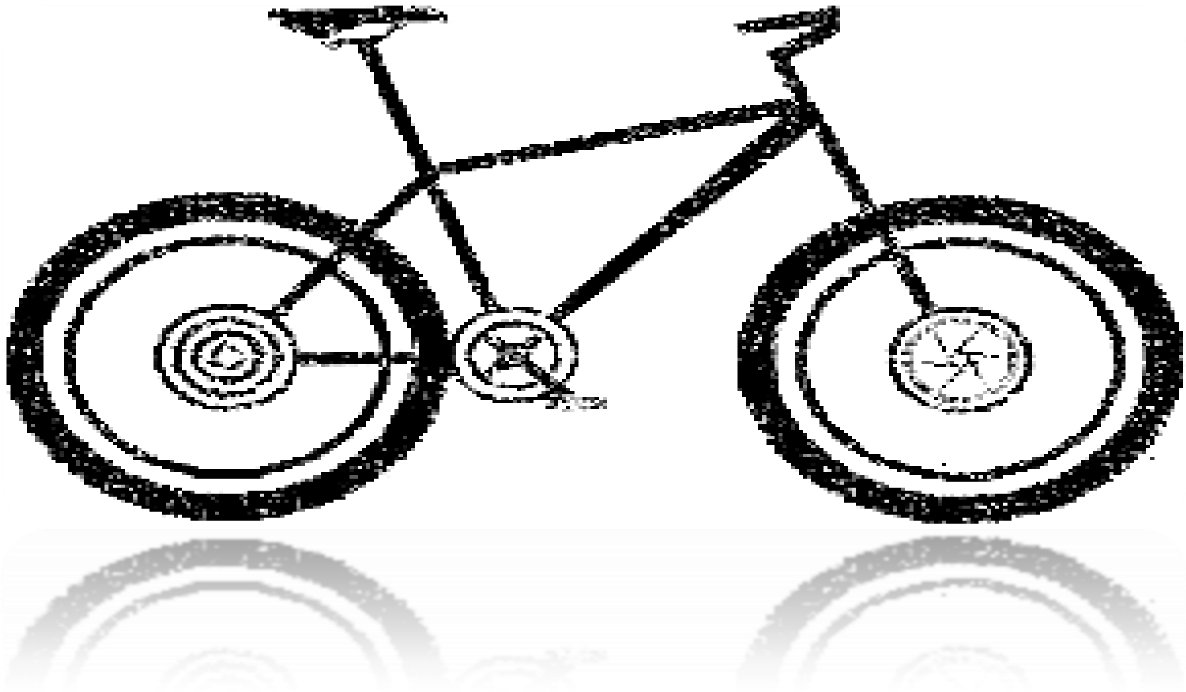
BIKE RENTING
DAILY COUNT PREDICTION



Prediction and Analysis Done in R and Python

3rd October 2018 **Parvesh Dhawan**

Contents

Chapter 1

Introduction

Biking is an excellent way to stay in shape while exploring local areas and communing with nature. With many biking enthusiasts eager to find new paths to explore in and around their local area . Our case study is regarding an organization who lends rental bike. According to business sense the demand of bikes for a particular day depends upon several factors like weather situation, season, holiday etc. It is important to know the demand of a particular day beforehand, so that they can meet the demand smoothly.

1.1 Problem Statement

The objective of this case 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:-

Variables	:	Description
Instant	:	Record index
Dteday	:	Date (Ranging from 1st Jan 2011 to 31st Dec 2012)
season	:	Season (1 : Spring , 2 : summer, 3 : fall, 4 : winter)
yr	:	Year (0 : 2011, 1: 2012)
mnth	:	Month (1 to 12)
Hoiliday	:	Weather day is holiday or not (Extracted from holiday schedule) (0 : Not Holiday, 1: Holiday)
Weekday	:	Day of week
Workingday	:	If day is neither weekend or holiday : 1 otherwise : 0
Weathersit	:	Situation of weather (extracted from Freemeteo) 1: Clear,Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken Clouds, Mist + Few Clouds, Mist 3: Light Snow, Light Rain + Thunderstrom + Scattered Clouds 4 : Heavy Rain + ICE Pallets + Thunderstrom, 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
atemp	:	Normalized feeling temperature in Celsius. The values are derived via (t – t_min) / (t_max / t_min)) t_min = -16, t_max = +50
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 Basciall it is (casual + registered)

Size of Dataset Provided : - 731 observation , 16 variables.

Let's have a look on data

instant	dteday	Season	yr	mnth	holiday	weekday	workingday	weathersit
1	1/1/2011	1	0	1	0	6	0	2
2	1/2/2011	1	0	1	0	0	0	2
3	1/3/2011	1	0	1	0	1	1	1
4	1/4/2011	1	0	1	0	2	1	1
5	1/5/2011	1	0	1	0	3	1	1

Temp	Atemp	Hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600

Categorical Variables :- season, yr, mnth, holiday, weekday, workingday, weathersit.

Continuous Variables :- Instant, temp, hum, windspeed, casual, registered, cnt

According to problem statement we have to predict bike rental count on daily based on the environmental and seasonal settings.

We have three variables as the count

1.) **Casual** = Number of casual users count
2.) **Registered** = Number of Registered user count
3.) **CNT** = Total count (We we look closly into data we can easily see that **CNT = Casual + Registered**)

We will Predict our Result according to **CNT** as our Target Variable.

Chapter 2

Methodology

We have to predict the total count of bike rental which falls in the category of regression. As our output will be a continuous number. We have divided our methodology in to these parts :-

- **Exploratory Data Analysis**

(Exploring data, Distribution of data, Visualization, Univarate, bivariate, Multivariate analysis)

- **Preprocessing Data**

(Outliers in data, Dependencies among variables (Correlation // Anova // Chi-square // Multicollinearity), Sampling, dummies for categorical data in case of Statistical models)

- **Basic Modeling & K-Fold Validation**

(Linear Regression, Decision Tree, Random Forest, SVR)

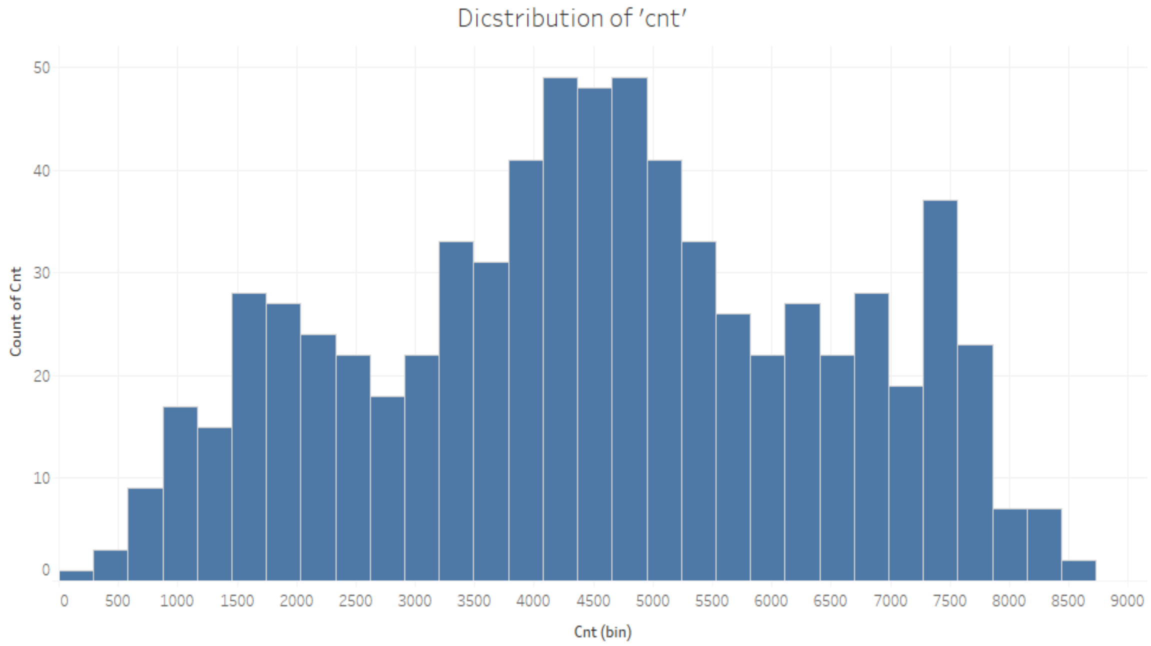
- **Evaluation & Optimization of Final Model**

(Evaluating performances and tuning parameters for final model)

2.1 Exploratory Data Analysis

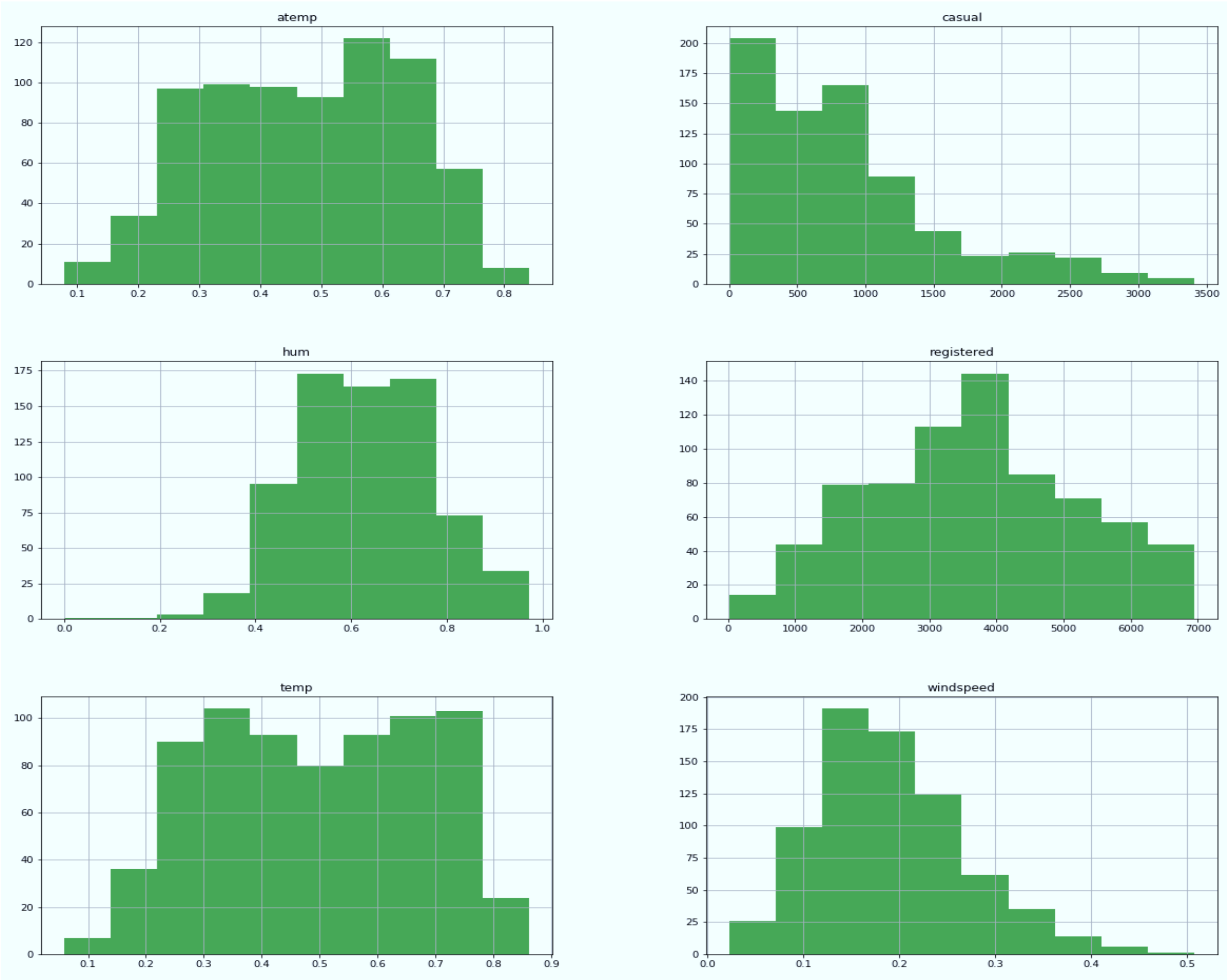
Exploratory Data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.

2.1.1 Univariate // Bivariate // Multivariate Analysis of Data



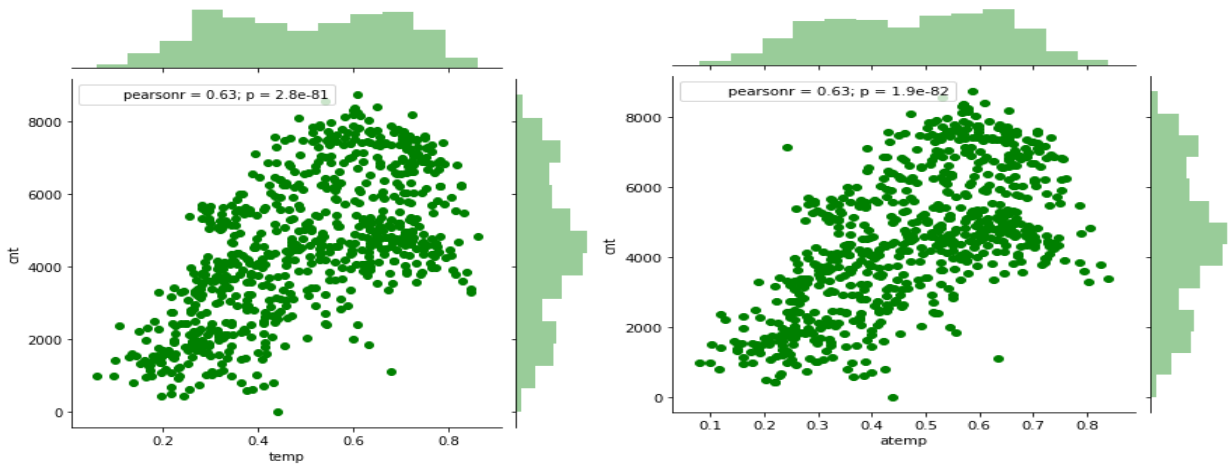
Our Target variable CNT is normally distributed.

□ **Distribution of different continuous variables**

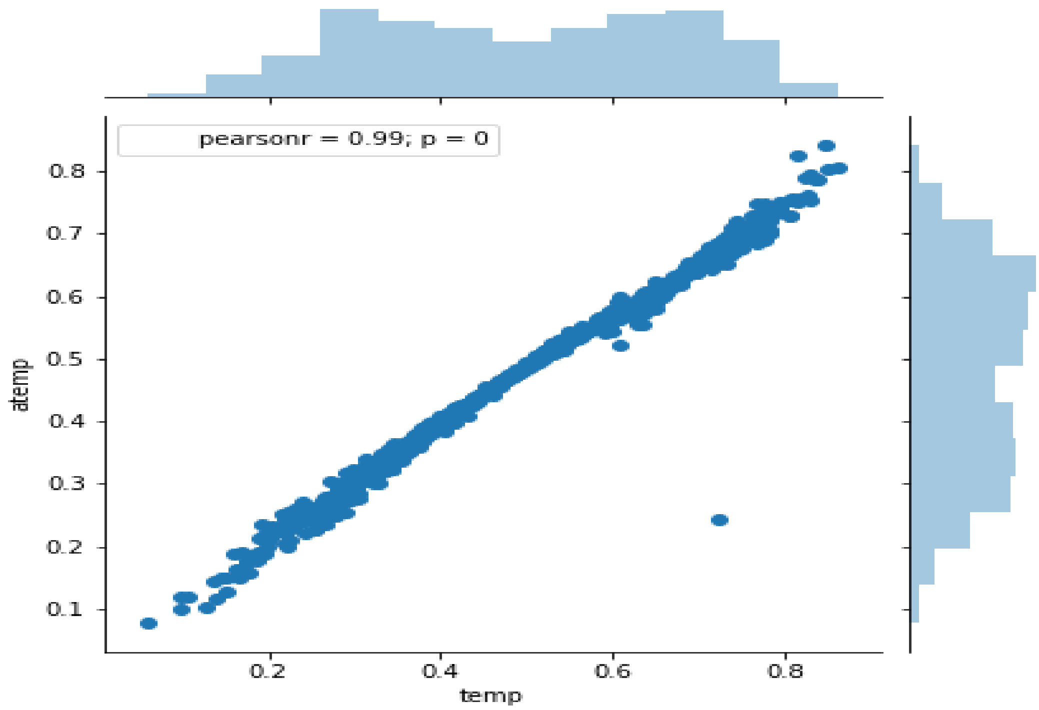


We know CNT = Casual + Registered

Here in the above visualization we can see that in way of describing our CNT variable registered cnt is normally distributed and Casual count is skewed. Mainly casual count remains in b/w 0 to 1000 counts and then they constantly decreasing.

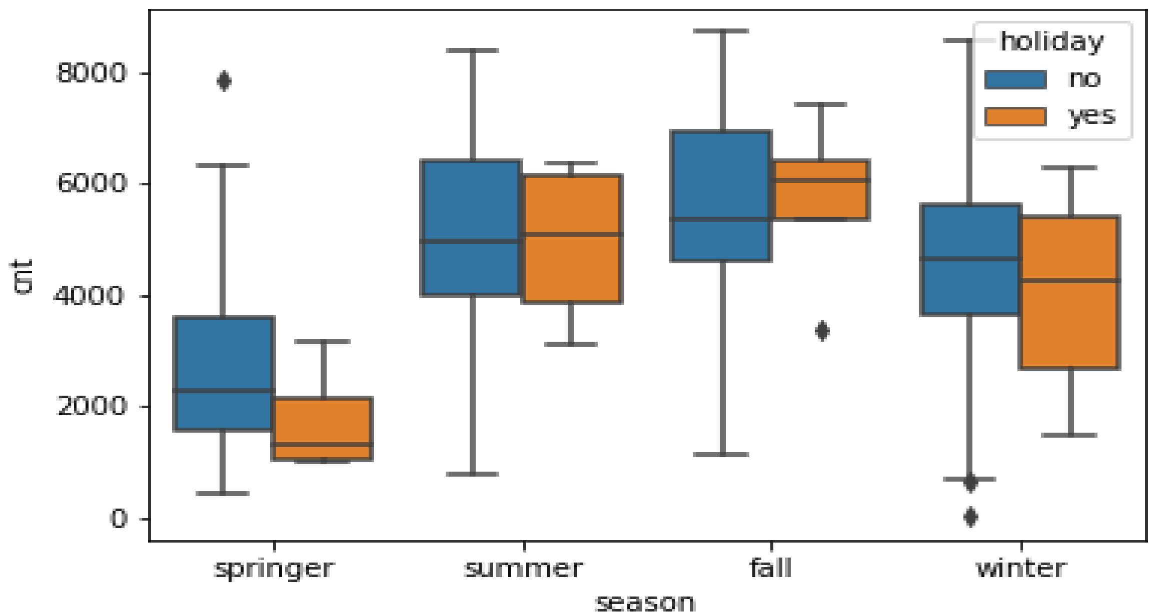


- **atemp and temp both the variables looking a bit similar.**



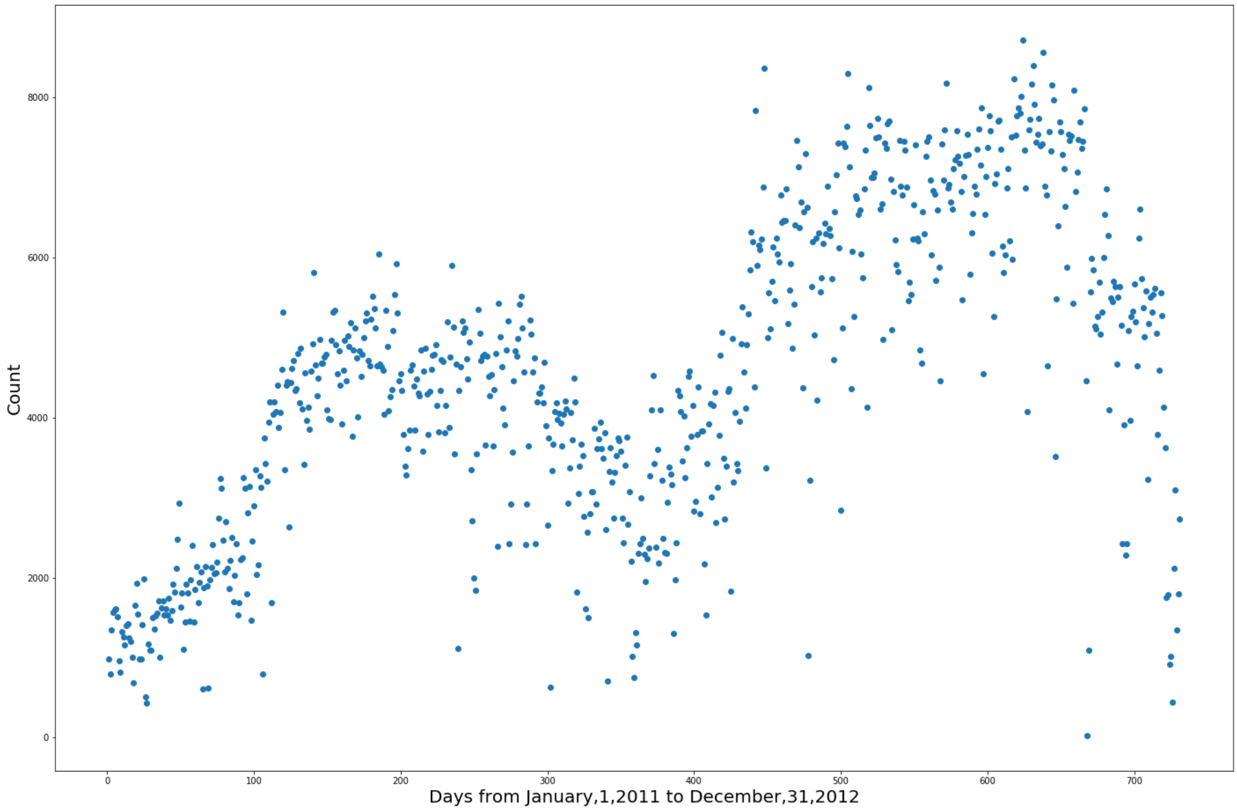
Above plot clearly shows that atemp and temp are highly correlated

