Project Report

Bike Rental

By: Krishna.R

29 August 2019

1. INTRODUCTION

1.1 Problem Statement

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

1.2 Data

Data provided with the problem is **day.csv**.

Let's have a look at sample of dataset:

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600
6	2011-01-06	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.089565	88	1518	1606
7	2011-01-07	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
8	2011-01-08	1	0	1	0	6	0	2	0.165000	0.162254	0.535833	0.266804	68	891	959

The details of data attributes in the dataset are as follows

- instant: Record index
- dteday: Date
- season: Season (1:springer, 2:summer, 3:fall, 4:winter)
- yr: Year (0: 2011, 1:2012)
- mnth: Month (1 to 12)
- holiday: weather day is holiday or not (extracted fromHoliday Schedule)
- weekday: Day of the week
- workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit: (extracted fromFreemeteo)
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),t_min=-8, t_max=+39 (only in hourly scale)
- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
- hum: Normalized wind speed. The values are divided to 67 (max)
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

2. Steps Performed in this Project

The whole project is divided in 7 phases (and further subphases). Below are the phases defined.

- Define problem statement
- Gather the data
- Prepare data for consumption
- Perform Exploratory Data Analysis
- Modelling
- Evaluate and compare Model performances and choose the best model
- Hypertune the selected model
- Produce sample output with tuned model

2.1 Categorize Problem

The problem statement is “to predict of bike rental count on daily basis on the environmental and seasonal settings”. We have to Predict the count with is continuous features with the help of independent features. There are both continuous and categorical features. Therefore this is a Regression Problem.

2.2 Gather the Data

The data is given in day.csv file. I imported data pandas dataframe.

```
# Importing Dataset
data_original = pd.read_csv("day.csv")
# I will keep the original data in the variable "data_original"
# And i will create a copy of the data with variable "data"
data = data_original.copy()
data.head(8)
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600
5	6	2011-01-06	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.089565	88	1518	1606
6	7	2011-01-07	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
7	8	2011-01-08	1	0	1	0	6	0	2	0.165000	0.162254	0.535833	0.266804	68	891	959

<u>Checking the Dimension of the data</u>

```
# Dimension of the data
data.shape
```

```
(731, 16)
```

➔ The data has 731 observations and 16 features.

- Checking the data types of each columns

```
# Datatypes of each columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dteday        731 non-null object
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

➔ There are 11 columns with int64 datatype
➔ There are 4 columns with float64 datatype
➔ There is 1 columns with object datatype

- In column dteday there is day, month and year. But we already have month and year column so I will extract date from dteday and remove dteday column

```
# We have a dteday column with dates. We also have month and year columns, so i will extract only date from dteday
# column with the help of pandas to_datetime function and then i will remove dteday columns

# Converting dteday from object to datetime
data['dteday'] = pd.to_datetime(data['dteday'])

# Extracting date from dteday column to another column 'date'
data['date'] = data['dteday'].dt.day

# Removing dteday columns as we dont need it now
data = data.drop(columns=['dteday'])
```

- Converting columns to their data types

```
# Converting Categorical columns from numeric to Categorical
for col in categorical_col:
    data[col] = data[col].astype('category')

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
season        731 non-null category
yr            731 non-null category
mnth          731 non-null category
holiday       731 non-null category
weekday       731 non-null category
workingday    731 non-null category
weathersit    731 non-null category
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
date          731 non-null category
dtypes: category(8), float64(4), int64(4)
memory usage: 54.4 KB
```

2.3 Prepare Data

Next step is to prepare the data for EDA and Model Building. We need to check for missing values and outliers. Then we showed remove the highly correlated features and Scale the data if necessary.

