Predicting Bike Count

Chintan Shah

Introduction

The aim of the project is to predict count of bike of using different attributes

1.1 Problem statement

The aim of the project is to predict count of bike of using different attributes

1.2 Data

Our task is to predict to count of bike rental on daily basis Sample of dataset is given below

Table 1.1 Data sample (Column 1: 9)

instant	dteday	season	yr	mnth	holiday	weekday	workingda	weathers
1	01-01-11	1	0	1	0	6	0	2
2	02-01-11	1	0	1	0	0	0	2
3	03-01-11	1	0	1	0	1	1	1
4	04-01-11	1	0	1	0	2	1	1
5	05-01-11	1	0	1	0	3	1	1
6	06-01-11	1	0	1	0	4	1	1
7	07-01-11	1	0	1	0	5	1	2
8	08-01-11	1	0	1	0	6	0	2

Table 1.2 Data Sample

temp	atemp	hum	windspee	casual	registerec	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600
0.204348	0.233209	0.518261	0.089565	88	1518	1606
0.196522	0.208839	0.498696	0.168726	148	1362	1510
0.165	0.162254	0.535833	0.266804	68	891	959

There are 17 attributes in the table by using these we can predict the count of bikes.

Table 1.3. Predictor

No	Predictor
1	Instant
2	Dtedat
3	Season
4	Yr
5	Mnth
6	Hr
7	Holiday
8	Weekday
9	Workingday
10	Weathersit
11	Temp
12	Atemp
13	Hum
14	Windspeed
15	Casual
16	Registered
17	cnt

Chapter - 2

Methodology

2.1 Preprocessing

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable

2.1.1. Missing values

As per dataset, there is no missing values. So there is no point to fill missing values. Please find screenshot for details.

```
In [5]: dataset.isnull().sum()
Out[5]:
instant       0
dteday        0
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64
```

2.1.2 Backward elimination for feature selection

In dataset, there are many attributes, which are not useful for prediction. Because it may possible that few attributes are same in value. Therefore, it does not provide much impact on prediction. Therefore, train the model with such attributes, there will no meaning so it is better to remove attributes.

For that, we have to set significant level to 0.05. In addition, we will set two hypothesis, if attribute value is below SL then we will consider attribute otherwise we will reject null hypothesis.

There are few steps to do it.

- Select significance level
- Fit our model with all possible independent variables.
- Consider variable with highest p-value.
- If p-value is greater than significance level, remove variable
- Again, fit the model without removed variable.

Here, significance level and p-value are statistical terms. Just remember these terms for now as we do not want to go in details. Just note that our python libraries will provide us these values for our independent variables.

