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

Sample Output with Sample Input Result in modeling may vary becasue We run this code mulitiple times and get different Error metrics values.

In [1]:
```
#before starting this project with dataset. Initially I open given file in exc
el and just look the data available

#starting the project we need to load some libraries to deal with this data

import pandas as pd #for data processing & I/O operations
import numpy as np #for mathematical calculations
import seaborn as sns #for data visualization
import matplotlib.pyplot as plt #for plotting graphs

import sklearn #for machine learning algorithms

import os #for setting directory, I/O file operations
```

In [2]:
```
#setting working directories
os.chdir("D:\Bike Renting")
```

In [3]:
```
#checking the file directory
os.getcwd()
```

Out[3]: 'D:\\Bike Renting'

In [4]:
```
#load required dataset
data = pd.read_csv("day.csv")
```

Understanding the GIven Data

In [5]:
```
#after loading dataset
#let's check the number of variables and observations in dataset
data.shape
```

Out[5]: (731, 16)

after performning above function We see that therer are **731 observations(Rows) and 16 variables(Columns)** in given dataset

In this **cnt** is our Target Variable and the others are predictor variables

In [6]:
```
#now explore dataset more
#checking few dataset
data.head()
```

Out[6]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	at
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.36
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.35
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.18
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.21
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.22

After checkingg few records of dataset, we get little bit confused about some of the variables values.

But don't worry in our problem statement, we get clarification about these variables and their values

Important Note about Variables:

- instant: record index
- dteday: date
- season: season (1:spring, 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 from
- 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, Party cloudy, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: Normalized temperature in Celsius. The values are divided 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 divided 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

this shows that most of the variables are already converted as Categorical Variables using Normalization methods.

In [7]:
```
#before moving further let's check the dtypes of each variables
data.dtypes
```

Out[7]:
```
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
casual          int64
registered      int64
cnt             int64
dtype: object
```

As we know, most of the variables are categorical, and season, yr, mnth, holiday, weekday, workingday, weathersit variables should be a categorical type, but they are int64. Now we need to convert them into catergorical variables

In [8]:
```
#converting to categorical variables

#so to convert muliple variables in one go, we need to create a loop function

#create a variable and store all variables
cat_var = ["season", "yr", "mnth", "holiday", "weekday", "workingday", "weathe
rsit"]

for i in cat_var:
    data[i] = data[i].astype('category')
```

In [9]:
```
#checking dtypes again
data.dtypes
```

Out[9]:
```
instant          int64
dteday          object
season        category
yr            category
mnth          category
holiday       category
weekday       category
workingday    category
weathersit    category
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

In this dataset, some of the varaibles are not useful for further analysis for that reason we are dropping some of the variables here:

Dropping Variables which are not required:

- instant - index number, which is not useful in analysis
- dteday - all the required parameters are already extracted from this variable such as year, month, weekday. So this varaible is not useful

In [10]:
```
#dropping instant and dteday variables
data = data.drop(['instant', 'dteday'], axis = 1)
```

In [11]:
```
#checking the dataset after dropping two variables
data.head()
```

Out[11]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum
0	1	0	1	0	6	0	2	0.344167	0.363625	0.805833
1	1	0	1	0	0	0	2	0.363478	0.353739	0.696087
2	1	0	1	0	1	1	1	0.196364	0.189405	0.437273
3	1	0	1	0	2	1	1	0.200000	0.212122	0.590435
4	1	0	1	0	3	1	1	0.226957	0.229270	0.436957

Missing Value Analysis

After converting dataset into proper format and dropping unuseful variables from dataset. Now its time to do Missing Value analysis

In [12]:
```
#checking the missing values using isnull function

data.isnull().sum().sort_values(ascending= False)
```

Out[12]:
```
cnt           0
registered    0
casual        0
windspeed     0
hum           0
atemp         0
temp          0
weathersit    0
workingday    0
weekday       0
holiday       0
mnth          0
yr            0
season        0
dtype: int64
```

There is no missing value present in given dataset

Outlier Analysis

In [13]:
```
#check summary of the dataset
data.describe()
```

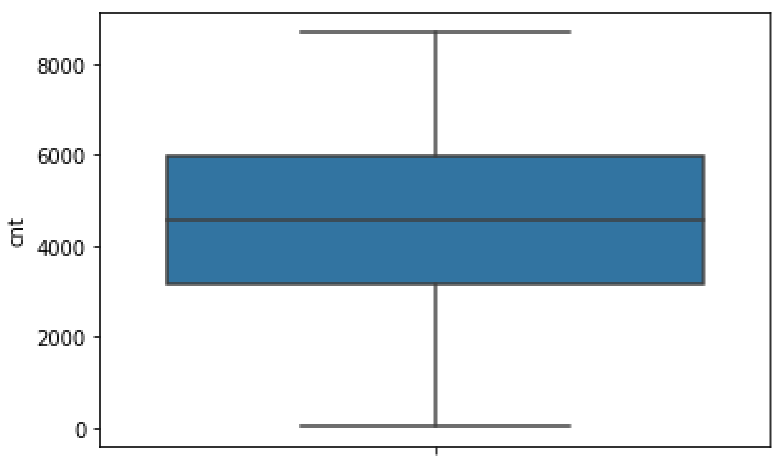
Out[13]:

	temp	atemp	hum	windspeed	casual	registered	cnt
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
std	0.183051	0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
min	0.059130	0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
25%	0.337083	0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
50%	0.498333	0.486733	0.626667	0.180975	713.000000	3662.000000	4548.000000
75%	0.655417	0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
max	0.861667	0.840896	0.972500	0.507463	3410.000000	6946.000000	8714.000000

Here, we use the **boxplot method** to visualize the outliers in our dataset

In [14]:
```
#checking outliers in Target Variable "cnt" using boxplot method

