Project Report – Bike Rental Count Prediction

By: Tarun Kumar Agarwal

Contents

Chapter 1 Introduction

1.1 Problem:

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

1.2 Data:

The details of data attributes in the dataset are as follows -

instant: Record index

dteday: Date

season: Season (1:springer, 2:summer, 3:fall, 4:winter)

yr: Year (0: 2011, 1:2012)

mnth: Month (1 to 12)

hr: Hour (0 to 23)

holiday: weather day is holiday or not (extracted from Holiday Schedule)

weekday: Day of the week

workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

weathersit: (extracted from Freemeteo)

1: Clear, Few clouds, Partly cloudy, Partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),

t_min=-8, t_max=+39 (only in hourly scale)

atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)

hum: Normalized humidity. The values are divided to 100 (max)

windspeed: Normalized wind speed. The values are divided to 67 (max)

casual: count of casual users

Chapter 2 Pre-Processing of Data

In this chapter, we manipulate the data before we start modeling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots.

2.1 Data exploration, Missing Values and Outlier analysis:

Firstly, we perform data exploration and cleaning which includes following points as per this project:

- Convert the data types into appropriate data types.
- Check the missing values in the data.

```
#Converting variables datatype to required datatypes

df['dteday'] = pd.to_datetime(df['dteday'],yearfirst = True)
df['season'] = df['season'].astype(str)
df['yr']     = df['yr'].astype(str)
df['mnth']   = df['mnth'].astype(str)
df['holiday']= df['holiday'].astype(str)
df['weekday']= df['weekday'].astype(str)
df['workingday']= df['workingday'].astype(str)
df['weathersit']= df['weathersit'].astype(str)
```

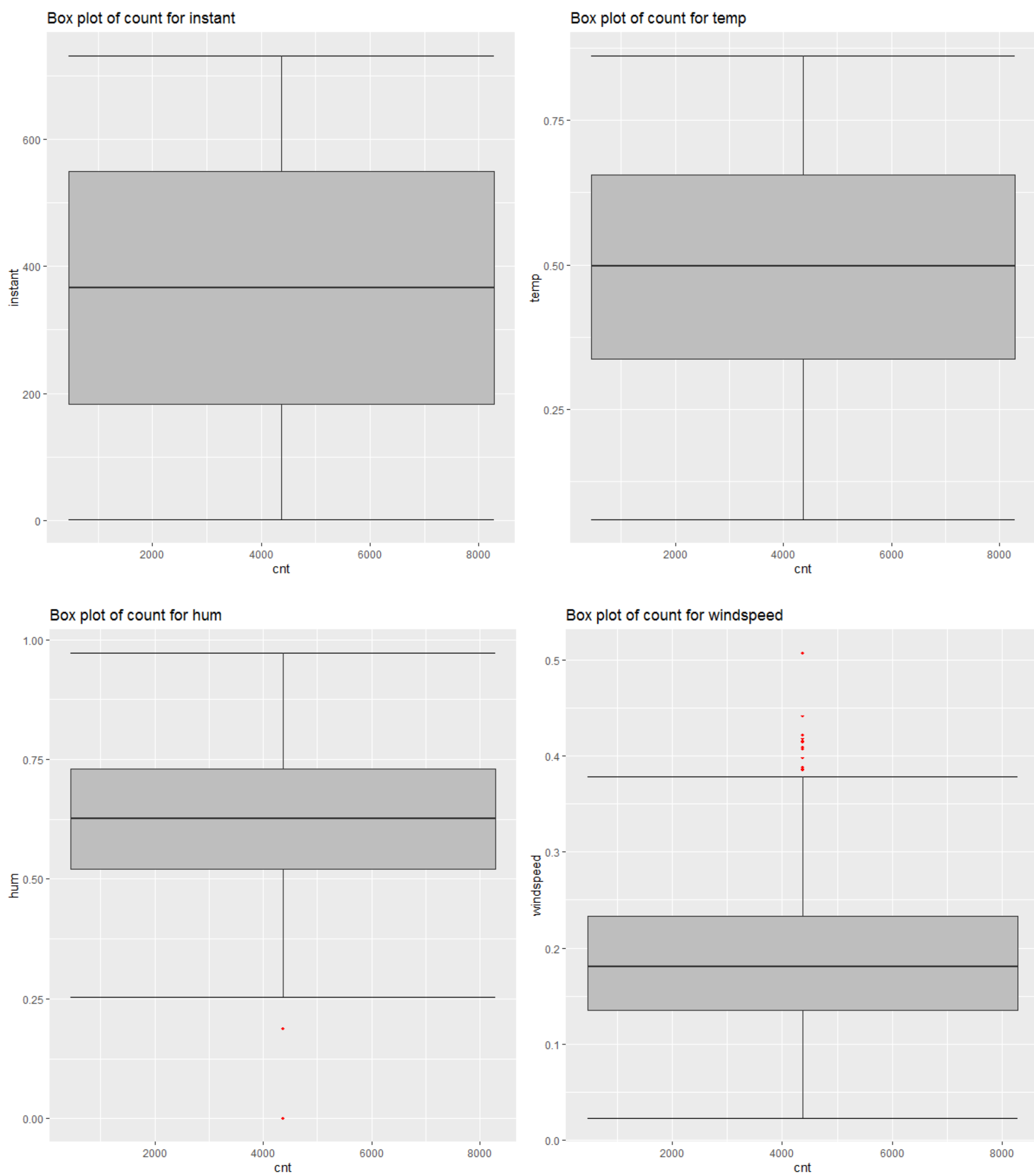
```
# checking missing values
df.isnull().sum()
```

