BIKE RENTING

(Project Report)

By: Parth Sarathi

Contents:

1. Introduction:

1.1 Problem Statement:

The objective of this project is to prediction 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 fromHoliday Schedule)
- weekday: Day of the week
- workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit: (extracted fromFreemeteo)
  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
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

---

2. Methodology:

2.1 Data Preprocessing:

Data pre-processing is the first stage of any type of project. In this stage we get the feel of the data. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory Data Analysis.

This stage generally involves data cleaning, merging, sorting, looking for outlier analysis, looking for missing values in the data, Imputing missing values if found by various methods such as mean, median, mode, KNN imputation, etc.

In this project, while loading the data I dropped ‘casual’ and ‘registered’ variables as they sum up to ‘cnt’ and are of no use. I made the ‘instant’ variable as index as this variable gives the record index.

2.1.1 Missing Value Analysis:

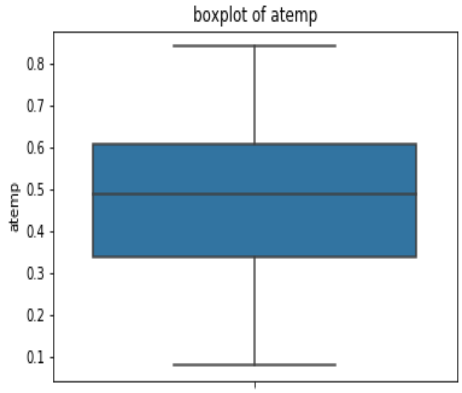
In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. Missing values left behind after outlier analysis; missing values can be in any form. In this dataset I haven’t found any missing values but, still I found out the best method for imputing missing values for further steps like outlier analysis, etc. Now, we will continue to next step.

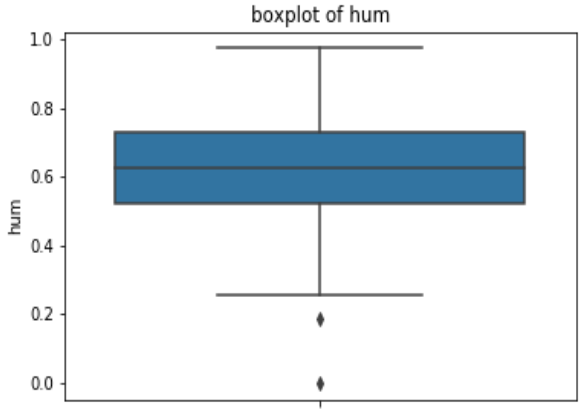
2.1.2 Outlier Analysis:

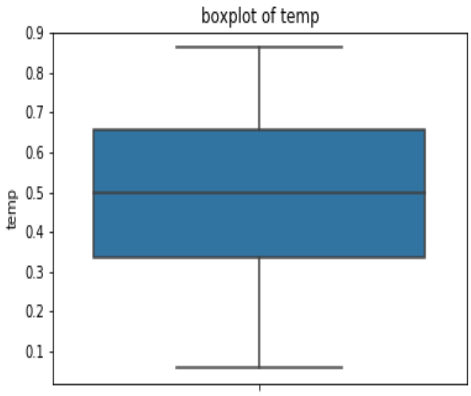
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. Now I have removed these outliers. This is how it's done,

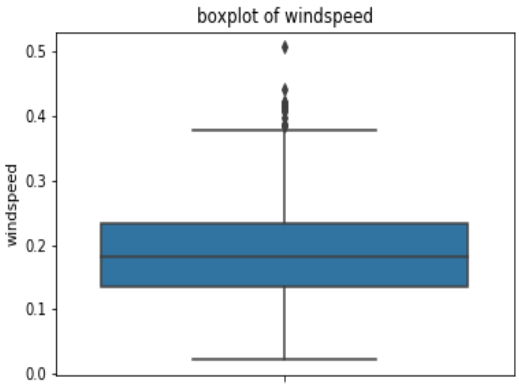
I. I replaced them with Nan values or we can say created missing values.
II. Then we imputed those missing values with KNN method in R and Mean in Python.
III. We tried three methods to impute the missing value: mean, median, KNN. But KNN method outperformed mean and median methods.

Univariate Boxplots: Boxplots for all Numerical Variables.