2.3.1 Missing Value Analysis

Checking for missing values in each columns

```
# Checking for missing values
data.isnull().sum()
```

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

➔ There are no missing values in this dataset so there is no need to impute values

2.3.2 Outlier Analysis

Checking for outliers with the help of Boxplot

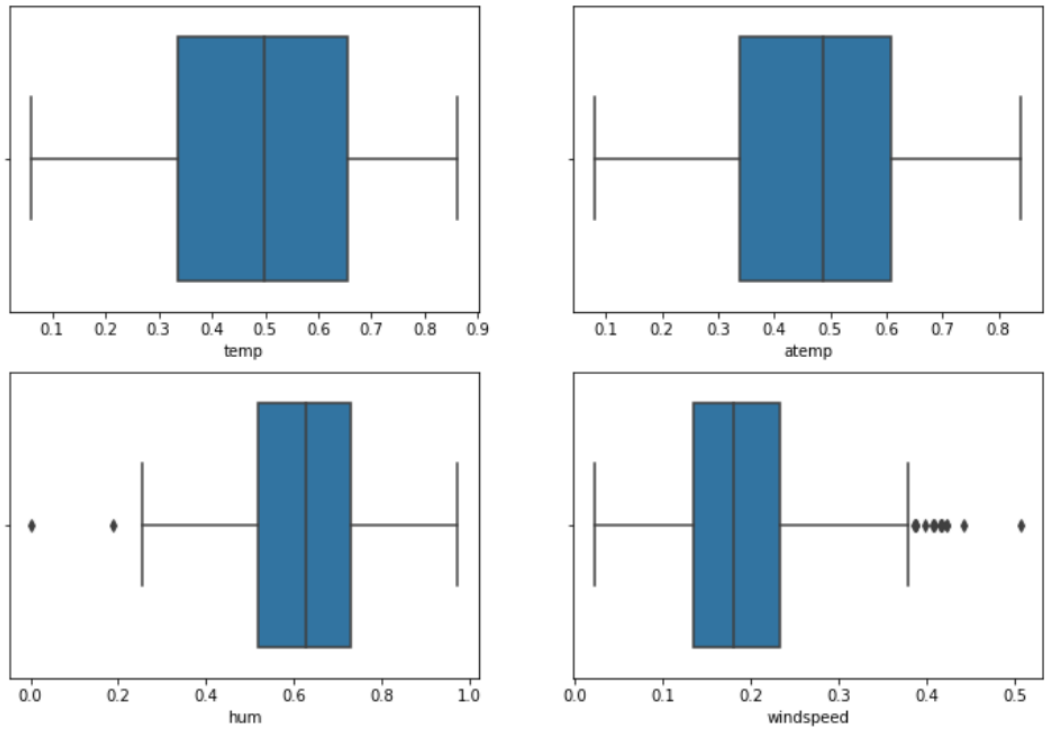
```
plt.figure(0, figsize=(12,8))

plt.subplot(221)
sns.boxplot(data["temp"])

plt.subplot(222)
sns.boxplot(data["atemp"])

plt.subplot(223)
sns.boxplot(data["hum"])

plt.subplot(224)
sns.boxplot(data["windspeed"])
```

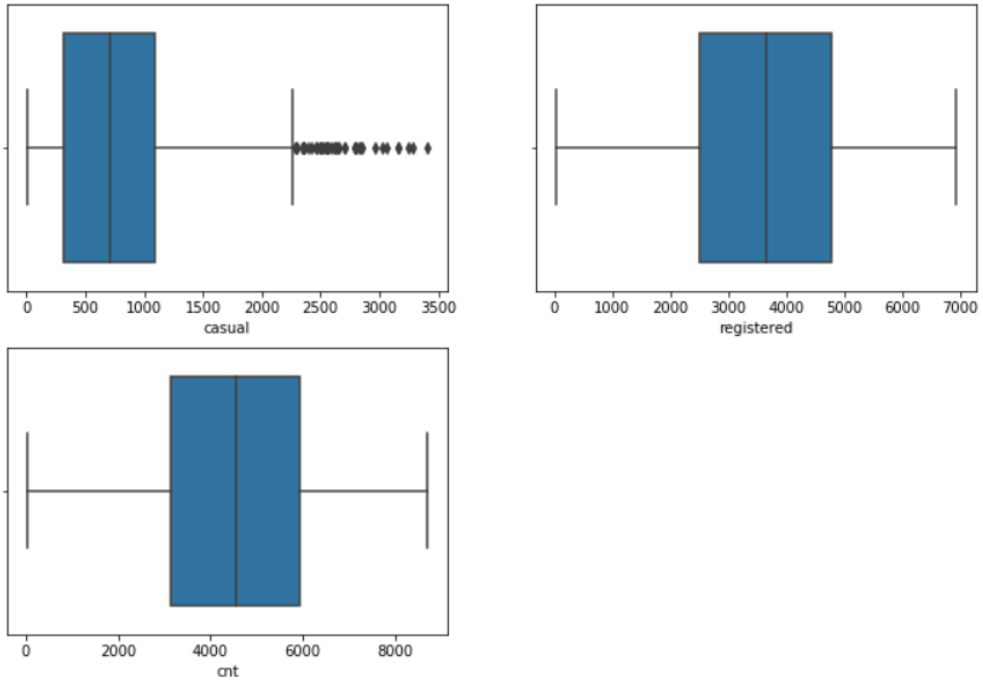


```
plt.figure(1, figsize=(12,8))

plt.subplot(221)
sns.boxplot(data["casual"])

plt.subplot(222)
sns.boxplot(data["registered"])

plt.subplot(223)
sns.boxplot(data["cnt"])
```

