

Project Report on Bike Renting Prediction

Submitted by

J Dheeraj Kumar



(7 – 21)st January 2019

Submitted to

edWisor

Table of Contents

Chapter 1

1 Introduction

1.1 Given Problem Description	1
1.2 Addressing The Problem Statement	1
1.3 Problem Level Approach.....	1
1.4 Understanding The Data	2

Chapter 2

2 Methodology

2.1 Data Pre Processing	4
2.1.1 Missing Value Analysis	6
2.1.2 Analyzing Data Through Visualization	7
2.1.3 Outlier Analysis	13
2.1.4 Feature Selection	15
2.1.4.1 Correlation Analysis.....	15
2.1.4.2 Chi-2 Square Test of Independence	18
2.1.4.3 ANOVA Test	19

2.1.4.4 Multi Collinearity Check	23
2.1.5 Dimension Reduction	24
2.1.6 Sampling of Data.	26
2.2 Model Development	27
2.2.1 Developing Regression Models.	28
2.2.2 Model Selecting and Evaluation.	45
2.2.3 Hyper Parameter Tuning and Optimization.	46
2.2.4 Final Model Selection.	53
 Chapter 3	
3 Conclusion	54
3.1 End Notes.	54
Appendix A - Extra Figures	55
Appendix B – Complete Python Code	62
Appendix C – Complete R Code.	74

CHAPTER 1

Introduction

Doing biking is really super fun which also helps us to be healthy all the time and which also reflects the environment in the cities and country's not being polluted so many countries have taken into account that we need to gain more bike renting kind of systems for the public so this system helps us to know how the people or the customers are using the bikes for their daily needs which also helps us to get the insights that whether the user is registered or a casual user. Which also tells how often the user uses this system in different environments and seasons.

1.1 Given Problem Description

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

1.2 Addressing The Problem Statement

The given problem description is kind of forecasting / prediction of a continuous number which is nothing but "Predictions" are made for new data when the actual outcome may not be known until some future date. The future is being predicted, but all prior observations are almost always treated equally. The objective here is to predict the bike rental count on daily based on basis of environmental and seasonal settings which states that it is a time-series kind of problem that is nothing but we are having a target label with continuous number that we need to predict with the features / predictors that is the independent variables

1.3 Problem Statement Approach

The objective of the problem statement falls under the Regression models and here we need to predict the bike rental count on daily basis that how many users / customers are going to rent the bike on daily basis or not so here we are expected to develop a predictive model by using the bike rental data in both Python and R programming languages.

1.4 Understanding The Data

The details of data attributes in the dataset are as follows

Table 1.1 – Description of Data

Features	Description
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 from Holiday Schedule)
weekday	Day of the week
workingday	If day is neither weekend nor holiday is 1, otherwise is 0.
weathersit	(extracted from Freemeteo) 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
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)
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including both casual and registered

The dimension of the dataset given has 731 observations and 16 Columns.

Let's check some of the observations of the dataset that is first five rows of the dataset.

Table: 1.2 – All Columns with top 5 observations of the dataset

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

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160466	331	654	958
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.200000	0.212122	0.590435	0.160296	108	1454	1562
0.266957	0.229270	0.436957	0.186900	82	1518	1600

Numeric columns: ['temp','atemp','humidity','windspeed','casual','registered', 'cnt']

Categorical columns: ['instant','season','year','month','holiday','weekday','workingday','weather']

Here the Target Variable is **cnt = casual + registered**

So here we need to predict the target variable 'cnt' that is whether the user he / she is going to rent the bike or not.

CHAPTER 2

Methodology

The methodology followed for the approach of developing the algorithm is as follows.

Exploratory Data Analysis (EDA): - Exploring the data such as the information and the datatypes of the columns checking for the unique values in the data checking if there are any missing values in the data or Checking the distribution of the data, and performing univariate, bivariate and multivariate analysis.

Data Pre-Processing: - Checking for outliers in the data doing feature selection using correlation analysis to check the dependency between the numeric columns and checking dependency between categorical columns with Chi-2 Square test of independence method and checking the dependency between numerical columns VS categorical columns with ANOVA test. Checking the important features and Checking if Multicollinearity exists between the features with the help of (VIF) Variance Inflation factor performing dimension reduction to remove features which are highly correlated and sampling of the data.

Model Development: - As the problem category falls under the regression models so all the regression algorithms like Linear Regression, Decision Tree Regressor, Random Forest Regressor, KNN Regressor, SVM Regressor and XGB Regressor models have been developed.

Optimization and Final Model Selection: - Hyper-parameter tuning and optimization of the outperformed model has been selected and finalized here.

2.1 Data Pre- Processing Exploratory Data Analysis (EDA)

As we know the data is the back-bone for developing the models if the quality of our data is good then the result outcome from the models will also be good else the results won't be accurate because in the real world data format may be structured, semi-structured or even unstructured the data may also contain noise, redundant columns such as same columns having the same information, so we need to pre-process the raw data and bring it into proper form for developing the models which accept mostly only the numeric values so we need to decode the datatypes, missing values, outliers, multicollinearity between the features and etc.

Information about the data and its data-types

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant          731 non-null int64
datetime         731 non-null object
season           731 non-null int64
year             731 non-null int64
month            731 non-null int64
holiday          731 non-null int64
weekday          731 non-null int64
workingday       731 non-null int64
weather          731 non-null int64
temp             731 non-null float64
atemp            731 non-null float64
humidity         731 non-null float64
windspeed        731 non-null float64
casual           731 non-null int64
registered       731 non-null int64
count            731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

understanding the data: datetime is the time base feature, season has four different (1: springer, 2: summer, 3: fall, 4: winter) which is a categorical feature, year we have 0:2011 & 1:2012 which can also be consider as categorical feature, month has 12 months which is also a time base feature and can be a categorical feature, holiday: weather day is holiday or not (extracted from Holiday Schedule) 0 for non-holiday / non-working day and 1 is the opposite so it can be considered as categorical, weekday Day of the week time variable so categorical feature itself,

weather has four unique values with different weather conditions so we consider as categorical feature, temp, atemp, humidity, and windspeed are continuous variables whereas registered, casual and cnt are the three target variables where registered + casual = cnt so it clearly describes it is a regression problem.

Let's See the Descriptive Statistics of the Dataset

Table 2.1: Descriptive Statistics of numeric columns from (9-15)

	temp	atemp	hum	windspeed	casual	registered	cnt
count	731	731	731	731	731	731	731
mean	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	0.183051	0.162961	0.142249	0.077498	686.622488	1560.256377	1937.211452
min	0.059130	0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
25 %	0.337083	0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
50 %	0.498333	0.486733	0.626667	0.180957	713.000000	3662.000000	4548.000000
75 %	0.655417	0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
max	0.861667	0.840896	0.972500	0.507463	341.000000	6946.000000	8714.000000

Here from the numeric feature columns we can clearly observe that most of the data has normalized values from a scale of 0-1 and the target variable cnt ranges from min of 22 to a max of 8714.

Let's see the total unique values in all the categorical variables now

```

season
3      188
2      184
1      181
4      178
Name: season, dtype: int64
*****
year
1      366
0      365
Name: year, dtype: int64
*****
month
12      62
10      62
8        62
7        62
5        62
3        62
1        62
11       60
9        60
6        60
4        60
2        57

```



```

Name: month, dtype: int64
*****
holiday
0      710
1       21
Name: holiday, dtype: int64
*****
weekday
6      105
1      105
0      105
5      104
4      104
3      104
2      104
Name: weekday, dtype: int64
*****
workingday
1      500
0      231
Name: workingday, dtype: int64
*****
weather
1      463
2      247
3       21
Name: weather, dtype: int64
*****

```

Total unique values in categorical columns

```

season =====> 4
year =====> 2
month =====> 12
holiday =====> 2
weekday =====> 7
workingday =====> 2
weather =====> 3

```

2.1.1 Missing Values Analysis

The steps involved in data pre-processing is firstly we explore the data and its data-types and its basic information, checking the descriptive statistics of the dataset then next we do the missing values check in the dataset. Missing values in the data may occur due to numerous ways and the steps to be taken towards it is either we can ignore or impute missing values in the dataset.

We impute the variables missing values whose data is less than (< 30%) only or else drop the variables whose percent is greater than 30 %, if we try to impute the data which has greater than thirty percent may lead us to **‘false predictions’** in the model development. The best way to understand each value is missing is by plotting a bar graph. The methods involved in imputing missing values are {‘Fill in with Central Statistics’: (Mean, Median, Mode), ‘Distance based or Data

Mining Method': (KNN Imputation Method), 'Prediction Method'}. We are lucky here because our dataset does contain any missing values and we don't need to apply these steps. This approach is same in both Python and R.

```
instant      0
datetime     0
season       0
year         0
month        0
holiday      0
weekday      0
workingday   0
weather      0
temp         0
atemp        0
humidity     0
windspeed    0
casual       0
registered   0
count        0
dtype: int64
No missing values are there in the dataset
```

2.1.2 Analyzing Data through Visualization (Uni/Bi/Multivariate analysis)

Let's see the distribution of the target variable cnt in R.

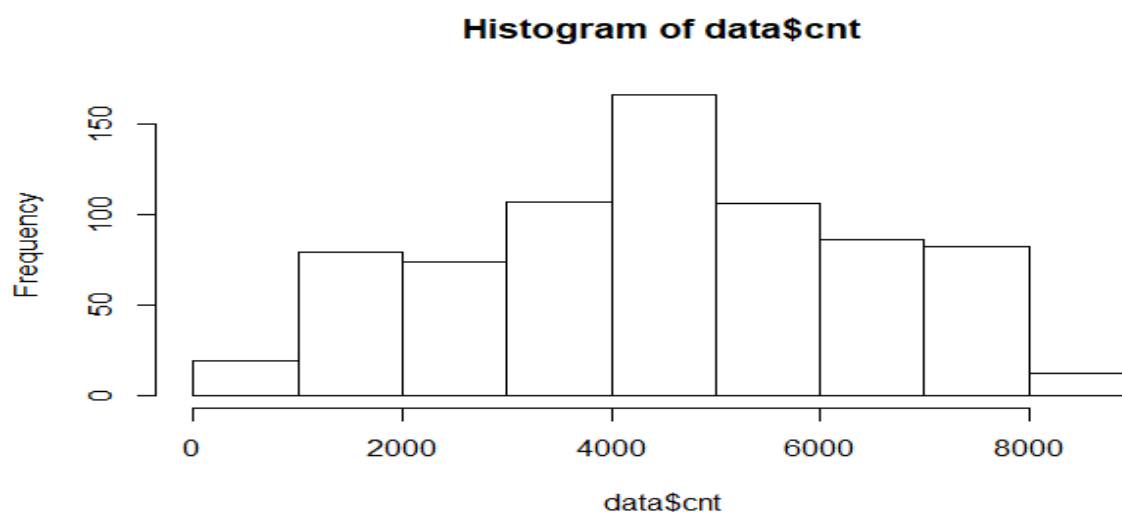


Fig 1.1 – Target variable 'cnt' is normally distributed in R

Let's see the distribution of the target variable 'cnt' in Python.

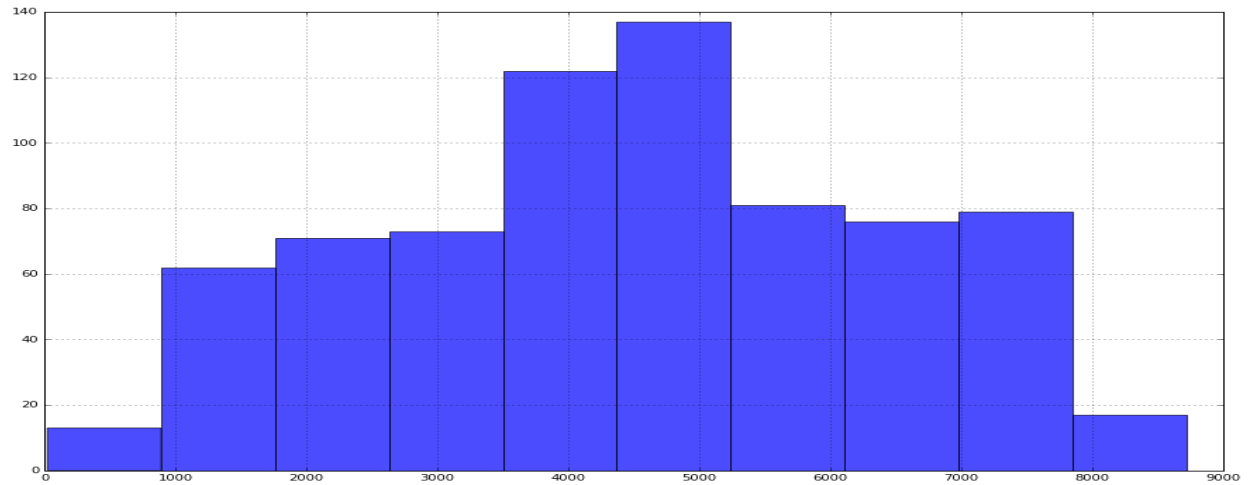


Fig 1.2 – Target variable 'cnt' is uniformly distributed in python

Let's see the distribution of the target variable 'casual' in Python.

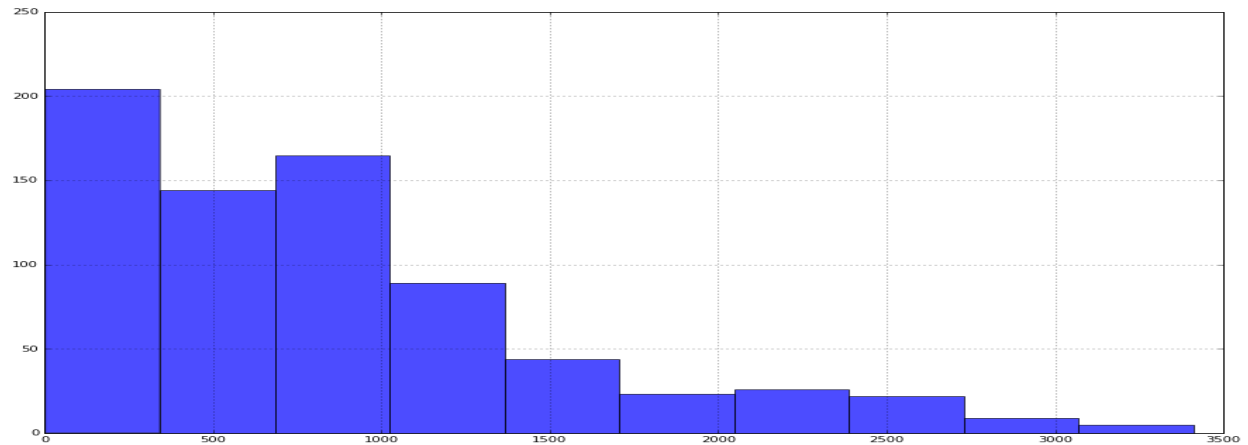


Fig 1.3 – Target variable 'casual' is Left Skewed in python

There might be some outliers in the feature casual that's why it is left skewed here.

Let's see the distribution of the target variable 'Registered' in Python.

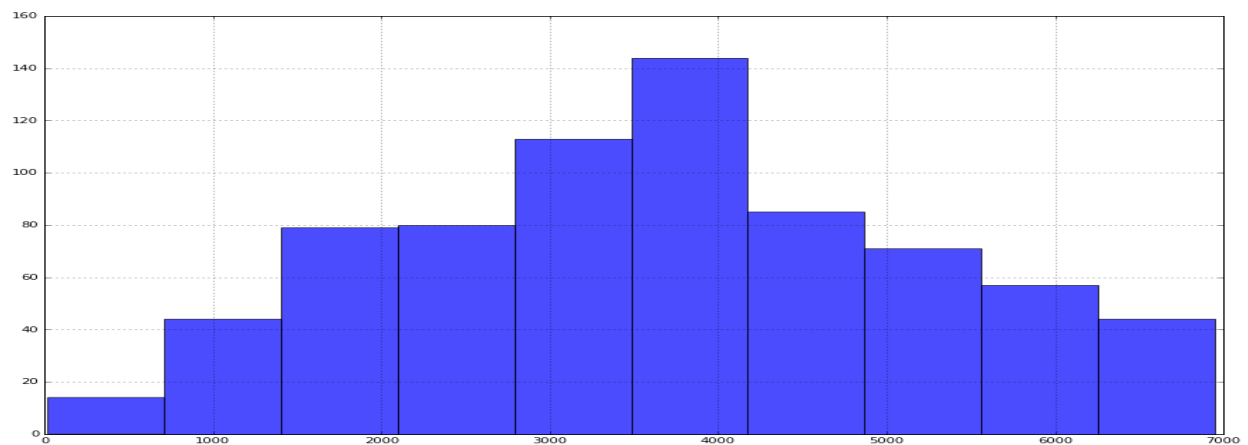


Fig 1.4 – Target variable 'registered' is uniformly distributed in python

Let's see the distribution of the various count in seasons

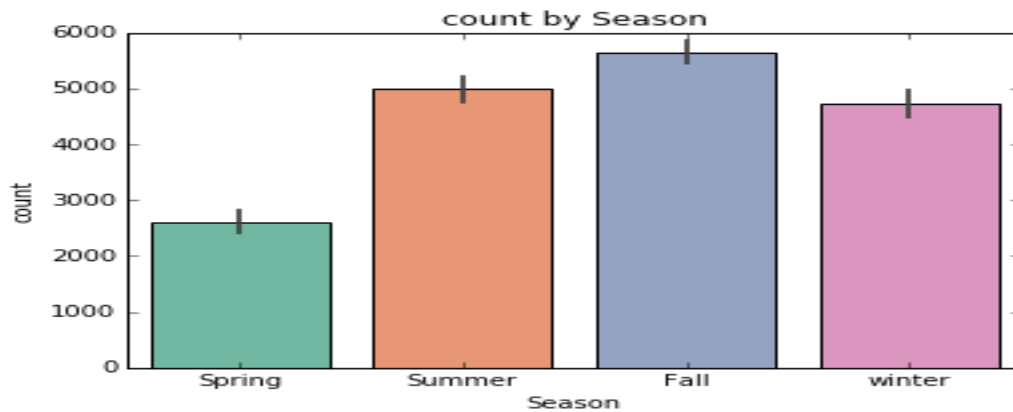


Fig 1.5 –distribution of various season where fall looks high here in python

Let's see the distribution of the count weather with particular seasons

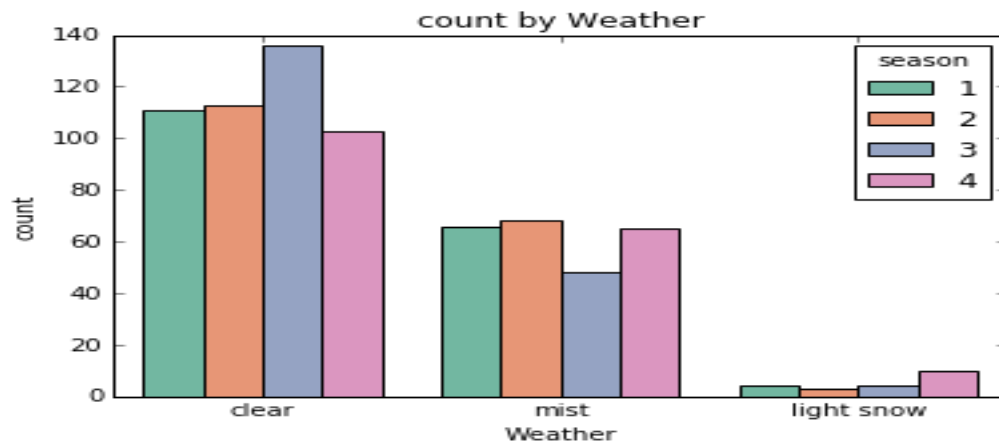


Fig 1.6 –distribution of weather with particular seasons here in python

Let's see the distribution of the count season in different years, there is no much difference

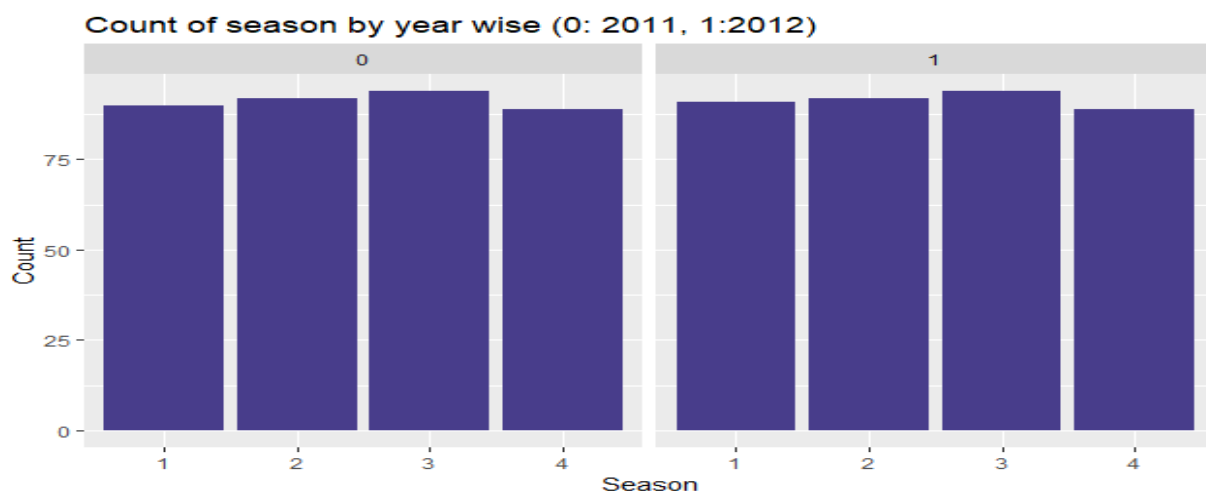


Fig 1.7 –distribution of season in different years in R

Let's see the distribution of the count holidays

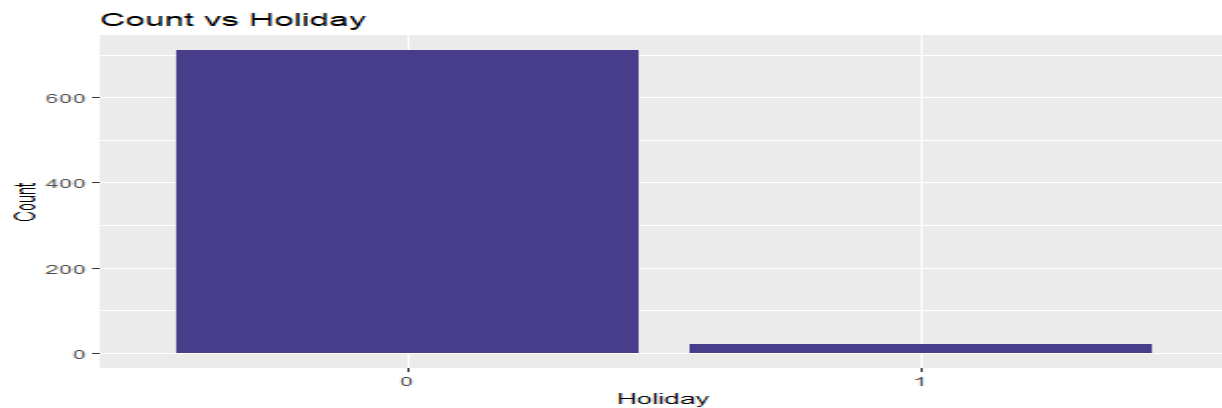


Fig 1.8 –distribution of Holiday in R

Weather day is working day or non-working day there is a target class imbalance in this variable and most of days here are non-holiday.