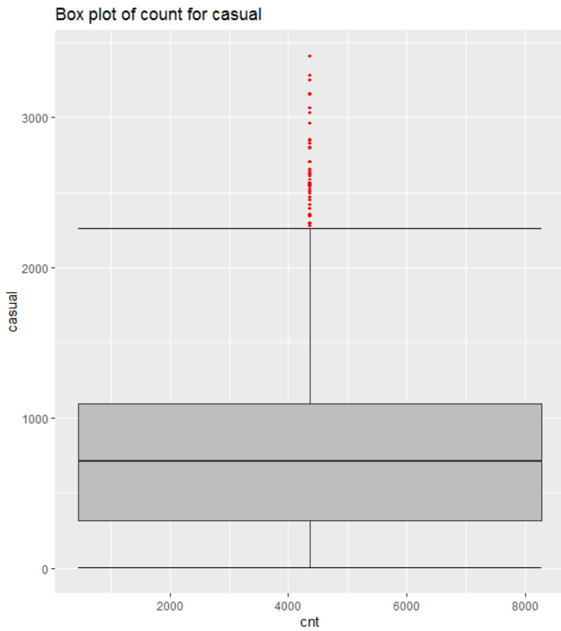
```
instant       0
dteday        0
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
```

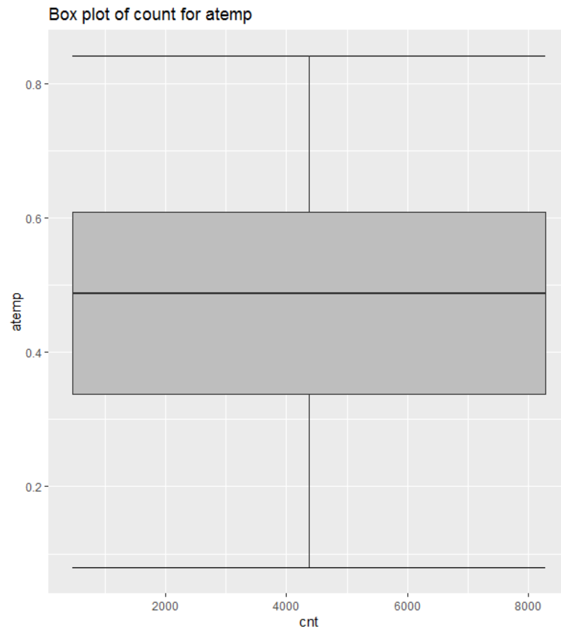
Now, we performed boxplot analysis on continuous variables to check the outlier.

```
# checking boxplot of continous variables

%matplotlib inline
sns.boxplot(data=df[['temp','atemp','windspeed','hum','casual','registered']])
fig=plt.gcf()
fig.set_size_inches(10,10)
```








```
# from the boxplot analysis, it is clear that continous variables windspeed, hum and casual includes the outliers.
# but we are not considering casual because this is not predictor variable.

count_names = ['windspeed','hum']
for i in count_names:
    print (i)
    q75,q25 = np.percentile(df.loc[:,i],[75,25])
    iqr = q75-q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    print (min)
    print (max)

    df.loc[df[i]<min,i]=np.nan
    df.loc[df[i]>max,i]=np.nan
```

```
# checking missing values
df.isnull().sum()
```

```
instant        0
dteday         0
season         0
yr             0
mnth           0
holiday        0
weekday        0
workingday     0
weathersit     0
temp           0
atemp          0
hum            2
windspeed     13
casual         0
registered     0
cnt            0
```

From boxplot, it is clear that hum includes 2 outliers and windspeed includes 13 outliers. So drop the outlier rows.

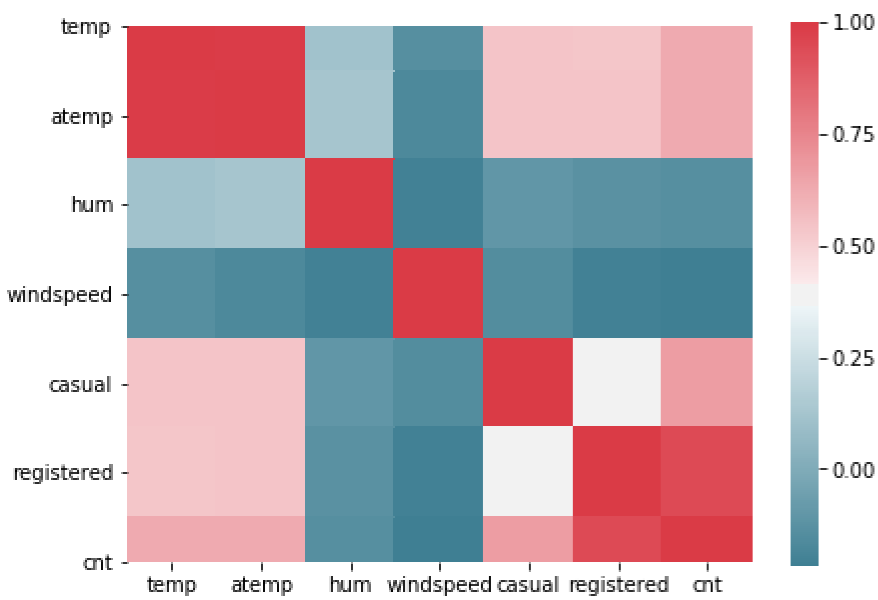
```
# hum includes 2 outlier and windspeed includes 13 outliers. so drop the outlier rows.

df = df.dropna(axis = 0)
```

