PREDICTION OF BIKE RENTAL COUNT

REGRESSION PROBLEM

ABSTRACT

The idea of this project is to combine historical usage patterns with weather data in order to predict bike rental demand and answers a question of "How many bikes will meet users' demand in a future certain time"

NIRAV THANKI

Table of Contents:

1 Introduction

This project focus on the bike share program's rebalancing issue, aiming to answer a question: "How many bikes will meet users' demand in a future certain time." The aim of this project is to build mathematical models to predict bike rental demand by combining historical usage patterns with the related information of users, weather, holiday and weekend.

1.1 Problem Statement

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

We are provided with dataset of bike rental based on environmental conditions and seasons for two years i.e. 2011 and 2012. We need to predict what could be the counting of bike rental for a given season, month and year with environmental condition.

1.2 The Data

My Dataset contain total 16 variables and 731 observations, out of those 16 Variables 13 are my Predictor or Independent Variables and "**cnt**" is my Target or Dependent Variable.

Snap shot of data set:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801

My Target Variable "**cnt**" is accumulation of "**casual**" & "**registered**". It means that I cannot use '**casual**' & '**registered**' in Modeling because they will not be present in my Test Data. But They provide some additional information.

I have feature which contain time series information as well environment condition of that particular day.

Description of Dataset:

No.	Features Name	Data Type		Feature Type
		Python	R	
1	instant	int64	integer	Predictor
2	dteday	object	Factor	Predictor
3	season	int64	integer	Predictor
4	yr	int64	integer	Predictor

5	mnth	int64	integer	Predictor
6	holiday	int64	integer	Predictor
7	weekday	int64	integer	Predictor
8	workingday	int64	integer	Predictor
9	weathersit	int64	integer	Predictor
10	temp	float64	numeric	Predictor
11	atemp	float64	numeric	Predictor
12	hum	float64	numeric	Predictor
13	windspeed	float64	numeric	Predictor
14	casual	int64	integer	
15	registered	int64	integer	
16	cnt	int64	integer	Target

Python code of above result

R code of above result

Narration of each variable:

Feature Name	Narration
Instant	Record Index
dteday	Date
season	1 = spring, 2 = summer, 3 = fall, 4 = winter
yr	0 = 2011, 1 = 2012
mnth	Month of the year (1 to 12)
holiday	Whether the day is considered a holiday
weekday	Day of the Week (0 to 6)
workingday	Whether the day is neither a weekend nor holiday
weathersit	1: Clear, Few clouds, Partly cloudy
	2: Mist + Cloudy, Mist + Broken clouds
	3: Light Snow, Light Rain, Thundersome
	4: Heavy Rain + Ice Pallets
temp	Normalized temperature in Celsius. The values are derived via: (t-t_min)/(t_max – t_min) Where, t_min = -8, t_max = +39
atemp	Normalized feeling temperature in Celsius. The values are derived via: (t-t_min)/(t_max – t_min) Where, t_min = -16, t_max = +50
hum	Normalize humidity, the value is divided by 100 (max)
windspeed	Normalize wind speed, the value is divided by 67 (max)
casual	Number of non-registered user rentals initiated
registered	Number of registered user rentals initiated
cnt	Number of total rentals

Before starting Exploratory data analysis, I have removed two columns ‘instant’ & ‘dteday’ because ‘instant’ is just index. And other variables (‘yr’, ‘mnth’, ‘weekday’) contain all the insight which are present in ‘dteday’

Second thing, which I did is rename below mention columns for better understanding.

Old Name	New Name
yr	year
mnth	month
weathersit	weather
hum	humidity
cnt	count

1.3 Hypothesis Generation:

Before exploring the data to understand the relationship between variables, I generate some hypothesis.

- **Daily Trend:** Registered users demand more bike on weekdays as compared to weekend or holiday. Because people who registered are regular users and I can imagine they are mostly service man. So, during weekday demand or registered users are high

- **Rain:** The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

- **Temperature:** In India, temperature has negative correlation with bike demand. But, after looking in this dataset's temperature graph, I presume it may have positive correlation.

- **Traffic:** It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc.

2 Data Pre-processing

Before feeding data set into a machine learning application, we must ensure our data is accurate, consistent and useful enough for the model to learn from. Because data can be gathered from any number of sources, and with that comes the possibility that our data isn't complete or fully accurate. To ensure our data is high quality, and therefore useful, it needs to be pre-processed before being used in a model. Otherwise, we'll be following the practice of putting "garbage in"

Partition of Variables into Categorical & Numerical:

Categorical Variables:

Any data attribute which is categorical in nature represents discrete values which belong to a specific finite set of categories or classes. These discrete values can be text or numeric in nature. There are two major classes of categorical data, nominal and ordinal.

In any nominal categorical data attribute, there is no concept of ordering amongst the values of that attribute. Music and video game genres, country names, food and cuisine types are other examples of nominal categorical attributes.

In our dataset we I consider below mention variables as a nominal category variable because they contain unique classes without any concept of notion or order.

- **season:** contain 4 unique class
- **weather:** contain 3 unique class
- **year:** contain 2 unique class
- **workingday:** contain 2 unique class
- **holiday:** contain 2 unique class
- **month:** Now question arise should I consider Month as a nominal categorical variables or not? Because. There is clear evidence for month-of-year dependence. count high during April to November, and low during January to March and December. Second thing count is not same Jan – 2011 & Jan – 2012. But trend is same in both years. So I will make bins for month.
- **weekday:** it's same like month, So I will make bins for weekday.

Numerical Variables:

Below mention 4 independent variables have continuous value in normalize format so, they are my Numerical predictor variables.

1) temp
2) atemp
3) humidity
4) windspeed

Our main target variable 'count' has continues value so our problem is regression.

Let's check Statistic summary of Numerical variables.

Feature Name	temp	atemp	humidity	windspeed	casual	registered	count
count	731	731	731	731	731	731	731
mean	0.4954	0.4744	0.6279	0.1905	848.1765	3656.1724	4504.349
std	0.1831	0.163	0.1424	0.0775	686.6225	1560.2564	1937.212
min	0.0591	0.0791	0	0.0224	2	20	22
25%	0.3371	0.3378	0.52	0.1349	315.5	2497	3152
50%	0.4983	0.4867	0.6267	0.181	713	3662	4548
75%	0.6554	0.6086	0.7302	0.2332	1096	4776.5	5956
max	0.8617	0.8409	0.9725	0.5075	3410	6946	8714

Python code of above result

R code of above result

Analysis:

We have four independent numerical variables and three dependent numerical variables (in light green shade)

All independent variables are in range between 0 to 1 it means they are in normalize form. And our main target variable 'count' range between 22 to 8714.

Let's check value counts of Categorical variables.

Feature Name	Unique Value
season	4
year	2
month	12
holiday	2
weekday	7
working day	2
weather	3

Python code of above result

R code of above result

Count of each unique value in each category.

Unique Value	season	year	month	holiday	weekday	working day	weather
0	-	366	-	710	105	231	-
1	181	365	62	21	105	500	463
2	184	-	57	-	104	-	247
3	188	-	62	-	104	-	21
4	178	-	60	-	104	-	-
5	-	-	62	-	104	-	-
6	-	-	60	-	105	-	-

7	-	-	62	-	-	-	-
8	-	-	62	-	-	-	-
9	-	-	60	-	-	-	-
10	-	-	62	-	-	-	-
11	-	-	60	-	-	-	-
12	-	-	62	-	-	-	-

Python code of above result

R code of above result

Analysis:

We have main 7 independent categorical variables.

As per above table I can say that ‘season’, ‘year’, ‘month’ and ‘weekday’ is balanced columns. But ‘holiday’ and ‘workingday’ is highly imbalanced column.

Our column weather contains 3 unique class. Count of value 3 (light rain) is 21 which is comparatively very low than other two class 1 & 2.

2.1 Missing Value Analysis:

Let’s check our data set contain missing value or not?

Feature Name	No. of Missing Value
instant	0
dteday	0
season	0
year	0
month	0
holiday	0
weekday	0
working day	0
weather	0
temp	0
atemp	0
humidity	0
windspeed	0
casual	0
registered	0
count	0

Python code of above result

R code of above result

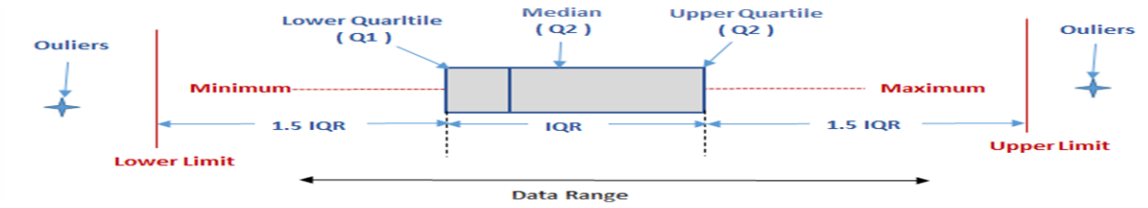
Above table depict that our data set has No missing value.

2.2 Outlier Analysis:

One of the simplest methods for detecting outliers is the use of box plots. A box plot is a graphical display for describing the distributions of the data. Box plots use the median and the lower and upper quartiles.

- The Median (Q2) is the middle value of the data set.
- The Lower quartile (Q1) is the median of the lower half of the data set
- The Upper quartile (Q3) is the median of the upper half of the data set.
- The Interquartile range (IQR) = Q3 – Q1
- Lower Limit = Q1 – 1.5 IQR.
- Upper Limit = Q3 + 1.5 IQR

So any value that will be more than the upper limit or lesser than the lower limit will be the outliers.

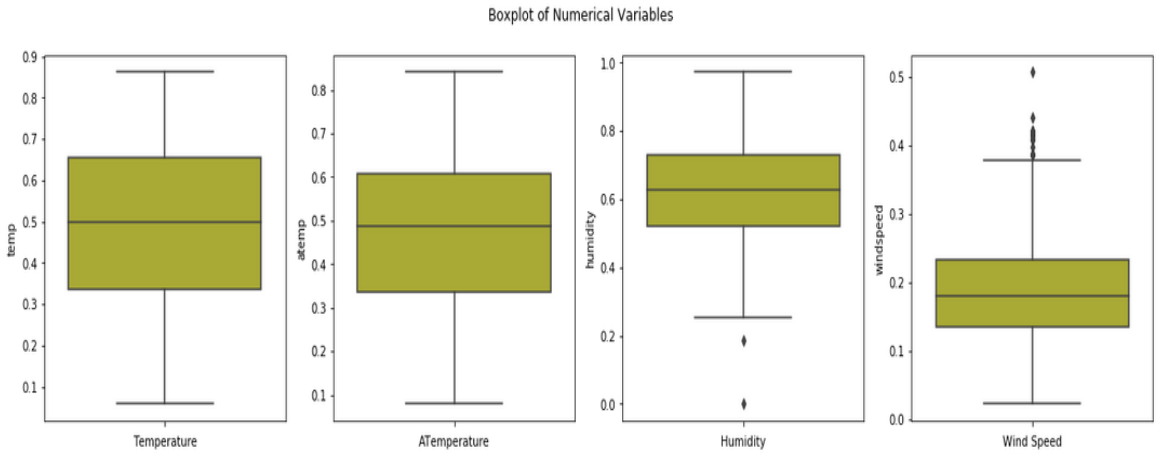


Before removing outlier from data set we need to know why does they occurs? Below are main reasons for outlier occurrence.