#plotting boxplot for cnt variable
sns.boxplot(data =data, y="cnt", orient ='v')
```

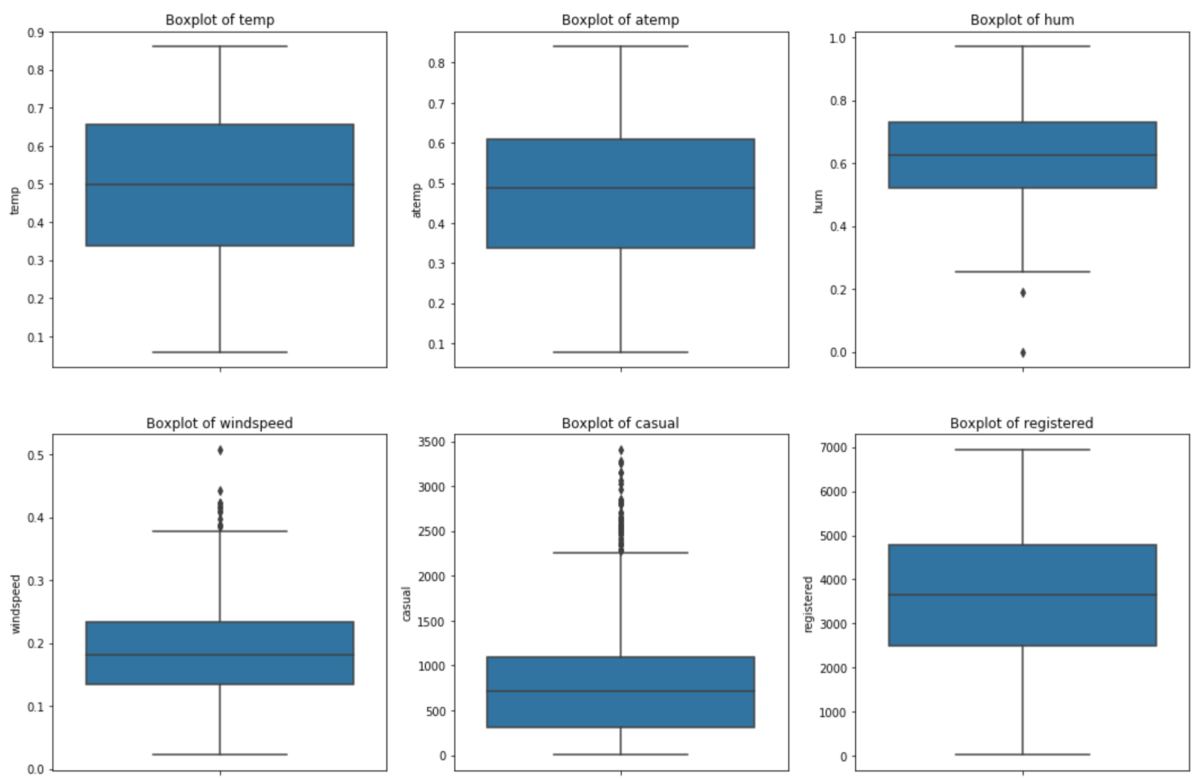
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1a8416dd828>



From the above boxplots, it is evident that there is no outliers present in cnt.

In [15]:
```
#let's check the outliers of predictor variables such as temp, atemp, hum, win
dspeed, casual, registered

fig, axes = plt.subplots(nrows=2, ncols= 3)

fig.set_size_inches(18,12)

#plotting boxplot of temp variable
sns.boxplot(data['temp'], orient ='v', ax=axes[0][0]).set_title("Boxplot of te
mp")

#plotting boxplot of atemp variable
sns.boxplot(data['atemp'], orient ='v', ax=axes[0][1]).set_title("Boxplot of a
temp")

#plotting boxplot of hum variable
sns.boxplot(data['hum'], orient ='v', ax=axes[0][2]).set_title("Boxplot of hu
m")

#plotting boxplot of windspeed variable
sns.boxplot(data['windspeed'], orient ='v', ax=axes[1][0]).set_title("Boxplot
 of windspeed")

#plotting boxplot of casual variable
sns.boxplot(data['casual'], orient ='v', ax=axes[1][1]).set_title("Boxplot of
 casual")

#plotting boxplot of registered variable
sns.boxplot(data['registered'], orient ='v', ax=axes[1][2]).set_title("Boxplot
of registered")
```

Out[15]: Text(0.5, 1.0, 'Boxplot of registered')

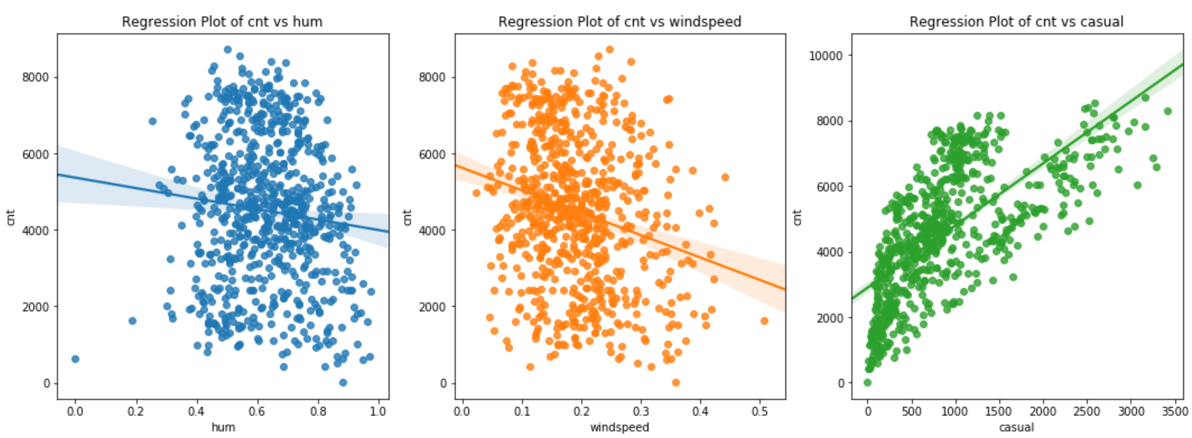


In [16]:
```
#as we see that there are some outliers value present in 'hum', 'windspeed',
 'casual' variables.

