Predicting Bike Rental Count

ABHISHEK KUMAR

Contents

Chapter 1

Introduction

1.1 Problem Statement

The objective of this Project is to Predict 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 indexdteday: Dateseason:
- 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 + Scatteredclouds
  - 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_max-t_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

Chapter 2

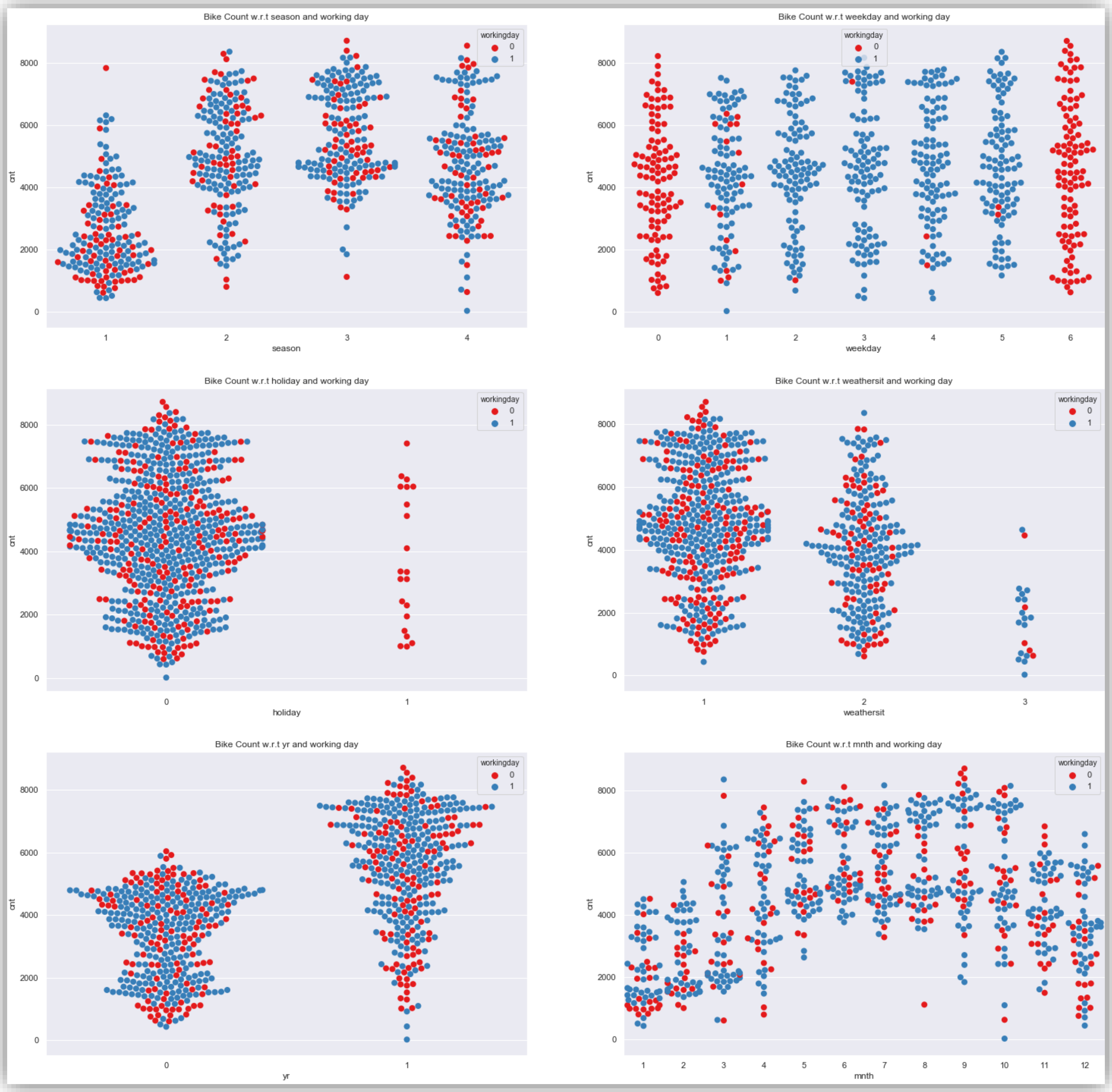
Methodology

2.1 Pre-Processing

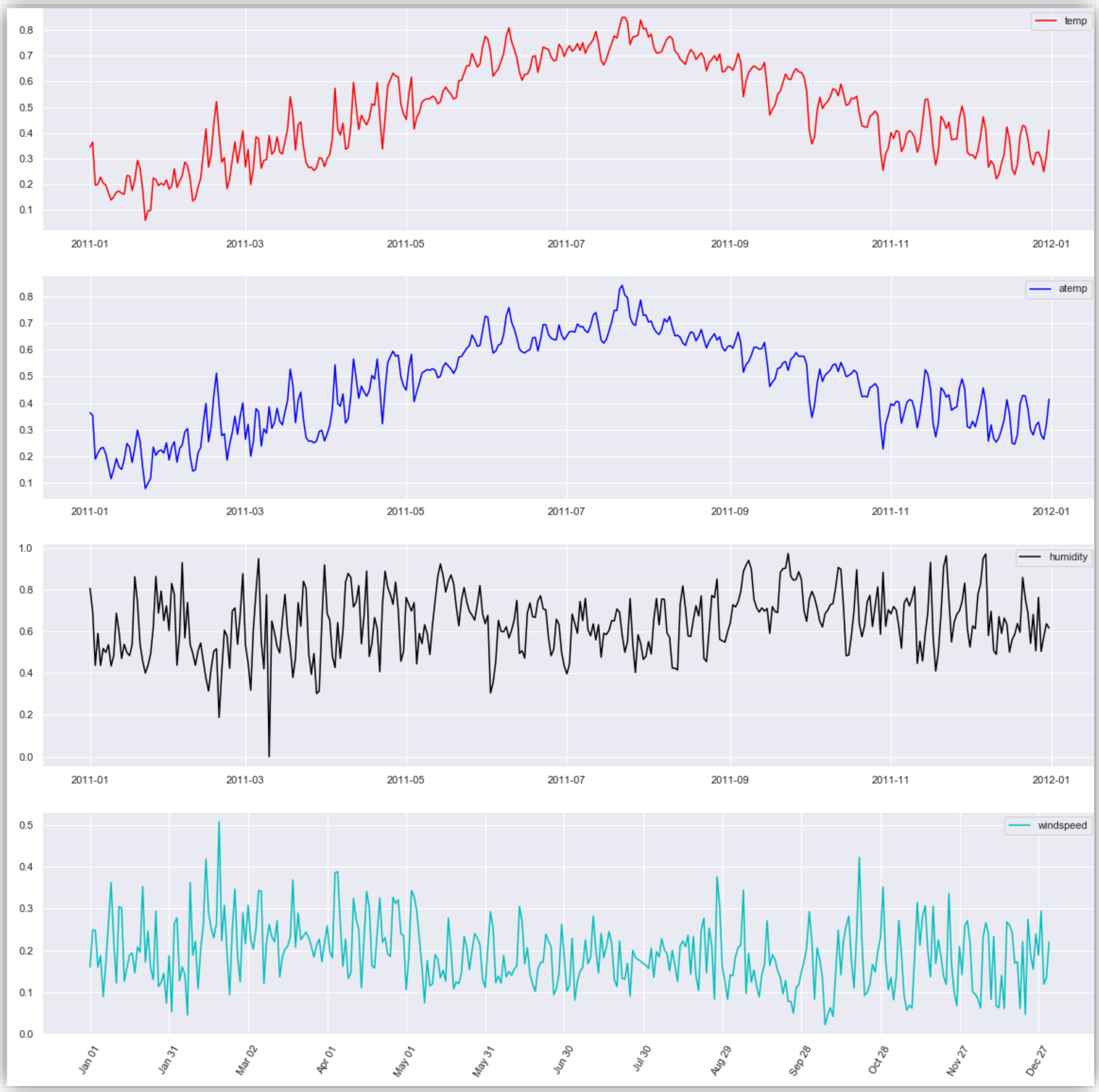
Data pre-processing is the first stage of any type of project. In this stage we get the feel of the data. We do this by looking at plots of independent variables vs target variables. If the data is messy, we try to improve it by sorting deleting extra rows and columns. This stage is called as Exploratory 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.
Further we will look into what pre processing steps do this project was involved in.

Getting feel of data via visualization:

Bee Swarmplots because no binning Bias unlike histogram:

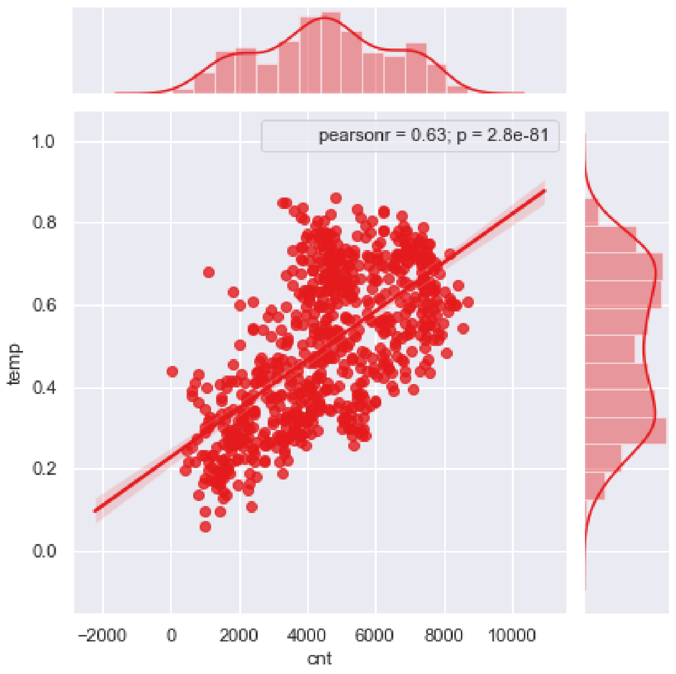


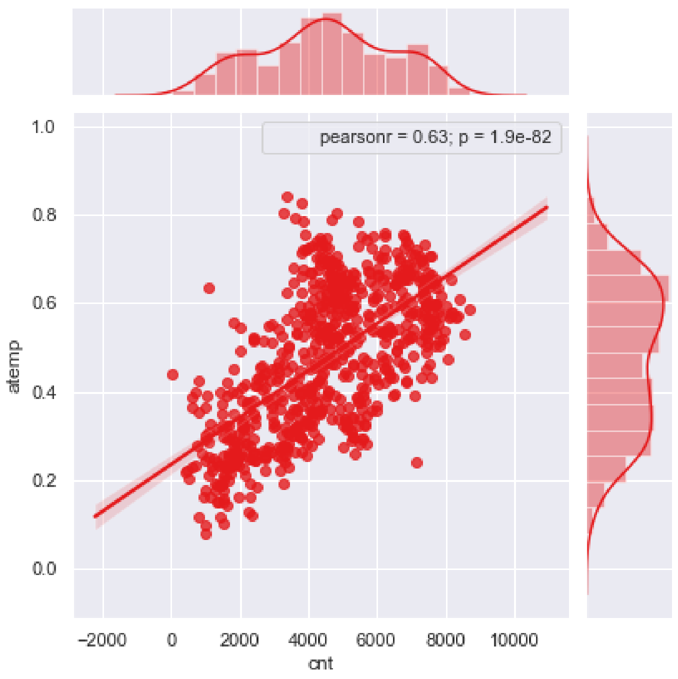
Time Series Visualization:

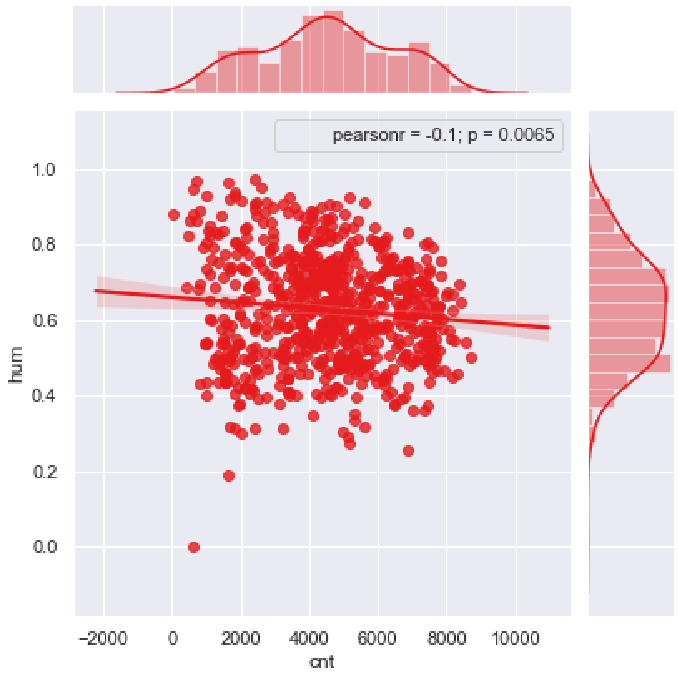


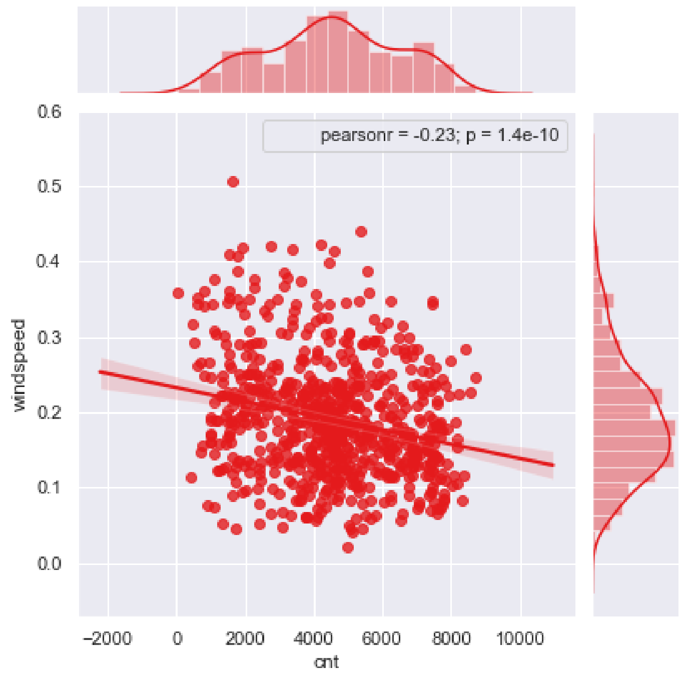
Some Jointplots:

- They are used for Bivariate Analysis.
- Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
- Also, we have annotated Pearson correlation coefficient and p value.
- Plotted only for numerical/continuous variables
- Each numerical variable at a time Vs ‘cnt’ which is Target Variable.

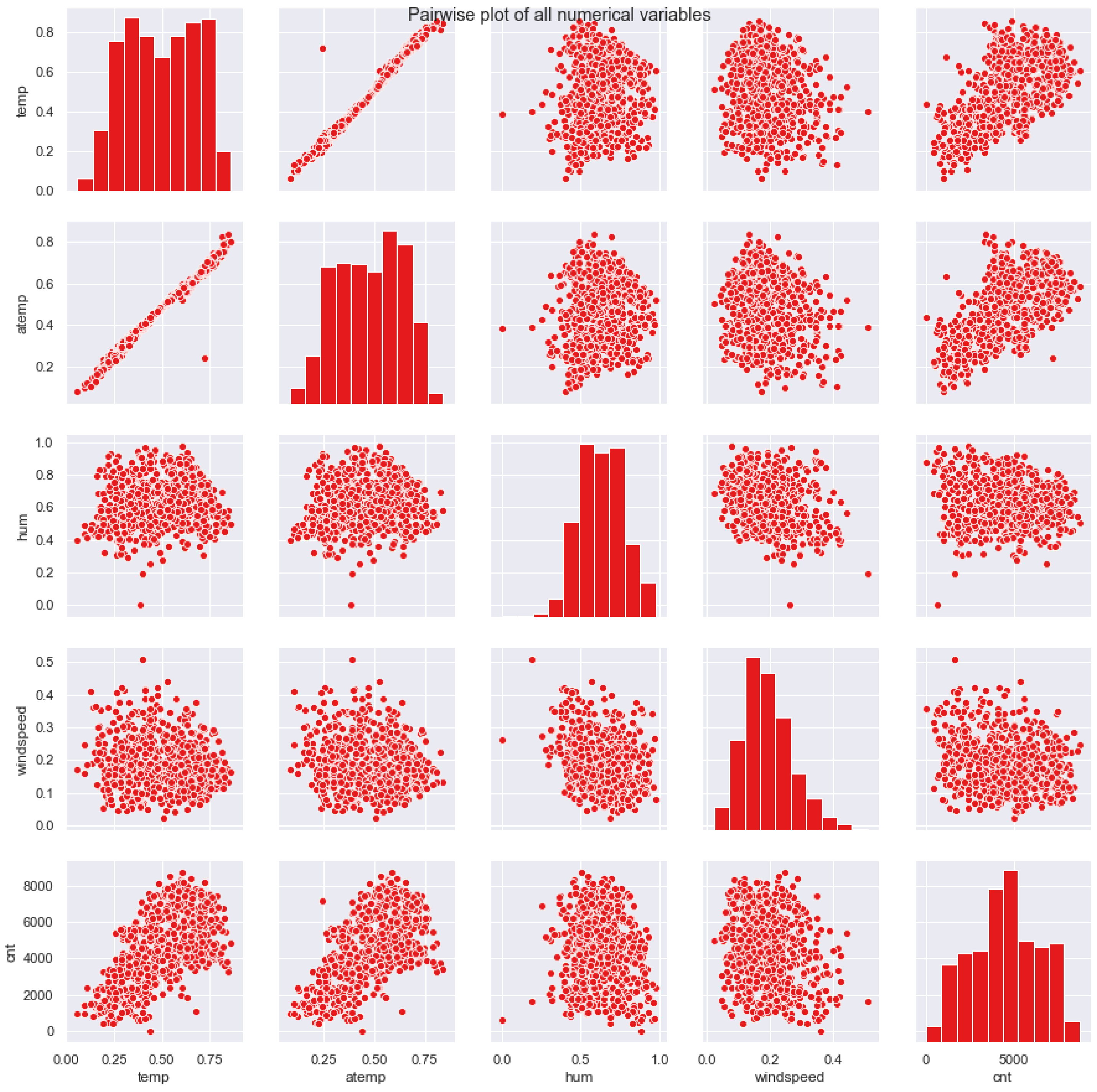








Pairwise Plots for all Numerical variables:



2.1.1 Missing value Analysis

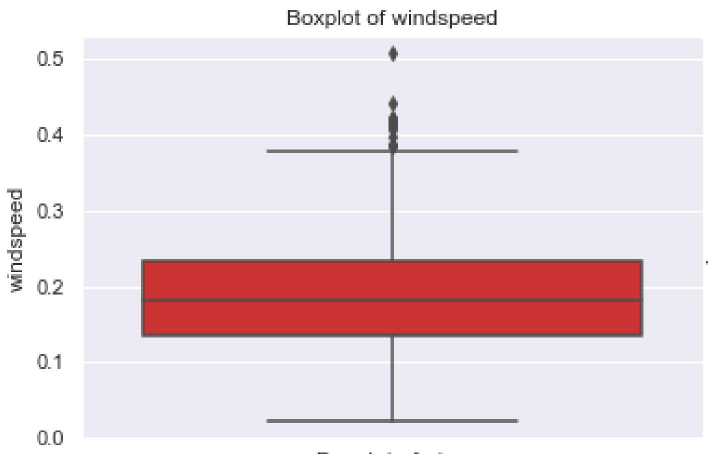
In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks.
Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form. Unfortunately, in this dataset we haven't found any missing values. Therefore, we will continue to next step.