Data entry error, Measurement error, Experimental error, Intentional error, Data processing error, Sampling error, and Natural outlier.

With the help of Box plot let's check is there any outlier present in our numerical independent variable or not?

Boxplot of Numerical Variables



Python code of above result

R code of above result

Analysis:

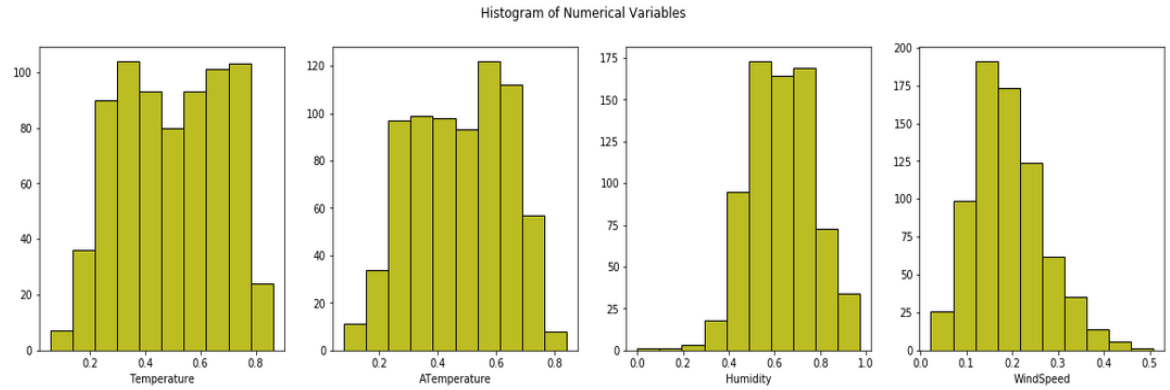
Above image depict that there is no outlier present in 'temp' and 'atemp' columns.

Humidity and wind speed have fewer outliers. It could be either natural outlier or due to above mention reasons that's why I build Multiple Linear Regression model without removing those outliers and after removing those outliers because Linear Regression is highly sensitive to outliers. And found that after removing outliers from 'humidity' & 'windspeed' performance of Multiple Linear Regression doesn't improve. It means that those outlier is only additional information.

Distribution of Numerical Variables:

Let's make histogram of Numerical Variables to check distribution

Histogram of Numerical Variables



Python code of above result

R code of above result

Let's Check Skewness of Numerical Variables.

Feature Name	Skewness
temp	-0.05
atemp	-0.13
humidity	-0.07
wind speed	0.68

Python code of above result

R code of above result

Analysis:

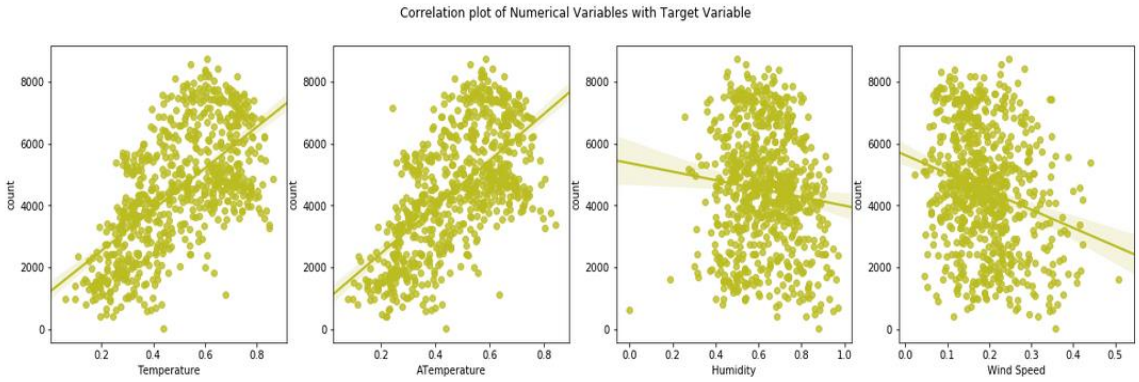
Skewness is usually described as a measure of a dataset's symmetry – or lack of symmetry. A perfectly symmetrical data set will have a skewness of 0.

The rule of thumb for skewness:

- If the skewness is between -0.5 and 0.5, the data are fairly symmetrical
- If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data are moderately skewed
- If the skewness is less than -1 or greater than 1, the data are highly skewed

As per above rules our three variables ‘temp’, ‘atemp’ & ‘humidity’ is fairly symmetrical and ‘windspeed’ is moderately skewed which will not much effect our model.

Scatter Plot of Numerical Variables with our Target Variable.



Python code of above result

R code of above result

Let’s Check Correlation of Numerical Variables

Feature Name	Correlation
temp	0.63
atemp	0.63
humidity	-0.1
wind speed	-0.23

Python code of above result

R code of above result

Analysis:

Above image depict that our columns ‘temp’ & ‘atemp’ is highly positively correlated with our target variable. So, they will play significant role in modelling.

‘humidity’ and ‘windspeed’ is slightly negatively correlated with our target variable.

2.3 Feature Engineering:

Feature engineering is the process of creating new features – predictor variables out of an existing dataset. Because a machine learning model can only learn from given features and properly constructing features will determine how well our model performs.

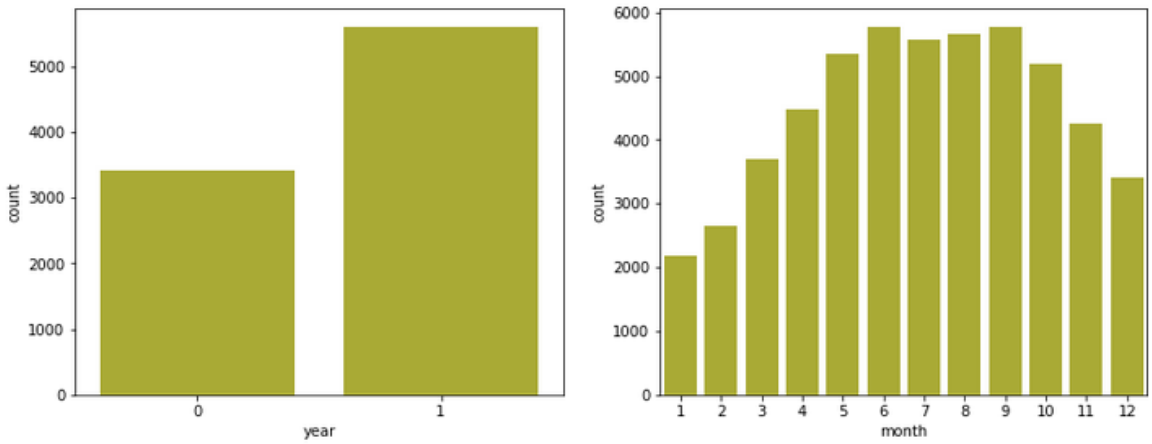
Create new variable ‘month_bin’

As we know column month is categorical, If I directly feed ‘month’ in machine learning model, it would consider it to be a continuous numeric feature thinking value 7 (July - Month) is greater than value 6 (June - Month) but that is meaningless because July month is not bigger or smaller than June

month. And if I consider each month separate category with the help of one hot encoding than my model will suffer curse of dimensionality.

Let's make bar-plot to check mean count of each month and year

Bar plot of Year & Month



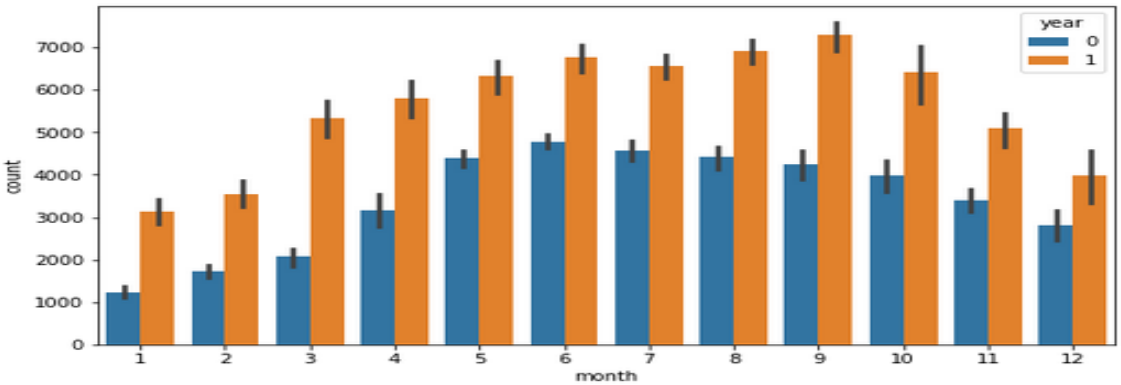
Python code of above result

R code of above result

Above image depict that mean count of year (0) 2011 is 3500 and year (1) 2012 is greater than 5500. And between April to November mean count is higher as compare to others.

Above distribution of month is consolidate for both year 2011 and 2012. Let's check distribution of month with respect to year.

Bar plot of Month with respect of year



Python code of above result

R code of above result

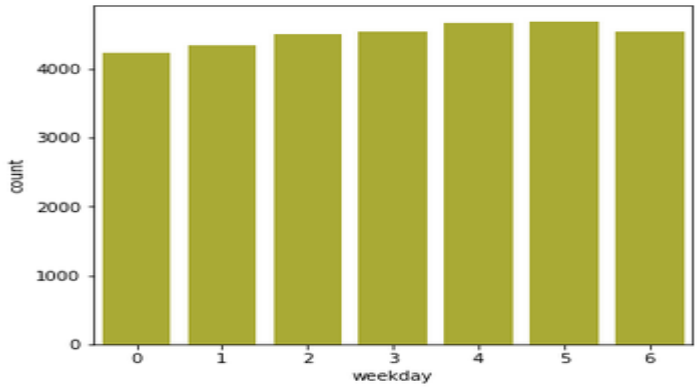
Above image depict trend is similar. Count is higher between April to November as compare to rest. But count of April - 2011 is not same as April - 2012.

Hence I will label month in two class, '1' for April to November & '0' for the rest. As April to November has higher value in both year 2011 and 2012

Create new variable 'weekday_bin'

Let's make bar-plot to check mean count of each day.