2.2 Feature Selection and Scaling:

We also performed correlation matrix on continuous variables to check the multicolinerity.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3fd2e0acf8>
```



Also, we performed VIF on the data to check the multicolinerity.

```
# checking VIF for multicolinerity

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

VIF_df = add_constant(df.iloc[:,9:15])
pd.Series([variance_inflation_factor(VIF_df.values, i)
               for i in range(VIF_df.shape[1])],
              index=VIF_df.columns)
```

```
const          54.847289
temp           63.442490
atemp          64.309759
hum             1.179328
windspeed       1.154450
casual          1.502061
registered      1.561168
dtype: float64
```

We also performed chi-square test for categorical variables:

```
from scipy.stats import chi2_contingency

# making every combinationfrom cat_columns
factors_paired = [(i,j) for i in lis for j in lis]
factors_paired
p_values = []
from scipy.stats import chi2_contingency
for factor in factors_paired:
    if factor[0] != factor[1]:
        chi2, p, dof, ex = chi2_contingency(pd.crosstab(df[factor[0]], df[factor[1]]))
        p_values.append(p.round(3))
    else:
        p_values.append('-')
p_values = np.array(p_values).reshape((7,7))
p_values = pd.DataFrame(p_values, index=lis, columns=lis)
print(p_values)
```

	season	yr	mnth	holiday	weekday	workingday	weathersit
season	-	0.999	0.0	0.641	1.0	0.946	0.013
yr	0.999	-	1.0	0.995	1.0	0.956	0.183
mnth	0.0	1.0	-	0.571	1.0	0.993	0.01
holiday	0.641	0.995	0.571	-	0.0	0.0	0.599
weekday	1.0	1.0	1.0	0.0	-	0.0	0.249
workingday	0.946	0.956	0.993	0.0	0.0	-	0.294
weathersit	0.013	0.183	0.01	0.599	0.249	0.294	-

Observations:

- From heatmap and VIF, we remove variables atemp because it is highly correlated with temp.
- From chi-square test, we remove weekday, holiday variables because they don't contribute much to the independent variables,
- We remove causal and registered variables because that's what we need to predict.
- We remove instant and dteday variables because they are not useful in generating model.

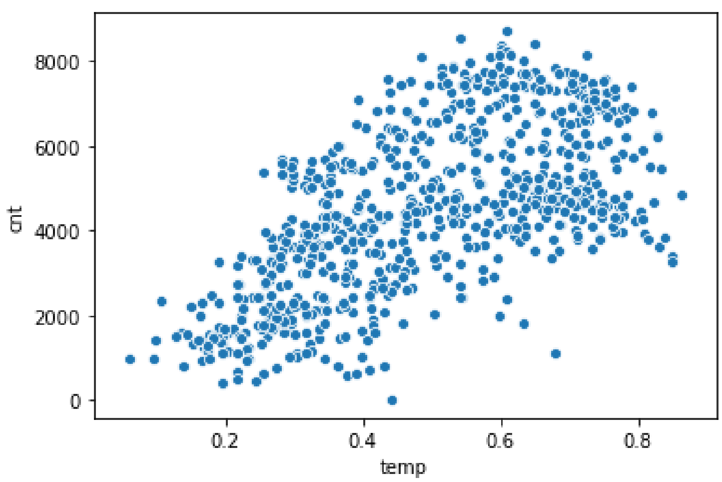
The cleaned data is look like:

	season	yr	mnth	workingday	weathersit	temp	hum	windspeed	cnt
0	1	0	1	0	2	0.344167	0.805833	0.160446	985