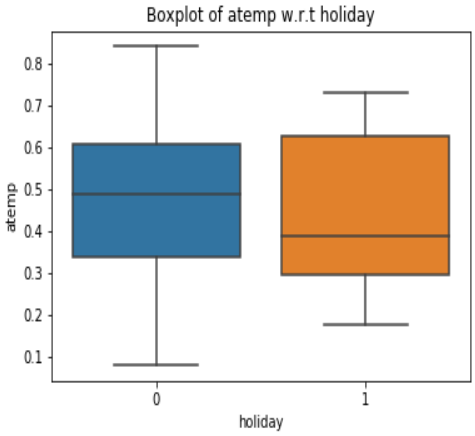


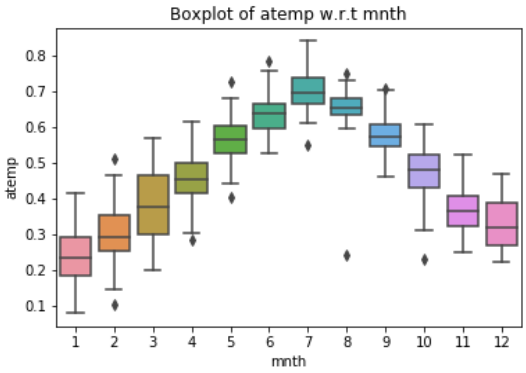


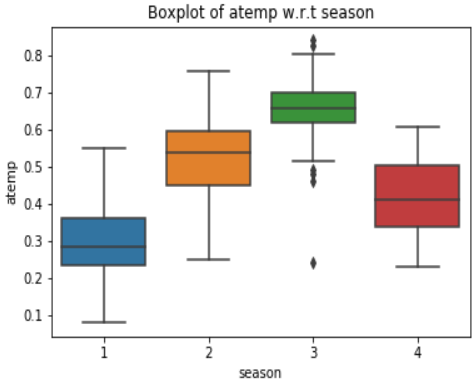


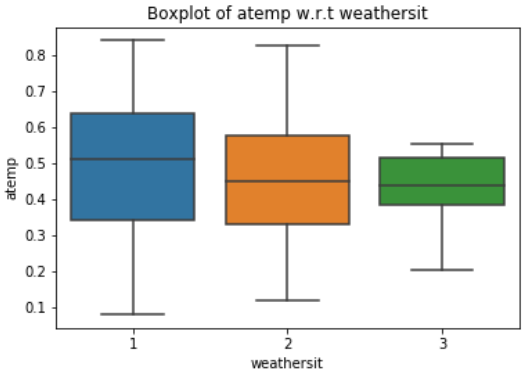


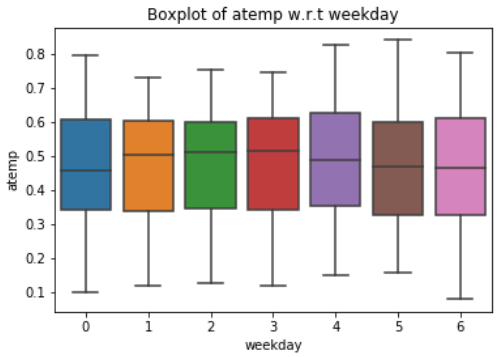
Bivariate Boxplots: Boxplots of all numeric variables vs all categorical variables.