- **Distribution of cnt variable in different seasons with respect to holidays**



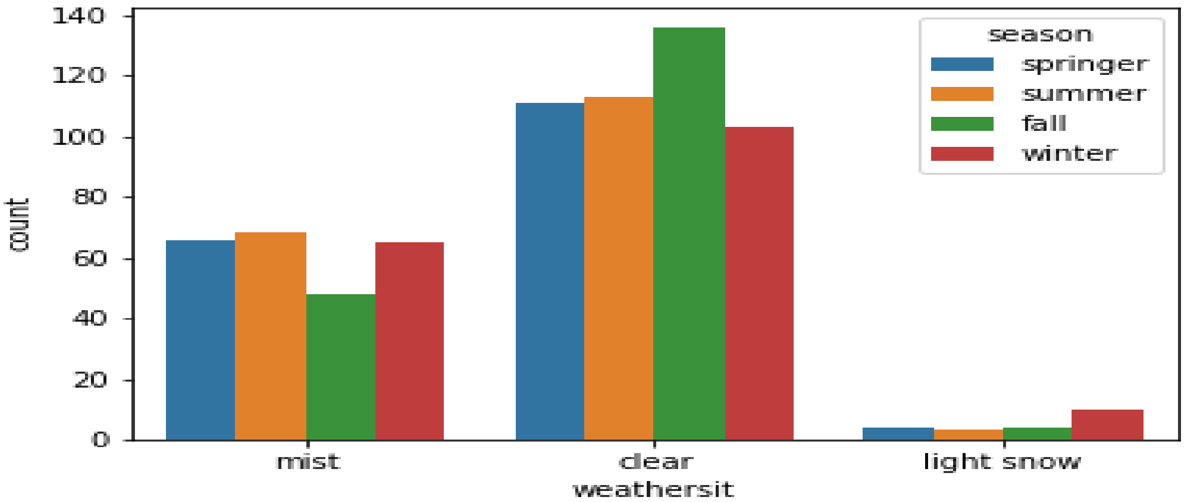
Number of CNT is increasing in springer season where day is not holiday while in season it’s decreasing

□Date wise CNT



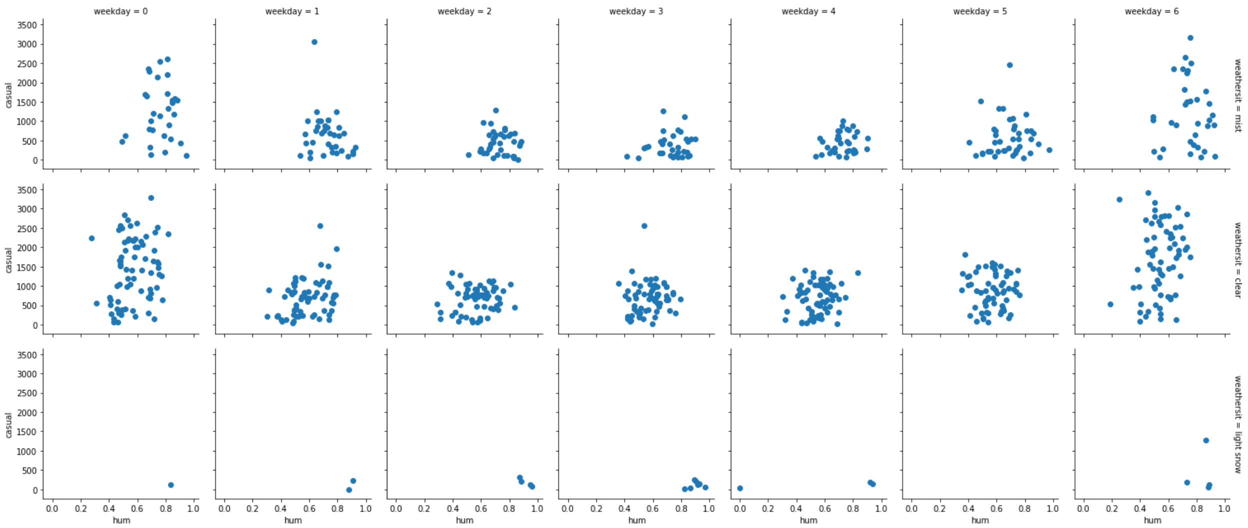
It's clearly shown that counts goes down in Starting and Ending phase of year. Also in comparison of second yr (2012) there is a massive increase in growth of CNT with respect to previous yr.

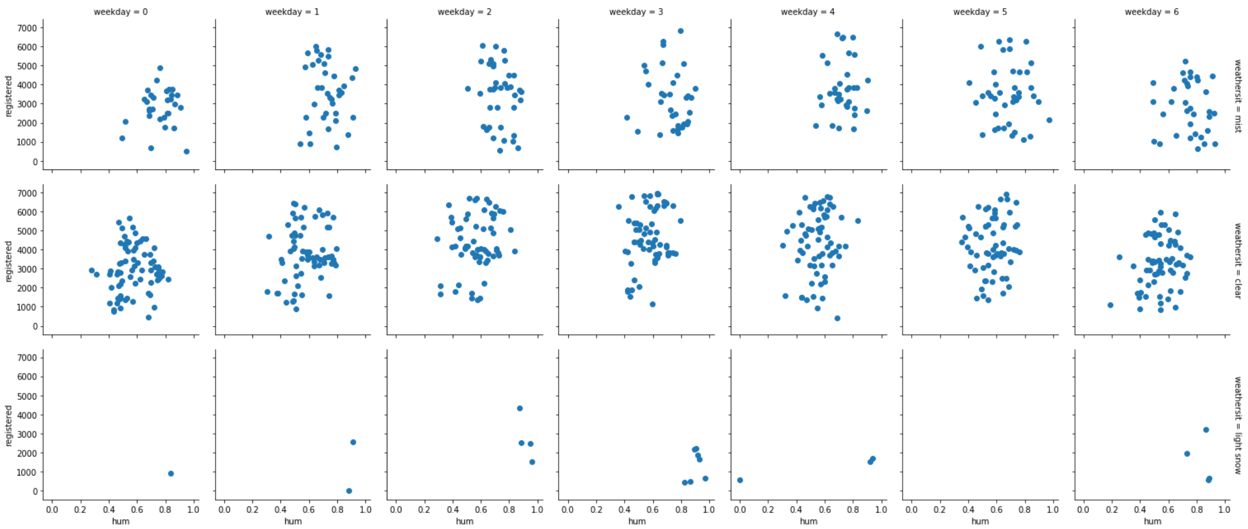
□CNT of particular season according to weather situation



More number of users prefer to ride bike in Clear weather rather than in snow or mist.

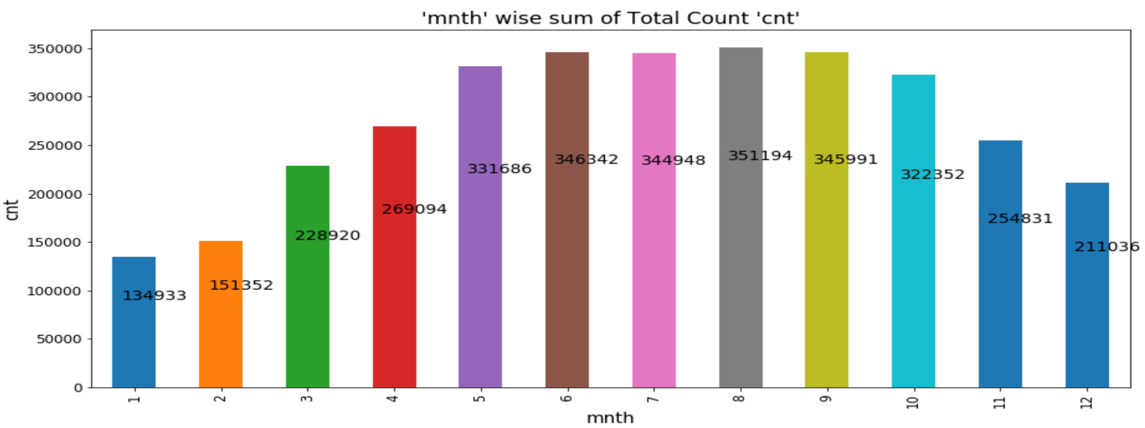
Casual & Register users count in specific day in different weather situation & humidity



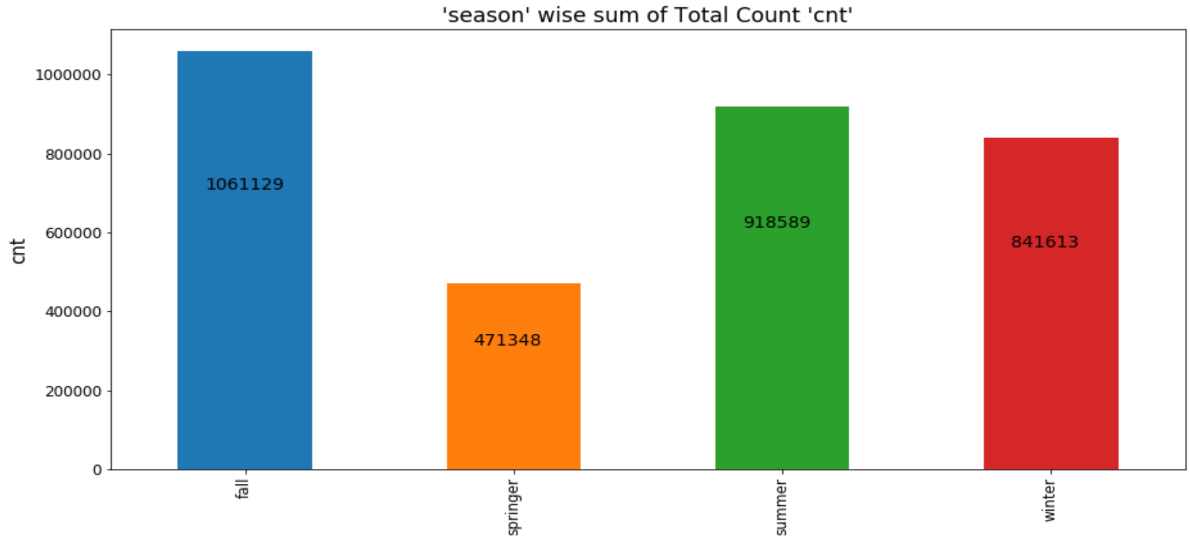


More number of casual users prefer to use bike in weekends (0,6), But there is no such case in registered users

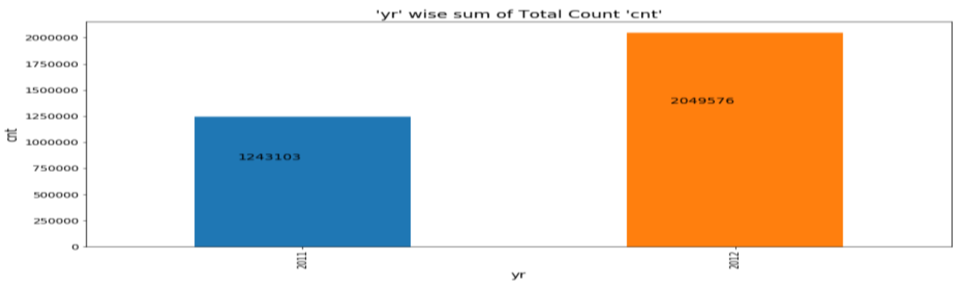
MNTH wise CNT



Maximum CNT was in Month of August (8), and over all JAN and FEB months has a little low cnt with respect to other months.

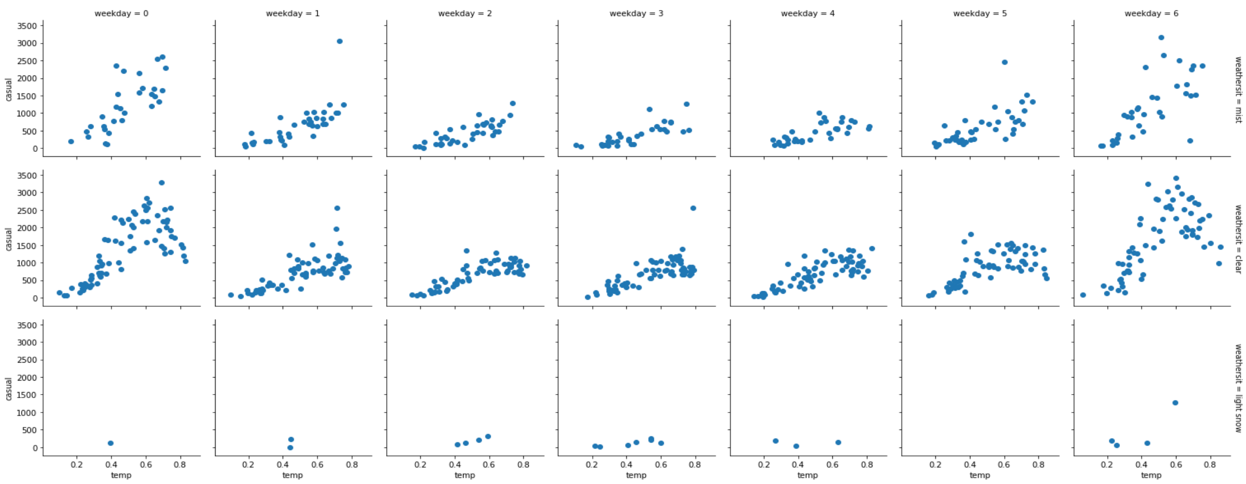


Very few people opt for Bikes in Springer season, Max users was in Fall season

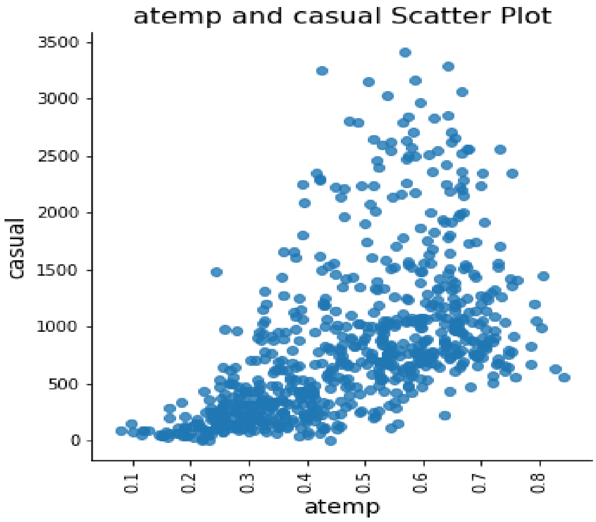


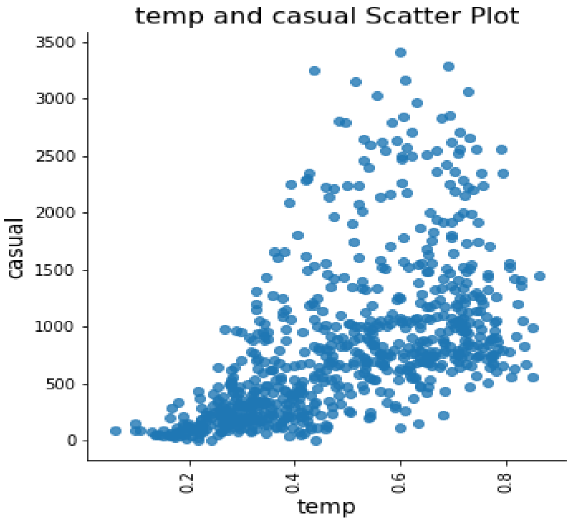
There is a really good growth in business from 2011 to 2012 END.

□ Casual Users in Particular day wrt to Temp and Weather Situation



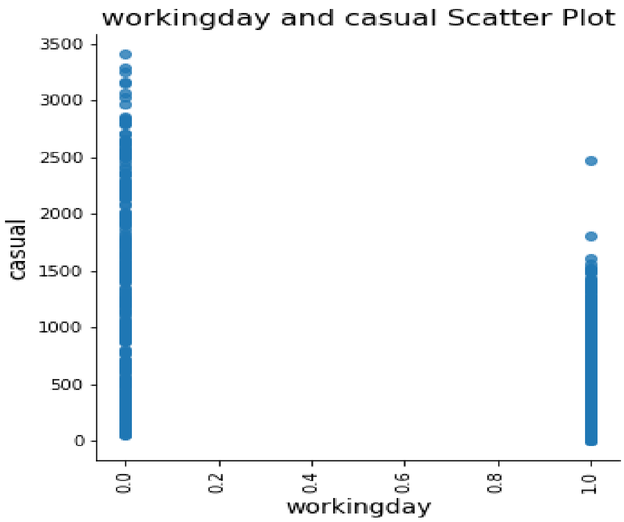
More Number of Casual Users Prefer weekend for bike rides, also as temp increases the growth of casual user increases when weather situation is clear and mist.

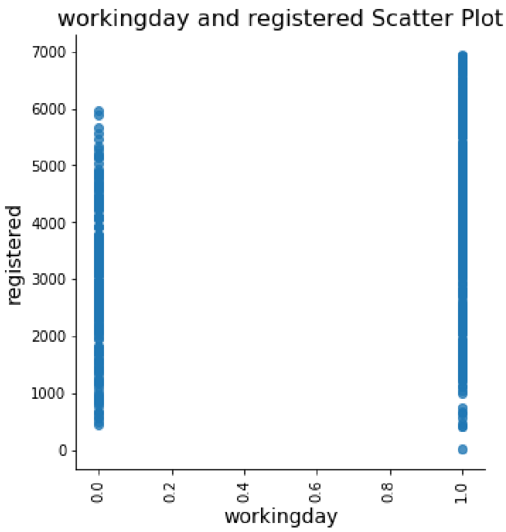




Also Casual Counts increasing even if atemp is increasing Casual count increasing.

Register user less prefer to go on bike rides in working days





2.1.2 Missing Values Check in Data

instant	dteday	Season	yr	mnth	holiday	weekday	workingday	weathersit
0	0	0	0	0	0	0	0	0

temp	Atemp	Hum	Windspeed	casual	registered	cnt
0	0	0	0	0	0	0

There is no missing values in the data.

2.2 Preprocessing Data

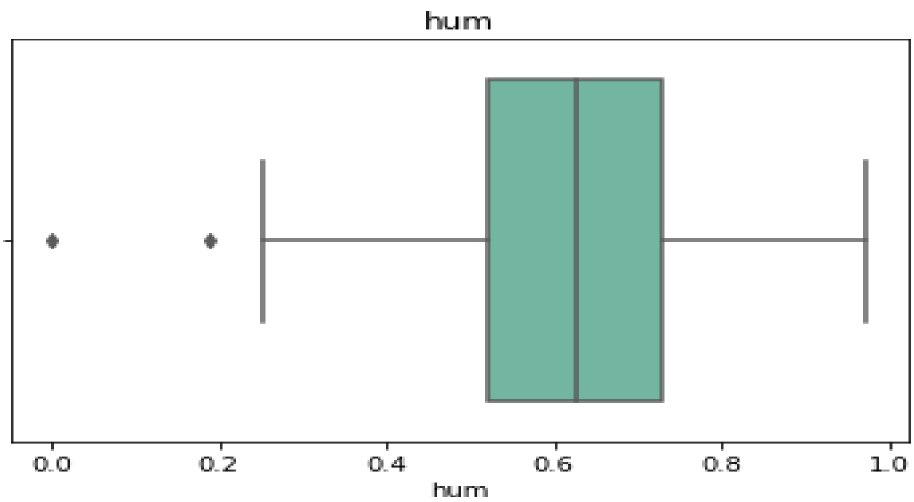
Preprocessing of the data is really important and it helps model to predict more accurately and learn accurately. Because if we are feeding raw data to machine learning models then the prediction and training won't work well.