1	1	0	1	0	2	0.363478	0.696087	0.248539	801
2	1	0	1	1	1	0.196364	0.437273	0.248309	1349
3	1	0	1	1	1	0.200000	0.590435	0.160296	1562
4	1	0	1	1	1	0.226957	0.436957	0.186900	1600

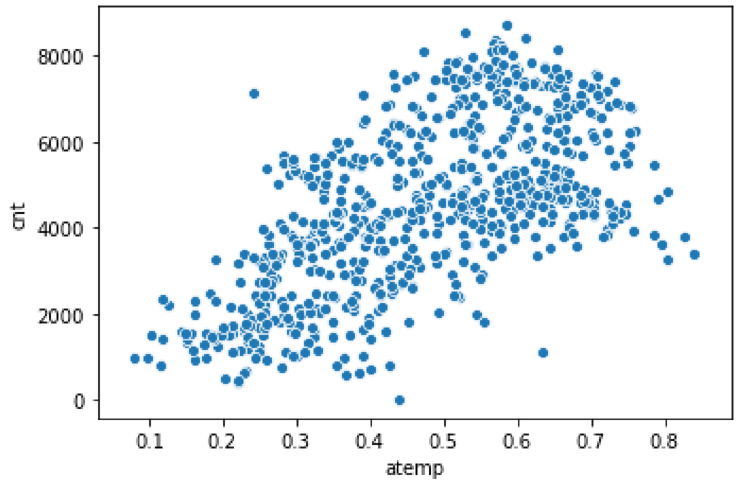
Visualization:

We also performed some visualization on the cleaned data.

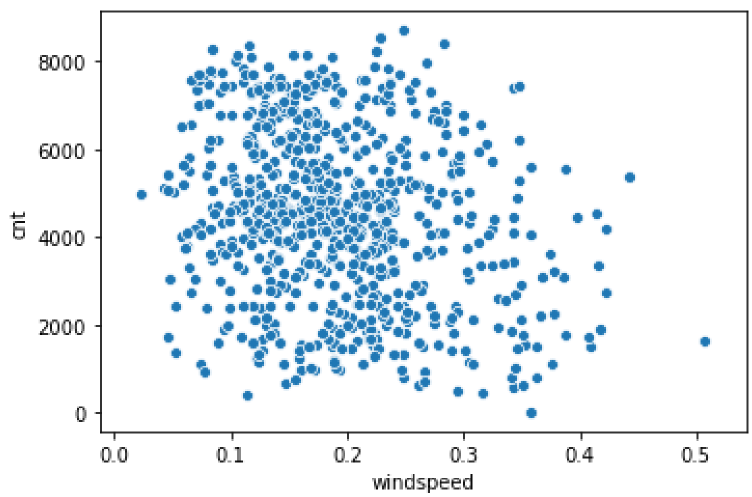
1. Scatter plot between temp and cnt:



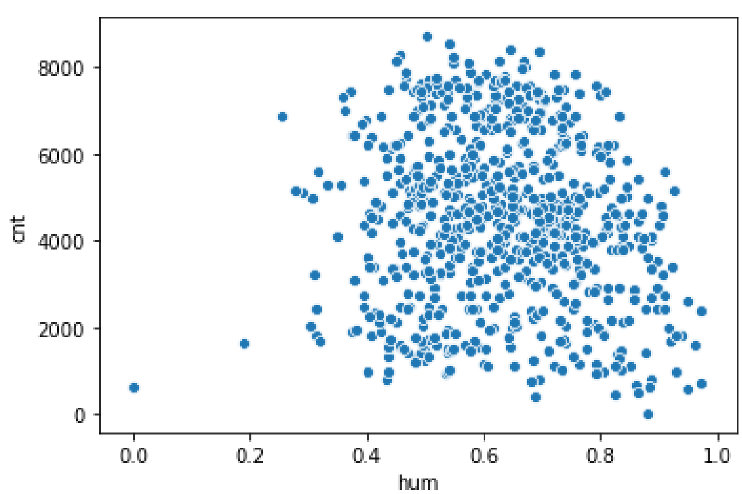
2. Scatter plot between atemp and cnt:



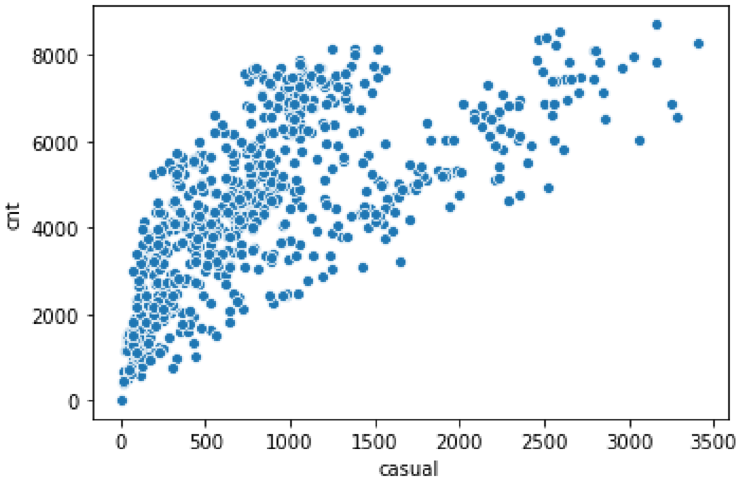
3. Scatter plot between windspeed and cnt:



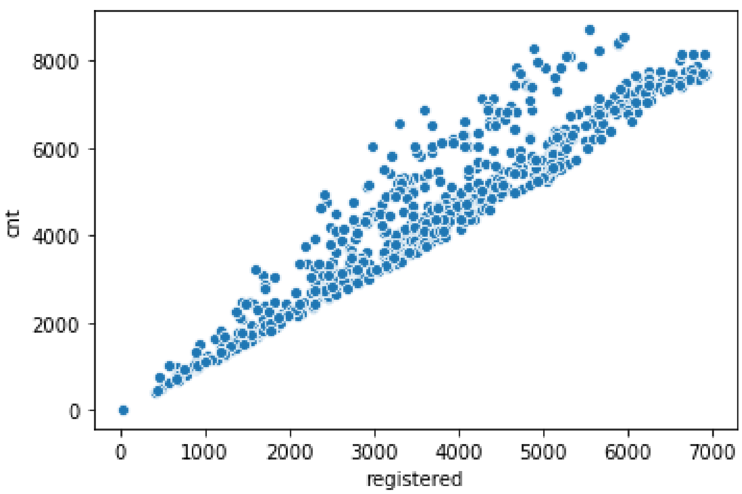
4. Scatter plot between casual and cnt:



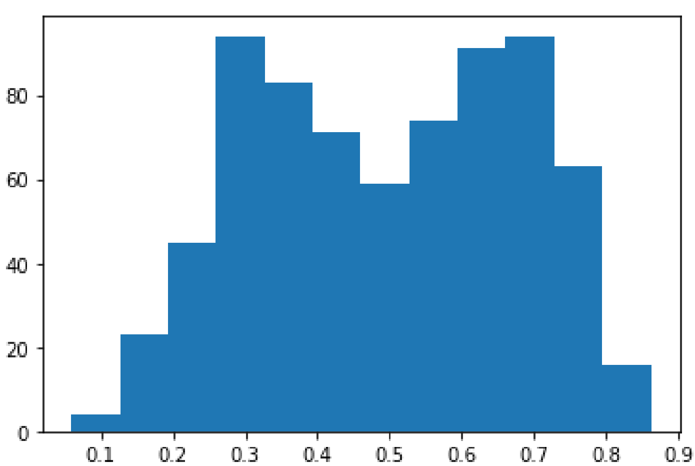
5. Scatter plot between hum and cnt:



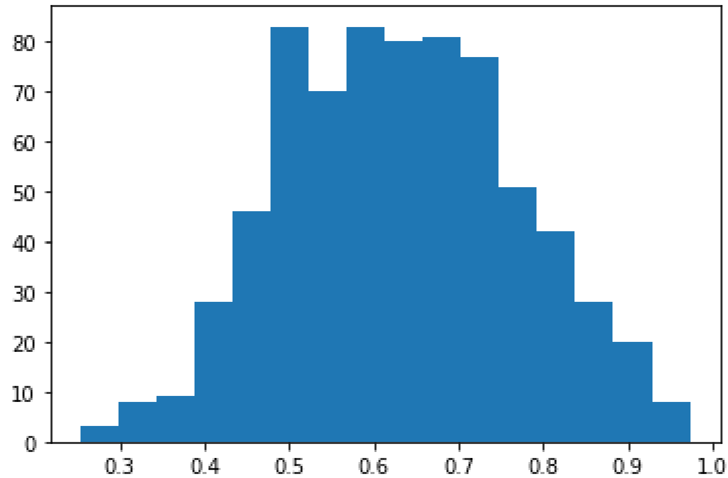
6. Scatter plot between registered and cnt:



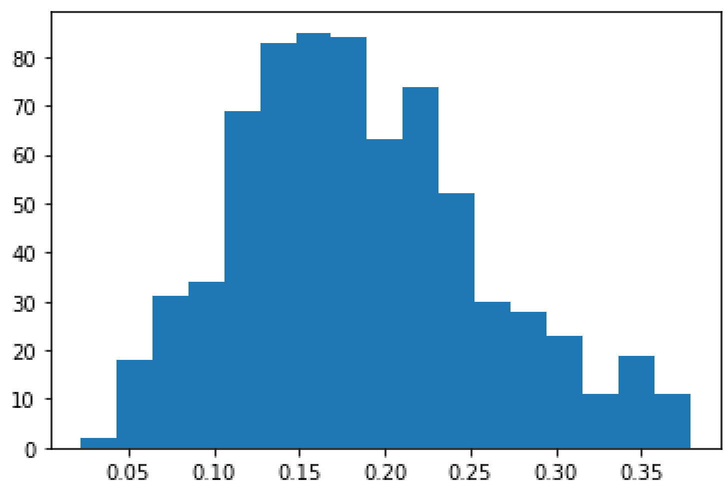
7. Histogram of temp variable:



8. Histogram of hum variable:



9. Histogram of windspeed variable:



Chapter 3 Model Development

In this chapter, we firstly split the cleaned dataset into train and test and then developed different models and also performed hyper-parameter tuning.

3.1 Linear Regression Model:

```
# Build model on train data
LR = LinearRegression().fit(X_train , y_train)
```

```
# predict on train data
pred_train_LR = LR.predict(X_train)
```

```
# predict on test data
pred_test_LR = LR.predict(X_test)
```

3.2 Decision Tree Model:

```
# Build model on train data
DT = DecisionTreeRegressor(max_depth = 2).fit(X_train, y_train)
```

```
# predict on train data
pred_train_DT = DT.predict(X_train)

# predict on test data
pred_test_DT = DT.predict(X_test)
```

3.3 Random Forest Model:

```
# Build model on train data
RF = RandomForestRegressor(n_estimators = 300).fit(X_train, y_train)
```

```
# predict on train data
pred_train_RF = RF.predict(X_train)

# predict on test data
pred_test_RF = RF.predict(X_test)
```

3.4 Gradient Boosting Model:

```
# Build model on train data
GB = GradientBoostingRegressor().fit(X_train, y_train)
```

```
# predict on train data
pred_train_GB = GB.predict(X_train)

# predict on test data
pred_test_GB = GB.predict(X_test)
```

3.5 Hyper-parameter Tuning:

There are two ways to apply hyper-parameter tuning:

1. RandomizedSearchCV
2. GridSearchCV

```
# 1. RandomizedSearchCV

from sklearn.model_selection import RandomizedSearchCV

# RandomizedSearchCV on Random Forest Model

RFR = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator, 'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RFR, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(X_train, y_train)
predictions_RFR = randomcv_rf.predict(X_test)

best_params_RFR = randomcv_rf.best_params_

best_estimator_RFR = randomcv_rf.best_estimator_

predictions_RFR = best_estimator_RFR.predict(X_test)
```

```python
# RandomizedSearchCV on gradient boosting model

GBR = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator, 'max_depth': depth}

randomcv_gb = RandomizedSearchCV(GBR, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_gb = randomcv_gb.fit(X_train, y_train)
predictions_gb = randomcv_gb.predict(X_test)

best_params_gb = randomcv_gb.best_params_

best_estimator_gb = randomcv_gb.best_estimator_

predictions_gb = best_estimator_gb.predict(X_test)
```

```python
# 2. GridSearchCV

from sklearn.model_selection import GridSearchCV

# GridSearchCV on Random Forest Model

rfr_gs = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator, 'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(rfr_gs, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)

best_params_GRF = gridcv_rf.best_params_
best_estimator_GRF = gridcv_rf.best_estimator_

#Apply model on test data
predictions_GRF = best_estimator_GRF.predict(X_test)
```

```python
# GridSearchCV on gradient boosting model

gbr_gs = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator, 'max_depth': depth}

# Grid Search Cross-Validation with 5 fold CV
gridcv_gb = GridSearchCV(gbr_gs, param_grid = grid_search, cv = 5)
gridcv_gb = gridcv_gb.fit(X_train,y_train)

best_params_Ggb = gridcv_gb.best_params_
best_estimator_Ggb = gridcv_gb.best_estimator_

#Apply model on test data
predictions_Ggb = best_estimator_Ggb.predict(X_test)
```

Chapter 4 Results and Conclusion

4.1 Models Evaluation:

1. Linear Regression Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))

# calculate RMSE on test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))
```

```
print("RMSE on training data = "+str(RMSE_train_LR))
print("RMSE on test data = "+str(RMSE_test_LR))
```

```
RMSE on training data = 896.3923566617818
RMSE on test data = 857.6499143825099
```

```
# calculate R^2 on train data
r2_train_LR = r2_score(y_train, pred_train_LR)

# calculate R^2 on test data
r2_test_LR = r2_score(y_test, pred_test_LR)
```

```
print("r2 on training data = "+str(r2_train_LR))
print("r2 on test data = "+str(r2_test_LR))
```

```
r2 on training data = 0.7764548707831629
r2 on test data = 0.8274355583166422
```

```
errors = abs(pred_test_LR - y_test)
mape = 100 * np.mean(errors / y_test)
accuracy = 100 - mape
print('MAPE = {:0.2f}'.format(mape))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

```
MAPE = 18.32
Accuracy = 81.68%.
```

2. Decision Tree Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

# calculate RMSE on test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
```

```
print("RMSE on training data = "+str(RMSE_train_DT))
print("RMSE on test data = "+str(RMSE_test_DT))
```

```
RMSE on training data = 1085.3010213923144
RMSE on test data = 1167.4458024984463
```

```
# calculate R^2 on train data
r2_train_DT = r2_score(y_train, pred_train_DT)

# calculate R^2 on test data
r2_test_DT = r2_score(y_test, pred_test_DT)
```

```
print("r2 on training data = "+str(r2_train_DT))
print("r2 on test data = "+str(r2_test_DT))
```

```
r2 on training data = 0.6723053541937293
r2 on test data = 0.6802543318105283
```

```
errors = abs(pred_test_DT - y_test)
mape = 100 * np.mean(errors / y_test)
accuracy = 100 - mape
print('MAPE = {:0.2f}'.format(mape))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

```
MAPE = 31.12
Accuracy = 68.88%.
```

3. Random Forest Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))

# calculate RMSE on test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

```
print("RMSE on training data = "+str(RMSE_train_RF))
print("RMSE on test data = "+str(RMSE_test_RF))
```

```
RMSE on training data = 258.0866981429467
RMSE on test data = 566.1706181648321
```

```
# calculate R^2 on train data
r2_train_RF = r2_score(y_train, pred_train_RF)

# calculate R^2 on test data
r2_test_RF = r2_score(y_test, pred_test_RF)
```

```
print("r2 on training data = "+str(r2_train_RF))
print("r2 on test data = "+str(r2_test_RF))
```