Let's see the count of working days in different seasons and both the years 2011 & 2012

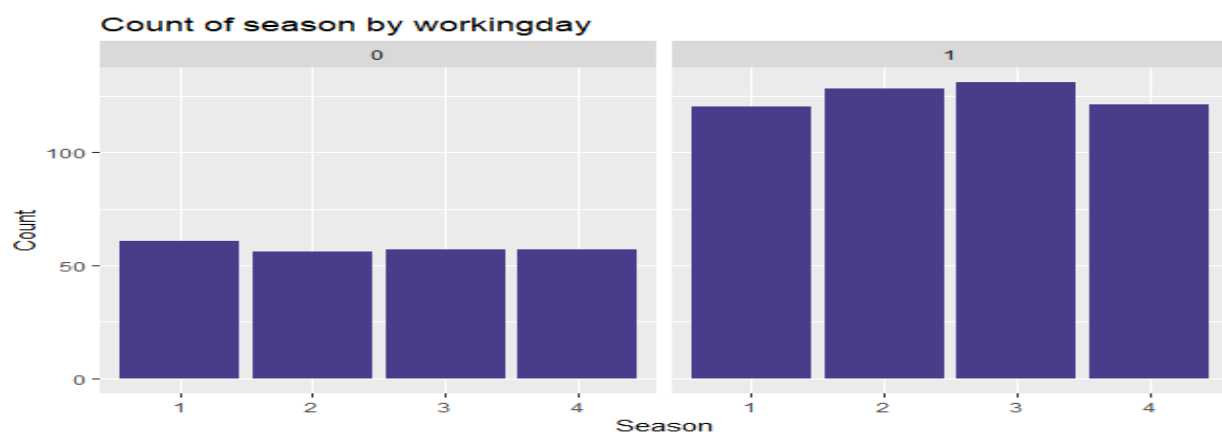


Fig 1.9 –distribution of working days in different years with respect to season in R

Let's see the distribution of working days in different months

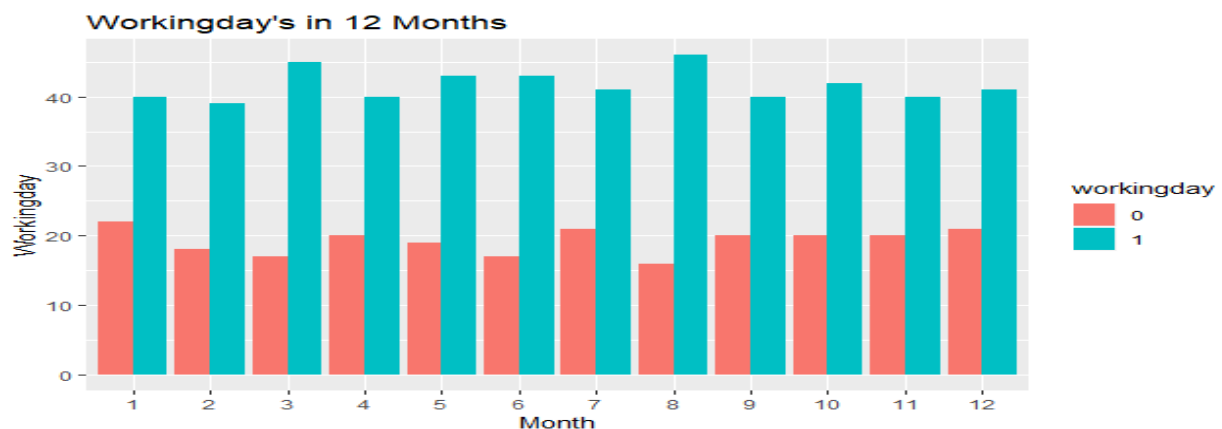


Fig 2 – If day is neither weekend nor holiday is 1, otherwise is 0 in R

Let's see the distribution of weekdays in different weather conditions

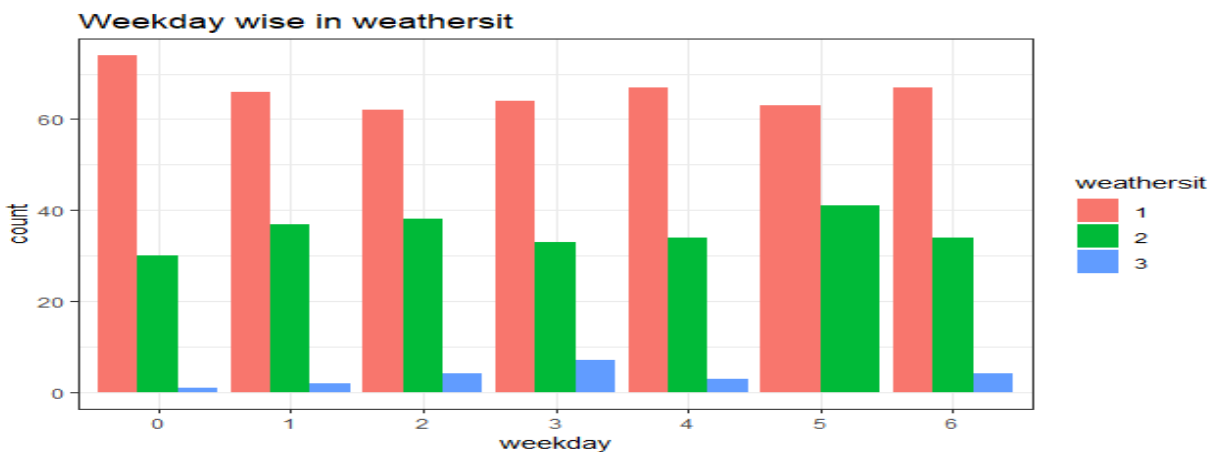


Fig 2.1 – distribution of weekday in different weather conditions in R

Let's see the count of working days in different weather conditions

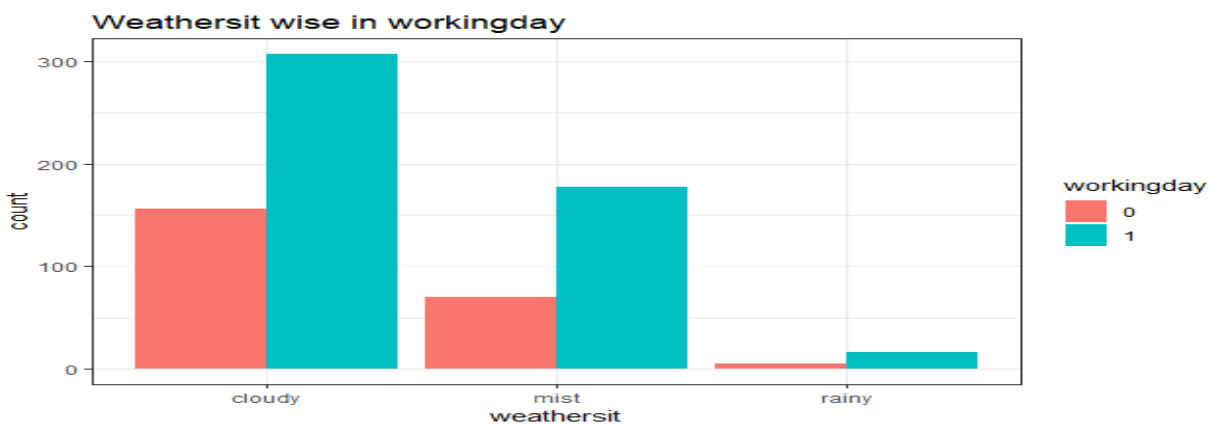


Fig 2.2 – distribution of working days in different weather conditions in R

Looks like most of the working days are when the weather condition is cloudy

Let's see the distribution of weekdays in different months

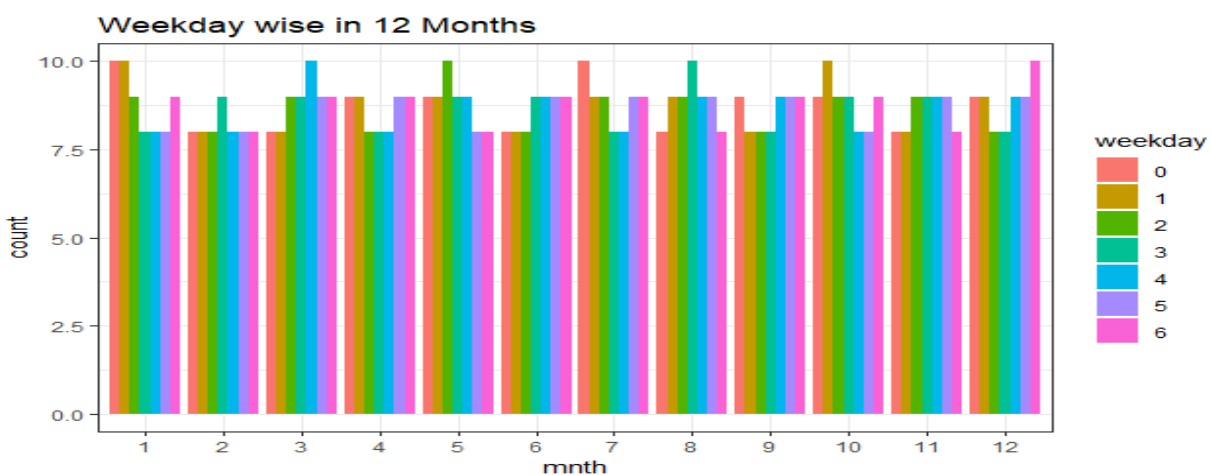


Fig 2.3 – distribution of week days in different months in R

Let's see the distribution count of all months

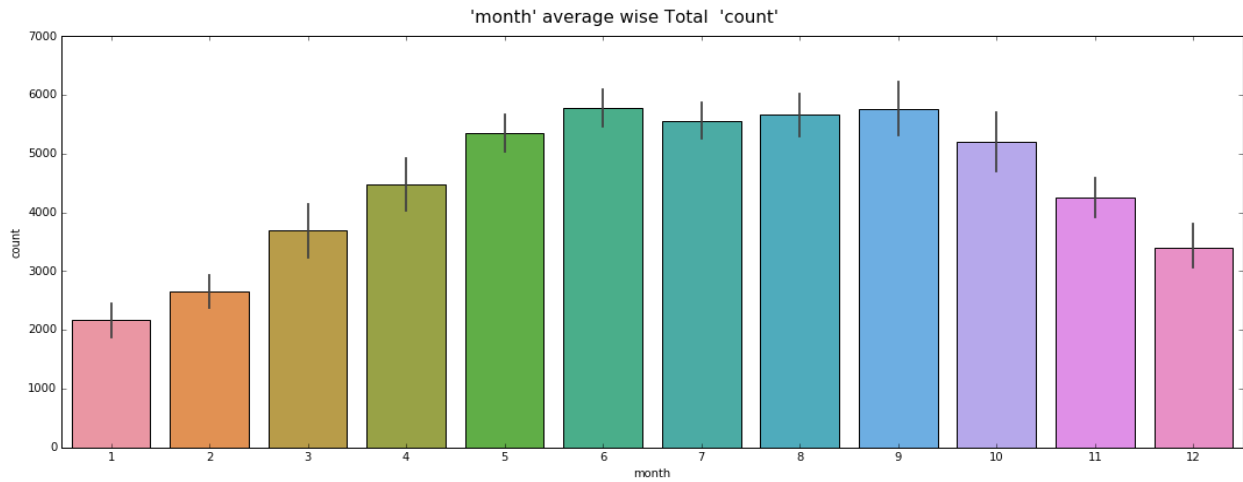


Fig 2.4 – distribution of all months in Python

Let's see the relationship between variables with joint plots

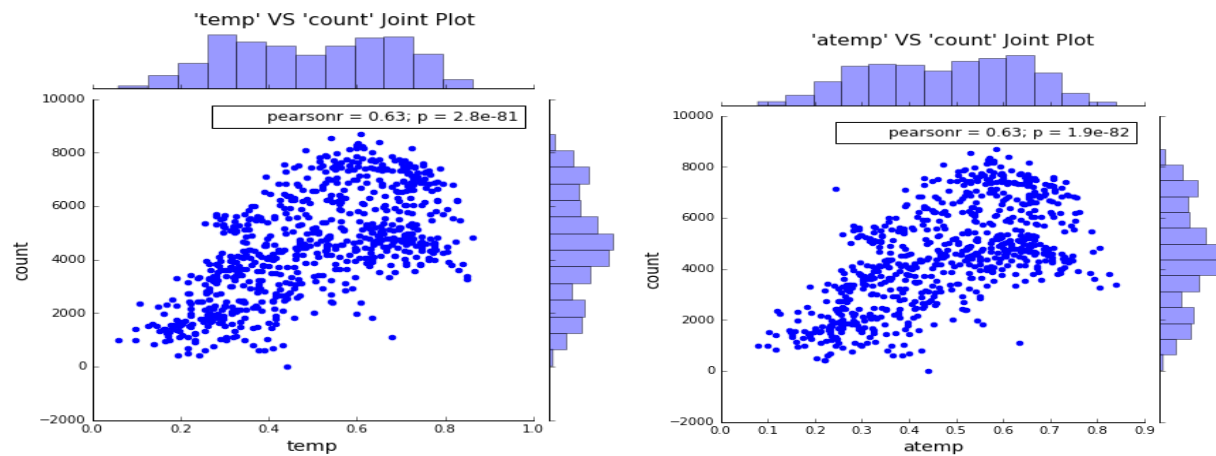


Fig 2.5 – Relationship of temp and atemp both look similar with cnt target variable in Python

Let's see the relationship of humidity and windspeed VS cnt variable

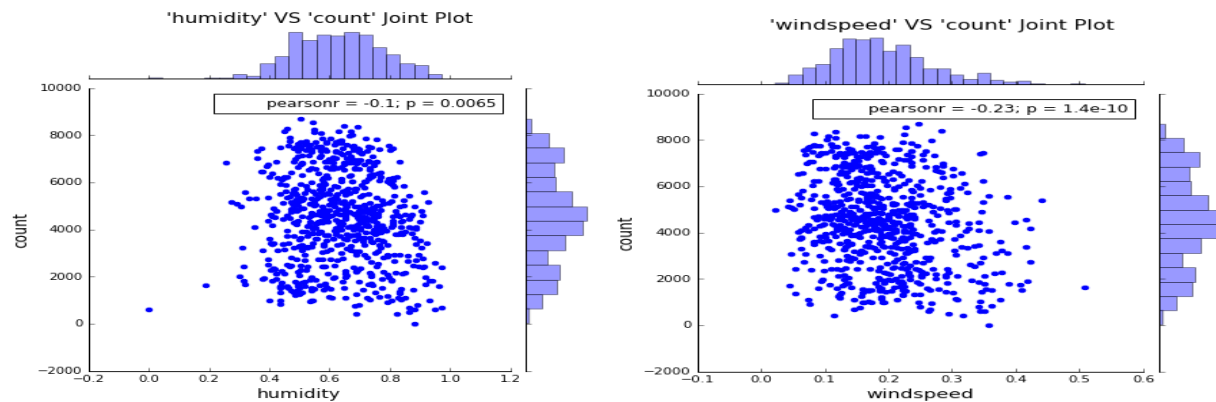


Fig 2.6 – Relationship of humidity and windspeed VS cnt target variable in Python

Now let's check the relationship between all three target variables that is $\text{cnt} = \text{casual} + \text{registered}$

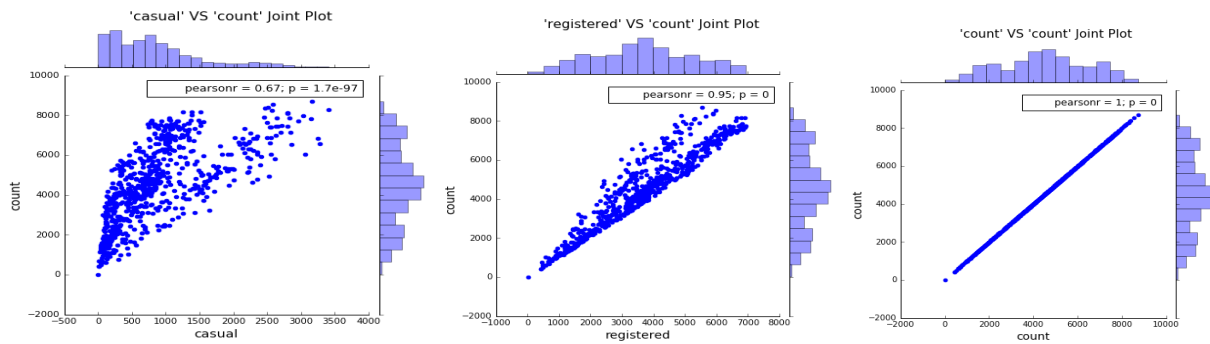


Fig 2.7 – Relationship of casual, registered and cnt target variable is completely linear in Python

Now let's check the relationship between $\text{temp} \sim \text{atemp} + \text{humidity} + \text{windspeed}$

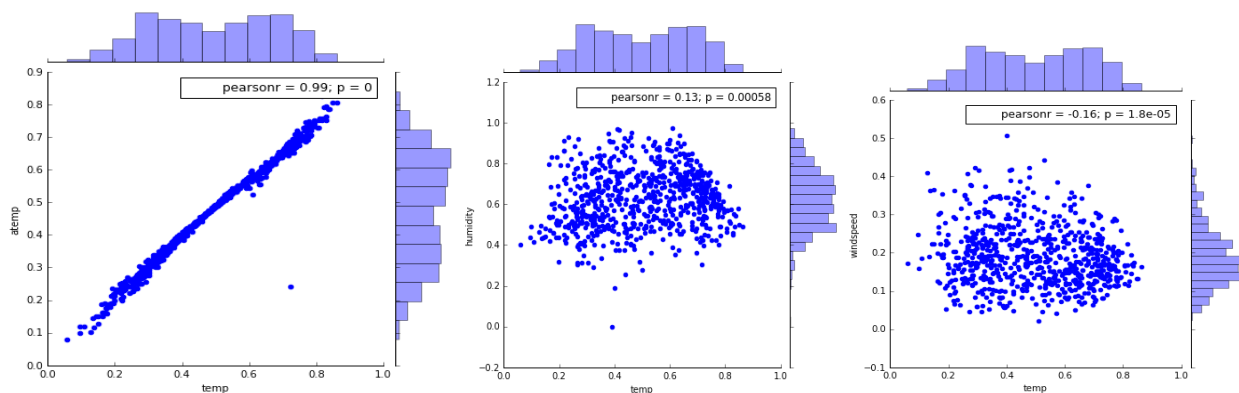


Fig 2.8 – Relationship of temp and atemp is completely linear, temp with humidity and windspeed in Python

2.1.3 Outlier Analysis

Using Boxplot Analysis for Outliers check

An outlier is nothing but which is **“inconsistent”** with the rest of the dataset. Observations inconsistent with the rest of the global outlier, in simple terms any value which is falling away from a bunch of values is nothing but an outlier. Outlier will be available only in continuous variable. This approach is same in both Python and R.

Causes of Outliers could be: -

- Poor data quality / contamination
- Low quality measurements and malfunctioning of equipment, manual errors
- Correct but exceptional data

R has Package Outlier which helps to detect outliers and it uses mean concept and then Replace with NA That is detect with boxplot method and replacing the values with NA's that is missing values and for imputation we can use any method such as very common and easy method to impute data with fill in central statistics mean, median and mode, distance based or data mining method KNN Imputation.

- Let's see the Boxplot visualization of outlier's detection which shows in our dataset.

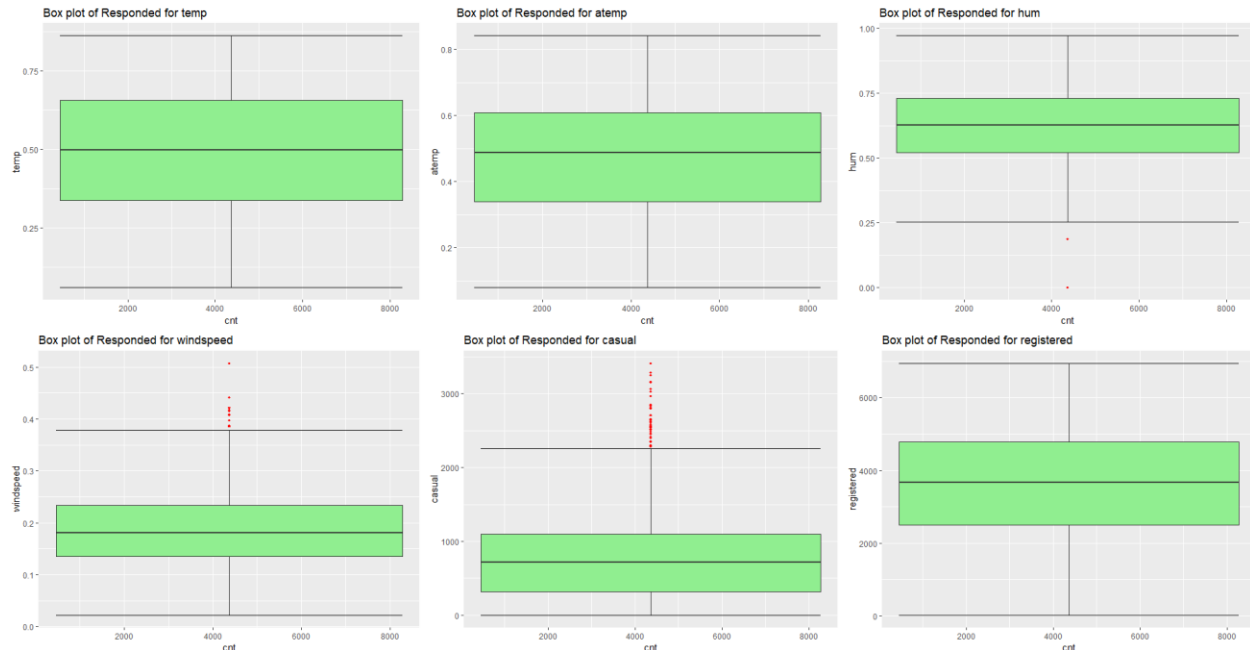


Fig 2.9 – Boxplot for outlier detection check

As we can clearly see that most of the outliers are casual variable then in windspeed and after that in humidity. So here we are going to drop outliers only from humidity and windspeed and not from casual because we may lose important information of the casual users who rent the bike and our predictions may be biased or false if we drop outliers from casual variable.

After applying boxplot method for outlier detection and replacing with NA's

```
temp=====>: 0 Missing_Values
atemp=====>: 0 Missing_Values
humidity=====>: 2 Missing_Values
windspeed=====>: 13 Missing_Values
```

total missing values after dropping outliers were 15 and this was same in both python and R here we are using fill in with central statistics method as there are very few missing values so using mean method to impute the missing values.

2.1.4 Feature Selection

2.1.4.1 Correlation Analysis

Extracting a relevant and meaningful features out of a dataset also known as “**feature engineering or feature extraction**” feature selection means you are selecting a subset of features out of all variables present in the dataset.

So the agenda of feature selection is to identify and to remove irrelevant features from the dataset which do not contribute much information to explain the variance of the target variable.

Because the fewer the features are desirable it reduces the complexity of the model and simpler model is simpler to understand and explain. This approach is same in both Python and R.

Formula for correlation: -

$r = \text{covariance}(X, Y) / \text{S.D}(X) * \text{S.D}(Y)$ here S.D stands for standard Deviation

Formula for Covariance: -

$\text{Cov}(X, Y) = \sum (X_i - \bar{x})(Y_i - \bar{y}) / n$

- Let's see the Correlation matrix of the numeric columns in Python.

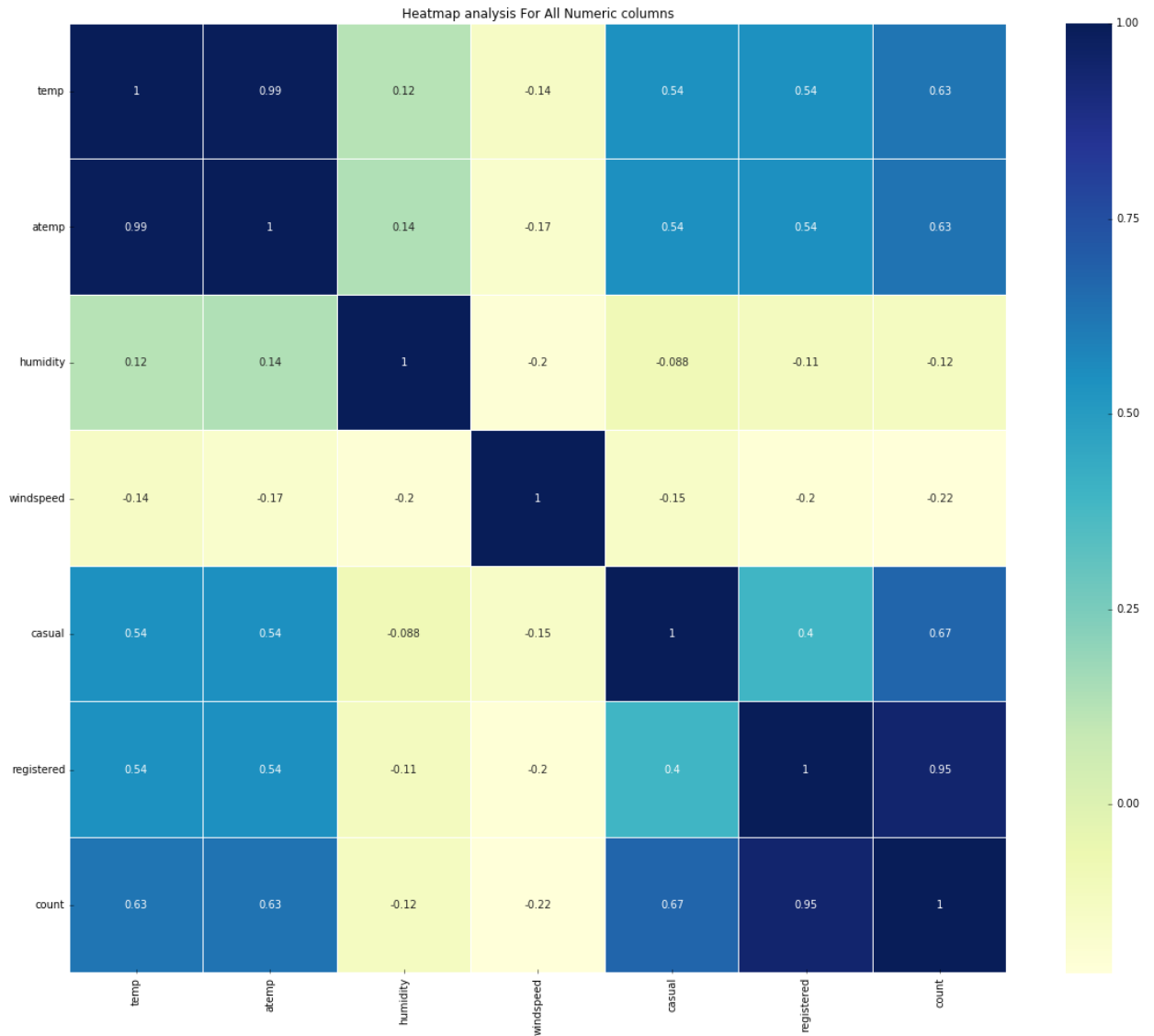


Fig 3 – Correlation analysis to check highly positively correlated features in Python

- Let's see the Correlation matrix of the numeric columns in R.