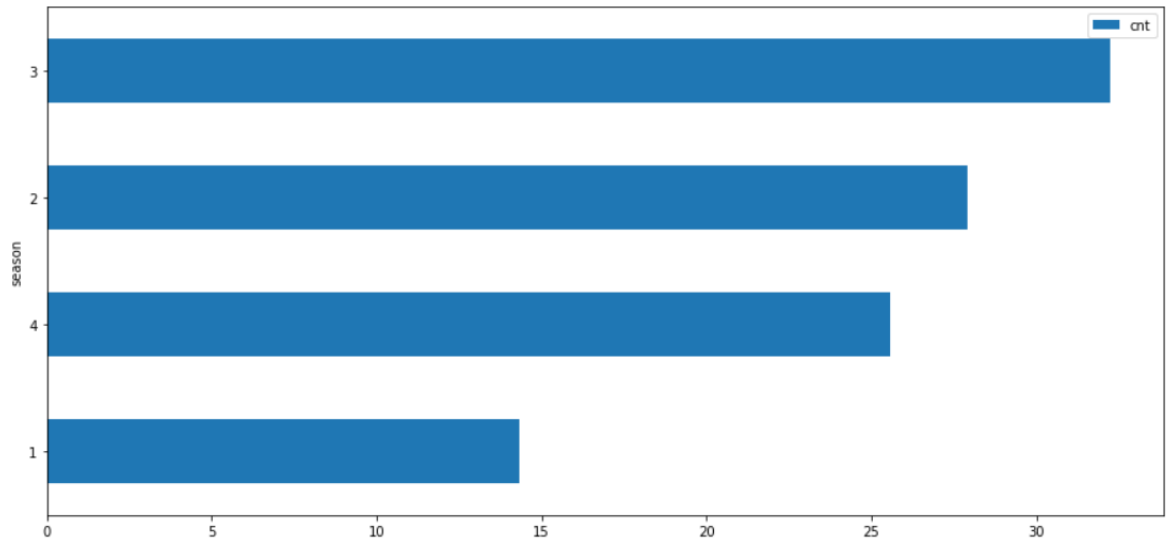


2.4 Perform EDA

Checking relation between Categorical Variable and Target Variable

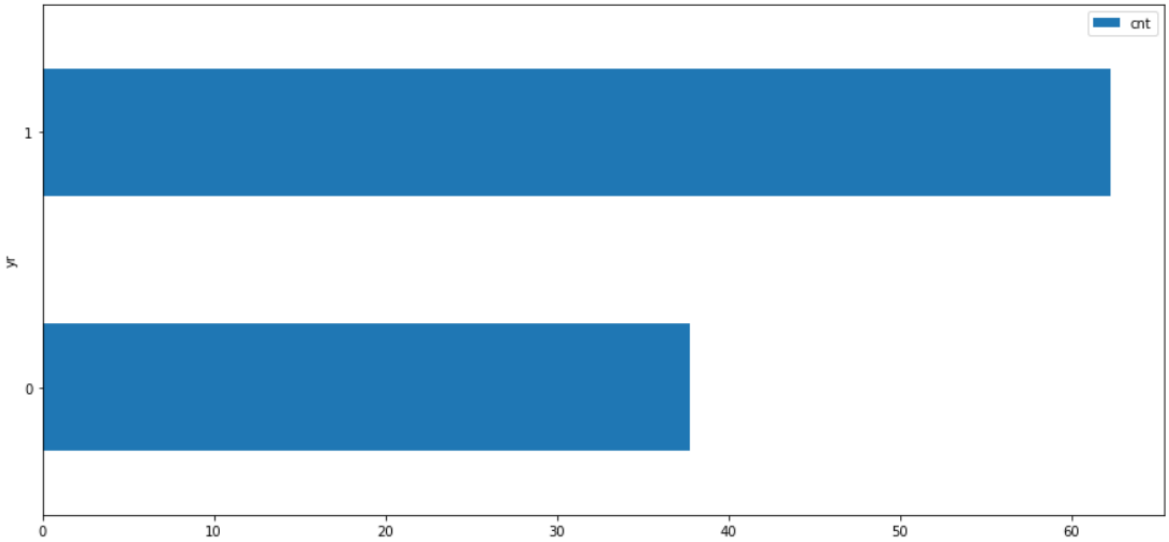
```
# Categorical Variables vs Target variable
# Creating a user defined function
n this cell lationship(a,b):
    df = data.groupby(a)[b].sum()
    df = df.reset_index()
    df[b] = df[b]*100/sum(data[b])