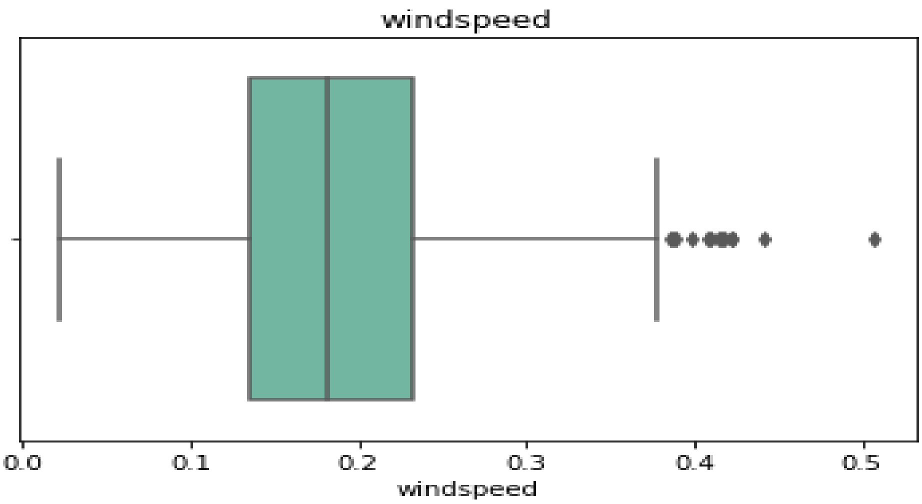
- Removing "instant" variable
- Carrying out day as (1 to 30 or 31) from dteday column and storing into same dteday column.

2.2.1 Outlier Analysis

We have used boxplot method for detecting outliers.

- Outliers majorly removed from Windspeed and Humidity column





We have replaced these values to NA and imputed with mean.

2.2.2 Feature Selection

2.2.2.1 Correlation Analysis

We have plotted all the numeric variable on plot with their correlation matrix. Correlation tells us that how strongly a pair of continuous variable are linearly related. (ranges from -1 to 1)

Variable “temp” and “atemp” are highly correlated.

2.2.2.2 Chi-sq Test for independence

Chi-square test compares two variables in a contingency table to see if they are related. It tests to see whether distributions of categorical variables differ from each another.

H0 (Null hypothesis) :- Variables are independent
H1 (Alternate hypothesis) :- Variables are not independent

We get a p-value and if p-value is less than 0.05 we will reject the null hypothesis by saying that alternate hypothesis are true, which says that two variables are not independent.

So we perform it over our all the categorical columns

	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
dteday	0.000000	1.000000	1.000000	1.000000	9.840008e-01	1.000000e+00	9.999990e-01	0.563115
season	1.000000	0.000000	0.999929	0.000000	6.831687e-01	1.000000e+00	8.865568e-01	0.021179
yr	1.000000	0.999929	0.000000	1.000000	9.949247e-01	9.999996e-01	9.799434e-01	0.127379
mnth	1.000000	0.000000	1.000000	0.000000	5.593083e-01	1.000000e+00	9.933495e-01	0.014637
holiday	0.984001	0.683169	0.994925	0.559308	0.000000e+00	8.567055e-11	4.033371e-11	0.600857
weekday	1.000000	1.000000	1.000000	1.000000	8.567055e-11	0.000000e+00	6.775031e-136	0.278459
workingday	0.999999	0.886557	0.979943	0.993350	4.033371e-11	6.775031e-136	0.000000e+00	0.253764
weathersit	0.563115	0.021179	0.127379	0.014637	6.008572e-01	2.784593e-01	2.537640e-01	0.000000

2.2.2.3 Anova Test (Analyzation of Variance)

In ANOVA we measure the mean of particular categories wise group of another column and compare all of the categories wise means.

H0 (Null hypothesis) :- Means are Same
H1 (Alternate hypothesis) :- Means are not Same

We get a p-value and if p-value is less than 0.05 we will reject the null hypothesis by saying that alternate hypothesis are true, which says that two variables (categories) means are not same and variables are a lot valuable.

So we perform it over our all the categorical columns

```
Anova p- value b/w cnt and dteday ----->   0.9999983403646867
Anova p- value b/w cnt and season ----->   6.720391362913557e-67
Anova p- value b/w cnt and yr ----->   2.483539904452293e-63
Anova p- value b/w cnt and mnth ----->   4.2510770151023976e-70
Anova p- value b/w cnt and holiday ----->   0.064759357926115
Anova p- value b/w cnt and weekday ----->   0.583494082505154
Anova p- value b/w cnt and workingday ----->   0.0984949616002635
Anova p- value b/w cnt and weathersit ----->   3.10631727005391e-17
```

2.2.2.4 Multicollinearity Check (Strictly check in Statistical models)

V.I.F.=1/(1−R2).

We checked multicollinearity among all the predictors with respect to our target variables.

After analyzation of all the techniques we will remove some variables from our data which are mostly redundant variables or highly dependent among predictors or also not explaning our target variables much.

In Python we have removed :- ‘holiday’,’workingday’,’atemp’
In R we have removed :- ‘holiday’,’workingday’,’atemp’,’dteday’

We are also using cnt as our target variable so removing casual & registered from our data too.

We don't need to scaled our data as already our data is normalized (values b/w 0 & 1)

2.2.3 Sampling

We are using random sampling as our target variable is continuous.

We are passing 80% of data for training
And 20% data for testing

Chapter 3

Modeling

We have a lot regression model to predict the total count of bike rental which falls in the category of regression. We have use these model on our data:-

- **Linear Regression**
- **Decision Tree**
- **Random Forest**
- **SVR (in R & Python Both)**

At first we build basic model with simplified approach and also test it on K-fold validation in python & R both.

3.1 Basic Modeling

3.1.1 Multiple Linear Regression (Python, R both)

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable x is associated with a value of the dependent variable y. In other terms, MLR examines how multiple independent variables are related to one dependent variable. Once each of the independent factors have been determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

The model for multiple linear regressions is: $y = B0 + B1x1 + B2x2 + ... + Bnxn + E$

Y = Target Variable

B0 = Intercept

B1 = regression coefficient that measures a unit change in the dependent variable when x1 changes – change in y

B2 = coefficient value that measures a unit change in the dependent variable when x2 changes – change in y

x1, x2, …, xn = Predictors

E = random error in prediction, that is variance that cannot be accurately predicted by the model. Also known as residuals.

Python Implementation & results :- We have implemented MLR using scikit learn library

Linear Regression

In [55]:
```
#from sklearn.linear_model import LinearRegression

linear_model = LinearRegression().fit(X_train,y_train)
test_scores(linear_model)

#cross_val(linear_model)
# # Mean Score of Cross validation =  0.77
# # Standard Deviation of CV =  0.05
```

```
<<<------------------- Training Data Score --------------------->

R2 score ==>   0.79
Mean absolute percentage error ==>  51.21 %
Root Mean Squared Error ==>  909.67

<<<------------------- Test Data Score --------------------->

R2 score ==>   0.83
Mean absolute percentage error ==>  15.91 %
Root Mean Squared Error ==>  749.36
```

R Implementation & Results :-

```
summary(lr_model)

Call:
lm(formula = cnt ~ ., data = dum_train_df)

Residuals:
    Min      1Q  Median      3Q     Max 
-3906.7  -373.0    95.0   460.1  2973.7 

Coefficients: (5 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3785.6127   420.8110   8.996  < 2e-16 ***
season.1    -1595.5295   214.1508  -7.450 3.57e-13 ***
season.2     -859.0061   247.0208  -3.477 0.000546 ***
season.3    -1094.2183   223.3764  -4.899 1.27e-06 ***
season.4            NA         NA      NA       NA    
yr.0        -2020.2957    67.0444 -30.134  < 2e-16 ***
yr.1                NA         NA      NA       NA    
mnth.1         17.4659   214.8438   0.081 0.935236    
mnth.2        151.9662   215.7609   0.704 0.481523    
mnth.3        656.9887   219.8251   2.989 0.002925 ** 
mnth.4        748.6509   279.3453   2.680 0.007580 ** 
mnth.5       1000.9090   294.8310   3.395 0.000736 ***
mnth.6        841.6672   303.6742   2.772 0.005764 ** 
mnth.7        513.1224   322.9773   1.589 0.112690    
mnth.8        943.9697   307.2705   3.072 0.002229 ** 
mnth.9       1368.8519   255.7440   5.352 1.27e-07 ***
mnth.10       529.9889   193.3434   2.741 0.006319 ** 
mnth.11       -86.6267   181.4724  -0.477 0.633297    
```

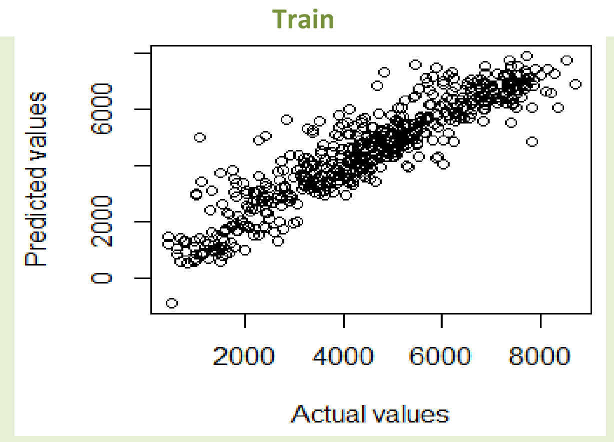
```
mnth.12                NA          NA       NA       NA
weekday.0       -329.9252    118.4391   -2.786 0.005525 **
weekday.1       -224.7965    119.5683   -1.880 0.060621 .
weekday.2       -148.1312    124.4084   -1.191 0.234285
weekday.3        -37.4670    121.8881   -0.307 0.758662
weekday.4         -0.7287    121.8525   -0.006 0.995231
weekday.5         23.7245    122.1958    0.194 0.846128
weekday.6              NA          NA       NA       NA
weathersit.1    1679.3626    230.9475    7.272 1.21e-12 ***
weathersit.2    1170.4357    212.5450    5.507 5.59e-08 ***
weathersit.3           NA          NA       NA       NA
temp            4074.8250    478.2404    8.520  < 2e-16 ***
hum            -1231.0254    345.0183   -3.568 0.000391 ***
windspeed      -2889.9802    500.4522   -5.775 1.28e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 783 on 557 degrees of freedom
Multiple R-squared:  0.8419,	Adjusted R-squared:  0.8345
F-statistic: 114.1 on 26 and 557 DF,  p-value: < 2.2e-16
```

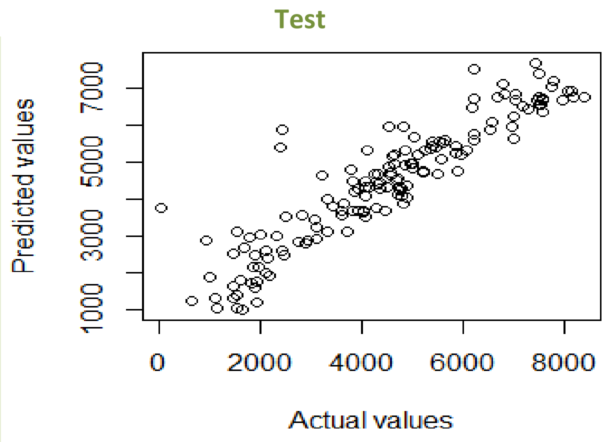
Train Score in R :-

RMSE	Rsquared	MAE	MAPE
764.7333338	0.8418996	567.3068215	18.10889

Test Score in R :-

RMSE	Rsquared	MAE	MAPE
813.5281365	0.8380681	570.5679825	132.6598



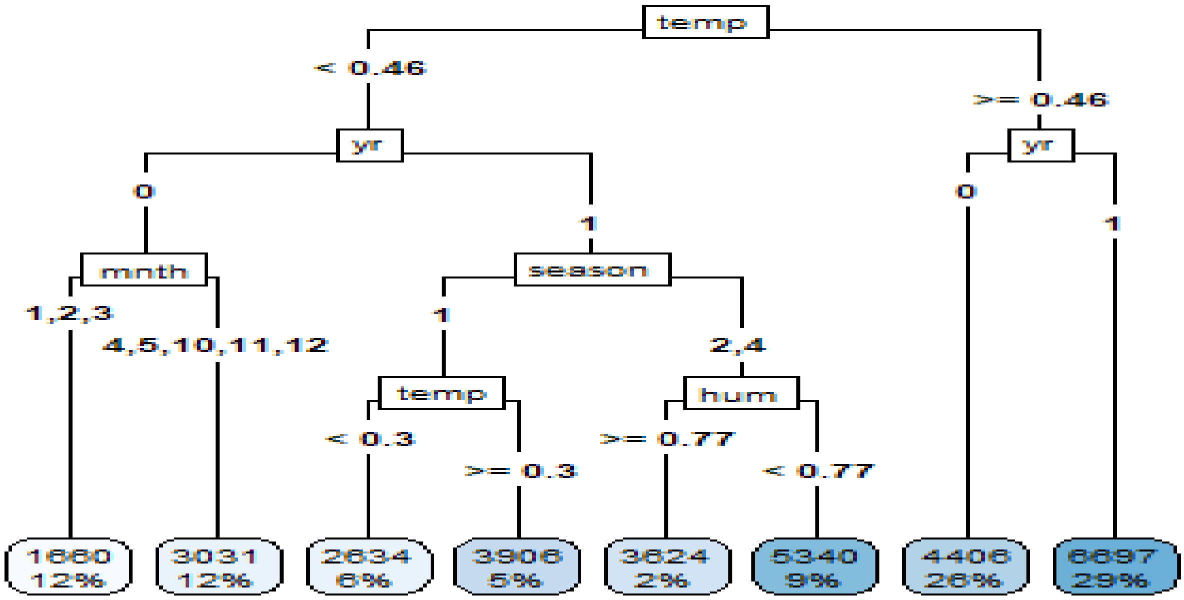


```
k-Fold Score--------------→
Linear Regression

584 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 526, 526, 526, 525, 525, 527, ...
Resampling results:

  RMSE      Rsquared   MAE
  787.4999  0.8330961  577.4846

Tuning parameter 'intercept' was held constant at a value of TRUE
[1] "Train Results____ "

       RMSE     Rsquared         MAE    MAPE
758.9749251   0.8436258     551.2978376  44.49676

[1] "Test Results____"
       RMSE     Rsquared         MAE      MAPE
819.1619491   0.8323536     607.1284566     18.1889
```

3.1.2 Decision Tree (Python, R both)

Decision tree belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms decision trees can also be used for solving regression and classification problem. The general motive of decision tree is to create a training model which can be used to predict class or value of target variables by learning decision rules inferred from training data.

Basically Decision tree is a rule based approach and it uses tree like structured. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label.



R Results :- Score on Train data Set

RMSE	Rsquared	MAE	MAPE
882.627663	0.788522	669.913980	50.53993

Score on Test data set

RMSE	Rsquared	MAE	MAPE
994.6383091	0.7534896	740.8181183	24.68454

R – KFOLD Results

```
CART


584 samples
  8 predictor


No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 526, 526, 526, 525, 525, 527, ...
Resampling results across tuning parameters:


  cp          RMSE      Rsquared   MAE
  0.0929130  1212.919  0.6115141   946.2181
```

```
  0.1954057  1380.578  0.4859650  1118.3734
  0.3967762  1753.185  0.3323691  1442.7208

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was cp = 0.092913.
```

R_K-fold On train & Test

```
Train Data Set
        RMSE      Rsquared          MAE          MAPE
1225.6830506     0.5921819      956.2100801       60.5995

[1] "Test Results____"
        RMSE      Rsquared          MAE            MAPE
1445.7222719     0.4797925     1111.5221958       41.44316
```

Python Implementation & Result

Decision Tree

```
In [57]: #from sklearn.tree import DecisionTreeRegressor

         tree_model = DecisionTreeRegressor(random_state=101).fit(X_train,y_train)
         test_scores(tree_model)

         #cross_val(tree_model)
         # # Mean Score of Cross validation =  0.73
         # # Standard Deviation of CV =  0.06
```

```
<<<------------------- Training Data Score --------------------->

R2 score ==>   1.0
Mean absolute percentage error ==>  0.0 %
Root Mean Squared Error ==>  0.0

<<<------------------- Test Data Score --------------------->

R2 score ==>   0.68
Mean absolute percentage error ==>  16.92 %
Root Mean Squared Error ==>  1031.36
```

3.1.3Random Forest (Python, R both)

A group of decisions Trees is random forest. The Random forest model is a type of additive model that makes prediction by combining decisions from a sequence of base models.

In case of regression while predicting the output we go for mean of all the favorable rule case values.

Python Implementation & Results :-

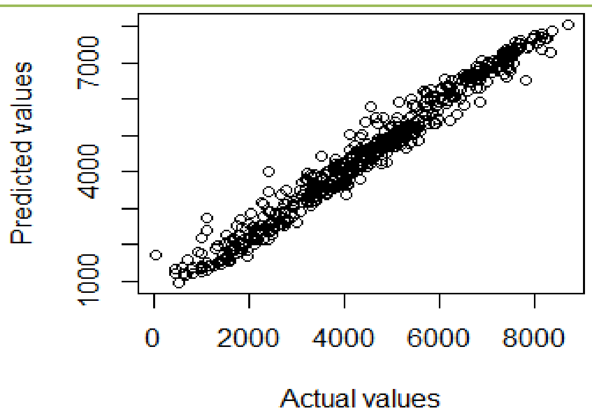
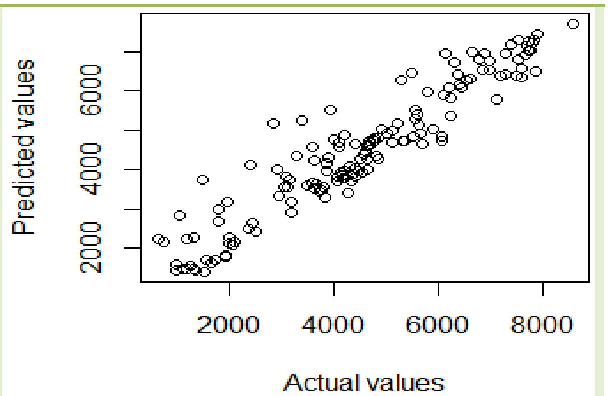
Random Forest

```
In [59]: #from sklearn.ensemble import RandomForestRegressor

         forest_model = RandomForestRegressor(n_estimators=500,random_state=101).fit(X_train,y_train)
         test_scores(forest_model)

         #cross_val(forest_model)
         # # Mean Score of Cross validation =  0.87
         # # Standard Deviation of CV =  0.03
```

```
<<<------------------- Training Data Score --------------------->

R2 score ==>   0.98
Mean absolute percentage error ==>  17.96 %
Root Mean Squared Error ==>  261.43

<<<------------------- Test Data Score --------------------->

R2 score ==>   0.92
Mean absolute percentage error ==>  11.98 %
Root Mean Squared Error ==>  511.86
```

R Implementation & Result :-

```
> #Train Result
        RMSE      Rsquared          MAE          MAPE
360.8331827    0.9715328     266.8893737     22.32626

> #Test Result
        RMSE      Rsquared          MAE          MAPE
696.6882159    0.8914143     515.0344634    18.40136
```

Train	Test
	

K-Fold _R

```
Random Forest

584 samples
  8 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 525, 525, 526, 524, 527, 526, ...
Resampling results across tuning parameters:

  mtry  RMSE       Rsquared   MAE
   2    1077.2514  0.8104534  874.4480
  14     713.1750  0.8639806  512.1298
  26     738.9963  0.8528719  527.0111

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 14.


Trainset Result

      RMSE    Rsquared        MAE        Mape
320.728461    0.974213     226.416473    22.12491


[1] "Test Results____"
       RMSE     Rsquared          MAE         Mape
691.9079736   0.8829989     490.4303747   16.82319
```

3.1.4Support Vector Regression (Python, R both)

Python Implementation & Execution :-

SVR

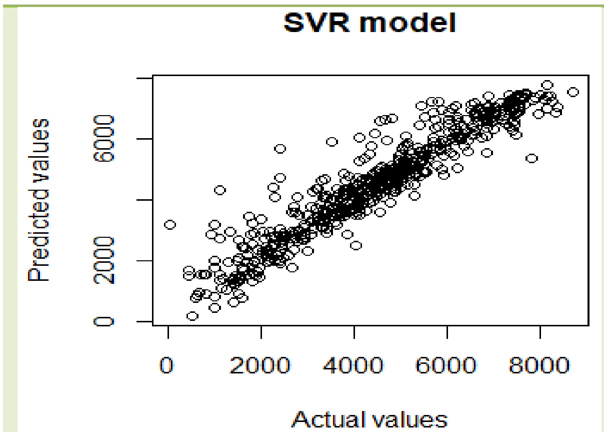
In [82]:
```python
#from sklearn.svm import SVR
svr_model = SVR(kernel='poly').fit(X_train,y_train)
test_scores(svr_model)

#cross_val(svr_model)
# Mean Score of Cross validation =  0.6
# Standard Deviation of CV =  0.05
```

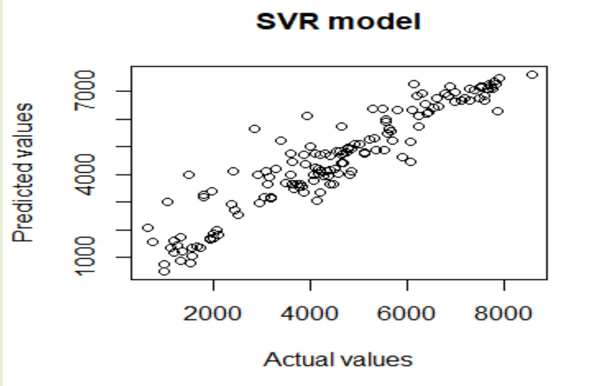
```
<<<------------------- Training Data Score --------------------->

R2 score ==>    0.63
Mean absolute percentage error ==>  60.32 %
Root Mean Squared Error ==>  1193.03

<<<------------------- Test Data Score --------------------->

R2 score ==>    0.57
Mean absolute percentage error ==>  33.13 %
Root Mean Squared Error ==>  1189.3
```

Implement & results in SVR in R :-

```
Train->

       RMSE     Rsquared          MAE        MAPE
630.8113042    0.8942029  431.6573680    39.08898

Test->
       RMSE     Rsquared          MAE         MAPE
723.3702920    0.8706781   518.0798590    18.16382
```

Train	Test





K-FOLD on SVR

```
Support Vector Machines with Polynomial Kernel


584 samples
  8 predictor


No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 526, 525, 526, 527, 525, 528, ...
Resampling results across tuning parameters:


  degree  scale  C     RMSE       Rsquared   MAE
  1       0.001  0.25  1681.7867  0.7032765  1372.5838
  1       0.001  0.50  1490.9693  0.7439150  1212.2818
  1       0.001  1.00  1218.7793  0.7754978   980.3416
  1       0.010  0.25   920.8427  0.8123237   708.2917
  1       0.010  0.50   837.4816  0.8237978   621.3859
  1       0.010  1.00   803.7380  0.8309600   585.1623
  1       0.100  0.25   792.0432  0.8335058   573.1250
  1       0.100  0.50   793.3774  0.8332051   575.0569
  1       0.100  1.00   797.5155  0.8316658   577.6770
  2       0.001  0.25  1490.1031  0.7440112  1211.6409
  2       0.001  0.50  1217.6915  0.7754459   979.2938
  2       0.001  1.00   970.5590  0.8045301   755.6620
  2       0.010  0.25   822.9899  0.8288525   611.5709
  2       0.010  0.50   784.1622  0.8385501   569.1975
  2       0.010  1.00   769.4221  0.8427435   556.0996
  2       0.100  0.25   730.6111  0.855828   519.2328
  2       0.100  0.50   748.8974  0.8539718   535.8722
  2       0.100  1.00   770.4169  0.8485512   555.3403
  3       0.001  0.25  1335.1974  0.7628408  1081.0658
```

```
  3       0.001  0.50  1060.2239  0.7933744   837.1814
  3       0.001  1.00   890.4314  0.8166279   678.3667
  3       0.010  0.25   789.2046  0.8375986   576.6057
  3       0.010  0.50   764.6491  0.8447100   553.0752
  3       0.010  1.00   745.4743  0.8514928   535.7259
  3       0.100  0.25   805.0688  0.8310277   581.1356
  3       0.100  0.50   843.8651  0.8168104   611.5403
  3       0.100  1.00   884.8757  0.8021562   648.8860

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were degree = 2, scale = 0.1 and
 C = 0.25.

Training Set------------------------→
       RMSE     Rsquared         MAE     MAPE
518.3628922   0.9279306 344.8343797   29.09403

 [1] "Test Results____"
       RMSE     Rsquared         MAE      MAPE
717.9788887   0.8745238 495.7036158      17.34821
```

Chapter 4

Model Evaluation

We have implemented four different model on bike rental prediction problem in both R and Python.