#before outlier removal lets findout the correlation analysis of these variabl
es with target variables

print(data['hum'].corr(data['cnt']))

print(data['windspeed'].corr(data['cnt']))

print(data['casual'].corr(data['cnt']))
```

```
-0.10065856213715531
-0.23454499742167
0.6728044333386831
```

In [17]:
```
fig, (ax1,ax2,ax3) = plt.subplots(ncols= 3)

fig.set_size_inches(18,6)

#Correlation between 'hum' and 'cnt'  before  removal of  outliers
sns.regplot(x="hum", y="cnt", data=data, ax=ax1).set_title("Regression Plot of
cnt vs hum")

#Correlation between 'windspeed' and 'cnt'  before  removal of  outliers
sns.regplot(x="windspeed", y="cnt", data=data, ax=ax2).set_title("Regression P
lot of cnt vs windspeed")

#Correlation between 'casual' and 'cnt'  before  removal of  outliers
sns.regplot(x="casual", y="cnt", data=data, ax=ax3).set_title("Regression Plot
of cnt vs casual")
```

Out[17]:
```
Text(0.5, 1.0, 'Regression Plot of cnt vs casual')
```



Outliers: As we see from boxplot, correlation and regression plot, varaibles **hum, windspeed, casual** has ouliers and that have to be remove by outlier removal method

In [18]:
```
#make copy of dataset
df = data.copy()
```

In [19]:
```
#Detect & Delete Outliers from the dataset

cnames = ['casual','hum','windspeed']

for i in cnames:
    q75, q25 = np.percentile(data.loc[:,i],[75,25])
    iqr = q75 - q25
    min = q25-(iqr*1.5)
    max = q75 +(iqr*1.5)
    print(min)
    print(max)
    data = data.drop(data[data.loc[:,i]< min].index)
    data = data.drop(data[data.loc[:,i]>max].index)
```

```
-855.25
2266.75
0.19999974999999992
1.0533337500000002
-0.012456500000000065
0.38062750000000006
```

In [20]:
```
#check shape of dataset
data.shape #58 obseravtions are dropped in outliers
```

Out[20]: (673, 14)

Correlation Analysis

Correlation Analysis: Here I am generating correlation matrix to understand how the each variable related with each other. In that I am plotting correlation matrix and generate plot using seabon library for better understanding

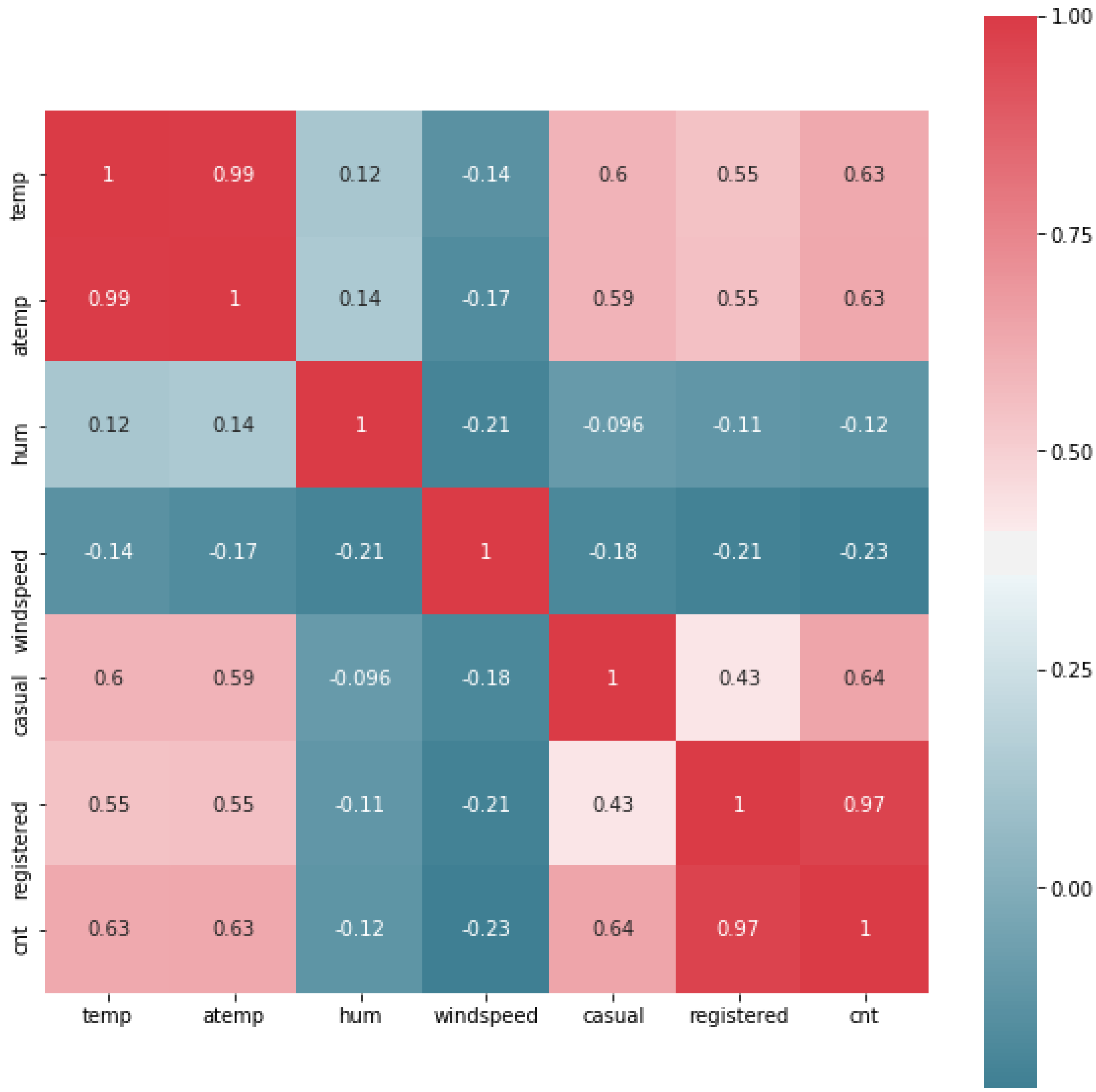
In [21]:
```
#generating correlation matrix
corr = data.corr()
corr
```

Out[21]:

	temp	atemp	hum	windspeed	casual	registered	cnt
temp	1.000000	0.991483	0.122486	-0.139599	0.595525	0.545120	0.629031
atemp	0.991483	1.000000	0.135356	-0.167087	0.593962	0.547850	0.630906
hum	0.122486	0.135356	1.000000	-0.206719	-0.096350	-0.113078	-0.122854
windspeed	-0.139599	-0.167087	-0.206719	1.000000	-0.184026	-0.212375	-0.231596
casual	0.595525	0.593962	-0.096350	-0.184026	1.000000	0.427474	0.642890
registered	0.545120	0.547850	-0.113078	-0.212375	0.427474	1.000000	0.967266
cnt	0.629031	0.630906	-0.122854	-0.231596	0.642890	0.967266	1.000000

In [22]:
```
#plotting correlation matrix and heatmap using seaborn libraty
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(corr,mask=np.zeros_like(corr, dtype=np.bool),cmap = sns.diverging_
palette(220,10,as_cmap=True),square =True, annot=True, ax=ax)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1a841eb6780>



