Bike Renting

*Namrata Srivastava*

**Contents**

**Introduction**

**1.1 Problem Statement**

**1.2 Data**

**1.3 Methodology**

**2.1 Pre Processing**

**2.1.1 Missing Value Analysis**

**2.1.2 Outlier Analysis**

**2.1.3 Feature Scaling**

**2.1.4 Feature Selection**

**2.1.5 Sampling**

**2.2 Modelling**

**2.2.1 Method Selection**

**2.2.2 Decision Tree**

**2.2.3 Random Forest**

**2.2.4 Linear Regression**

**2.2.5 KNN**

**Conclusion**

**3.1 Model Evaluation**

**3.2 MAPE**

**3.3 Model Selection**

**Appendix**

**A – Python Code**

**B – R Code**

**References**

1

**Chapter 1**

**Introduction**

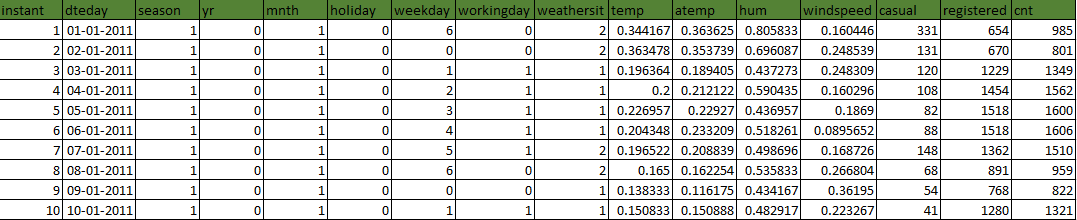
**1.1 Problem Statement**

Predicting the count of bikes renting based on given conditions. The aim of the project is to reduce man in analysing the business and making profit in it We would like to predict the count of rented bike based on days and weather conditions which are already known and easy to calculate using sensors.

**1.2 Data**

Our task is to build regression models which will calculate the count of bike to be rented depending on multiple weather factors. Given below is a sample of the data set that we are using to predict the count of bike to be rented:

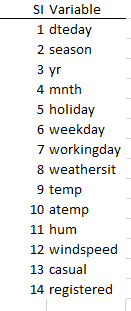
Table 1.1: Bike Renting Sample Data



2

As you can see in the table below we have the following 15 variables, using which we have to correctly predict the bikes to be rented:

Table 1.5: Predictor Variables



3

**Chapter 2**

**Methodology**

**2.1** **Pre Processing**

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**.

**2.1.1 Missing Value Analysis**

We will check for the percentage of missing value in our data.

Python :

missing\_val = pd.DataFrame(data.isnull().sum())

# print(missing\_val)#No missing data

R :

#Missing Value Analysis

missing\_per = data.frame(apply(data,2,function(x){sum(is.na(x))}))#No missing value

As you can see missing percentage is 0. So we don’t have any missing value and no actions to be taken further.

**2.1.2 Outlier Analysis**

**Python**: missing\_val\_after\_outlier is empty means no outlier.

for name in cnames:

q75,q25 = np.percentile(data.loc[:,name],[75,25])

iqr = q75-q25

min=q25-(iqr\*1.5)

max = q75 + (iqr \* 1.5)

data=data.drop(data[data.loc[:,name]<min].index)

data = data.drop(data[data.loc[:, name] > max].index)

missing\_val\_after\_outlier = pd.DataFrame(data.isnull().sum())