In regression we have different metrics for checking the performance or out models.

We have used r2 score, MAPE, RMSE.

According to these different evaluation metrics we evaluate our models in R and Python.

Let's Compare all the results:-

Base models result in Python :-

	Train Data			Test Data		
	R2	MAPE	RMSE	R2	MAPE	RMSE
MLR	0.79	51.21	909.67	0.83	15.91	749.36
Decision Tree	1	0	0	0.68	16.92	1031.36
Random Forest	0.98	17.96	261.43	0.92	11.98	511.86
SVR	0.63	60.32	1193.03	0.57	33.13	1189.3

We can see that Random Forest out perform every model in Python.

K-fold validation Results in Python :-

	MEAN	SD
MLR	0.77	0.05
DT	0.73	0.6
RF	0.87	0.03
SVR	0.6	0.05

Result Comparison In R

Base Models Result on Train & Test Data

	Train Data			
	R2	MAPE	RMSE	MAE
MLR	0.84	18.11	764.73	567.31
Decision Tree	0.79	50.54	882.63	669.91
Random Forest	0.97	22.32	360.83	266.89
SVR	0.89	39.09	630.81	431.66

Test Data				
	R2	MAPE	RMSE	MAE
MLR	0.84	132.66	813.53	570.57
Decision Tree	0.75	24.68	994.64	740.82
Random Forest	0.89	18.4	696.69	515.03
SVR	0.87	18.16	723.37	518.08

K-fold Cross Validation Results :-

	CV-10		
	RMSE	R2	MAE
MLR	787.5	0.83	577.48
Decision Tree	1212.92	0.61	946.22
Random Forest	713.17	0.86	512.13
SVR	730.61	0.85	519.23

Cross fold model applied on Data separately Train & Test

	CV_TRAIN				CV_Test			
	RMSE	R2	MAE	MAPE	RMSE	R2	MAE	MAPE
MLR	758.97	0.84	551.3	44.5	819.16	0.83	607.13	18.19
Decision Tree	1225.68	0.59	956.21	60.6	1445.72	0.48	1111.52	41.44
Random Forest	320.73	0.97	226.42	22.12	691.91	0.88	490.43	16.82
SVR	518.36	0.92	344.83	29.09	717.98	0.87	795.7	17.35

Finally in R & Python we are finalizing Random Forest as our Final Model.

FINAL MODEL ==== RANDOM FOREST

Final Model Result after hyper parameter optimization :-

<<<------------------- Training Data Score -------------------->

R2 score ==>   0.98

Mean absolute percentage error ==> 15.74 %

Root Mean Squared Error ==> 252.41

<<<------------------- Test Data Score -------------------->

R2 score ==>   0.93

Mean absolute percentage error ==> 11.75 %

Root Mean Squared Error ==> 489.5

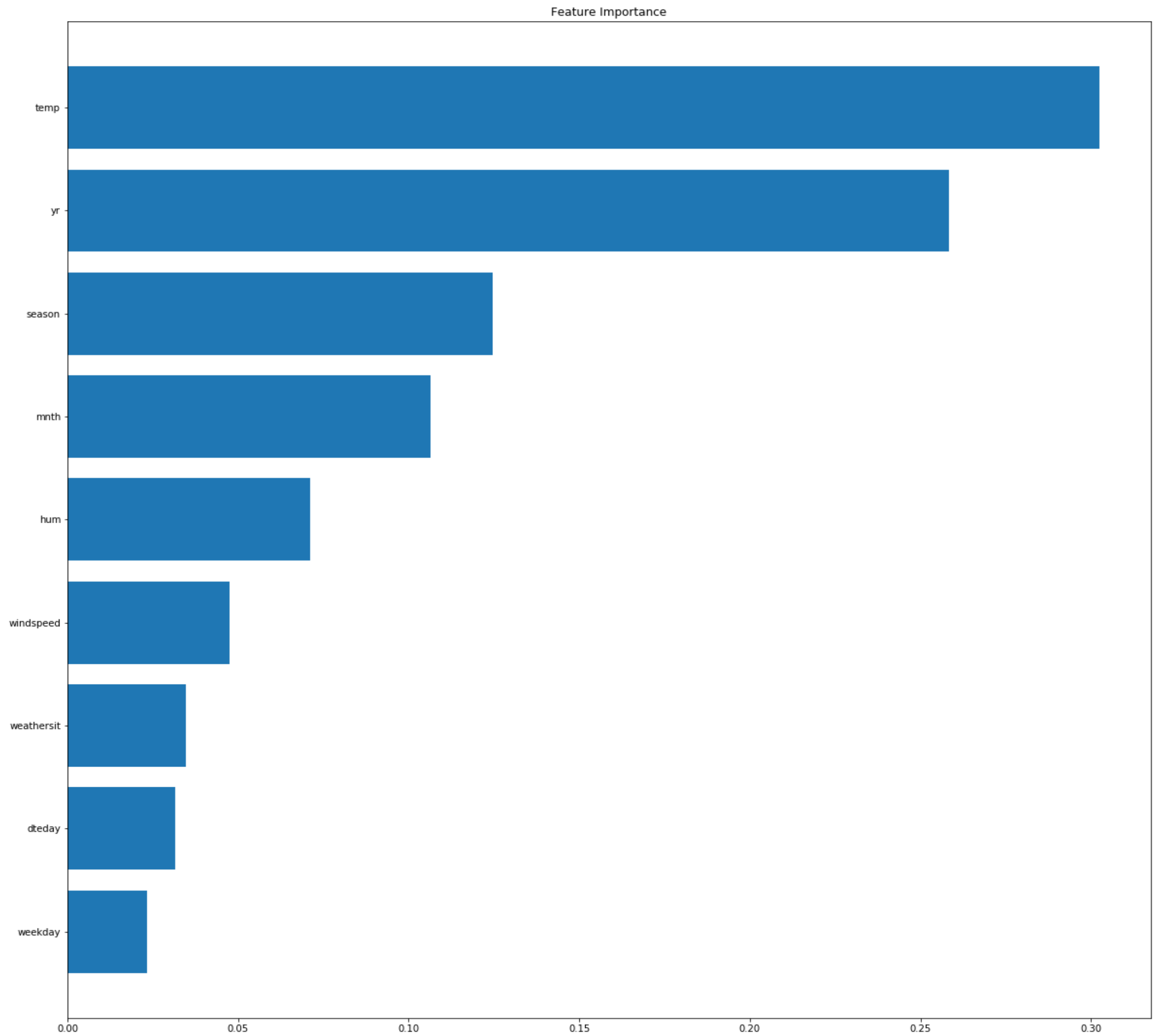
CV Mean = 0.88

Standard Dev = 0.03

Final Model with Optimized Parameters :- RANDOM FOREST

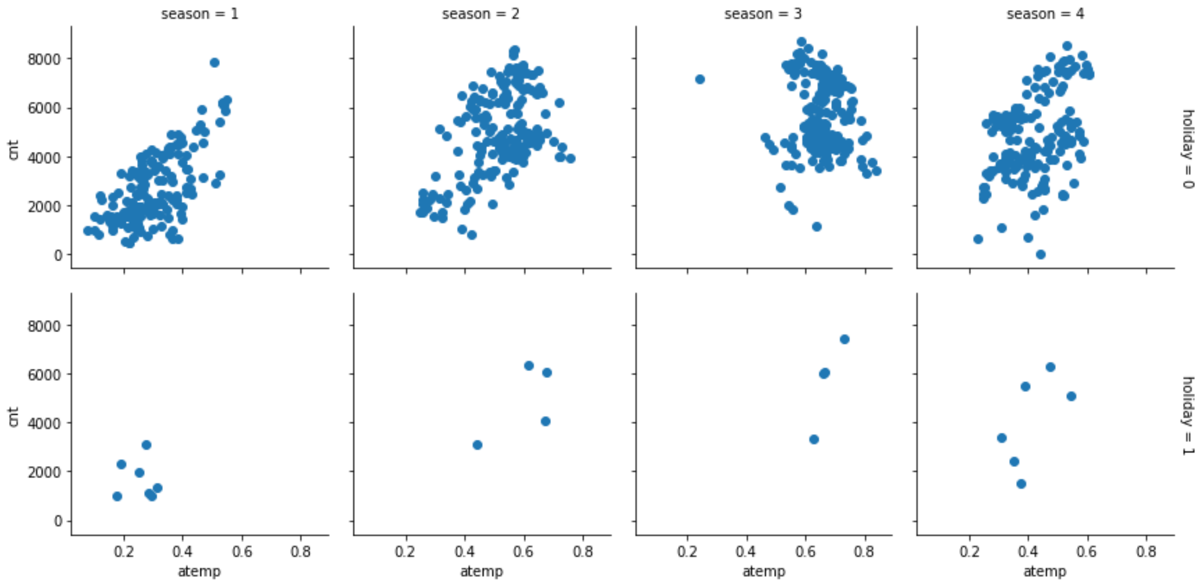
```
In [62]:
#from sklearn.ensemble import RandomForestRegressor

forest_model = RandomForestRegressor(max_depth= 15, max_features = 'sqrt',n_estimators = 500,random_state=101).fit(X_train,y_t
test_scores(forest_model)

#cross_val(forest_model)
# # Mean Score of Cross validation =  0.88
# # Standard Deviation of CV =  0.03

# #max_depth= 10, max_features = 'sqrt',n_estimators = 500
# #max_depth = 15, max_features = 'sqrt', min_samples_leaf = 2, n_estimators = 500
# #max_depth = 10, max_features = 'sqrt', n_estimators = 300
```

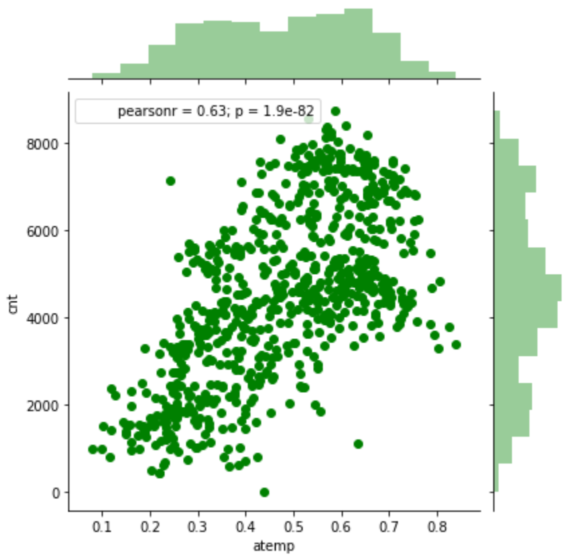
Features Importance to Our total count variable :-



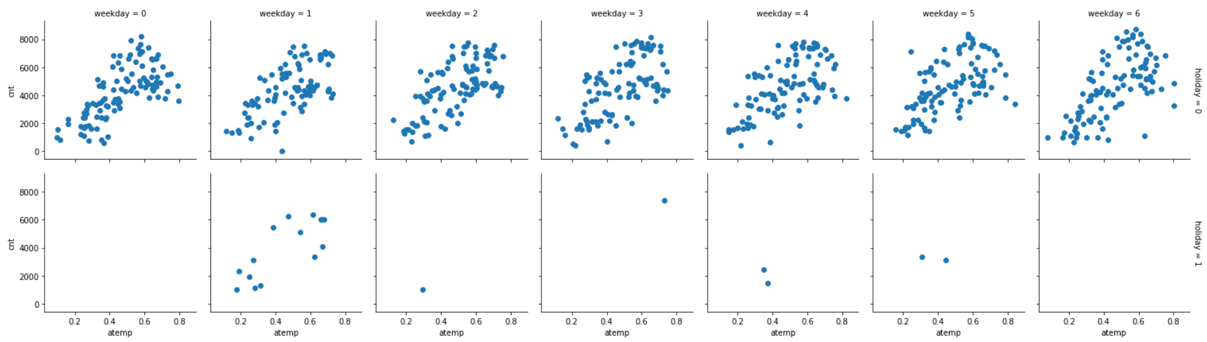
Appendix : I (Extra analysis Images)