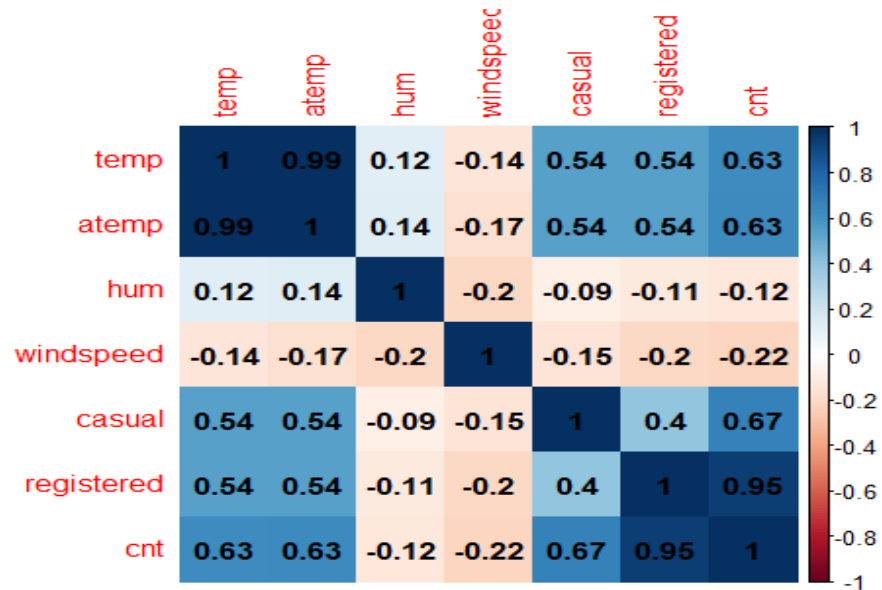


Fig 3.1 – Correlation analysis to check highly positively correlated features in R

As we can clearly observe that temp and atemp features are highly positively correlated with each other and casual and registered are also highly positively correlated with each other and registered and cnt are highly positively correlated with each other so we need to drop these features respectively in both Python and R.

- Let's See the PairPlot on numerical columns

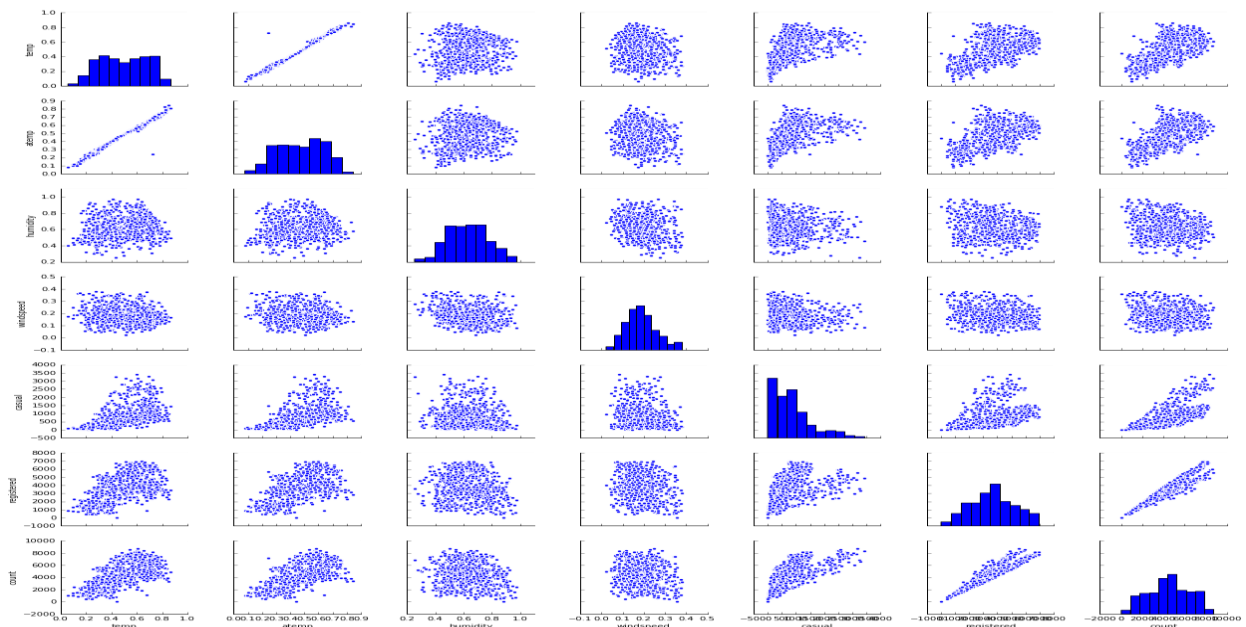


Fig 3.2 – Pair plot gives a clear picture of all numerical features in Python

2.1.4.2 Chi-2 Test of Independence

If we have two categorical features in that case chi-2 square test is the best suitable statistical test to measure the dependency between two categorical features.

Here the dependency between the independent and dependent features should be high and if you are measuring the dependency between the independent features then it should be very low. This approach is same in both Python and R.

Before running chi-2 square test of independence we set the “**Hypothesis**” that is

Null Hypothesis (H₀): - Two features are independent with each other.

Alternate Hypothesis (H₁): - Two features are not independent with each other.

Which means that there is some dependency between the two features.

Using the chi-2 square **p-value = 0.05** we reject null hypothesis saying and accept that alternate hypothesis saying that two features are dependent and we will decide to keep which features and drop the features if the **p- values** of it is **greater than > 0.05**.

Formula for Chi-2 Square Test of Independence: -

$$\chi^2 = \sum_{i=1}^k \left(\frac{O_i - E_i}{E_i} \right)^2$$

- Let's see the results after we ran chi-2 square test on the dataset.

	season	year	month	holiday	weekday	workingday	weather
season	*	1.0	0.0	0.683	1.0	0.887	0.021
year	1.0	*	1.0	0.995	1.0	0.98	0.127
month	0.0	1.0	*	0.559	1.0	0.993	0.015
holiday	0.683	0.995	0.559	*	0.0	0.0	0.601
weekday	1.0	1.0	1.0	0.0	*	0.0	0.278
workingday	0.887	0.98	0.993	0.0	0.0	*	0.254
weather	0.021	0.127	0.015	0.601	0.278	0.254	*

Fig 3.3 – Chi-2 test of independence result in Python

2.1.4.3 ANOVA (Analysis of Variance Test)

Anova is a statistical test which analyzes variance. It is helpful in making comparison of two or more means which enables a researcher to draw various results and predictions about two or more sets of data. Anova test includes one-way anova, two-way anova or multiple anova depending upon the type and arrangement of the data. One-way anova.

Analysis of variance which is nothing but a numerical variable, in case of anova we use “1 categorical and 1 numeric variable”. Anova is a statistical technique used to compare the means of 2 or more groups of observations. Anova compares the means between the groups you are interested in and determines whether any of those means are significantly different from each other or not and specifically if tests the null hypothesis.

Here Null Hypothesis (H_0): - Means Are Equal

Alternate Hypothesis (H_1): - Means Are Not Equal

Then if the P-Values is < 0.05 by rejecting null hypothesis and by saying that all the means of different categories values are not equal and we accept alternate hypothesis is true and by saying that two categorical variables means are not same and they are carrying a lot of valuable information. This approach is same in both Python and R.

Here We are doing Anova Test for 3 Numerical Variables i.e, ('count', 'casual', 'registered') VS all Categorical Variables because we know that target variable count = casual + registered.

ANOVA Formula: -

$$F = MST / MSE$$

Where, F = Anova Coefficient, MST = Mean sum of squares due to treatment, MSE = Mean sum of squares due to error.

Formula for MST is given below: -

$$MST = SST / p-1, SST = \sum n (X_i - \bar{x})^2$$

Where, SST = Sum of squares due to treatment, p = Total number of populations, n = Total number of samples in a population.

Formula for MSE is given below: -

$$MSE = SSE / N - p, SSE = \sum (n - 1) S^2$$

Where, SSE = Sum of squares due to error, S = Standard deviation of the samples, N = Total number of observations

- Let's see the results after we ran Anova test target variable cnt in Python.

	df	sum_sq	mean_sq	F	PR(>F)
season	1.0	4.517974e+08	4.517974e+08	288.165767	1.198209e-54
year	1.0	8.821591e+08	8.821591e+08	562.659433	1.907047e-92
month	1.0	2.936495e+07	2.936495e+07	18.729577	1.718470e-05
holiday	1.0	1.102860e+07	1.102860e+07	7.034272	8.171774e-03
weekday	1.0	1.262220e+07	1.262220e+07	8.050701	4.675982e-03
workingday	1.0	4.116578e+06	4.116578e+06	2.625639	1.055862e-01
weather	1.0	2.148995e+08	2.148995e+08	137.067354	4.101328e-29
Residual	723.0	1.133547e+09	1.567838e+06	NaN	NaN

Fig 3.4 – Anova test result for cnt vs all categorical columns in Python

- Let's see the results after we ran Anova test target variable cnt in R.

```
> aov_cnt = aov(cnt~season+yr+mnth+holiday+weekday+workingday+weathersit,data = data)
> summary(aov_cnt)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
season	3	950595868	316865289	436.234	< 2e-16 ***
yr	1	884008263	884008263	1217.030	< 2e-16 ***
mnth	11	187311622	17028329	23.443	< 2e-16 ***
holiday	1	3306975	3306975	4.553	0.03321 *
weekday	6	15839061	2639843	3.634	0.00147 **
weathersit	2	185659616	92829808	127.800	< 2e-16 ***
Residuals	706	512813988	726365		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fig 3.5 – Anova test result for cnt vs all categorical columns in R

Here we can clearly tell which features are more significant for considering and drop others that is season, year, month, weekday, weathersit are very significant with 3 stars' indication and whereas holiday has only 1 star.

- Let's see the results after we ran Anova test target variable casual in Python.

	df	sum_sq	mean_sq	F	PR(>F)
season	1.0	1.523516e+07	1.523516e+07	57.438401	1.070606e-13
year	1.0	2.132690e+07	2.132690e+07	80.405031	2.579127e-18
month	1.0	2.998770e+06	2.998770e+06	11.305730	8.133578e-04
holiday	1.0	1.209921e+06	1.209921e+06	4.561548	3.303283e-02
weekday	1.0	1.709156e+06	1.709156e+06	6.443726	1.134316e-02
workingday	1.0	9.521097e+07	9.521097e+07	358.956985	2.643549e-65
weather	1.0	1.469696e+07	1.469696e+07	55.409345	2.788224e-13
Residual	723.0	1.917710e+08	2.652434e+05	NaN	NaN

Fig 3.6 – Anova test result for casual vs all categorical columns in Python

- Let's see the results after we ran Anova test target variable casual in R.

```
> aov_casual = aov(casual~season+yr+mnth+holiday+weekday+workingday+weathersit,data = data)
> summary(aov_casual)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
season	3	86060770	28686923	199.88	< 2e-16 ***
yr	1	21435934	21435934	149.36	< 2e-16 ***
mnth	11	22768984	2069908	14.42	< 2e-16 ***
holiday	1	2292827	2292827	15.98	7.09e-05 ***
weekday	6	100158620	16693103	116.31	< 2e-16 ***
weathersit	2	10116390	5058195	35.24	2.57e-15 ***
Residuals	706	101325297	143520		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fig 3.7 – Anova test result for casual vs all categorical columns in R

All features seem to be significantly important here.

- Let's see the results after we ran Anova test target variable registered in Python.

	df	sum_sq	mean_sq	F	PR(>F)
season	1.0	3.011024e+08	3.011024e+08	393.987905	2.549080e-70
year	1.0	6.291597e+08	6.291597e+08	823.246021	1.831695e-121
month	1.0	1.359580e+07	1.359580e+07	17.789894	2.780276e-05
holiday	1.0	1.954434e+07	1.954434e+07	25.573470	5.403202e-07
weekday	1.0	5.041941e+06	5.041941e+06	6.597303	1.041296e-02
workingday	1.0	1.389227e+08	1.389227e+08	181.778221	4.044656e-37
weather	1.0	1.171977e+08	1.171977e+08	153.351398	4.448678e-32
Residual	723.0	5.525474e+08	7.642427e+05	NaN	NaN

Fig 3.8 – Anova test result for registered vs all categorical columns in Python

- Let's see the results after we ran Anova test registered variable casual in R.

```
> aov_reg = aov(registered~season+yr+mnth+holiday+weekday+workingday+weathersit,data = data)
> summary(aov_reg)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
season	3	494855113	164951704	390.15	< 2e-16 ***
yr	1	630129518	630129518	1490.43	< 2e-16 ***
mnth	11	90811099	8255554	19.53	< 2e-16 ***
holiday	1	11107006	11107006	26.27	3.83e-07 ***
weekday	6	141730911	23621818	55.87	< 2e-16 ***
weathersit	2	109992296	54996148	130.08	< 2e-16 ***
Residuals	706	298486030	422785		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Fig 3.9 – Anova test result for registered vs all categorical columns in R

All features seem to be significantly important here. But the target variable `cnt = casual + registered` so overall holiday feature isn't contributing much information is the conclusion of anova test we can say.

2.1.4.4 Multicollinearity Check

Using Variance Inflation Factor (V.I.F): - formula for vif $1 / (1-R^2)$

Variance inflation factors measure how much the variance of the estimated regression coefficients are inflated as compared to when the predictor variables are not linearly related. It is used to explain how much multi-collinearity, correlation between predictors exists in an analysis. To interpret the Variance Inflation factors (**VIF**) =1 (**Not correlated**) If $1 < VIF < 5$ (**Moderately correlated**) $VIF \geq 5$ (**Highly Correlated**).

Let's see multi-collinearity in R

```
> vif(data[,c(10,11,12,13)])
Variables      VIF
1      temp 63.010048
2     atemp 63.632085
3      hum  1.059230
4 windspeed 1.097383
```

Fig 4: Here we can see that temp and temp are correlated with each other in R

```
> vifcor(data[,c(10,11,12,13)])
1 variables from the 4 input variables have collinearity problem:
```

atemp

After excluding the collinear variables, the linear correlation coefficients ranges between:

min correlation (hum ~ temp): 0.123703

max correlation (windspeed ~ hum): -0.2018941

```
----- VIFs of the remained variables -----
Variables      VIF
1      temp 1.029967
2      hum 1.052689
3 windspeed 1.057303
```

Fig 4.1: – Multi-collinearity after excluding the collinear variable in R

Let's see multi-collinearity in Python

```
const      45.50
temp       63.01
atemp      63.63
humidity   1.06
windspeed  1.10
dtype: float64
```

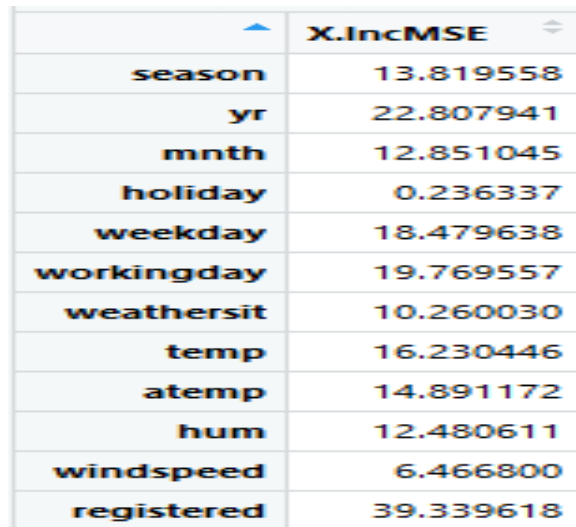
In python also we can see the same multi collinearity between temp and atemp features.

Feature Importance's

Let's see the important features from the dataset in Python.