# print(missing\_val\_after\_outlier)#No missing value means no outlier

**R :** var is empty means no outlier

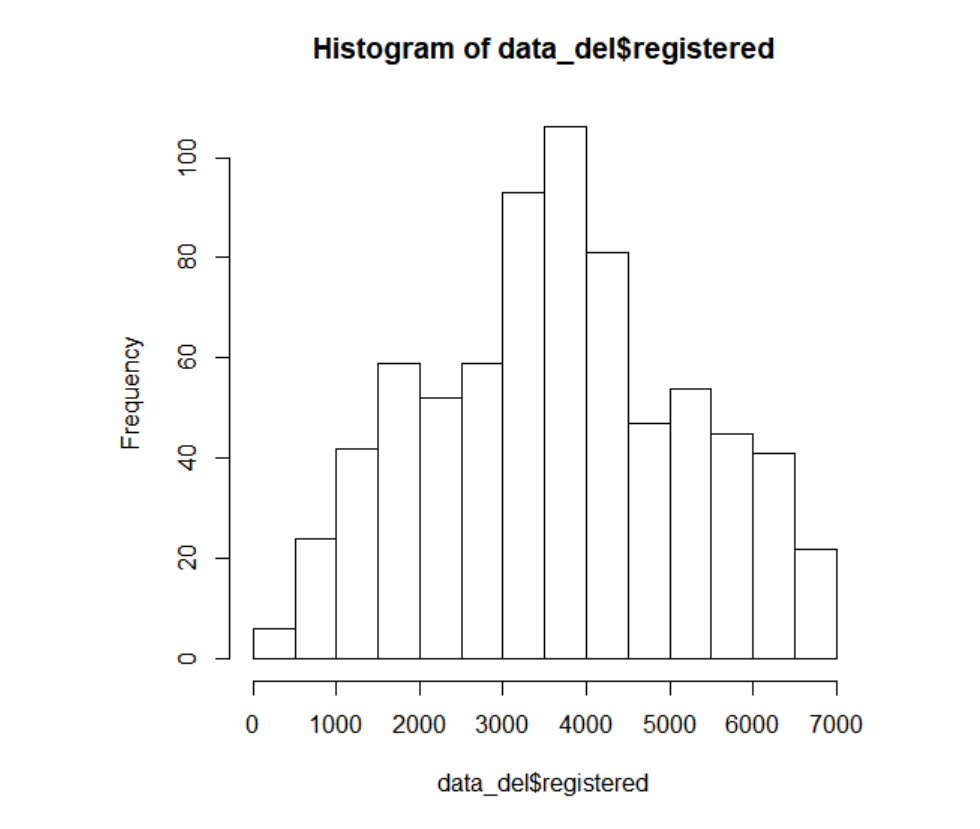
Var = data$previous[data$previous%in%boxplot.stats(data$previous)$out]

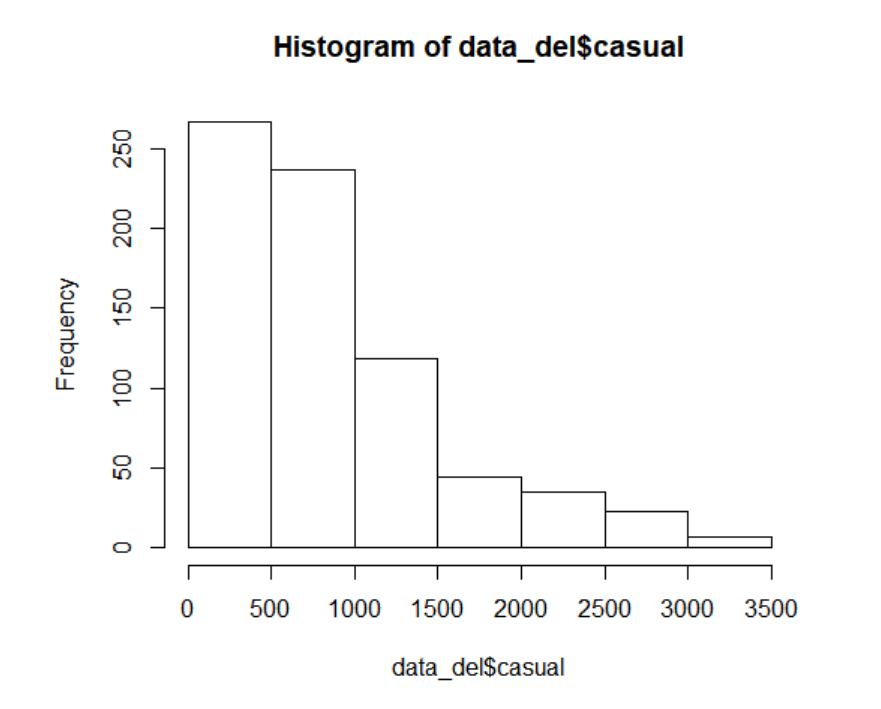
#Null

9

**2.1.3 Feature Scaling**

As we can already see there are only two numerical column that needs to scaled. Those are “casual” and “registered”. For selecting the method for scaling we need to check uniform distribution of data.