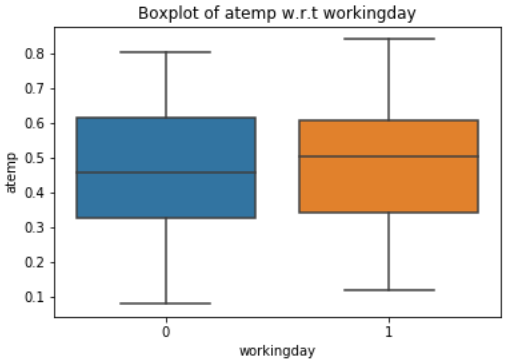


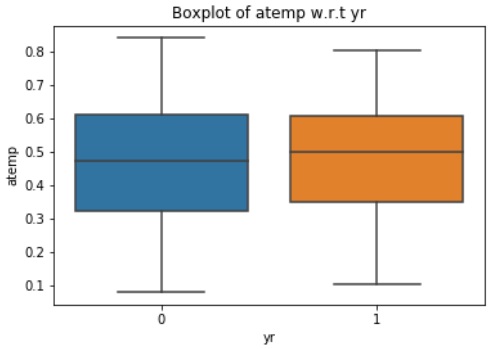


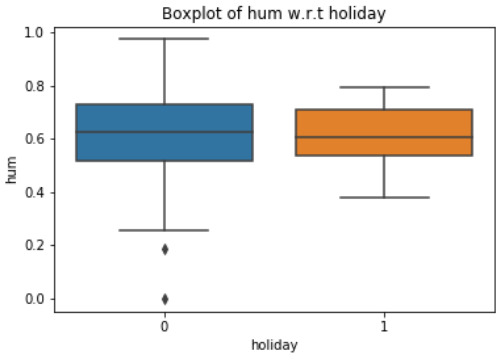


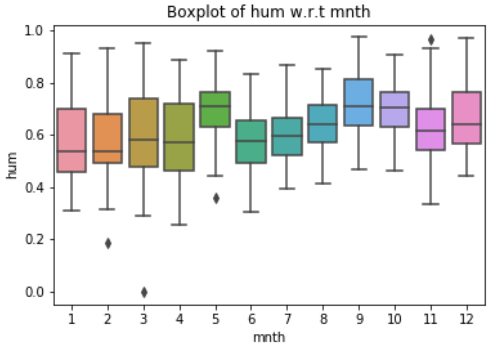


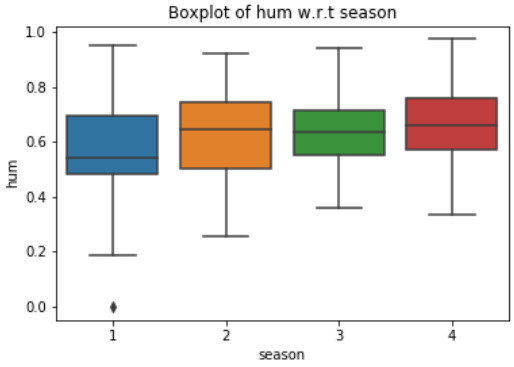


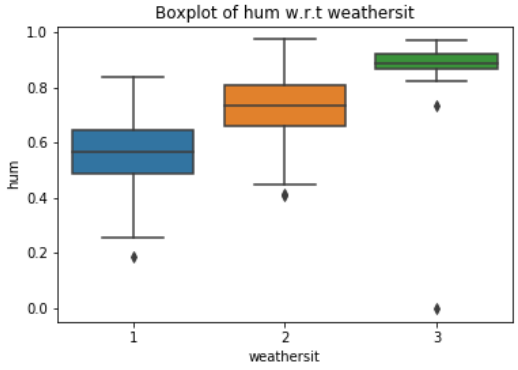


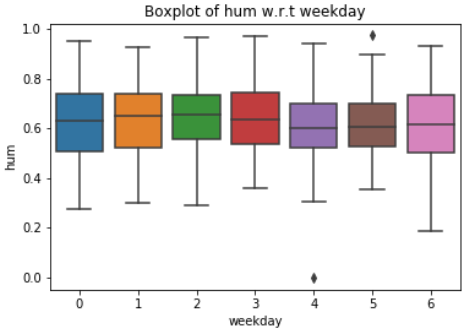

Boxplot of atemp w.r.t yr
atemp
yr
0
1

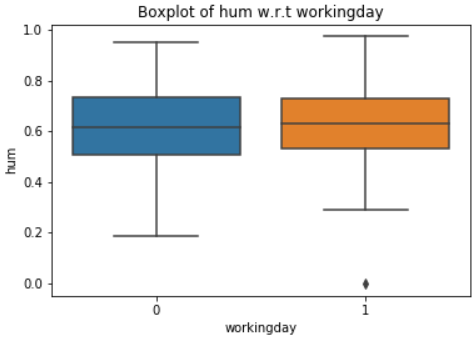

Boxplot of hum w.r.t holiday
hum
holiday
0
1

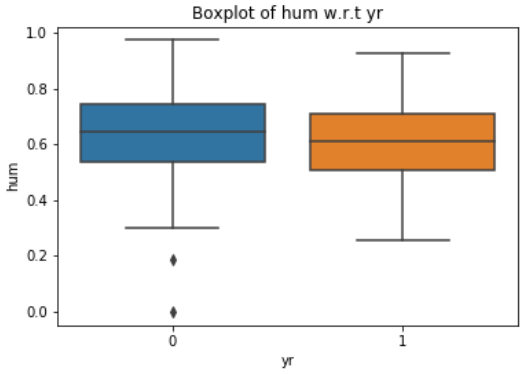

Boxplot of hum w.r.t mnth
hum
mnth

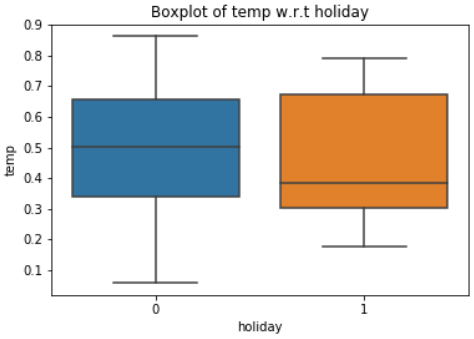

Boxplot of hum w.r.t season
hum
season

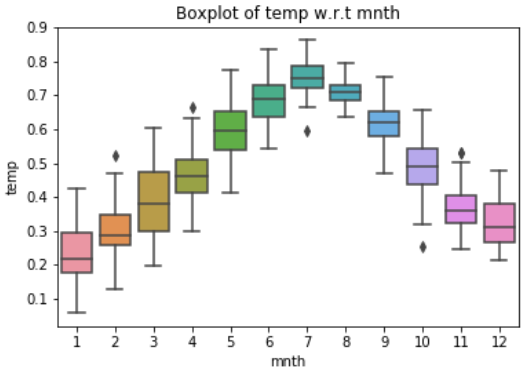

Boxplot of hum w.r.t weathersit
hum
weathersit

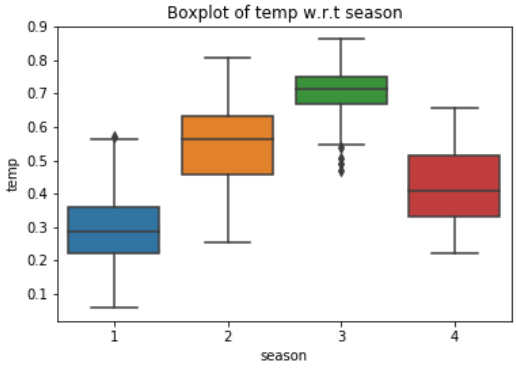

Boxplot of hum w.r.t weekday
hum
weekday

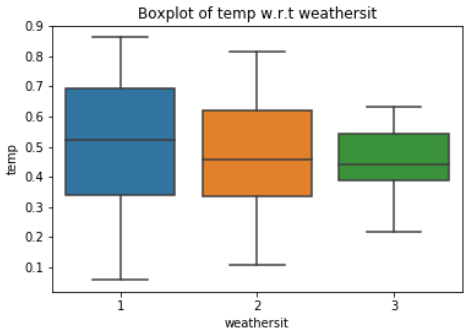

Boxplot of hum w.r.t workingday
hum
workingday