Now coming to our scenario, we want our salary predictions to be more accurate and we have to decide which independent variables to consider for making a final model.

```
import statsmodels.formula.api as sm
X = np.append(arr = np.ones((731, 1)).astype(int), values = X, axis = 1)
X_opt = X[:, [0, 1, 2, 3, 4, 5,6,7,8,9,10,11,12]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [2,3,7,8,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [ 1, 2,7,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [7,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()

X_opt = X[:, [12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Figure 1. Backward elimination for feature selection

Which will produce following result.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.892
Model:	OLS	Adj. R-squared:	0.890
Method:	Least Squares	F-statistic:	493.0
Date:	Fri, 07 Sep 2018	Prob (F-statistic):	0.00
Time:	18:21:19	Log-Likelihood:	-5757.0
No. Observations:	731	AIC:	1.154e+04
Df Residuals:	718	BIC:	1.160e+04
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	397.6375	182.209	2.182	0.029	39.911	755.364
x1	415.2598	40.516	10.249	0.000	335.715	494.804
x2	1602.4225	51.204	31.295	0.000	1501.894	1702.951
x3	-14.9098	12.618	-1.182	0.238	-39.683	9.863
x4	-99.7766	149.061	-0.669	0.503	-392.424	192.871
x5	28.7579	12.117	2.373	0.018	4.969	52.546
x6	1386.5780	73.852	18.775	0.000	1241.587	1531.569
x7	-438.1584	58.147	-7.535	0.000	-552.317	-324.000
x8	202.2540	1036.602	0.195	0.845	-1832.879	2237.387
x9	2205.2287	1172.064	1.881	0.060	-95.853	4506.310
x10	-417.6952	232.573	-1.796	0.073	-874.299	38.909
x11	-1239.6698	340.324	-3.643	0.000	-1907.820	-571.520
x12	1.5288	0.062	24.640	0.000	1.407	1.651

Figure 2. Backward elimination Statistics

2.1.3 Outlier Analysis

We can clearly observe from these probability distributions that most of the variables are skewed, for example, registered user, Casual users. The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data. We can see the effect of the skew in figures. The red vertical line represents the mean and we can see that the mean is slightly displaced from the position of the median. This is clearly the effect of outliers and extreme values.

One of the other steps of pre-processing apart from checking for normality is the presence of outliers. In this case we use a classic approach of removing outliers, **RobustScaler**.

The RobustScaler uses a similar method to the Min-Max scaler but it instead uses the interquartile range, rathar than the min-max, so that it is robust to outliers. Therefore, it follows the formula:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

For each feature.