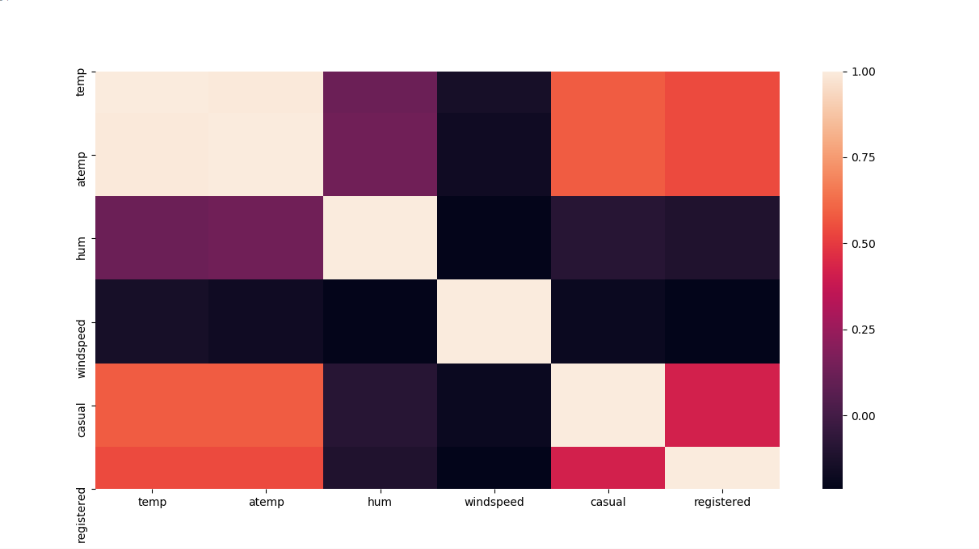




As we can see casual is left- skewed and registered is uniformly distributed. So we will apply Normalization to casual and Standardization to registered**.**

**2.1.4 Feature Selection**

Before performing any type of modeling we need to assess the correlation of among the0 predictor variable in our analysis. There is a possibility that the variables in our analysis are highly positively correlated.



10

As we can see that temp and atemp are highly positively correlated with each other, so we will remove atemp from the data.

Also dteday is the date column and as we already know that models do not include date data types and some columns are already derived from dteday like season, weekday, yr, month etc. So we will remove dteday from the data.

**2.1.5 Sampling :**

We need to divide our data in two parts that is train and test data, Train Data will be used for training the model and test data will be used for testing the model performance.

There are different methods for Sampling like :

1. Simple Random Sampling
2. Systematic Sampling
3. Stratified Sampling

But we are using Stratified Sampling because in our data we g=have categorical variable and to make sure that the train and test data is not biased to any particular category we are creating strata for the column holiday.

**Python :**

#Stratified sampling

#select categorical variable

cat\_var = data['holiday']

#select subset

train,test= train\_test\_split(data,test\_size=0.3,stratify=cat\_var)

**R :**

#Stratified Sampling Technique

set.seed(1234)

train.index = createDataPartition(data\_del$holiday,p=0.70,list = FALSE)

train= data\_del[train.index,]

test = data\_del[-train.index,]

We are using ratio of 7:3 for train and test data.

11

**2.2 Modeling**

**2.2.1 Method**

In our early stages of analysis during pre-processing we have come to understand that red wine and white wine have completely diﬀerent chemical behaviours. Therefore, we can neither combine the data sets nor use a single model for predicting both variables. Hence, we need to analyse the data sets separately and generate separate models for both types of data sets.

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

1. Nominal
2. Ordinal
3. Interval
4. Ratio

If the dependent variable, in our case *cnt*, is Nominal 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. But the dependent variable we are dealing with is *Numerical*, for regression can be done.

**2.2.2 Decision Tree**

**Python :**

#Decision Tree

fit = sklearn.tree.DecisionTreeRegressor(max\_depth=500).fit(train.iloc[:,0:13],train.iloc[:,13])

# summary(fit)

predict\_DT = fit.predict(test.iloc[:,0:13])

# print(predict\_DT)

**R :**

#Decision Tree

fit = rpart(cnt~.,data = train, method = "anova")

summary(fit)

predict\_DT = predict(fit,test[,-14]

**2.2.3** **Random Forest**

**Python :**

#Random Forest

regr = RandomForestRegressor(n\_estimators=500,max\_depth=2,random\_state=0)

regr.fit(train.iloc[:,0:13],train.iloc[:,13])

predict\_RF = regr.predict(test.iloc[:,0:13])

**R :**

#Random Forest

RF = randomForest(cnt~.,train,ntree = 500)

predict\_RF = predict(RF,test[,-14])

**2.2.4** **Linear Regression**

**Python :**

#linear regression

model = sm.OLS(train.iloc[:,13],train.iloc[:,0:13]).fit()

predict\_LR = model.predict(test.iloc[:,0:13])

**R :**

#Linear Regression

LM = lm(cnt~.,data = train)

summary(LM)

predict\_LM = predict(LM,test[,-14])

**2.2.5 KNN**

Python :

model = neighbors.KNeighborsRegressor(n\_neighbors = 8)

model.fit(train.iloc[:,0:13], train.iloc[:,13]) #fit the model

predict\_KNN=model.predict(test.iloc[:,0:13]) #make prediction on test set

I have tried different values of K and found minimum error at k =8.