Cnt with atemp according to season & holiday

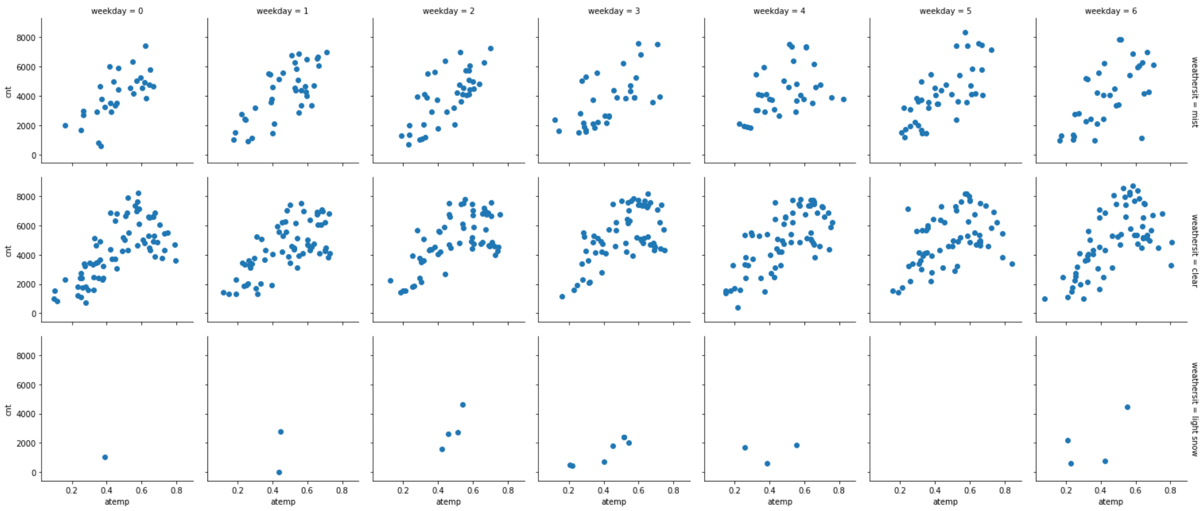


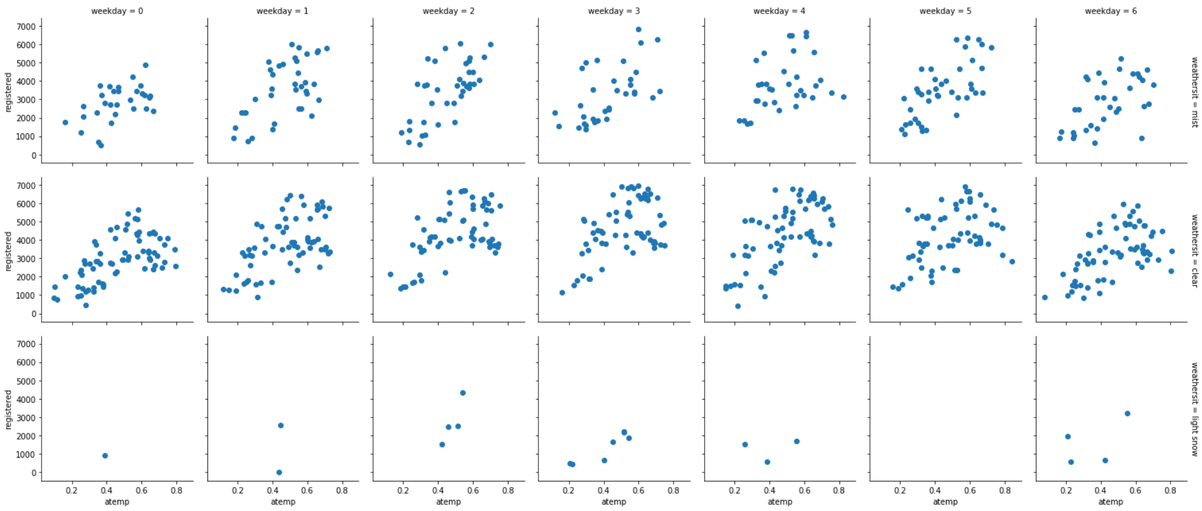


Weekday & holiday → cnt & atemp

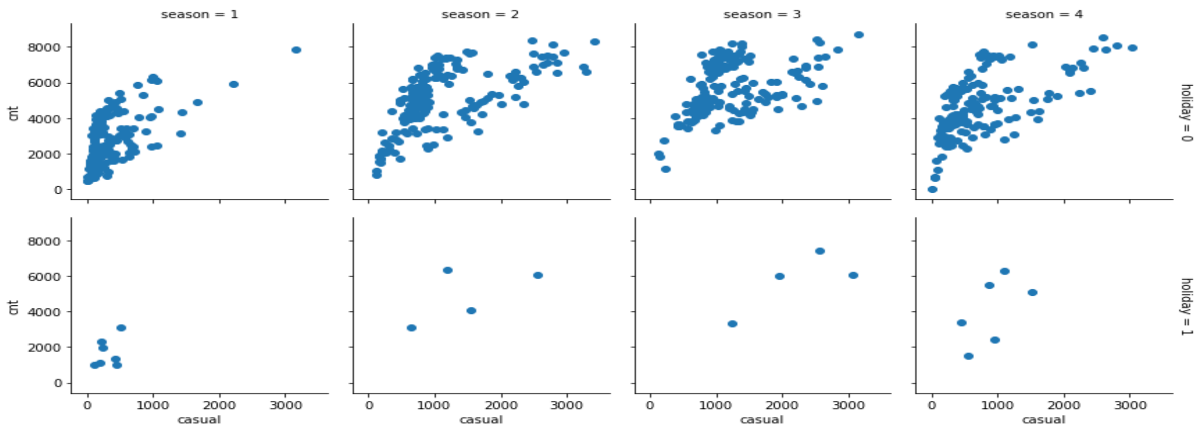


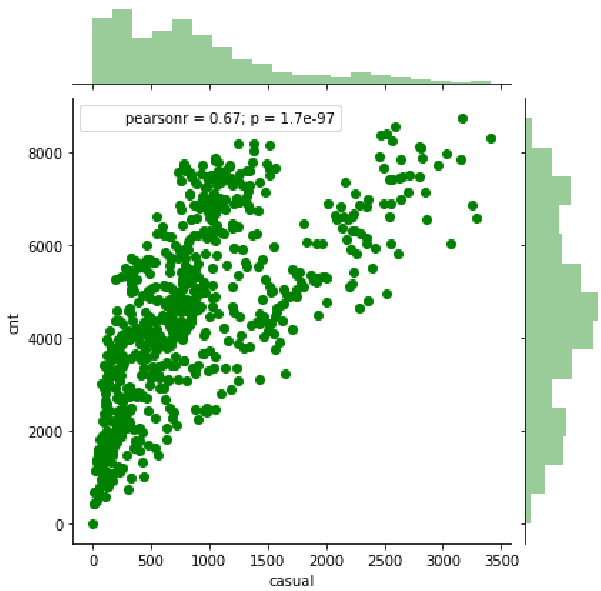
Weekday & weathersit → cnt atemp

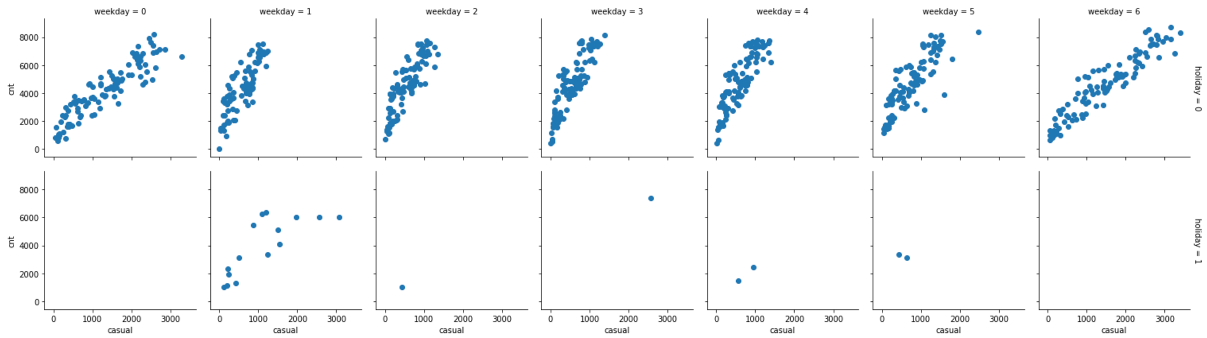


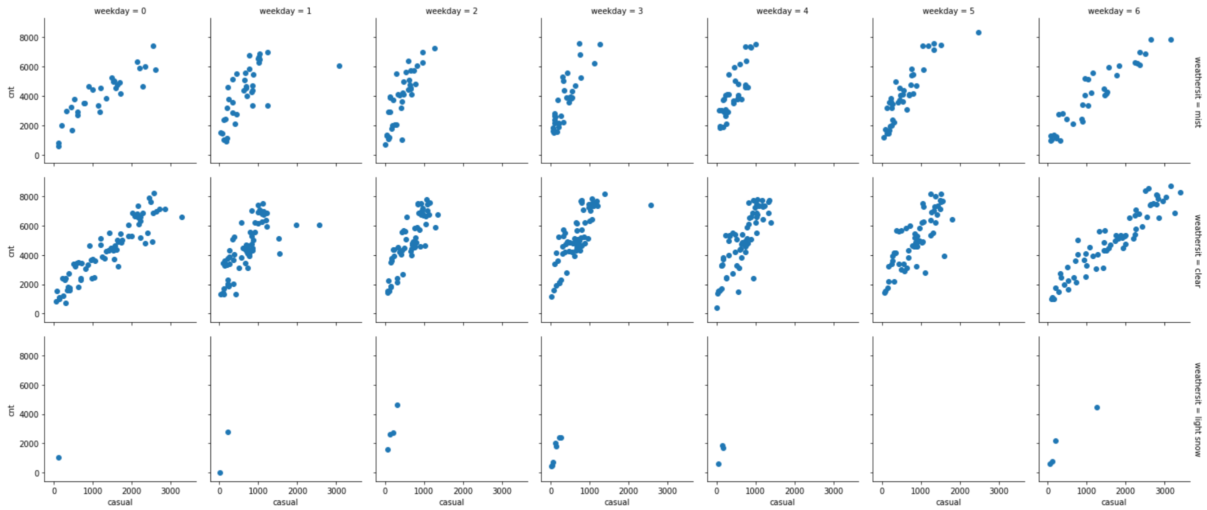


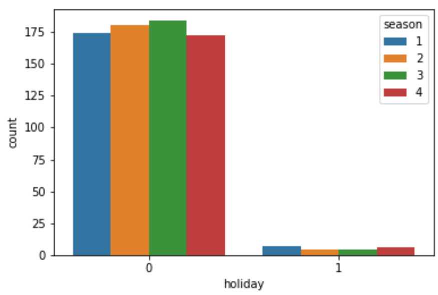
CNT and Casual → according to season & holiday

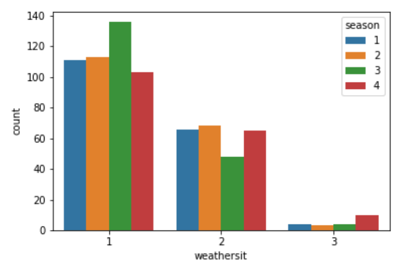


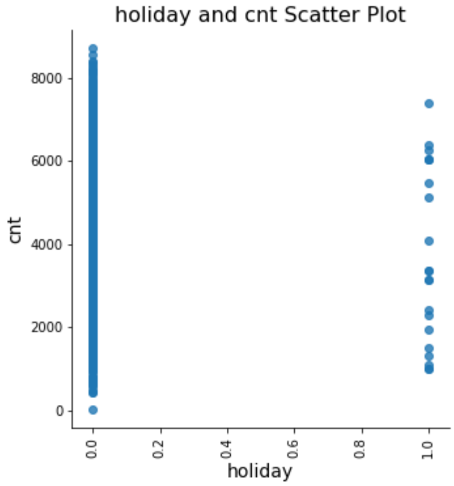

pearsonr = 0.67; p = 1.7e-97
cnt
casual

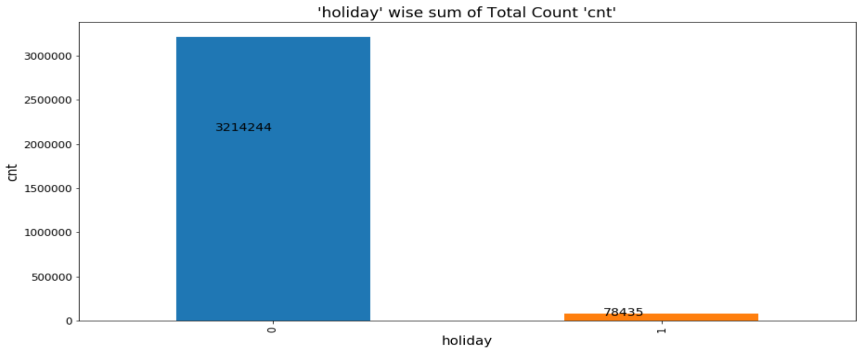

weekday = 0
weekday = 1
weekday = 2
weekday = 3
weekday = 4
weekday = 5
weekday = 6
holiday = 0
holiday = 1
cnt
casual

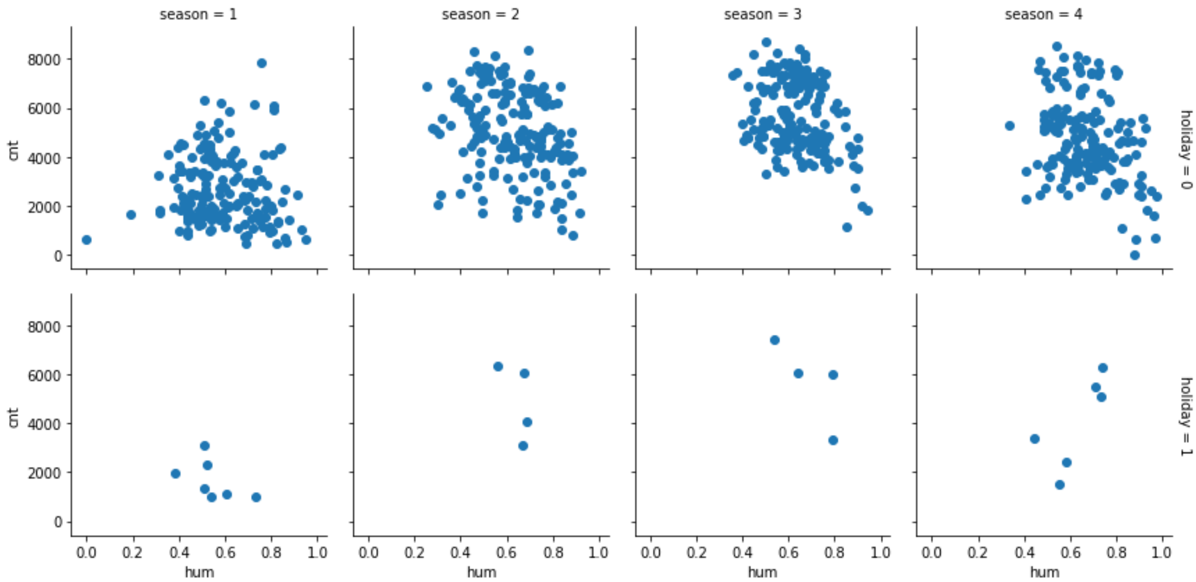

weekday = 0
weekday = 1
weekday = 2
weekday = 3
weekday = 4
weekday = 5
weekday = 6
weathersit = mist
weathersit = clear
weathersit = light snow
cnt
casual

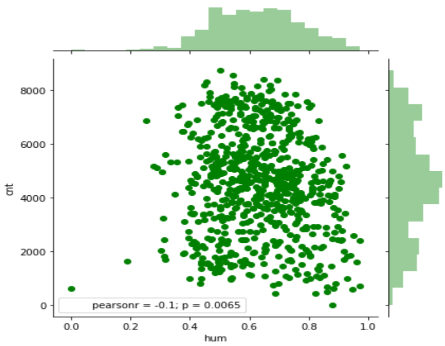

season
count
holiday

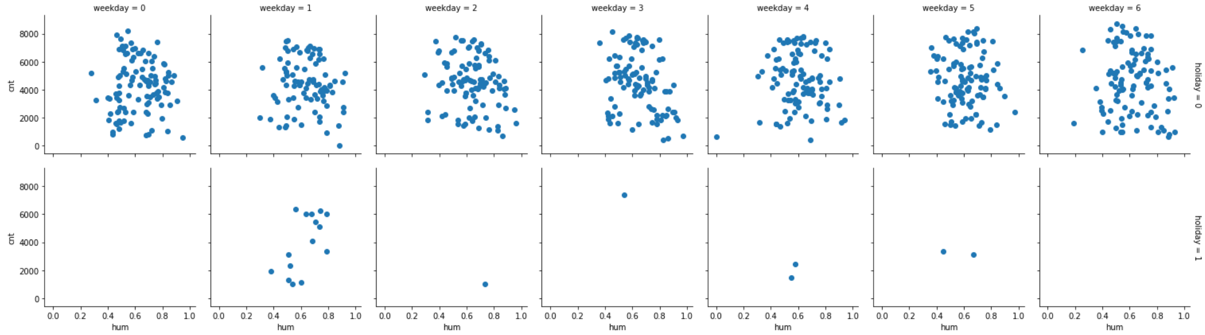

season
count
weathersit

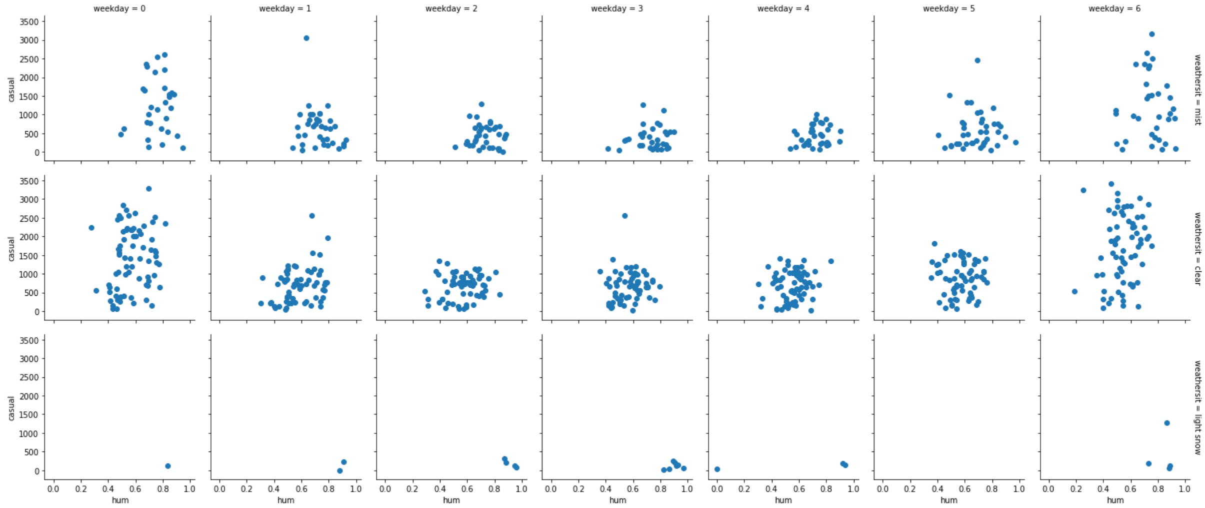

holiday and cnt Scatter Plot
cnt
holiday
0
2000
4000
6000
8000
0.0
0.2
0.4
0.6
0.8
1.0

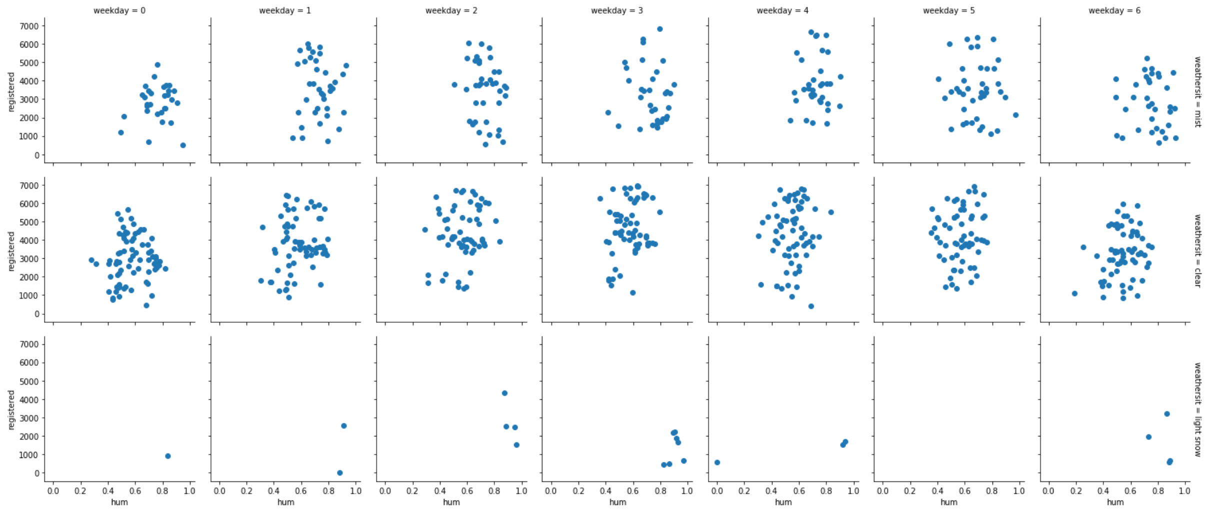

'holiday' wise sum of Total Count 'cnt'
3214244
78435
cnt
holiday
0
500000
1000000
1500000
2000000
2500000
3000000
0
1

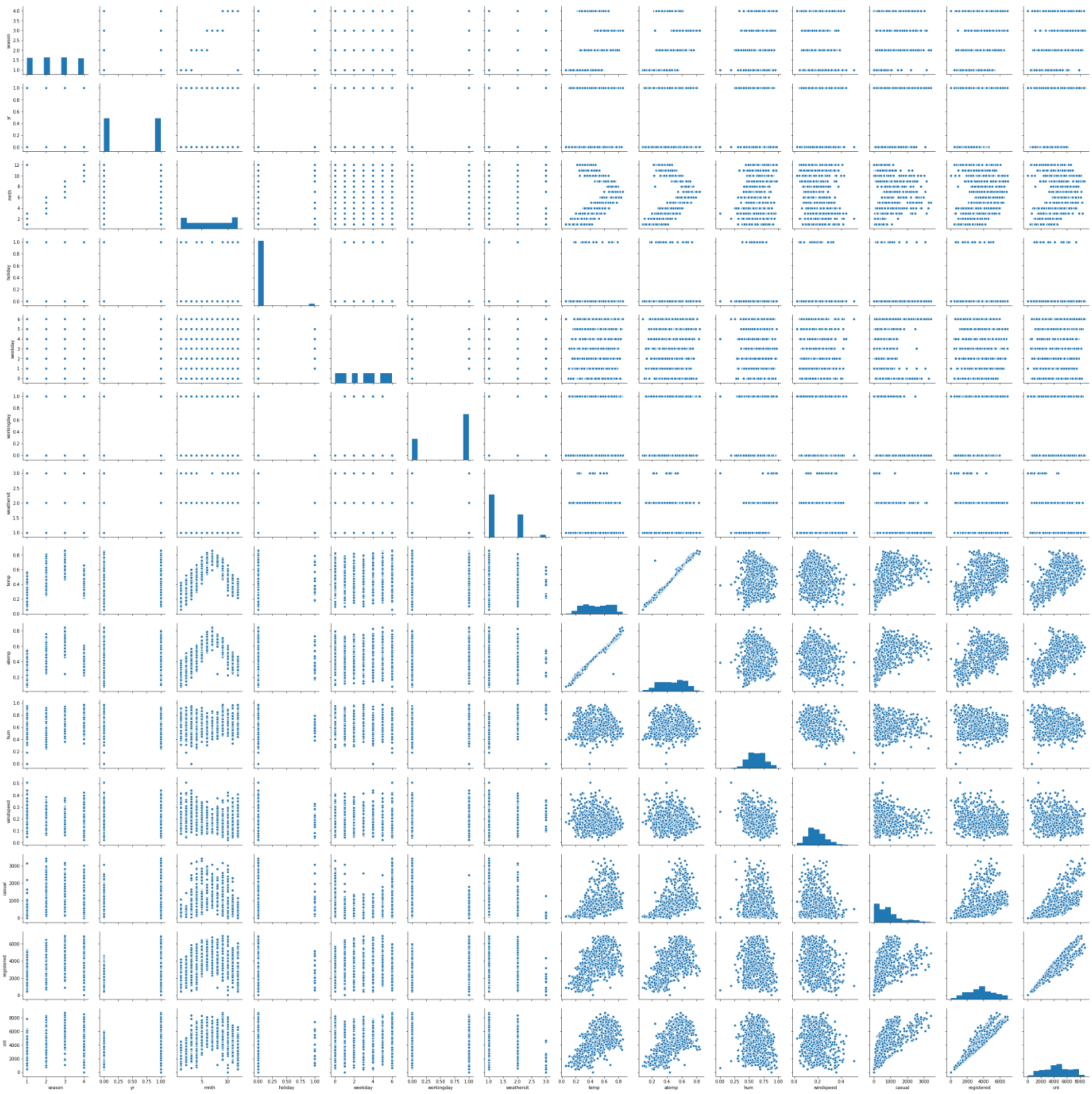

season = 1
season = 2
season = 3
season = 4
holiday = 0
holiday = 1
cnt
hum
0
2000
4000
6000
8000
0.0
0.2
0.4
0.6
0.8
1.0

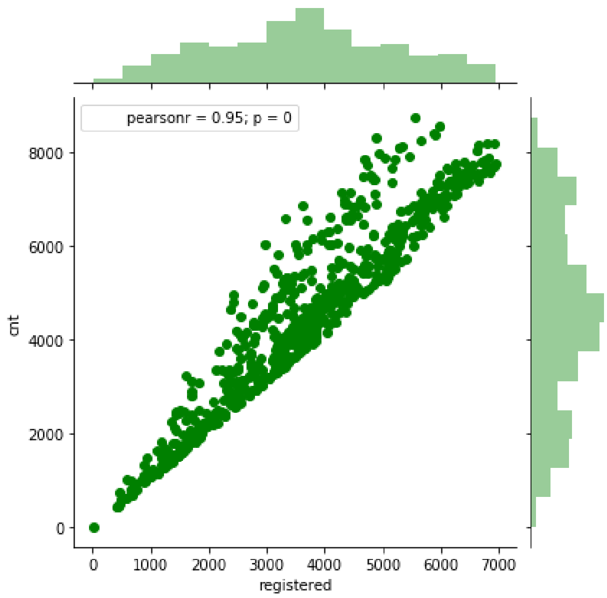

pearsonr = -0.1; p = 0.0065
cnt
hum
0
2000
4000
6000
8000
0.0
0.2
0.4
0.6
0.8
1.0