Of course this means it is using the less of the data for scaling so it's more suitable for when there are outliers in the data.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

Centering and scaling happen independently on each feature (or each sample, depending on the axis argument) by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the transform method.

Standardization of a dataset is a common requirement for many machine-learning estimators. Typically, this is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean / variance in a negative way. In such cases, the median and the interquartile range often give better results.

Code snippet

```
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
Xtr_s = standard_scaler.fit_transform(X_train)
Xte_s = standard_scaler.transform(X_test)

from sklearn.preprocessing import RobustScaler
robust_scaler = RobustScaler()
X_train = robust_scaler.fit_transform(X_train)
X_test = robust_scaler.transform(X_test)
```

Figure 3. StandardScaler & RobustScaler code

After perform RobustScaler, we will visualize the result with following code

```
# Plot data
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
ax[0].scatter(X_train[:, 0], X_train[:, 1],
              color=np.where(y_train > 0, 'r', 'b'))
ax[1].scatter(Xtr_s[:, 0], Xtr_s[:, 1], color=np.where(y_train > 0, 'r', 'b'))
ax[2].scatter(X_train[:, 0], X_train[:, 1],color=np.where(y_train > 0, 'r', 'b'))
ax[0].set_title("Unscaled data")
ax[1].set_title("After standard scaling (zoomed in)")
ax[2].set_title("After robust scaling (zoomed in)")
# for the scaled data, we zoom in to the data center (outlier can't be seen!)
for a in ax[1:]:
    a.set_xlim(-3, 3)
    a.set_ylim(-3, 3)
plt.tight_layout()
plt.show()
```

Figure 4. Visualization of standard scaling & robust scaling

Above code will produce following result.

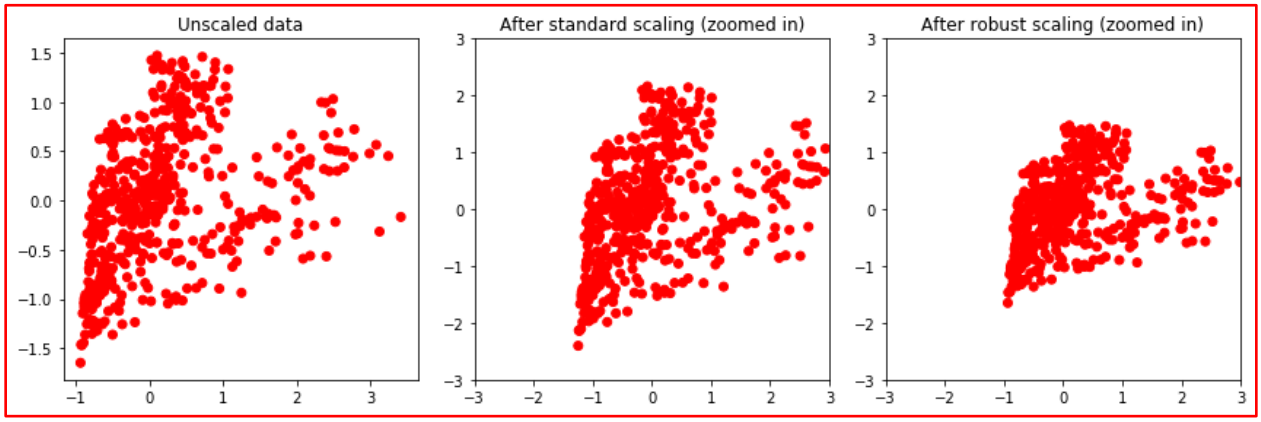


Figure 5. Visualization

2.2 Modeling

2.2.1 Model Selection

In our early stages of analysis during pre-processing we have come to understand that we have to predict number of bikes

The dependent variable can fall in any of the four categories:

1. Nominal

2. Ordinal

3. Interval

4. Cardinal

If the dependent variable, in our case is that number of bikes, is Cardinal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio the normal method is to do a Regression analysis, or classification after binning.

2.2.2 Logistic Regression

Regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Code

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
logi_y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
Logi_acc_score=accuracy_score(y_test, logi_y_pred)
from sklearn.metrics import r2_score
r2_score(y_test,logi_y_pred)
```