14

**Chapter 3**

**Conclusion**

**3.1 Model Evaluation**

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

1. Predictive Performance
2. Interpretability
3. Computational Eﬃciency

In our case of Wine Data, the latter two, *Interpretability* and *Computation Eﬃciency*, do not hold much significance. Therefore we will use *Predictive performance* as the criteria to compare and evaluate models.

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

**3.1.1 Mean Absolute Percentage Error (MAPE)**

MAPE 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.

**Python :**

# # #MAPE FUnction

def MAPE(test\_data, pred):

mape = np.mean(np.abs((test\_data-pred)/test\_data))

return mape

**R :**

#MAPE

mape = function(x, xpred){

mean(abs(x-xpred)/x)

}

**3.2 Model Selection**

We can see that both models perform comparatively on average and therefore we can select either of the two models without any loss of information.

**Python :**

print(MAPE(test.iloc[:,13],predict\_DT))

#error for Decision Tree : 0.03%

print(MAPE(test.iloc[:,13],predict\_RF))

#error for Random Forest : 0.53%

print(MAPE(test.iloc[:,13],predict\_LR))

#error for Linear Regression : 0.07%

print(MAPE(test.iloc[:,13],predict\_KNN))

#error for KNN Algorithm = 0.22%

**R** :

mape(test[,14],predict\_DT)

#Error Rate for Decision Tree is 0.12%

mape(test[,14],predict\_RF)

#Error Rate Random Forest 0.06%

mape(test[,14],predict\_LM)

#Error Rate Linear Regression is 1.23%

mape(test[,14],predict\_knn)

#Error Rate for KNN Algorithm is 0.06%

As we can see the error rate for all the algorithms are very less so any of them can be selected. But if we want to choose in Python Decision Tree works best and for R Random Forst and KNN works better. This is because the sample data for both R and Python is different.

So we will select either Decision Tree or Random Forest. We haven’t selected KNN because KNN is worst performing in case of Python.

16

**Appendix A – Python Code**

import os

import pandas as pd

import numpy as np

import matplotlib

import seaborn as sns

from matplotlib import pyplot as plt

import datetime

import sklearn

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

import sklearn.tree

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestRegressor

import statsmodels.api as sm

from sklearn import neighbors

# %matplotlib inline

data = pd.read\_csv("day.csv", sep = ',',encoding='unicode\_escape')

# print(data.shape)#

#Missing Data Analysis

missing\_val = pd.DataFrame(data.isnull().sum())

# print(missing\_val)#No missing data

cnames = ["temp", "atemp", "hum", "windspeed", "casual", "registered"]

for name in cnames:

data[name]= pd.to\_numeric(data[name],errors='coerce')

# #Outlier Analysis

# for names in cnames:

# plt.hist(data[names])

# plt.show()

for name in cnames:

q75,q25 = np.percentile(data.loc[:,name],[75,25])

iqr = q75-q25

min=q25-(iqr\*1.5)

max = q75 + (iqr \* 1.5)

data=data.drop(data[data.loc[:,name]<min].index)

data = data.drop(data[data.loc[:, name] > max].index)

missing\_val\_after\_outlier = pd.DataFrame(data.isnull().sum())

# print(missing\_val\_after\_outlier)#No missing value means no outlier

##Feature Selection

#Correlation analysis for numerical data

df\_cor=data.loc[:,cnames]

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

corr = df\_cor.corr()

sns.heatmap(corr)

# plt.show()

#temp and atemp are highly correlated

#removing atemp

data = data.drop('atemp',axis=1)

# print(data.shape)

#Dropping dteday as we already have columns that are derived from this column

data = data.drop('dteday', axis=1)

#Feature Scaling

#Checking for uniform distribution

fig, axs = plt.subplots(2, 1, constrained\_layout=True)

axs[0].hist(data['casual'])#non uniform

axs[0].set\_title('casual')

axs[1].hist(data['registered'])#uniform

axs[1].set\_title('registered')