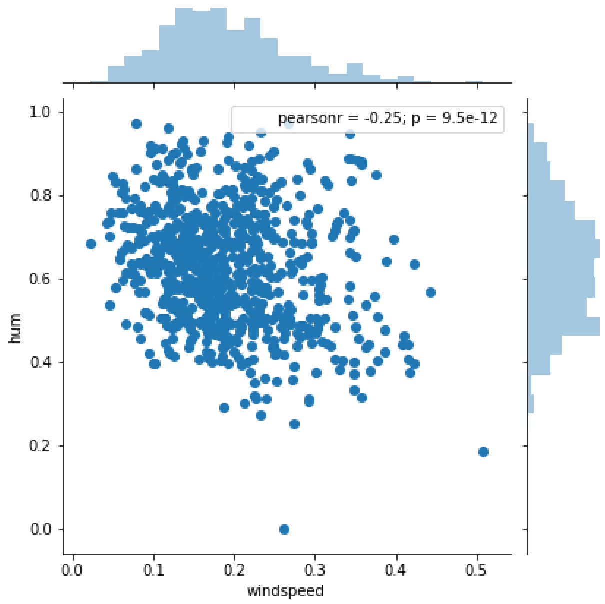

weekday = 0
weekday = 1
weekday = 2
weekday = 3
weekday = 4
weekday = 5
weekday = 6
holiday = 0
holiday = 1
cnt
hum

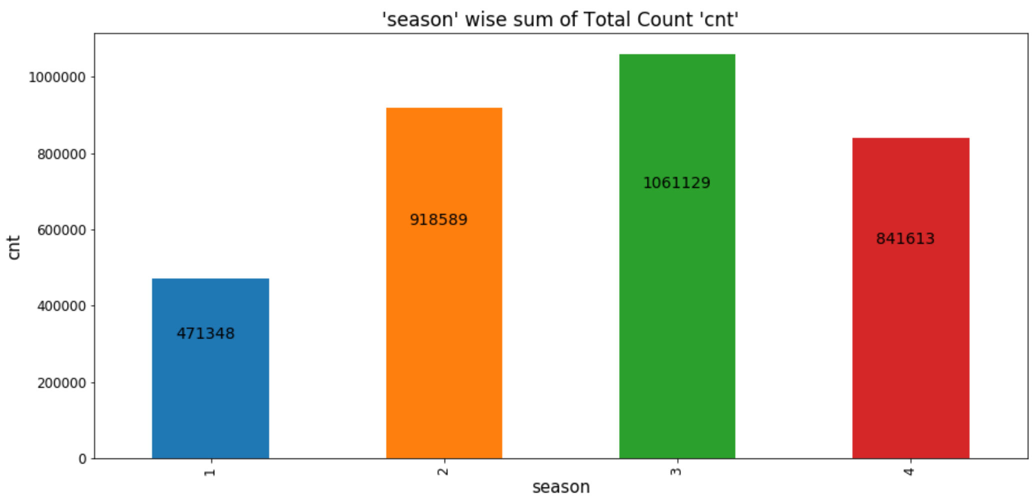

weekday = 0
weekday = 1
weekday = 2
weekday = 3
weekday = 4
weekday = 5
weekday = 6
weathersit = mist
weathersit = clear
weathersit = light snow
casual
hum

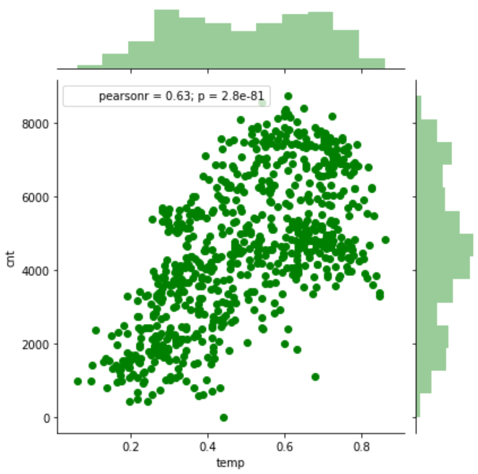

weekday = 0
weekday = 1
weekday = 2
weekday = 3
weekday = 4
weekday = 5
weekday = 6
weathersit = mist
weathersit = clear
weathersit = light snow
registered
hum

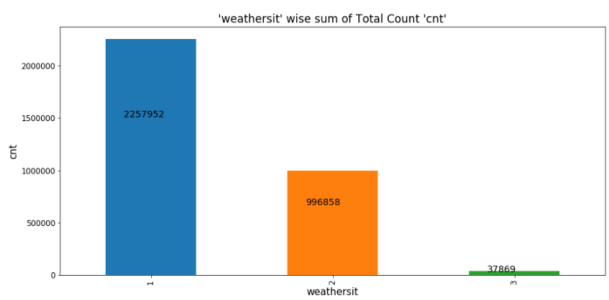


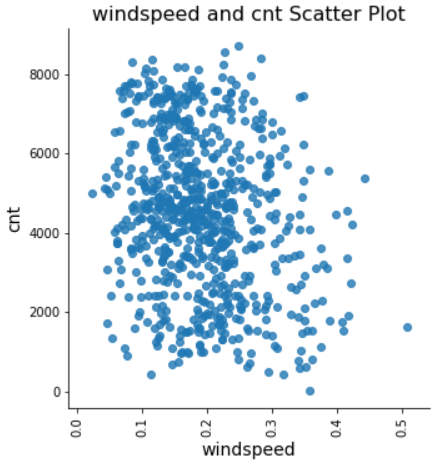

pearsonr = 0.95; p = 0
cnt
registered

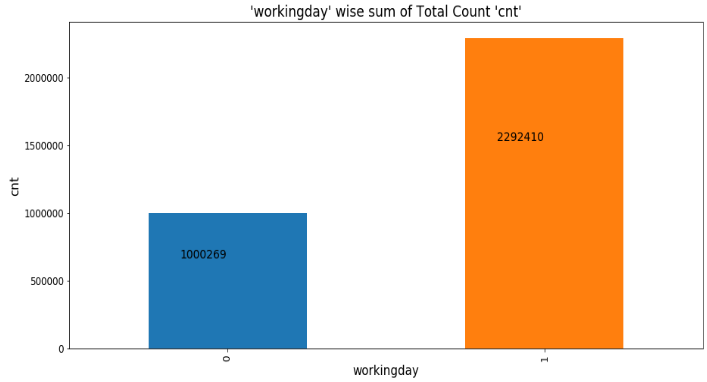

pearsonr = -0.25; p = 9.5e-12
hum
windspeed
0.0
0.2
0.4
0.6
0.8
1.0
0.1
0.3
0.5

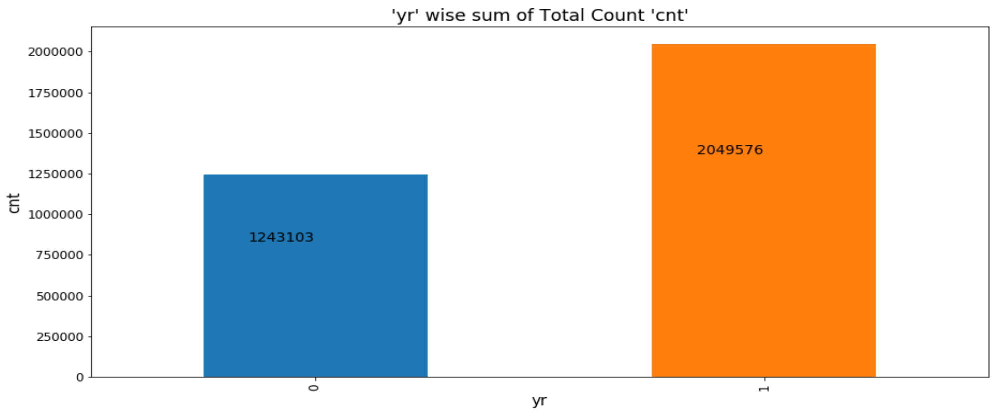

'season' wise sum of Total Count 'cnt'
cnt
season
471348
918589
1061129
841613
0
200000
400000
600000
800000
1000000
1
2
3
4

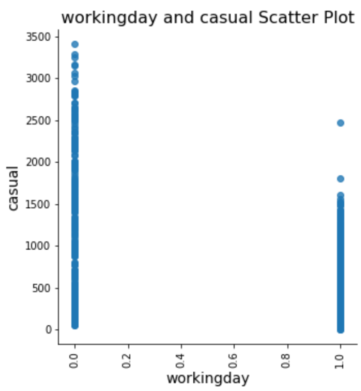

pearsonr = 0.63; p = 2.8e-81
cnt
temp
0
2000
4000
6000
8000
0.2
0.4
0.6
0.8

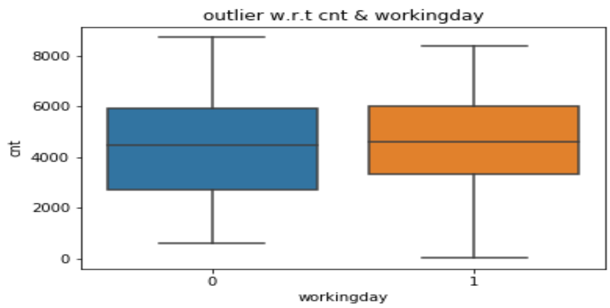

'weathersit' wise sum of Total Count 'cnt'
cnt
weathersit
2257952
996858
37869
0
500000
1000000
1500000
2000000
1
2
3

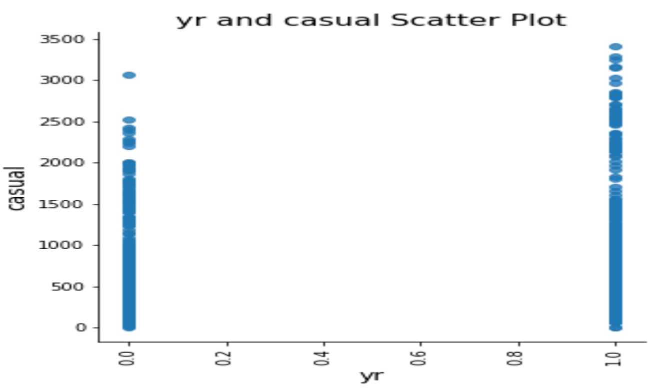

windspeed and cnt Scatter Plot
cnt
windspeed
0
2000
4000
6000
8000
0.0
0.1
0.2
0.3
0.4
0.5

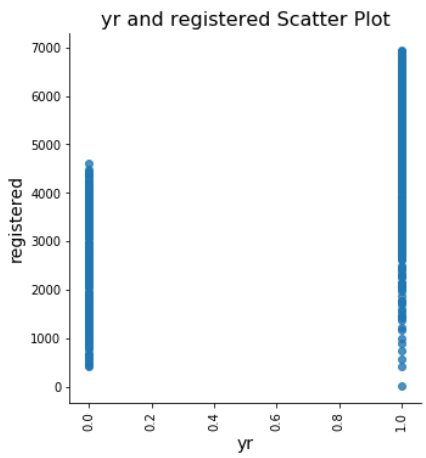

'workingday' wise sum of Total Count 'cnt'
cnt
workingday
1000269
2292410
0
500000
1000000
1500000
2000000
0
1

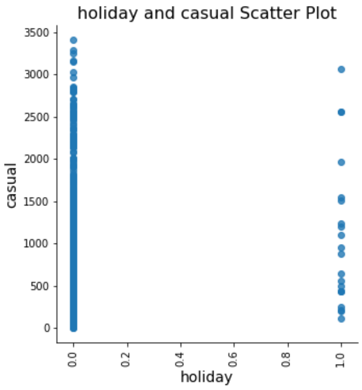

'yr' wise sum of Total Count 'cnt'
cnt
yr
1243103
2049576
0
250000
500000
750000
1000000
1250000
1500000
1750000
2000000
0
1

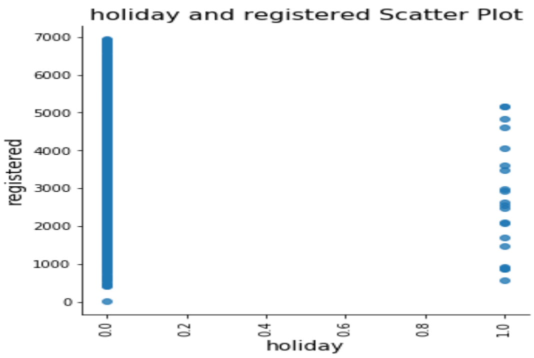

workingday and casual Scatter Plot
casual
workingday
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0

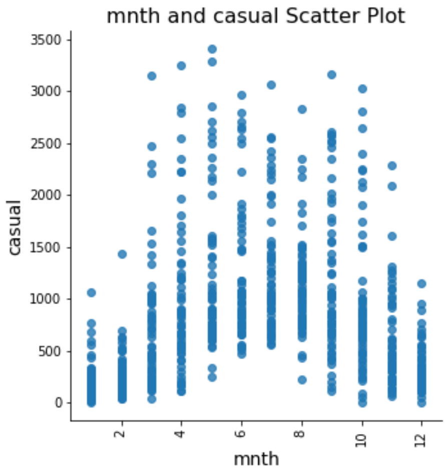

outlier w.r.t cnt & workingday
cnt
workingday
0
2000
4000
6000
8000
0
1

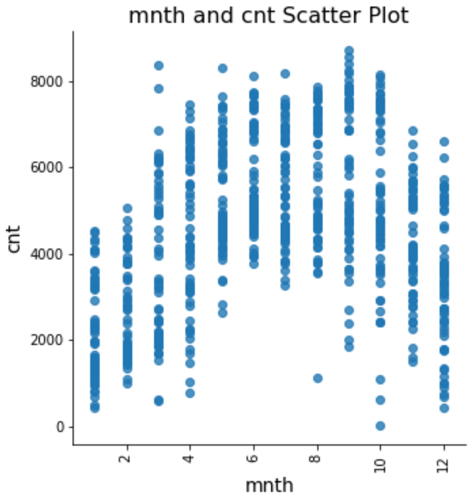

yr and casual Scatter Plot
casual
yr
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0

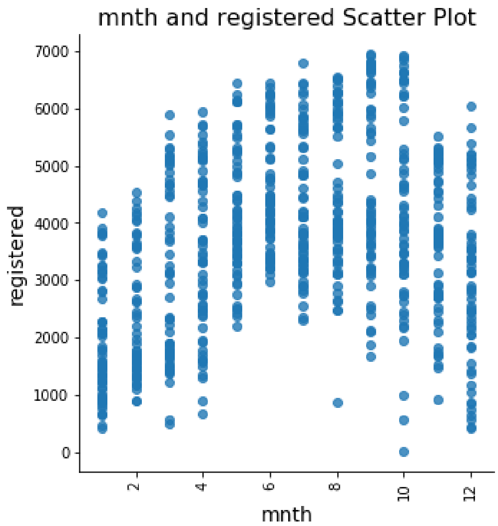

yr and registered Scatter Plot
registered
yr
0
1000
2000
3000
4000
5000
6000
7000
0.0
0.2
0.4
0.6
0.8
1.0

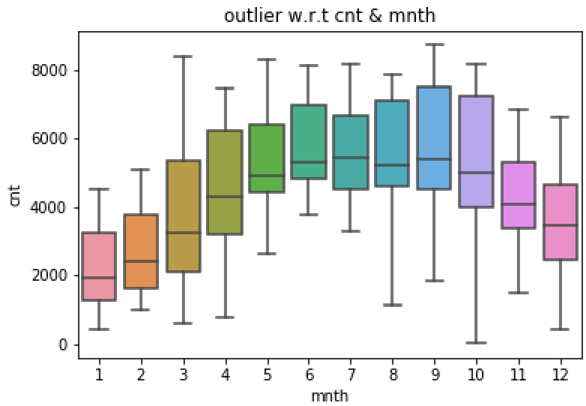

holiday and casual Scatter Plot
casual
holiday

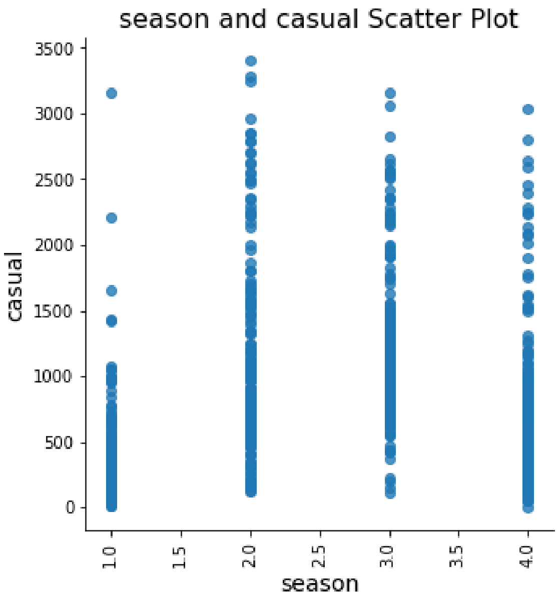

holiday and registered Scatter Plot
registered
holiday

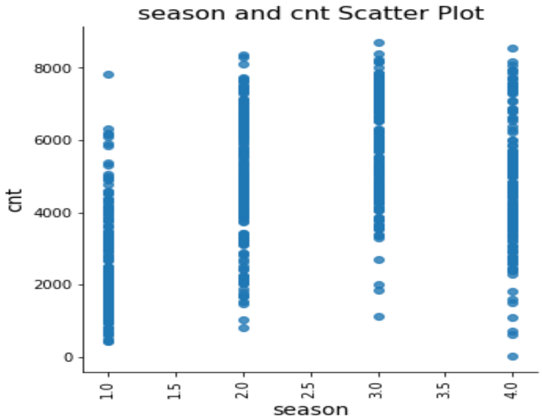

mnth and casual Scatter Plot
casual
mnth

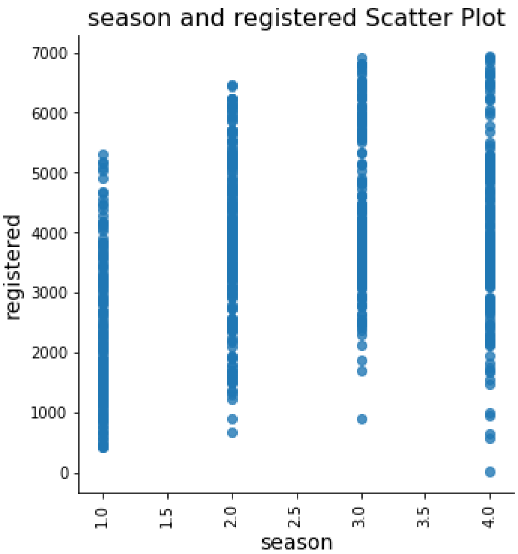

mnth and cnt Scatter Plot
cnt
mnth

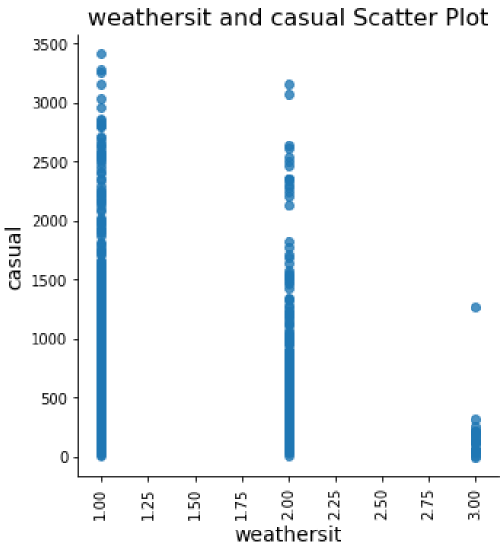

mnth and registered Scatter Plot
registered
mnth

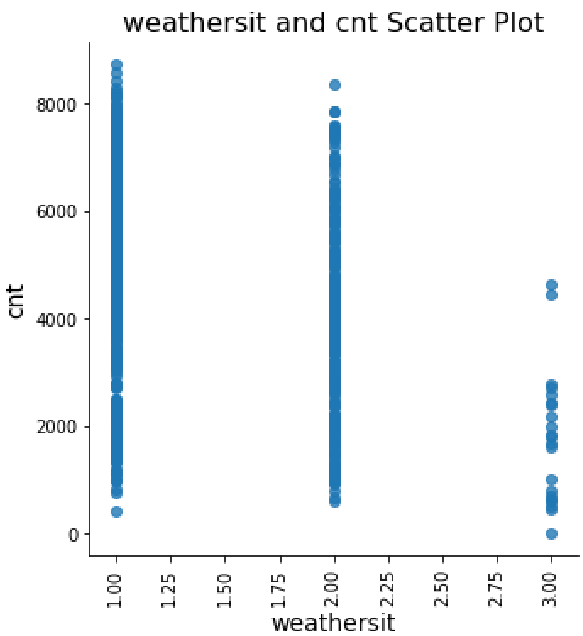

outlier w.r.t cnt & mnth
cnt
mnth

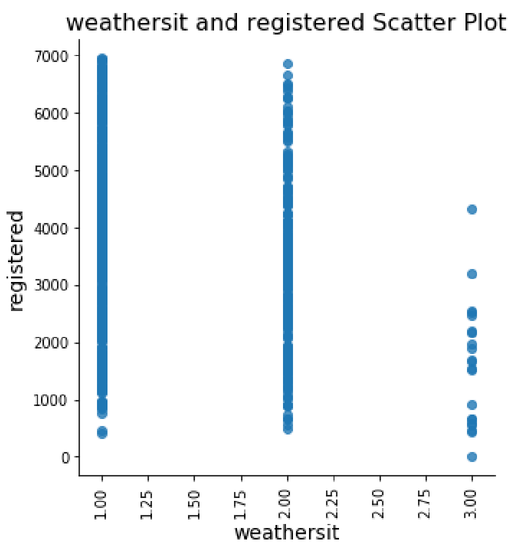

season and casual Scatter Plot
casual
0
500
1000
1500
2000
2500
3000
3500
1.0
1.5
2.0
2.5
3.0
3.5
4.0
season