Bar plot of weekday

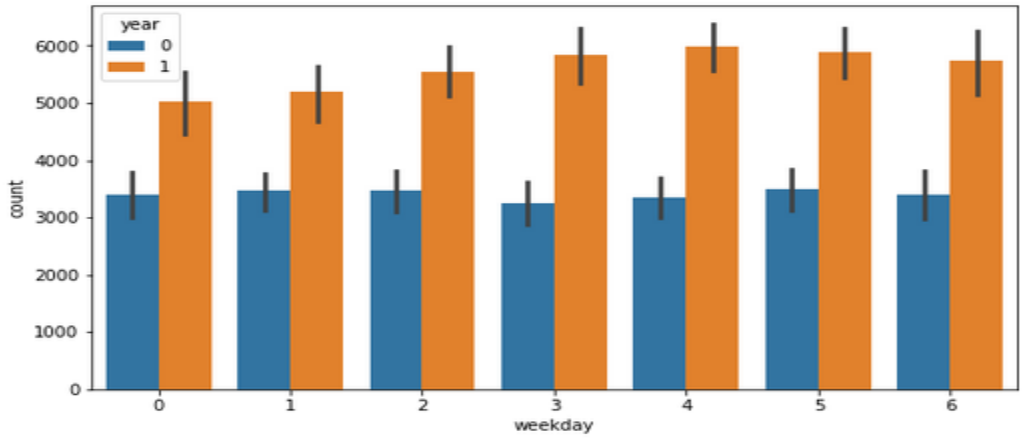


Python code of above result

R code of above result

Above distribution of weekday is consolidate for both year 2011 and 2012. Let's check distribution of month with respect to year.

Bar plot of weekday with respect of year



Python code of above result

R code of above result

Above image depict there is trend for weekday & weekend. hence I will label weekday in two class, '1' for weekday and '0' for weekend.

One hot encoding for 'season' & 'weather'

R deals with categories using a variable type called "**factor**" which elegantly fits all models without an issue and Python would require you to convert the categories into label encoding for the models to work.

Our both variables ‘season’ and ‘weather’ are integer encoded. using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results in python.

In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for n - 1 unique integer value.

If I create dummy variable for each unique integer value than it would be ‘dummy variable trap’, because it will cause the regression to fail. There will be one too many parameters to estimate when an intercept is also included. The general rule is to use one fewer dummy variables than categories.

2.4 Feature Selection:

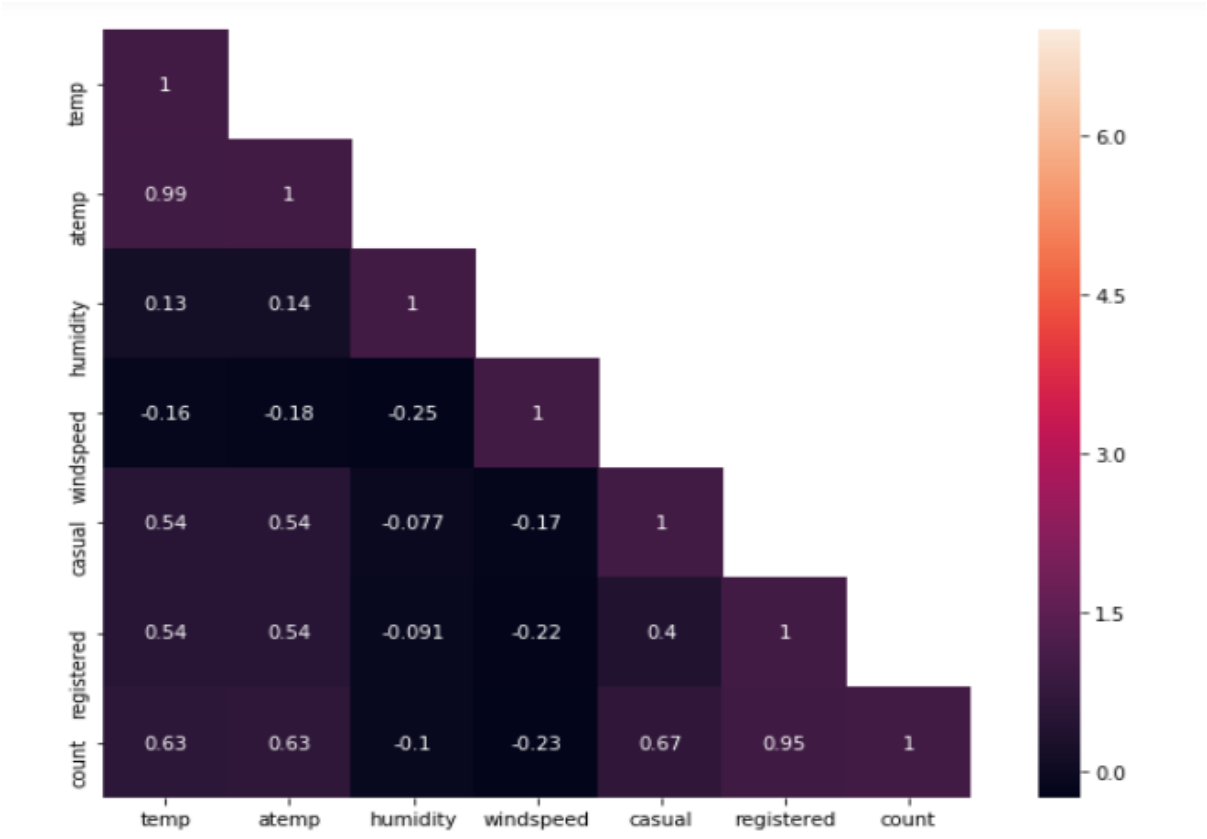
Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of our model. The data features that we use to train our machine learning models have a huge influence on the performance we can achieve.

Irrelevant or partially relevant features can negatively impact model performance. Benefit of feature selection are Reduce overfitting, Improve Accuracy and Reduce training time.

After data cleaning, Feature selection should be the most important step of our model building.

Analyse Numerical Variables.

Let’s check heat-map of Numerical variables:



Python code of above result

R code of above result

Above heat-map help to check how the features are correlated to each other or the target variable.

Collinearity occurs because our Independent variable 'temp' and 'atemp' are highly correlated with each other. This correlation is problem because independent variable should be independent. If the degree of correlation between variables is high enough it can cause problems when I fit the model and interpret results.

There is very simple test to assess Collinearity in regression model is Variance Inflection Factor (VIF).

The rule of thumb to interpret VIF is below.

- VIF is 1 means that not correlated
- VIF between 1 to 5 means that moderately correlated
- VIF greater than 5 means that highly correlated

Let's check VIF score with 'atemp' and without 'atemp'

Feature Name	VIF Score with 'atemp'	VIF Score without 'atemp'
temp	62.96	1.03
atemp	63.63	-
humidity	1.07	1.07
windspeed	1.12	1.08

Python code of above result

R code of above result

As per above VIF score it's better to remove 'atemp' for modelling.

Analyse Categorical Variable

We have 7 predictor categorical variables. For checking correlation between categorical I did Chi-Square test of independent.

Chi-Square test of independent:

Chi-Square test of independent used to check whether or not two variable independent.

- Null Hypothesis (H0): The two variable are independent.
- Alternative Hypothesis (H1): The two variable are not independent. (They are Dependent)

P-value determine whether to reject or fail to reject Null Hypothesis (H0)

Here, I assume significant level $\alpha = 0.05$

The rule of thumb for P-Value:

- P-value ≤ 0.05 reject Null Hypothesis (H0)
- P-value > 0.05 fail to reject Null Hypothesis (H0)

For checking P-value for each categorical variable I made below matrix.

P-Value matrix:

	season	year	holiday	workingday	weather	month_bin	weekday_bin
season	NA						
year	0.99	NA					
holiday	0.68	0.99	NA				
workingday	0.88	0.97	**0.00**	NA			
weather	**0.02**	0.12	0.60	0.25	NA		
month_bin	**0.00**	0.97	0.35	0.65	0.36	NA	
weekday_bin	0.98	0.95	**0.00**	**0.00**	0.22	0.97	NA

Python code of above result

R code of above result

With the help of above table, we can see at some place P-value are less than 0.05 (highlighted in green)

- Working-day is correlated with weekday-bin
- Season is correlated with weather & month-bin
- Holiday is correlated with working-day & weekday-bin

As per assumption independent variable should be independent. So if we want to remove multi-collinearity effect, we need to drop them.

I tried to drop multi-collinear columns one by one and made model but my RMSLE score getting increase in each attempt which is not acceptable. It means that those variables are caring some useful information. but when I removed 'holiday' column it doesn't affect much to my RMSLE score. Still I can't decide is it appropriate to remove 'holiday' column or not? Hence with the help of ensemble machine learning algorithm "Random Forest" I checked Feature Importance.

Result of Random Forest Feature Importance is below.

Python	
Feature Name	**Importance**
temp	0.36266
year	0.290088
atemp	0.118819
season	0.074201
humidity	0.069487
windspeed	0.035943
weather	0.01992
month_bin	0.014833
workingday	0.005719
weekday_bin	0.004926
holiday	0.003404

R	
Feature Name	**Importance**
year	101.10685
season	28.184595
temp	23.352791
weather	23.10883
humidity	22.916328
atemp	21.647484
windspeed	14.944067
month_bin	12.845756
weekday_bin	6.507573
workingday	5.950275
holiday	1.27954

Python code of above result

R code of above result

Above result of Random Forest Feature Importance also provide evidence that column ‘holiday’ is not contributing much information in model building.

Hence finally I drop column ‘holiday’ for model building.

2.5 Feature Scaling:

Feature scaling is a method used to standardize the range of independent Numerical variables of data.

The Rule of thumb for Feature Scaling:

Any algorithm that computes distance or assumes normality, scale our features!!

Here, we have main 4 Numeric variables ‘temp’, ‘atemp’, ‘humidity’ & ‘wind-speed’ which are already in normalized form range between 0 to 1.

So, fortunately we need not to do ‘Feature Scaling’

2.6 Dataset after EDA

After Exploratory data analysis my dataset is ready for modelling.

Let’s check view of data set.

I create two data set one is without outliers and another is with outlier as per below

1) train (731 observations – With outliers)
2) train_WO (717 observations – Without outliers)

Feature Name	Feature Type	
	Python (13)	R (10)
year	int64	integer
workingday	int64	integer
temp	float64	numeric
humidity	float64	numeric
windspeed	float64	numeric
month_bin	int64	numeric
weekday_bin	int64	numeric
season	-	factor
weather	-	factor
season_2	uint8	-
season_3	uint8	-

season_4	uint8	-
weather_2	uint8	-
weather_3	uint8	-
count	float64	numeric

3 Modelling

3.1 Split Data set into training & testing data.

In order to evaluate the performance of model, I split the data into a training set (80% of Train Dataset) and testing set (20% of Train Dataset)

Python:

Dataset with outliers [Python]			
Dataset	**Narration**	**Rows**	**Columns**
X_train	Training dataset which contain 12 predictor variables	584	12
X_test	Testing dataset which contain 12 predictor variables	147	12
y_train	Contain target variable 'count' of training dataset	584	1
y_test	Contain target variable 'count' of testing dataset	147	1

Python code of above result

Dataset without outliers [Python]			
Dataset	**Narration**	**Rows**	**Columns**
X_train_WO	Training dataset which contain 12 predictor variables	574	12
X_test_WO	Testing dataset which contain 12 predictor variables	144	12
y_train_WO	Contain target variable 'count' of training dataset	574	1
y_test_WO	Contain target variable 'count' of testing dataset	144	1

Python code of above result

R:

Dataset with outliers [R]			
Dataset	**Narration**	**Rows**	**Columns**
train_1	Training dataset	584	10
test_1	Testing dataset	147	10

R code of above result

Dataset without outliers [R]			
Dataset	**Narration**	**Rows**	**Columns**
train_WO_1	Training dataset	573	10
Test_WO_1	Testing dataset	144	10

R code of above result

3.2 Evaluation Matrix RMSLE:

There has been a lot of evaluation metrics when it comes to Regression problem. I choose RMSLE for evaluate performance of my model.