    df = df.sort_values(by=['cnt'])
    print(df)
    return df.plot.barh(x=a, y=b, figsize=(15,7))
```

❖ Relationship between Season and Count



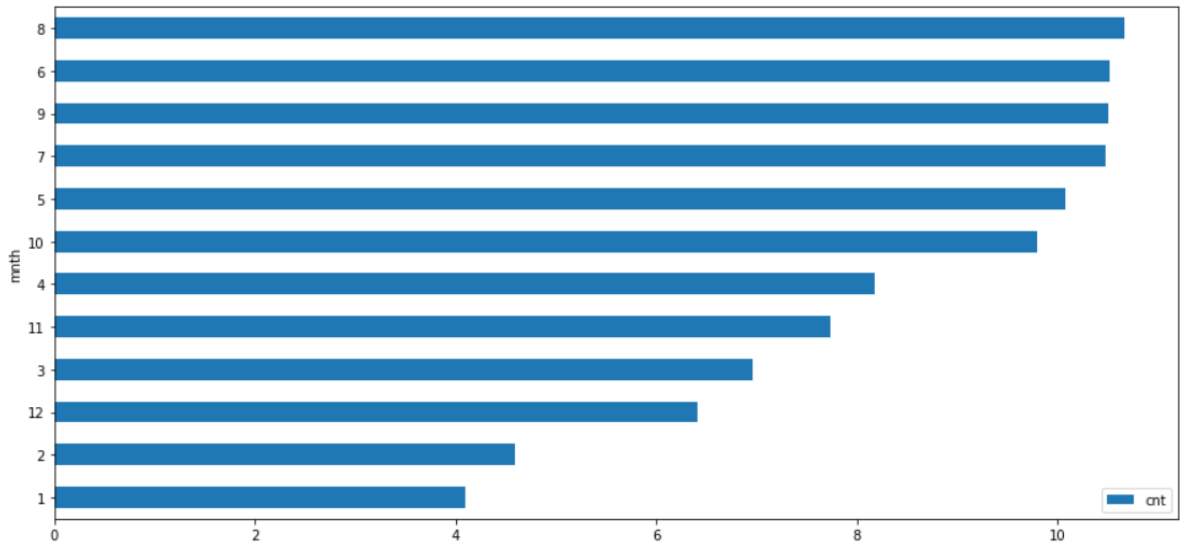
➔ Bike rental count is high on Season 3 That is during Fall
➔ Summer also has high Bike Rental count
➔ Bike rental count during winter is almost same as during in summer
➔ Bike rental count is low during Spring