```
const          17.86
season         3.25
year           1.00
month          3.26
holiday        1.08
workingday     1.07
weekday        1.01
weather        1.01
dtype: float64
```

Let's see the important features from the dataset in R.

A screenshot of an R console output showing a table of feature importance calculated using X.IncMSE. The table has two columns: the feature name and its corresponding X.IncMSE value. The features are listed in descending order of importance. The table is displayed with a light blue header and alternating row colors.

	X.IncMSE
season	13.819558
yr	22.807941
mnth	12.851045
holiday	0.236337
weekday	18.479638
workingday	19.769557
weathersit	10.260030
temp	16.230446
atemp	14.891172
hum	12.480611
windspeed	6.466800
registered	39.339618

Fig 4.2: Important Features in R

Here we are not considering instant, datetime, atemp, holiday, casual and registered features which we are going to drop in dimension reduction stage.

2.1.5 Dimension Reduction

As the number of features grow size of dimension increases so when you are selecting a relevant feature from a raw feature that is called a dimensionality reduction. Dimension reduction stage we are going to drop the features which are irrelevant or redundant or not contributing important information or carrying to explain our target label. The features which we are going to drop are which are either highly positively correlated or not contributing any information explain the target variable are ['instant', 'datetime', 'holiday', 'atemp', 'casual', 'registered'].

Dummy Columns for Categorical Features

In both python and R, we are creating dummy columns for features 'season' and 'weather' because these two features have various levels so we need to create dummy columns and models won't accept the categorical data types especially linear regression model.

Feature season has (1: springer, 2: summer, 3: fall, 4: winter) has four different levels

And Feature weather has (extracted from Freemeteo)

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

temp: Normalized temperature in Celsius. The values are derived

so here we need to create dummy columns for these two feature columns.

Data After creating dummy columns in Python.

	year	month	weekday	workingday	temp	humidity	windspeed	count	season_2	season_3	season_4	weather_2	weather_3
0	0	1	6	0	0.344167	0.805833	0.160446	985	0	0	0	1	0
1	0	1	0	0	0.363478	0.696087	0.248539	801	0	0	0	1	0
2	0	1	1	1	0.196364	0.437273	0.248309	1349	0	0	0	0	0
3	0	1	2	1	0.200000	0.590435	0.160296	1562	0	0	0	0	0
4	0	1	3	1	0.226957	0.436957	0.186900	1600	0	0	0	0	0

Fig 4.3: Data After creating dummy columns in Python

Feature Scaling

It involves in dealing with the parameters of different units and scale. Feature scaling is a method to “**limit the range of variables**”, so that they can be compared on the common ground and it is performed only on the “**continuous variables**”.

Let’s check the distribution of numeric columns

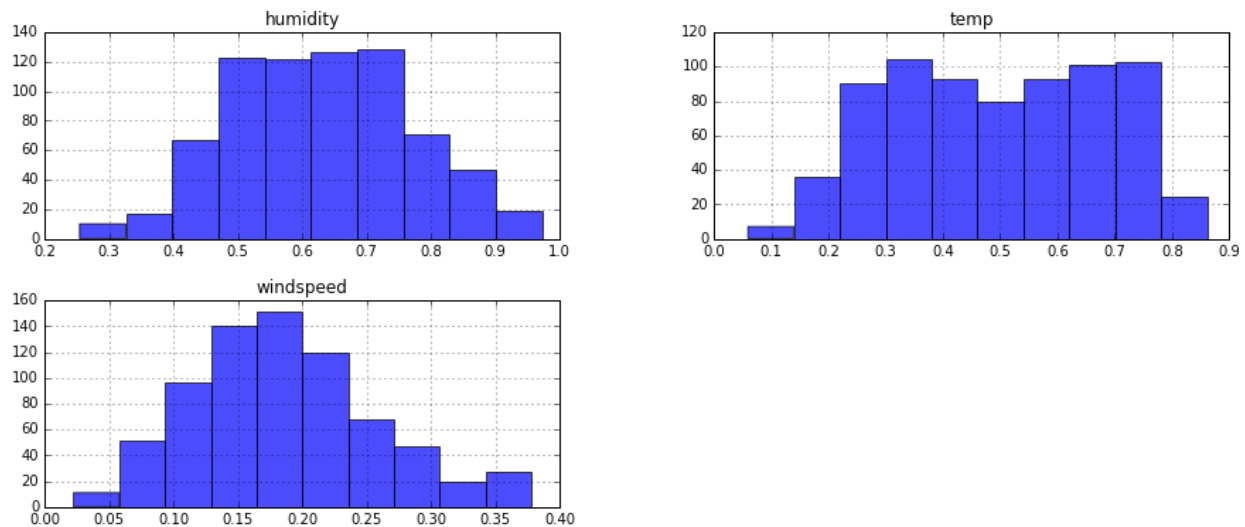


Fig 4.4: Data Uniformly distributed and already scaled in Python

Here data is uniformly distributed and already normalized and the scale ranges between 0 to 1 so we are not applying feature scaling methods but only checking the distribution of the three numeric columns which are only there after dimension reduction stage.

2.1.6 Sampling Data

We are using Simple Random Sampling for splitting the data into train and test sets that is **80 %** of the data to **train** and remaining **20 %** to **test** set.

Random sampling is the purest form of probability sampling it is selected by using chance or random numbers.

2.2 Model Development

The problem category falls under the Supervised learning algorithms that is regression models and we also we have the target variable cnt which is continuous. The algorithms which we have used here to develop are:

- **Linear Regression**
- **K Nearest Neighbor Regressor**
- **Support Vector Regressor (SVM)**
- **Decision Tree Regressor**
- **Random Forest Regressor**
- **XGB (Xtreme Gradient Boosting) Regressor**

Error Metrics for Regression Models used are

- **Explained Variance**
- **Root_Mean_Squared_Error (RMSE)**
- **Mean_Absolute_Percentage_Error (MAPE)**
- **Root_Mean_Squared_Log_Error (RMSLE)**
- **R2**

Explained variance best possible score is 1.0 perfect prediction and less is the worse, If the data is time-series or transition data it is better to use 'RMSE' and if we want to convert error number to a particular percentage the we use 'MAPE'

Formula for MAPE: - Mean Absolute Percentage Error: -

$M = 1/n \sum_{t=1}^n \left(\frac{A_t - F_t}{F_t} \right)$, where n is number of observations, At- Actual values, Ft- Predicted values.

Formula for RMSE/RMSD Root Mean Square Error/Deviation: -

$$RMSE = \sqrt{\sum_{t=1}^n \frac{(\hat{y}_t - y_t)^2}{n}}$$

We want know how well the model predicts new data, not how well it fits the data it was trained with. The key component of most models is difference between actual y and predicted y ('error') to know the percentage of error we are getting for the model. Here in statistical models evaluate the model on both train and test sets.

2.2.1 Developing Regression Models

Multiple Linear Regression Model

Which is one of the statistical model for prediction machine learning algorithms save the 'Patterns' from historical or past data suppose if you are building a model on training data to apply on the same model on test data, what machine learning will do is it will save the patterns, insights or rules and apply the saved rules on the test data.

But that is not the case in statistical models in statistical models it will save 'Numbers' in terms of 'coefficients' it means that for independent variables it will calculate the 'weights', it is also called the coefficients, so the weight will reflect the 'amount of variance' or the amount of information it carries to explain your 'Target Variable'. So here we calculate the weights of each independent variables and use the weight of each variable to calculate on the 'test data'. Weights or coefficients are used to develop the model to perform the predictions on test data.

If we have more than one independent and dependent variable or target variable, then it is a multiple linear regression. Also helps in identifying relationship between two variables same like correlation analysis.

So when we are developing the Multiple Linear Regression Model basically you are building a 'mathematical equation' that is:

The model equation for the multiple linear regression is $y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n + E$

Where,

Y – Target variable

b_0 – is the Intercept

b_1 – is the regression coefficient that measures a unit change in independent variable x_1 changes – change in y

b_2 – is the regression coefficient that measures a unit change in independent variable x_1 changes – change in y

X_1, X_2, \dots, X_n – are the Predictors or the weights of the independent variables

E = is the random error in the prediction, that is the variance that can be accurately predicted by the model, which is also known as residuals.

The ultimate goal of linear regression model is to build the 'equation' which helps to predict our new test cases. The ultimate goal of every model is to reduce the 'Error Rate'. Which our 'Mean

Absolute Error'. The linear regression model uses Least Square estimators for calculating the independent variables.

To build any model we need to have a Hypothesis or Assumption or Guess. The most common test is that a change in x variable does not induce a change in y, which can be stated as:

Null Hypothesis $H_0: b_1 = 0$

Alternate Hypothesis $H_1: b_1 \neq 0$

It means that this $b_1 * x_1$ has no contribution to predict the target variable that is when $[b_1=0]$

If $b_1=0$ it means that it is not carrying information, it is not carrying any variance to explain our target variable.

Once we build the Linear Regression Model We will get some **P-Value** based on that P-Value we will decide that first variable x_1 is useful or not and same for $x_2, x_3 \dots$. We will see if they are useful or not. The P-value is the probability value if it is less than $p < 0.05$ then we **reject** the **null hypothesis** saying that b_1 or x_1 is **useful** to explain the **target variable**.

If H_1 is **true** it implies the population regression equation is a '**flat line**', that is regardless of the value of x & y has the same distribution saying that $b_1 \neq 0$ and the variable is carrying **variance** to **explain** the **target variable**.

Properties of R: - r measures the strength of the linear association between two variables that is strength = how tightly the points follow the straight line. $-1 \leq r \leq 1$.

If r = positive that is **+1**: All the data fall on a line with positive slope.

If r = negative that is **-1**: All the data fall on a line with negative slope.

If r = **zero (0)** that is: no (linear) pattern in the data.

R contains no information about the degree of slope, only the direction of the slope that is (+1 or -1).

On what basis we are going to decide our model is good or bad after developing?

If whatever ever the points falling **below** the line is **negative deviations** or **errors** and **above** the line is **positive deviations**.

R -Squared: - Here the r-squared and Adjusted r-square are the parameters to test on the '**training side**' or training stage but machine learning algorithms we will not freeze our assumption on training data-analysis but in machine learning algorithms we freeze the assumption on test models. But here in statistical models we need to also look at how your model is trained at your '**training data**' these two functions rand adjusted r-square help us to extract the

performance of our model on the ‘training side’. When you extract the r-square numeric value which tells you the **amount of variance** of the **dependent variable** which is explained by **all the independent variables**. For example, if r-square value is 0.86 it means 86 % of the target variable is explained by all the independent variables. It overall measures the strength of association and what is the strength of your model on the training data.

R-Square can be computed as: -

$R^2 = S^2_{\hat{y}} / S^2_y$, which is $\sum_{i=1}^n \left(\frac{(\hat{y}_i - \bar{y})^2}{(y_i - \bar{y})^2} \right)$, where $\hat{y}_i - \bar{y}$ is -> Predicted values- mean of actual values / $\bar{y} - y_i$ is -> actual mean – mean of target variable.

Adjusted R-Square: - is derived from r-squared values but why do we need to go for adjusted r-square values, it may happen that you keep on adding independent variables to the models, at some time it will lead to under the ‘**overfitting**’ it means that your model will be biased it means that if you add unnecessary or irrelevant variables it will low down your ‘**strong predictors**’, **efficiency** of your strong independent variables. Adjusted r-square that penalizes the addition of extraneous predictors to the model.

Adjusted R-square can be computed as: -

$1 - ((1 - R_{sq}) ((N - 1) / (N - K - 1)))$, where N-is number of observations, K is the number of predictors or independent variables.

Adjusted R-square is always \leq R-Square and in real time we always go for Adjusted R-square because it will give exact and accurate information about the variance of variables.

P-Value: - probability value is used to accept or reject null hypothesis and to estimate whether variable is statistically significant or not.

Key Assumptions of Linear Regression model: -

- Linear Relationship
- Multivariate Normality
- No or little Multi-Collinearity
- No auto Correlation

First assumption is that the linear regression model has assumption is that the data fed to the model has linear relationship between the dependent and independent variable, either it is positive or negative or linear relationship between two variables or dependent VS all independent variables.

Second assumption is that linear regression model will assume that your target variable is normally distributed it means that it is following the normality assumption.

Third assumption if you consider two independent variables which are carrying same information it means that you are feeding two highly correlated variables to the model and will lead to multicollinearity effect, which is nothing but it will reduce the information for both the variables.

Fourth assumption it means that there should be no correlation between the residuals.

Linear Regression Results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.823164961689

Explained Variance on Train Data

<=====>

0.831827131222

Explained Variance on Test Data

<=====>

0.772240721771

Performance of linear regression model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **82.31 %**

Linear Regression Error Metrics Results in Python

Root Mean Squared Error -----> 881.23

<=====>

Mean Absolute Percentage Error--> 20.27 %

<=====>

R2 Score -----> 0.77

Accuracy of model is **79.73 %** and MAPE error rate is **20.27 %**

Linear Regression Results in R

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

Residuals:
    Min       1Q   Median       3Q      Max
-3834.4  -325.8   101.6   449.5  2221.9

Coefficients: (6 not defined because of singularities)
(Intercept)      4214.36      462.18      9.119    < 2e-16 ***
season.1       -1669.52     216.11     -7.725    5.24e-14 ***
season.2       -740.60     243.76     -3.038    0.002491 **
season.3       -747.30     216.89     -3.445    0.000613 ***
season.4            NA           NA           NA           NA
yr.0          -2029.37      65.60    -30.935    < 2e-16 ***
yr.1            NA           NA           NA           NA
mnth.1         147.88     221.31      0.668    0.504262
mnth.2         305.73     219.17      1.395    0.163583
mnth.3         606.80     221.51      2.739    0.006352 **
mnth.4         522.25     279.28      1.870    0.062012 .
mnth.5         774.45     298.89      2.591    0.009818 **
mnth.6         567.16     300.54      1.887    0.059666 .
mnth.7          88.66     318.01      0.279    0.780512
mnth.8         480.38     301.24      1.595    0.111358
mnth.9         953.49     247.56      3.851    0.000131 ***
mnth.10        532.62     183.74      2.899    0.003894 **
mnth.11        -70.68     178.27     -0.396    0.691896
mnth.12            NA           NA           NA           NA
weekday.0      -306.85     120.04     -2.556    0.010846 *
weekday.1     -896.62     202.59     -4.426    1.16e-05 ***
weekday.2     -803.53     225.90     -3.557    0.000407 ***
weekday.3     -674.71     226.03     -2.985    0.002960 **
weekday.4     -726.80     223.83     -3.247    0.001236 **
weekday.5     -686.29     223.38     -3.072    0.002228 **
weekday.6            NA           NA           NA           NA
workingday.0   -694.76     193.33     -3.594    0.000355 ***
workingday.1            NA           NA           NA           NA
weathersit.1    2042.13     242.16      8.433    2.91e-16 ***
weathersit.2    1551.83     224.80      6.903    1.39e-11 ***
weathersit.3            NA           NA           NA           NA
temp          4453.60     463.13      9.616    < 2e-16 ***
hum          -1656.88     347.53     -4.768    2.38e-06 ***
windspeed     -2883.77     500.93     -5.757    1.42e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 777.3 on 556 degrees of freedom
Multiple R-squared: 0.8438, Adjusted R-squared: 0.8362
F-statistic: 111.2 on 27 and 556 DF, p-value: < 2.2e-16

Here we can see that **Adjusted R-Squared** value is **83.62 %** explains the **variance** of **target variable**.

Linear Regression Error metrics on train data in R

RMSE	Rsquared	MAE	MAPE
758.4540191	0.8437638	556.8700427	42.93592

Linear Regression Error metrics on test data in R

RMSE	Rsquared	MAE	MAPE
805.2292082	0.8480556	603.4942786	17.45633

MAPE error rate has reduced on the test data **17.45 %** and **R2- is 84.80 %** and **Accuracy** of the model here is **82.55 %**

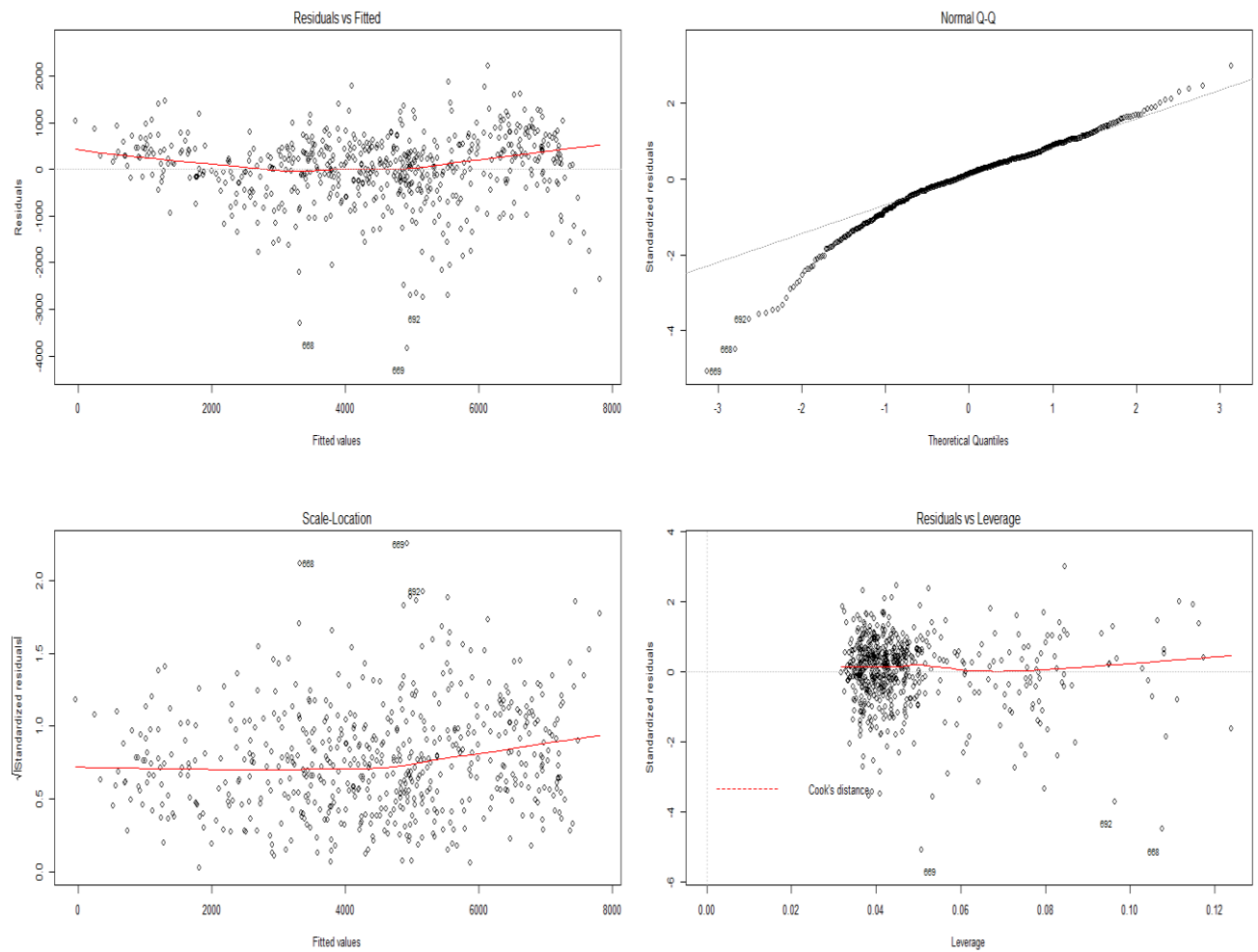


Fig 4.5: Linear Regression Plot for Residuals and Fitted in R

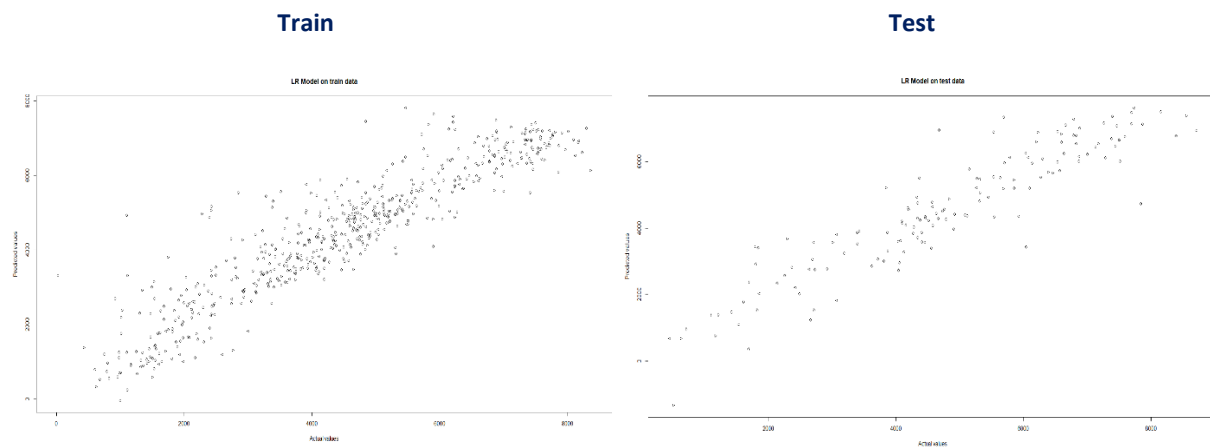


Fig 4.6: Linear Regression Plot for train and test in R

Linear Regression Step-Wise selection forward and backward results in R