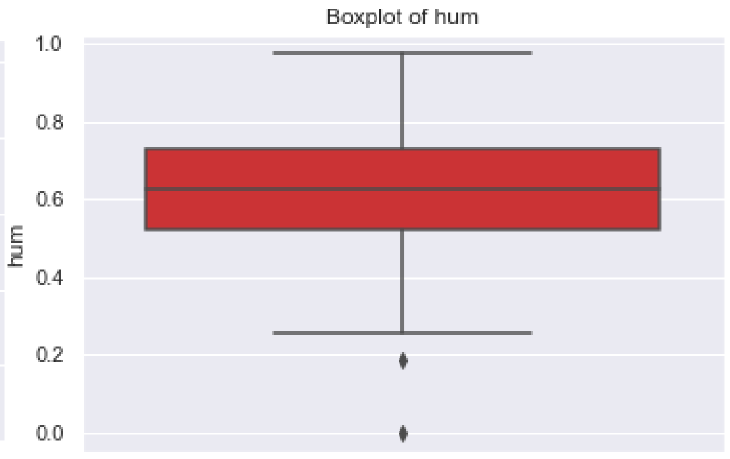
2.1.2 Outlier Analysis

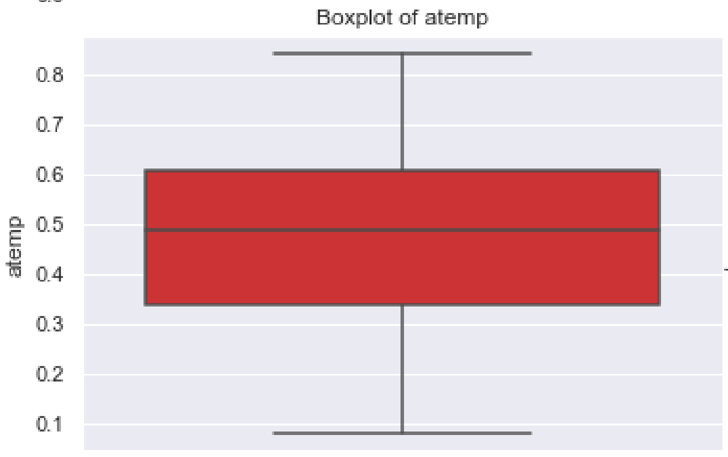
We look for outlier in the dataset by plotting Boxplots. There are outliers present in the data. Now we have removed these outliers. This is how we done,

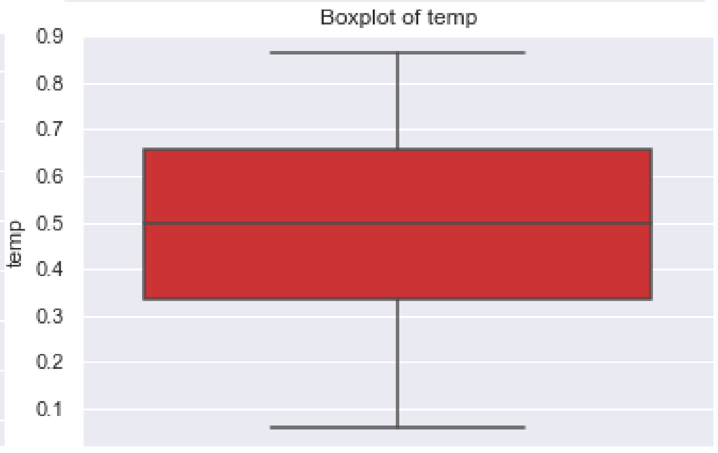
I. We replaced them with Nan values or we can say created missing values.
II. Then we imputed those missing values with KNN method.
III. We tried three methods to impute the missing value: mean, median, KNN. But KNN method outperformed mean and median methods.
IV. We checked the performance of each method by checking Standard Deviation of that variable which has outliers before imputation and after imputation.

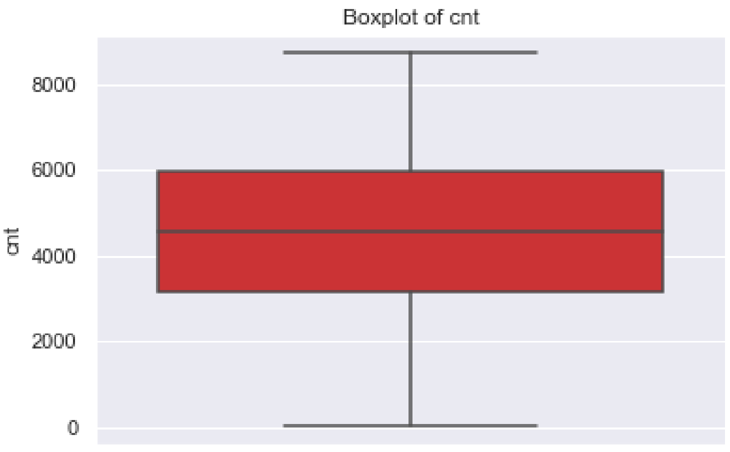
Univariate Boxplots: Boxplots for all Numerical Variables also for target variable



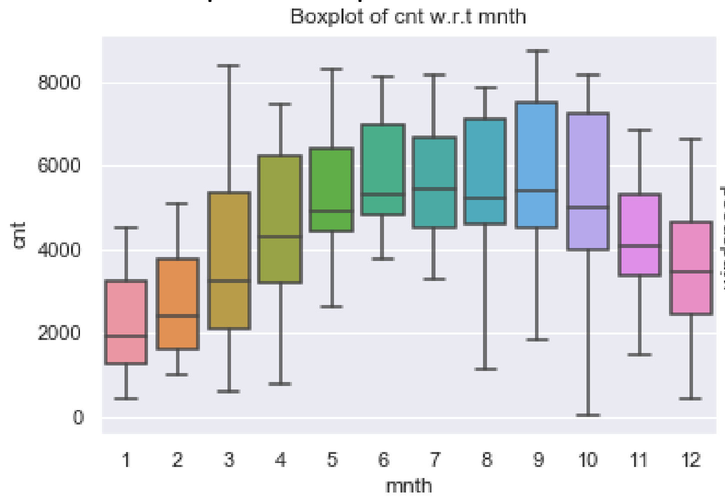


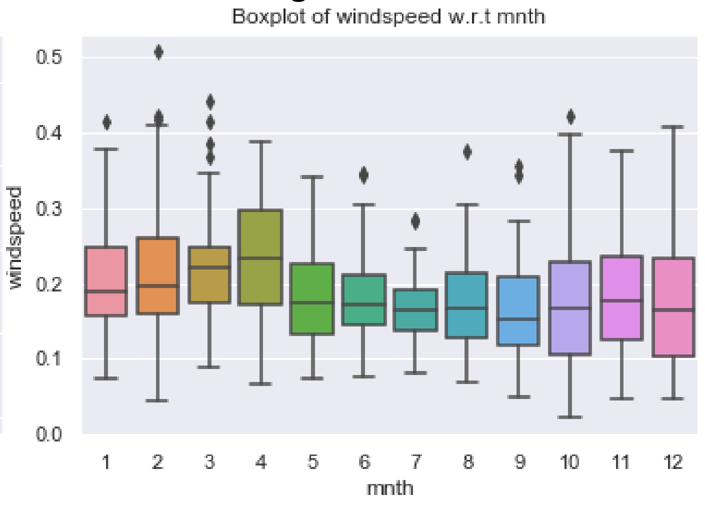


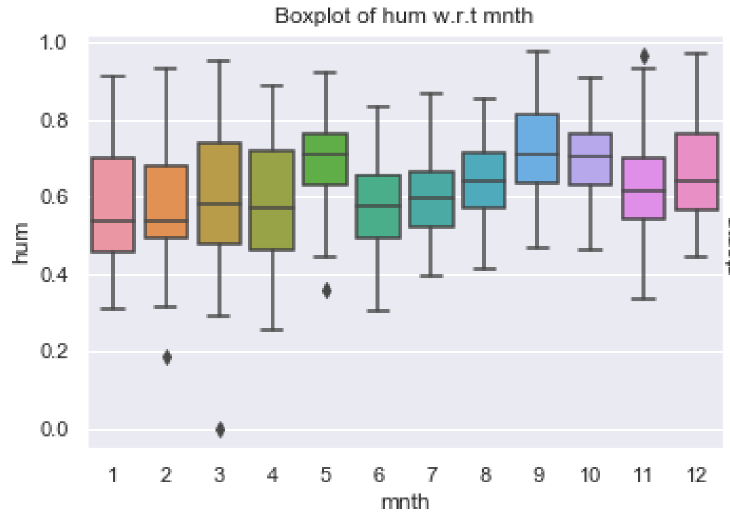


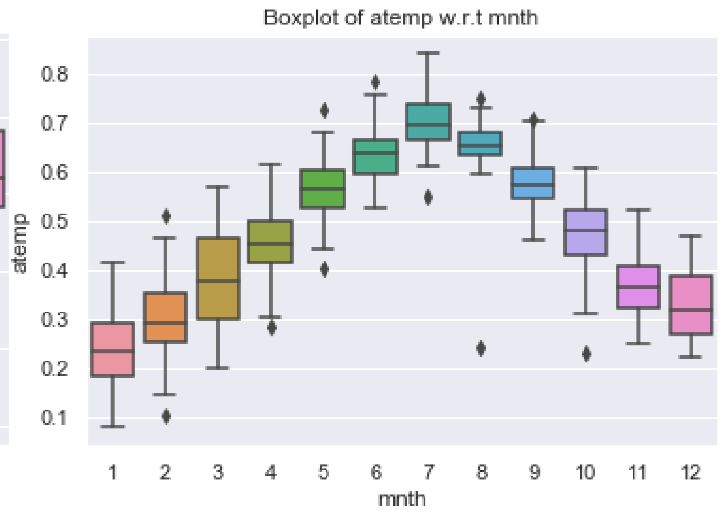


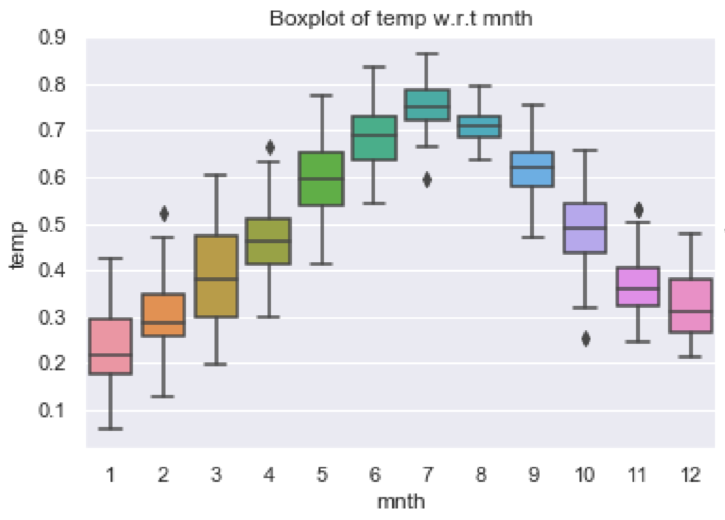
Bivariate Boxplots: Boxplots for all Numerical Variables Vs all Categorical Variables

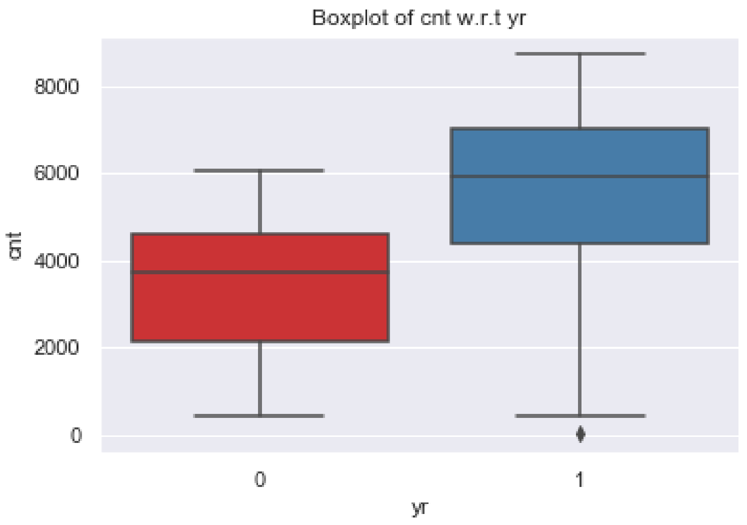


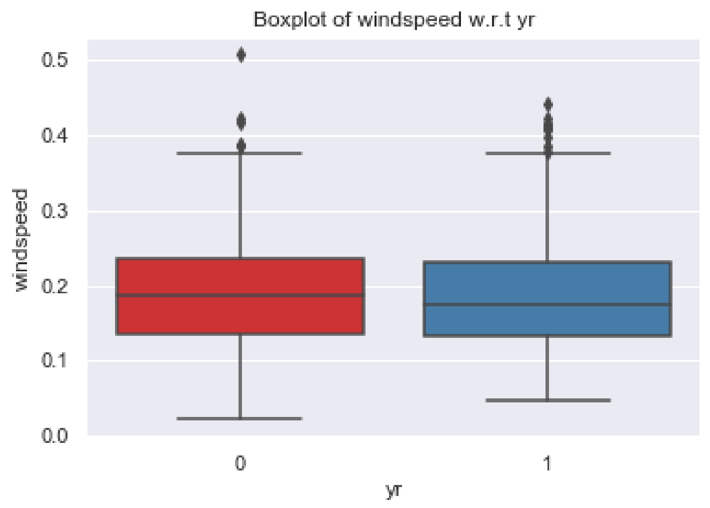




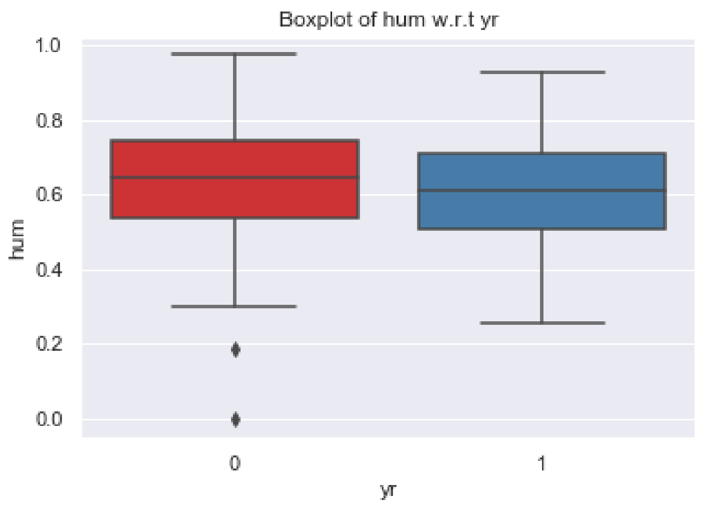




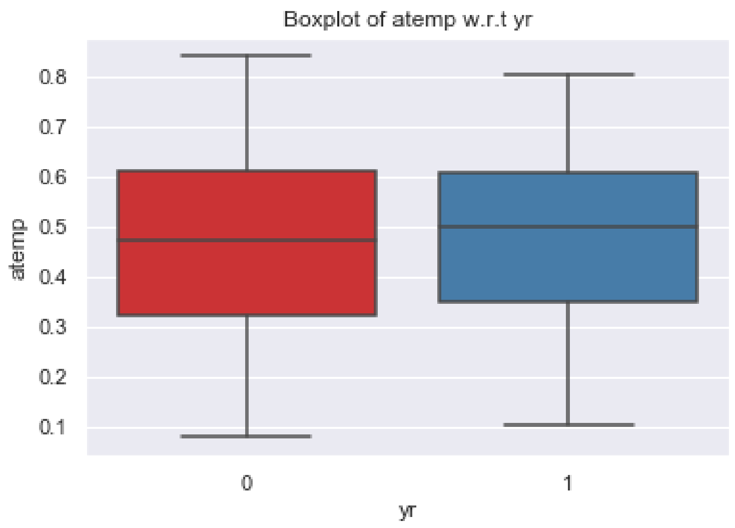




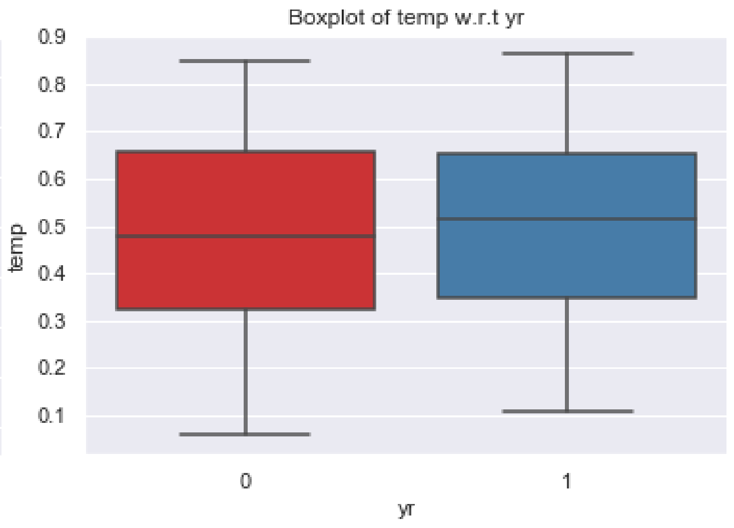
Boxplot of windspeed w.r.t yr
windspeed
0.0
0.1
0.2
0.3
0.4
0.5
0
1
yr