Figure 6. Training on LogisticRegression

Logistic Regression Result

After train our model as per above code, we use X-test to predict the result. So we compare to X-test and prediction result. We come to know that it is wide difference between actual and predicted result. Also we have tested r2 score for performance measure for same.

Please find result of predicted and actual result

Result Logistic Regression

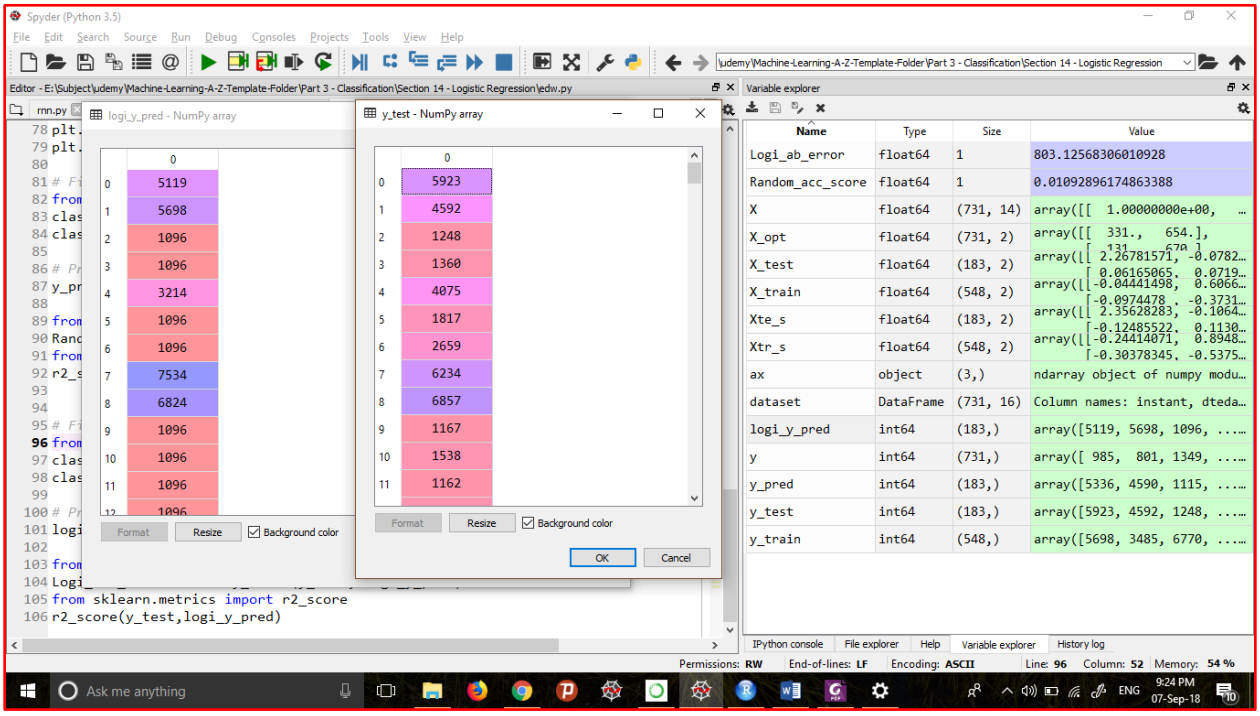


Figure 7. Comparison of LogisticRegression predictioin and actual set

As you can see the *Adjusted R-squared* value, we can explain only about 76% of the data using our multiple linear regression model. This is not very impressive.

Therefore, we have used another approach that is random forest classifier to predict the result.

2.2.3 Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

Code

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier_random = RandomForestClassifier(n_estimators = 10,
                                           criterion = 'entropy'
                                           , random_state = 0)
classifier_random.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier_random.predict(X_test)

from sklearn.metrics import accuracy_score
Random_acc_score=accuracy_score(y_test, y_pred)
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

Figure 8. RandomForest training on training set

Random Forest Result

After train our model as per above code, we use X-test to predict the result. So we compare to X-test and prediction result. We come to know that it is not much difference between actual and predicted result. Also we have tested r2 score for performance measure for same.