❖ Relationship between Year and Count



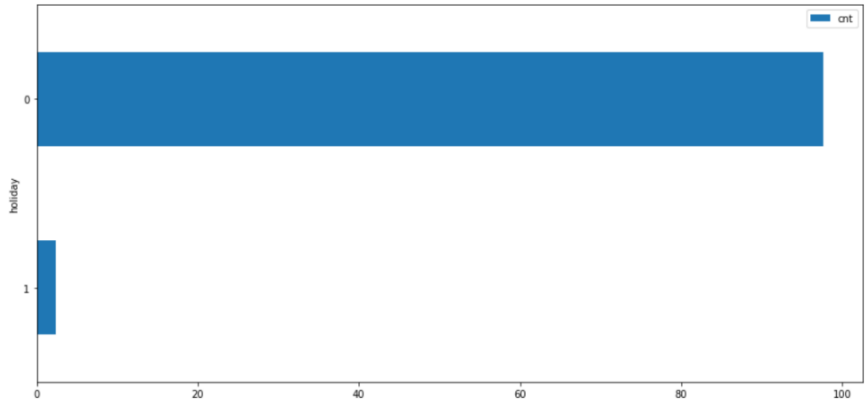
➔ Year 2012 (1) has more bike rental count than 2011 (0)
➔ So there is a increase in bike rental count from 2011 to 2012

❖ Relationship between Month and count



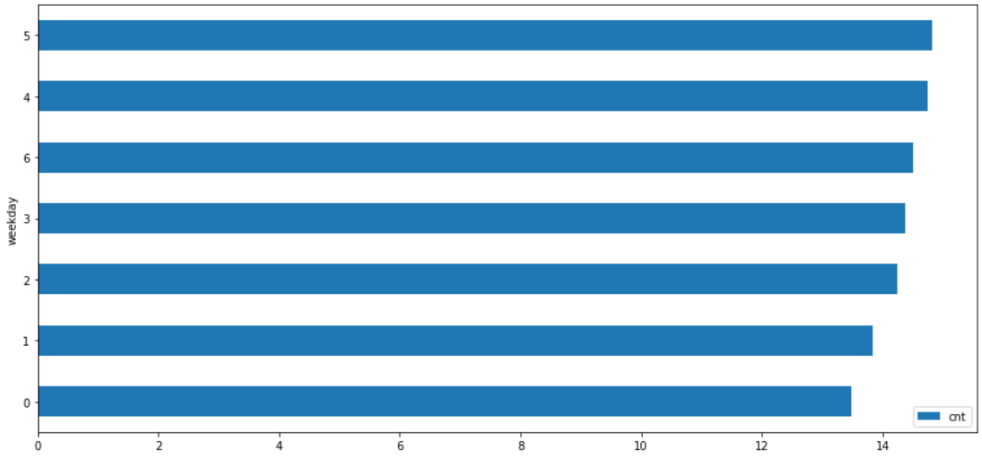
➔ August, June, September and July has high bike rental counts
➔ Bike rental count is low on January and February

❖ Relationship between Holiday and Count



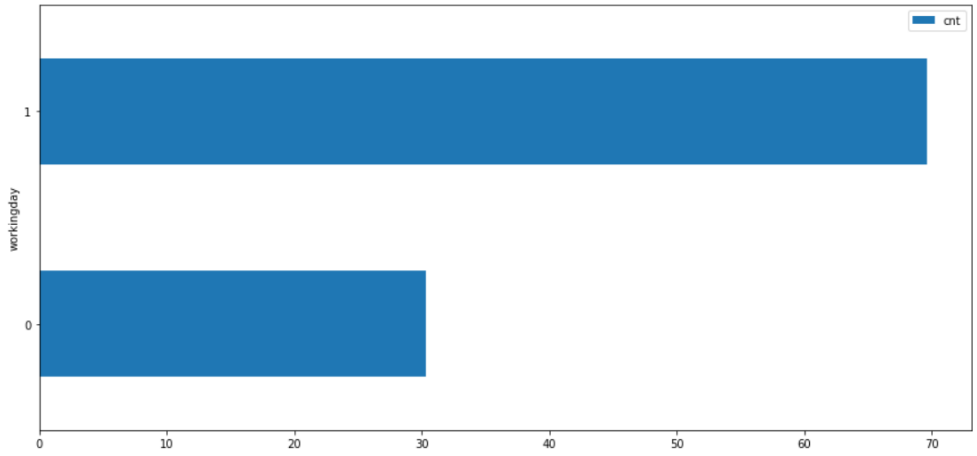
➔ There is huge difference on rental counts based on if there is a holiday or not
➔ Bike rental count is very high when there is no holiday
➔ Bike rental count is very low during holidays