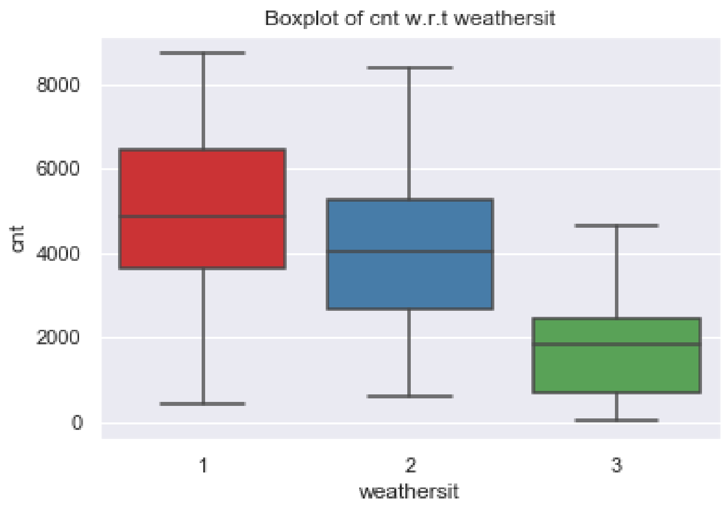
Boxplot of hum w.r.t yr
hum
0.0
0.2
0.4
0.6
0.8
1.0
0
1
yr



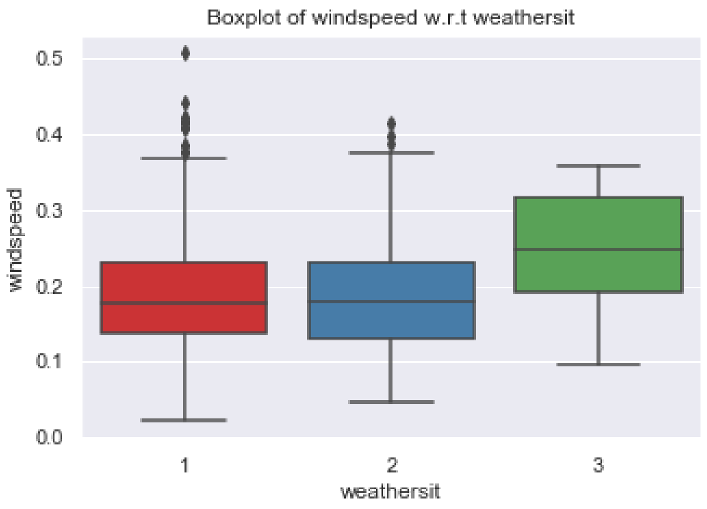
Boxplot of atemp w.r.t yr
atemp
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
yr



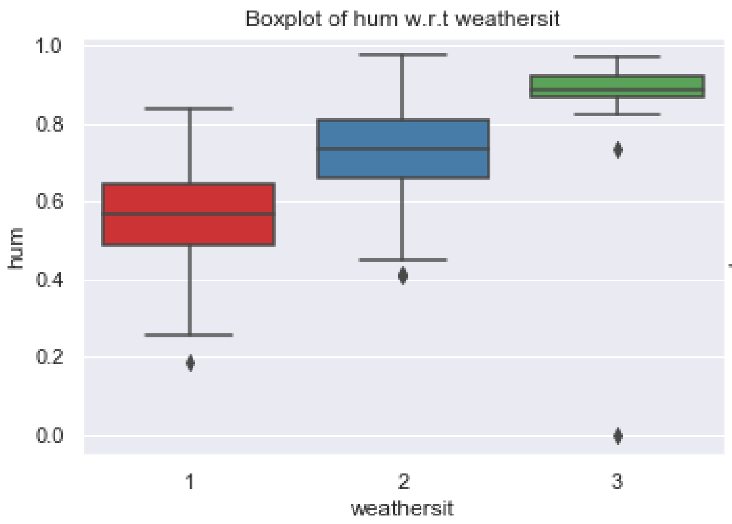
Boxplot of temp w.r.t yr
temp
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
1
yr

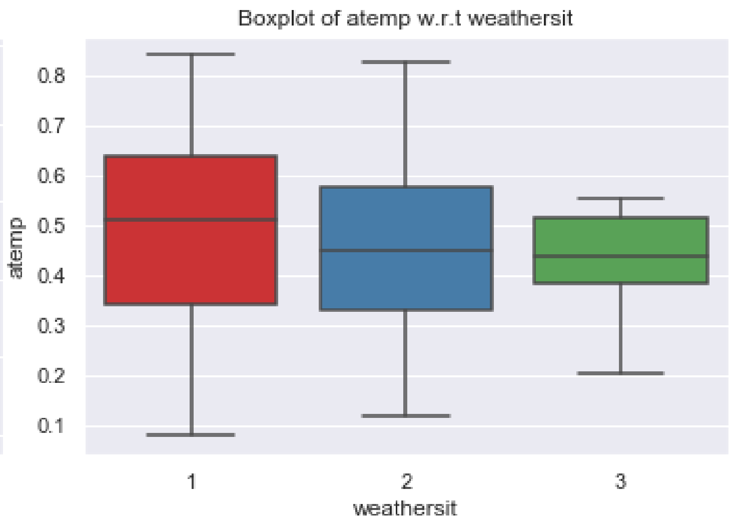


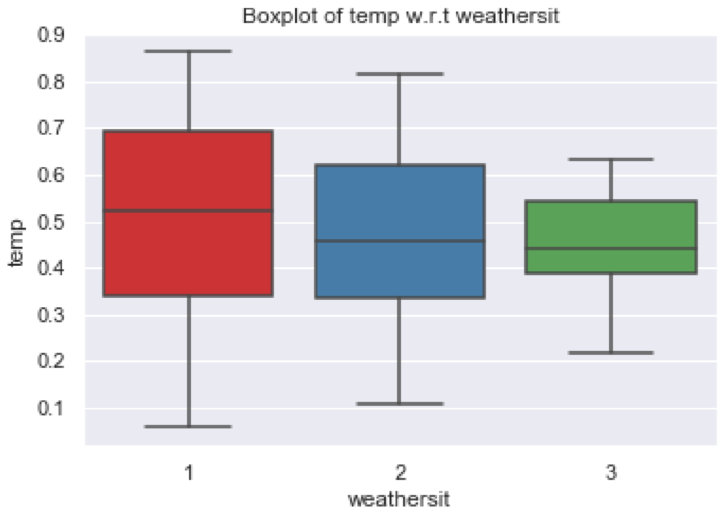
Boxplot of cnt w.r.t weathersit
cnt
0
2000
4000
6000
8000
1
2
3
weathersit

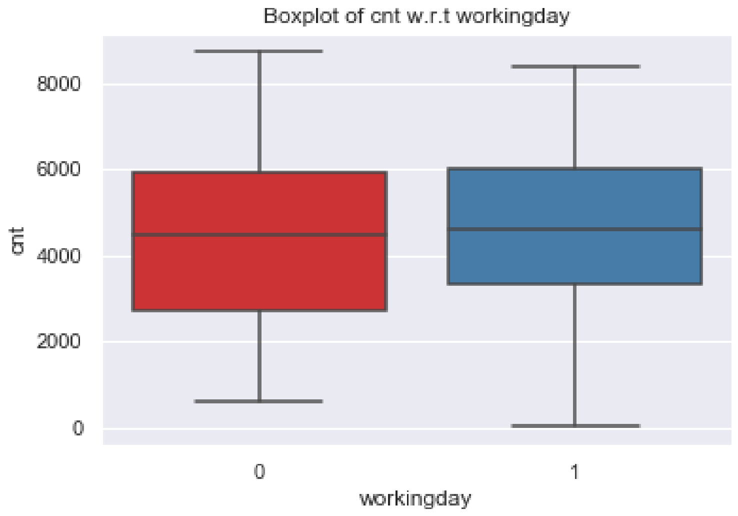


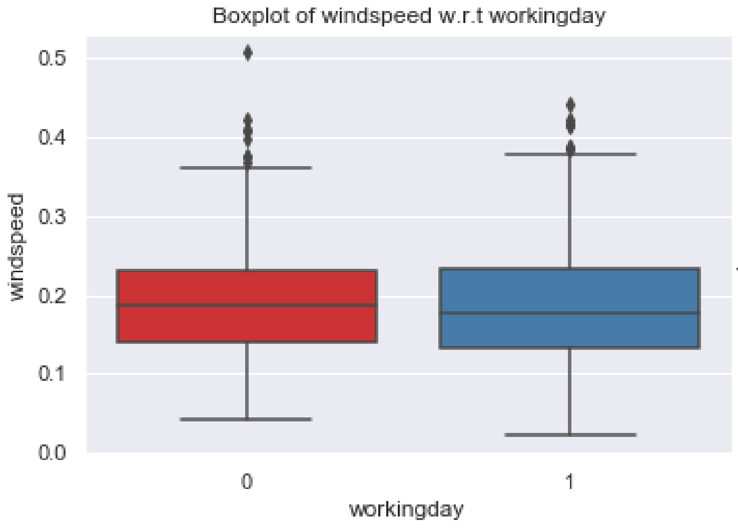
Boxplot of windspeed w.r.t weathersit
windspeed
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
weathersit

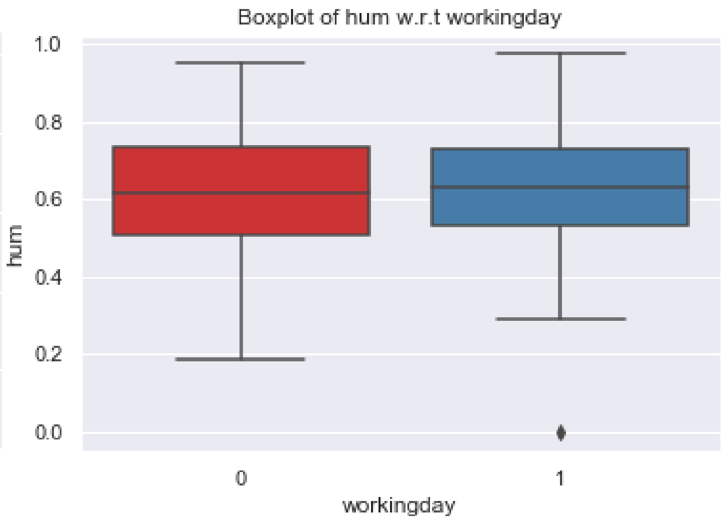

Boxplot of hum w.r.t weathersit
hum
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
weathersit

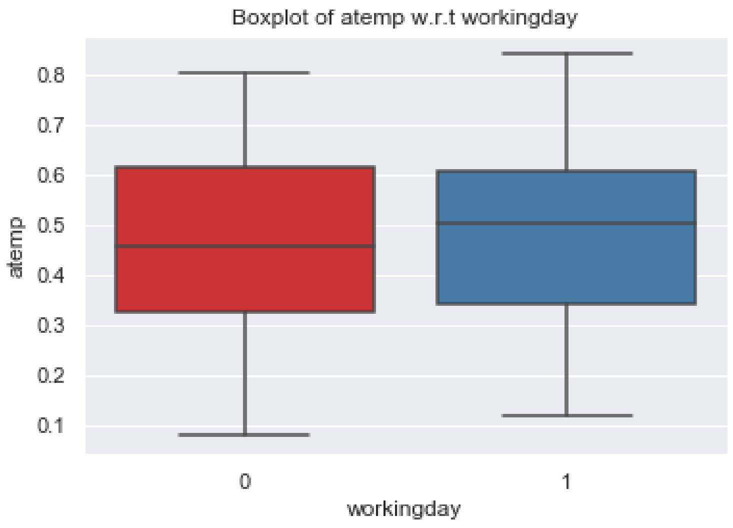

Boxplot of atemp w.r.t weathersit
atemp
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
weathersit

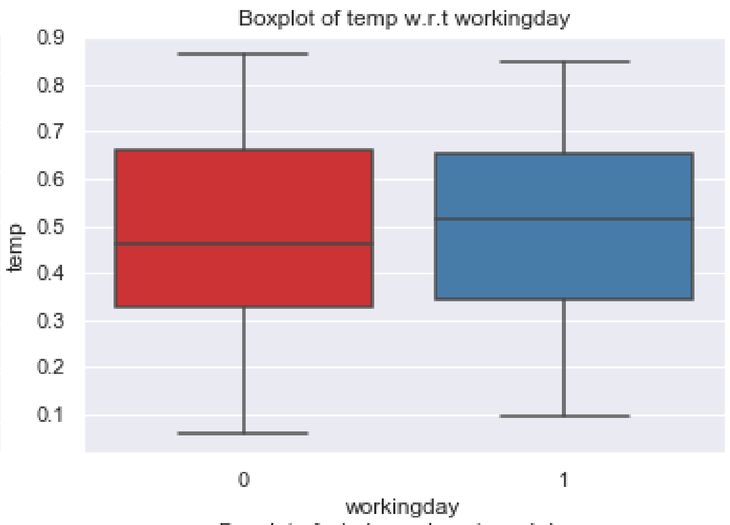

Boxplot of temp w.r.t weathersit
temp
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
weathersit

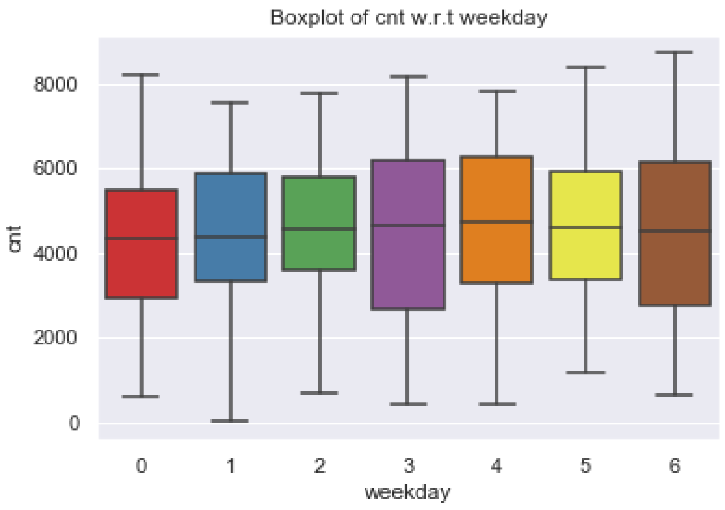

Boxplot of cnt w.r.t workingday
cnt
8000
6000
4000
2000
0
0
1
workingday

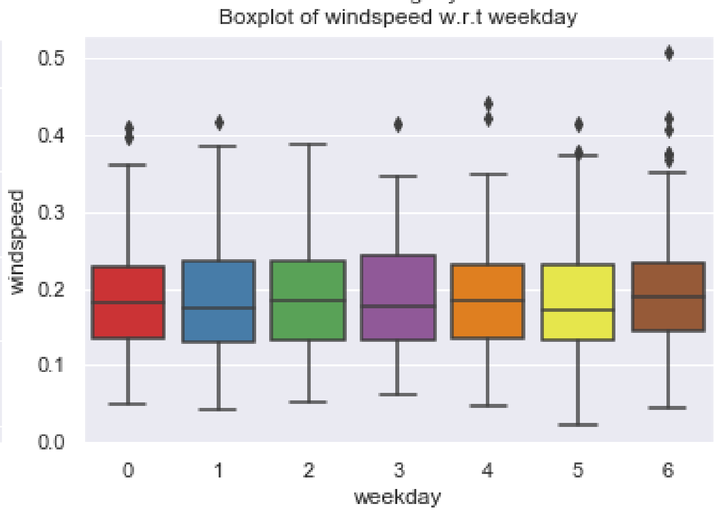

Boxplot of windspeed w.r.t workingday
windspeed
0.5
0.4
0.3
0.2
0.1
0.0
0
1
workingday

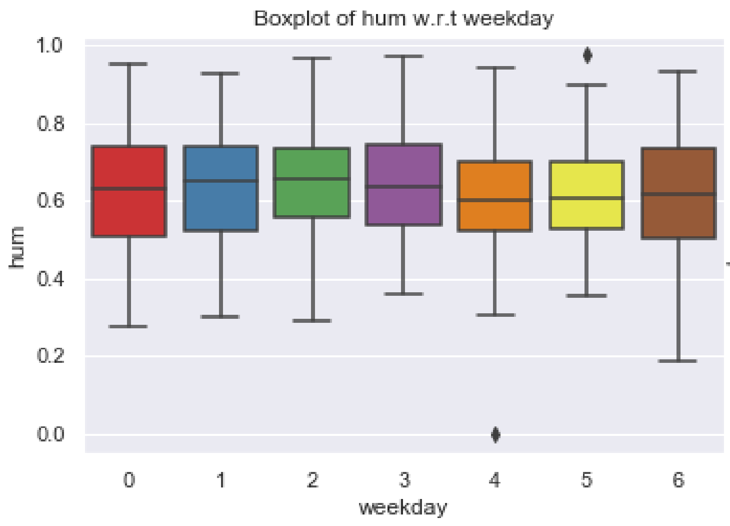

Boxplot of hum w.r.t workingday
hum
1.0
0.8
0.6
0.4
0.2
0.0
0
1
workingday

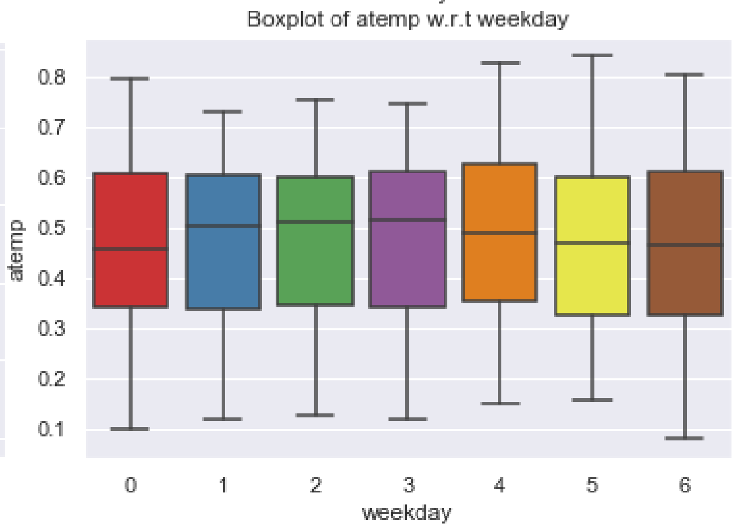

Boxplot of atemp w.r.t workingday
atemp
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
workingday