Please find result of predicted and actual result

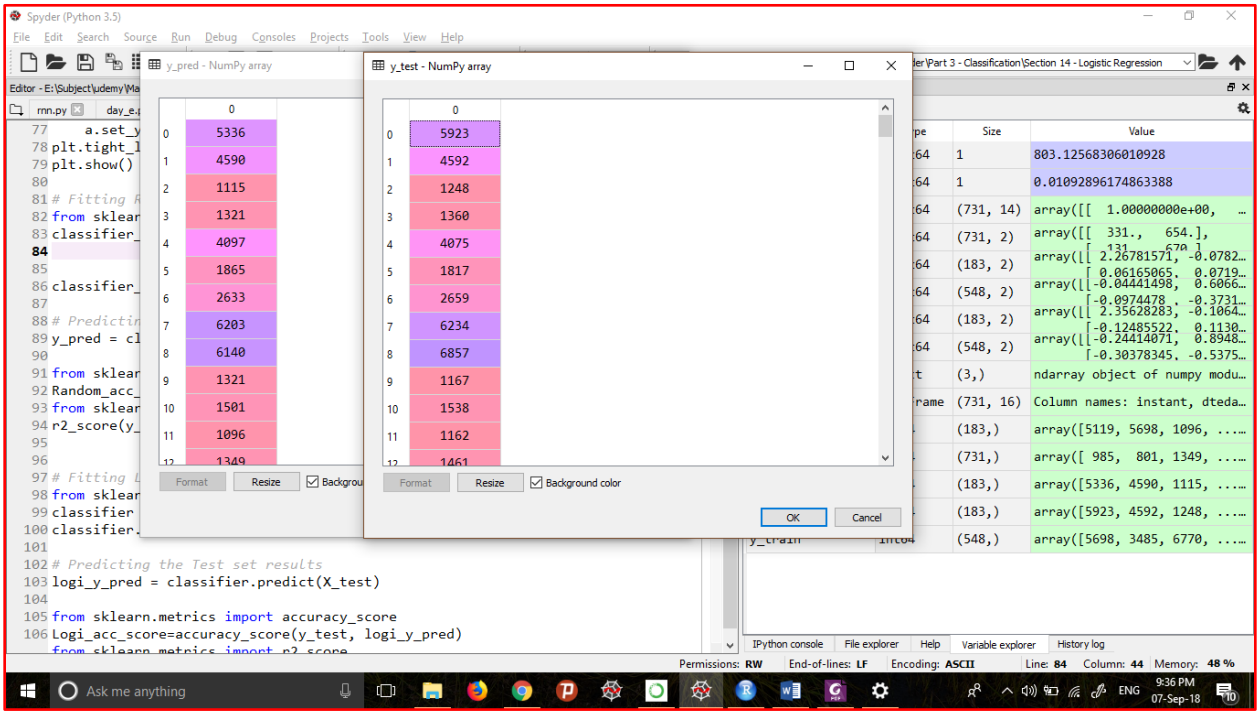


Figure 9. Comparison of RandomForest predictioin and actual set

2.2.4 Comparison

Comparison between Logistic Regression, Random Forest and Actual result

Figure 10. Comparison

Chapter -3

Model Evaluation

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

1. Predictive Performance

2. Interpretability

3. Computational Efficiency

3.1. Mean Absolute Error (MAE)

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our models that we have generated in the previous section.

Random Forest MAE

```
from sklearn.metrics import mean_absolute_error
random_m_a_e=mean_absolute_error(y_test, y_pred)
```

Answer: -

random_m_a_e	float64	1	131.04918032786884

Figure 11. Random Forest MAE & Result

Logistic Regression MAE

```
from sklearn.metrics import mean_absolute_error
logi_m_a_e=mean_absolute_error(y_test, logi_y_pred)
```

Answer: -

logi_m_a_e	float64	1	803.12568306010928

Figure 12. Logistic Regression MAE & Result

3.2 Adjsuted R2

Adjusted R2 is defined as follows

Logistic Regression R2

```
from sklearn.metrics import r2_score
Logi_r=r2_score(y_test,logi_y_pred)
```

Answer: -

Logi_r	float64	1	0.76714367021566621

Figure 13. Logistic Regression R2 & Result

Random Forest R2

```
from sklearn.metrics import r2_score
rand_r=r2_score(y_test,y_pred)
```

Answer:

rand_r	float64	1	0.9894372998716604

Figure 14. Random Forest R2 & Result

Appendix –A Python Code

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('day.csv')
X = dataset.iloc[:, 2:15].values
y = dataset.iloc[:, -1].values
# Building the optimal model using Backward Elimination
import statsmodels.formula.api as sm
X = np.append(arr = np.ones((731, 1)).astype(int), values = X, axis = 1)
X_opt = X[:, [0, 1, 2, 3, 4, 5,6,7,8,9,10,11,12]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [2,3,7,8,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [ 1, 2,7,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [7,12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [12,13]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

```
# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_opt, y, test_size = 0.25,
                                                    random_state = 0)
# Scale data
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
Xtr_s = standard_scaler.fit_transform(X_train)
Xte_s = standard_scaler.transform(X_test)

from sklearn.preprocessing import RobustScaler
robust_scaler = RobustScaler()
X_train = robust_scaler.fit_transform(X_train)
X_test = robust_scaler.transform(X_test)

# Plot data
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
ax[0].scatter(X_train[:, 0], X_train[:, 1],
              color=np.where(y_train > 0, 'r', 'b'))
ax[1].scatter(Xtr_s[:, 0], Xtr_s[:, 1], color=np.where(y_train > 0, 'r', 'b'))
ax[2].scatter(X_train[:, 0], X_train[:, 1],color=np.where(y_train > 0, 'r', 'b'))
ax[0].set_title("Unscaled data")
ax[1].set_title("After standard scaling (zoomed in)")
ax[2].set_title("After robust scaling (zoomed in)")
# for the scaled data, we zoom in to the data center (outlier can't be seen!)
for a in ax[1:]:
    a.set_xlim(-3, 3)
    a.set_ylim(-3, 3)
plt.tight_layout()
plt.show()
```

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier_random = RandomForestClassifier(n_estimators = 10,
                                           criterion = 'entropy'
                                           , random_state = 0)
classifier_random.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier_random.predict(X_test)

from sklearn.metrics import mean_absolute_error
random_m_a_e=mean_absolute_error(y_test, y_pred)
from sklearn.metrics import r2_score
rand_r=r2_score(y_test,y_pred)


# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
logi_y_pred = classifier.predict(X_test)

from sklearn.metrics import mean_absolute_error
logi_m_a_e=mean_absolute_error(y_test, logi_y_pred)
from sklearn.metrics import r2_score
Logi_r=r2_score(y_test,logi_y_pred)
```

Variable & Value

Logi_r	float64	1	0.76714367021566621
Random_acc_score	float64	1	0.01092896174863388
X	float64	(731, 14)	array([[1.00000000e+00, 1.00…
X_opt	float64	(731, 2)	array([[331., 654.], [131., 670.]
X_test	float64	(183, 2)	array([[2.26781571, -0.07823244… [0.06165065, 0.07191974…
X_train	float64	(548, 2)	array([[-0.04441498, 0.60669598… [-0.0974478 , -0.37312592…
Xte_s	float64	(183, 2)	array([[2.35628283, -0.10642452… [-0.12485522, 0.11306883…
Xtr_s	float64	(548, 2)	array([[-0.24414071, 0.8948079 … [-0.30378345, -0.53750154…
ax	object	(3,)	ndarray object of numpy module
dataset	DataFrame	(731, 16)	Column names: instant, dteday, s…
logi_m_a_e	float64	1	803.12568306010928
logi_y_pred	int64	(183,)	array([5119, 5698, 1096, ..., 75…
rand_r	float64	1	0.9894372998716604
random_m_a_e	float64	1	131.04918032786884
y	int64	(731,)	array([985, 801, 1349, ..., 13…
y_pred	int64	(183,)	array([5336, 4590, 1115, ..., 73…
y_test	int64	(183,)	array([5923, 4592, 1248, ..., 72…
y_train	int64	(548,)	array([5698, 3485, 6770, ..., 81…

Appendix – B R – Code

```
rm(list=ls(all=T))
setwd("E:/Subject/edwisor")

# Importing the dataset

dataset = read.csv('day.csv')
dataset1 = dataset[3:16]

# Building the optimal model using Backward Elimination

regressor = lm(formula = cnt ~
      season+ yr+mnth+holiday+weekday+workingday+weathersit+temp+atemp+hum+windspeed+casual+registered,
               data = dataset1)
summary(regressor)

regressor = lm(formula = cnt ~ casual+registered,
               data = dataset1)
summary(regressor)

final_dataset=regressor[["model"]]
final_dataset1=as.vector(final_dataset[2:3])

#Ploting outliers
boxplot(final_dataset1$casual,horizontal = T)
hist(final_dataset1$casual)

boxplot(final_dataset1$registered,horizontal = T)
```

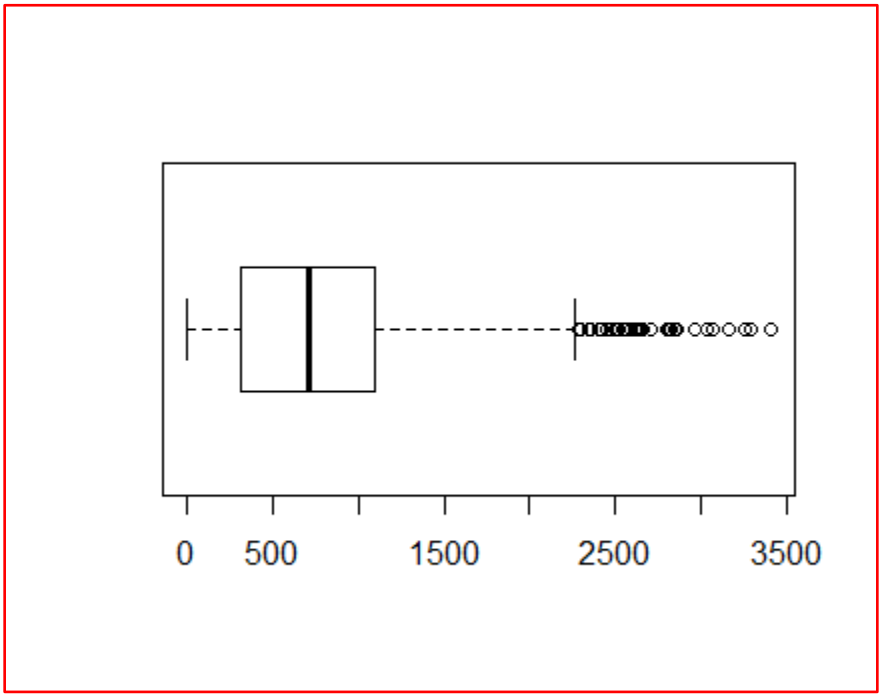
```
quantiles =quantile(final_dataset$casual, probs = c(.25, .75))
range =1.5 * IQR(final_dataset$casual)
pre_process_data = subset(final_dataset,
                      final_dataset$casual > (quantiles[1] - range) &
                        final_dataset$casual < (quantiles[2] + range))