0
1

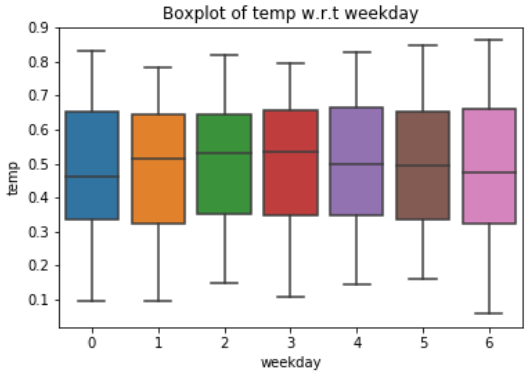

Boxplot of hum w.r.t yr
hum
yr
0
1

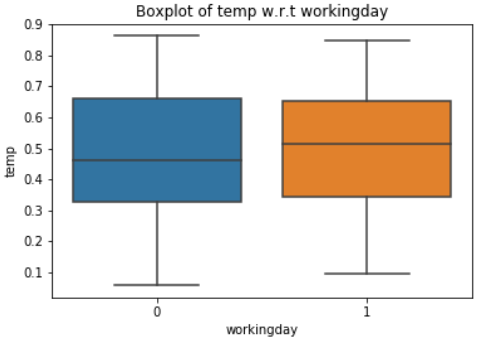

Boxplot of temp w.r.t holiday
temp
holiday
0
1

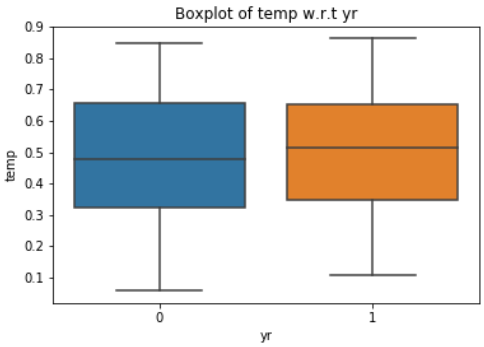

Boxplot of temp w.r.t mnth
temp
mnth

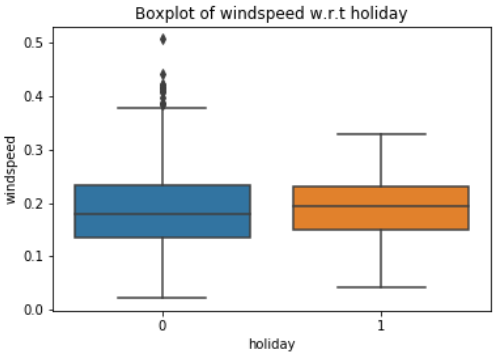

Boxplot of temp w.r.t season
temp
season

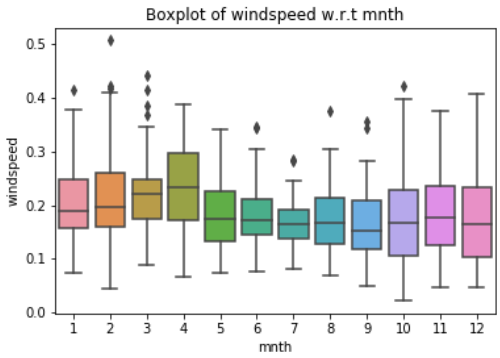

Boxplot of temp w.r.t weathersit
temp
weathersit

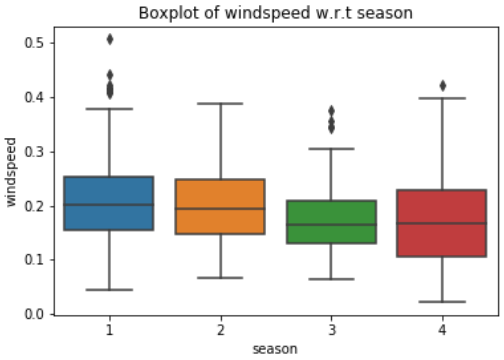

Boxplot of temp w.r.t weekday
temp
weekday
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6

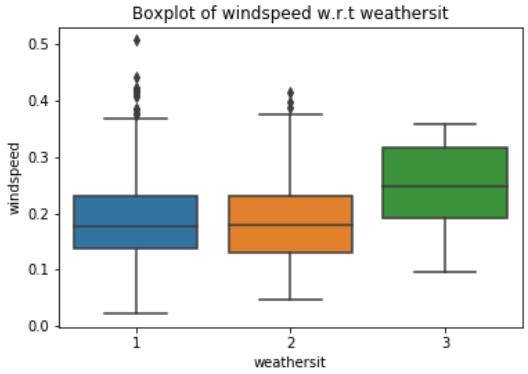

Boxplot of temp w.r.t workingday
temp
workingday
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1

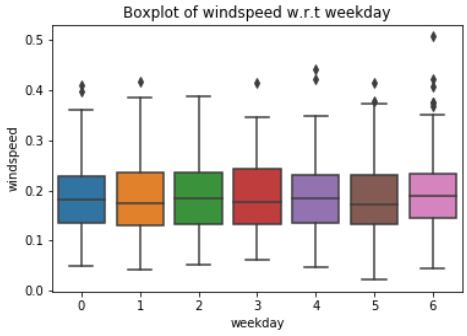

Boxplot of temp w.r.t yr
temp
yr
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1