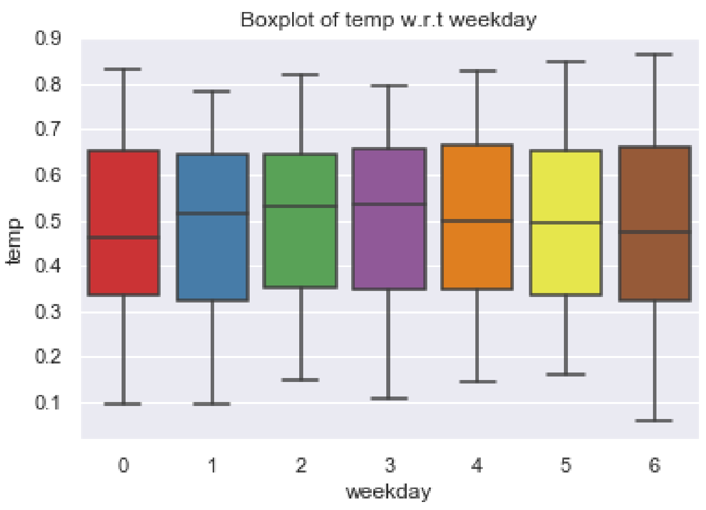

Boxplot of temp w.r.t workingday
temp
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
1
workingday


Boxplot of cnt w.r.t weekday
cnt
0
2000
4000
6000
8000
0
1
2
3
4
5
6
weekday

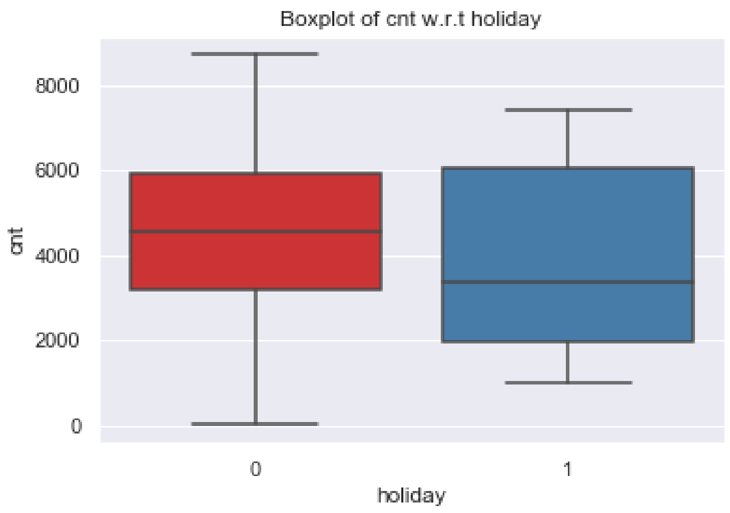

Boxplot of windspeed w.r.t weekday
windspeed
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
weekday


Boxplot of hum w.r.t weekday
hum
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
weekday

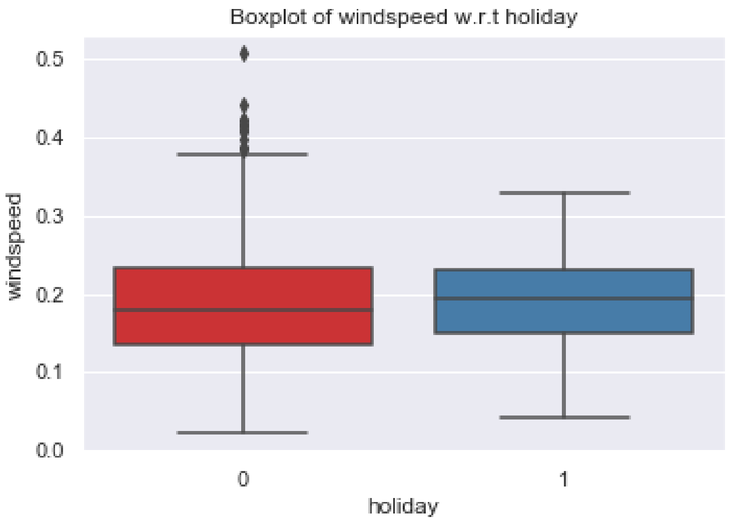

Boxplot of atemp w.r.t weekday
atemp
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
3
4
5
6
weekday



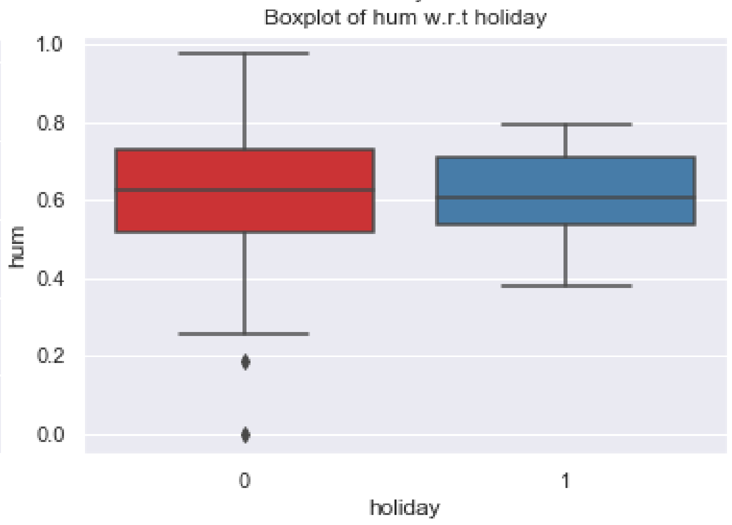
Boxplot of temp w.r.t weekday
temp
weekday
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6



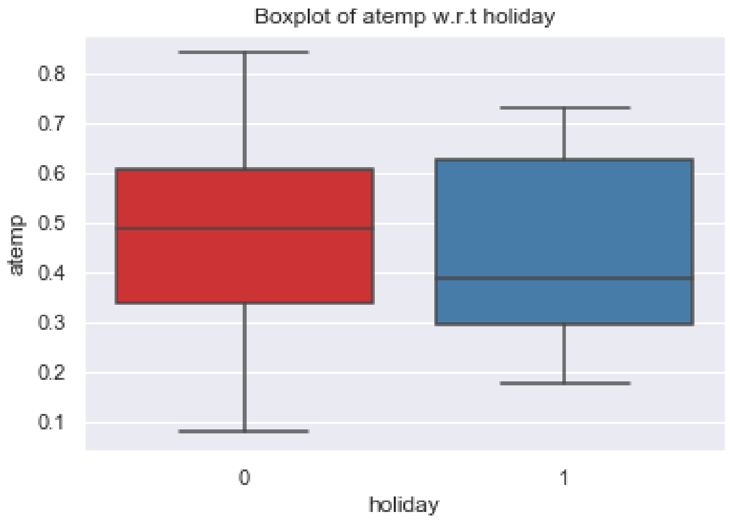
Boxplot of cnt w.r.t holiday
cnt
holiday
8000
6000
4000
2000
0
0
1



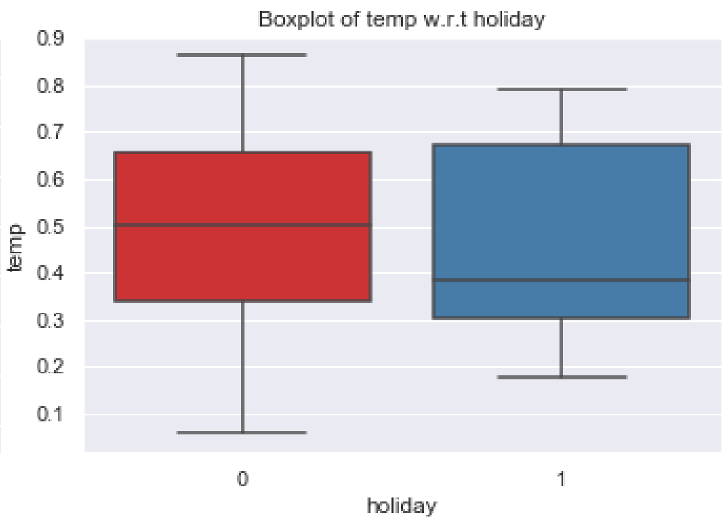
Boxplot of windspeed w.r.t holiday
windspeed
holiday
0.5
0.4
0.3
0.2
0.1
0.0
0
1



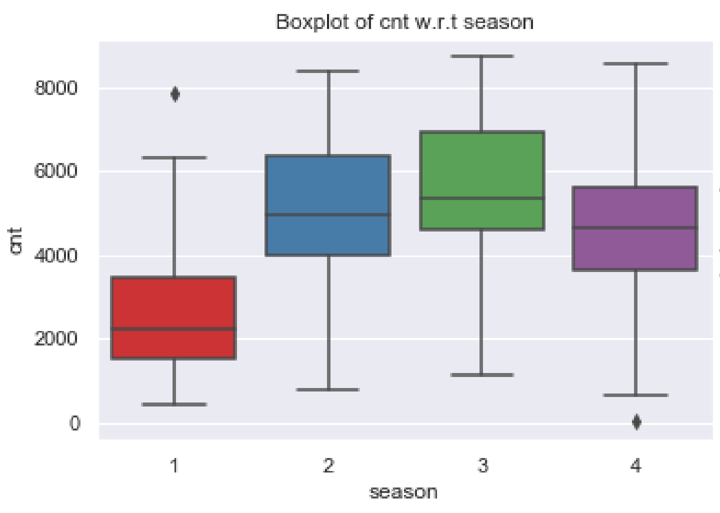
Boxplot of hum w.r.t holiday
hum
holiday
1.0
0.8
0.6
0.4
0.2
0.0
0
1

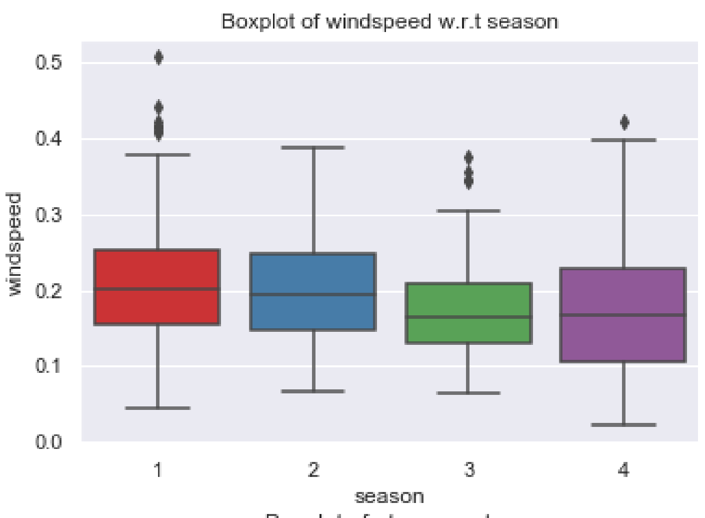


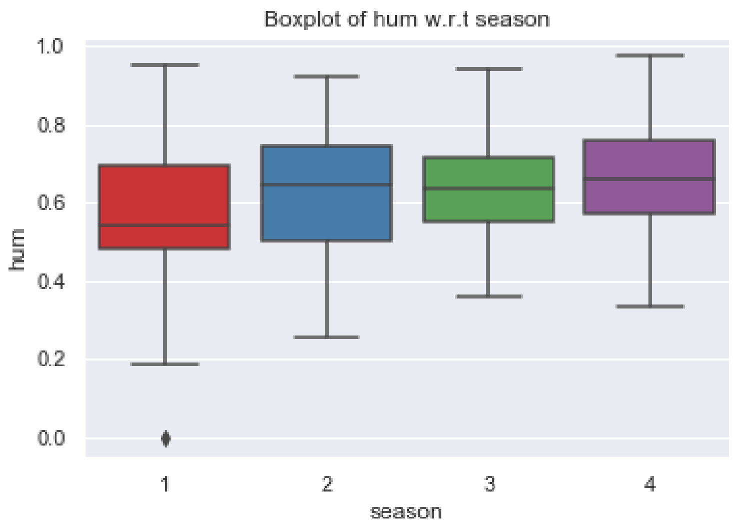
Boxplot of atemp w.r.t holiday
atemp
holiday
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1

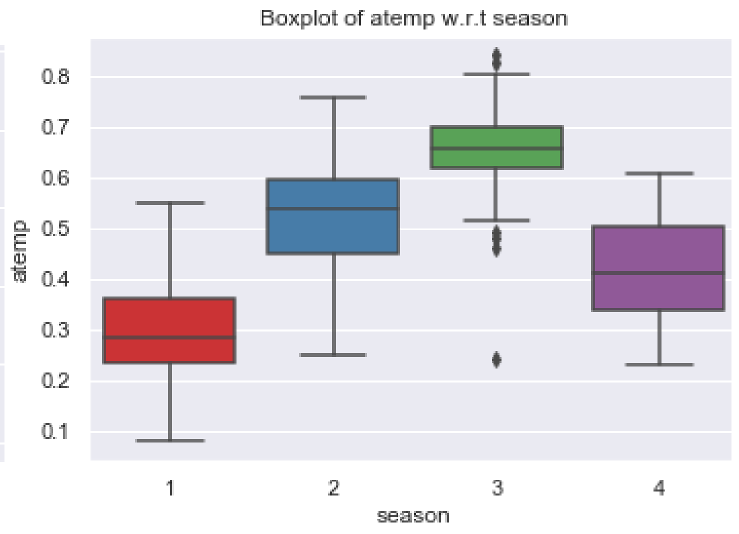


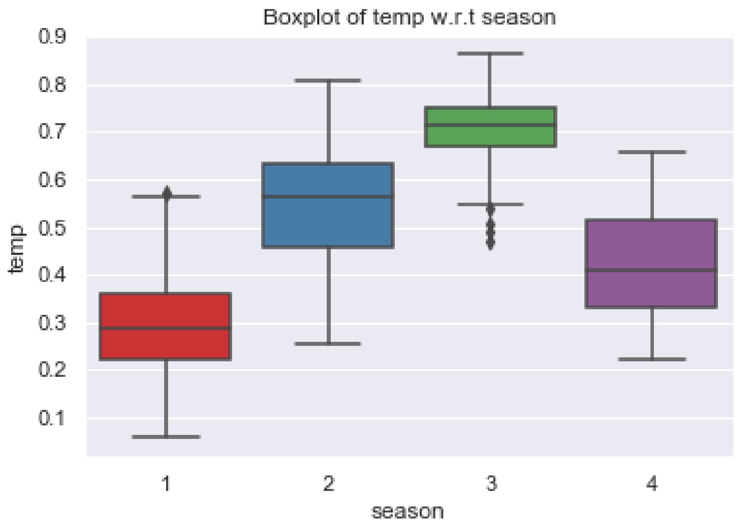
Boxplot of temp w.r.t holiday
temp
holiday
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1











From above Boxplots we see that only ‘hum’ and ‘windspeed’ have outliers in them. ‘hum’ has 2 outliers and ‘windspeed’ has 13 outliers.

Here’s the standard deviation before imputing:

```
hum                 0.142429
windspeed           0.077498
       dtype: float64
```

Here’s the standard deviation after imputing:

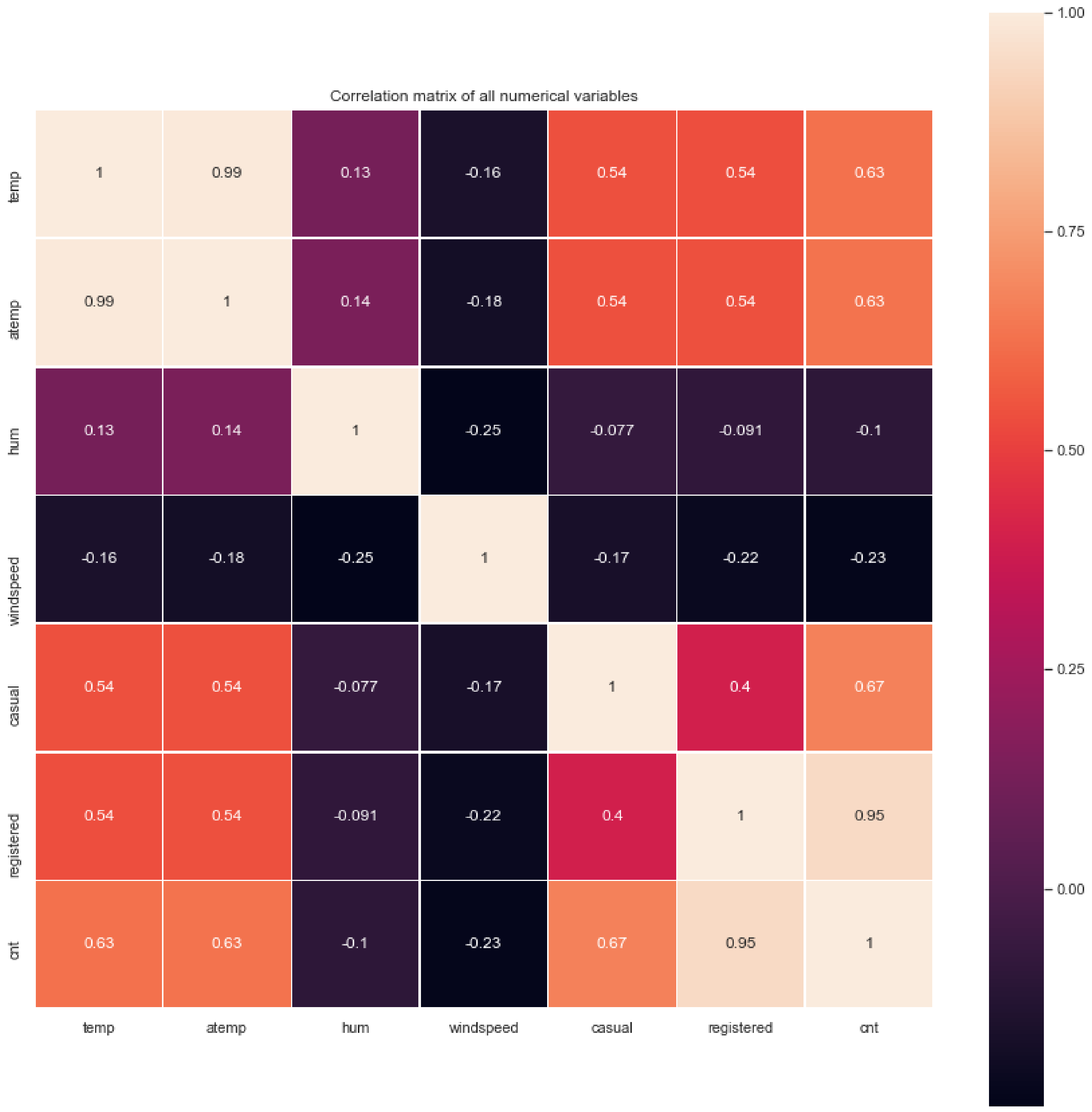
```
hum                 0.140021
windspeed           0.071707
       dtype: float64
```

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.
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.
   - Similarly, 'registered' and 'cnt' are highly correlated with each other.
   - We also came to know that--'cnt'='casual'+'registered' .