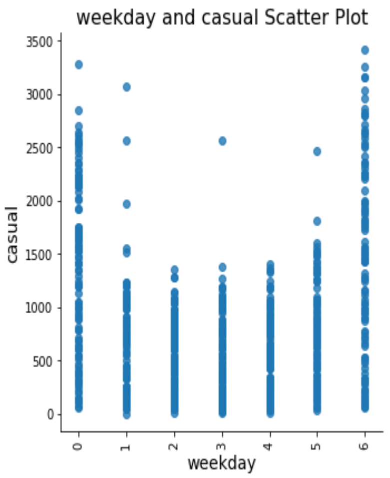

season and cnt Scatter Plot
cnt
0
2000
4000
6000
8000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
season

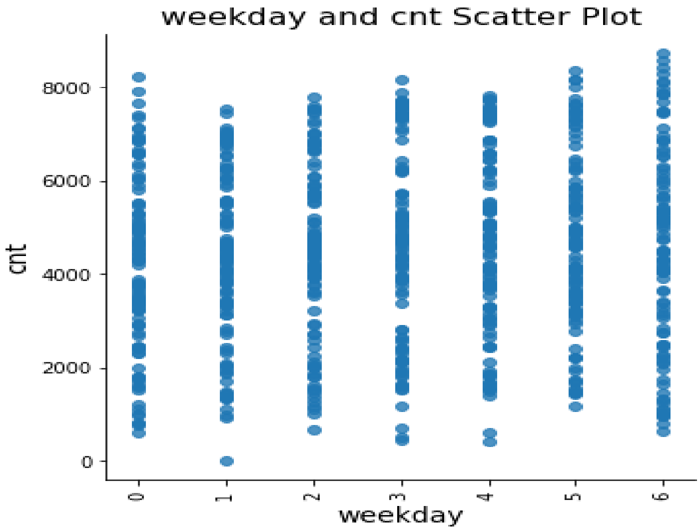

season and registered Scatter Plot
registered
0
1000
2000
3000
4000
5000
6000
7000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
season

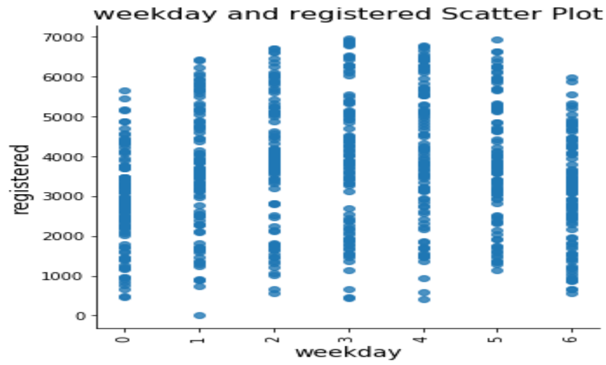

weathersit and casual Scatter Plot
casual
0
500
1000
1500
2000
2500
3000
3500
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
weathersit

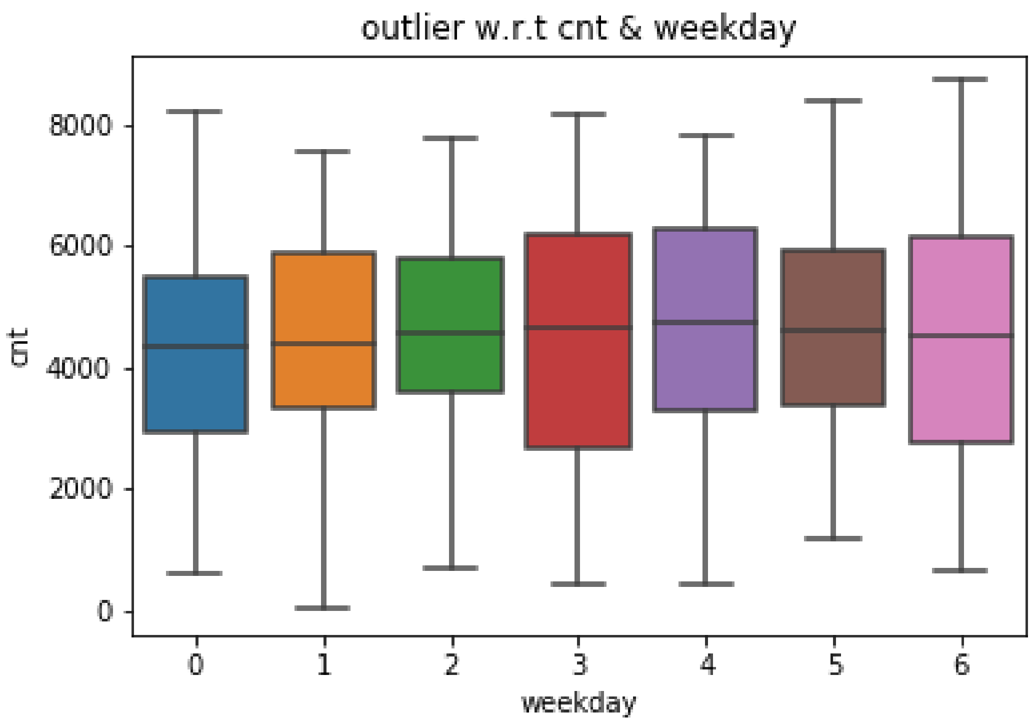

weathersit and cnt Scatter Plot
cnt
0
2000
4000
6000
8000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
weathersit


weathersit and registered Scatter Plot
registered
0
1000
2000
3000
4000
5000
6000
7000
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
weathersit


weekday and casual Scatter Plot
casual
weekday


weekday and cnt Scatter Plot
cnt
weekday


weekday and registered Scatter Plot
registered
weekday


outlier w.r.t cnt & weekday
cnt
weekday

Appendix – ll (Python Code)

```
# Importing Libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression,Ridge
from sklearn import metrics
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split,GridSearchCV,cross
_val_score
from sklearn.svm import SVR
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
os.getcwd()
original = pd.read_csv('../Data/day.csv')
df = original.copy()
df.head()
```

```
#Info Of data (dtypes // Shape)
df.info()
```

#Exploratory Data Analysis

```
#Convert to proper Date type
df.dteday = pd.to_datetime(df.dteday)

#Extracting only day Sequence
df['dteday'] = df.dteday.apply(lambda x: x.day)

#Converting to proper dtype
cat_var = ['dteday','season', 'yr', 'mnth', 'holiday', 'weekday', 'work
ingday','weathersit']
num_var = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered']

# #function for converting cat to num codes
for i in cat_var:
    df[i] = df[i].astype('object')
```

```
df.describe()
```

```
df.head()
```

```
#calculating all the unique values for all df columns
for i in df.columns:
    print(i,' --------> ',len(df[i].value_counts()))
```

```
# #function for converting cat to num codes
# for i in cat_var:
#     df[i] = df[i].astype('object')

# df = df.replace({'season':{1 : 'springer', 2 : 'summer',3 : 'fall' ,
4 : 'winter'}})
# df = df.replace({'yr':{0 : '2011', 1 : '2012'}})
# df = df.replace({'holiday' : {0 : 'no', 1 : 'yes'}})
# df = df.replace({'workingday' : {0 : 'no', 1 : 'yes'}})
# df = df.replace({'weathersit' : {1:"clear",2:"mist",3:"light snow"}})

# #df[['holiday','workingday']]
# #df.loc[(df['holiday'] == 'no') & (df['workingday'] == 'no'),:]

# df.to_csv('Labeled.csv')
```

#Missing Value Analysis

```
df.isnull().sum()
```

#No Missing Values Find

#Data Visualization

#♦ we know variable 'cnt = casual + registered'

```
plt.figure(figsize=(24,16))
plt.scatter(df['instant'], df['cnt'])
plt.xlabel('Days from January,1,2011 to December,31,2012', fontsize = 2
0)
plt.ylabel('Count', fontsize =20)
#plt.savefig('RentCount.png')
```

```
#removing instant
df.drop('instant',axis=1,inplace=True)
```

```
#Checking distribution of data via pandas visualization
df[num_var].hist(figsize=(20,20),color='g',alpha = 0.7)
#plt.savefig('distribution.png')
plt.show()
```

```
# Total count by season & holiday
fig = plt.figure(figsize=(10,7))
fig = sns.boxplot(x='season', y='cnt',hue='holiday', data=df)
plt.xlabel('Season',fontsize = 14)
```

```
plt.ylabel('cnt',fontsize = 14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.title('Distribution of Total Count in particular season with effect
 of holiday',fontsize=15)
#plt.savefig('dist_plot.png')
plt.show()
```

```
#Bar Plot Bivariate analysis
def _barplot(x,y,df):
    ss = df.groupby([x]).sum().transpose()
    ss = round(ss)
    ax = ss.loc[y].plot(kind='bar', figsize=(15,7))
    for i in ax.patches:
        ax.annotate(str(round(i.get_height())), (i.get_x()+.1, i.get_he
ight()/1.5),fontsize=14)
        #ax.text(i.get_x()/1.5, i.get_height()/1.5,str(round((i.get_hei
ght()))), fontsize=14)
    plt.xlabel(x,fontsize= 15)
    plt.ylabel(y,fontsize= 15)
    plt.xticks(fontsize=12,rotation = 90)
    plt.yticks(fontsize=12)
    plt.title("'{X}' wise sum of total '{Y}'".format(X=x,Y=y),fontsize
= 17)
    #plt.savefig("{X}_Vs_{Y}.png".format(X=x,Y=y))
    plt.show()
```

```
#Bar Plot of CNT w.r.t categorical variable
for i in ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday','w
eathersit']:
    _barplot(i,'cnt',df)
```

```
# #Each weekday wheather weather or not checking distribution of total
count
# for i in num_var:
#     g = sns.FacetGrid(df, col='weekday', row='weathersit', margin_tit
les=True)
#     ax = g.map(plt.scatter, i,"cnt")
#     #ax.savefig("{}_weekday_weathersituation_.png".format(i))
#     plt.show()
```

```
# #Each weekday wheather holiday or not checking distribution of casual
 count
# for i in num_var:
#     g = sns.FacetGrid(df, col='weekday', row='weathersit', margin_tit
les=True)
#     ax = g.map(plt.scatter, i,"casual")
#     #ax.savefig("{}_weekday_weathersituation_casual.png".format(i))
#     plt.show()
```

```
# #Each weekday weather situdtion checking distribution of registered c
ount
```

```
# for i in ['temp', 'atemp', 'hum', 'windspeed']:
#     g = sns.FacetGrid(df, col='weekday', row='weathersit',palette="Se
t1",hue="holiday")
#     ax = g.map(plt.scatter, i,"registered").add_legend()
#     #ax.savefig("{}_weekday_weathersituation_registered.png".format(i
))
#     plt.show()
```

```
#Joint plot of all numeric column
for i in num_var:
    fig = plt.figure(figsize=(10,7))
    fig = sns.jointplot(x=i, y="cnt", data=df,color='g')
    fig.set_axis_labels(xlabel=i,ylabel='cnt',fontsize=14)
    plt.suptitle("'{X}' and '{Y}' Scatter Plot".format(X=i,Y='cnt'),y =
 1.02,fontsize=15)
    #fig.savefig("{X}_and_{Y}_Scatter_Plot.png".format(X=i,Y='cnt'))
    plt.show()
```

```
#Total count by weather situation in particular season
fig = plt.figure()
fig = sns.countplot(x="weathersit", hue="season",data=df)
#plt.savefig('figg.png')
```

```
#atemp vs temp scatter plot
fig = plt.figure()
fig = sns.jointplot(x="temp", y="atemp", data=df)
#plt.savefig('scatt.png')
```

```
#hum vs windspeed
fig = plt.figure()
sns.jointplot(x="windspeed", y="hum", data=df)
#plt.savefig('scatt_hum_windspeed.png')
```

```
# fig = plt.figure()
# fig = sns.pairplot(df,size=2.5)
# plt.show()
# # fig.savefig('pairplot.png')
```

#Till Now we have analyse our data very breifly

#♦ Now Proceeding for Outliers

#Outlier Analysis

#Data spread According to total count. Scatter ploot od data will give us some instution about out #lier as the must farthest point or data point from entire data. We will consider that as an Outlier #and will treat it

```
#Scatter plot function
def diff_scattr(x,y):
    fig = plt.figure()
```

```
    fig = sns.lmplot(x,y, data=df,fit_reg=False)
    plt.xlabel(x,fontsize= 14)
    plt.ylabel(y,fontsize= 14)
    plt.xticks(fontsize=10, rotation=90)
    plt.yticks(fontsize=10)
    plt.title("{X} and {Y} Scatter Plot".format(X=x,Y=y),fontsize = 16)
    #fig.savefig("{X}_and_{Y}_Scatter_Plot..png".format(X=x,Y=y))
    plt.show()
```

```
for i in num_var:
    diff_scattr(x=i,y='cnt')
```

```
for i in ['temp','atemp','hum','windspeed']:
    diff_scattr(x=i,y='casual')
```

```
for i in ['temp','atemp','hum','windspeed']:
    diff_scattr(x=i,y='registered')
```

```
for i in cat_var:
    diff_scattr(x=i,y='cnt')
```

```
for i in cat_var:
    diff_scattr(x=i,y='casual')
```

```
for i in cat_var:
    diff_scattr(x=i,y='registered')
```

#♦ There are Few outliers in our data

```
# #Plotting Box Plot
for i in ['temp','atemp','hum','windspeed']:
    plt.figure()
    plt.clf() #clearing the figure
    sns.boxplot(df[i],palette="Set2")
    plt.title(i)
    #plt.savefig('{}_.png'.format(i))
    plt.show()
```

```
# #Plotting Box Plot
for i in cat_var:
    plt.figure()
    plt.clf() #clearing the figure
    sns.boxplot(x=i, y="cnt", data=df)
    plt.title(('outlier w.r.t cnt & {}').format(i))
    #plt.savefig('{}_cat_box_.png'.format(i))
    plt.show()
```

```
#Treating Out Liers and Converting them to nan
for i in ['temp','atemp','hum','windspeed']:
    #print(i)
    q75, q25 = np.percentile(df.loc[:,i], [75 ,25])
    iqr = q75 - q25
    minn = q25 - (iqr*1.5)
    maxx = q75 + (iqr*1.5)
#Converting to nan
    df.loc[df.loc[:,i] < minn,i] = np.nan
    df.loc[df.loc[:,i] > maxx,i] = np.nan
    print('{var} --------- :- {X}   Missing'.format(var = i, X = (df.lo
c[:,i].isnull().sum())))
```

```
df[df['windspeed'].isnull() | df['hum'].isnull()]
#null value indexed = [44, 49, 68, 93, 94, 292, 382, 407, 420, 432, 433
, 450, 666, 721]
```

```
df[['hum','windspeed']].describe().transpose()
```

```
#Imputing values as mean
df.windspeed = df.windspeed.fillna(df.windspeed.mean())
df.hum = df.hum.fillna(df.hum.mean())
```

#Creating Weekend Column

```
# end = []

# for i in df.weekday:
#     if i == 0:
#         end.append(1)
#     elif i == 6:
#         end.append(1)
#     else:
#         end.append(0)

# df['weekend'] = end
# df['weekend'] = df['weekend'].astype('object')
# df = df[['dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday','work
ingday','weekend','weathersit', 'temp',\
#           'atemp', 'hum', 'windspeed','casual', 'registered', 'cnt']]
```

 # Correlation Check

```
#Setting up the pane or matrix size
f, ax = plt.subplots(figsize=(10,8))  #Width,height

#Generating Corelation Matrix
corr = df[['temp','atemp','hum','windspeed','casual', 'registered','cnt
']].corr()

#corr = df[['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'we
ekday',\
#        'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed'
,\
```

```
#       'casual', 'registered', 'weekend','cnt']].corr()

#Plot using Seaborn library
sns.heatmap(corr,mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.dive
rging_palette(220,10, as_cmap=True),\
            square=True, ax=ax,annot=True,linewidths=1 , linecolor= 'bl
ack',vmin = -1, vmax = 1)

plt.show()
#f.savefig('heatmap.png')
```

Variable 'atemp' & 'temp' are highly correlated

 # Chi-Square Test Among different Independent Variable

```
#H1 = Variables are not independent
#H0 = Variable are independent
#If p-value is less than 0.05 we will reject null hyothesis by saying a
lternate hypothesis is true

#from scipy.stats import chi2_contingency

#Chi Function
def chi_check(df):
    #getting all the column name as object or category
    cat_names = df.select_dtypes(exclude=np.number).columns.tolist()
    cat_pair = [(i,j) for i in cat_names for j in cat_names] #creating
pairs of column

    p_values =[]
    for i in cat_pair:
        #print(i[0],i[1])
        if i[0] != i[1]:
            chi_result = chi2_contingency(pd.crosstab(df.loc[:,i[0]], d
f.loc[:,i[1]]))
            p_values.append(chi_result[1])
        else:
            p_values.append(0)

    chi_mat = np.array(p_values).reshape(len(cat_names),len(cat_names))
    chi_mat = pd.DataFrame(chi_mat, index = cat_names, columns = cat_na
mes)
    return chi_mat
```

```
chi_check(df)
```

#As ['holiday','workingday','weekend'] are not so independent and might cause problem so
#removing them

#Season and mnth column are also highly related to each other

```
chi_check(df[['dteday','season','yr','mnth','weekday','weathersit']])
```

#Anova Test

```
# import statsmodels.api as sm
# from statsmodels.formula.api import ols

def one_way_anova(df,target):
    predictor_list = df.select_dtypes(exclude=np.number).columns.tolist
()
    for i in predictor_list:
        mod = ols(formula=('{} ~ {}').format(target, i),data=df).fit()
        rs = sm.stats.anova_lm(mod, typ=1)
        print(('Anova p- value b/w {} and {} ----->   {}').format(targe
t,i,rs.iloc[0][4]))
```

```
print('------------------------------------------------------------target v
ar = CNT')
one_way_anova(df,'cnt')
print()
print('------------------------------------------------------------target v
ar = Casual')
one_way_anova(df,'casual')
print()
print('------------------------------------------------------------target v
ar = REGISTERED')
one_way_anova(df,'registered')
```

#After Anova analysation if we go with cnt as our target variable the we have to remove the #variable 'weekday' & 'workingday' as both have pvalues more than 0.05.But if we mark casual #and register as our target var then we dnt need to remove it

#Also cnt = casual + registered

#Multicollenierity Check

#V.I.F.=1/(1−R2).

```
from patsy import dmatrices
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_fac
tor

# get y and X dataframes based on CNT this regression:
#y, X = dmatrices('cnt ~ + season + yr + mnth + weathersit + temp + hum
 + windspeed',\
#                   df, return_type='dataframe')

y, X = dmatrices('cnt ~ + dteday + season + yr + mnth + workingday + we
athersit + temp + hum + windspeed',df, return_type='dataframe')

# For each X, calculate VIF and save in dataframe
```

```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in ra
nge(X.shape[1])]
vif["features"] = X.columns
vif.round(1)
```

```
df.columns
```

```
#Removing Variables
#df = df.drop(['atemp','holiday','weekday'],axis=1)
df = df.drop(['atemp','holiday','workingday'],axis=1)
```

#Creating Dummies of Categorical variables

```
# def dummy_func(df):
#     #Extracting all object var
#     cat_names = df.select_dtypes(exclude=np.number).columns.tolist()

#     ##Creating Dummies
#     for i in cat_names:
#         dummies = pd.get_dummies(df[i], prefix= i, dummy_na=False)
#         df = df.drop(i, 1)
#         df = pd.concat([df, dummies],axis = 1)

#     #Converting back to object
#     for i in df.columns:
#         if df[i].dtypes == 'uint8':
#             df[i] = df[i].astype('object')
#     return df
```

```
# #Creating Dummies
# df_dummy = dummy_func(df)
# df_dummy.shape, df.shape
```

```
# df_dummy = df_dummy[['temp','hum','windspeed','dteday_1','dteday_2','
dteday_3','dteday_4','dteday_5','dteday_6','dteday_7',\
#                        'dteday_8','dteday_9','dteday_10','dteday_11','d
teday_12','dteday_13','dteday_14','dteday_15',\
#                        'dteday_16','dteday_17','dteday_18','dteday_19',
'dteday_20','dteday_21','dteday_22','dteday_23',\
#                        'dteday_24','dteday_25','dteday_26','dteday_27',
'dteday_28','dteday_29','dteday_30','dteday_31',\
#                        'season_1','season_2','season_3','season_4','yr_
0','yr_1','mnth_1','mnth_2','mnth_3','mnth_4',\
#                        'mnth_5','mnth_6','mnth_7','mnth_8','mnth_9','mn
th_10','mnth_11','mnth_12','workingday_0',\
#                        'workingday_1','weathersit_1','weathersit_2','we
athersit_3','casual','registered','cnt']]
# df_dummy.shape
```

Feature Scaling¶

```
# df[['cnt','casual','registered']].describe().transpose()
# #cnt_min = 22.0 // cnt_max = 8714
```

```
# # #Normalization of cnt
# df['total_cnt'] = (df['cnt'] - min(df['cnt'])) / (max(df['cnt']) - mi
n(df['cnt']))

# #Checking Normalised
# df[['cnt','total_cnt']].describe().transpose()
```

Sampling

```
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,:-3],
                                                    df.iloc[:,-1],
                                                    test_size=0.20,
                                                    random_state=101)
```