❖ Relationship between Week day and Count



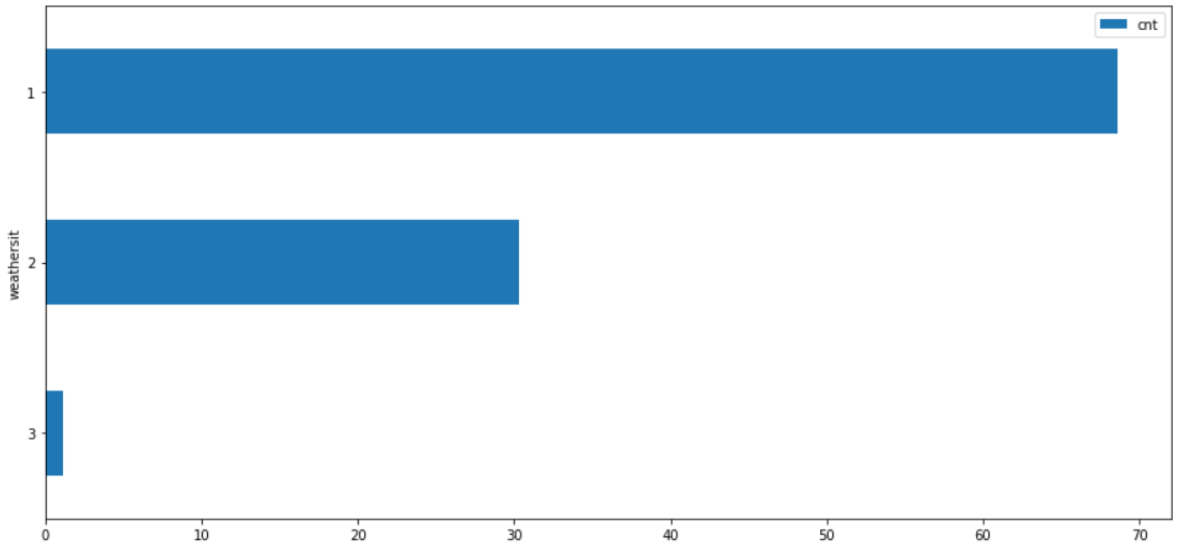
- ➔ Looks like there is no effect of week days on rental count
- ➔ Bike rental count is almost same on all days but we can see Friday and Thursday has more rental count
- ➔ And Sunday and Monday has low bike rental count