```
r2 on training data = 0.981468944148885
r2 on test data = 0.9247986098947548
```

```
errors = abs(pred_test_RF - y_test)
mape = 100 * np.mean(errors / y_test)
accuracy = 100 - mape
print('MAPE = {:0.2f}'.format(mape))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

```
MAPE = 12.25
Accuracy = 87.75%.
```

4. Gradient Booting Model:

```
# Model Evaluation

# calculate RMSE on train data
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))

# calculate RMSE on test data
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

```
print("RMSE on training data = "+str(RMSE_train_GB))
print("RMSE on test data = "+str(RMSE_test_GB))
```

```
RMSE on training data = 432.4278455391316
RMSE on test data = 590.5043008062454
```

```
# calculate R^2 on train data
r2_train_GB = r2_score(y_train, pred_train_GB)

# calculate R^2 on test data
r2_test_GB = r2_score(y_test, pred_test_GB)
```

```
print("r2 on training data = "+str(r2_train_GB))
print("r2 on test data = "+str(r2_test_GB))
```

```
r2 on training data = 0.9479769002245042
r2 on test data = 0.9181954719254386
```

```
errors = abs(pred_test_GB - y_test)
mape = 100 * np.mean(errors / y_test)
accuracy = 100 - mape
print('MAPE = {:0.2f}'.format(mape))
print('Accuracy = {:0.2f}%.'.format(accuracy))
```

```
MAPE = 11.83
Accuracy = 88.17%.
```

5. **Hyper-parameter Tuning:**

```
RandomizedSearchCV - Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 23}
R-squared = 0.91.
RMSE =  616.2173240028509
MAPE = 13.18
Accuracy = 86.82%.
```

```
RandomizedSearchCV - Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.86.
RMSE =  780.0028174969768
MAPE = 20.87
Accuracy = 79.13%.
```

```
GridSearchCV - Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 13, 'n_estimators': 14}
R-squared = 0.91.
RMSE =  606.165881337705
MAPE = 13.08
Accuracy = 86.92%.
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.89.
RMSE =  699.546726312639
MAPE = 17.48
Accuracy = 82.52%.
```

From above models, it is clear that Gradient Boosting Model is providing best results having R-squared = 0.92 and Accuracy = 88.17%

References

1. https://stackoverflow.com/questions/43577086/pandas-calculate-haversine-distance-within-each-group-of-rows/43577275
2. https://www.r-bloggers.com/great-circle-distance-calculations-in-r/
3. https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/
4. http://benalexkeen.com/gradient-boosting-in-python-using-scikit-learn/
5. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
6. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Appendix

R code – Bike Rental Count

```
rm(list=ls())


# set working directory

setwd("F:/R_Programming/Edwisor")

getwd()


#################################################################


# loading Libraries

x = c("tidyr", "ggplot2", "corrgram", "usdm", "caret", "DMwR", "rpart", "randomForest",'xgboost')


# tidyr - drop_na

# ggplot2 - for visulization, boxplot, scatterplot

# corrgram - correlation plot

# usdm - vif

# caret - createDataPartition

# DMwR - regr.eval

# rpart - decision tree

# randomForest - random forest

# xgboost - xgboost


# load Packages

lapply(x, require, character.only = TRUE)
```

```
rm(x)

###############################################################

# loading dataset
df = read.csv("day.csv", header = T, na.strings = c(" ", "", "NA"))

#######################
# Exploring Datasets
#######################

# Structure of data
str(df)

# Summary of data
summary(df)

# Viewing the data
head(df,5)

######################################
# EDA, Missing value and Outlier analysis
######################################

# Changing the data types of variables
```

```
df$dteday = as.Date(as.character(df$dteday))

catnames=c("season","yr","mnth","holiday","weekday","workingday","weathersit")
for(i in catnames){
  print(i)
  df[,i]=as.factor(df[,i])
}

# Checking Missing data
apply(df, 2, function(x) {sum(is.na(x))})
# No missing values are present in the given data set.

#Outlier Analysis
num_index = sapply(df, is.numeric)
numeric_data = df[,num_index]
num_cnames = colnames(numeric_data)

for (i in 1:length(num_cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (num_cnames[i]), x = "cnt"), data = subset(df))+
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                      outlier.size=1, notch=FALSE) +
         theme(legend.position="bottom")+
         labs(y=num_cnames[i],x="cnt")+
```

```
        ggtitle(paste("Box plot of count for",num_cnames[i])))
}

# ## Plotting plots together
gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn4,gn5,ncol=2)
gridExtra::grid.arrange(gn6,gn3,ncol=2)

# continous variables hum, windspeed and casual includes outliers
# we do not consider casual variable for outlier removal bcz this is not predictor variable

outlier_var=c("hum","windspeed")

#Replace all outliers with NA

for(i in outlier_var){
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(val))
  df[,i][df[,i] %in% val] = NA
}

# Checking Missing data - after outlier
apply(df, 2, function(x) {sum(is.na(x))})

# hum includes 2 outliers and windspeed includes 13 outliers, so drop them
```

```
df = drop_na(df)