Correlation Plot:



	temp	atemp	hum	windspeed	casual	registered	cnt
temp	1	0.99	0.13	-0.16	0.54	0.54	0.63
atemp	0.99	1	0.14	-0.18	0.54	0.54	0.63
hum	0.13	0.14	1	-0.25	-0.077	-0.091	-0.1
windspeed	-0.16	-0.18	-0.25	1	-0.17	-0.22	-0.23
casual	0.54	0.54	-0.077	-0.17	1	0.4	0.67
registered	0.54	0.54	-0.091	-0.22	0.4	1	0.95
cnt	0.63	0.63	-0.1	-0.23	0.67	0.95	1

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. Assumption 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 theorical 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
- hoilday 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 analysing p value of all categorical features, we come to a conclusion that, Variables which we have removed and kept:

Removed: mnth, weekday, workingday, weathersit.

Kept: season, holiday, yr

3 **Analysis of Variance(Anova) Test** –

I. It is carried out to compare between each group in a categorical variable.
II. ANOVA only lets us know the means for different groups are same or not. It doesn't help us identify which mean is different.

Hypothesis testing:

- **Null Hypothesis**: mean of all categories in a variable are same.
- **Alternate Hypothesis**: mean of at least one category in a variable is different.

- If p-value is less than 0.05 then we reject the null hypothesis.
- And if p-value is greater than 0.05 then we accept the null hypothesis.

4 **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 refer 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. Normalisation 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, our most of our data is not distributed normally so we had choose normalization over standardization.
- We have checked variance for each column in dataset before Normalisation
- High variance will affect the accuracy of the model. So, we want to normalise that variance.

Graphs based on which standardization was chosen:
Note: It is performed only on Continuous variables.

Chapter 3

Splitting train and test Dataset

a. With the time series data, we will break up our train and test into continuous chunks.
b. The training data should be the earliest data and test data should be the latest data.
c. 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.
d. 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.

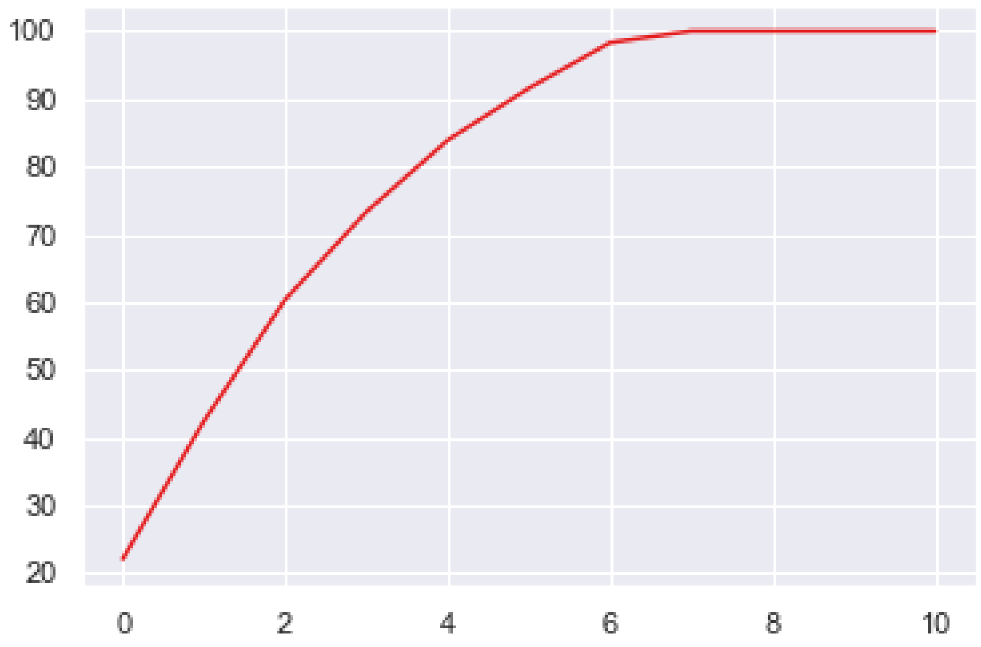
Chapter 4

Dimensionality Reduction using PCA technique

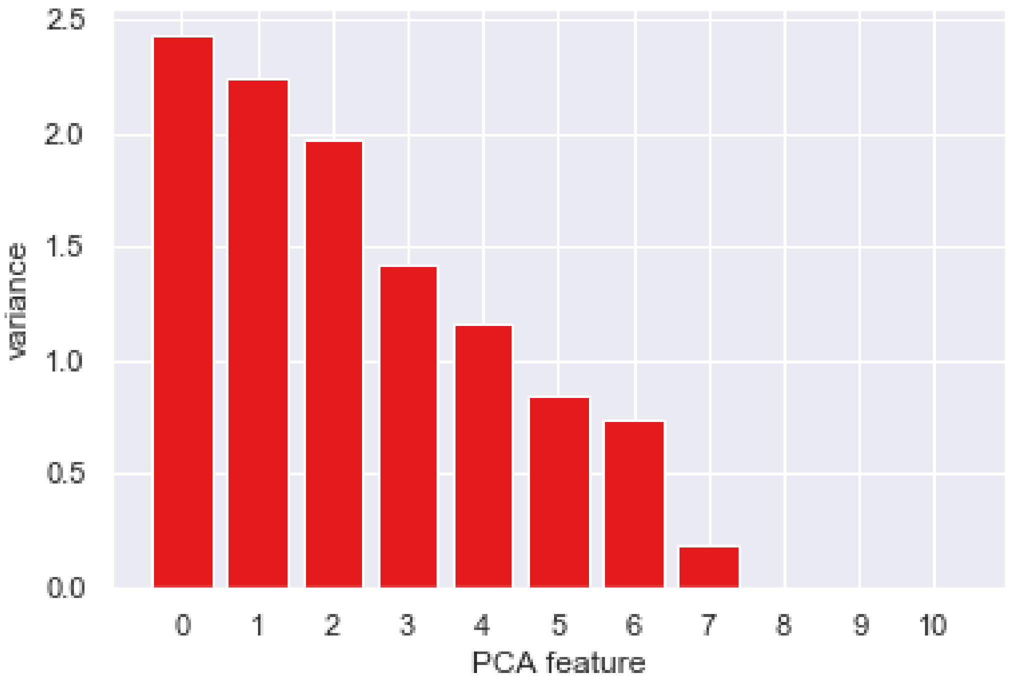
We have used PCA for our Dimensionality reduction technique

I. We have fitted only training dataset in PCA.
II. Dimensionality Reduction finds patterns in the data and uses that pattern to re-express it in a compressed form.
III. Removes less information noise features.
IV. First, we will fit training data in PCA and see the PCA components and we will find which PCA component explains variance and then we will count those number of components and use that number to use only those number of components to filter out PCA components.
V. That number we just found out is called as intrinsic dimension.
VI. The intrinsic dimension is the number of PCA features with significant variance.
VII. So, for our data: intrinsic dimension = 8.
VIII. And then we will transform our training and testing data.
IX. Thus, we have reduced dimension of our data using PCA technique.

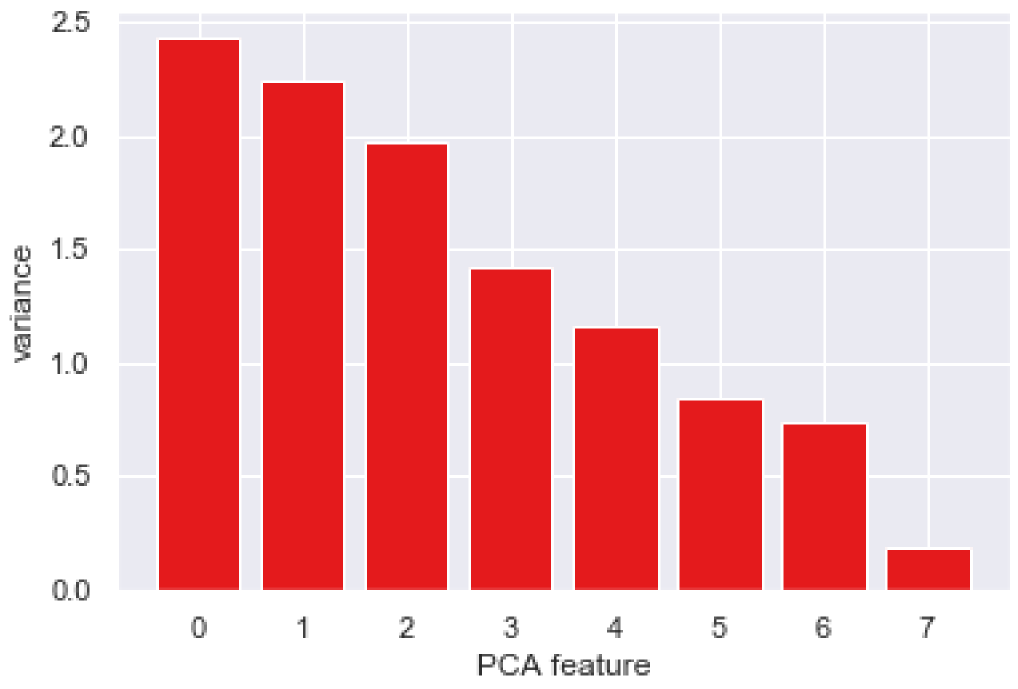
Line plot for cumulative variance explained, after fitting training data in PCA:



Bar plot for variance explained, after fitting training data in PCA:



Bar plot after filtering PCA components:



Chapter 5

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': True}
Best score is 0.32346023751676756

II. Ridge Regression:
Tuned Decision ridge Parameters: {'alpha': 0.10481131341546852, 'max_iter': 500, 'normalize': True}
Best score is 0.3584492123896776

III. Lasso Regression:
Tuned Decision lasso Parameters: {'alpha': 0.0016768329368110067, 'max_iter': 1000, 'normalize': False}
Best score is 0.40677751497154

IV. Decision Tree Regression:
Tuned Decision Tree Parameters: {'max_depth': 6, 'min_samples_split': 4}
Best score is -0.13004573964781863

V. Random Forest Regression:
Tuned Decision Forest Parameters: {'max_depth': 15, 'min_samples_split': 2, 'n_estimators': 100}
Best score is 0.353453500891312

Chapter 6

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 5 Regression Algorithms:

I. Linear Regression
II. Ridge Regression
III. Lasso Regression
IV. Decision Tree
V. 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)
- RMSLE(Root Mean Squared Log Error)

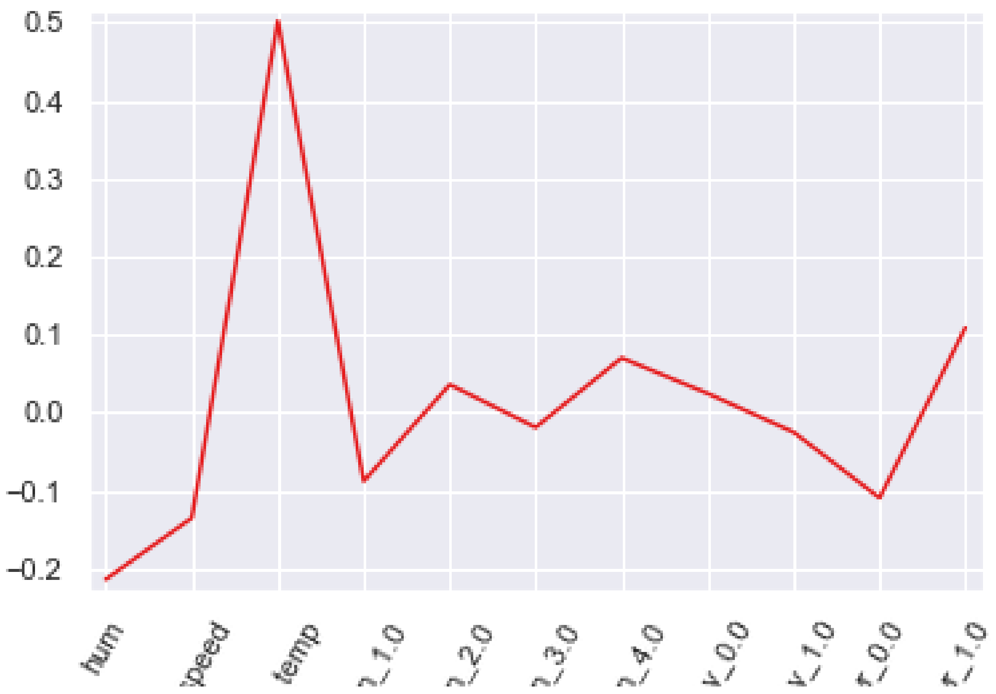
2.3.1 Model Performance—Without PCA

Here, we will evaluate the performance of different Regression models based on different Error Metrics

I. Linear Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.819	0.8163	18.88	0.0076	0.0872	0.058
Test	0.554	0.5177	Inf	0.020	0.144	0.093

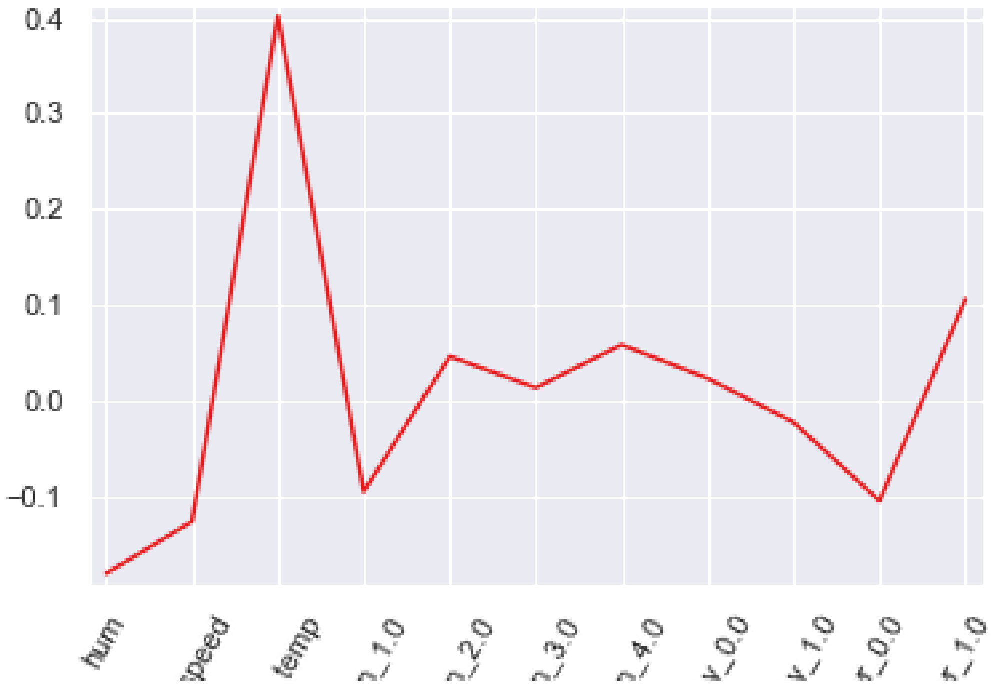
Line Plot for Coefficients of Linear regression:



II. Ridge Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.814	0.810	19.66	0.0078	0.088	0.060
Test	0.51	0.477	Inf	0.022	0.1498	0.097