Correlation Analysis Result

- temp and atemp are highly correlated
- temp and atemp have positive and strong correlation with cnt
- hum and windspeed have negative and weak correlation with cnt

In [23]:
```
#dropping atemp variable from a dataset
data = data.drop(['atemp'], axis = 1)
```

In [24]:
```
data.head(5)
```

Out[24]:

	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	hum	windspeed
0	1	0	1	0	6	0	2	0.344167	0.805833	0.160446
1	1	0	1	0	0	0	2	0.363478	0.696087	0.248539
2	1	0	1	0	1	1	1	0.196364	0.437273	0.248309
3	1	0	1	0	2	1	1	0.200000	0.590435	0.160296
4	1	0	1	0	3	1	1	0.226957	0.436957	0.186900

Exploratory Data Analysis

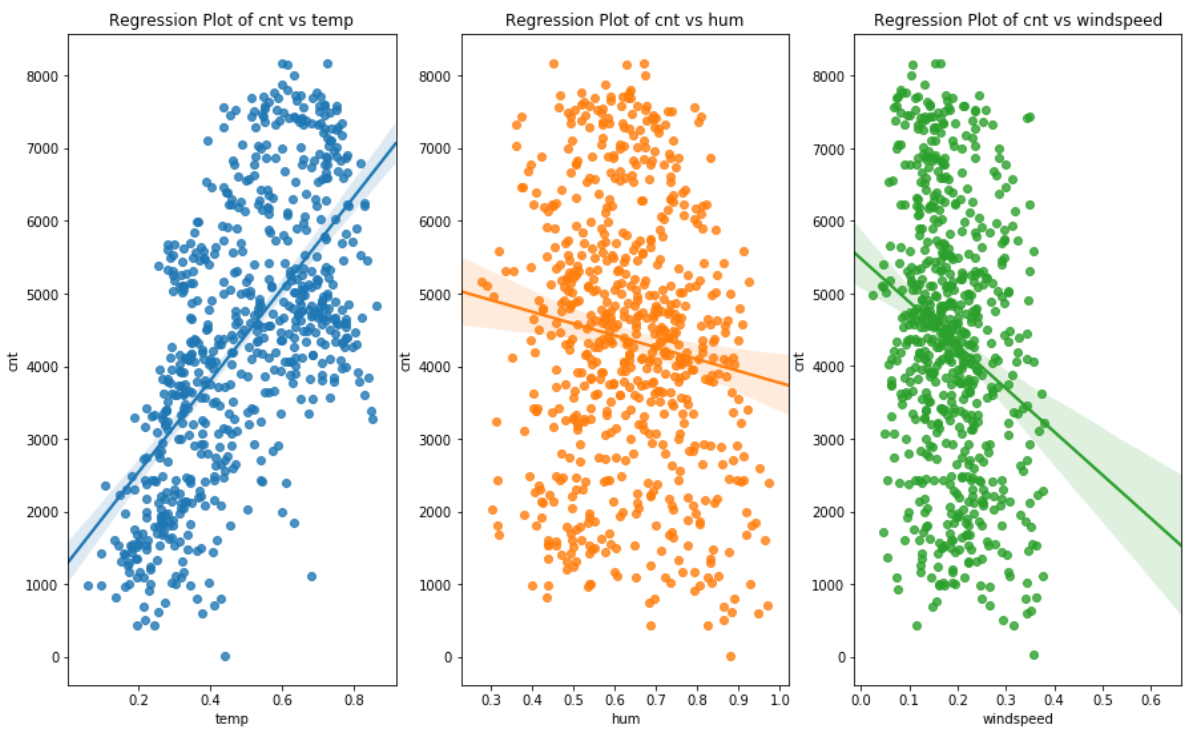
In **Exploratory Data Analysis** we are going to find the how each predictor or variables related with Target Varaible:

- relation between Numerical Variable 'temp', 'hum', 'windspeed' and target variable 'cnt'

Bivariate analysis

In [25]:
```
#finding relationship between Numerical Variable 'temp', 'hum', 'windspeed' wi
th target variable 'cnt'

fig, (ax1,ax2,ax3) = plt.subplots(ncols= 3)

fig.set_size_inches(15,9)

#relation between 'temp' and 'cnt'
sns.regplot(x="temp", y="cnt", data=data, ax=ax1).set_title("Regression Plot o
f cnt vs temp")

#relation between 'hum' and 'cnt'
sns.regplot(x="hum", y="cnt", data=data, ax=ax2).set_title("Regression Plot of
cnt vs hum")

#relation between 'windspeed' and 'cnt'
sns.regplot(x="windspeed", y="cnt", data=data, ax=ax3).set_title("Regression P
lot of cnt vs windspeed")
```

Out[25]:
```
Text(0.5, 1.0, 'Regression Plot of cnt vs windspeed')
```

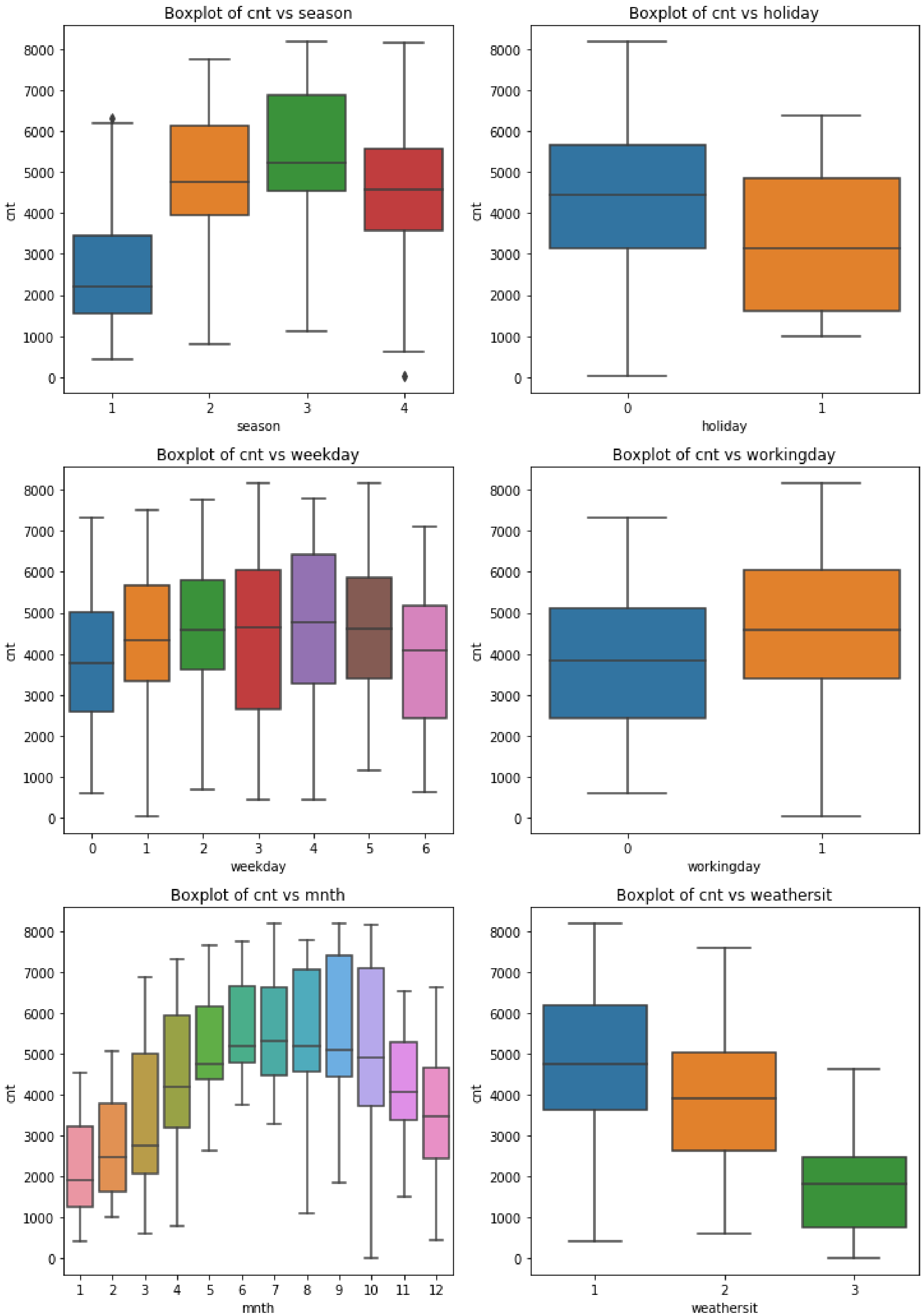


From the above plot, we see that

- cnt has positive linear relationship with temp,
- on the other side, cnt has a negative linear relationship with windspeed.
- But hum(Humidity) has a little negative linear relationship with cnt.