```
Call:
lm(formula = cnt ~ season.1 + season.2 + season.3 + yr.0 + mnth.3 +
    mnth.4 + mnth.5 + mnth.6 + mnth.8 + mnth.9 + mnth.10 + weekday.0 +
    weekday.1 + weekday.2 + weekday.3 + weekday.4 + weekday.5 +
    workingday.0 + weathersit.1 + weathersit.2 + temp + hum +
    windspeed, data = dummy_df_train)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-3817.5  -317.2   111.7   446.2  2253.6
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4235.95    450.67   9.399  < 2e-16 ***
season.1      -1441.96    117.15 -12.309  < 2e-16 ***
season.2       -584.11    194.98  -2.996  0.002859 **
season.3       -659.39    160.76  -4.102  4.71e-05 ***
yr.0          -2027.42     65.24 -31.076  < 2e-16 ***
mnth.3         427.59    141.40   3.024  0.002609 **
mnth.4         386.03    209.01   1.847  0.065280 .
mnth.5         632.82    216.79   2.919  0.003653 **
mnth.6         438.18    181.06   2.420  0.015833 *
mnth.8         396.86    150.55   2.636  0.008619 **
mnth.9         903.17    141.18   6.397  3.34e-10 ***
mnth.10        553.24    143.76   3.848  0.000133 ***
weekday.0      -303.68    119.69  -2.537  0.011446 *
weekday.1      -921.68    200.88  -4.588  5.52e-06 ***
weekday.2      -833.39    223.84  -3.723  0.000217 ***
weekday.3      -701.38    223.45  -3.139  0.001785 **
weekday.4      -760.31    221.18  -3.437  0.000631 ***
weekday.5      -716.16    221.30  -3.236  0.001283 **
workingday.0   -727.17    191.00  -3.807  0.000156 ***
weathersit.1    2030.51    241.36   8.413  3.34e-16 ***
weathersit.2    1546.29    224.08   6.901  1.41e-11 ***
temp          4520.54    364.87  12.389  < 2e-16 ***
hum           -1701.96    341.77  -4.980  8.49e-07 ***
windspeed     -2920.12    497.19  -5.873  7.33e-09 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 776.3 on 560 degrees of freedom
Multiple R-squared:  0.8431,    Adjusted R-squared:  0.8366
F-statistic: 130.8 on 23 and 560 DF,  p-value: < 2.2e-16
```

Fig 4.7: Linear Regression Plot Step-wise selection model in R

Linear Regression Step-Wise Selection Error metrics on test data in R

RMSE	Rsquared	MAE	MAPE
801.883727	0.847627	601.222217	17.6618

We can observe that MAPE error rate is 17.66 % and R2- 84 % and Accuracy of model is 82.34 %

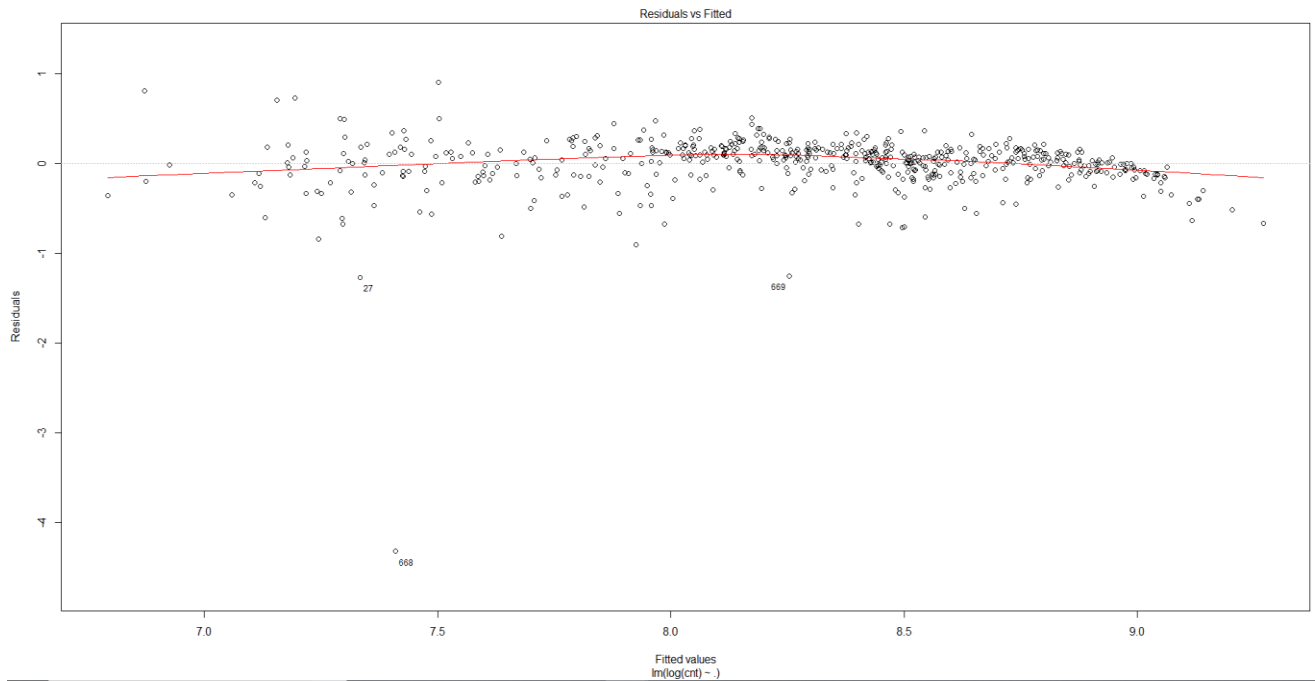


Fig 4.8: Linear Regression Plot for Residuals and Fitted for step-wise selection in R

Here the line fits better to previous one

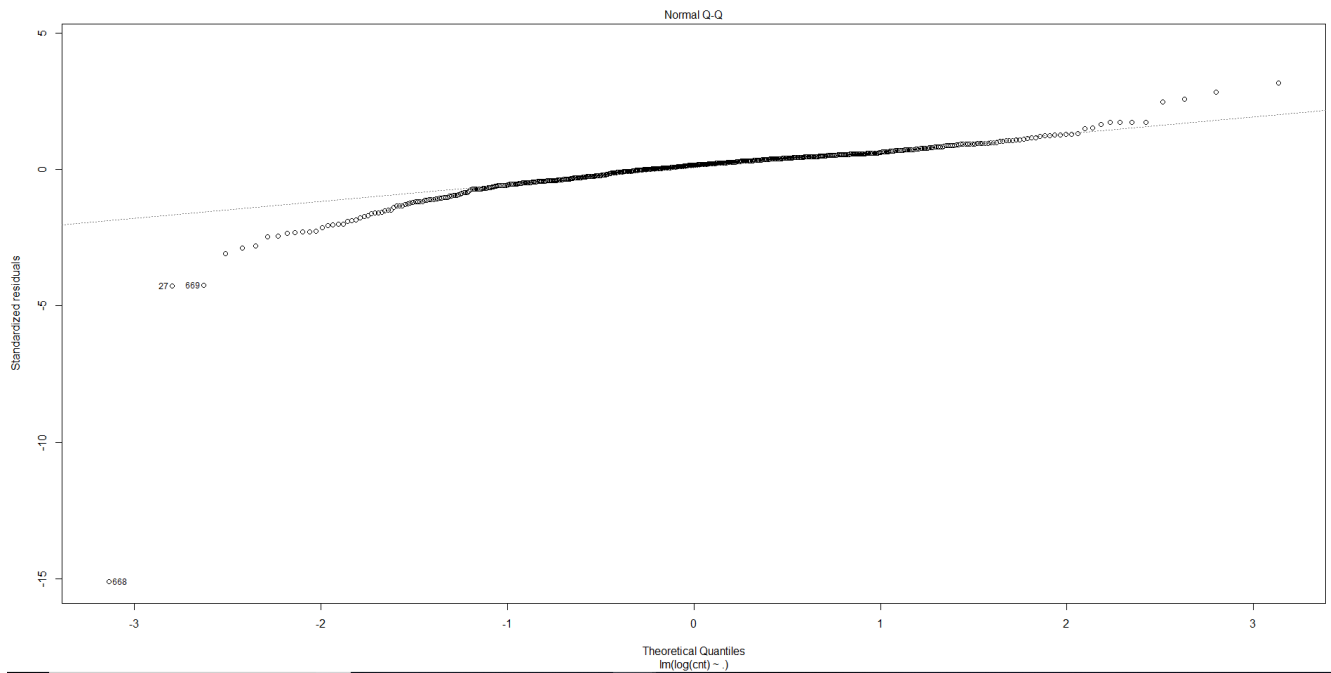


Fig 4.9: Linear Regression Plot for Theoretical quantiles & Normality Q-Q Plot for step-wise selection in R

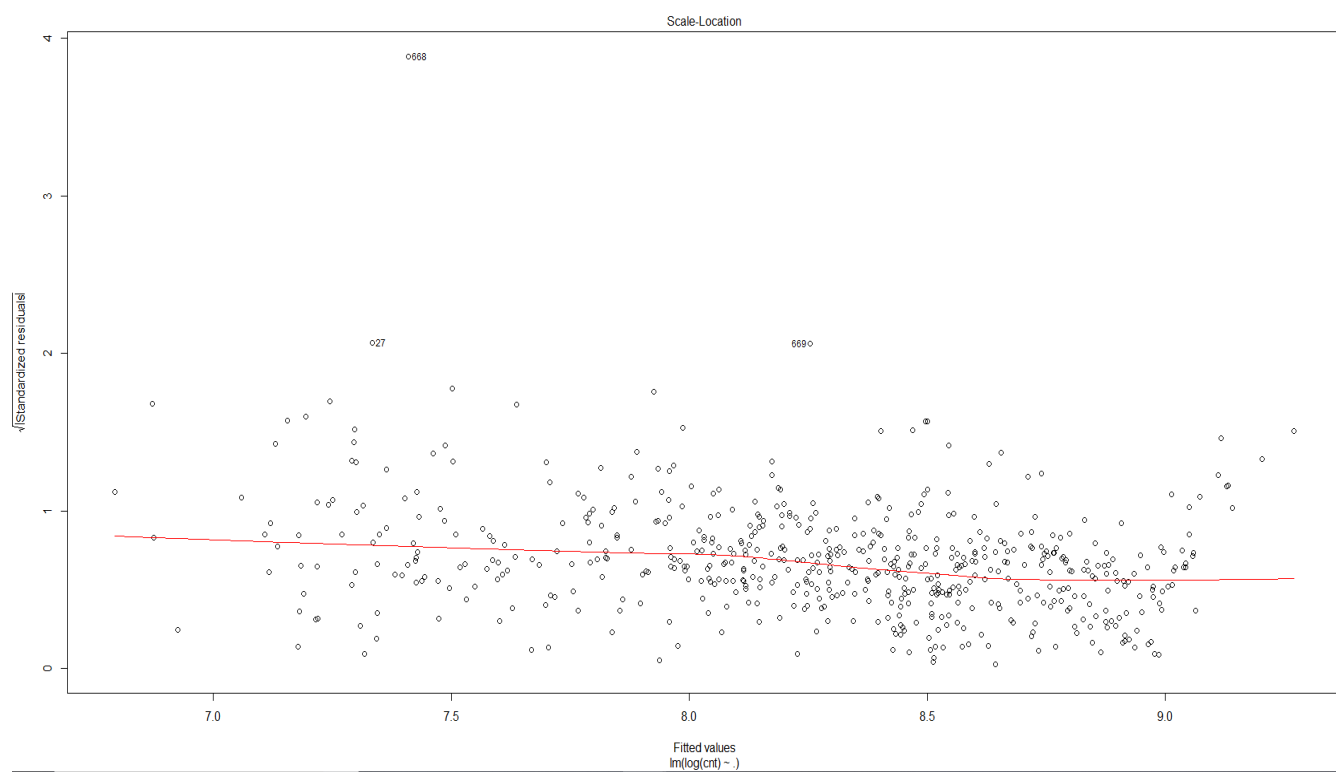


Fig 5: Linear Regression Plot for Fitted Values & Standardized Residuals for step-wise selection in R

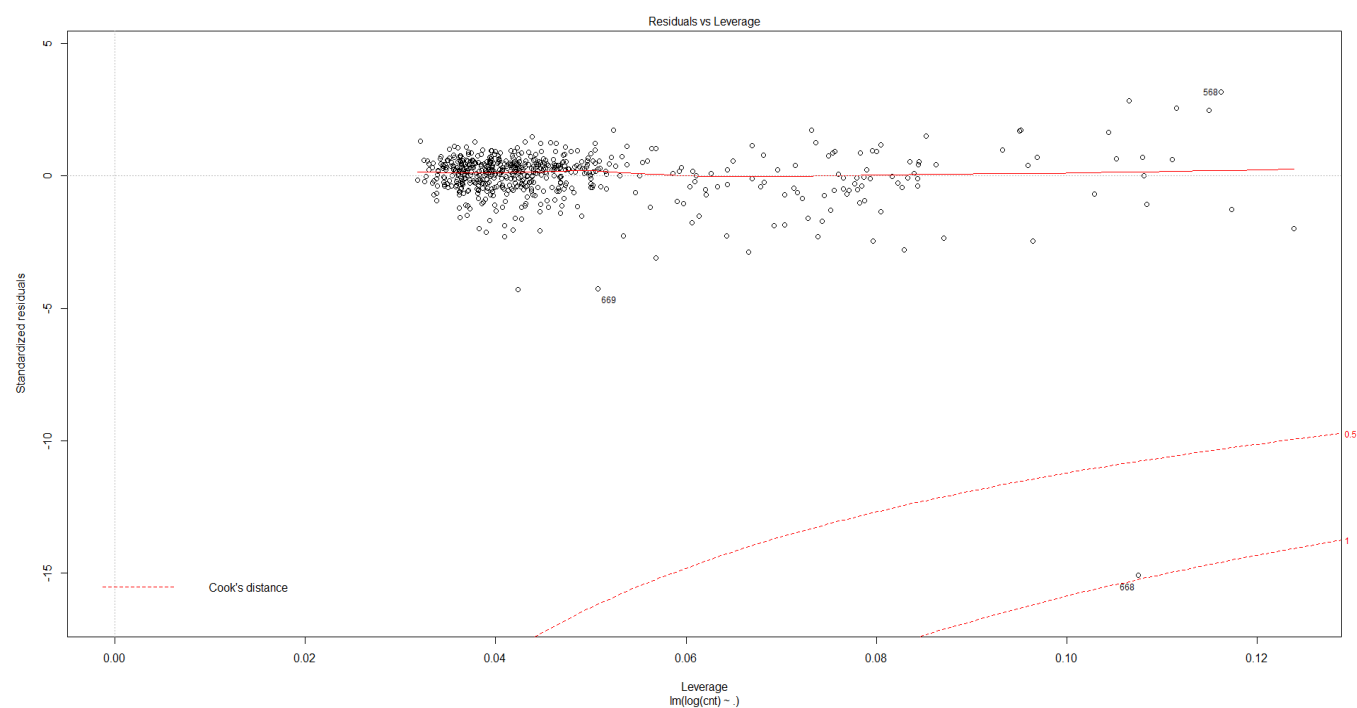


Fig 5.1: Linear Regression Plot for Residuals VS Leverage after log transformation for step-wise selection in R

The Line fits almost better for this model.

Let's Check the summary of the predicted count values

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-1278	3400	4545	4578	6119	7551

From above summary we saw **negative values** of **predicted count**.

We don't want **negative values** as **forecast** for **bike count**. Replace all **negative** numbers with **1**

Let's Check the summary of the actual count values

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
441	3524	4672	4796	6426	8714

As we replaced the negative values, the rmse value got reduced for step-wise forward and backward selection model.

RMSE

789.3738

If we want to penalize under-prediction of demand, rmsle might be a better metric

RMSLE after replacing the negative values with 1 = 0.5201974

Applying **Log Transformation** for **Linear Regression Step-wise forward** and **backward Selection model**.

Let's check the summary of predicted count values, it shows there are no negative values

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
487	3030	4106	4548	6147	9334

RMSLE After **log transformation** & again converted to **exponential**: 0.2388554

K Nearest Neighbor Regressor model results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.722216955855

Explained Variance on Train Data

<=====>

0.824756630648

Explained Variance on Test Data

<=====>

0.724071257821

Performance of KNN model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **72.22 %**

K Nearest Neighbor Regressor Error Metrics Results in Python

Root Mean Squared Error -----> 969.95

<=====>

Mean Absolute Percentage Error---> 26.67 %

<=====>

R2 Score -----> 0.72

Accuracy of model is **73.33 %** and MAPE error rate is **26.67 %**

Support Vector Regressor Model results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.0199276683063

Explained Variance on Train Data

<=====>

0.0217576990689

Explained Variance on Test Data

<=====>

0.019281737858

Support vector Regressor is not performing well with Gaussian kernel on both the data-sets.

Support Vector Regressor Model Error Metrics Results in Python

Root Mean Squared Error -----> 1828.62

<=====>

Mean Absolute Percentage Error--> 54.55 %

<=====>

R2 Score -----> 0.02

Accuracy of model is **45.45 %** and MAPE error rate is **54.55 %**

Support Vector Regressor Model Error Metrics on train data results in R

RMSE	Rsquared	MAE	MAPE
626.0047375	0.8961997	426.7444956	37.97459

We can observe that MAPE error rate on the train data **37.97 %** and R-squared- **89.16%**

Support Vector Regressor Model on test data results in R

RMSE	Rsquared	MAE	MAPE
690.0739705	0.8800138	502.0440156	15.4358

We can observe that **MAPE** error rate has reduced on the test data **15.43 %** and R-squared- **88 %** of variance is explained to the target variables and also got reduced, whereas **accuracy** is **84.57 %** of the model here and SVM performs well in R when compared to in Python.

Let's see the SVM plots on train and test data sets in R

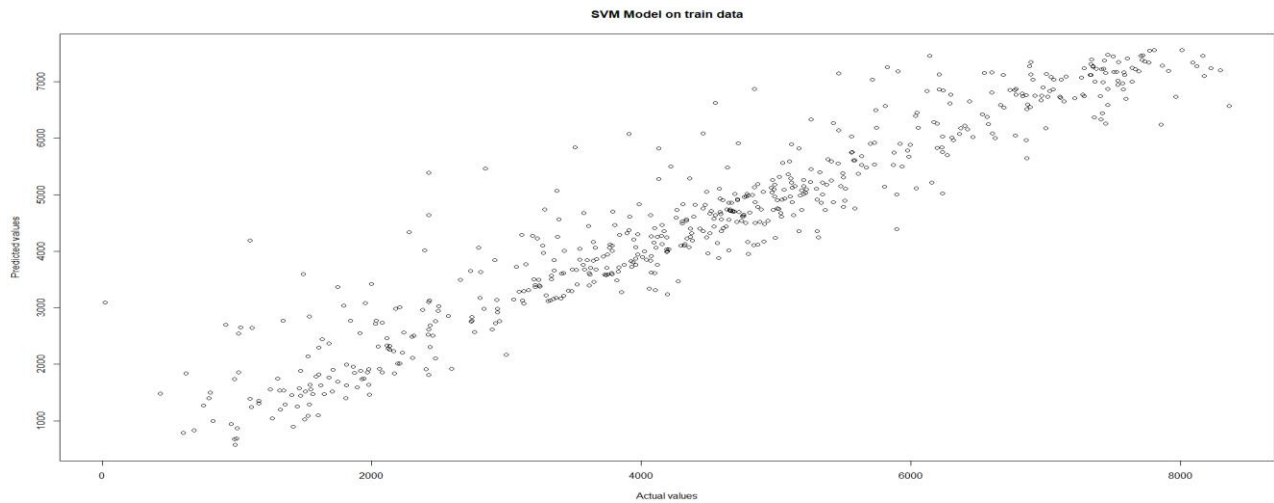


Fig 5.5: SVM Regressor Plot on train data in R

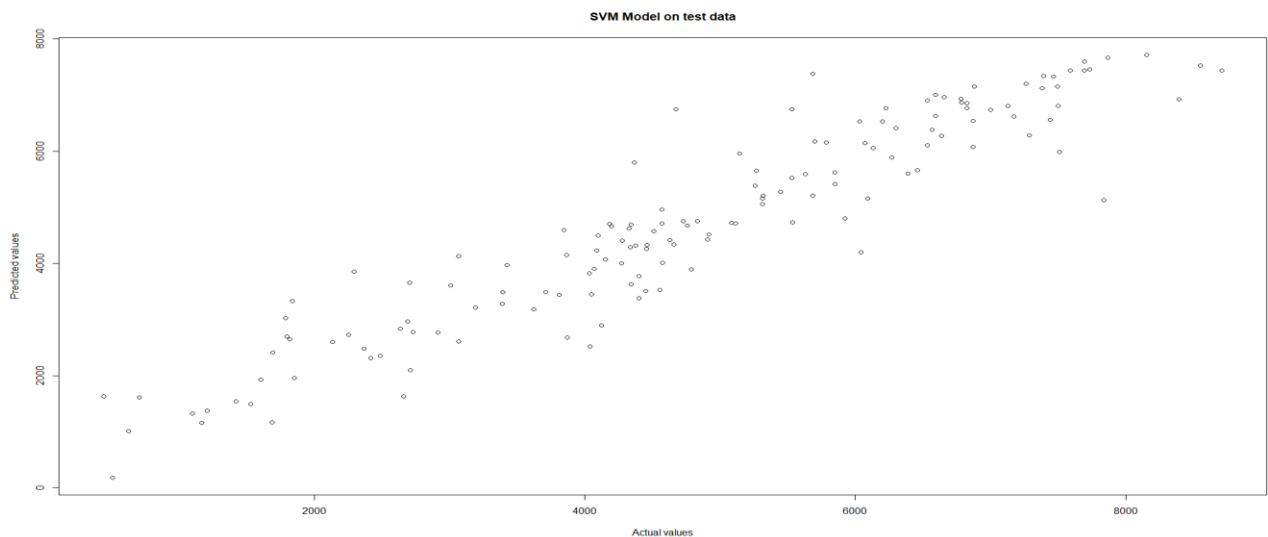


Fig 5.6: SVM Regressor Plot on test data in R

Decision Tree Regressor Model results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.662445439664

Explained Variance on Train Data

<=====>

0.683729557525

Explained Variance on Test Data

<=====>

0.678459809753

Performance of Decision Tree model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **66.24 %**

Decision Tree Regressor Model Error Metrics Results in Python

Root Mean Squared Error -----> 1047.05

<=====>

Mean Absolute Percentage Error--> 27.11 %

<=====>

R2 Score -----> 0.68

Accuracy of model is **72.89 %** and MAPE error rate is **27.11 %**

Let's see the Decision Tree Plot in R

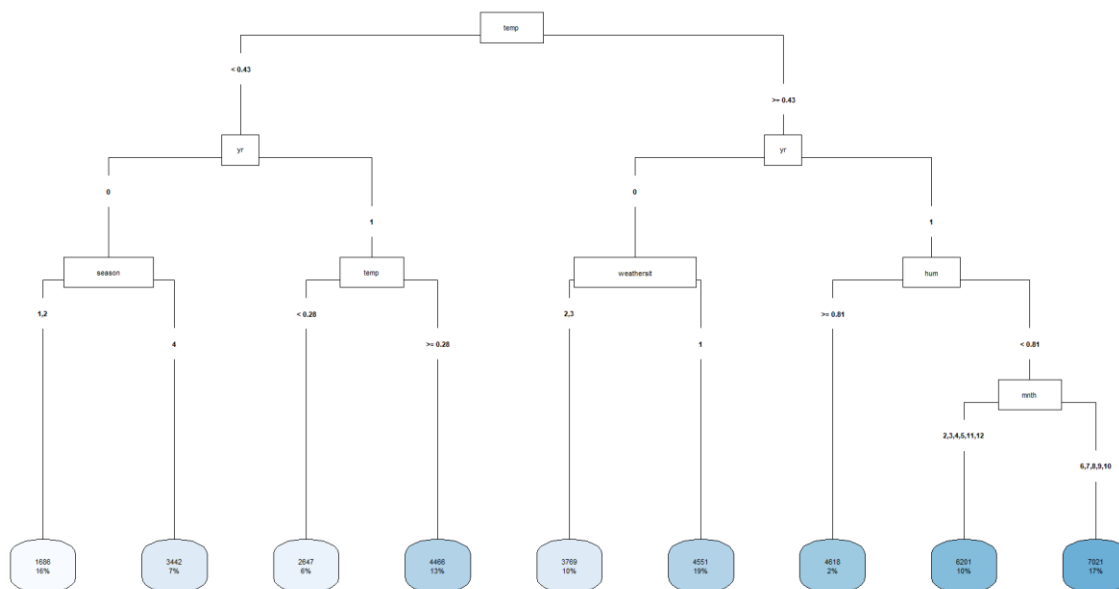


Fig 5.7: Decision Tree Regressor Graph in R

Decision Tree Regressor Model Error Metrics on train data results in R

RMSE	Rsquared	MAE	MAPE
851.7111567	0.8029812	632.6669863	56.38422

We can observe that MAPE error rate is **56.38 %** and R-squared is **80 %** and Accuracy is **43.26 %**.

Decision Tree Regressor Model Error Metrics on test data results in R

RMSE	Rsquared	MAE	MAPE
952.3817605	0.7683867	720.0017702	26.99524

We can observe that **MAPE error rate** has reduced on test data is **26.99 %** and **R-squared** is **76.83 %** it has reduced and the **accuracy** of the model is **73.01 %** it has increased.

Random Forest Regressor Model results in Python

K-Fold Cross Validation Explained Variance

<=====>
0.868696945482

Explained Variance on Train Data

<=====>
0.98226807977

Explained Variance on Test Data

<=====>
0.893250328622

Performance of Random Forest model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **86.86 %** random forest is performing well when compared to other models.

Random Forest Regressor Model Error Metrics Results in Python

Root Mean Squared Error -----> 603.3

<=====>

Mean Absolute Percentage Error--> 13.55 %

<=====>

R2 Score -----> 0.89

Accuracy of model is **86.54 %** and MAPE error rate is **13.55 %** when compared other models the mape is less hen compared to other models till now and accuracy of random forest is performing well.

RF Model in R summary: -

Mean of squared residuals: 503733

% Var explained: 86.32

86.32 % of Variance is explained by the model of target variable cnt here.

Random Forest Regressor Model Error Metrics on train data results in R

RMSE	Rsquared	MAE	MAPE
333.8655091	0.9746288	235.2725488	22.57292

We can observe that MAPE error rate is 22.57 % and R-squared is 97 % and Accuracy of the model on train data is 77.43 %.

Random Forest Regressor Model Error Metrics on test data results in R

RMSE	Rsquared	MAE	MAPE
662.707111	0.900422	498.064969	18.46774

We can observe that MAPE error rate has reduced here on test data is 18.67 % and R-squared is 90 % it has reduced here and Accuracy of the model on test data is 81.33 % and it has increased on test data.

Important Features of Random Forest Model: -

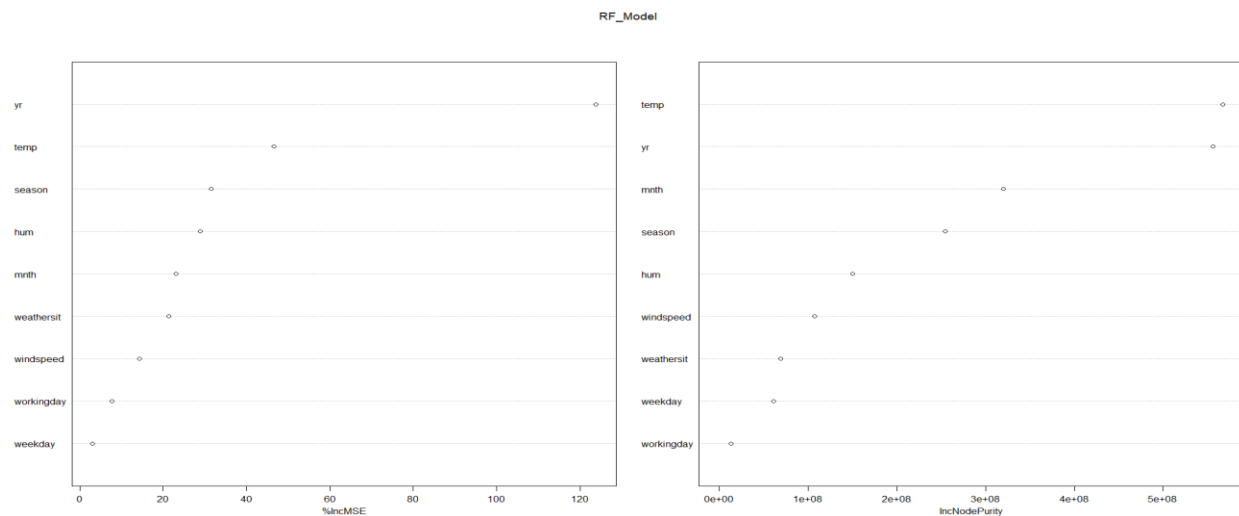


Fig 5.8: Random Forest Regressor Important Features in R

Let's see the Random Forest Regressor plots on train and test data sets in R

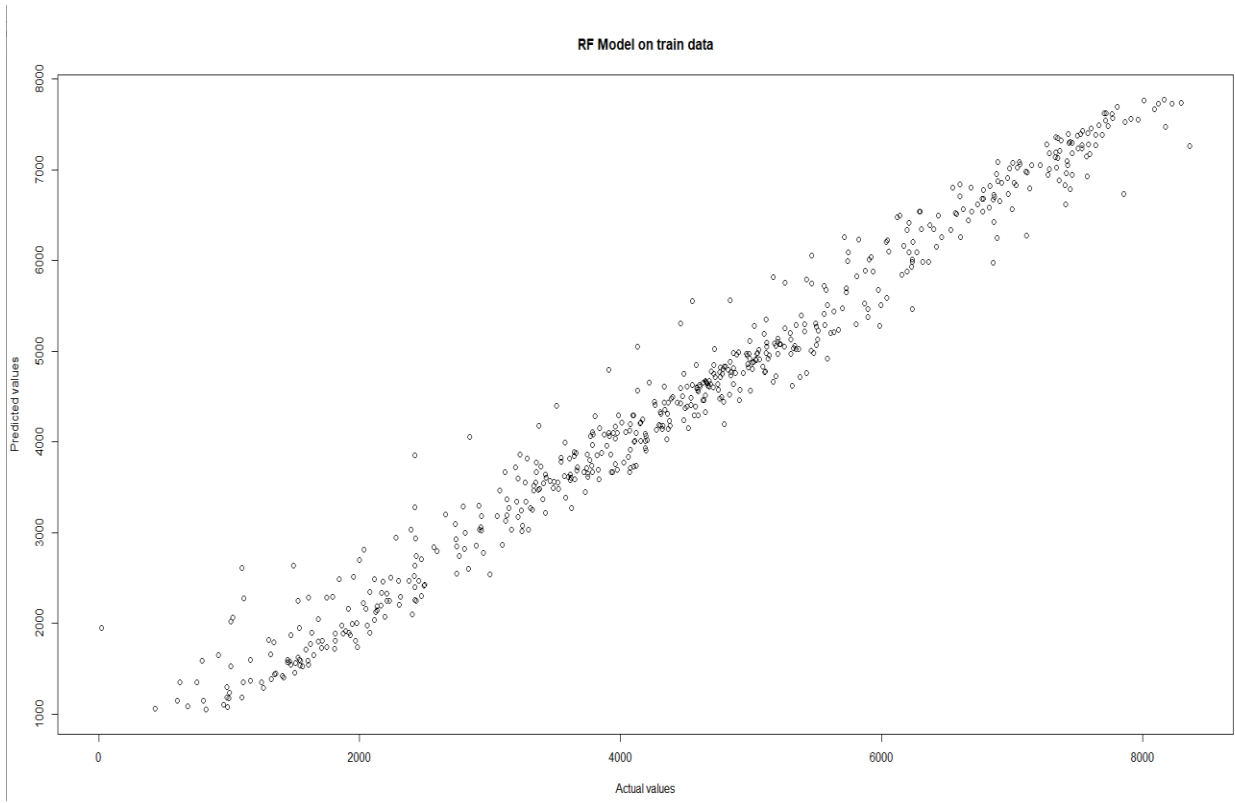


Fig 5.9: Random Forest Regressor Plot on train data in R

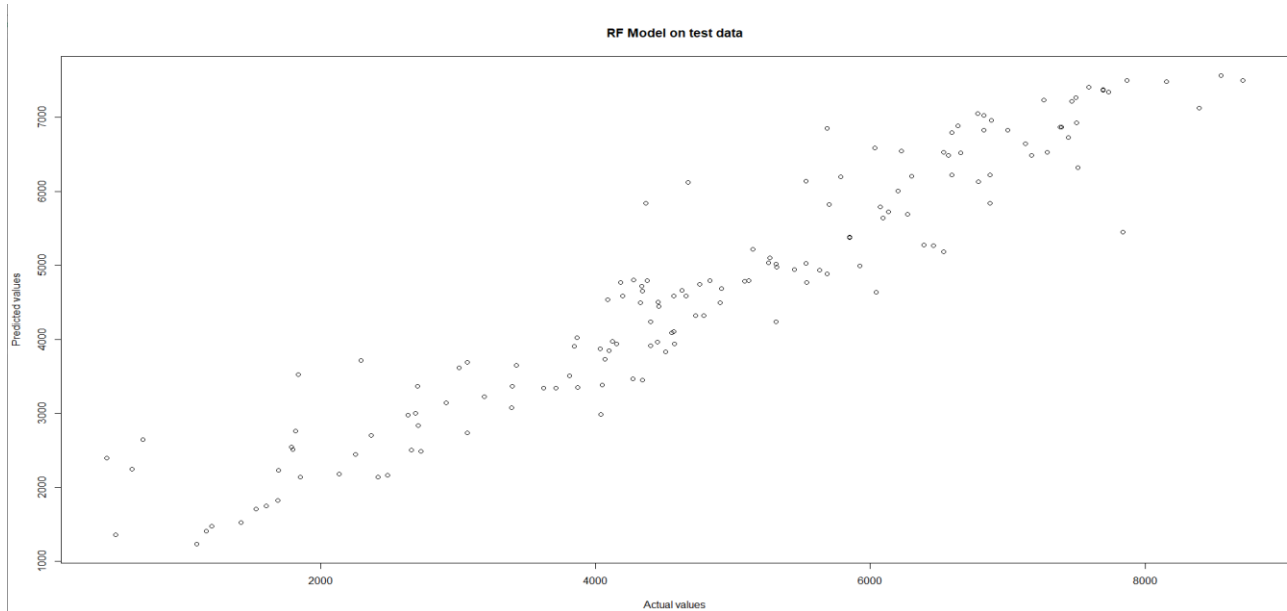


Fig 6: Random Forest Regressor Plot on test data in R

XGB Regressor Model results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.882039850436

Explained Variance on Train Data

<=====>

0.950247746798

Explained Variance on Test Data

<=====>

0.873498282733

Performance of XGB Regressor model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **82.20 %** is performing well now when compared to other models.

XGB Regressor Model Error Metrics Results in Python

Root Mean Squared Error -----> 656.75

<=====>

Mean Absolute Percentage Error--> 13.97 %

<=====>

R2 Score -----> 0.87

Accuracy of model is **86.13 %** is also good and MAPE error rate is **13.97 %** it is also performing well with less mape error rate.

2.2.2 Model Selection and Evaluation

Table 2.2: Best models which performed well in Python comparison

Error Metrics in Python	MLR	Random Forest	Decision Tree	KNN	XGB
K-Fold-CV Explained variance	82.31 %	86.86 %	66.24 %	72.22 %	88.20 %
Explained variance on train data	83.18 %	98.22 %	68.37 %	82.47 %	95.02 %
Explained variance on test data	77.22 %	89.32 %	67.84 %	72.40 %	87.34 %
RMSE	881.23	603.3	1047.05	969.95	656.75
MAPE	20.27 %	13.55 %	27.11 %	26.67 %	13.97 %
R ² - Score	0.77	0.89	0.68	0.72	0.87

Random forest model has outperformed when compared with other model and next XGB model is doing well so we will select the final model from XGB and Random Forest model after doing Hyper-parameter tuning and optimization.

Table 2.3: Best models which performed well in R on Train Data comparison

Error Metrics				
TRAIN DATA	R ² - Score	MAPE	RMSE	MAE
MLR	0.843	42.93	758.45	556.87
SVM	0.896	37.97	626.00	426.74
Decision Tree	0.802	56.38	851.71	636.66
Random Forest	0.974	22.57	333.86	235.27

Table 2.4: Best models which performed well in R on Test Data comparison

TEST DATA	Error Metrics			
	R ² - Sore	MAPE	RMSE	MAE
MLR	0.84	17.45	805.22	603.49
SVM	0.88	15.43	690.07	502.04
Decision Tree	0.76	26.99	952.38	720.00
Random Forest	0.90	18.46	662.70	498.06

In R Also we can see that Random Forest model outperforms other models on both the train and test data. So after Random Forest model SVM is performing well in R with less MAPE than compared to Random Forest so let's apply Hyper-parameter tuning and optimization for Random Forest and SVM model and Select the Final Model.

2.2.3 Hyper Parameter Tuning and Optimization

Now let's Tune both Random Forest and XGB Regressor models so that we can find the best parameters for the model which would help for tuning and getting the optimum values which can be used in function to increase the accuracy of the model.

Hyper parameter tuning is done for finding the optimum parameters for the particular model Such as 'n_estimators', 'max_depth', 'Learning_rate', 'gamma', 'subsample', 'max_features', 'min_samples_leaf', 'min_samples_split' are selected for getting the best performance on the tuned model.

XGB Regressor Hyper Parameter Tuning & Optimization in Python

```
# Hyper Paramter Tuning XGB Model to Find Optimum parameters
xgb_model = XGBRegressor()

params = [{'n_estimators':[250,350,450,550], 'max_depth':[2,3,5,7],
            'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1], 'gamma':[0, 0.001, 0.01, 0.03, 0.05],
            'subsample':[1, 0.5, 0.7, 0.8, 0.9]}]

grid_search = GridSearchCV(estimator=xgb_model, param_grid=params, cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)
print('Best Score ==>', grid_search.best_score_)
print('Best Params ==>', grid_search.best_params_)

