,1:5])
cnt_prediction = predict(cnt_model, test_data[,1:5])

#Model Evaluation

print("RMSE")
print(paste("casual - ",(RMSE(test_data[,6],casual_prediction))))
print(paste("registered - ",(RMSE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(RMSE(test_data[,8],cnt_prediction))))

print("MAPE")
print(paste("casual - ",(MAPE(test_data[,6],casual_prediction))))
print(paste("registered - ",(MAPE(test_data[,7],registered_prediction))))
print(paste("cnt - ",(MAPE(test_data[,8],cnt_prediction))))


#The outputs obtained are given as comments

#RMSE
#"casual -  0.128242142735652"
#"registered -  0.10249082905932"
#"cnt -  0.107447951947048"

#MAPE
#"casual -  54.2798720007824"
#"registered -  18.8597522591259"
#"cnt -  18.1615320907755"

#To visualise the summary of the model
summary(cnt_model)

#Based on the error metrics, for this dataset, RandomForest model performs better when compared to all the other models

############################### Sample Output ############################################

#Sample input of 5 rows is fetched from the dataset
sample_input = df[c(100,200,300,400,500),]

#Random Forest model is built using the complete dataset
casual_model = randomForest(casual ~., data = df[,1:6], ntree = 10)
registered_model = randomForest(registered ~., data = df[,c(1:5,7)], ntree = 10)
cnt_model = randomForest(cnt ~., data = df[,c(1:5,8)], ntree = 10)

#Using these models the dependent variables are predicted for the sample_input
casual_prediction = predict(casual_model, sample_input[,1:5])
registered_prediction = predict(registered_model, sample_input[,1:5])
cnt_prediction = predict(cnt_model, sample_input[,1:5])

#Output dataframe is created and it is stored in csv file and exported
sample_output = data.frame(sample_input,casual_pd=casual_prediction, +
                           registered_pd=registered_prediction,cnt_pd=cnt_prediction)
write.csv(sample_output,file = "R Sample Output.csv", row.names = F)
```

```
############################### Interesting Patterns ####################################

#Let us initiate the dataframe once again for finding patterns
rental_data = read.csv("Day.csv", header=T, na.strings=c(" ","","NA"))

#Pattern to check how cnt has varied on a daily basis
ggplot(rental_data, aes_string(x=rental_data$dteday,y=rental_data$cnt,group=1))+geom_line()

#Bar plot to check how cnt has varied on workingday and based on weather situation
ggplot(rental_data, aes_string(x=rental_data$workingday,y=rental_data$cnt,group=1))+geom_bar(stat="identity")
ggplot(rental_data, aes_string(x=rental_data$weathersit,y=rental_data$cnt,group=1))+geom_bar(stat="identity")

#Bar plot to check the relationship between temp, hum, windspeed with cnt variable
for (i in c("temp","hum","windspeed")){
  assign(i,ggplot(rental_data, aes_string(x=rental_data[,i],y=rental_data[,"cnt"]))+geom_point())
}

gridExtra::grid.arrange(temp,hum,windspeed,ncol=3)

#From these graphs, we can see that the total count of rental bikes increase mid year and decreases at the beginning and end of the year
#There is no dierence between holiday and working day in terms of count of rental bikes
#Durinng hotter days, more bkes are rented, but no relationship between hum, windspeed and count of rental bikes

#Thus we have trained a model which can predict the cost of rental bikes based on environmental and seasonal settings.
```

References

Matplotlib library attributes – referred from www.matplotlib.org

Ggplot2 library attributes – referred from www.tutorialspoint.com

Pydotplot library and graphviz for visualising decisiontree – referred from blogs in www.medium.com

R programming basic functionalities – referred from rprogramming.net

Small functionalities in R and Python – referred from www.stackoverflow.com

Modeling and evaluation metrics – referred from www.edwisor.com