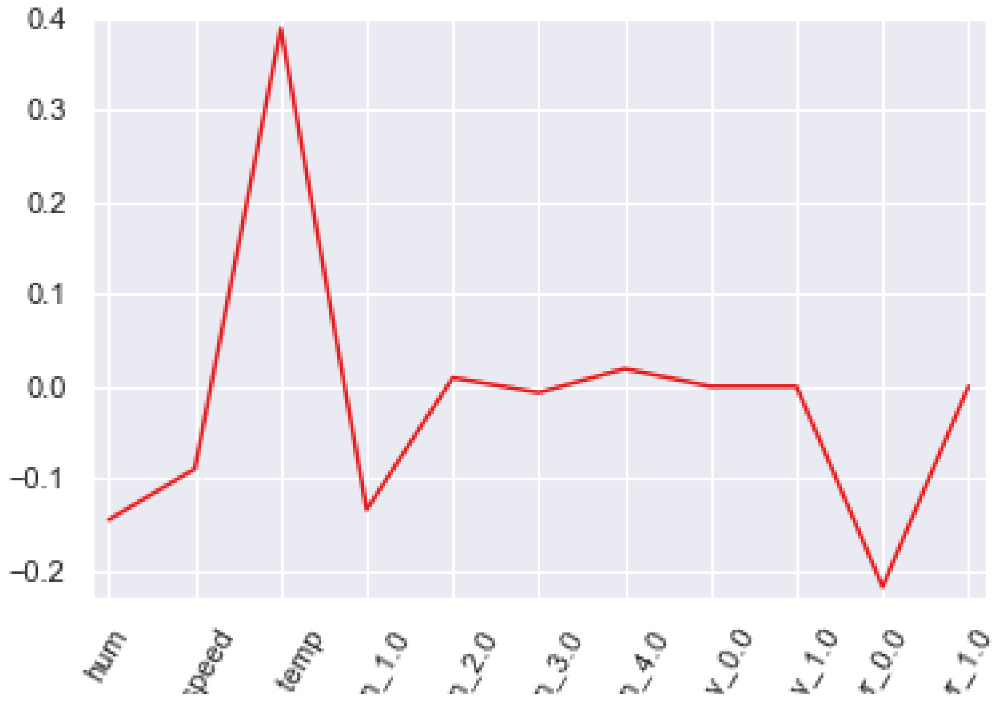
Line Plot for Coefficients of Ridge regression:



III. Lasso Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.807	0.803	20.31	0.008	0.090	0.061
Test	0.511	0.472	Inf	0.022	0.150	0.098

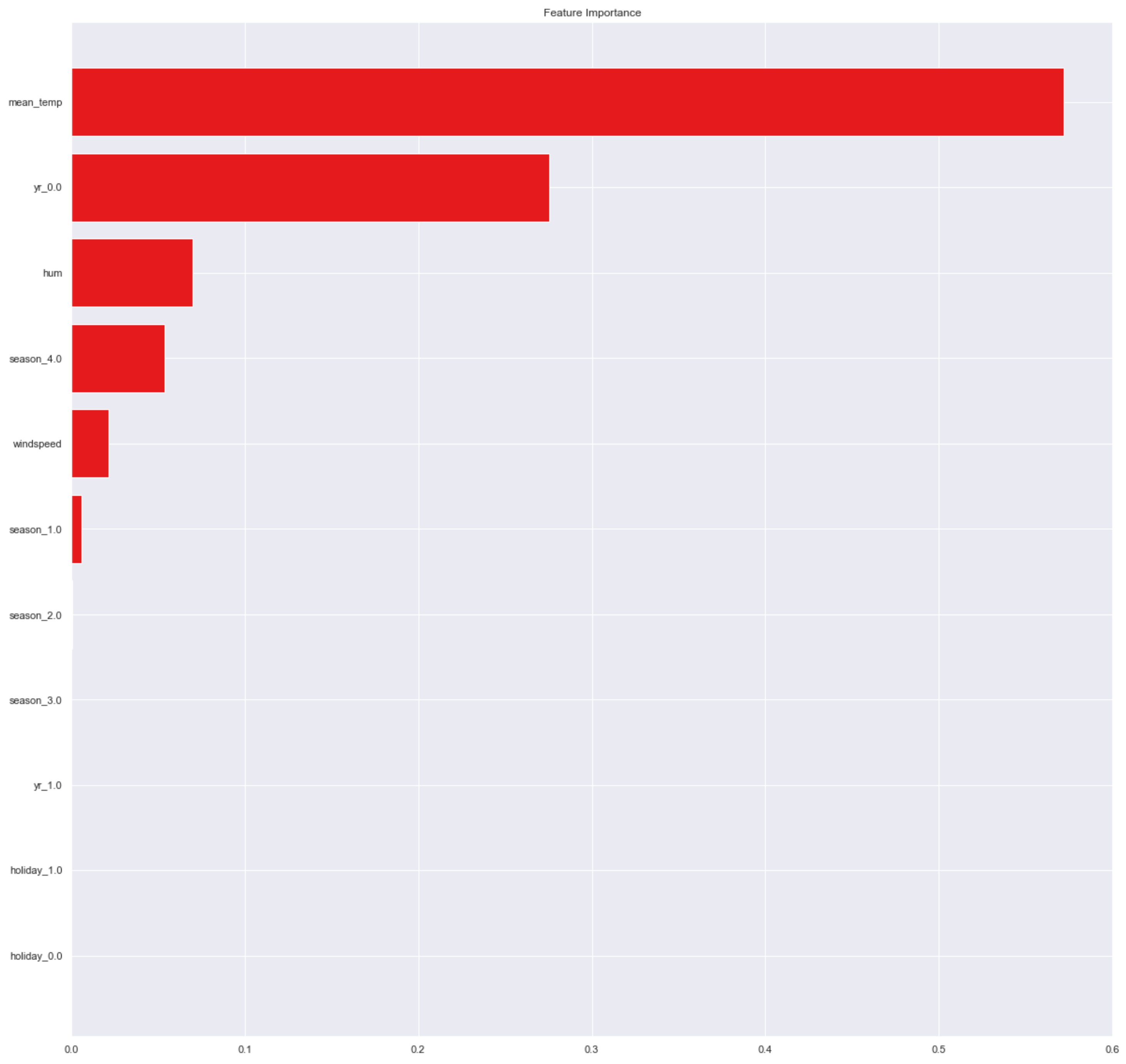
Line Plot for Coefficients of Lasso regression:



IV. Decision Tree Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.913	0.911	11.85	0.003	0.060	0.040
Test	0.519	0.480	Inf	0.022	0.149	0.097

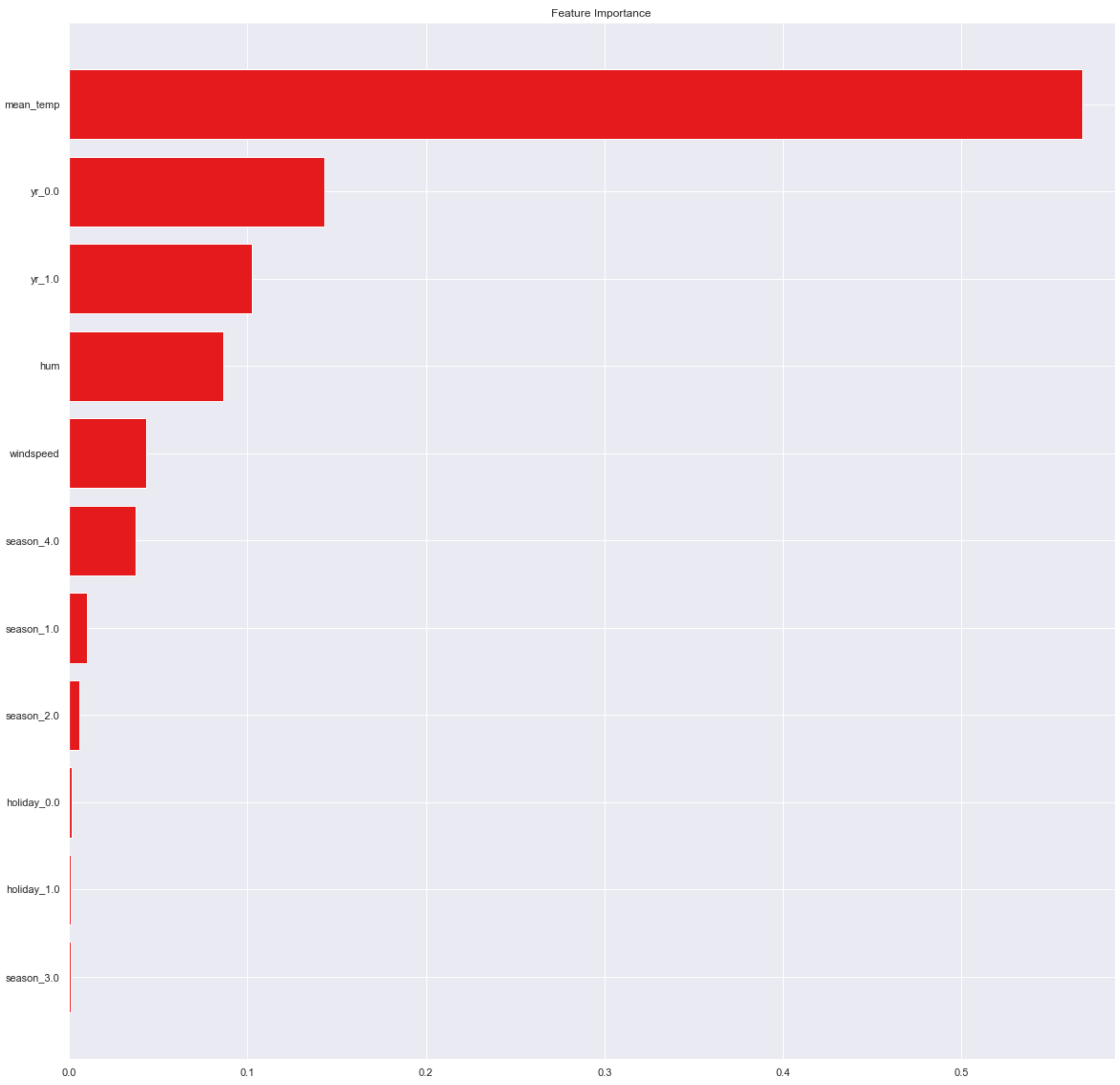
Bar Plot of Decision tree Feature Importance:



V. Random Forest Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.981	0.98	6.29	0.0007	0.027	0.019
Test	0.551	0.515	Inf	0.020	0.144	0.093

Bar Plot of Random Forest Feature Importance:



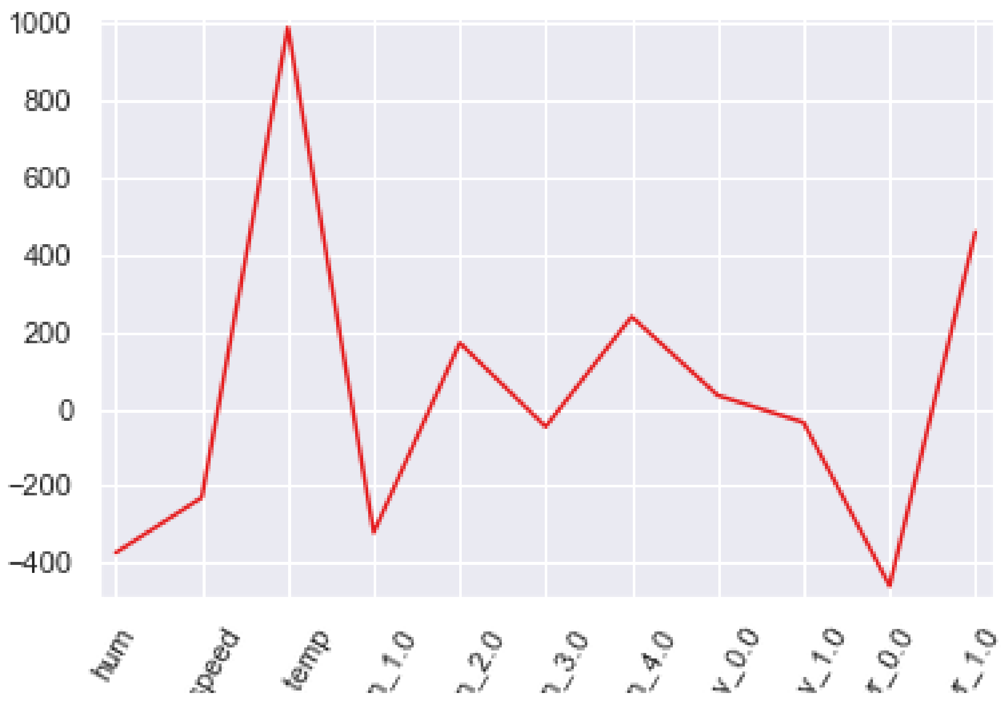
2.3.1 Model Performance—With PCA

Here, we will evaluate the performance of different Regression models based on different Error Metrics

I. Linear Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.819	0.816	18.66	575780.07	758.80	0.24
Test	0.554	0.51779	165.99	1566482.82	1251.59	0.54

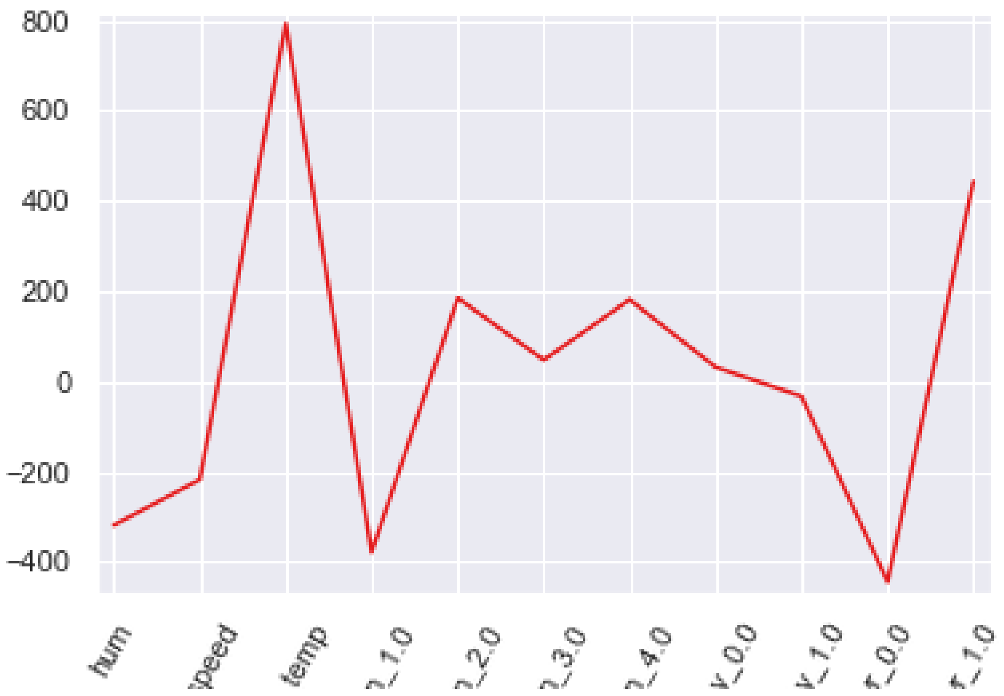
Line Plot for Coefficients of Linear regression:



II. Ridge Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.814	0.810	19.42	593902.81	770.65	0.245
Test	0.517	0.477	166.77	1696674.40	1302.56	0.54

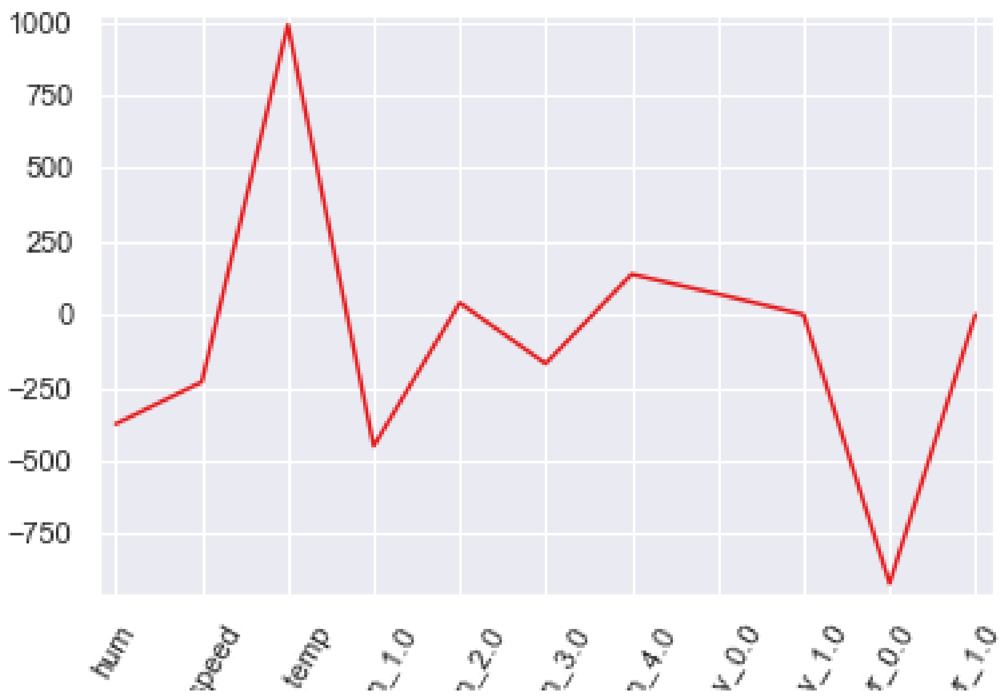
Line Plot for Coefficients of Ridge regression:



III. Lasso Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.819	0.8163	18.66	575780.07	758.80	0.24
Test	0.554	0.517	165.99	1566486.24	1251.59	0.54