Reason:

here our case is a little specific. Under any scenario, we won't like to lose customers because of shortage of supply. Therefore, we need to be careful about the cases when we underestimate the demand than the ones where we overestimate it. RMSE penalises both cases equivalently. Therefore, instead of RMSE, I use RMSLE

Calculation methodology of RMSLE:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2}$$

Note, X is the predicted value and Y is the actual value

Most Important factor of RMSLE is, RMSLE incurs a larger penalty for the underestimation of the Actual variable than the Overestimation.

In simple words, more penalty is incurred when the predicted Value is less than the Actual Value. On the other hand, less penalty is incurred when the predicted value is more than the actual value and that what we want here. Because we don’t want to lose our customer because of short supply of Bike.

Example:

When we under estimate predicted value

Actual value: 1000

Predicted value: 600

RMSLE: 0.510

When we over estimated predicted value

Actual value: 1000

Predicted value: 1400

RMSLE: 0.33

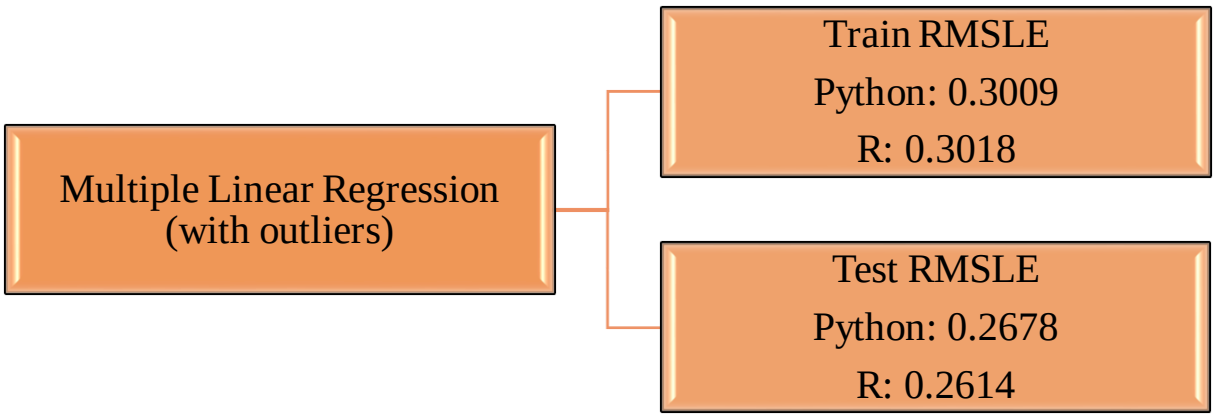
With the help of above example, I can confidently say that RMSLE drastically increase when my predicted value is under estimate than actual value.

3.3 Machine Learning:

After EDA my train & test dataset is ready for machine learning. As proven by "No free lunch" theorem there is no algorithm that is always superior to all others. I have used 4 Machine learning algorithm, Multiple linear regression, K-Nearest neighbour, Decision tree & Random Forest.

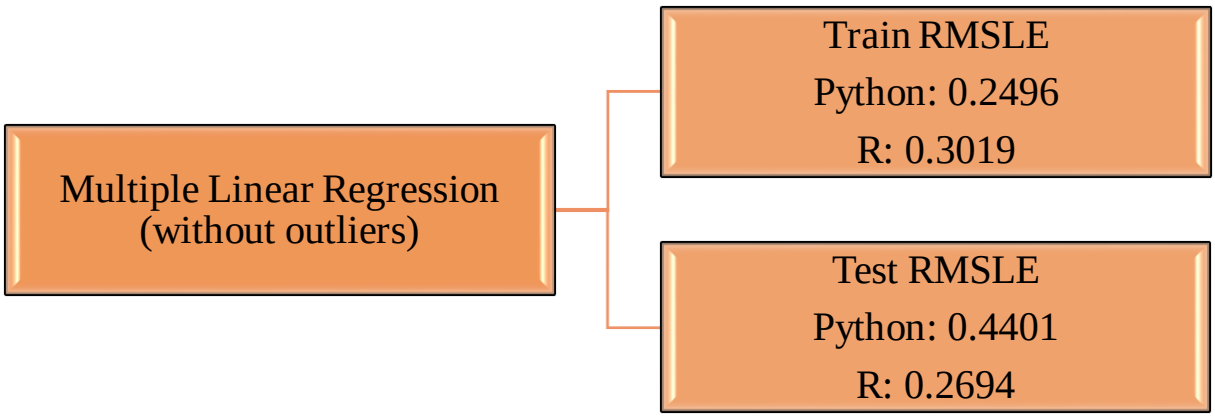
3.3.1 Multiple Linear Regression:

Let's check performance of Multiple Linear Regression (with outliers)



Python code of above result
R code of above result

Let's check performance of Multiple Linear Regression (without outliers)



Python code of above result
R code of above result

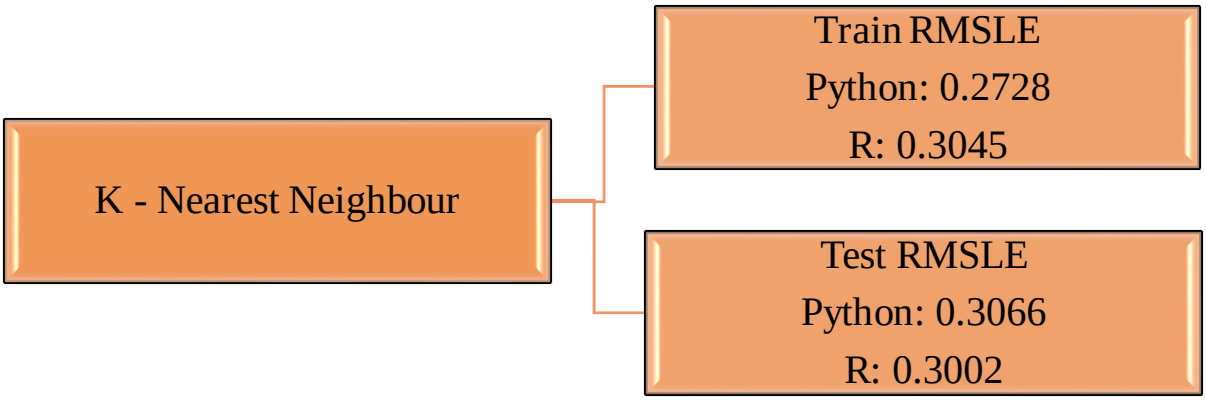
Analysis:

Above output provide significant evidence that Multiple Linear Regression does well with outliers.

Hence for my other models I will use data set with outliers.

3.3.2 K-Nearest Neighbour:

Let's check performance of K-Nearest Neighbour



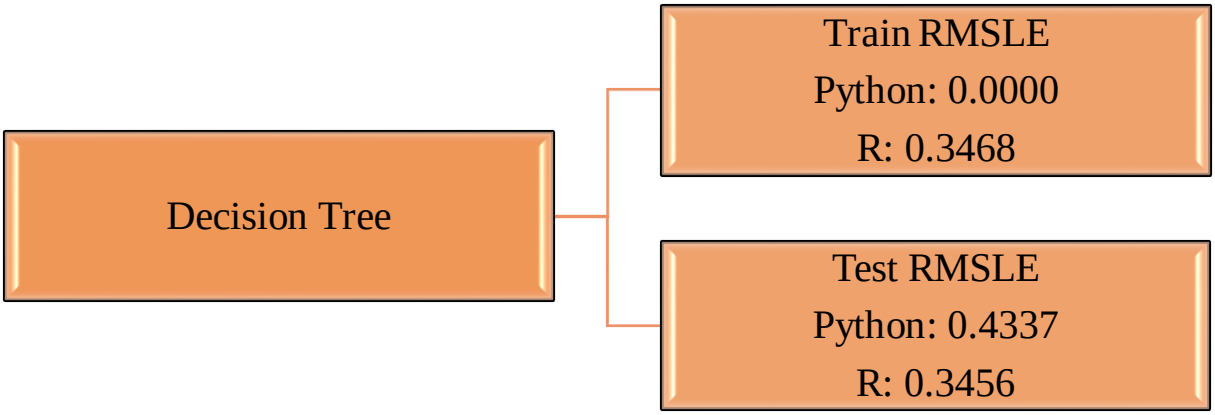
Python code of above result

R code of above result

As per above output of KNN regressor, RMSLE score on test data set is 0.31 & 0.30 in Python & R respectively.

3.3.3 Decision Tree:

Let's check performance of Decision Tree.



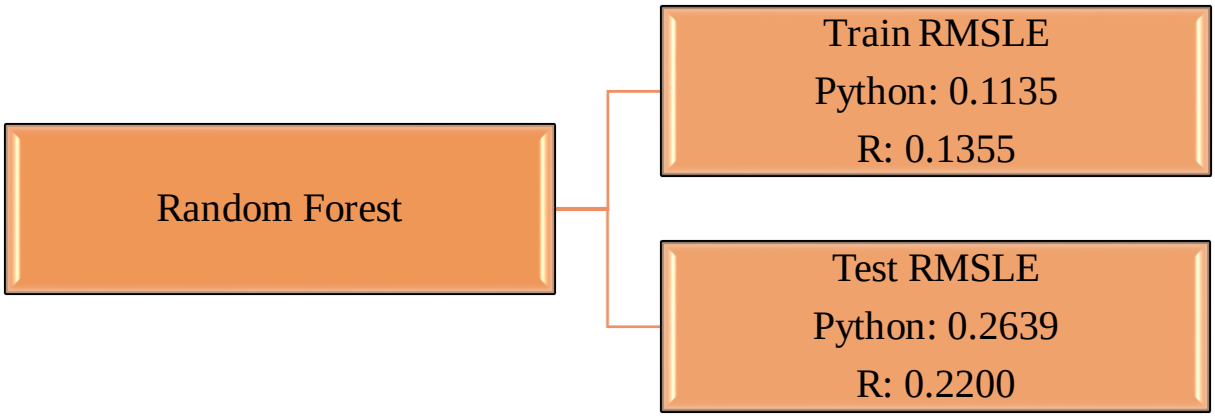
Python code of above result

R code of above result

As per above output of Decision Tree regressor, RMSLE score on test data set is 0.43 & 0.34 in Python & R respectively.

3.3.4 Random Forest (Default):

Let's check performance of Random Forest.



Python code of above result

R code of above result

As per above output of Random Forest, RMSLE score on test data set is 0.26 & 0.22 in Python & R respectively.