❖ <u>Relationship between working days and Count</u>



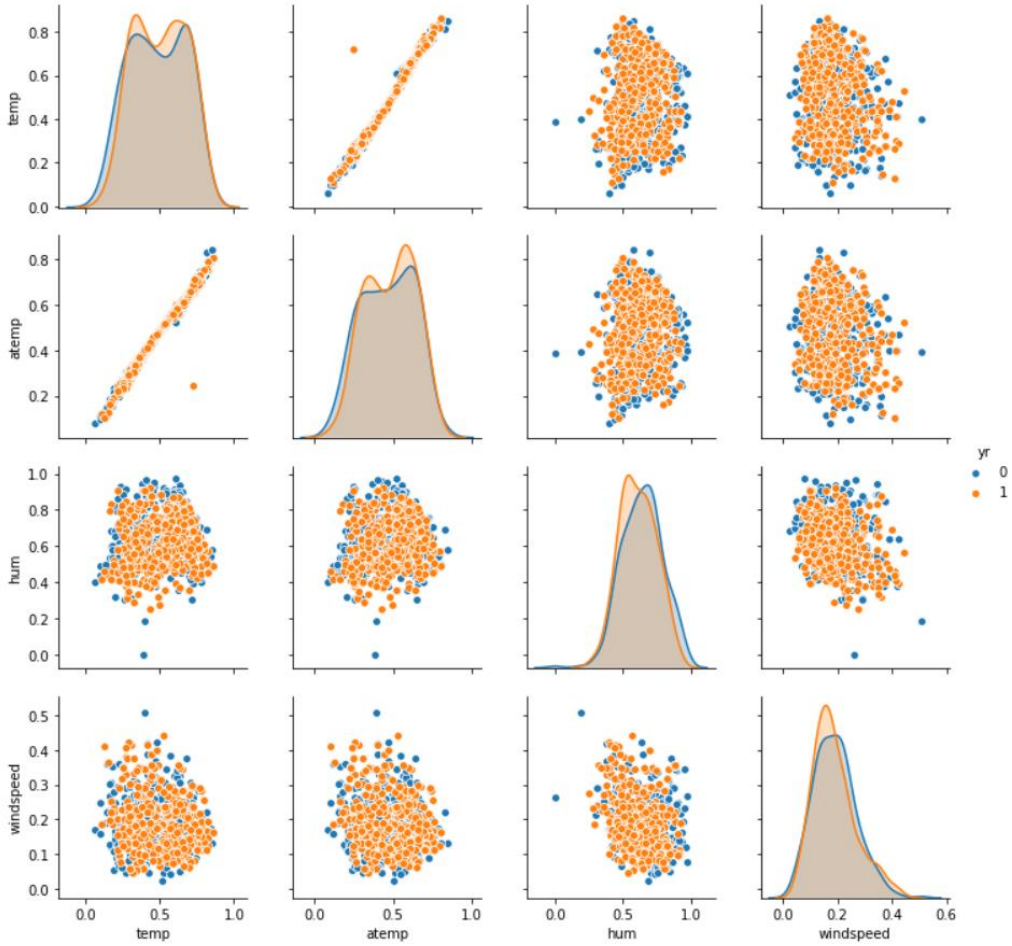
- ➔ Bike rental count is more when its not weekend or holiday
- ➔ Bike rental count is low when its weekend or holiday