boxplot(pre_process_data,horizontal = T)
hist(pre_process_data$casual)

#change order of column number
pre_process_data=pre_process_data[c(2,3,1)]


# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(pre_process_data$cnt, SplitRatio = 0.8)
training_set = subset(pre_process_data, split == TRUE)
test_set = subset(pre_process_data, split == FALSE)

# Feature Scaling
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
```

```
# Fitting Random Forest Classification to the Training set
#install.packages('randomForest')
library(randomForest)
set.seed(123)
classifier = randomForest(x = training_set[-3],
                          y = training_set$cnt,
                          ntree = 500)

# Predicting the Test set results

y_pred = data.frame(predict(classifier, newdata = test_set[-3]))
colnames(y_pred)=c("Prediction")

#Merge test case and prediction to visualize result
res=data.frame(test_set$cnt,round(y_pred,digits=0))

colnames(res)=c("test_case","prediction")
MAE(y_test,y_pred)
```

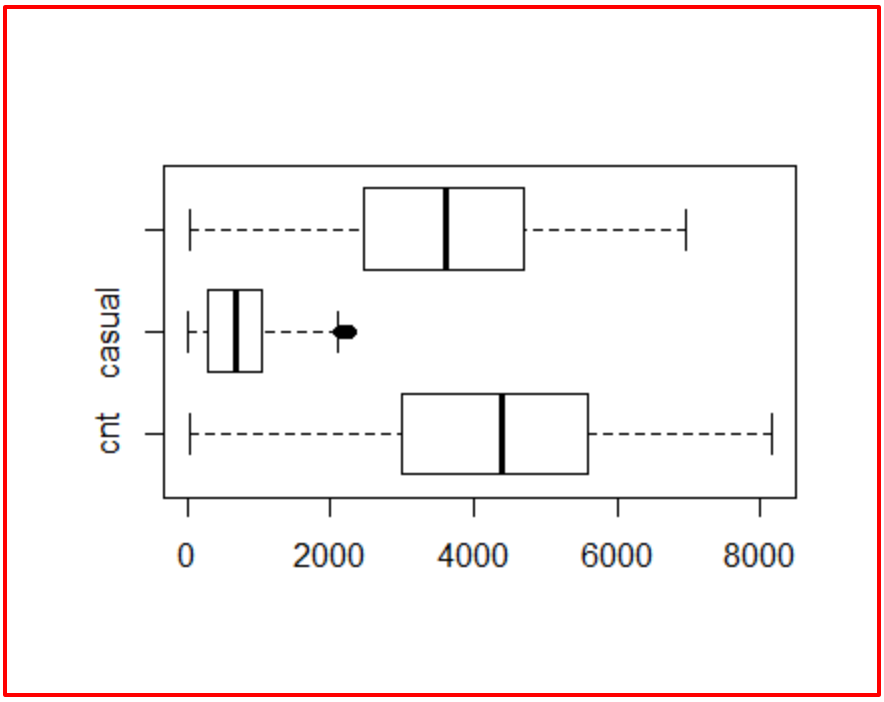
Appendix C -

Box plot for outlier in R

Box plot for casual user before outlier



Box plot after outlier remove


cnt
casual
0
2000
4000
6000
8000