3.3.5 Tune Random Forest:

Above RMSLE score of Random Forest default is satisfactory but what if I still want to improve predictive power or decrease our RMSLE score. There are certain ways to do it.

1) Gather more data
2) Feature Engineering
3) Hyper parameter tuning

Gathering more data usually has the greatest payoff in terms of time invested versus improved performance and I have fully exhausted all the variables and data is limited so, now times to move on model hyper-parameter tuning.

Grid search is the basic hyper parameter tuning method. With this technique, I simply build a model for each possible combination of all of the hyper parameter values provided, evaluating each model, and selecting the architecture which produces the best results.

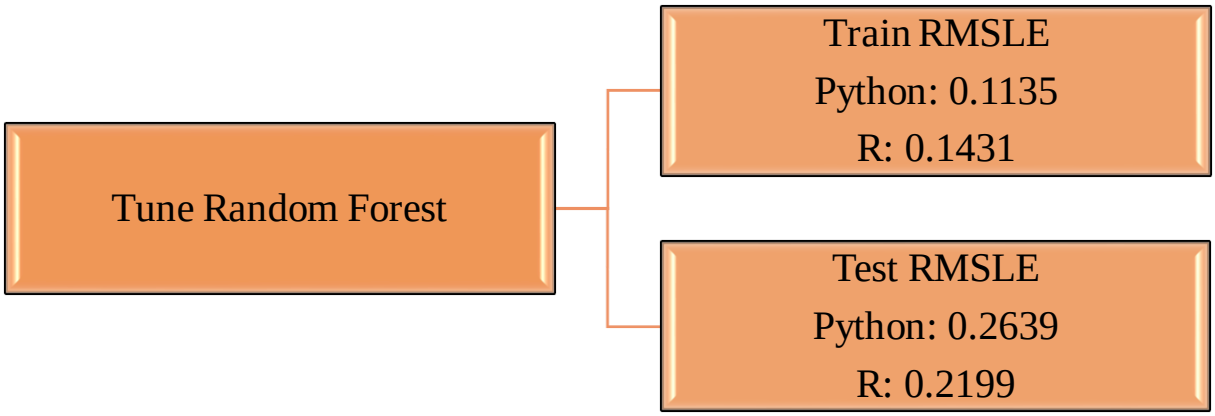
Best hyper parameter in Python

Parameter	Value
max_depth	14
max_features	auto
min_sample_leaf	2
min_sample_split	2
n_estimators	600

Best hyper parameter in R

Parameter	Value
ntree	500
mtry	7

Let's check performance of Tune Random Forest.



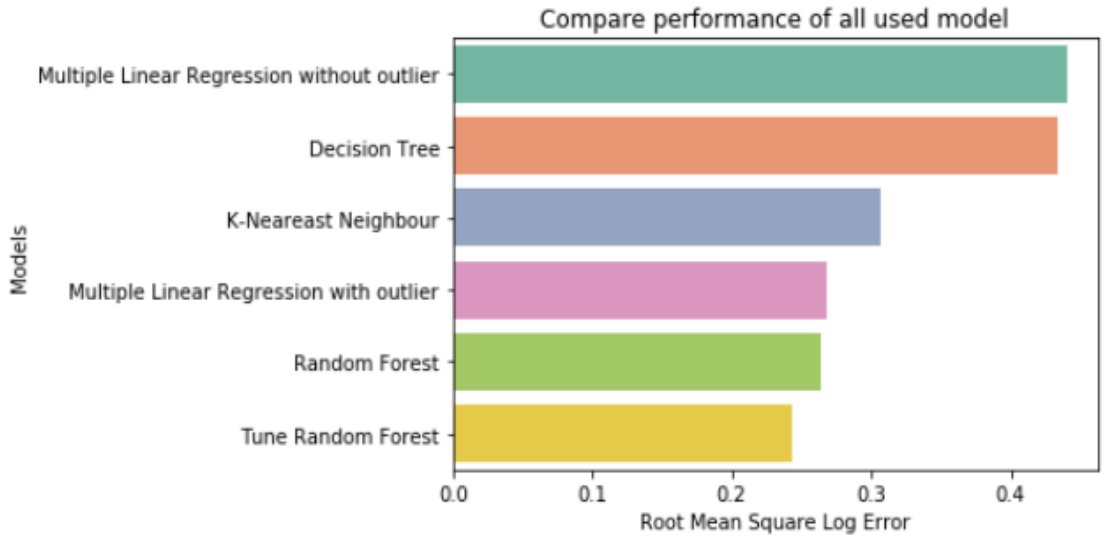
Python code of above result

R code of above result

After hyper-parameter tuning my RMSLE score on test data set comes down from 0.26 to 0.24 in Python & 0.22 to 0.21 in R.

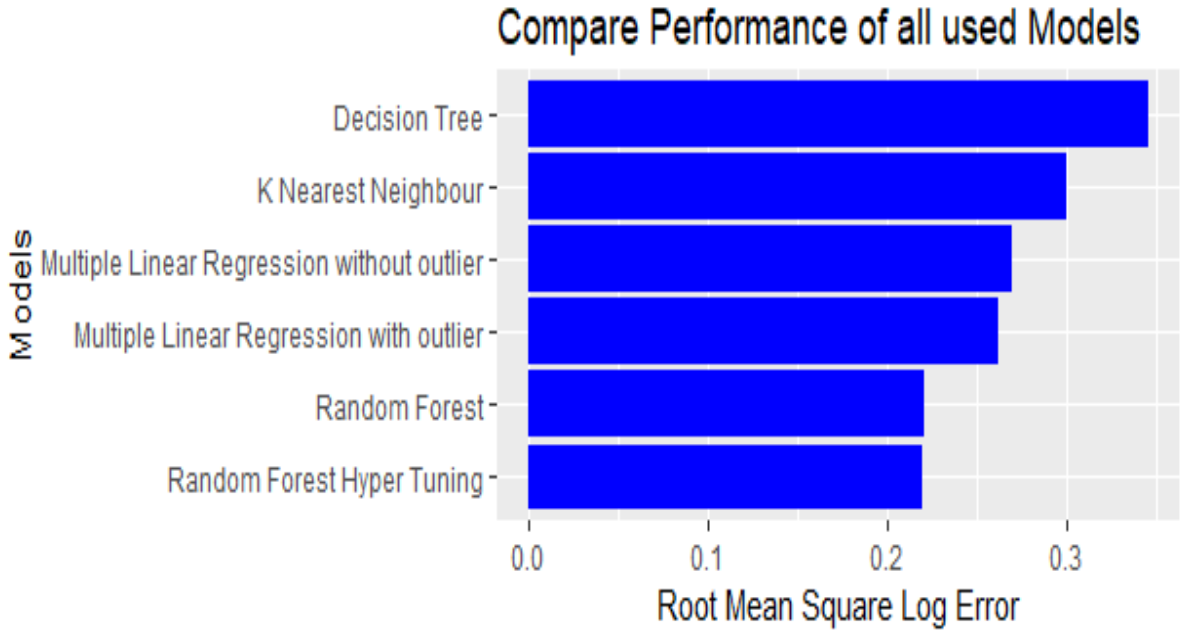
3.4 Compare the RMSLE score of all used models

Python



Python Code of above result

R



R code of above result

Model Name	Test RMSLE	
	Python	R
Multiple Linear Regression (with outliers)	0.26	0.26
Multiple Linear Regression (without outliers)	0.44	0.27
K-Nearest Neighbour	0.31	0.30
Decision Tree	0.43	0.34
Random Forest (Default)	0.26	0.22
Tune Random Forest	0.24	0.21

4 Conclusion

4.1 Final Dataset

All the 731 observations are used.

Remove columns ‘instant’, ‘dteday’, ‘atemp’, ‘holiday’, ‘casual’ and ‘registered’

Rename columns ‘yr : year’, ‘mnth : month’, ‘weathersit : weather’ and ‘cnt : count’

Made bins for ‘month’ & ‘weekday’

Made dummy variables for ‘weather’ & ‘season’ in Python and convert them into the ‘factor’ in R

Outliers from ‘humidity’ & ‘windspeed’ are not removed because statistically they are outlier but removal of those outliers hamper my RMSLE score.

4.2 Final Model

Tune Random Forest gives lowest RMSLE score as compare to other models hence I have fixed Tune Random Forest as a final model for this problem.

4.3 Conclusion

Based on my analysis, I have some recommendations that company could follow to expand their rental business. First, one opportunity for growth is expansion of the company’s customer base in the current stations. Out of the two main customer segments “casual” and “registered” the one the company should focus on is casual users because registered users generally have a lot more ridership and consistency so company should focus on converting casual users to registered users. Additionally, the number of casual rentals is very low during the week compared to registered users so along with trying to have casual users register, company should also focus on increasing weekday ridership by casual users in general.

Above recommendation could be achieve by below mention ways.

- Offering incentives or discounts for casual users to rent bikes during the week.
- company could expand its operations to cities with warm weather patterns that encourage bike sharing because warm weather conditions are a significant factor in achieving high ridership.
- Expatiation of business operation somehow required huge investment, so for maximize profit in areas where the company already operates, it could offer “dynamic pricing” to offer discounts when the weather is forecasted to be bad, such as cold or rainy.
- Company could also vary its prices based on seasons and offer seasonal discounts to increase ridership in months of low rentals.