# Best Score ==> 0.897321837387
# Best Params ==> {'gamma': 0, 'learning_rate': 0.045, 'max_depth': 3, 'n_estimators': 550, 'subsample': 0.5}

Best Score ==> 0.897321837387
Best Params ==> {'gamma': 0, 'learning_rate': 0.045, 'max_depth': 3, 'n_estimators': 550, 'subsample': 0.5}
```

Tuned XGB Regressor Model Results in Python

K-Fold Cross Validation Explained Variance

<=====>

0.894248319085

Explained Variance on Train Data

<=====>

0.980059869362

Explained Variance on Test Data

<=====>

0.88862483886

Performance of Tuned XGB Regressor model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **89.42 %** XGB Regressor is performing well when compared to other models.

Tuned XGB Regressor Model Error Metrics in Python

Root Mean Squared Error -----> 616.23

<=====>

Mean Absolute Percentage Error--> 12.9 %

<=====>

R2 Score -----> 0.89

Accuracy of model is **87.1 %** is also good and MAPE error rate is **12.9 %** it is also performing well after XGB Regressor model with less mape error rate.

Let's see the Scatter Plot of the Tuned XGB Regressor Model and see how many data points fall on the line, above the line and below the line.

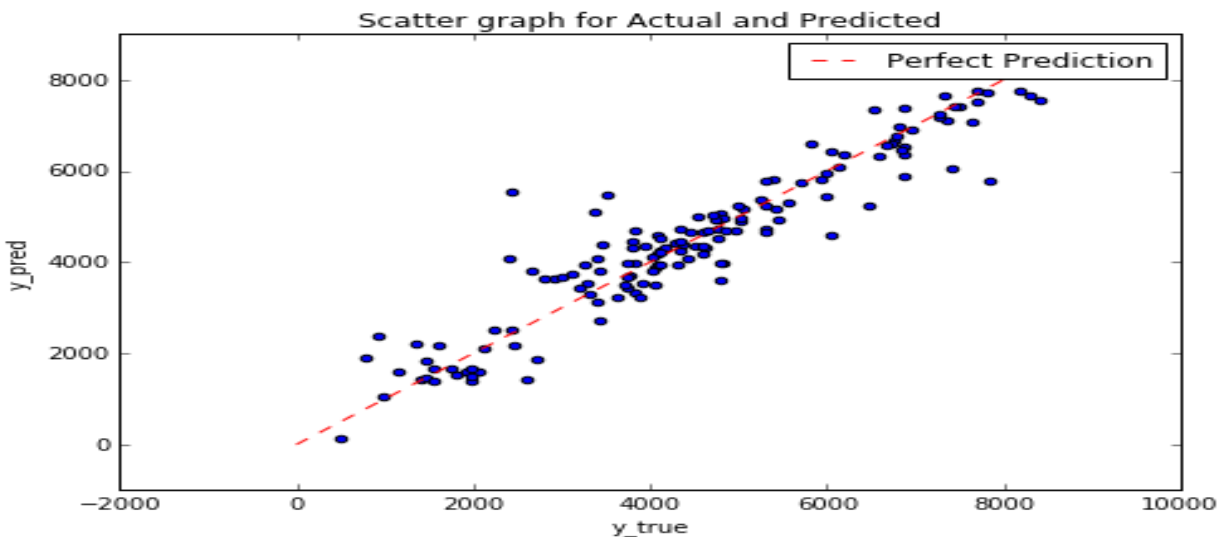


Fig 6.1: Scatter Plot for Tuned XGB Regressor model in Python

Most of the data points are on the fitted regression line and few are slightly deviating from the line the data points which ever are falling below the line are the negative deviations or error and above the line are the positive deviations we could interpret them in such way. The prediction is not perfect because any model cannot be perfect because of the randomness in the data that is the customer ay have rented the bikes randomly so there no such particular pattern here the model could have interpreted the values so the prediction is not perfect but is still line looks quite a good fit for the model and data points here.

Random Forest Regressor Hyper Parameter Tuning & Optimization in Python

```
# Hyper Paramter Tuning Random Forest Model to Find Optimum parameters
rf_model = RandomForestRegressor(random_state=1)
params = [{'n_estimators':[200,300,400,500,600,700,800,1000], 'max_features':['auto', 'sqrt', 'log2'],
          'min_samples_split':[2,4,6], 'max_depth':[2,4,6,8,10,12,14,16], 'min_samples_leaf':[2,3,5]}]

grid_search = GridSearchCV(estimator = rf_model, param_grid = params, cv = 5,scoring = 'explained_variance', n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)
print('Best Score ==>', grid_search.best_score_)
print('Best Params ==>', grid_search.best_params_)

# Best Score ==> 0.865267759445
# Best Params ==> {'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 800}

Best Score ==> 0.865267759445
Best Params ==> {'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 800}
```

Tuned Random Forest Regressor Model Results in Python

K-Fold Cross Validation Explained Variance