# Make a copy

df_after_outlier = df


############### Visualization ########################

# Scatter plot between temp and cnt

ggplot(data = df, aes_string(x = df$temp, y = df$cnt))+

  geom_point()


# Scatter plot between atemp and cnt

ggplot(data = df, aes_string(x = df$atemp, y = df$cnt))+

  geom_point()


# Scatter plot between hum and cnt

ggplot(data = df, aes_string(x = df$hum, y = df$cnt))+

  geom_point()


# Scatter plot between windspeed and cnt

ggplot(data = df, aes_string(x = df$windspeed, y = df$cnt))+

  geom_point()


# Scatter plot between season and cnt

ggplot(data = df, aes_string(x = df$season, y = df$cnt))+

  geom_point()
```

```
# Scatter plot between month and cnt

ggplot(data = df, aes_string(x = df$mnth, y = df$cnt))+

  geom_point()


# Scatter plot between weekday and cnt

ggplot(data = df, aes_string(x = df$weekday, y = df$cnt))+

  geom_point()


##################### Feature Selection and Scaling #######################

# generate correlation plot between numeric variables


numeric_index=sapply(df, is.numeric)

corrgram(df[,numeric_index], order=F, upper.panel=panel.pie,

        text.panel=panel.txt, main="Correlation plot")


# check VIF

vif(df[,10:15])

# if vif is greater than 10 then variable is not suitable/multicollinerity


# ANOVA test for checking p-values of categorical variables

for (i in catnames) {

  print(i)

  print(summary(aov(df$cnt ~df[,i], df)))

}
```

```
#From correlation plot and VIF, Removing variables atemp beacuse it is highly correlated with temp,

#From Anova, Removing weekday, holiday because they don't contribute much to the independent cariable

#Removing Causal and registered becuase that's what we need to predict.

#Removing instant and dteday because they are not useful in generating model.


# remove the variables

df=subset(df,select=-c(instant,dteday,atemp,casual,registered,holiday,weekday))


# Make a copy

df_clean = df


# generate histogram of continous variables

qqnorm(df$temp)

hist(df$temp)


qqnorm(df$hum)

hist(df$hum)


qqnorm(df$windspeed)

hist(df$windspeed)


######################### Model Development ###################


############ Splitting df into train and test ###################
```

```
set.seed(101)

split_index = createDataPartition(df$cnt, p = 0.80, list = FALSE)

train_data = df[split_index,]

test_data = df[-split_index,]


#############  Linear regression Model  #################

lm_model = lm(cnt ~., data=train_data)


# summary of trained model

summary(lm_model)


# prediction on test_data

lm_predictions = predict(lm_model,test_data[,1:8])


regr.eval(test_data[,9],lm_predictions)

#   mae        mse        rmse      mape

#  5.618664e+02 5.535047e+05 743.979  0.1728075


# compute r^2

rss_lm = sum((lm_predictions - test_data$cnt) ^ 2)

tss_lm = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)

rsq_lm = 1 - rss_lm/tss_lm

#    r^2 - 0.8407258


############## Decision Tree Model ###############
```

```
Dt_model = rpart(cnt ~ ., data=train_data, method = "anova")


# summary on trainned model

summary(Dt_model)


#Prediction on test_data

predictions_DT = predict(Dt_model, test_data[,1:8])


regr.eval(test_data[,9], predictions_DT)

#   mae         mse         rmse        mape

# 7.382928e+02 9.544681e+05 976.968     0.263328


# compute r^2

rss_dt = sum((predictions_DT - test_data$cnt) ^ 2)

tss_dt = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)

rsq_dt = 1 - rss_dt/tss_dt

#    r^2 - 0.7253463


#############  Random forest Model #####################

rf_model = randomForest(cnt ~., data=train_data)


# summary on trained model

summary(rf_model)


# prediction of test_data
```

```
rf_predictions = predict(rf_model, test_data[,1:8])


regr.eval(test_data[,9], rf_predictions)

#  mae          mse        rmse        mape

# 4.944837e+02 4.197740e+05 647.899    0.172765


# compute r^2

rss_rf = sum((rf_predictions - test_data$cnt) ^ 2)

tss_rf = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)

rsq_rf = 1 - rss_rf/tss_rf

#    r^2 - 0.8792076


############  XGBOOST Model ###########################

train_data_matrix = as.matrix(sapply(train_data[-9],as.numeric))

test_data_matrix = as.matrix(sapply(test_data[-9],as.numeric))


xgboost_model  =  xgboost(data  =  train_data_matrix,label  =  train_data$cnt,  nrounds  =  15,verbose  =
FALSE)


# summary of trained model

summary(xgboost_model)


# prediction on test_data

xgb_predictions = predict(xgboost_model,test_data_matrix)


regr.eval(test_data[,9], xgb_predictions)
```

```
#   mae          mse         rmse      mape

# 4.848618e+02 4.511432e+05 671.671   0.153581


# compute r^2

rss_xgb = sum((xgb_predictions - test_data$cnt) ^ 2)

tss_xgb = sum((test_data$cnt - mean(test_data$cnt)) ^ 2)

rsq_xgb = 1 - rss_xgb/tss_xgb

#    r^2 - 0.870181


# from above models, it is clear that xgboost is best model
```