5 Python Code:

```
# load required libraries
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
os.chdir('D:/Bike') # set working directory


# load train dataset
train = pd.read_csv("day.csv")


# head of train data
train.head(2)

# check variable type and data information python => see result
train.info()


# drop non useful columns 'instant' & 'dteday'
train = train.drop(['instant', 'dteday'], axis = 1)
print("Variables Name:",train.columns)

# rename some columns for understanding
train = train.rename(columns = {'yr': 'year',
                                'mnth': 'month',
                                'weathersit': 'weather',
                                'hum': 'humidity',
                                'cnt': 'count'})
print("Variables Name:",train.columns)

# list of numerical columns & categorical columns
num_col = ['temp', 'atemp', 'humidity', 'windspeed', 'casual','registered', 'count']
cat_col = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']


# statistical summary of numerical columns python => see result
round(train[num_col].describe(),4)

# unique values in categorical columns python => see result
train[cat_col].nunique()

# count of unique categories in each columns python => see result
for c in cat_col:
    print("---- %s ---" % c)
    print(train[c].value_counts())

# checking missing value in Training Dataset python => see result

train.isnull().sum().sort_values(ascending = False)


# # Visualization of Numerical Variables
###################################
#                                 #
#        Histogram of             # => see result
#  Numerical Variables Python     #
#                                 #
###################################
```

```
fig, axi = plt.subplots(ncols = 4, figsize = (20,5))

fig.suptitle("Histogram of Numerical Variables")
axi[0].hist(x = 'temp', data = train, edgecolor = 'black', color = 'tab:olive')
axi[0].set(xlabel = 'Temperature')
axi[1].hist(x = 'atemp', data = train, edgecolor = 'black', color = 'tab:olive')
axi[1].set(xlabel = 'ATemperature')
axi[2].hist(x = 'humidity', data = train, edgecolor = 'black', color = 'tab:olive')
axi[2].set(xlabel = 'Humidity')
axi[3].hist(x = 'windspeed', data = train, edgecolor = 'black', color = 'tab:olive')
axi[3].set(xlabel = 'WindSpeed')

# Skewness of Independent Numerical Variables python => see result
print("Skewness of Temperature:",round(train['temp'].skew(),2))
print("Skewness of ATemperature:",round(train['atemp'].skew(),2))
print("Skewness of Humidity:",round(train['humidity'].skew(),2))
print("Skewness of Windspeed:",round(train['windspeed'].skew(),2))



##################################
#                                #
#          Boxplot of            #
#   Numerical Variables Python   # => see result
#                                #
##################################

fig, axi = plt.subplots(ncols = 4, figsize = (20,5))

fig.suptitle("Boxplot of Numerical Variables")
sns.boxplot(x = 'temp', data = train, orient = 'v', color = "tab:olive", ax = axi[0])

axi[0].set(xlabel = 'Temperature')
sns.boxplot(x = 'atemp', data = train, orient = 'v', color = "tab:olive", ax = axi[1])

axi[1].set(xlabel = 'ATemperature')
sns.boxplot(x = 'humidity', data = train, orient = 'v', color = "tab:olive", ax = axi[
   2])
axi[2].set(xlabel = 'Humidity')
sns.boxplot(x = 'windspeed', data = train, orient = 'v', color = "tab:olive", ax = axi
   [3])
axi[3].set(xlabel = 'Wind Speed')


##################################
#                                #
#       Scatter plot of          #
#  Numerical Variables Python    # => see result
#                                #
##################################

fig, axi = plt.subplots(ncols = 4, figsize = (20,5))

fig.suptitle("Correlation plot of Numerical Variables with Target Variable")
sns.regplot(x = "temp", y = "count", data = train, color = "tab:olive", ax = axi[0])
axi[0].set(xlabel = "Temperature")
sns.regplot(x = "atemp", y = "count", data = train, color = "tab:olive", ax = axi[1])

axi[1].set(xlabel = "ATemperature")
sns.regplot(x = "humidity", y = "count", data = train, color = "tab:olive", ax = axi[2
   ])
axi[2].set(xlabel = "Humidity")
sns.regplot(x = "windspeed", y = "count", data = train, color = "tab:olive", ax = axi[
   3])
axi[3].set(xlabel = "Wind Speed")
```

```
# Correlation of Numerical variable with Target Variable python => see result
print('Correlation of Temperature with Target Variable:',round(train['temp'].corr(trai
   n['count']),2))
print('Correlation of ATemperature with Target Variable:',round(train['atemp'].corr(tr
   ain['count']),2))
print('Correlation of Humidity with Target Variable:',round(train['humidity'].corr(tra
   in['count']),2))
print('Correlation of Windspeed with Target Variable:',round(train['windspeed'].corr(t
   rain['count']),2))


# # Visualization of Categorical Variables
###################################
#                                 #
#          Bar plot of            #
#     Numerical Variables         #
#                                 #
###################################

# Mean Count of Year & Month python => see result
# Mean Count of weekday python => see result
fig, axi = plt.subplots(ncols = 3, figsize = (20,5))

fig.suptitle('Mean Count of Year, Month & Weekday')

year_mean = pd.DataFrame(train.groupby('year')['count'].mean()).reset_index()
month_mean = pd.DataFrame(train.groupby('month')['count'].mean()).reset_index()
weekday_mean = pd.DataFrame(train.groupby('weekday')['count'].mean()).reset_index()

sns.barplot(x = 'year', y = 'count', data = year_mean, color = 'tab:olive',ax = axi[0]
   )
sns.barplot(x = 'month', y = 'count', data = month_mean,color = 'tab:olive', ax = axi[
   1])
sns.barplot(x = 'weekday', y = 'count', data = weekday_mean,color = 'tab:olive', ax =
   axi[2])

# count of Month with respect of Year python => see result
# count of weekday with respect of Year python => see result
fig, axi = plt.subplots(ncols = 2, figsize = (20,5))
fig.suptitle('Count of Month & Weekday with respect of year')

sns.barplot(x = 'weekday', y = 'count', data = train, hue = 'year',ax = axi[0])
sns.barplot(x = 'month', y = 'count', data = train, hue = 'year',ax = axi[1])

# Mean count of Season & Weather

fig, axi = plt.subplots(ncols = 2, figsize = (20,4))

fig.suptitle('Mean Count of Season & Weather')
season_mean = pd.DataFrame(train.groupby('season')['count'].mean()).reset_index()
weather_mean = pd.DataFrame(train.groupby('weather')['count'].mean()).reset_index()

sns.barplot(x = 'season', y = 'count', data = season_mean, color = 'tab:olive',ax = ax
   i[0])
sns.barplot(x = 'weather', y = 'count', data = weather_mean,color = 'tab:olive', ax =
   axi[1])



###################################
#                                 #
#     Feature Engineering         #
#                                 #
###################################

# create function for month_bin
```

```
def month_bin(df):
    if df['month'] <= 4 or df['month'] >= 11:
        return(0)
    else:
        return(1)

# create function for weekday_bin
def weekday_bin(df):
    if df['weekday'] < 2:
        return(0)
    else:
        return(1)

# create two new variables 'month_bin' & 'weekday_bin' and drop original variable 'mon
   th' & 'weekday'
train['month_bin'] = train.apply(lambda df: month_bin(df), axis = 1)
train = train.drop(['month'], axis = 1)
train['weekday_bin'] = train.apply(lambda df: weekday_bin(df), axis = 1)
train = train.drop(['weekday'], axis = 1)


###################################
#                                 #
#          Heat Map of            # => see result
#   Numerical variables python    #
#                                 #
###################################

corr = train[num_col].corr()
mask = np.array(corr)
mask[np.tril_indices_from(mask)] = False
fig, ax = plt.subplots(figsize = (12,8))
sns.heatmap(corr, mask = mask, vmax = 7, square = True, annot = True)



###################################
#                                 #
#   Chi-Square Test python        # => see result
#                                 #
###################################

# list of Categorical Independent Variables
cat_col = ['season', 'year', 'holiday', 'workingday','weather','month_bin','weekday_bi
   n']

category_paired = [(a,b) for a in cat_col for b in cat_col]

# make p-value matrix
p_values = []
from scipy.stats import chi2_contingency
for f in category_paired:
    if f[0] != f[1]:
        chi2, p, dof, ex = chi2_contingency(pd.crosstab(train[f[0]],train[f[1]]))

        p_values.append(p.round(4))
    else:
        p_values.append('NA')
p_values = np.array(p_values).reshape((7,7))
p_values = pd.DataFrame(p_values, index=cat_col, columns=cat_col)
print(p_values)


###################################
#                                 #
#   Feature Importance python     # => see result
```

```
#                                 #
###################################

# list of unwanted columns to check feature importance
unwanted_col = ['count', 'registered', 'casual']

# Check Feature importance with the help of Random Forest
from sklearn.ensemble import RandomForestRegressor
rf_imp = RandomForestRegressor(n_estimators = 300)
x = train.drop(columns = unwanted_col)
y = train['count']
rf_imp.fit(x,y)
feature_imp = pd.DataFrame({'features': train.drop(columns = unwanted_col).columns,
                            'importance': rf_imp.feature_importances_})
feature_imp.sort_values(by = 'importance', ascending = False).reset_index(drop = True)



###################################
#                                 #
#  Multi - Collinearity Python    # => see result
#                                 #
###################################

# Check VIF score of Independent Numerical Variables (before removal)
from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF

from statsmodels.tools.tools import add_constant
df_numeric = add_constant(train[['temp','atemp','humidity','windspeed']])
VIF = pd.Series([VIF(df_numeric.values, i) for i in range(df_numeric.shape[1])],
                index = df_numeric.columns)
print(VIF)

# Check VIF score of Independent Numerical Variables (after removal)
from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF

from statsmodels.tools.tools import add_constant
df_numeric = add_constant(train[['temp','humidity','windspeed']])
VIF = pd.Series([VIF(df_numeric.values, i) for i in range(df_numeric.shape[1])],
                index = df_numeric.columns)
print(VIF)



###################################
#                                 #
#    one hot encoding for         #
#      season & weather           #
#                                 #
###################################

# list of columns for dummy variables
one_hot_col = ['season','weather']


# define function to create dummy variables
def one_hot(df, cols):
    dummies = pd.get_dummies(df[cols], prefix=cols, drop_first=True)
    df = pd.concat([df, dummies], axis=1)
    df = df.drop([cols], axis = 1)
    return df

# create dummy variables
for cols in one_hot_col:
    train = one_hot(train, cols)
```

```
# # remove outlier from windspeed and humidity

# make copy of original dataframe
train_WO = train.copy()


#################################
#                               #
#    remove outliers form       #
#    humidity & windspeed       #
#                               #
#################################

# define function for outlier removal from selected columns
def remove_out(df, col):
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3-q1
    low_fen = q1-1.5*iqr
    upr_fen = q3+1.5*iqr
    df = df.loc[(df[col] > low_fen) & (df[col] < upr_fen)]
    return df

# list of outlier columns
outlier = ['humidity','windspeed']

# remove outlier form 'train_WO'
print('Shape of Train Data before removing Outliers:', train_WO.shape)
for col in outlier:
    train_WO = remove_out(train, col)
print('Shape of Train Data after removing Outliers:', train_WO.shape)

# check variable type and data information
train.info()


# # Remove unwanted columns

train = train.drop(['holiday', 'atemp', 'casual', 'registered'], axis = 1)
train_WO = train_WO.drop(['holiday', 'atemp', 'casual', 'registered'], axis = 1)


# Split data in Train & Test
from sklearn.model_selection import train_test_split


#################################
#                               #
#     Train & Test dataset      # => see result
#    (with outliers) python     #
#                               #
#################################

X = train.drop(['count'], axis = 1)
y = train['count']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
   e = 0)

# log conversion
y_train_log = np.log1p(y_train)
y_test_log = np.log1p(y_test)

# with outlier
print(X_train.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_test.shape)


#################################
#                               #
#     Train & Test dataset      # => see result
# (without outliers) python_WO  #
#                               #
#################################

X = train_WO.drop(['count'], axis = 1)
y = train_WO['count']
X_train_WO, X_test_WO, y_train_WO, y_test_WO = train_test_split(X, y, test_size = 0.2,
    random_state = 0)

# log conversion
y_train_WO_log = np.log1p(y_train_WO)
y_test_WO_log = np.log1p(y_test_WO)

# without outliers
print(X_train_WO.shape)
print(X_test_WO.shape)
print(y_train_WO.shape)
print(y_test_WO.shape)


#################################
#                               #
#      Evaluation Metircs       #
#        RMSLE Function         #
#                               #
#################################

def rmsle(actual, pred):
    actual = np.exp(actual)
    pred = np.exp(pred)
    rmsle = np.sqrt(np.mean((np.log1p(actual) - np.log1p(pred))**2))
    return rmsle



#################################
#                               #
#  Multiple Linear Regression   # => see result
#   (with outliers) python      #
#                               #
#################################

from sklearn.linear_model import LinearRegression
lr = LinearRegression()

# create Multiple Linear Regression model
lr.fit(X_train, y_train_log)

# make prediction of Multiple Linear regression Model for Train & Test data
lr_train_pred = lr.predict(X_train)
lr_test_pred = lr.predict(X_test)

lr_tr_rmsle = round(rmsle(y_train_log, lr_train_pred),4)
lr_te_rmsle = round(rmsle(y_test_log, lr_test_pred),4)

print("Multiple Linear Regression RMSLE Score for Train & Test Data")
print(f'Linear Regression Train Dataset: RMSLE:', lr_tr_rmsle)
print(f'Linear Regression Test Dataset:  RMSLE:', lr_te_rmsle)
```

```
#################################
#                               #
#  Multiple Linear Regression   # => see result
#  (without outliers) Python    #
#                               #
#################################

from sklearn.linear_model import LinearRegression
lr_WO = LinearRegression()

# create Multiple Linear Regression model
lr_WO.fit(X_train_WO, y_train_WO_log)

# make prediction of Multiple Linear regression Model for Train & Test data
lr1_train_pred = lr_WO.predict(X_train_WO)
lr1_test_pred = lr_WO.predict(X_test_WO)

lr1_tr_rmsle = round(rmsle(y_train_WO_log, lr1_train_pred),4)
lr1_te_rmsle = round(rmsle(y_test_WO_log, lr1_test_pred),4)

print("Multiple Linear Regression RMSLE Score for Train & Test Data")
print(f'Linear Regression Train Dataset: RMSLE:',lr1_tr_rmsle)
print(f'Linear Regression Test Dataset:  RMSLE:',lr1_te_rmsle)



#################################
#                               #
#  K Nearest Neighbour python  # => see result
#                               #
#################################

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()

# create KNN model
knn.fit(X_train, y_train_log)

# make prediction of KNN Model for Train & Test data
knn_train_pred = knn.predict(X_train)
knn_test_pred = knn.predict(X_test)

knn_tr_rmsle = round(rmsle(y_train_log, knn_train_pred),4)
knn_te_rmsle = round(rmsle(y_test_log, knn_test_pred),4)

print("K Neareast Neighbour RMSLE Score for Train & Test Data")
print(f'K Neareast Neighbour Train Dataset: RMSLE:',knn_tr_rmsle)
print(f'K Neareast Neighbour Test Dataset:  RMSLE:',knn_te_rmsle)



#################################
#                               #
#    Decision Tree python       # => see result
#                               #
#################################

from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()

# create Decision Tree Model
dt.fit(X_train, y_train_log)

# make prediction of Decision Tree Model for Train & Test Data
```

```
dt_train_pred = dt.predict(X_train)
dt_test_pred = dt.predict(X_test)

dt_tr_rmsle = round(rmsle(y_train_log, dt_train_pred),4)
dt_te_rmsle = round(rmsle(y_test_log, dt_test_pred),4)

print("Decision Tree RMSLE Score for Train & Test Data")
print(f'Decision Tree Train Dataset: RMSLE:',dt_tr_rmsle)
print(f'Decision Tree Test Dataset:  RMSLE:',dt_te_rmsle)



#################################
#                               #
#  Random Forest Default python # => see result
#                               #
#################################

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

# create Random Forest Model
rf.fit(X_train, y_train_log)

# make prediction of Random Forest Model for Train & Test Data
rf_train_pred = rf.predict(X_train)
rf_test_pred = rf.predict(X_test)

rf_tr_rmsle = round(rmsle(y_train_log,rf_train_pred ),4)
rf_te_rmsle = round(rmsle(y_test_log, rf_test_pred),4)

print("Random Forest RMSLE Score for Train & Test Data")
print(f'Random Forest Train Dataset: RMSLE:',rf_tr_rmsle)
print(f'Random Forest Test Dataset:  RMSLE:',rf_te_rmsle)


#################################
#                               #
#   Tune Random Forest python   # => see result
#                               #
#################################

from sklearn.model_selection import GridSearchCV
rf2 = RandomForestRegressor(random_state=1)

params = [{'n_estimators' : [500, 600, 800],'max_features':['auto', 'sqrt'],
           'min_samples_split':[2,4,6],'max_depth':[12, 14, 16],'min_samples_leaf':[2,
   3,5],
           'random_state' :[1]}]

grid_search = GridSearchCV(estimator=rf2, param_grid=params,cv = 5, n_jobs=-1)

grid_search = grid_search.fit(X_train, y_train)
print(grid_search.best_params_)

rf22 = RandomForestRegressor(random_state=1,max_depth = 14, max_features = 'auto', min
   _samples_leaf = 2,
                             min_samples_split = 2, n_estimators =600)

# create Tune Random Forest Model
rf_final = rf22.fit(X_train, y_train_log)

# make prediction of Tune Random Forest Model for Train & Test Data
rf_tr = rf_final.predict(X_train)
rf_te = rf_final.predict(X_test)
```

```
rf1_tr_rmsle = round(rmsle(y_train_log,rf_tr ),4)
rf1_te_rmsle = round(rmsle(y_test_log, rf_te),4)

print("Random Forest RMSLE Score for Train & Test Data")
print(f'Random Forest Train Dataset: RMSLE:',rf_tr_rmsle)
print(f'Random Forest Test Dataset:  RMSLE:',rf_te_rmsle)



##################################
#                                #
#  Compare the performance of    # => see result
#    all used models python      #
#                                #
##################################

# make data frame 'all_model' for store RMSE score of all used model
all_model = pd.DataFrame({"model": ['Multiple Linear Regression with outlier',
                                    'Multiple Linear Regression without outlier',
                                    'K-Neareast Neighbour',
                                    'Decision Tree',
                                    'Random Forest','Tune Random Forest'],
                          "train_rmsle": [lr_tr_rmsle, lr1_tr_rmsle, knn_tr_rmsle,
                                          dt_tr_rmsle, rf_tr_rmsle, rf1_tr_rmsle],
                         "test_rmsle": [lr_te_rmsle, lr1_te_rmsle, knn_te_rmsle,
                                        dt_te_rmsle, rf_te_rmsle, rf1_te_rmsle]},
                         columns = ['model','train_rmsle','test_rmsle'])
all_model = all_model.sort_values(by = 'test_rmsle', ascending = False)

# create a barplot for all_model
sns.barplot(all_model['test_rmsle'], all_model['model'], palette = 'Set2')
plt.xlabel('Root Mean Square Log Error')
plt.ylabel('Models')
plt.title('Compare performance of all used model')

print(all_model)
```