```
<=====>
0.868224404272
```

Explained Variance on Train Data

```
<=====>
0.952460550882
```

Explained Variance on Test Data

```
<=====>
0.866280370665
```

Performance of Tuned random forest model with K-Fold cross validation accuracy and explained variance on both train and test data the explained variance for whole data is **86.82 %** random forest is performing well when compared to other models.

Tuned Random Forest Regressor Model Error Metrics in Python

Root Mean Squared Error -----> 675.22

<----->

Mean Absolute Percentage Error--> 16.8 %

<----->

R2 Score -----> 0.87

Accuracy of model is 83.2 % is also good and MAPE error rate is 16.8 % it is also performing well after random forest model with less maape error rate.

Random Forest Regressor Hyper Parameter Tuning & Optimization on Train Data in R

RMSE	Rsquared	MAE	MAPE
334.0120332	0.9742905	232.8790834	24.27306

We can observe that MAPE error rate is 24.27 % and R-squared is 0.97 and Accuracy of the model on train data is 75.73 %.

Random Forest Regressor Hyper Parameter Tuning & Optimization on Test Data in R

RMSE	Rsquared	MAE	MAPE
629.9692316	0.9033033	466.7887276	17.33545

We can observe that MAPE error rate has reduced here on test data is 17.33 % and R-squared is 0.90 and Accuracy of the model on test data is 81.67 % and it has increased on test data.

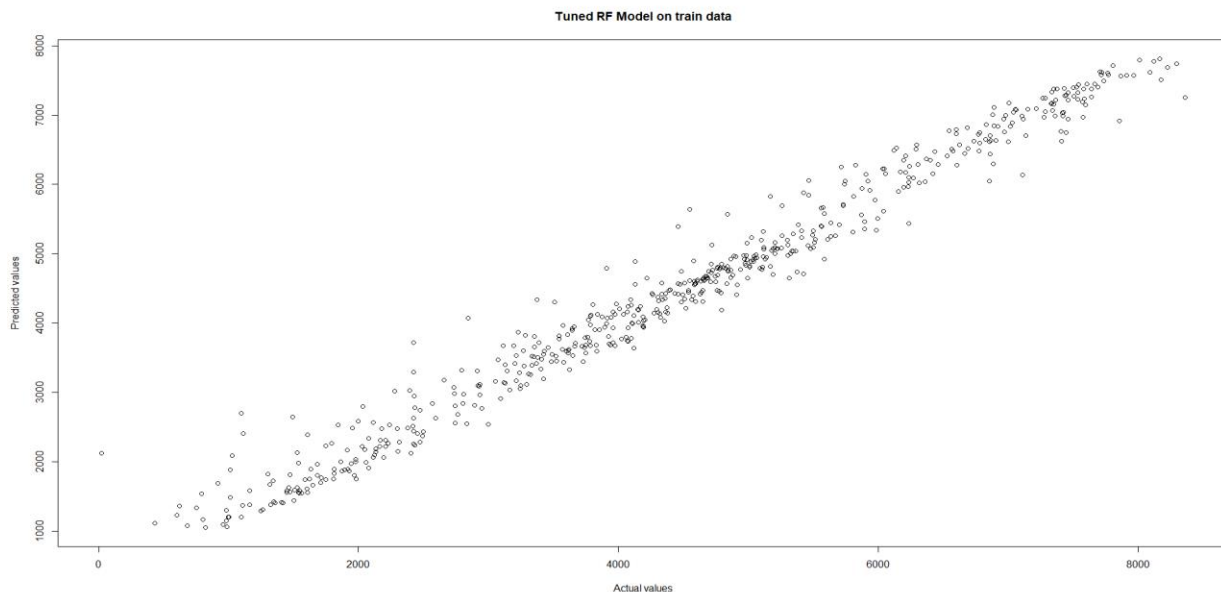


Fig 6.2: Tuned Random Forest Regressor model on train data in R

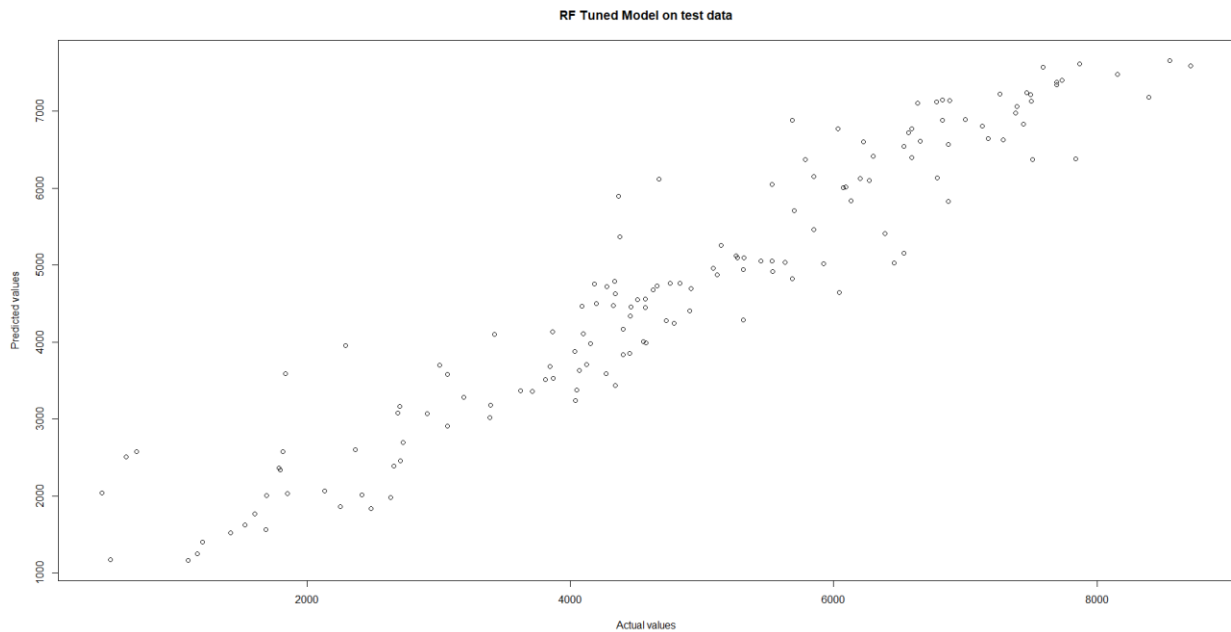


Fig 6.2: Tuned Random Forest Regressor model on test data in R

Let's See the Error plot of Tuned Random Forest Model

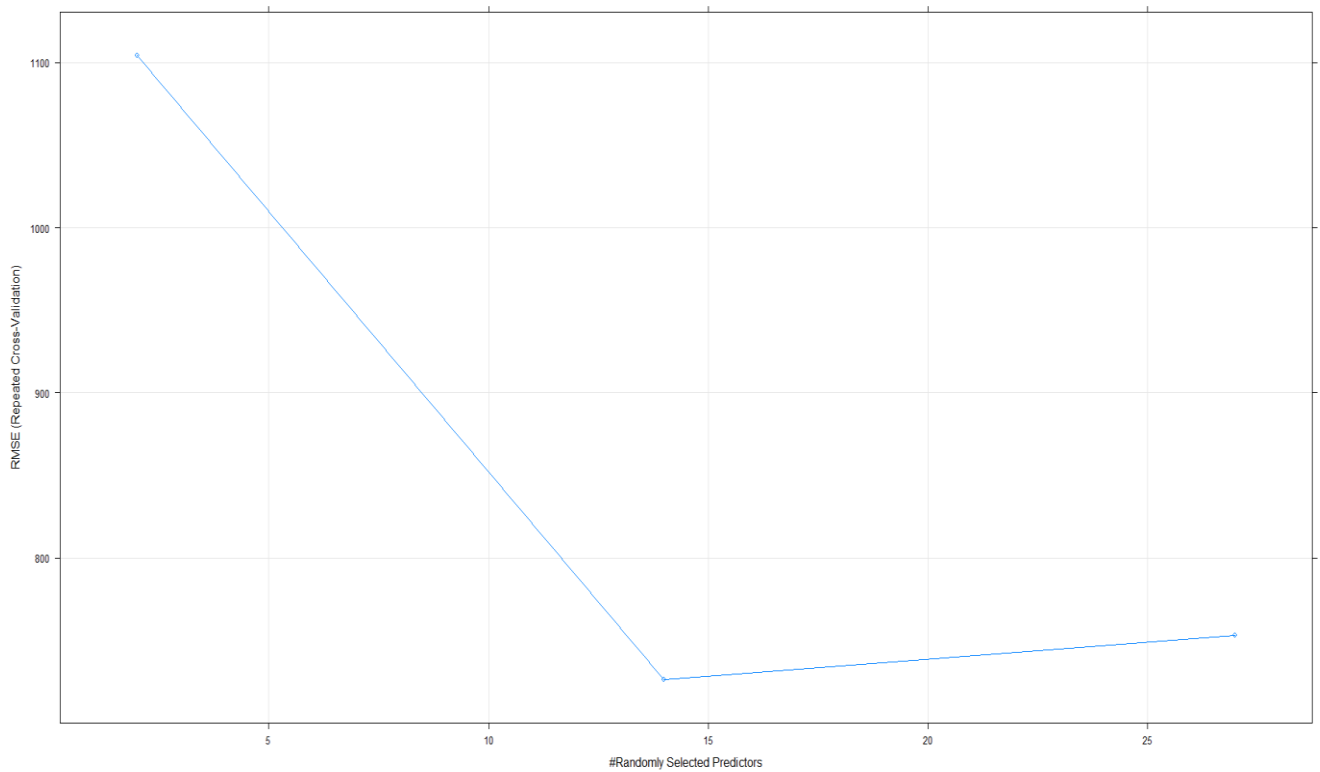


Fig 6.3: Tuned Random Forest Regressor model Error Plot for RMSE in R

SVM Regressor Hyper Parameter Tuning & Optimization on Train Data in R

RMSE	Rsquared	MAE	MAPE
760.4524668	0.8430114	557.6934236	44.26525

We can observe that MAPE error rate is **44.26 %** and **R-squared** is **0.84** and **Accuracy** of the model on train data is **55.74 %**.

SVM Regressor Hyper Parameter Tuning & Optimization on Test Data in R

RMSE	Rsquared	MAE	MAPE
797.3890684	0.8503455	600.1752801	17.66163

We can observe that MAPE error rate has reduced here **17.66 %** and **R-squared** is **0.85** and **Accuracy** of the model on test data has increased here **82.34 %**.

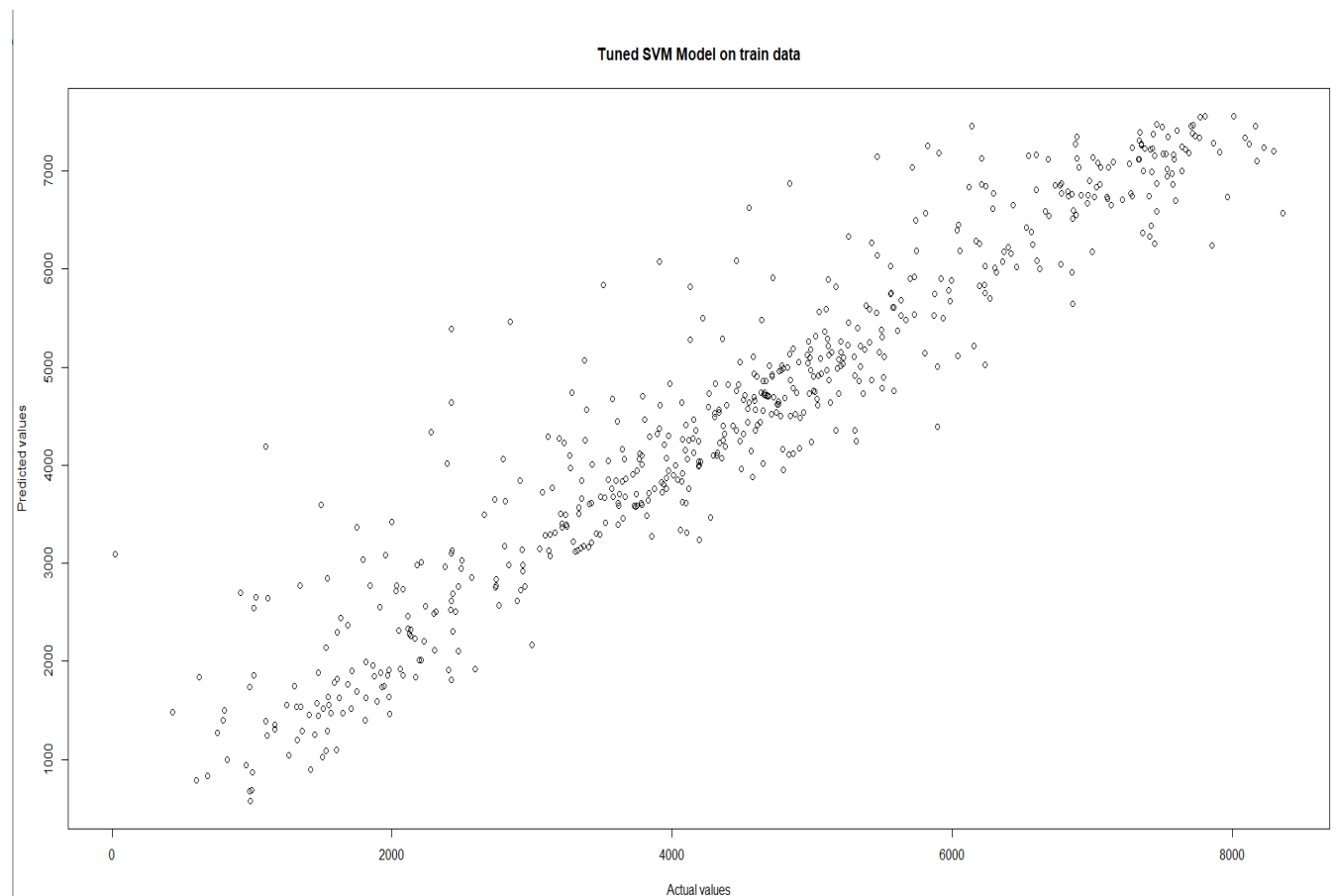


Fig 6.4: Tuned SVM Regressor model on train Data in R

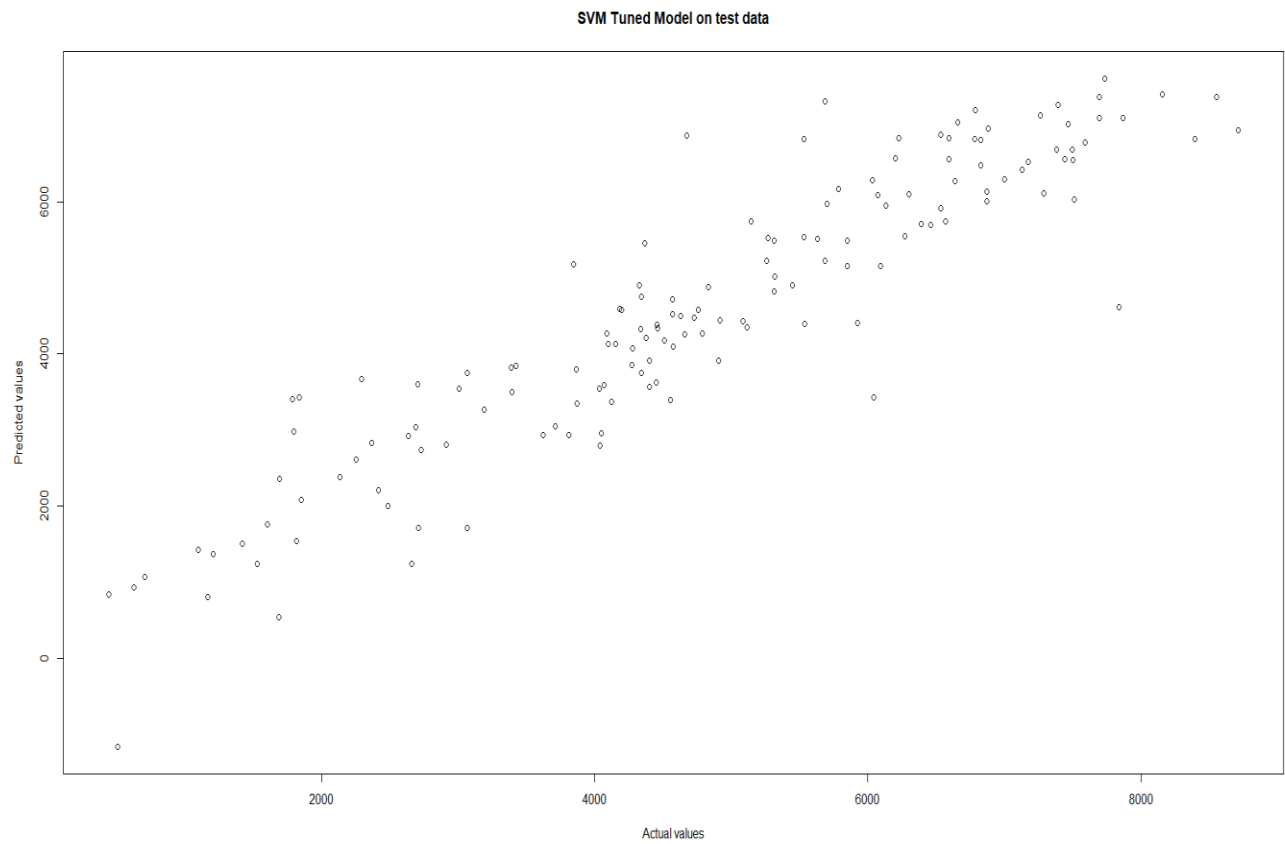


Fig 6.5: Tuned SVM Regressor model on test Data in R

Let's See the Error plot of Tuned SVM Model

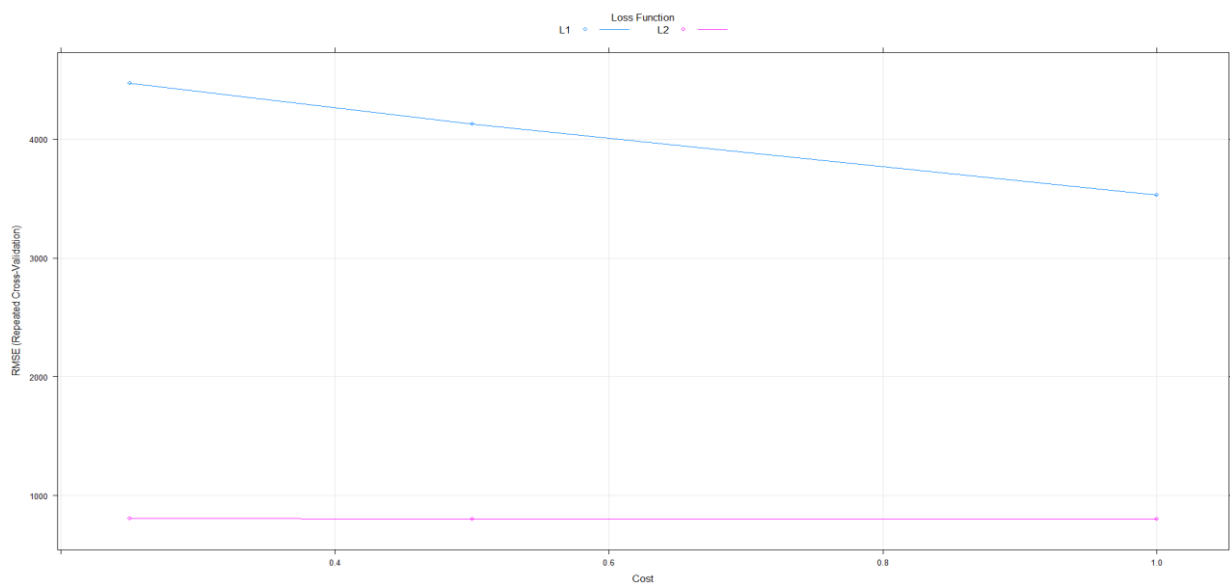


Fig 6.6: Tuned SVM Regressor model Error Plot for RMSE Loss Function L1 and L2 in R

2.2.4 Final Model Selection in Both R & Python

Table 2.5: Comparing Hyper Parameter Tuned Models XGB & Random Forest in Python

Error Metrics in Python	Random Forest	XGB
K-Fold-CV Explained variance	86.91 %	89.42 %
Explained variance on Train data	95.23 %	98.00 %
Explained variance on Test data	86.60 %	88.86 %
RMSE	675.73	616.23
MAPE	16.73 %	12.9 %
R ² - Score	0.87	0.89

As we can clearly see that **XGB Regressor** outperforms with random forest model the **Final and best model** in **Python** is XGB Regressor model.

Table 2.6: Comparing Hyper Parameter Tuned Models SVM & Random Forest on train data in R

Error Metrics				
TRAIN DATA	R ² - Sore	MAPE	RMSE	MAE
Random Forest	0.97	24.27	334.01	232.87
SVM	0.84	44.26	760.45	557.69

Table 2.7: Comparing Hyper Parameter Tuned Models SVM & Random Forest on test data in R

Error Metrics				
TEST DATA	R ² - Sore	MAPE	RMSE	MAE
Random Forest	0.90	17.33	629.96	466.78
SVM	0.85	17.66	797.38	600.17

We can clearly observe that the **Tuned Random Forest** performs well on both **train & test data**.

CHAPTER 3

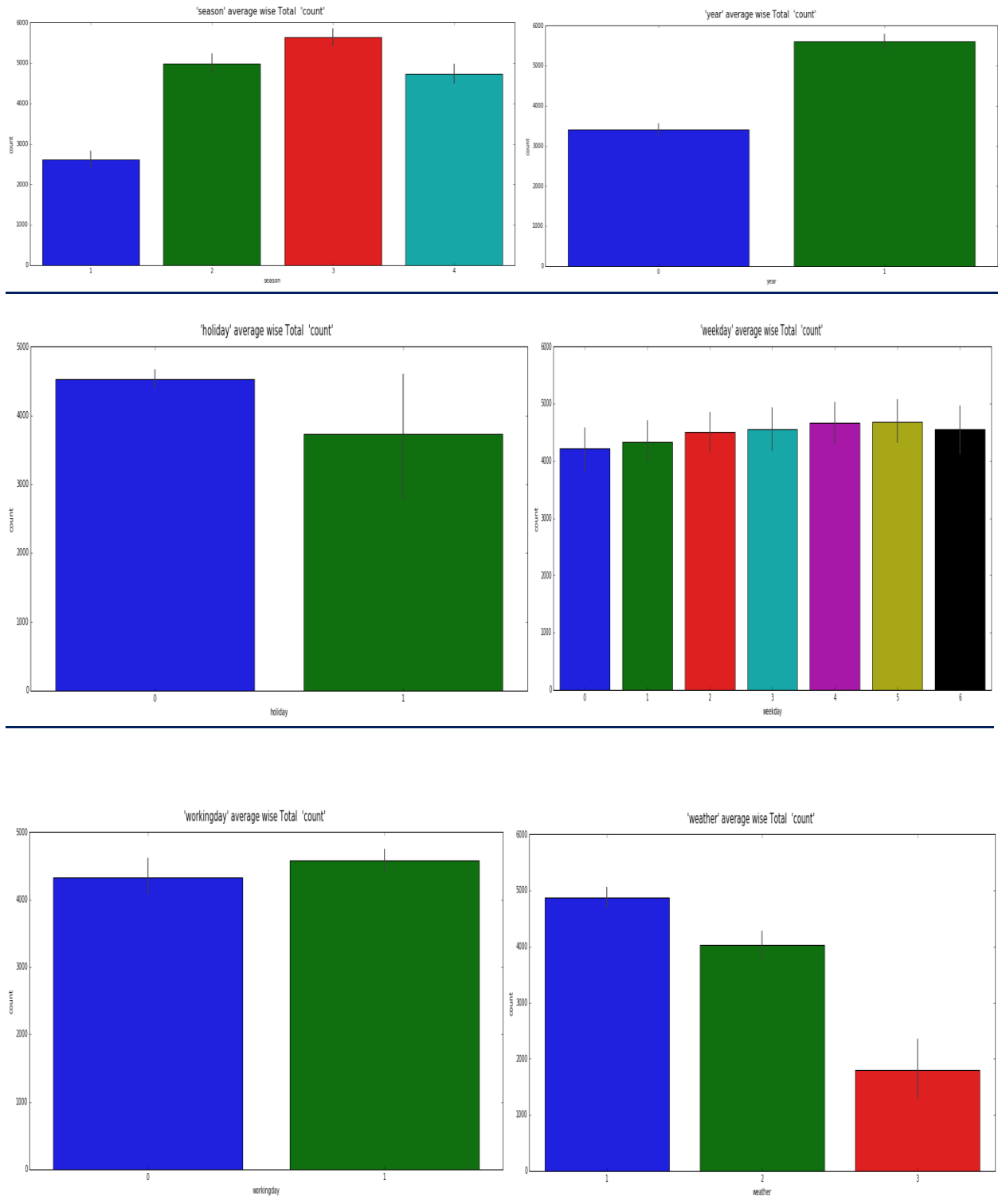
3 Conclusion

After developing all the regression models in both Python and R we have evaluated all the models based on their R^2 - Score, RMSE, MAE and most importantly **MAPE mean absolute percentage error** that is the error rate of the models were concentrated and try to reduce the error rate of the models as much as possible by doing Hyper-Parameter-Tuning & Optimization in both Python and R for the best models and in Python **XGB model** with **MAPE of 12.9 % error rate** which is **least** when compared all the models in python and **MAPE of Random Forest** model in **R** was **17.33 % error rate** after tuning which is performing well.

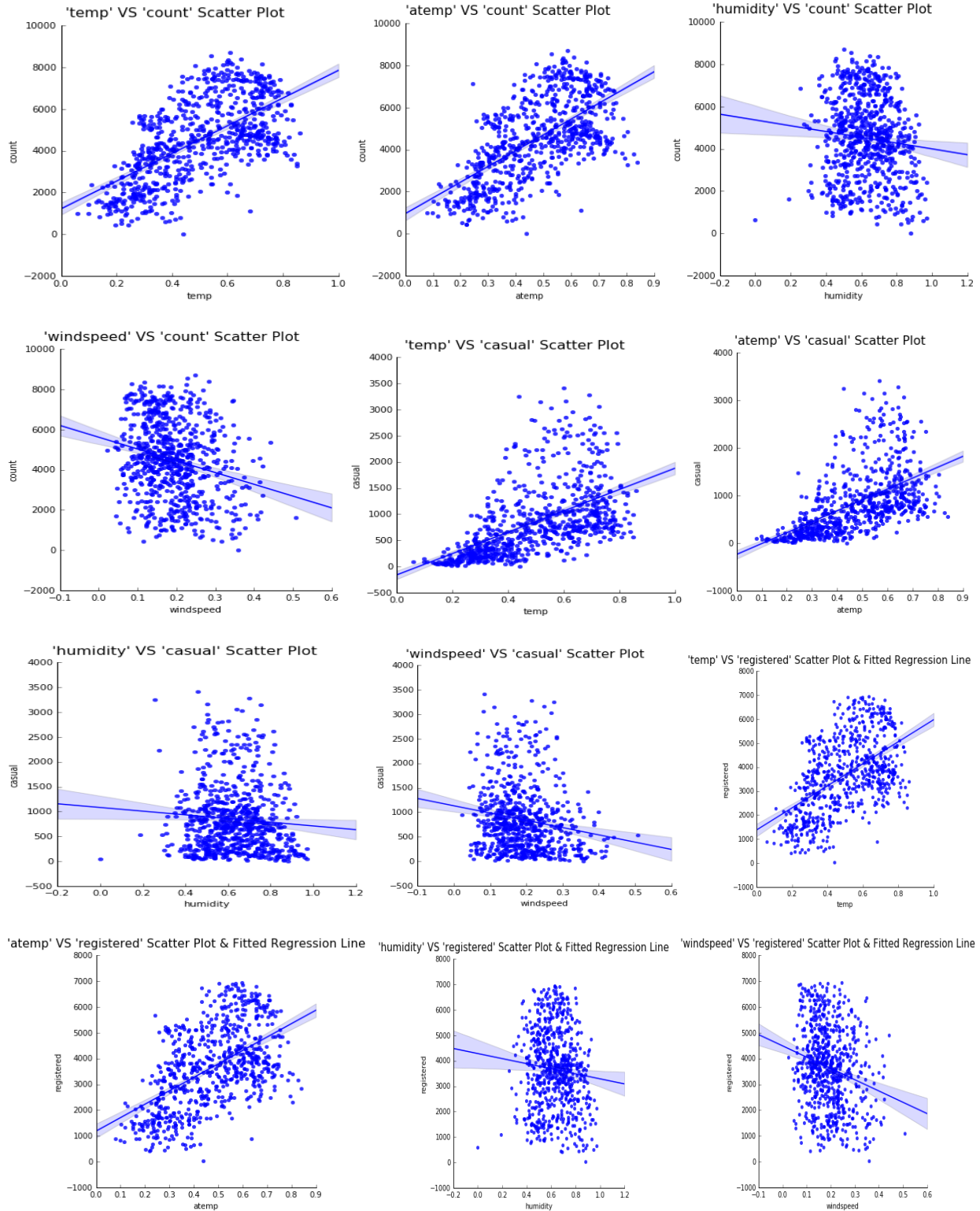
3.1 End Notes

- Whatever steps carried out in Python are almost same in R Also Right from Exploratory Data Analysis (EDA), Data-Pre-processing, Model development, Model Evaluation and Optimization were done in both Python and R.
- The Ultimate goal of the regression models developed here was to reduce the error rate of the models.
- All the regression models were evaluated on both train and test side because in the statistical models unlike like machine learning algorithms we evaluate only on test data here we also validate models on the training side also.

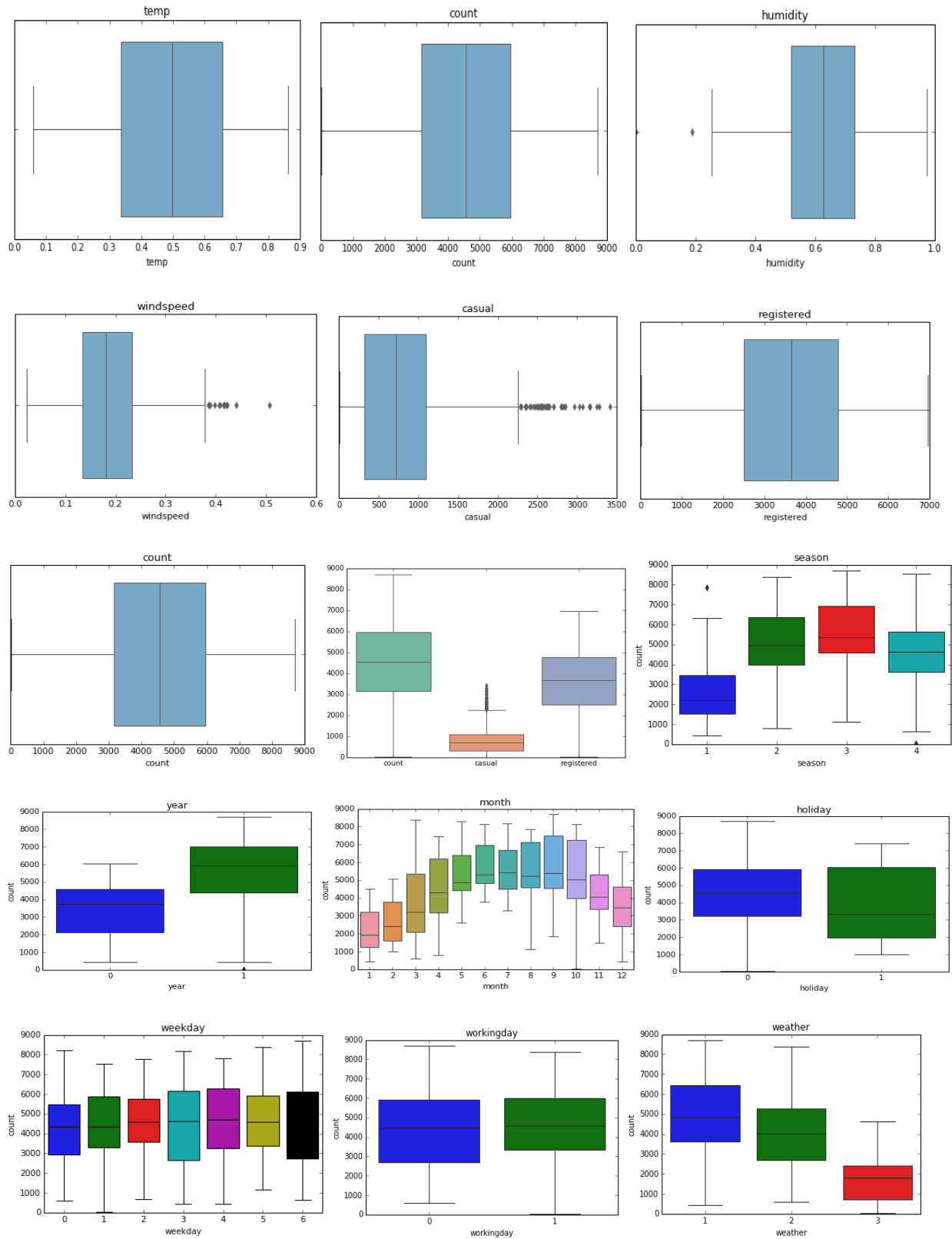
Appendix A: Extra Figures of Python and R



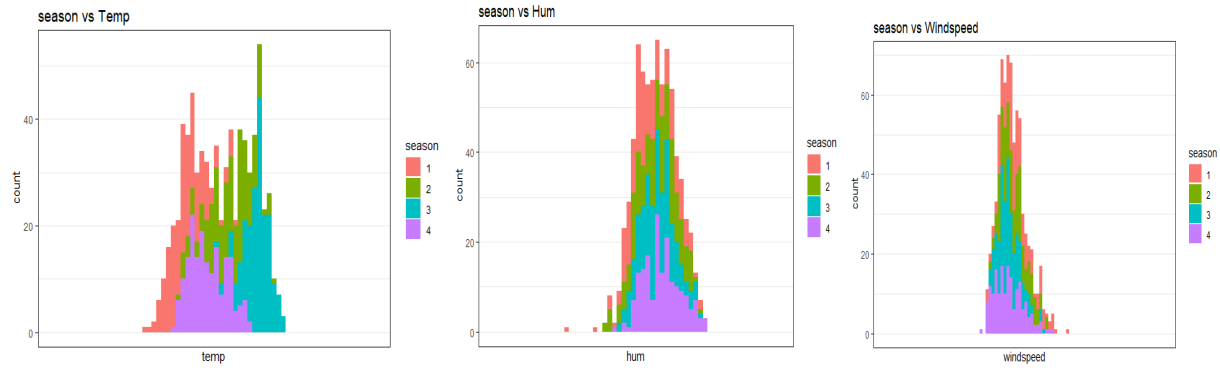
Bar plot Visualization of both numerical and categorical columns



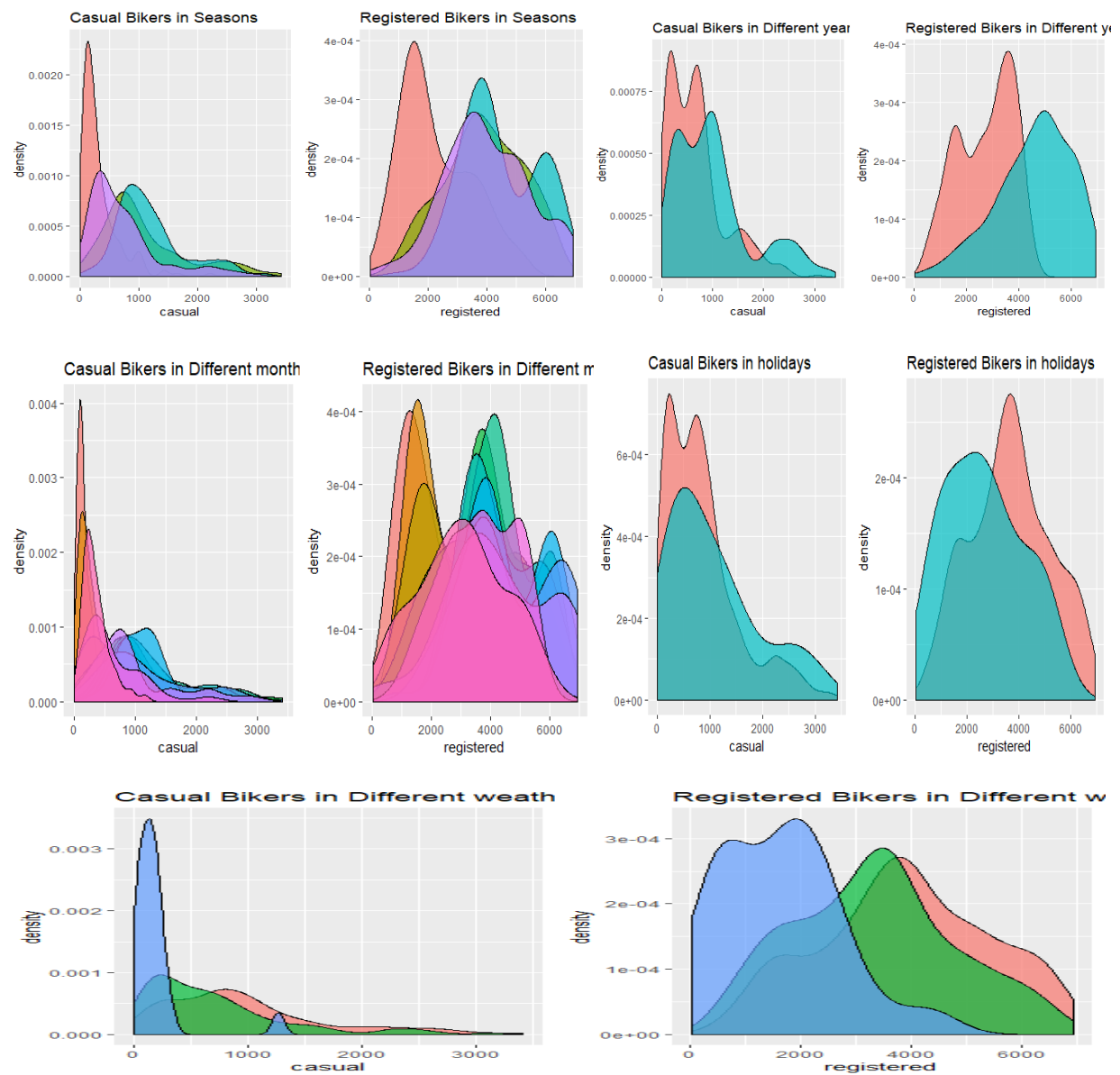
Scatter Plots and Plots with Fitted Line using lm plot



Boxplot for all numeric and categorical features in Python



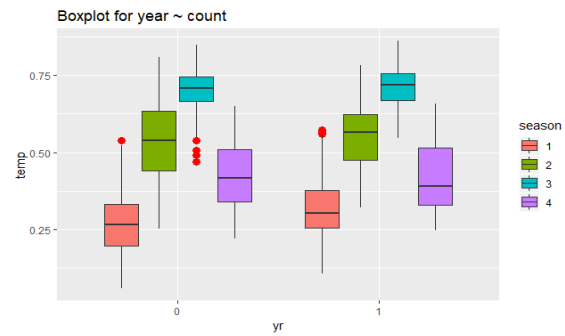
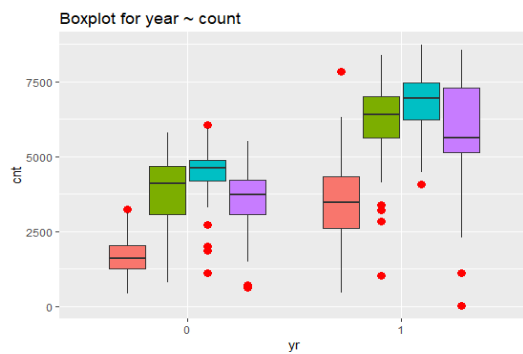
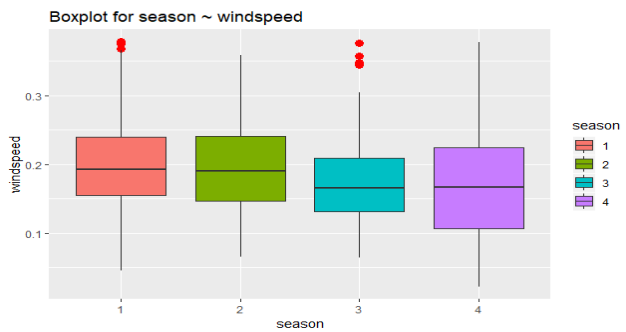
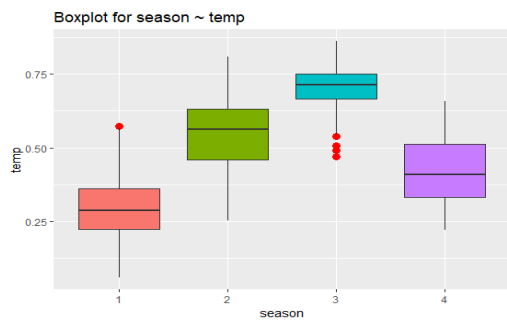
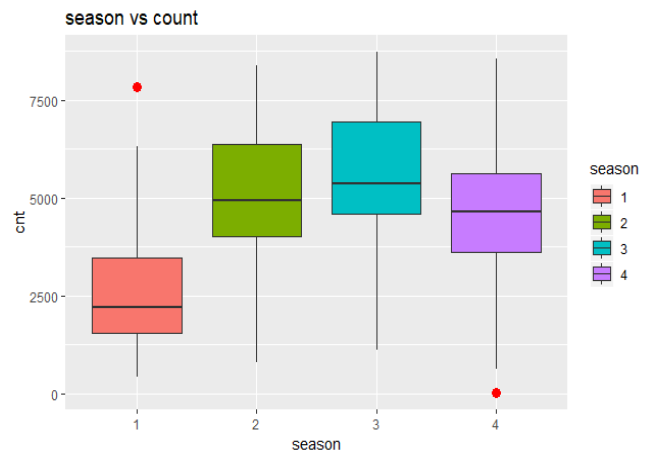
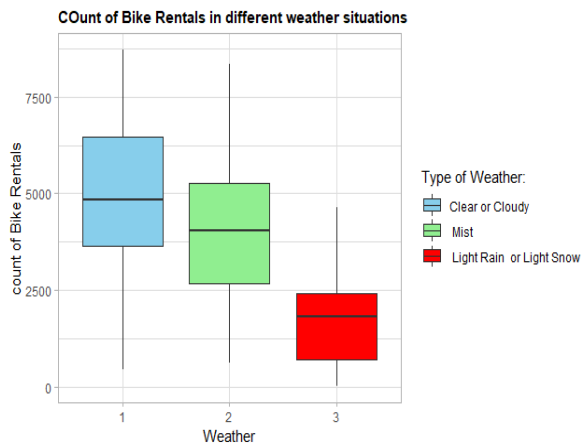
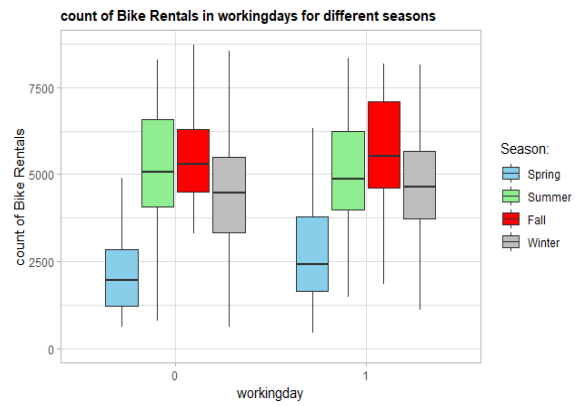
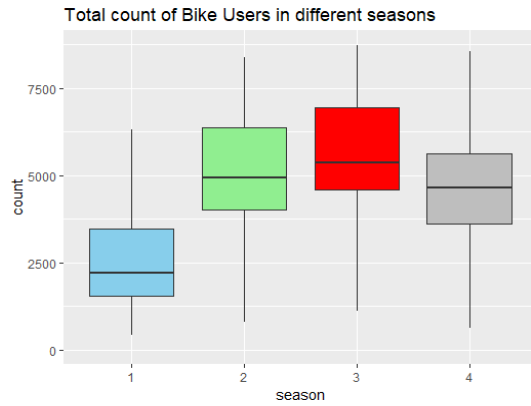
Histogram plot for season ~ temp + hum + windspeed Plots in R



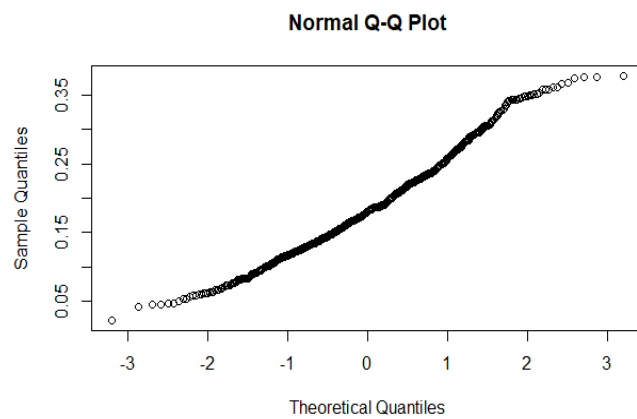
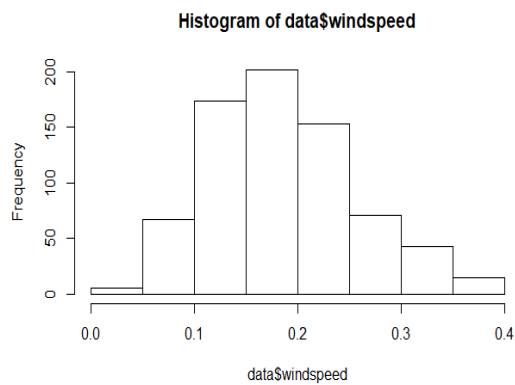
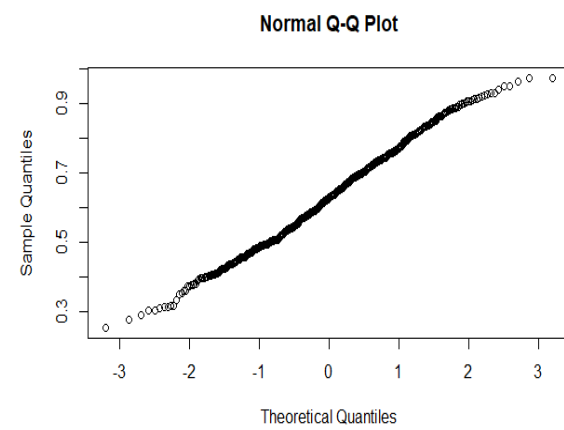
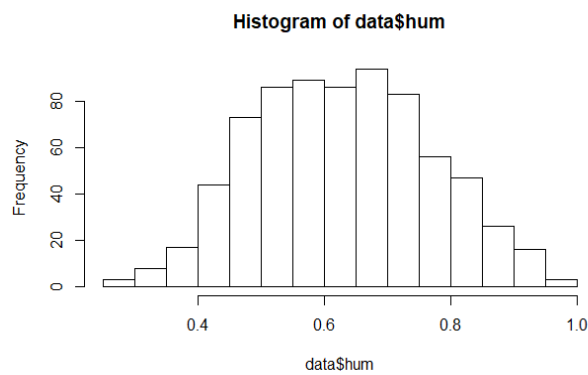
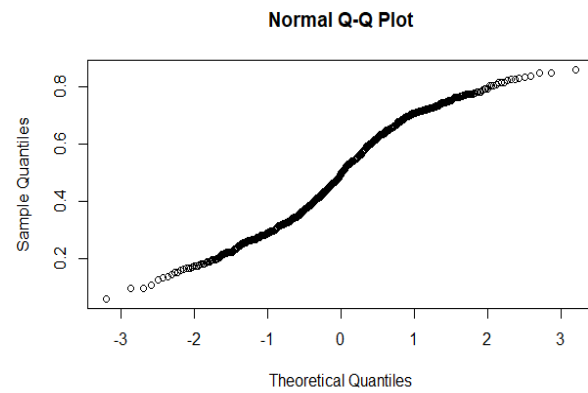
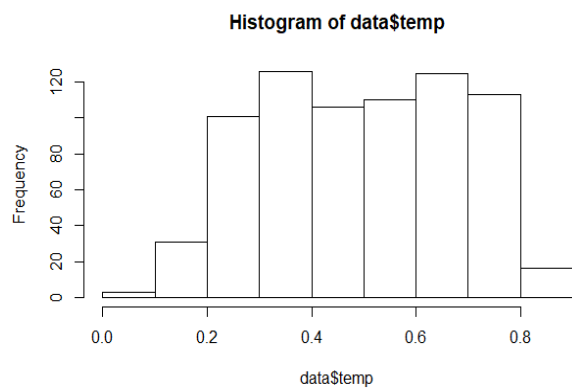
Density Plots for casual and registered biker Plots in R



Scatter Plot for all numerical columns VS cnt variable and bike riders in different weather conditions in R



Boxplot for outliers check in R



Histogram and Normality Check Plots for checking the distribution data in R

Appendix B: Complete Python Code

Importing Standard Libraries