# plt.show()

#normalization for casual beacuse it is left skewed

data['casual'] = (data['casual']-data['casual'].min())/(data['casual'].max()-data['casual'].min())

# print(data['casual'].head(5))

#Standardization for registered

data['registered']= (data['registered'] - data['registered'].mean())/data['registered'].std()

# print(data['registered'].head(5))

##Sampling

#Stratified sampling

#select categorical variable

cat\_var = data['holiday']

#select subset

train,test= train\_test\_split(data,test\_size=0.3,stratify=cat\_var)

# print(train.shape)

# print(test.shape)

# print(test.head(10))

#Decision Tree

fit = sklearn.tree.DecisionTreeRegressor(max\_depth=500).fit(train.iloc[:,0:13],train.iloc[:,13])

# summary(fit)

predict\_DT = fit.predict(test.iloc[:,0:13])

# print(predict\_DT)

# # #MAPE FUnction

def MAPE(test\_data, pred):

mape = np.mean(np.abs((test\_data-pred)/test\_data))

return mape

# print(MAPE(test.iloc[:,13],predict\_DT))

#error : 0.03%

#Random Forest

regr = RandomForestRegressor(n\_estimators=500,max\_depth=2,random\_state=0)

regr.fit(train.iloc[:,0:13],train.iloc[:,13])

predict\_RF = regr.predict(test.iloc[:,0:13])

print(MAPE(test.iloc[:,13],predict\_RF))

#error 0.53%

#linear regression

model = sm.OLS(train.iloc[:,13],train.iloc[:,0:13]).fit()

predict\_LR = model.predict(test.iloc[:,0:13])

print(MAPE(test.iloc[:,13],predict\_LR))

#error 0.07%

#KNN

#minimum error rate is at k = 8

model = neighbors.KNeighborsRegressor(n\_neighbors = K)

model.fit(train.iloc[:,0:13], train.iloc[:,13]) #fit the model

predict\_KNN=model.predict(test.iloc[:,0:13]) #make prediction on test set

print(MAPE(test.iloc[:,13],predict\_KNN))

#error = 0.22%

**Appendix B - R Code**

library(ggplot2)

library(corrgram)

library(sampling)

library(rpart)

library(caret)

library(randomForest)

library(LinearRegressionMDE)

library(FNN)

data = read.csv("day.csv")

#Missing Value Analysis

missing\_per = data.frame(apply(data,2,function(x){sum(is.na(x))}))#No missing value]

#Outlier Analysis

Var = data$previous[data$previous%in%boxplot.stats(data$previous)$out]

#Null

#Feature Selection

corrgram(data[,num\_index],order = F, upper.panel=panel.pie, text.panel=panel.txt, main = "correlation plot")

#temp and atemp are hoghly correlated, dropping atemp and we are removing date column as some columns are already extracted from it

data\_del = subset(data, select = -c(atemp,dteday))

#Feature Scaling

# Casual and registered are unscaled

#Normality check

qqnorm(data\_del$casual)

hist(data\_del$casual)#left or positive skewed

#Applying Normalization

data\_del[,"casual"]= (data\_del[,"casual"]-min(data\_del[,"casual"]))/(max(data\_del[,"casual"])-min(data\_del[,"casual"]))

qqnorm(data\_del$registered)

hist(data\_del$registered)# Normally Distributed

data\_del[,"registered"]=(data\_del[,"registered"]-mean(data\_del[,"registered"]))/sd(data\_del[,"registered"])

#Sampling

#Stratified Sampling Technique

set.seed(1234)

train.index = createDataPartition(data\_del$holiday,p=0.70,list = FALSE)

train= data\_del[train.index,]

test = data\_del[-train.index,]

#Data Modelling

#Decision Tree

fit = rpart(cnt~.,data = train, method = "anova")

summary(fit)

predict\_DT = predict(fit,test[,-14])

#MAPE

mape = function(x, xpred){

mean(abs(x-xpred)/x)

}

mape(test[,14],predict\_DT)#Error Rate 0.12%

#Random Forest

RF = randomForest(cnt~.,train,ntree = 500)

predict\_RF = predict(RF,test[,-14])

mape(test[,14],predict\_RF)#Error Rate 0.06%

#Linear Regression

LM = lm(cnt~.,data = train)

summary(LM)

predict\_LM = predict(LM,test[,-14])

mape(test[,14],predict\_LM)#Error Rate 1.23%

predict\_knn <- knn.reg(train = train, test = test,train$cnt, k=10)

mape(test[,14],predict\_knn)#Error Rate 0.06%

**References**

Google

29