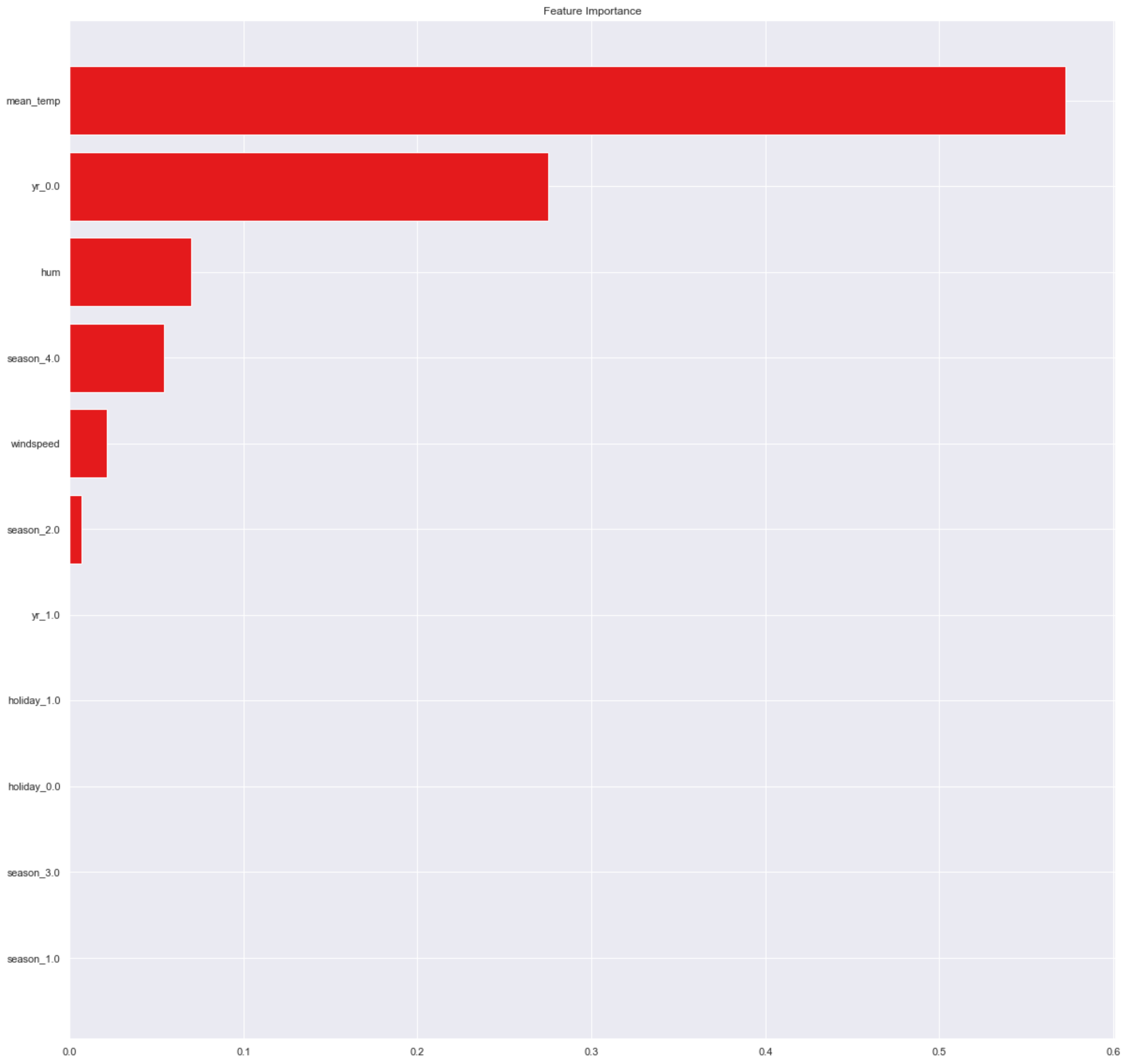
Line Plot for Coefficients of Lasso regression:



IV. Decision Tree Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.913	0.911	11.73	276087.26	525.44	0.160
Test	0.511	0.471	167.54	1718405.67	1310.87	0.543

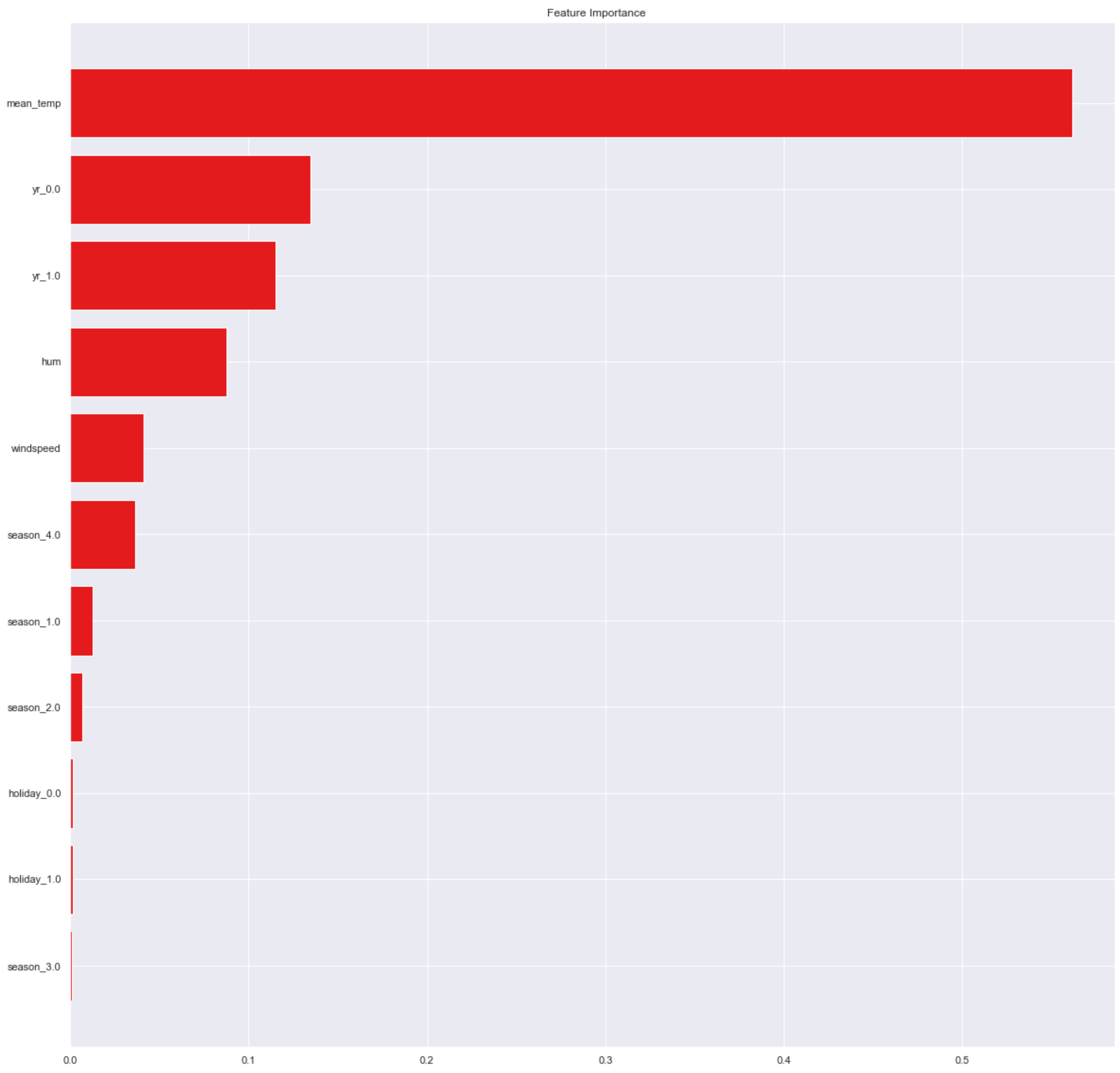
Bar Plot of Decision tree Feature Importance:



V. Random Forest Regression:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.980	0.980	6.41	61368.633	247.72	0.110
Test	0.553	0.5167	162.20	1570322.70	1253.12	0.535

Bar Plot of Random Forest Feature Importance:



Chapter 7

Conclusion

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

I. Random Forest Regression with PCA:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.980	0.980	6.41	61368.633	247.72	0.110
Test	0.553	0.5167	162.20	1570322.70	1253.12	0.535

II. Random Forest Regression without PCA:

Error Metrics	r square	Adj r sq	MAPE	MSE	RMSE	RMSLE
Train	0.981	0.98	6.29	0.0007	0.027	0.019
Test	0.551	0.515	Inf	0.020	0.144	0.093

Predicted Bike Rental Count By Random Forest Regression Model Using PCA.

	cnt
dteday	
2012-08-07	7301.233364
2012-08-08	7302.736364
2012-08-09	6631.646463
2012-08-10	6170.619286
2012-08-11	6468.002879

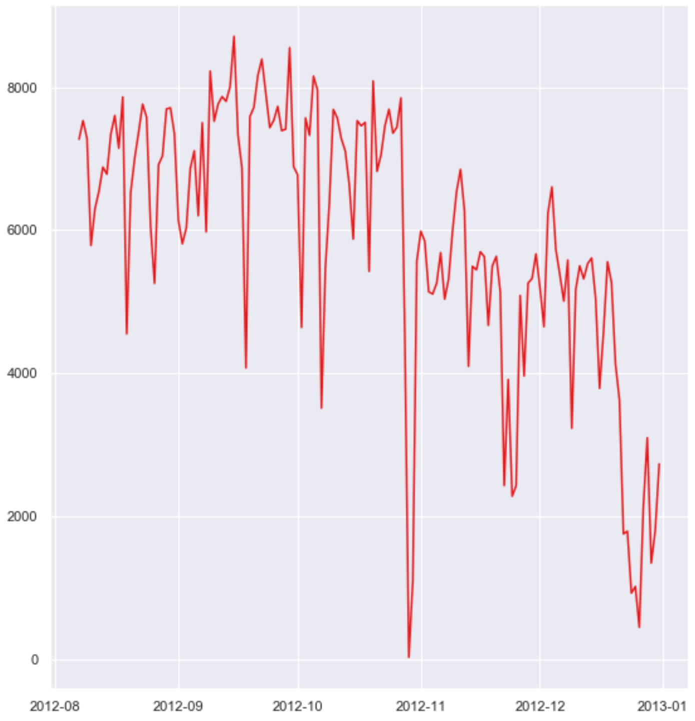
Predicted Bike Rental Count By Random Forest Regression Model without Using PCA.

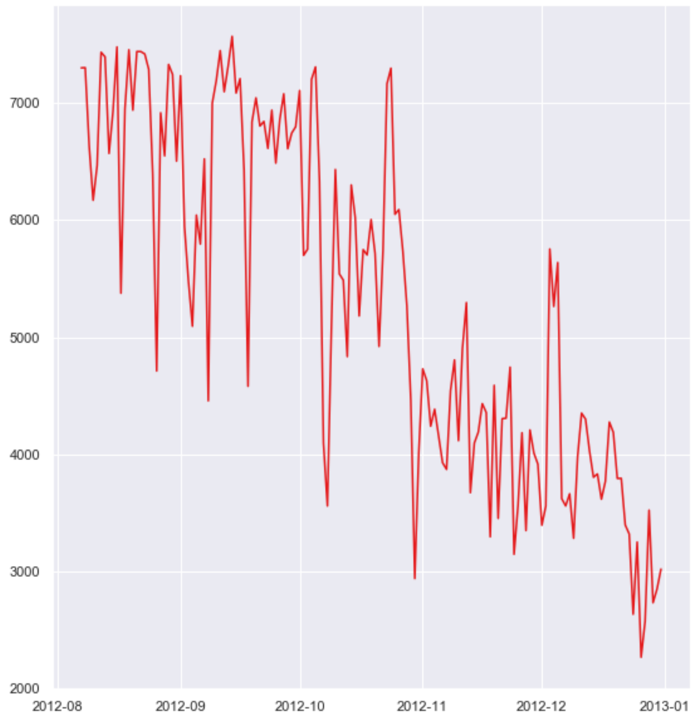
	cnt
dteday	
2012-08-07	7164.005139
2012-08-08	7216.232500
2012-08-09	6661.975841
2012-08-10	5995.545167
2012-08-11	6379.173818

Original Bike Rental Count.

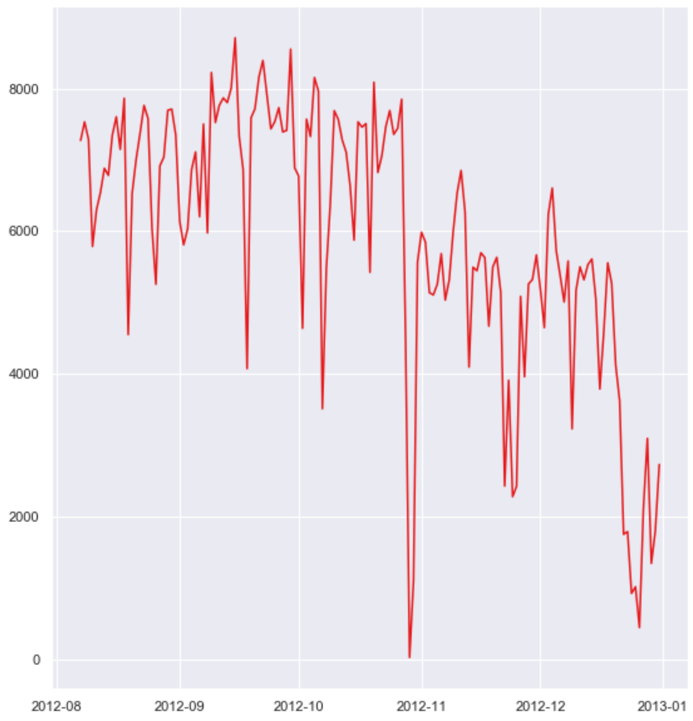
	cnt
dteday	
2012-08-07	7273.0
2012-08-08	7534.0
2012-08-09	7286.0
2012-08-10	5786.0
2012-08-11	6299.0

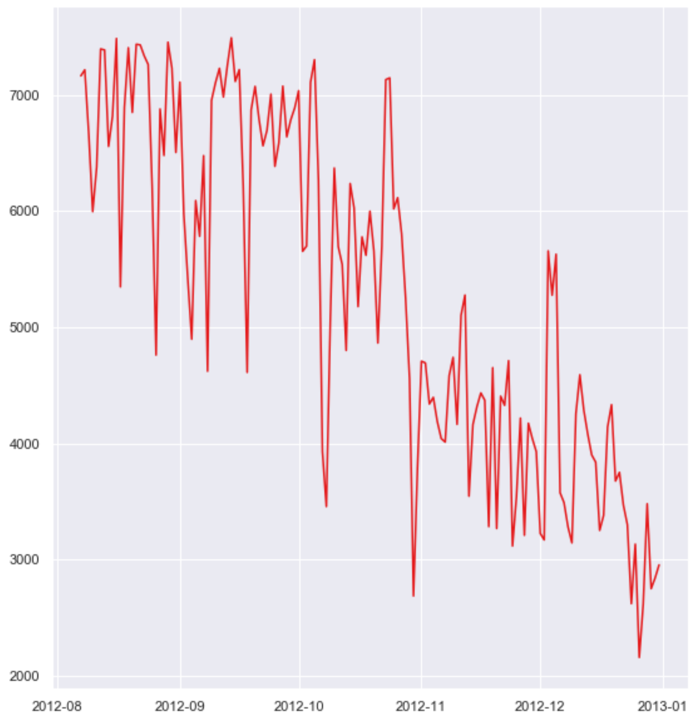
Line plot of original Vs Predicted Bike Rental count with PCA:





Line plot of original Vs Predicted Bike Rental count without PCA:





Chapter 8

Python-Code

Predicting Bike Rental Count using Python

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

```
# loading the required libraries

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from pandas.plotting import register_matplotlib_converters

register_matplotlib_converters()

import datetime

import scipy.stats as stats

from sklearn.preprocessing import StandardScaler

from fancyimpute import KNN

from scipy.stats import chi2_contingency

from patsy import dmatrices

import statsmodels.api as sm

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.decomposition import PCA

import statsmodels.api as sm

from statsmodels.formula.api import ols

from sklearn import metrics

# set the working directory

os.chdir('C:/Users/admin/Documents/Python Files')

os.getcwd()
```

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)
   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 + Scatteredclouds
      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_max-t_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
```

```
# Importing data

df = pd.read_csv('day.csv',index_col='dteday',parse_dates=True,infer_datetime_format=True,dayfirst=True)

# Dropping the 1st column i.e. 'instant' Coz it is seems statisticaly insignificant

df=df.drop('instant',axis=1)

df.head(5)

df.describe(),df.info()
```

   We have used dteday as index and converted it into DatetimeIndex

```
type(df.index)
```

Graphical EDA - Data Visualization

```
# setting up the sns for plots

sns.set(style='darkgrid',palette='Set1')

plt.figure(figsize=(10,10))

_ = sns.violinplot(x='weekday',y='cnt',data=df,inner=None,color='lightgray')

_ = sns.stripplot(x='weekday',y='cnt',data=df,size=4,jitter=True)

plt.title('')

plt.show()
```

We will plot some Bee Swarmplots coz unlike histogram there is no binnig bias involved

```
plt.figure(figsize=(25,25))

plt.subplot(321)

_ = sns.swarmplot(x='season',y='cnt',data=df,hue='workingday',size=8)

plt.title('Bike Count w.r.t season and working day')

plt.subplot(322)

_ = sns.swarmplot(x='weekday',y='cnt',data=df,hue='workingday',size=8)

plt.title('Bike Count w.r.t weekday and working day')

plt.subplot(323)

_ = sns.swarmplot(x='holiday',y='cnt',data=df,hue='workingday',size=8)

plt.title('Bike Count w.r.t holiday and working day')

plt.subplot(324)

_ = sns.swarmplot(x='weathersit',y='cnt',data=df,hue='workingday',size=8)

plt.title('Bike Count w.r.t weathersit and working day')

plt.subplot(325)

_ = sns.swarmplot(x='yr',y='cnt',data=df,hue='workingday',size=8)

plt.title('Bike Count w.r.t yr and working day')

plt.subplot(326)

_ = sns.swarmplot(x='mnth',y='cnt',data=df,hue='workingday',size=8)
```

```
plt.title('Bike Count w.r.t mnth and working day')

# plt.savefig('Bee Swarmplots.png')

plt.show()

temp = df['temp']

temp=temp['2011']

atemp = df['atemp']

atemp = atemp['2011']

hum = df['hum']

hum = hum['2011']

windspeed = df['windspeed']

windspeed = windspeed['2011']

temp_index = temp.index[::30]

labels = temp_index.strftime('%b %d')

labels
```

Now we will do some Time Series Analysis

```
plt.figure(figsize=(20,20))

plt.subplot(411)

plt.plot(temp,color = 'r',label = 'temp')

plt.legend(loc = 1)

plt.subplot(412)

plt.plot(atemp,color = 'b',label = 'atemp')

plt.legend(loc = 1)

plt.subplot(413)

plt.plot(hum,color = 'k',label = 'humidity')

plt.legend(loc = 1)

plt.subplot(414)

plt.plot(windspeed,color = 'c',label = 'windspeed')

plt.xticks(temp_index,labels,rotation=60)
plt.legend(loc = 1)

# plt.savefig('Time.png')

plt.show()
```

Now we will see at some Jointplots.
They are used for Bivariate Analysis.
Here we have plotted Scatter plot with Regression line between 2 variables along with separate Bar plots of both variables.
Also we have annotated pearson correlation coefficient and p value.