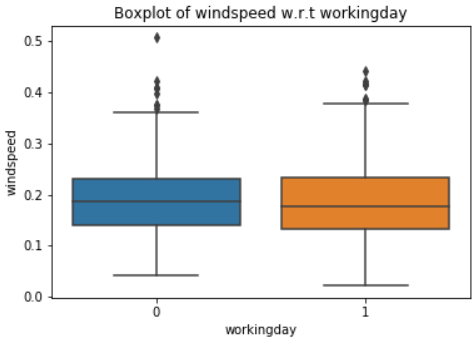

Boxplot of windspeed w.r.t holiday
windspeed
holiday
0.5
0.4
0.3
0.2
0.1
0.0
0
1

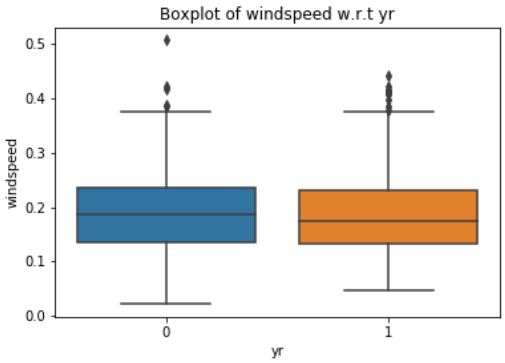

Boxplot of windspeed w.r.t mnth
windspeed
mnth
0.5
0.4
0.3
0.2
0.1
0.0
1
2
3
4
5
6
7
8
9
10
11
12


Boxplot of windspeed w.r.t season
windspeed
season
0.5
0.4
0.3
0.2
0.1
0.0
1
2
3
4









From the above boxplots we see that variables ‘hum’ and ‘windspeed’ have outliers in them.

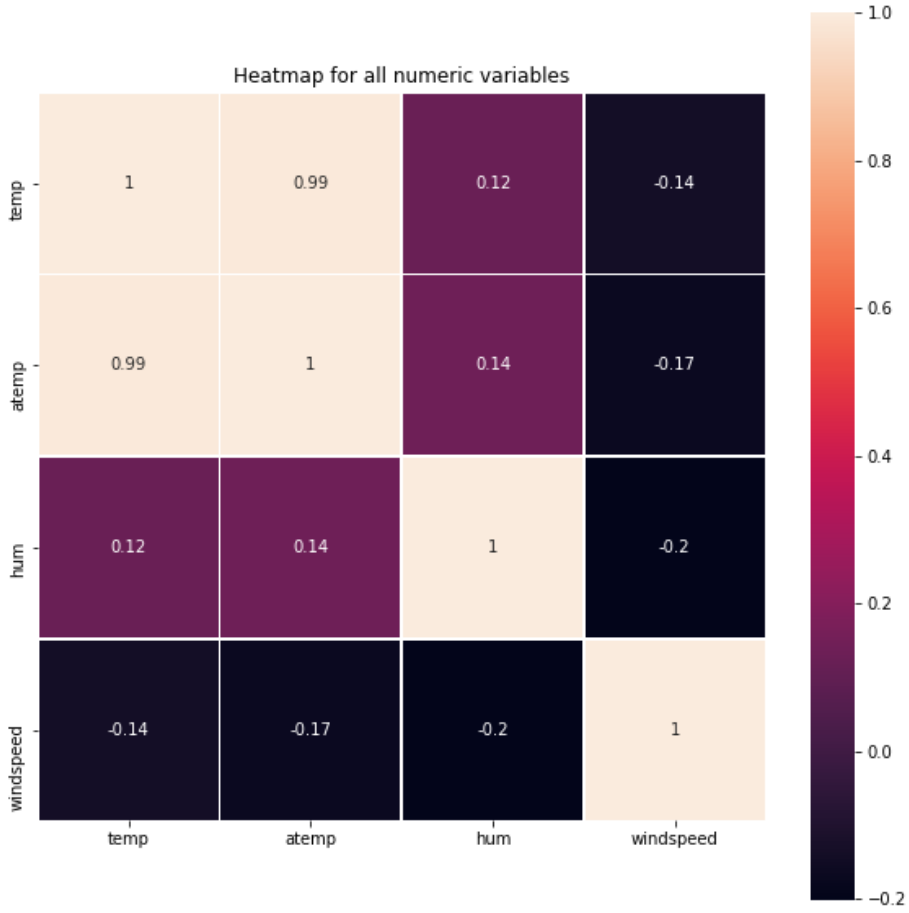
‘hum’ has 2 outliers and ‘windspeed’ has 13 outliers. I imputed the outliers by creating missing values and applying missing value imputation.

2.1.3 Feature Selection:

In this step we would allow only to pass relevant features to further steps. We remove irrelevant features from the dataset. We do this by understanding the domain knowledge of our features like we look for features which will not be helpful in predict the target variables. In this dataset we have to predict the Count based on environmental and seasonal features. Features which excludes this list is ‘instant’, ‘casual’ and ‘registered’.

Further below are some types of test involved for feature selection:

1. **Correlation analysis**: This requires only numerical variables. Therefore, we will filter out only numerical variables and feed it to correlation analysis. We do this by plotting correlation plot for all numerical variables. There should be no correlation between independent variables but there should be high correlation between independent variable and dependent variable. So, we plot the correlation plot. We can see that in correlation plot faded colour like skin colour indicates that 2 variables are highly correlated with each other. As the colour fades correlation values increases. From below correlation plot we see that:
   • 'temp' and 'atemp' are very highly correlated with each other.
   Correlation Plot:



2. **Chi Square test of Independence**: Unlike correlation analysis we will filter out only categorical variables and pass it to Chi-Square test. Chi-square test compares 2 categorical variables in a contingency table to see if they are related or not.
    I. For chi-square test: Dependency between Independent variable and dependent variable should be high and there should be no dependency among independent variables.
    II. Before proceeding to calculate chi-square statistic, we do the hypothesis testing:
    Null hypothesis: 2 variables are independent.
    Alternate hypothesis: 2 variables are not independent

The interpretation of chi-square test:

I. For theoretical or excel sheet purpose: If chi-square statistics is greater than critical value then reject the null hypothesis saying that 2 variables are dependent and if it's less, then accept the null hypothesis saying that 2 variables are independent.
II. While programming: If p-value is less than 0.05 then we reject the null hypothesis saying that 2 variables are dependent and if p-value is greater than 0.05 then we accept the null hypothesis saying that 2 variables are independent.
    Here we did the test between categorical independent variables pairwise.
    - If p-value < 0.05 then remove the variable
    - If p-value > 0.05 then keep the variable

variables which are highly dependent on each other based on p-values are:

- season and weathersit
- season and month
- holiday and weekday
- holiday and workingday
- weekday and holiday
- weekday and workingday
- workingday and holiday
- workingday and weekday
- weathersit and season

- weathersit and mnth
- mnth and season
- mnth and weathersit

After analyzing p value of all categorical features, I came to a conclusion that, Variables which I have removed and kept:

Removed: mnth, weekday, workingday, weathersit.

Kept: season, holiday, yr

3. **Multicollinearity**: In regression, "multicollinearity" refers to predictors that are correlated with other predictors. Multicollinearity occurs when your model includes multiple factors that are correlated not just to your response variable, but also to each other.
   I. Multicollinearity increases the standard errors of the coefficients.
   II. Increased standard errors in turn means that coefficients for some independent variables may be found not to be significantly different from 0.
   III. In other words, by overinflating the standard errors, multicollinearity makes some variables statistically insignificant when they should be significant. Without multicollinearity (and thus, with lower standard errors), those coefficients might be significant.
   IV. VIF is always greater or equal to 1.
      - if VIF is 1 --- Not correlated to any of the variables.
      - if VIF is between 1-5 --- Moderately correlated.
      - if VIF is above 5 --- Highly correlated.

      If there are multiple variables with VIF greater than 5, only remove the variable with the highest VIF.

   V. And if the VIF goes above 10, you can assume that the regression coefficients are poorly estimated due to multicollinearity.

   We have checked for multicollinearity in our Dataset and VIF values for temp and atemp are above 5.

2.1.4 Feature Scaling:

Data scaling methods are used when we want our variables in data to scaled on common ground. It is performed only on continuous variables.

- **Normalization**: Normalization refers to the dividing of a vector by its length. Normalization normalizes the data in the range of 0 to 1. It is generally used when we are planning to use distance method for our model development purpose such as KNN. Normalizing the data improves convergence of such algorithms. Normalization of data scales the data to a very small interval, where outliers can be loosed.
- Standardization: Standardization refers to the subtraction of mean from individual point and then dividing by its SD. Z is negative when the raw score is below the mean and Z is positive when above mean. When the data is distributed normally you should go for standardization.

Linear Models assume that the data you are feeding are related in a linear fashion, or can be measured with a linear distance metric. Also, most of our data is not distributed normally so we had chose normalization over standardization.

Note: It is performed only on Continuous variables.

Splitting train and test Dataset:

a. The training data should be the earliest data and test data should be the latest data.

b. we will fit our model on the training data and test on the newest data, to understand how our model performs on new, unseen data.

c. we can't use sklearn's train_test_split bcoz it randomly shuffles the train and test data. We have divided our data in 80-20% i.e. 80% in train and 20% in test.

2.2 Model Development:

Our problem statement wants us to predict the Bike rental count. This is a Regression problem. So, we are going to build regression models on training data

and predict it on test data. In this project I have built models using 3 Regression Algorithms:

I. Linear Regression
II. Decision Trees
III. Random Forest

The complete dataset is split into train and test using date as a variable for sampling.

We will evaluate performance on test dataset generated using Sampling. We will deal with specific error metrics like,

Regression metrics for our Models:

- r square
- Adjusted r square
- MAPE(Mean Absolute Percentage Error)
- MSE(Mean square Error)
- RMSE(Root Mean Square Error)

2.3 Hyperparameter Optimization:

a) To find the optimal hyperparameter we have used sklearn.model_selection.GridSearchCV.
b) GridSearchCV tries all the parameters that we provide it and then returns the best suited parameter for data.
c) We gave parameter dictionary to GridSearchCV which contains keys which are parameter names and values are the values of parameters which we want to try for.

Below are best hyperparameter we found for different models:

I. Linear Regression:

```
Tuned Decision reg Parameters: {'copy_X': True,
'fit_intercept': False}
Best score is 0.32130976079473167
```

II. Decision Tree:

```
Tuned Decision Tree Parameters: {'max_depth': 4,
'min_samples_split': 4}
Best score is -0.07643340870481896
```

III. Random Forest:

```
Tuned Decision Forest Parameters: {'max_depth': 18,
'min_samples_split': 2, 'n_estimators': 100}
 Best score is 0.31606561812439565
```

2.4. Model Performance:

Here, I have evaluated the performance of different regression models based on different error metrics:

I. Linear Regression:

Error Metrics	R square	Adj r square	MAPE	MSE	RMSE
Train	0.8192789600	0.8158035554	18.823319860	577338.50026	759.8279412220
Test	0.5559290185	0.519745457	165.6741682	1560554.56705	1249.221584

II. Decision Tree:

Error Metrics	R Square	Adj r square	MAPE	MSE	RMSE
Train	0.9122819012	0.910595014	12.17463217	280227.66802	529.36534455
Test	0.603310856	0.570988037	165.177644	1394045.278	1180.6969458

III. Random Forest:

Error Metrics	R Square	Adj r Square	MAPE	MSE	RMSE
Train	0.981875458	0.98152690	6.2373819	57901.37005	240.62703
Test	0.56068414	0.52488803	167.6567515	1543844.1045	1242.51523312

---

3. Conclusion:

We have selected Random Forest Regression as our Best Model to Predict Bike Rental Count.