In [26]:
```
#now we find the relationship between categorical variables and Target Variabl
e 'cnt'

# categorical variables are "Season", "holiday", "Weekday", "Workingday", "Wea
thersit", "month"
fig, axes = plt.subplots(nrows = 3, ncols=2)

fig.set_size_inches(12,18)

#plotting boxplot for cnt vs season variables
sns.boxplot(data =data, y="cnt", x="season", orient='v', ax=axes[0][0]).set_ti
tle("Boxplot of cnt vs season")

#plotting boxplot for cnt vs holiday variables
sns.boxplot(data =data, y="cnt", x="holiday", orient='v', ax=axes[0][1]).set_t
itle("Boxplot of cnt vs holiday")

#plotting boxplot for cnt vs weekday variables
sns.boxplot(data =data, y="cnt", x="weekday", orient ='v', ax=axes[1][0]).set_
title("Boxplot of cnt vs weekday")

#plotting boxplot for cnt vs workingday variables
sns.boxplot(data=data, y="cnt", x="workingday", orient='v', ax=axes[1][1]).set
_title("Boxplot of cnt vs workingday")

#plotting boxplot for cnt vs month variables
sns.boxplot(data=data, y="cnt", x="mnth", orient='v', ax=axes[2][0]).set_title
("Boxplot of cnt vs mnth")

#plotting boxplot for cnt vs Weathershit variables
sns.boxplot(data=data, y="cnt", x="weathersit", orient='v', ax=axes[2][1]).set
_title("Boxplot of cnt vs weathersit")
```

Out[26]: Text(0.5, 1.0, 'Boxplot of cnt vs weathersit')



Results:

- **Graph 1:cnt vs season**:
  - cnt is very low in Spring Season and cnt is large in Fall Season
- **Graph 2: cnt vs holiday**
  - cnt is more on weekday i.e. no holiday
- **Gradph 3: cnt vs weekday**
  - as per the graph more number of bikes are used on Friday
- **Gradph 4: cnt vs workingday**
  - as per the graph more number of bikes are used on Workingday, this conclusion we already get from Graph 2
- **Gradph 5: cnt vs month**
  - as per the graph more number of bikes are used in July Month of year
- **Gradph 6: cnt vs Weathersit**
  - as per the graph more number of bikes are used when weather condition is Clear, Few clouds, Party cloudy, Partly cloudy
  - and less number of bikes are used when weather condition is Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - No bikes where used when weather condition is Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

Summary:

- cnt is maximum in good weather condition and minimum in bad weather condition
- more number of bikes are rented on weekdays.