Modeling

Base Models

- Linear Regresion
- Decision Tree
- Random Forest
- Ridge

```
#from sklearn import metrics

# Regression
# 'explained_variance'       metrics.explained_variance_score
# 'neg_mean_absolute_error'  metrics.mean_absolute_error
# 'neg_mean_squared_error'   metrics.mean_squared_error
# 'neg_mean_squared_log_error'       metrics.mean_squared_log_error
# 'neg_median_absolute_error'metrics.median_absolute_error
# 'r2' metrics.r2_score

def results(y_test,y_pred):
    print('R2 score ==>  ', round(metrics.r2_score(y_test, y_pred), 2))
    print(('Mean absolute percentage error ==>  {} % ').format(round(np
.mean(np.abs((y_test - y_pred) / y_test))*100, 2)))
    #print('Mean Squared Error ==>  ', round(metrics.mean_squared_error
(y_test, y_pred), 2))
    print('Root Mean Squared Error ==> ', round(np.sqrt(metrics.mean_sq
uared_error(y_test, y_pred)), 2))

def cross_val(model):
```

```
    acc = cross_val_score(model, X_train, y_train, cv=10,scoring='r2',n
_jobs=-1,)
    print('Mean Score of Cross validation = ',round(acc.mean(),2))
    print('Standard Deviation of CV = ',round(acc.std(),2))

def test_scores(model):
    print('<<<------------------- Training Data Score -----------------
---->')
    print()
    #Predicting result on Training data
    y_pred = model.predict(X_train)
    results(y_train,y_pred)
    print()
    print('<<<------------------- Test Data Score ---------------------
>')
    print()
    # Evaluating on Test Set
    y_pred = model.predict(X_test)
    results(y_test,y_pred)
```

Linear Regression

```
#from sklearn.linear_model import LinearRegression

linear_model = LinearRegression().fit(X_train,y_train)
test_scores(linear_model)

#cross_val(linear_model)
# # Mean Score of Cross validation =  0.77
# # Standard Deviation of CV =  0.05
```

```
# # Grid Search on LR model for best Parameters
# _model = LinearRegression(normalize=True)
# pdict = [{'copy_X':[True, False],
#           'fit_intercept':[True,False]}]
# g_srch_lm = GridSearchCV(_model, param_grid = pdict, scoring='r2' , c
v =10, n_jobs =-1).fit(X_train,y_train)

# #Best Score
# print('Best Score ===> ',g_srch_lm.best_score_)
# print('Best Param ===> ',g_srch_lm.best_params_)

# test_scores(g_srch_lm)
```

Decision Tree

```
#from sklearn.tree import DecisionTreeRegressor

tree_model = DecisionTreeRegressor(random_state=101).fit(X_train,y_trai
n)
test_scores(tree_model)
```

```
#cross_val(tree_model)
# # Mean Score of Cross validation =  0.73
# # Standard Deviation of CV =  0.06
```

```
# # Grid Search on Decision Tree model for best Parameters
# _model = DecisionTreeRegressor(random_state=101)
# pdict = [{'max_depth':[2,4,6,8,10,12,15],
#           'max_features':['auto','sqrt'],
#           'min_samples_leaf':[2,4,6,8,10]}]
# g_srch = GridSearchCV(_model, param_grid = pdict, scoring='r2' , cv =
10, n_jobs =-1).fit(X_train,y_train)

# #Best Score
# print('Best Score ===> ',g_srch.best_score_)
# print('Best Param ===> ',g_srch.best_params_)
# test_scores(g_srch)
```

Random Forest

```
#from sklearn.ensemble import RandomForestRegressor

forest_model = RandomForestRegressor(n_estimators=500,random_state=101)
.fit(X_train,y_train)
test_scores(forest_model)

#cross_val(forest_model)
# # Mean Score of Cross validation =  0.87
# # Standard Deviation of CV =  0.03
```

```
# # Grid Search for finding Random forest best Parameters
# _model = RandomForestRegressor(random_state=101, n_jobs=-1)
# pdict = [{'max_depth':[2,4,6,8,10],
#           'max_features':['auto','sqrt'],
#           'n_estimators': [200,300,400,500,600,700,800,1000]}]

# g_srch = GridSearchCV(_model, param_grid = pdict, cv =10, n_jobs =-1)
.fit(X_train,y_train)

# #Best Score
# print('Best Score ===> ',g_srch.best_score_)
# print('Best Param ===> ',g_srch.best_params_)
# test_scores(g_srch)

# #'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 500
# #{'max_depth': 15, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'n_
estimators': 300}
# # Best Score ===>  0.871143807987822
# # Best Param ===>  {'max_depth': 10, 'max_features': 'sqrt', 'n_estim
ators': 300}
```

SVR

```
#from sklearn.svm import SVR
svr_model = SVR(kernel='poly').fit(X_train,y_train)
```

```
test_scores(svr_model)

#cross_val(svr_model)
# Mean Score of Cross validation =  0.6
# Standard Deviation of CV =  0.05
```

Random forest model has out perfomed out of all models used. Final Model with Optimized Parameters :- RANDOM FOREST

```
#from sklearn.ensemble import RandomForestRegressor

forest_model = RandomForestRegressor(max_depth= 15, max_features = 'sqr
t',n_estimators = 500,random_state=101).fit(X_train,y_train)
test_scores(forest_model)

#cross_val(forest_model)
# # Mean Score of Cross validation =  0.88
# # Standard Deviation of CV =  0.03

# #max_depth= 10, max_features = 'sqrt',n_estimators = 500
# #max_depth = 15, max_features = 'sqrt', min_samples_leaf = 2, n_estim
ators = 500
# #max_depth = 10, max_features = 'sqrt', n_estimators = 300
```

Feature Importance

```
#Calculating feature importances
importances = forest_model.feature_importances_

# Sort feature importances in descending order
indices = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances
names = [df.columns[i] for i in indices]

# Creating plot
fig = plt.figure(figsize=(20,20))
plt.title("Feature Importance")

# Add horizontal bars
plt.barh(range(X_train.shape[1]),importances[indices],align = 'center')
plt.yticks(range(X_train.shape[1]), names)
plt.show()
#fig.savefig('feature_importance.png')
```

Saving Output

In [ ]:

```
#Predicting Output On entire Data
pred_rf = forest_model.predict(df.iloc[:,:-3])
df['predict'] = pred_rf
```

```
#Standard result with original
entire_data = pd.concat([original,df['predict']], axis=1)
```

```
entire_data.head()
#Entire _ENV
entire_data.to_csv('../Data/output/Py_output/Entire_output.csv')
#Season
entire_data[['dteday','weathersit','season','cnt','predict']].to_csv('.
./Data/output/Py_output/Season_output.csv')
```

Appendix – lll (R Code)

```
rm(list = ls())

setwd("C:/Users/parve/Documents/Bike_Renting/R_Code")

# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
    "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart")

# #install.packages if not
# #lapply(x, install.packages)
#
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)
######################################################################

#Loading Data
original_data = read.csv('../Data/day.csv',header = T,na.strings = c(""," ","NA"))
df = original_data    #Creating backup of orginal data

############################################################################
#              EXPLORING DATA
############################################################################

#viewing the data
head(df,4)
```

```
dim(df) #shape of data = row 731  ===  col = 16

#structure of data or data types
str(df)

#Summary of the data
summary(df)

#Carrying out date number
df$dteday <- format(as.Date(df$dteday,format="%Y-%m-%d"), "%d")

#removing instant
df$instant <- NULL

#unique value of each count
apply(df, 2,function(x) length(table(x)))

#Distribution of cnt variable
hist(df$cnt)

#Converting to proper dtype
cat_var = c('dteday','season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday','weathersit')
num_var = c('temp', 'atemp', 'hum', 'windspeed','casual','registered')

#Data Type Conversion Function
typ_conv = function(df,var,type){
  df[var] = lapply(df[var], type)
```

```
  return(df)
}

df = typ_conv(df,cat_var, factor)

###########################################################################
#         Checking Missing data
###########################################################################
apply(df, 2, function(x) {sum(is.na(x))}) #2 for columns as in R 1 = Row & 2 = Col

#Hence no missing data found

###########################################################################
#         Visualizing teh data
###########################################################################
hist(df$casual)
hist(df$registered)
hist(df$cnt)

#library(ggplot2)
# CNT according to Season
ggplot(df, aes(fill=cnt, x=season)) +
  geom_bar(position="dodge") + labs(title="cnt ~ season")

# CNT according to holiday
ggplot(df, aes(fill=cnt, x=holiday)) +
  geom_bar(position="dodge") + labs(title="cnt ~ holiday")
```

```
# CNT according to season by yr
ggplot(df, aes(fill=cnt, x=season)) +
  geom_bar(position="dodge") + facet_wrap(~yr)+
  labs(title="CNT according to season by yr")

# CNT according to season by workingday
ggplot(df, aes(fill=cnt, x=season)) +
  geom_bar(position="dodge") + facet_wrap(~workingday)+
  labs(title="CNT according to season by workingday")

# CNT according to season by workingday
ggplot(df, aes(fill=cnt, x=workingday)) +
  geom_bar(position="dodge") + facet_wrap(~weekday)+
  labs(title="CNT according to workingday by weekday")

###################################################################
#              Outlier Analysis
###################################################################

#  #We are skipping outliers analysis becoz we already have an Class Imbalance problem.

# for (i in 1:length(num_var))
# {
#   assign(paste0("gn",i),
#          ggplot(aes_string(y = (num_var[i]), x = 'cnt'),data = df) +
#            stat_boxplot(geom = "errorbar", width = 0.5) +
```

```
#          geom_boxplot(outlier.colour="blue", fill = "skyblue",
#                     outlier.shape=18,outlier.size=1, notch=FALSE) +
#          labs(y=num_var[i],x="cnt")+
#          ggtitle(paste("Box plot of responded for",num_var[i])))
# }

#gn1
#
# Plotting plots together
# gridExtra::grid.arrange(gn1,gn2,gn3,gn4,ncol=4)
# gridExtra::grid.arrange(gn5,gn6,gn7,ncol=3)
# gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,gn7,ncol=4,nrow = 2)



#Removing oulier by replacing with NA and then impute
for(i in c('temp', 'atemp', 'hum', 'windspeed')){
  print(i)
  outv = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(outv))
  df[,i][df[,i] %in% outv] = NA
}

#checking all the missing values
#library(DMwR)
sum(is.na(df))

df$hum[is.na(df$hum)] = mean(df$hum,na.rm = T)
```

```
df$windspeed[is.na(df$windspeed)] = mean(df$windspeed, na.rm = T)

#df = knnImputation(df, k=3)

sum(is.na(df))
################################################################
#           Feacture Selection
################################################################

#Here we will use corrgram library to find corelation

##Correlation plot
# library(corrgram)
num_var = c('temp', 'atemp', 'hum', 'windspeed','casual','registered','cnt')

corrgram(df[,num_var],
        order = F,  #we don't want to reorder
        upper.panel=panel.pie,
        lower.panel=panel.shade,
        text.panel=panel.txt,
        main = 'CORRELATION PLOT')
#We can see var the highly corr related var in plot marked dark blue.
#Dark blue color means highly positive cor related

df = subset(df, select=-c(atemp,casual,registered))
```

```
# #Checking dependency among different categorical variables

cat_var = c('dteday','season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday','weathersit')

cat_df = df[,cat_var]


for (i in cat_var){

  for (j in cat_var){

    print(i)

    print(j)

    print(chisq.test(table(cat_df[,i], cat_df[,j]))$p.value)

  }

}


#anova test


anova_season =(lm(cnt ~ season, data = df))

summary(anova_season)


anova_year =(lm(cnt ~ yr, data = df))

summary(anova_year)


anova_month =(lm(cnt ~ mnth, data = df))

summary(anova_month)


anova_holiday =(lm(cnt ~ holiday, data = df))

summary(anova_holiday)


anova_weekday =(lm(cnt ~ weekday, data = df))
```

```
summary(anova_weekday)


anova_workingday =(lm(cnt ~ workingday, data = df))
summary(anova_workingday)


anova_weathersit =(lm(cnt ~ weathersit, data = df))
summary(anova_weathersit)


anova_season =(lm(cnt ~ dteday, data = df))
summary(anova_season)



##################################################
# #check multicollearity
##################################################
# #Linear Regression
#library(usdm)


vif(df)
#vifcor(df[,c(7,8,9)])


df = subset(df, select=-c(holiday, workingday,dteday))
#dteday


#####################################
## Feature Scaling
#####################################
```

```
#min(df$cnt) ---->  22

#max(df$cnt) ---->  8714

hist(df$cnt)

colnames(df)


# #Normalization of cnt

#df$total_cnt = (df$cnt - min(df$cnt)) / (max(df$cnt) - min(df$cnt))


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

#            Sampling of Data

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


#sampling

set.seed(12345)

t_index = sample(1:nrow(df), 0.8*nrow(df))

train = df[t_index,]

test = df[-t_index,]




#Removing All the custom variable from memory

#library(DataCombine)

rmExcept(c("test","train","original_data",'df'))


#library(caret)


#mape
```

```
mape = function(actual, predict){
  mean(abs((actual-predict)/actual))*100
}

############################################
# # # ??? Linear Regression
# #########################################
#
##Linear regression
dumy = dummyVars(~., df)
dummy_df = data.frame(predict(dumy, df))

set.seed(101)
dum_index = sample(1:nrow(dummy_df), 0.8*nrow(dummy_df))
dum_train_df = dummy_df[dum_index,]
dum_test_df = dummy_df[-dum_index,]

#Linear model
lr_model = lm(cnt ~. , data = dum_train_df)
summary(lr_model)

#predictions on Train data set
LR_predict_train = predict(lr_model, dum_train_df[,-32])
plot(dum_train_df$cnt, LR_predict_train,
     xlab = 'Actual values',
     ylab = 'Predicted values',
     main = 'LR model')
```

```
#evaluation

postResample(LR_predict_train, dum_train_df$cnt)#R-sq = 0.85

mape(dum_train_df$cnt, LR_predict_train)



#predictions on test

LR_predict_test = predict(lr_model, dum_test_df[,-32])

plot(dum_test_df$cnt, LR_predict_test,

    xlab = 'Actual values',

    ylab = 'Predicted values',

    main = 'LR model')


#evaluation

postResample(LR_predict_test, dum_test_df$cnt)#R-sq = 0.85

mape(dum_test_df$cnt, LR_predict_test)



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

# # # # ??? Decision Tree

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

#

##Decison tree


# library(rpart.plot)

# library(rpart)
```

```
set.seed(121)

#model

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

summary(dt_model)

plt = rpart.plot(dt_model, type = 5, digits = 2, fallen.leaves = TRUE)


#predictions on train

DT_Predict_train = predict(dt_model, train[,-9])

plot(train$cnt, DT_Predict_train,

    xlab = 'Actual values',

    ylab = 'Predicted values',

    main = 'DT model')


#evaluation

postResample(DT_Predict_train, train$cnt)

mape(train$cnt, DT_Predict_train)


#predictions on test

DT_Predict_test = predict(dt_model, test[,-9])

plot(test$cnt, DT_Predict_test,

    xlab = 'Actual values',

    ylab = 'Predicted values',

    main = 'DT model')


#evaluation

postResample(DT_Predict_test, test$cnt)

mape(test$cnt, DT_Predict_test)
```

```
# ##############################################
# # # # ??? Random Forest
# # ##########################################
#
##Random forest
#library(randomForest)
#library(inTrees)
set.seed(101)
#model
rf_model = randomForest(cnt ~. , train, importance = TRUE, ntree = 500)
rf_model

#error plotting
plot(rf_model)

#Variable Importance plot
varImpPlot(rf_model)

#Plotting predict train data using RF model
RF_predict_train = predict(rf_model, train[,-9])
plot(train$cnt, RF_predict_train,
    xlab = 'Actual values',
    ylab = 'Predicted values',
    main = 'RF model')
```

```
#Train Result
postResample(RF_predict_train, train$cnt)#R-sq = 0.89
mape(train$cnt, RF_predict_train)


#Plotting predict test data using RF model
RF_predict_test = predict(rf_model, test[,-9])
plot(test$cnt, RF_predict_test,
    xlab = 'Actual values',
    ylab = 'Predicted values',
    main = 'RF model')

#Test Result
postResample(RF_predict_test, test$cnt)#R-sq = 0.89
mape(test$cnt, RF_predict_test)


# ###########################################
# # # # ??? Support Vector Regression
# # #########################################
#
##SVM
#library(e1071)
set.seed(121)
#model
SVM_model = svm(cnt ~., train)
```

```
#predictions on train

SVM_predict = predict(SVM_model, train[,-9])

plot(train$cnt, SVM_predict, xlab = 'Actual values', ylab = 'Predicted values', main = 'SVR model')


#evaluation

postResample(SVM_predict, train$cnt)

mape(train$cnt, SVM_predict)


#predictions on test

SVM_predict = predict(SVM_model, test[,-9])

plot(test$cnt, SVM_predict, xlab = 'Actual values', ylab = 'Predicted values', main = 'SVR model')


#evaluation

postResample(SVM_predict, test$cnt)

mape(test$cnt, SVM_predict)


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


#K-fold cross validation function

kfold_train <- function(model){

  x=trainControl(method = "cv",number = 10)

  model= train(cnt ~.,data=train,metric="RMSE",method=model,trControl=x)

  print(model)

  return(model)

}

result <- function(model){
```

```
  model = model
  set.seed(101)
  pred = predict(model, train[,-9])
  print(postResample(pred, train$cnt))
  print(mape(train$cnt, pred))

  print('Test Results____')
  pred = predict(model,test[,-9])
  print(postResample(pred, test$cnt))
  print(mape(test$cnt, pred))
}

################################################## We have commented KFOLD validation code
and Hyper parameter tuning as it takes a lot time ##############
# ###########################
# #CV-Fold check
# ###########################
# library(doSNOW)
# cl <- makeCluster(10) #clustering approach using doSNOW pkg
# registerDoSNOW(cl)

#Random Forest # R2 = 87
# forest = kfold_train('rf')
# result(forest)
#
# stopCluster(cl)
#
# #Linear Regression # R2 = 83
```

```
# lm_model = kfold_train('lm')

# result(lm_model)

#

# #Decision Tree # R2 = 60

# dtree = kfold_train('rpart')

# result(dtree)

# #

# #SVR # R2 = 86

# svr_model = kfold_train('svmPoly')

# result(svr_model)

# #

# # # stopCluster(cl)




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

# #          Knowing the right hyper parameters tuning

# # As this process will take a bit time so here i have commented the code

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


#Using doSNOW lib for segmenting the clustering onto task as a faster approch

# library(doSNOW)


# # #Best mtry  ======   found best as = 4

# cl <- makeCluster(6) #clustering approach using doSNOW pkg

# registerDoSNOW(cl)

#
```

```
# trControl <- trainControl(method = "cv",number = 10,search = "grid")
# set.seed(101)
# tuneGrid <- expand.grid(.mtry = c(2:8))
# rf_mtry <- train(cnt~.,data = train,method = "rf",metric = "RMSE",
#              tuneGrid = tuneGrid,trControl = trControl,importance = TRUE,ntree = 800)
# best_mtry <- rf_mtry$bestTune$mtry
# print(best_mtry)


# # #Looking for best ntree  ====  found best as = 500
# store_maxtrees <- list()
# tuneGrid <- expand.grid(.mtry = best_mtry)
# for (ntree in c(200, 300, 350, 400, 450, 500, 550, 600, 700,800, 1000)) {
#   set.seed(101)
#   rf_maxtrees <- train(cnt~.,data = train,method = "rf",metric = "RMSE",tuneGrid = tuneGrid,
#                  trControl = trControl,importance = TRUE,ntree = ntree)
#   key <- toString(ntree)
#   store_maxtrees[[key]] <- rf_maxtrees
# }
# results_tree <- resamples(store_maxtrees)
# summary(results_tree)
#
# stopCluster(cl)




# ################################################################
# ## Final Model Random Forest
```

```
# ##############################################################
#
final_model = randomForest(cnt ~. , train, importance = TRUE, ntree = 500)
final_model

#error plotting
plot(final_model)

#Variable Importance plot
varImpPlot(final_model)

#Plotting predict train data using RF model
Final_predict_train = predict(final_model, train[,-9])
plot(train$cnt, Final_predict_train,
    xlab = 'Actual values',
    ylab = 'Predicted values',
    main = 'RF model')

#Train Result
postResample(Final_predict_train, train$cnt)#R-sq = 0.89
mape(train$cnt, Final_predict_train)



#Plotting predict test data using RF model
Final_predict_test = predict(final_model, test[,-9])
plot(test$cnt, Final_predict_test,
    xlab = 'Actual values',
```

```
    ylab = 'Predicted values',
    main = 'RF model')

#Test Result
postResample(Final_predict_test, test$cnt)#R-sq = 0.89
mape(test$cnt, Final_predict_test)

#############################################################################
## Saving the output
rmExcept(c("final_model",'mape',"original_data",'df'))
df$predict_cnt <-  round(predict(final_model, df[,-9]))
original_data$predict_cnt <- df$predict_cnt
write.csv(original_data, '../Data/output/R_Output/Entire_output_R.csv',row.names = F)
write.csv(original_data[,c("dteday","weathersit","season","mnth",'temp',"hum","windspeed","cnt","
predict_cnt")], '../Data/output/R_Output/Seasonal_output_R.csv',row.names = F)
```