Random Forest:

Error Metrics	R Square	Adj r Square	MAPE	MSE	RMSE
Train	0.981875458	0.98152690	6.2373819	57901.37005	240.62703
Test	0.56068414	0.52488803	167.6567515	1543844.1045	1242.51523312

Predicted Bike Rental Count by Random Forest Regression Model:

dteday	cnt
2012-08-07	7251.7600
2012-08-08	7296.6200
2012-08-09	6647.7185
2012-08-10	6376.3400
2012-08-11	6547.3200

Original Bike Rental Count:

dteday	cnt
2012-08-07	7273
2012-08-08	7534
2012-08-09	7286
2012-08-10	5786
2012-08-11	6299

---

4. Python Code:

```
import os
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import chi2_contingency
import scipy.stats as stats
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import metrics
#Set working directory
os.chdir('C:/Users/Best Lappy/Desktop/EDWISOR/Projects/2')
os.getcwd()
#load dataset
df = pd.read_csv("day.csv", index_col = "dteday")
```

```
#Dropping the variables 'registered' and 'casual' as both sum is 'cnt'.Also dropping 'instant' as its of no use.

df = df.drop(['registered', 'casual', 'instant'], axis=1)

#Type conversion

numeric = ['temp', 'atemp', 'hum', 'windspeed']

cat = ['season','holiday','weekday','workingday','weathersit','yr','mnth']

df[cat] = df[cat].astype('category')

#missing value Analysis

df.isnull().sum()

#Checking for the best missing imputation method for further operations

df1 = df.copy()

#df = df1.copy()

df['hum'].iloc[29]

#Actual value = 0.722174

#mean = 0.6277649123287666

#median = 0.62625

#Create missing value, a small test to identify which method is good for imputation

#df['hum'].iloc[29] = np.nan

#Mean Imputation

#df['hum'] = df['hum'].fillna(df['hum'].mean())

#df['hum'].iloc[29]

#Median Imputation

#df['hum'] = df['hum'].fillna(df['hum'].median())

#df['hum'].iloc[29]

# Outlier Analysis

for i in numeric:
```

```
    sns.boxplot(y=i, data = df)
    plt.title("boxplot of " +i)
    #plt.savefig('Boxplot'+str(i)+'.png')
    plt.show()
for i in cat:
    for j in numeric:
        sns.boxplot(x = i, y = j, data = df)
        plt.title('Boxplot of '+j+' w.r.t '+i)
        #plt.savefig('Boxplot of '+str(j)+' w.r.t '+str(i)+'.png')
        plt.show()
def outlier_treatment(col):
    #Extract quartiles
    q75, q25 = np.percentile(df[col], [75, 25])
    #Calculate IQR
    IQR = q75-q25
    #Calculate inner and outer fence
    minimum = q25 - (1.5*IQR)
    maximum = q75 + (1.5*IQR)
    #Replace with NA
    df.loc[df[col] < minimum,col] = np.nan
    df.loc[df[col] > maximum,col] = np.nan
outlier_treatment('hum')
outlier_treatment('windspeed')
#As mean is the best imputation method, we impute the missing values(Outliers) with mean.
df['hum'] = df['hum'].fillna(df['hum'].mean())
```

```
df['windspeed'] = df['windspeed'].fillna(df['windspeed'].mean())
#Heatmap using corelation matrix
plt.figure(figsize = (10,10))
sns.heatmap(df[numeric].corr(),linewidths=0.5,linecolor='w',square=True,annot=True)
plt.title("Heatmap for all numeric variables")
#plt.savefig('corelation.png')
plt.show()
#Loop for chi square test
for i in cat:
    for j in cat:
        if(i!=j):
            chi2,p,dof,ex = chi2_contingency(pd.crosstab(df[i], df[j]))
            if(p < 0.05):
                print(i,"and",j,"are dependent on each other with",p,'----Remove')
            else:
                print(i,"and",j,"are independent on each other with",p,'----Keep')
df = df.drop(['mnth','weekday','weathersit','workingday'],axis = 1)
# Feature Scaling
#Normality check by plotting distplot and probplot
fig,ax = plt.subplots(nrows=5,ncols=2)
fig.set_size_inches(25, 25)
sns.distplot(df['temp'],bins =50,ax = ax[0][0])
ax[0][0].set(title="temp distribution")
_ = stats.probplot(df['temp'], dist='norm', fit=True,plot=ax[0][1])
ax[0][1].set(title="Probability Plot")
```

```
sns.distplot(df['atemp'],bins =50,ax = ax[1][0])
ax[1][0].set(title="atemp distribution")
_ = stats.probplot(df['atemp'], dist='norm', fit=True,plot=ax[1][1])
ax[1][1].set(title="Probability Plot")
sns.distplot(df['hum'],bins =50,ax = ax[2][0])
ax[2][0].set(title="hum distribution")
_ = stats.probplot(df['hum'], dist='norm', fit=True,plot=ax[2][1])
ax[2][1].set(title="Probability Plot")
sns.distplot(df['windspeed'],bins =50,ax = ax[3][0])
ax[3][0].set(title="windspeed distribution")
_ = stats.probplot(df['windspeed'], dist='norm', fit=True,plot=ax[3][1])
ax[3][1].set(title="Probability Plot")
sns.distplot(df['cnt'],bins =50,ax = ax[4][0])
ax[4][0].set(title="cnt distribution")
_ = stats.probplot(df['cnt'], dist='norm', fit=True,plot=ax[4][1])
ax[4][1].set(title="Probability Plot")
#plt.savefig('Distribution before Normaliation.png')
plt.show()
#Normalization
for i in numeric:
    print(i)
    df[i] = (df[i] - min(df[i]))/(max(df[i]) - min(df[i]))