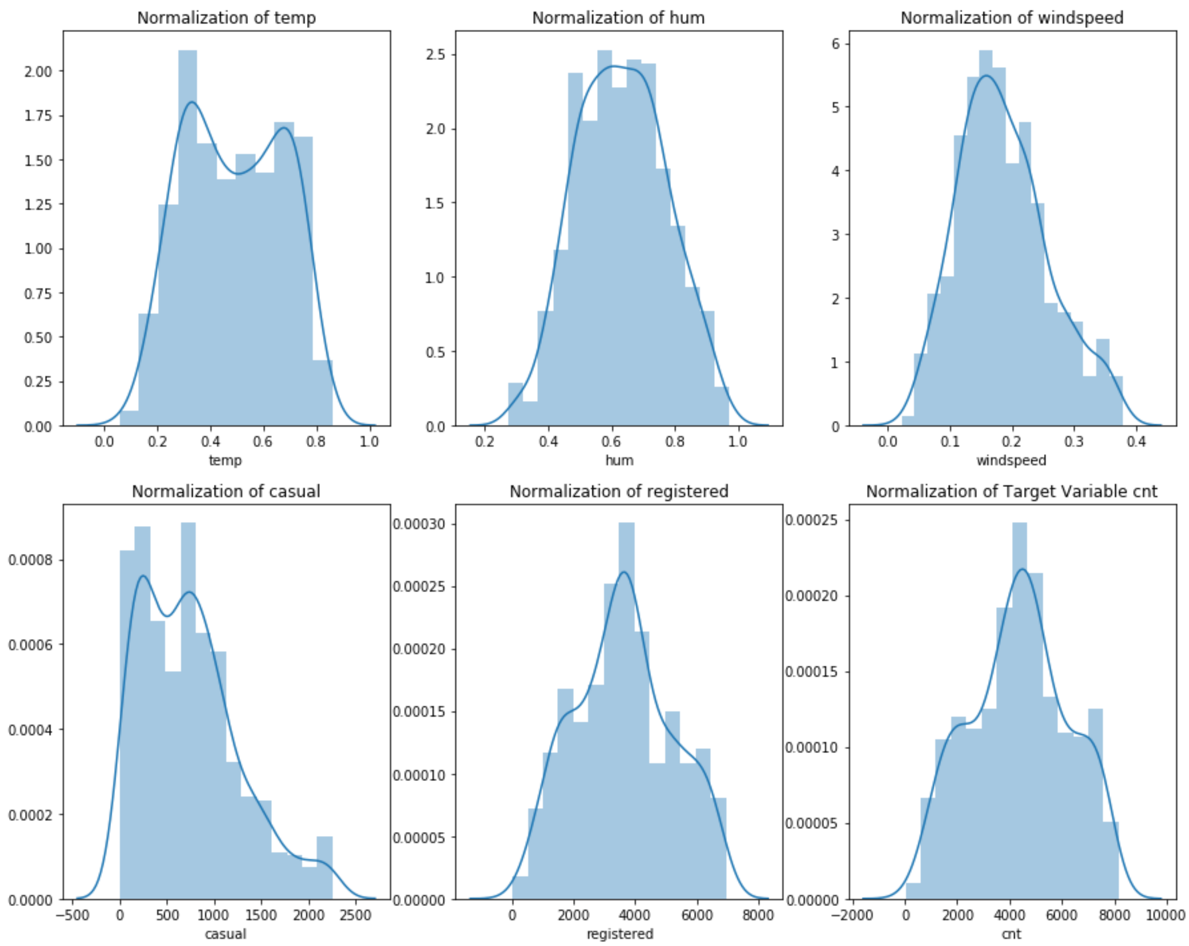
Feature Scalling

as we know that, most of given variables are already Normalised

But let's clarify that using Visualization techniques

In [27]:
```
fig, axes = plt.subplots(nrows = 2, ncols = 3)

fig.set_size_inches(15,12)

#Check whether  variable 'temp'is normal or not
sns.distplot(data['temp'], ax=axes[0][0]).set_title("Normalization of temp")

#Check whether  variable 'hum'is normal or not
sns.distplot(data['hum'], ax=axes[0][1]).set_title("Normalization of hum")

#Check whether  variable 'windspeed'is normal or not
sns.distplot(data['windspeed'], ax=axes[0][2]).set_title("Normalization of win
dspeed")

#Check whether  variable 'casual'is normal or not
sns.distplot(data['casual'],ax=axes[1][0]).set_title("Normalization of casual"
)

#Check whether  variable 'registered'is normal or not
sns.distplot(data['registered'], ax=axes[1][1]).set_title("Normalization of re
gistered")

#as well as Check whether  Target variable 'cnt'is normal or not
sns.distplot(data['cnt'], ax=axes[1][2]).set_title("Normalization of Target Va
riable cnt")
```

Out[27]:
```
Text(0.5, 1.0, 'Normalization of Target Variable cnt')
```



As we can see, our data is in proper scalling form. Numerical Predictor Variables are Normalized

Also our Target Variable 'cnt' is also close to Normal Distribution

In [28]:
```
data.shape
```

Out[28]:
```
(673, 13)
```

Modeling & Prediction

Data Cleaning is done! Now lets bulid the model and predict the results

In machine Learning there is Two main types:

- **Supervised Machine Learning :** knowledge of output. Target Variable is fix
- **Unsupervised Machine Learning:** No knowledge of Output. Self Guided Learnig Algorithms.

Selecting model is main Part of Modelling, We have various model algorithms some of the basic algorithms are:

- **Linear Regression :** Best suitable for Regression Model
- **Logistic Regression:** Suitable for Classification Model
- **Decision Tree:** Best suitable for Regression & Classification model
- **Random Forest:** Mostly used for Classification model analysis but can be use for Regression model
- **KNN algorithms:** Can be used for Regression and Classification model
- **Naive Bayes:** used for Classification Model

In our given dataset, the Target Variable 'cnt' is Numerical Continuous Variable. So we are dealing with **Regression Model** Analysis.

for that reason, we are considering following Algorithms:

- Linear Regression
- Decision Tree
- Random Forest
- KNN Algorithms

In [29]:
```
# before moving further
#let's droped the casual and registered variables because there sum is equal t
o target variable ie. 'cnt'

data = data.drop(['casual', 'registered'], axis =1)
```

In [30]:
```
data.shape
```

Out[30]:
```
(673, 11)
```

In [31]:
```
#now let's define the feature matrix and response vector
X = data.drop('cnt', axis=1)
y = data.iloc[:,-1].values
```

In [32]:
```
#splitting X and y into training and testing dataset
#import sklearn train_test_split library
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y =train_test_split(X,y,test_size=0.2)
```

Linear Regression Model

In [33]:
```
#train the model using traing dataset
#import LinearRegression libraries from sklearn
from sklearn import linear_model
import statsmodels.api as sm
```

In [34]:
```
#train the model using training dataset
model_LR = sm.OLS(train_y, train_X.astype(float)).fit()
```

In [35]:
```
model_LR.summary()
```

Out[35]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.968
Model:	OLS	**Adj. R-squared:**	0.968
Method:	Least Squares	**F-statistic:**	1611.
Date:	Tue, 03 Sep 2019	**Prob (F-statistic):**	0.00
Time:	15:59:54	**Log-Likelihood:**	-4390.9
No. Observations:	538	**AIC:**	8802.
Df Residuals:	528	**BIC:**	8845.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
season	513.7337	63.584	8.080	0.000	388.825	638.642
yr	2035.8414	73.376	27.745	0.000	1891.697	2179.986
mnth	-27.1461	20.113	-1.350	0.178	-66.658	12.366
holiday	-193.4227	233.126	-0.830	0.407	-651.390	264.545
weekday	89.2967	18.629	4.793	0.000	52.701	125.892
workingday	468.7285	85.955	5.453	0.000	299.872	637.585
weathersit	-767.6094	91.934	-8.350	0.000	-948.211	-587.008
temp	5102.2030	219.849	23.208	0.000	4670.317	5534.089
hum	706.7866	301.001	2.348	0.019	115.481	1298.092
windspeed	-1156.1858	457.684	-2.526	0.012	-2055.290	-257.081

Omnibus:	83.730	**Durbin-Watson:**	1.912
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	157.694
Skew:	-0.900	**Prob(JB):**	5.72e-35
Kurtosis:	4.947	**Cond. No.**	107.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

A few things, we learn from this output

- season, yr, weekday, workingday, weathersit, temp have small p-values, where as mnth, holiday, hum, windspeed have a larger p-values
- Here we reject the null-hypothesis for season, yr, weekday, workingday, weathersit, temp

```
* There is assicoation between these variables and Target Variable cnt
```

- Fail to reject the null hypothesis for mnth, holiday, hum, windspeed
  - There is no association between these variables and Target Variable cnt

R-squared (0.967) means this model provides best fit for the given data

but Selecting the model with the highest R-squared is not a reliable approach for choosing the best linear model.