```
_ = sns.jointplot(x='cnt',y='temp',data=df,kind = 'reg')

_.annotate(stats.pearsonr)

# plt.savefig('jointct.png')

plt.show()

_ = sns.jointplot(x='cnt',y='atemp',data=df,kind = 'reg')

_.annotate(stats.pearsonr)

# plt.savefig('jointcat.png')

plt.show()

_ = sns.jointplot(x='cnt',y='hum',data=df,kind = 'reg')

_.annotate(stats.pearsonr)

# plt.savefig('jointch.png')

plt.show()

_ = sns.jointplot(x='cnt',y='windspeed',data=df,kind = 'reg')

_.annotate(stats.pearsonr)

# plt.savefig('jointcw.png')

plt.show()
```

EDA - Data type conversion

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

df[cat_var]=df[cat_var].apply(lambda x: x.astype('category') )

num_var=['temp','atemp','hum','windspeed','cnt']

#Pairplot for all numerical variables
```

```
_ =sns.pairplot(data=df[num_var],kind='scatter')

_.fig.suptitle('Pairwise plot of all numerical variables')

# plt.savefig('Pairwise.png')

plt.show()
```

Missing Value Analysis

```
pd.DataFrame(df.isnull().sum())

df.info()
```

Outlier Analysis using Boxplot

Univariate Boxplots: Boxplots for all Numerical Variables also for target variable.

```
for i in num_var:

    sns.boxplot(y=i,data=df)

    plt.title('Boxplot of '+i)

#     plt.savefig('bp'+str(i)+'.png')

    plt.show()
```

Bivariate Boxplots: Boxplots for all Numerical Variables Vs all Categorical Variables

```
for a in cat_var:

    for b in num_var:

        _ = sns.boxplot(x=a,y=b,data=df)

        plt.title('Boxplot of '+b+' w.r.t '+a)

#         plt.savefig('Boxplot of '+str(b)+' w.r.t '+str(a)+'.png')

        plt.show()
```

Outlier Treatment

As we can see from the above Boxplots only 'hum' and 'windspeed' columns in the dataset has outliers

```
df.std()
```

Std Deviation before outlier treatment :

standard deviation for 'hum'= 0.142429
standard deviation for 'windspeed'= 0.077498

```
def outlier_treatment(col):
    ''' calculating outlier indices and replacing them with NA  '''

    #Extract quartiles

    q75, q25 = np.percentile(df[col], [75 ,25])

    #Calculate IQR

    iqr = q75 - q25

    #Calculate inner and outer fence

    minimum = q25 - (iqr*1.5)

    maximum = q75 + (iqr*1.5)

    #Replace with NA

    df.loc[df[col] < minimum,col] = np.nan

    df.loc[df[col] > maximum,col] = np.nan

outlier_treatment('hum')

df['hum'].isnull().sum()
```

We have checked standard deviation for 3 imputation methods-mean,median,KNN and standard deviation remains nearly same with KNN imputation. so, KNN is selected for imputation method.

And we have also checked standard deviation for different values of K for KNN and we have selected K=3 for 'hum' and K=1 for 'windspeed'.

```
#Imputing with missing values using KNN

df = pd.DataFrame(KNN(k = 3).fit_transform(df), columns = df.columns, index=df.index)

outlier_treatment('windspeed')

df['windspeed'].isnull().sum()

#Imputing with missing values using KNN

df = pd.DataFrame(KNN(k = 1).fit_transform(df), columns = df.columns, index=df.index)

# #Impute with mean

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

# #Impute with median

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

df.std()
```

Std Deviation after outlier treatment :

standard deviation for 'hum'= 0.140021
standard deviation for 'windspeed'= 0.071707

```
df.head()
```

Feature Selection
Correlation Analysis for Numerical Variables/Features

We will plot a Heatmap of correlation whereas, correlation measures how strongly 2 quantities are related to each other.
We go for correlation to avoid redundant information in our model development.

```
# heatmap using correlation matrix

plt.figure(figsize=(15,15))

_ = sns.heatmap(df[['temp','atemp','hum','windspeed','casual',
'registered','cnt']].corr(),linewidths=0.5,linecolor='w',square=True,annot=True)

plt.title('Correlation matrix of all numerical variables')

# plt.savefig('correlation.png')

plt.show()
```

'temp' and 'atemp' are very highly correlated with each other.
similarly, 'registered' and 'cnt' are highly correlated with each other.
we also came to know that--'cnt'='casual'+'registered'

Chi-square test of Independence for Categorical Variables/Features

Hypothesis testing :
Null Hypothesis: 2 variables are independent.
Alternate Hypothesis: 2 variables are not independent.
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.
There should be no dependencies between Independent variables.
So we will remove that variable whose p-value with other variable is low than 0.05.
And we will keep that variable whose p-value with other variable is high than 0.05

```
#loop for chi square values

for i in cat_var:

    for j in cat_var:

        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')
```

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

```
    season and weathersit-0.0211
    season and month-0
    holiday and weekday-8.56e-11
    hoilday and workingday-4.033e-11
    weekday and holiday-8.56e-11
    weekday and workingday-6.77e-136
    workingday and holiday-4.033e-11
    workingday and weekday-6.77e-11
    weathersit and season-0.0211
    weathersit and mnth-0.014
    mnth and season-0
    mnth and weathersit-0.014
```

So besides season,holiday and yr we will remove weekday,weathersit,workingday,mnth.

```
# Back up data

# aj = df

df = aj

df = df.drop(['mnth','weekday','weathersit','workingday'],axis = 1)
```

Feature Scaling

Let us check variance for each column in dataset before Normalisation
High variance will affect the accuracy of the model. so we want to normalise that variance.

```
df[num_var].var()
```

Normality Check by Plotting distplot and probplot

Distribution before Normaliation

```
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()
```

Note: If you want to use PCA don't normalise the data use scaler method instead which is in PCA section of code.

```
#Normalisation

for i in num_var:

   print(i)

   df[i] = (df[i] - min(df[i]))/(max(df[i]) - min(df[i]))

   Distribution before Normaliation

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()
```

Let us check variance for each column in dataset after Normalisation

```
df[num_var].var()
```

More Feature Selection Test
Analysis of Variance(Anova) Test

It is carried out to compare between each groups in a categorical variable.
ANOVA only lets us know the means for different groups are same or not. It doesn’t help us identify which mean is different.
Hypothesis testing :
- Null Hypothesis: mean of all categories in a variable are same.
- Alternate Hypothesis: mean of at least one category in a variable is different.

If p-value is less than 0.05 then we reject the null hypothesis.
And if p-value is greater than 0.05 then we accept the null hypothesis.

```
df.head()

cat_var=['season','holiday','yr']
```

```
df[cat_var]=df[cat_var].apply(lambda x: x.astype('category') )

num_var=['temp','atemp','hum','windspeed','cnt']

df[num_var]=df[num_var].apply(lambda x: x.astype('float') )

def anova_test(df,target):

    for i in cat_var:

        formula=('{} ~ {}').format(target, i)

        df.lm = ols(formula,data=df).fit()

        table = sm.stats.anova_lm(df.lm, typ=1)

        print('Anova table between',target,'and',i,'is\n',table)

# print('\n For target var = casual--')

# anova_test(df,'casual')

# print('\n For target var = registered--')

# anova_test(df,'registered')

print('\n For target var = cnt--')

anova_test(df,'cnt')
```

After looking at above table we see that p-value is greater than 0.05 for weekday and less for season,weathersit,yr.
Therefore, we will accept the NULL hypothesis and we can say that some means are nearly/closely same for holiday.
and accept alternate hypothsis for season,yr say that some means are not equal.

Multicollinearity Test

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.

```
df.dtypes

outcome, predictors = dmatrices('cnt ~ +season+ yr +weekday + weathersit + 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
```

From above Dataframe we see that there is Multicollinearity in our Data
temp and atemp has highest VIF value

Feature Engineering
Feature Engineering on numerical Features

we know that temp and atemp are both temperature values and are very highly correlated with each other.
we will aggregate those 2 columns and derive a new feature.

```
columns = ['temp','atemp']

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

df.head()
```

Now that we have derived a new Column 'mean_temp' we can drop 'temp' and 'atemp'
Also we will drop 'casual' and 'registered' columns from our dataframe, as their addition is included in 'cnt' column.

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

df.head()
```

Feature Engineering on Categorical Features

we will use one-hot encoding techniques on categorical variables - season,mnth,weekday,weathersit.

```
df.nunique()

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

#Creating dummies for categorical variables

for i in one_hot_var:

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

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

    df = df.join(temp)

df.columns
```

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

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

Splitting data into train and test

With the time series data we will break up our train and test into continuous chunks.
The training data should be the earliest data and test data should be the latest data.

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.
we can't use sklearn's train_test_split bcoz it randomly shuffles the train and test data.

Separating features from target variable.

```
target_cnt = df.iloc[:,4]

target_casual = df.iloc[:,2]

target_registered = df.iloc[:,3]

target_cnt.head(),target_casual.head(),target_registered.head()

feature = df.drop(['cnt','casual','registered'],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)

df[train_size:].tail()

pd.DataFrame(test_features).tail()
```

Dimensionality Reduction using PCA technique

Dimensionality Reduction finds patterns in the data and uses that pattern to reexpress it in a compressed form.
Removes less information noise features.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()



# Fit on training set only.

scaler.fit(train_features)



# Apply transform to both the training set and the test set.

train_features = scaler.transform(train_features)
```

test_features = scaler.transform(test_features)

Fit PCA on training set. Note: you are fitting PCA on the training set only

pca = PCA()

pca.fit(train_features)

#The amount of variance that each PC explains

var= pca.explained_variance_ratio_

#Cumulative Variance explains

var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)

print(var1)

plt.plot(var1)

# plt.savefig('cummulative var.png')

plt.show()

features = range(pca.n_components_)

plt.bar(features, pca.explained_variance_)

plt.xlabel('PCA feature')

plt.ylabel('variance')

plt.xticks(features)

# plt.savefig('bar plot before filtering pca components.png')

plt.show()

The intrinsic dimension is the number of PCA features with significant variance.
So from the above barplot: intrinsic dimension = 8.
Now we have reduced the dimension of our data.

pca = PCA(n_components=8)

pca.fit(train_features)

features = range(pca.n_components_)

plt.bar(features, pca.explained_variance_)

plt.xlabel('PCA feature')

plt.ylabel('variance')

plt.xticks(features)

```
# plt.savefig('bar plot after filtering pca components.png')

plt.show()
```

Apply the mapping (transform) to both the training set and the test set.

```
test_features

pd.DataFrame(test_features).shape
```

Model Development

Regression metrics for our Models:

  r square
  MAE(Mean Absolute Error)
  MSE(Mean square Error)
  RMSE(Root Mean Square Error)
  RMSLE(Root Mean Squared Log Error)

```
def rmsle(y,y_):

    log1 = np.nan_to_num(np.array([np.log(v + 1) for v in y]))

    log2 = np.nan_to_num(np.array([np.log(v + 1) for v in y_]))

    calc = (log1 - log2) ** 2

    return np.sqrt(np.mean(calc))

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('RMSLE:',rmsle(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)

    print('RMSLE:',rmsle(test_target_cnt,y_pred))

from sklearn.linear_model import LinearRegression,Ridge,Lasso

from sklearn.model_selection import TimeSeriesSplit

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=True)


# Fit the regressor to the data

reg.fit(train_features,train_target_cnt)


# Compute and print the coefficients

reg_coef = reg.coef_

print(reg_coef)


# Plot the coefficients

plt.plot(range(len(feature.columns)), reg_coef)

plt.xticks(range(len(feature.columns)), feature.columns.values, rotation=60)

plt.margins(0.02)

plt.savefig('linear coefficients')

plt.show()

test_scores(reg)

# from regressors import stats

# stats.summary(reg,train_features,train_target_cnt)

X = feature.values

splits = TimeSeriesSplit(n_splits=3)

plt.figure(figsize=(20,10))

index = 1

for train_index, test_index in splits.split(X):

    train = X[train_index]
    test = X[test_index]

    print('Observations: %d' % (len(train) + len(test)))

    print('Training Observations: %d' % (len(train)))

    print('Testing Observations: %d' % (len(test)))

    plt.subplot(310 + index)

    plt.xlim(0,731)

    plt.plot(pd.DataFrame(train).iloc[:,2])

    plt.plot(pd.DataFrame(test).iloc[:,2])

    index += 1

plt.show()
```

Ridge Regression

```
# Setup the parameters and distributions to sample from: param_dist

param_dist = {'alpha':np.logspace(-4, 0, 50),

        'normalize':[True,False],

          'max_iter':range(500,5000,500)}

# Instantiate a Decision ridge classifier: ridge

ridge = Ridge()


# Instantiate the gridSearchCV object: ridge_cv

ridge_cv = GridSearchCV(ridge, param_dist, cv=5,scoring='r2')


# Fit it to the data

ridge_cv.fit(feature, target_cnt)


# Print the tuned parameters and score

print("Tuned Decision ridge Parameters: {}".format(ridge_cv.best_params_))

print("Best score is {}".format(ridge_cv.best_score_))

# Instantiate a ridge regressor: ridge
```

```
ridge = Ridge(alpha=0.10481131341546852, normalize=True,max_iter = 500)


# Fit the regressor to the data

ridge.fit(train_features,train_target_cnt)


# Compute and print the coefficients

ridge_coef = ridge.coef_

print(ridge_coef)


# Plot the coefficients

plt.plot(range(len(feature.columns)), ridge_coef)

plt.xticks(range(len(feature.columns)), feature.columns.values, rotation=60)

plt.margins(0.02)

# plt.savefig('ridge coefficients')

plt.show()

test_scores(ridge)
```

Lasso Regression

```
# Setup the parameters and distributions to sample from: param_dist

param_dist = {'alpha':np.logspace(-4, 0, 50),

         'normalize':[True,False],

           'max_iter':range(1000,5000,500)}

# Instantiate a lasso regressor: lasso

lasso = Lasso()


# Instantiate the gridSearchCV object: lasso_cv

lasso_cv = GridSearchCV(lasso, param_dist, cv=5,scoring='r2')
```

```
# Fit it to the data

lasso_cv.fit(feature, target_cnt)


# Print the tuned parameters and score

print("Tuned Decision lasso Parameters: {}".format(lasso_cv.best_params_))

print("Best score is {}".format(lasso_cv.best_score_))

# Instantiate a lasso regressor: lasso

lasso = Lasso(alpha=0.0016768329368110067,max_iter= 1000, normalize=False)


# Fit the regressor to the data

lasso.fit(train_features,train_target_cnt)


# Compute and print the coefficients

lasso_coef = lasso.coef_

print(lasso_coef)


# Plot the coefficients

plt.plot(range(len(feature.columns)), lasso_coef)

plt.xticks(range(len(feature.columns)), feature.columns.values, rotation=60)

plt.margins(0.02)

# plt.savefig('lasso coefficients')

plt.show()

test_scores(lasso)
```

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)


# Compute and print the coefficients

tree_features = tree.feature_importances_

print(tree_features)


# Sort feature importances in descending order

indices = np.argsort(tree_features)[::1]


# Rearrange feature names so they match the sorted feature importances

names = [feature.columns[i] for i in indices]
```

```
# Creating plot

fig = plt.figure(figsize=(20,20))

plt.title("Feature Importance")


# Add horizontal bars

plt.barh(range(pd.DataFrame(train_features).shape[1]),tree_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(train_features).shape[1]), names)

# plt.savefig('tree feature importance')

plt.show()

# Make predictions and cal error

test_scores(tree)
```

Random Forest Regression

```
# Create the random grid

random_grid = {'n_estimators': range(100,700,100),

            '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= 15, min_samples_split=2,n_estimators=100)


# Fit the regressor to the data

Forest.fit(train_features,train_target_cnt)


# Compute and print the coefficients

Forest_features = Forest.feature_importances_

print(Forest_features)


# Sort feature importances in descending order

indices = np.argsort(Forest_features)[::1]


# Rearrange feature names so they match the sorted feature importances

names = [feature.columns[i] for i in indices]


# Creating plot

fig = plt.figure(figsize=(20,20))

plt.title("Feature Importance")


# Add horizontal bars

plt.barh(range(pd.DataFrame(train_features).shape[1]),Forest_features[indices],align = 'center')

plt.yticks(range(pd.DataFrame(train_features).shape[1]), names)

# plt.savefig('Random forest feature importance')

plt.show()# Make predictions

test_scores(Forest)
```

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

Lets compare the distribution of train and test results. More or less the distribution of train and test looks identical. It confirms visually that our model has not predicted really bad and not suffering from major overfitting problem.

Predicted Bike Rental Count By Random Forest Regression Model Using PCA.

```
test_predicted = Forest.predict(test_features)

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

```
            cnt
dteday
2012-08-07  7301.233364
2012-08-08  7302.736364
2012-08-09  6631.646463
2012-08-10  6170.619286
2012-08-11  6468.002879
```

Predicted Bike Rental Count By Random Forest Regression Model without Using PCA.

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

```
            cnt
dteday
2012-08-07  7164.005139
2012-08-08  7216.232500
2012-08-09  6661.975841
2012-08-10  5995.545167
2012-08-11  6379.173818
```

Original Bike Rental Count.

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

```
            cnt
dteday
2012-08-07  7273.0
2012-08-08  7534.0
2012-08-09  7286.0
2012-08-10  5786.0
2012-08-11  6299.0
```

Line plot of original Vs Predicted Bike Rental count with PCA

```
plt.figure(figsize=(20,10))

plt.subplot(121)

plt.plot(test_target_cnt)

plt.subplot(122)

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

```
# plt.savefig('Line plot of original Vs Predicted Bike Rental count')

plt.show()
```

Line plot of original Vs Predicted Bike Rental count without PCA.

```
plt.figure(figsize=(20,10))

plt.subplot(121)

plt.plot(test_target_cnt)

plt.subplot(122)

plt.plot(pd.DataFrame(test_predicted,index = test_target_cnt.index,columns=['cnt']))

# plt.savefig('Line plot of original Vs Predicted Bike Rental count without PCA.')

plt.show()
```