# Multicollinearity Test
outcome, predictors = dmatrices('cnt ~ +season+ yr +holiday + temp+atemp + hum
+ windspeed',df, return_type='dataframe')
```

```
# calculating VIF for each individual Predictors

vif = pd.DataFrame()

vif["VIF"] = [variance_inflation_factor(predictors.values, i) for i in
range(predictors.shape[1])]

vif["features"] = predictors.columns

vif

# Feature Engineering

columns = ['temp','atemp']

df['mean_temp'] = df.apply(lambda row: row[columns].mean(), axis=1)

df = df.drop(['atemp', 'temp'], axis = 1)

cat = ['season', 'holiday', 'yr']

#Creating dummies for categorical variables

for i in cat:

    ''' Creating dummies for each variable in cat and merging dummies dataframe to
our original dataframe '''

    temp = pd.get_dummies(df[i], prefix = i)

    df = df.join(temp)

#We will remove some variables which were used to generate one hot encoding
variables

df = df.drop(['season','holiday','yr'],axis = 1)

# Model Development

from sklearn import tree

from sklearn.model_selection import train_test_split

target_cnt = df.iloc[:,2]

target_cnt.head()

feature = df.drop(['cnt'],axis=1)
```

```
feature.head()
train_size = int(0.80 * df.shape[0]) # train_size = 584
train_features = feature[:train_size]
train_target_cnt = target_cnt[:train_size]
test_features = feature[train_size:]
test_target_cnt = target_cnt[train_size:]
print(df.shape, train_features.shape,
test_features.shape,train_target_cnt.shape,test_target_cnt.shape)
def scores(y, y_):
    print('r square  ', metrics.r2_score(y, y_))
    print('Adjusted r square:{}'.format(1 - (1-metrics.r2_score(y, y_))*(len(y)-
1)/(len(y)-train_features.shape[1]-1)))
    print('MAPE:{}'.format(np.mean(np.abs((y - y_) / y))*100))
    print('MSE:', metrics.mean_squared_error(y, y_))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y, y_)))
def test_scores(model):
    print('<<<---------- Training Data Score ---------->')
    print()
    #Predicting result on Training data
    y_pred = model.predict(train_features)
    scores(train_target_cnt,y_pred)
    print()
    print('<<<---------- Test Data Score ---------->')
    print()
    # Evaluating on Test Set
    y_pred = model.predict(test_features)
```

```
    scores(test_target_cnt,y_pred)
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
# Linear Regression
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'copy_X':[True, False],
        'fit_intercept':[True,False]}
# Instantiate a Decision reg classifier: reg
reg = LinearRegression()

# Instantiate the gridSearchCV object: reg_cv
reg_cv = GridSearchCV(reg, param_dist, cv=5,scoring='r2')

# Fit it to the data
reg_cv.fit(feature, target_cnt)

# Print the tuned parameters and score
print("Tuned Decision reg Parameters: {}".format(reg_cv.best_params_))
print("Best score is {}".format(reg_cv.best_score_))
# Instantiate a reg regressor: reg
reg = LinearRegression(copy_X= True, fit_intercept=False)
```

```
# Fit the regressor to the data
reg.fit(train_features,train_target_cnt)
test_scores(reg)
# Decision Tree Regression
# Setup the parameters and distributions to sample from: param_dist
param_dist = {'max_depth': range(2,16,2),
              'min_samples_split': range(2,16,2)}

# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeRegressor()

# Instantiate the gridSearchCV object: tree_cv
tree_cv = GridSearchCV(tree, param_dist, cv=5)

# Fit it to the data
tree_cv.fit(feature, target_cnt)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
# Instantiate a tree regressor: tree
tree = DecisionTreeRegressor(max_depth= 6, min_samples_split=4)

# Fit the regressor to the data
tree.fit(train_features,train_target_cnt)
```

```
# Make predictions and cal error
test_scores(tree)
# Random Forest Regression
# Create the random grid
random_grid = {'n_estimators': range(100,500,100),
               'max_depth': range(10,20,1),
               'min_samples_split':range(2,5,1)}
# Instantiate a Decision Forest classifier: Forest
Forest = RandomForestRegressor()

# Instantiate the gridSearchCV object: Forest_cv
Forest_cv = GridSearchCV(Forest, random_grid, cv=5)

# Fit it to the data
Forest_cv.fit(feature, target_cnt)

# Print the tuned parameters and score
print("Tuned Decision Forest Parameters: {}".format(Forest_cv.best_params_))
print("Best score is {}".format(Forest_cv.best_score_))
# Instantiate a Forest regressor: Forest
Forest = RandomForestRegressor(max_depth= 18,
min_samples_split=2,n_estimators=100)
# Fit the regressor to the data
Forest.fit(train_features,train_target_cnt)
test_scores(Forest)
test_predicted = Forest.predict(test_features)
```

```
pd.DataFrame(test_predicted,index = test_target_cnt.index,columns=['cnt']).head()
pd.DataFrame(test_target_cnt,columns=['cnt']).head()
```

----------END----------