Solution:

- **Adjusted R-squared**

  Penalizes model complexity (to control for overfitting), but it generally under-penalizes complexity.

- **Better Solution:**

Do model Evaluation based on the Error Metrics for Regression:

For classification problems, we have only used classification accuracy as our evaluation metric. But here we used Error Metrics to evaluate the model

Mean Absolute Error (MAE): is the mean of the absolute value of the errors: In $[0,\infty)$, the smaller the better

Mean Squared Error (MSE): is the mean of the squared errors: In $[0,\infty)$, the smaller the better

Mean Absolute Percent Error (MAPE): is the mean of the absolute percent value of the errors: In $[0,1)$, the smaller the better

Root Mean Squared Error (RMSE) :is the square root of the mean of the squared errors: In $[0,\infty)$, the smaller the better

Let's calculate these by hand, to get an intuitive sense for the results:

In [36]:
```
#make the predictions by model
predict_LR = model_LR.predict(test_X)
```

In [37]:
```
def MAPE(true_y, pred_y):
    mape = np.mean(np.abs(true_y-pred_y)/true_y)
    return mape
```

In [38]:
```
#importing important error metrics libraries
from sklearn.metrics import mean_absolute_error, mean_squared_error
# calculate MAE, MSE, MAPE, RMSE
print("MAE:",mean_absolute_error(test_y, predict_LR))
print("MSE:",mean_squared_error(test_y, predict_LR))
print("MAPE:",MAPE(test_y,predict_LR))
print("RMSE:",np.sqrt(mean_squared_error(test_y, predict_LR)))
```

```
MAE: 640.4105026069809
MSE: 745705.0469316511
MAPE: 0.19485534370012916
RMSE: 863.5421512188337
```

- MAE gives less weight to outliers means it is not sensitive to outliers.
- MAPE is similar to MAE, but normalized the true obeservations. When true observation is zero then this metric will be problematic
- MSE is a combination measurement of bias and variance of predictions. It is more popular.
- RSME is square Root of MSE, Root square is taken to make the units of the error be the same as the units of the target. This measure gives more weight to large deviations such as outliers, since large differences squared become larger and small (smaller than 1) differences squared become smaller.

Selection: Outoff these 4 error metrics, MSE and RMSE are mainly used for Time-Series dataset. As we know, current working data is not a time dependend or time-series data.

for that Reason the Model Evaluation is based on **MAPE Error Metrics**

Decision Tree Regression Model

In [39]:
```
#decision tree regression
#import DecisionTreeRegressor Analysis

from sklearn.tree import DecisionTreeRegressor
```

In [40]:
```
model_DT = DecisionTreeRegressor(max_depth = 2).fit(train_X,train_y)
```

In [41]:
```
#apply the model on test data
predict_DT = model_DT.predict(test_X)
```

In [42]:
```
# calculate MAE, MSE, MAPE, RMSE
print("MAE:",mean_absolute_error(test_y, predict_DT))
print("MSE:",mean_squared_error(test_y, predict_DT))
print("MAPE:",MAPE(test_y,predict_DT))
print("RMSE:",np.sqrt(mean_squared_error(test_y, predict_DT)))
```

```
MAE: 758.7652122867178
MSE: 970994.7824656902
MAPE: 0.25892890404909763
RMSE: 985.3906750450251
```

Random Forest Regression Model

In [43]:
```
#Random forest analysis
#imnport RandomForestRegressor

from sklearn.ensemble import RandomForestRegressor
```

In [44]:
```
##create Random Forest object
model_RF = RandomForestRegressor(n_estimators = 50)

##train the model using training dataset
model_RF.fit(train_X, train_y)
```

Out[44]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

In [45]:
```
#make the predictions by model
predict_RF = model_RF.predict(test_X)
```

In [46]:
```
# calculate MAE, MSE, MAPE, RMSE
print("MAE:",mean_absolute_error(test_y, predict_RF))
print("MSE:",mean_squared_error(test_y, predict_RF))
print("MAPE:",MAPE(test_y,predict_RF))
print("RMSE:",np.sqrt(mean_squared_error(test_y, predict_RF)))
```

```
MAE: 410.77644444444445
MSE: 390411.3732888889
MAPE: 0.15098572520129147
RMSE: 624.8290752588974
```

KNN Regression Algorithms

In [47]:
```
#KNN implementation
from sklearn.neighbors import KNeighborsRegressor
```

In [48]:
```
model_KNN = KNeighborsRegressor(n_neighbors =5).fit(train_X, train_y)
```

In [49]:
```
predict_KNN = model_KNN.predict(test_X)
```

In [50]:
```
# calculate MAE, MSE, MAPE, RMSE
print("MAE:",mean_absolute_error(test_y, predict_KNN))
print("MSE:",mean_squared_error(test_y, predict_KNN))
print("MAPE:",MAPE(test_y,predict_KNN))
print("RMSE:",np.sqrt(mean_squared_error(test_y, predict_KNN)))
```

```
MAE: 598.6977777777779
MSE: 672366.6616296296
MAPE: 0.19498794728844585
RMSE: 819.9796714734027
```

Selecting Best Fit model for Future Analysis

We are cosidering the MAPE for model evaluatiomn becasue, it calculate average absolute percent error for each time period minus actual values divided by actual values.

Reason we already discussed, let's explain again:

Selection: Outoff these 4 error metrics, MSE and RMSE are mainly used for Time-Series dataset. As I know, current working data is not a time dependend or time-series data.

Random Forest Model has smallest error metrics i.e.

- **MAPE = 0.1290541**

So, for further analysis We are selecting **Random Forest Model.**