6 R Code:

```
# import librararies
library(ggplot2)
library(dplyr)
library(randomForest)
library(gridExtra)
library(ggcorrplot)
library(usdm)
library(caret)
library(moments)


#  Set Working Directory
setwd("D:/Bike")

# Lodad Data Set
train = read.csv('day.csv')

cat("Dimension of Train Data:", dim(train))

# head of train data
head(train)

# check variable type and data information R => see result
str(train)

summary(train)

# drop non useful columns 'instant' & 'dteday'
train = train %>%
  dplyr::select(-instant, -dteday)

# rename some columns for understanding
train = train %>%
  dplyr::rename(year = yr,
                month = mnth,
                weather = weathersit,
                humidity = hum,
                count = cnt)

# list of numerical columns & categorical columns
num_col = c('temp', 'atemp', 'humidity', 'windspeed', 'casual','regi
stered', 'count')
cat_col = c('season', 'year', 'month', 'holiday', 'weekday', 'workin
gday','weather')

# statistical summary of numerical columns R => see result
summary(train[,num_col])
```

```
# unique values in categorical columns R => see result
rapply(train[,cat_col], function(x)length(unique(x)))

# count of unique categories in each columns R => see result
lapply(train[,cat_col], function(x)table(x))

################################
#                              #
#      Missing Value R         # => see result
#                              #
################################

apply(train, 2, function(x){sum(is.na(x))})



# # Visualization of Numerical Variables
##################################
#                                #
#          Boxplot of            # => see result
#     Numerical Variables R      #
#                                #
##################################

box_plot = function(df, col){
  df$x = 1
  ggplot(aes_string(x = 'x', y = col), data = df)+
    geom_boxplot()+
    xlab(col)+
    ggtitle(paste("Box-Plot: ",col))
}

all_boxplot = lapply(c('temp','atemp','humidity','windspeed'), box_p
lot, df = train)

grid.arrange(all_boxplot[[1]],all_boxplot[[2]],all_boxplot[[3]],all_
boxplot[[4]], ncol = 4)



##################################
#                                #
#        Histogram of            # => see result
#    Numerical Variables R       #
#                                #
##################################

histogram = function(df, col){
  ggplot(aes_string(col), data = df)+
    geom_histogram(fill = 'blue', color = 'black',bins = 10)+
    xlab(col)+
```

```
    ggtitle(paste("Histogram: ",col))

}

all_histogram = lapply(c('temp','atemp','humidity','windspeed'), his
togram, df = train)

grid.arrange(all_histogram[[1]],all_histogram[[2]],all_histogram[[3]
],all_histogram[[4]],
             ncol = 4)

# skewness of independent Numerical Variables R => see result

cat('Skewness of Temperature:',round(skewness(train$temp),2))
cat('Skewness of ATemperature:',round(skewness(train$atemp),2))
cat('Skewness of Humidity:',round(skewness(train$humidity),2))
cat('Skewness of Windspeed:',round(skewness(train$windspeed),2))


###################################
#                                 #
#       Scatter plot of           # => see result
#    Numerical Variables R        #
#                                 #
###################################

scatter = function(df, col){
  ggplot(aes_string(y = 'count', x = col),data = df)+
    geom_point(color = 'blue')+
    geom_smooth(method = 'lm', se = FALSE)+
    xlab(col)+
    ggtitle(paste("Scatter Plot: ",col))
}


all_scatter = lapply(c('temp','atemp','humidity','windspeed'), scatt
er, df = train)

grid.arrange(all_scatter[[1]],all_scatter[[2]],all_scatter[[3]],all_
scatter[[4]],ncol = 4)

# Correlation of Numerical Variables with target R => see result

cat('Correlation of Temperature with Target Variable:',round(cor(tra
in$temp, train$count),2))
cat('Correlation of ATemperature with Target Variable:',round(cor(tr
ain$atemp, train$count),2))
cat('Correlation of Humidity with Target Variable:',round(cor(train$
humidity, train$count),2))
cat('Correlation of Windspeed with Target Variable:',round(cor(train
$windspeed, train$count),2))
```

```
# # Visualization of Categorical Variables
###################################
#                                 #
#         Bar plot of             # => see result year & month
#    Categorical Variables R      # => see result weekday
#                                 #
###################################

bar_plot = function(df, cat_col, val_col){

  cc <- enquo(cat_col)
  vc <- enquo(val_col)
  cc_name <- quo_name(cc) # generate a name from the enquoted statem
ent!

  df <- df %>%
    dplyr::group_by(!!cc) %>%
    dplyr::summarise (mean_count = mean(!!vc)) %>%
    dplyr::mutate(!!cc_name := factor(!!cc, !!cc)) # insert pc_name
here!

  ggplot(df) + aes_(y = ~mean_count, x = substitute(cat_col)) +
    geom_bar(stat="identity", width = 0.5)+
    ggtitle(paste("Bar Plot: ", substitute(cat_col)))

}

# create bar plot
year = bar_plot(train, year, count)
month = bar_plot(train, month, count)
weekday = bar_plot(train, weekday, count)


grid.arrange(year, month, weekday, ncol = 3)

season = bar_plot(train, season, count)
weather = bar_plot(train, weather, count)

grid.arrange(season, weather, ncol = 2)




# mean count of Month with respect of Year R => see result

# mean count of weekday with respect of Year R => see result
```

```
month_yr = ggplot(train, aes(x = factor(month), y = count, fill = fa
ctor(year)))+
  stat_summary(fun.y = 'mean', geom = 'bar',position = 'dodge')

week_yr = ggplot(train, aes(x = factor(weekday), y = count, fill = f
actor(year)))+
  stat_summary(fun.y = 'mean', geom = 'bar',position = 'dodge')

grid.arrange(month_yr, week_yr, ncol = 2)



###################################
#                                 #
#     Feature Engineering         #
#                                 #
###################################

# create function for month_bin
month_bin = function(df){
  df %>%
    mutate(month_bin = ifelse(month <= 4, 0,
                              ifelse(month >= 11, 0, 1))) %>%
    dplyr::select(-month)
}


# create function for weekday_bin
weekday_bin = function(df){
  df %>%
    mutate(weekday_bin = ifelse(weekday < 2, 0, 1)) %>%
    dplyr::select(-weekday)
}


# create two new variables 'month_bin' & 'weekday_bin' and drop orig
inal variabl
train = month_bin(train)
train = weekday_bin(train)

###################################
#                                 #
#       Heat Map of               # => see result
#  Numerical variables R          #
#                                 #
###################################

corr = round(cor(train[,6:12]),2)

ggcorrplot(corr, hc.order = TRUE, type = 'lower', lab = TRUE)
```

```
###################################
#                                 #
#       Chi-Square Test R         # => see result
#                                 #
###################################

chisqmatrix <- function(x){
  names = colnames(x)
  num = length(names)
  m = matrix(nrow=num,ncol=num,dimnames=list(names,names))
  for (i in 1:(num-1)) {
    for (j in (i+1):num) {
      m[i,j] = round(chisq.test(x[,i],x[,j],)$p.value,4)
    }
  }
  return (m)
}


chisquare_mat = chisqmatrix(train[,c(1,2,3,4,5,13,14)])

## Warning in chisq.test(x[, i], x[, j], ): Chi-squared approximatio
n may be
## incorrect

print(chisquare_mat)


###################################
#                                 #
#    Feature Importance R         # => see result
#                                 #
###################################

# Check Feature importance with the help of Random Fores
rf = randomForest(count ~., data = train[,-c(10,11)], ntree = 300, i
mportance = TRUE)


imp = data.frame(importance(rf, type = 1))
imp$features = rownames(imp)
names(imp)[1] = 'feature_imp'
rownames(imp) = NULL
imp = imp[,c(2,1)]
imp = imp[order(-imp$feature_imp),]
print(imp)

###################################
#                                 #
#    Multi - Collinearity R       # => see result
#                                 #
```

```
#################################

# Check VIF score of Independent Numerical Variables (before removal
)
vif(train[,c(6,7,8,9)])

# Check VIF score of Independent Numerical Variables (after removal)
vif(train[,c(6,8,9)])

#################################
#                               #
#    factor conversion of       #
#      season & weather         #
#                               #
#################################

train[,c('season','weather')] = lapply(train[,c('season','weather')]
, as.factor)


# make copy of original dataframe
train_WO = train


#################################
#                               #
#    remove outliers form       #
#    humidity & windspeed       #
#                               #
#################################

for (i in c('humidity', 'windspeed')){
  outlier = train_WO[,i] [train_WO[,i] %in% boxplot.stats(train_WO[,
i])$out]
  train_WO = train_WO[which(!train_WO[,i] %in% outlier),]
}


# # Remove unwanted columns

train = train %>%
  dplyr::select(-holiday, -atemp, -casual, - registered)

train_WO = train_WO %>%
  dplyr::select(-holiday, -atemp, -casual, - registered)


#################################
#                               #
#     Train & Test dataset      # => see result
#       (with outliers) R       #
```

```
#                                  #
####################################

set.seed(1)

sample = sample.int(n = nrow(train), size = floor(.80*nrow(train)),
replace = FALSE)
train_1 = train[sample,]
test_1 = train[-sample,]


####################################
#                                  #
#     Train & Test dataset         # => see result
#   (without outliers) R_WO        #
#                                  #
####################################

sample_WO = sample.int(n = nrow(train_WO), size = floor(0.80*nrow(tr
ain_WO)), replace = FALSE)
train_WO_1 = train_WO[sample_WO,]
test_WO_1 = train_WO[-sample_WO,]


####################################
#                                  #
#      Evaluation Metircs          #
#        RMSLE Function            #
#                                  #
####################################

rmsle = function(actual, pred){
  actual = exp(actual)
  pred = exp(pred)
  sqrt(mean((log1p(actual) - log1p(pred))**2))
}


####################################
#                                  #
#  Multiple Linear Regression      # => see result
#      (with outliers) R1          #
#                                  #
####################################

# create Multiple Linear Regression model
lr_mod = caret::train(log1p(count) ~., data = train_1, method = 'lm'
)

# make prediction of Multiple Linear regression Model for Train & Te
```

```
st data
lr_train_pred = predict(lr_mod, train_1)
lr_test_pred = predict(lr_mod, test_1)

lr_tr_rmsle = round(rmsle(log1p(train_1$count), lr_train_pred),4)
lr_te_rmsle = round(rmsle(log1p(test_1$count), lr_test_pred),4)

cat("Linear Regression Train RMSLE", lr_tr_rmsle)

cat("Linear Regression Test RMSLE", lr_te_rmsle)



###################################
#                                 #
#  Multiple Linear Regression     # => see result
#   (without outliers) R2         #
#                                 #
###################################

# create Multiple Linear Regression model
lr_mod_WO = caret::train(log1p(count) ~., data = train_WO_1, method
= 'lm')

# make prediction of Multiple Linear regression Model for Train & Te
st data
lr1_train_pred = predict(lr_mod_WO, train_WO_1)
lr1_test_pred = predict(lr_mod_WO, test_WO_1)

lr1_tr_rmsle = round(rmsle(log1p(train_WO_1$count), lr1_train_pred),
4)
lr1_te_rmsle = round(rmsle(log1p(test_WO_1$count), lr1_test_pred),4)

cat("Linear Regression Train RMSLE:", lr1_tr_rmsle)

cat("Linear Regression Test RMSLE:", lr1_te_rmsle)



###################################
#                                 #
#    K Nearest Neighbor R         # => see result
#                                 #
###################################

# create KNN model
knn_mod = caret::train(log1p(count) ~., data = train_1, method = 'kn
n')

# make prediction of KNN Model for Train & Test data
knn_train_pred = predict(knn_mod, train_1)
knn_test_pred = predict(knn_mod, test_1)
```

```
knn_tr_rmsle = round(rmsle(log1p(train_1$count), knn_train_pred),4)
knn_te_rmsle = round(rmsle(log1p(test_1$count), knn_test_pred),4)

cat("K-Nearest Neighbour Train RMSLE:", knn_tr_rmsle)

cat("K-Nearest Neighbour Test RMSLE:", knn_te_rmsle)



###################################
#                                 #
#       Decision Tree R           # => see result
#                                 #
###################################

# create Decision Tree Model
dt_mod = caret::train(log1p(count) ~., data = train_1, method = 'rpa
rt2')

# make prediction of Decision Tree Model for Train & Test Data
dt_train_pred = predict(dt_mod, train_1)
dt_test_pred = predict(dt_mod, test_1)

dt_tr_rmsle = round(rmsle(log1p(train_1$count), dt_train_pred),4)
dt_te_rmsle = round(rmsle(log1p(test_1$count), dt_test_pred),4)

cat("Decision Tree Train RMSLE:", dt_tr_rmsle)

cat("Decision Tree Test RMSLE:", dt_te_rmsle)



###################################
#                                 #
#   Random Forest Default R       # => see result
#                                 #
###################################

# create Random Forest Model
rf_mod = caret::train(log1p(count) ~., data = train_1, method = 'rf'
)

# make prediction of Random Forest Model for Train & Test Data
rf_train_pred = predict(rf_mod, train_1)
rf_test_pred = predict(rf_mod, test_1)

rf_tr_rmsle = round(rmsle(log1p(train_1$count), rf_train_pred),4)
rf_te_rmsle = round(rmsle(log1p(test_1$count), rf_test_pred),4)

cat("Random Forest Train RMSLE:", rf_tr_rmsle)
```

```
cat("Random Forest Test RMSLE:", rf_te_rmsle)



###################################
#                                 #
#     Tune Random Forest R        # => see result
#                                 #
###################################

cnt = trainControl(method = 'repeatedcv', n = 5, repeats = 4)
rf1_mod = caret::train(log1p(count) ~., data = train_1, method = 'rf
', trControl = cnt)

rf1_mod$bestTune

##   mtry
## 2    7

rf1_mod$finalModel

##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 0.09183866
##                     % Var explained: 73.51

rf1_train_pred = predict(rf1_mod, train_1)
rf1_test_pred = predict(rf1_mod, test_1)

rf1_tr_rmsle = round(rmsle(log1p(train_1$count), rf1_train_pred),4)
rf1_te_rmsle = round(rmsle(log1p(test_1$count), rf1_test_pred),4)

cat('Tune Random Forest Train RMSLE', rf1_tr_rmsle)

cat('Tune Random Forest Test RMSLE', rf1_te_rmsle)

###################################
#                                 #
#  Compare the performance of     # => see result
#       all used models R         #
#                                 #
###################################



all_model = data.frame("model" = c("Multiple Linear Regression with
outlier",
```

```
                                    "Multiple Linear Regression witho
ut outlier",
                                    "K Nearest Neighbour", "Decision
Tree",
                                    "Random Forest", "Tune Random For
est"),
                          "train_rmsle" = c(lr_tr_rmsle, lr1_tr_rmsle,
knn_tr_rmsle,
                                            dt_tr_rmsle, rf_tr_rmsle,rf
1_tr_rmsle),
                          "test_rmsle" = c(lr_te_rmsle, lr1_te_rmsle, k
nn_te_rmsle,
                                           dt_te_rmsle, rf_te_rmsle, rf
1_te_rmsle))


all_model = all_model[order(all_model$test_rmsle),]

print(all_model)

# Bar graph of model performance
ggplot(data = all_model, aes(x = reorder(model, test_rmsle),test_rms
le))+
  geom_bar(stat = "identity", fill = "blue")+
  coord_flip()+
  ggtitle("Compare Performance of all used Models")+
  xlab("Models")+
  ylab("Root Mean Square Log Error")
```

Thank You