```
# importing libraries for data pre-processing steps
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy.stats import chi2_contingency
%matplotlib inline

# importing libraries for model development, Performance, Evaluation and Optimization
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import warnings
warnings.filterwarnings('ignore')

# Setting up the working directory
os.chdir("E:\DataScienceEdwisor\PythonScripts")

#Checking the current working directory
os.getcwd()
# Loading the dataset which is in '.CSV' format i.e; (Comma-Separated-Values)
Bike_Actual = pd.read_csv('day.csv')

data = Bike_Actual.copy()
```

Exploratory Data Analysis

```
# Lets see few observations of the dataset
data.head()
# Lets Rename Column names with proper names
data = data.rename(columns = {'dteday':'datetime', 'yr':'year', 'mnth':'month', 'weathersit':'weather',
                              'hr':'hour', 'hum':'humidity', 'cnt': 'count'})
# Lets see the dimensions of the dataset
print(data.shape)
# Information about the data and its data-types
data.info()
# Lets see the descriptive statistics of the data
data.describe()
```

```

# Lets Check the Column Names
data.columns
# Lets separate numeric and categorical columns
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
cat_cols = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday', 'weather']
# Lets check the counts in each categorical columns
for i in cat_cols:
    print(i)
    print(data[i].value_counts())
    print("*****")
# Lets Check unique values in the whole data
for i in data.columns:
    print(i, '=====>', len(data[i].value_counts()))

```

Missing value analysis

```

# Lets if there are any missing values
data.isnull().sum()

```

Data Visualization

W.K.T target variable 'count' = 'casual' + 'registered'

```

# Average Count of all Bikes Distribution
for i in cat_cols:
    fig = plt.figure(figsize=(15,6))
    fig = sns.barplot(x=i, y="count", data=data)
    fig.set(xlabel=i, ylabel='count')
    plt.suptitle("'{}X' average wise Total  '{}Y' ".format(X=i,Y='count'), y = 1.02, fontsize=15)
    plt.tight_layout()
    plt.show()

# Count of seasons with labels 1:spring, 2-summer, 3-fall, 4-winter
fig, ax = plt.subplots()
sns.barplot(data=data[['season', 'count']],
            x='season',
            y='count',
            ax=ax, palette = 'Set2')

plt.title('count by Season')
plt.ylabel('count')
plt.xlabel('Season')

tick_val=[0, 1, 2, 3]
tick_lab=['Spring', 'Summer', 'Fall', "winter"]
plt.xticks(tick_val, tick_lab)

plt.show()

```

```

# Count of particular seasons with labels 1: clear, 2: mist, 3: light snow
fig, ax = plt.subplots()
sns.countplot(x= 'weather', hue = 'season' , data=data, palette = 'Set2')

plt.title('count by Weather')
plt.ylabel('count')
plt.xlabel('Weather')

tick_val=[0, 1, 2]
tick_lab=['clear','mist','light snow']
plt.xticks(tick_val, tick_lab)

plt.show()
# Lets check the count of the target variable cnt and registered and casual
fig = plt.subplots(figsize=(7,5))
sns.boxplot(data=data[['count', 'casual', 'registered']], palette='Set2')
# Lets see the distribution of the target variable cnt
data['count'].hist(figsize=(15,10), color='Blue', alpha=0.7)
# Lets see the distribution of the target variable casual
data['casual'].hist(figsize=(15,10), color='Blue', alpha=0.7)
# Lets see the distribution of the target variable registered
data['registered'].hist(figsize=(15,10), color='Blue', alpha=0.7)
# Distribution of weather in all four different seasons
plot= sns.FacetGrid(data=data,
                    col='season',
                    row='weather',hue='season', palette='Set2')
plot.map(plt.hist, 'count')

plt.subplots_adjust(top=1.0)
plot.fig.suptitle('count vs weather')

plot.set_xlabels('count')
plot.set_ylabels('Frequency')

plt.show()
# Scatter Plot For Cheking the Variables Scatter VS Count
for i in ['temp','atemp','humidity','windspeed']:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="count", data=data)
    fig.set(xlabel=i, ylabel='count')
    plt.suptitle("{}{X}' VS '{Y}' Scatter Plot ".format(X=i,Y='count'),y = 1.0
2,fontsize=15)
    plt.tight_layout()
    plt.show()

```

```

# Scatter Plot For Cheking the Variables Scatter VS Casual
for i in ['temp', 'atemp', 'humidity', 'windspeed']:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="casual", data=data)
    fig.set(xlabel=i, ylabel='casual')
    plt.suptitle("'{X}' VS '{Y}' Scatter Plot ".format(X=i,Y='casual'),y = 1.02,fontsize=15)
    plt.tight_layout()
    plt.show()

# Scatter Plot For Cheking the Variables Scatter VS Registered
for i in ['temp', 'atemp', 'humidity', 'windspeed']:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="registered", data=data)
    fig.set(xlabel=i, ylabel='registered')
    plt.suptitle("'{X}' VS '{Y}' Scatter Plot & Fitted Regression Line ".format(X=i,Y='registered'),y = 1.02,fontsize=15)
    plt.tight_layout()
    plt.show()

# Scatter Plot For Cheking the Variables Scatter VS Count
for i in cat_cols:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="count", data=data,fit_reg=False)
    fig.set(xlabel=i, ylabel='count')
    plt.suptitle("'{X}' VS '{Y}' Scatter Plot ".format(X=i,Y='count'),y = 1.02,fontsize=15)
    plt.tight_layout()
    plt.show()

# Scatter Plot For Cheking the Variables Scatter VS Casual
for i in cat_cols:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="casual", data=data,fit_reg=False)
    fig.set(xlabel=i, ylabel='casual')
    plt.suptitle("'{X}' VS '{Y}' Scatter Plot ".format(X=i,Y='casual'),y = 1.02,fontsize=15)
    plt.tight_layout()
    plt.show()

# Scatter Plot For Cheking the Variables Scatter VS Registered
for i in cat_cols:
    fig = plt.figure(figsize=(15,6))
    fig = sns.lmplot(x=i, y="registered", data=data,fit_reg=False)
    fig.set(xlabel=i, ylabel='registered')
    plt.suptitle("'{X}' VS '{Y}' Scatter Plot ".format(X=i,Y='registered'),y = 1.02,fontsize=15)
    plt.tight_layout()

```

```

plt.show()
#Joint plot For all numeric column
for i in num_cols:
    fig = plt.figure(figsize=(10,7))
    fig = sns.jointplot(x=i, y="count", data=data)
    fig.set_axis_labels(xlabel=i,ylabel='count',fontsize=14)
    plt.suptitle("'{X}' VS '{Y}' Joint Plot".format(X=i,Y='count'),y = 1.02,fontsize=15)
    plt.show()
# Jointplots For temp, atemp VS humidity and winsped
plot = sns.jointplot(x = 'temp' , y = 'atemp', data=data)
plot = sns.jointplot(x = 'temp' , y = 'humidity', data=data)
plot = sns.jointplot(x = 'temp' , y = 'windspeed', data=data)
plot = sns.jointplot(x = 'atemp' , y = 'humidity', data=data)
plot = sns.jointplot(x = 'atemp' , y = 'windspeed', data=data)
plot = sns.jointplot(x = 'humidity' , y = 'windspeed', data=data)
plt.subplots_adjust(top=1.0)
plt.show()
# Lets see the Distribution in weather season wise
fig = plt.figure()
fig = sns.countplot(x='weather', hue = 'season', data=data, palette='Set2' )

Outlier Analysis using Boxplot method
# Lets Check Outlier in Numeric Columns
for i in num_cols:
    plt.figure()
    sns.boxplot(data[i],palette="Blues")
    plt.title(i)
    plt.show()
# Lets Check Outlier in Categorical Columns
for i in cat_cols:
    plt.figure()
    sns.boxplot(x=i, y='count', data=data)
    plt.title(i)
    plt.show()

# Here we are not considering casual registered and count features
# Detecting and revoing outliers and then replacing the values with nan
for i in ['temp', 'atemp', 'humidity', 'windspeed']:
    print(i)
    q75, q25 = np.percentile(data.loc[:,i], [75, 25])
    # iqr-Inter Quartile Range
    iqr = q75 - q25
    min = q25 - (iqr * 1.5)
    max = q75 + (iqr * 1.5)

```

```

# Replace the values with np.nan
data.loc[data.loc[:,i] < min,i] = np.nan
data.loc[data.loc[:,i] > max,i] = np.nan
#Calculate missing value
print('{numeric}=====>: {x} Missing_Values'.format(numeric = i, x =
(data.loc[:,i].isnull().sum())))
# Lets Use fillna method to impute values into the missing values using the m
ean method
data.humidity = data.humidity.fillna(data.humidity.mean())
data.windspeed = data.windspeed.fillna(data.windspeed.mean())

```

Feature Selection

Correlation Analysis

Generate the Correlation matrix

```
corr = data[['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']].corr()
```

Ploting using seaborn library

```
sns.heatmap(corr, annot=True, cmap="YlGnBu", linewidths=0.2)
```

```
fig=plt.gcf()
```

```
fig.set_size_inches(20,16)
```

```
plt.title("Heatmap analysis For All Numeric columns")
```

```
plt.show()
```

Lets analyze through pair plot on numeric columns

```
sns.pairplot(data[['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']], palette = "husl")
```

```
plt.tight_layout()
```

```
plt.show()
```

Let's Analyze Chi2-Square Test of Independence for Categorical Variables

Null Hypothesis H0 = Variables are Not Independent

ALternate Hypothesis H1 = Variables are Independent

```
cat_cols = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday', 'weather']
```

Creating all combinational pairs because we have different levels in all categorical columns

```
chi_pairs = [(i,j) for i in cat_cols for j in cat_cols ]
```

Extracting all combinations for chi2-test

```
p_vals = []
```

```
for i in chi_pairs:
```

```
    if i[0] != i[1]:
```

```
        # here Chi2-Square test compares two variables in contingency table
```

```

        chi2, p, dof, ex = chi2_contingency(pd.crosstab(data[i[0]], data[i[1]
]))

    # Lets round the decimal upto 3
    p_vals.append(p.round(3))
else:
    p_vals.append('*')

p_vals = np.array(p_vals).reshape((7,7))
p_vals = pd.DataFrame(p_vals, index=cat_cols, columns=cat_cols)
# Here we are Printing only the P-Value
print(p_vals)

# Here Chi2 - is the observed/actual value
# Here P - is the pearson correlation value i.e: (p<0.05)
# Here DOF( Degrees of Freedom) = (no. of rows-1) - (no. of columns-1)
# Here EX - is the expected value
# Here Conclusion is that if the P-Value is less than (p<0.05) we reject Null
Hypothesis saying that 2 values depend
# on each other else we accept alternate hypothesis saying that these 2 value
s are independent of each other

```

Anova Test

```

# Lets See the Analysis of Variance Between Count Numerical Variable and All
categorical Variables First
aov_test_count = ols('count~season+year+month+holiday+weekday+workingday+weat
her', data=data).fit()
anova_table_count = sm.stats.anova_lm(aov_test_count, typ=1)
anova_table_count
# Lets See the Analysis of Variance Between Casual Numerical Variable and All
categorical Variables First
aov_test_casual = ols('casual~season+year+month+holiday+weekday+workingday+we
ather', data=data).fit()
anova_table_casual = sm.stats.anova_lm(aov_test_casual, typ=1)
anova_table_casual
# Lets See the Analysis of Variance Between Registered Numerical Variable and
All categorical Variables First
aov_test_registered = ols('registered~season+year+month+holiday+weekday+worki
ngday+weather', data=data).fit()
anova_table_registered = sm.stats.anova_lm(aov_test_registered, typ=1)
anova_table_registered

```

Feature Importance

```
# Checking Features Importances
# Lets drop some of the columns
drop_cols = ['instant', 'datetime', 'casual', 'registered', 'count']

from sklearn.ensemble import ExtraTreesRegressor
regressor = ExtraTreesRegressor(n_estimators=300)
X = data.drop(columns=drop_cols)
y = data['count']
regressor.fit(X,y)
feat_imp = pd.DataFrame({'Features':data.drop(columns=drop_cols).columns,
                        'importance':regressor.feature_importances_})
feat_imp.sort_values(by = 'importance', ascending=False).reset_index(drop=True)
e)
```

Multi Collinearity Check

```
# For Numeric Columns
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant
numeric_cols = add_constant(data[['temp', 'atemp', 'humidity', 'windspeed']])
VIF = pd.Series([vif(numeric_cols.values, i) for i in range (numeric_cols.shape[1])],
                index = numeric_cols.columns)
VIF.round(2)

# For Categorical Columns
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant
cat_names = add_constant(data[['season', 'year', 'month', 'holiday', 'workingday',
'weekday', 'weather']])
VIF = pd.Series([vif(cat_names.values, i) for i in range (cat_names.shape[1])],
                index = cat_names.columns)
VIF.round(2)
```

Dimension Reduction

```
# Dropping the fetures highly correlated and not useful features
data.drop(columns=['instant', 'datetime', 'holiday', 'atemp', 'casual', 'registered'], inplace=True)
# Dimesnion of the dataset after dimension reduction
print(data.shape)
```


Creating Dummy Columns

```
# Lets Create Dummy Variables For season and weather columns
season_dv = pd.get_dummies(data['season'], drop_first=True, prefix='season')
data = pd.concat([data, season_dv],axis=1)
data = data.drop(columns = ['season'])
weather_dv = pd.get_dummies(data['weather'], drop_first=True, prefix= 'weather')
data = pd.concat([data, weather_dv],axis=1)
data = data.drop(columns= ['weather'])
# lets See the Dataset
data.head()
```

Feature Scaling

```
#Checking distribution of data pandas inbuilt histogram visualization
names = ['temp', 'humidity', 'windspeed']
data[names].hist(figsize=(15,6), alpha=0.7)
plt.show()
```

Sampling Data Splitting the Data to Train and Test sets

```
X = data.drop(columns=['count'])
y = data['count']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
```

Model Development

```
# Defining function to fit the model and predict the results
from sklearn.metrics import explained_variance_score
def reg_models(regression_models, features, comparison):
    regression_models.fit(features, comparison)
    y_pred = regression_models.predict(X_test)
    KFCV = cross_val_score(estimator=regression_models, X=features, y=comparison, cv=10,
                           scoring = 'explained_variance' )
    KVC_Performance = KFCV.mean()
    print("K-Fold Cross Validation Explained Variance")
    print('<=====>')
    print(KVC_Performance)
    print()
    print("Explained Variance on Train Data")
    print('<=====>')
    print(regression_models.score(features, comparison))
    print()
    print("Explained Variance on Test Data")
    print('<=====>')
    print(regression_models.score(X_test, y_test))
```

```

# Defining Function to Evaluate models on different error metrics
def eval_model(actual_vals, pdict_vals):
    print("Root Mean Squared Error ----->", round(np.sqrt(metrics.mean_squared_error(actual_vals, pdict_vals)), 2))
    print('<=====')
    print(('Mean Absolute Percentage Error-->  {} % ').format(round(np.mean(np.abs((actual_vals - pdict_vals)/actual_vals))*100, 2)))
    print('<=====')
    print("R2 Score ----->", round(metrics.r2_score(actual_vals, pdict_vals), 2))

```

Linear Regression Model

```

# Linear Regression model variance explained
LR_Model = LinearRegression()
reg_models(LR_Model, X_train, y_train)
# Predict new test cases Linear Regression model
LR_Predict = LR_Model.predict(X_test)
# Evaluation of linera Regression Model
eval_model(y_test, LR_Predict)

```

K Nearest Neighbor Model

```

# KNN model variance explained
KNN_Model = KNeighborsRegressor(n_neighbors=5)
reg_models(KNN_Model, X_train, y_train)
# Predict new test case for KNN model
KNN_Predict = KNN_Model.predict(X_test)
# Evaluation of KNN Model
eval_model(y_test, KNN_Predict)

```

Support Vector Regressor Model

```

# SVR model variance explained
SVR_Model = SVR()
reg_models(SVR_Model, X_train, y_train)
# Predict new test case for SVR model
SVR_Predict = SVR_Model.predict(X_test)
# Evaluation of SVR Model
eval_model(y_test, SVR_Predict)

```

Decision Tree Model

```

# Decision Tree model variance explained
DT_Model = DecisionTreeRegressor(max_depth=2, random_state=100)
reg_models(DT_Model, X_train, y_train)
# Predict new test case for Decision Tree model
DT_Predict = DT_Model.predict(X_test)
# Evaluation of Decision Tree Model

```

```
eval_model(y_test, DT_Predict)
```

Random Forest Model

```
# Random Forest Tree model variance explained
RF_Model = RandomForestRegressor(n_estimators=500, random_state=100)
reg_models(RF_Model, X_train, y_train)
# Predict new test case for Random Forest model
RF_Predict = RF_Model.predict(X_test)
# Evaluation of Random Forest Model
eval_model(y_test, RF_Predict)
```

XGB Regressor Model

```
# Xtreme Gradient Boosting Regressor model variance explained
XGB_Model = XGBRegressor()
reg_models(XGB_Model, X_train, y_train)
# Predict new test case for XGB model
XGB_Predict = XGB_Model.predict(X_test)
# Evaluation of XGB Model
eval_model(y_test, XGB_Predict)
```

Hyper Parameter Tuning & Optimization Lets Tune Both Random Forest Model and XGB Model and Then we will finalize the Outperformed Model from these both models

```
# Hyper Paramter Tuning XGB Model to Find Optimum parameters
xgb_model = XGBRegressor()

params = [{'n_estimators':[250,350,450,550], 'max_depth':[2,3,5,7],
          'learning_rate':[0.01, 0.045, 0.05, 0.055, 0.1], 'gamma':[0, 0.001
, 0.01, 0.03, 0.05],
          'subsample':[1, 0.5, 0.7, 0.8, 0.9]}]

grid_search = GridSearchCV(estimator=xgb_model, param_grid=params, cv = 5,
                           scoring = 'explained_variance', n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)
print('Best Score ==>', grid_search.best_score_)
print('Best Params ==>', grid_search.best_params_)

# Best Score ==> 0.897321837387
# Best Params ==> {'gamma': 0, 'learning_rate': 0.045, 'max_depth': 3, 'n_estimators': 550, 'subsample': 0.5}
# Tuned Xtreme Gradient Boosting Regressor model variance explained

xgb_model = XGBRegressor(learning_rate=0.045, max_depth=3, n_estimators=550,
                        gamma = 0, subsample = 0.5)
```

```

reg_models(xgb_model, X_train, y_train)
# Predict new test case for Tuned XGB model
xgb_predict = xgb_model.predict(X_test)
# Evaluation of Tuned XGB Model
eval_model(y_test, xgb_predict)

```

Tuned Random Forest Model

```

# Hyper Paramter Tuning Random Forest Model to Find Optimum parameters
rf_model = RandomForestRegressor(random_state=1)
params = [{'n_estimators': [200, 300, 400, 500, 600, 700, 800, 1000], 'max_features': [
'auto', 'sqrt', 'log2'],
          'min_samples_split': [2, 4, 6], 'max_depth': [2, 4, 6, 8, 10, 12, 14, 16], 'min
_samples_leaf': [2, 3, 5]}]

grid_search = GridSearchCV(estimator = rf_model, param_grid = params, cv = 5,
scoring = 'explained_variance', n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)
print('Best Score ==>', grid_search.best_score_)
print('Best Params ==>', grid_search.best_params_)

# Best Score ==> 0.865267759445
# Best Params ==> {'max_depth': 12, 'max_features': 'sqrt', 'min_samples_lea
f': 2, 'min_samples_split': 2, 'n_estimators': 800}
# Tuned Random Forest model variance explained
rf_model = RandomForestRegressor(n_estimators=800, max_depth=12, max_features
='sqrt', min_samples_leaf=2,
                                min_samples_split=2)

reg_models(rf_model, X_train, y_train)
# Predict new test case for Random Forest model
rf_predict = rf_model.predict(X_test)
# Evaluation of Random Forest Model
eval_model(y_test, rf_predict)

```

```

# Lets Plot Scatter Graph for predicted values for tuned XGBRegressor Model
XGB_Regressor = XGBRegressor(random_state=1, learning_rate=0.045, max_depth=3
, n_estimators=550,
                            gamma = 0, subsample=0.5)
XGB_Regressor.fit(X_train, y_train)
y_pred = XGB_Regressor.predict(X_test)
# Plotting Scatter plot for Tuned XGB Regressor for Actual and Predicted valu
es
fig , ax = plt.subplots(figsize=(7,5))

```

```
ax.scatter(y_test, y_pred)
ax.plot([0,8000],[0,8000], 'r--', label = 'Perfect Prediction')
ax.legend()
plt.title("Scatter graph for Actual and Predicted")
plt.xlabel('y_true')
plt.ylabel('y_pred')
plt.tight_layout()
plt.show()

# Saving results back to hard disk
# Setting up the working directory
#os.chdir("E:\DataScienceEdwisor\PROJECT-2\Python")

#data.to_csv("final_day.csv", index = False)
```

Appendix C: Complete R Code Link

<https://drive.google.com/open?id=1MtxuHtg5fn8ZbjDQI-Obyah8VviOSsm2>