❖ Relationship between Weather situation and Count

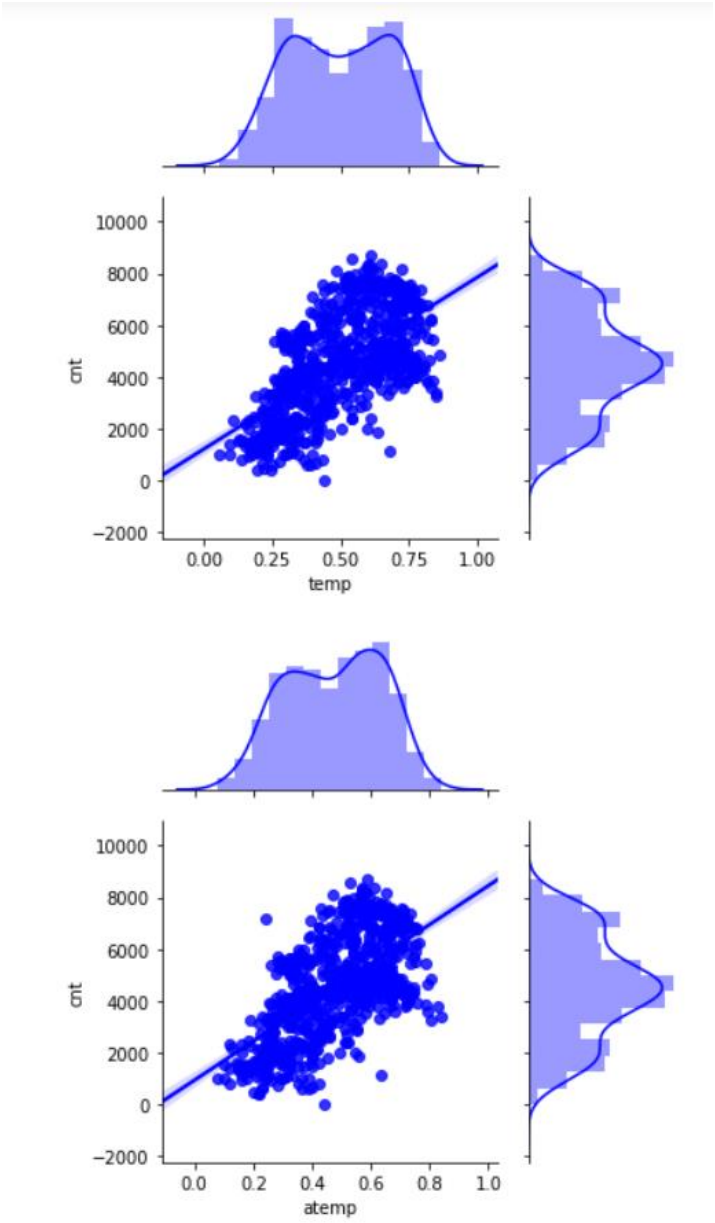


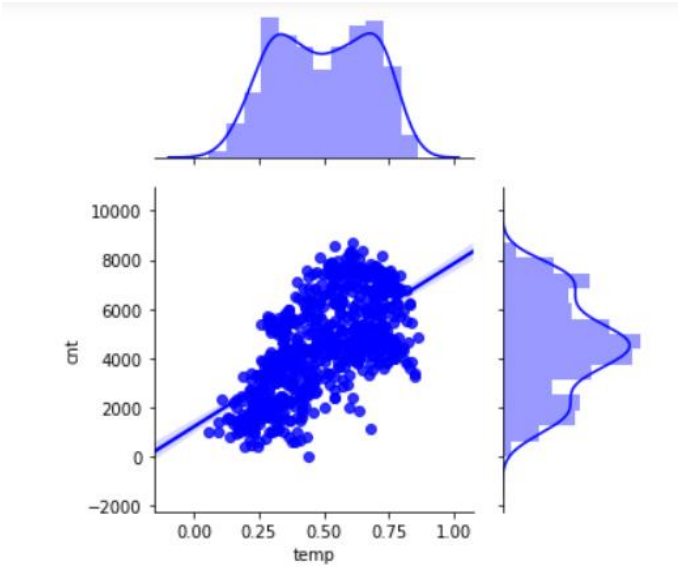
➔ Bike rental count is high when the weather is Clear, Few clouds and Partly cloudy
➔ Bike rental count is low when there is rain and thunderstorm

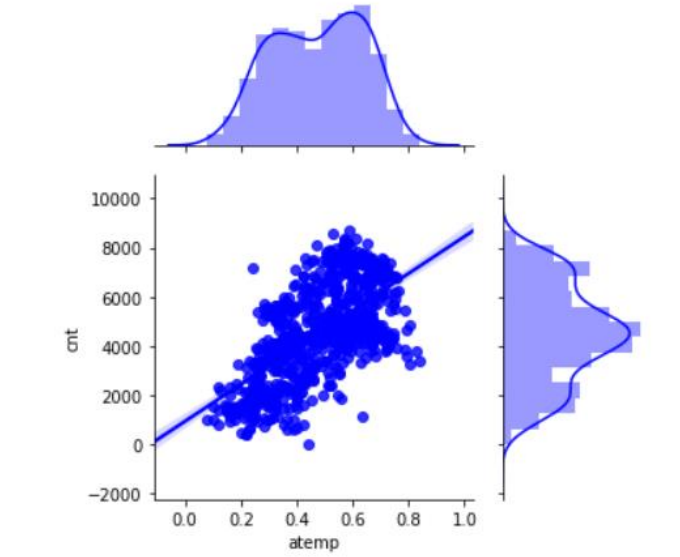
❖ Relationship among all continuous Variables using Pairplot



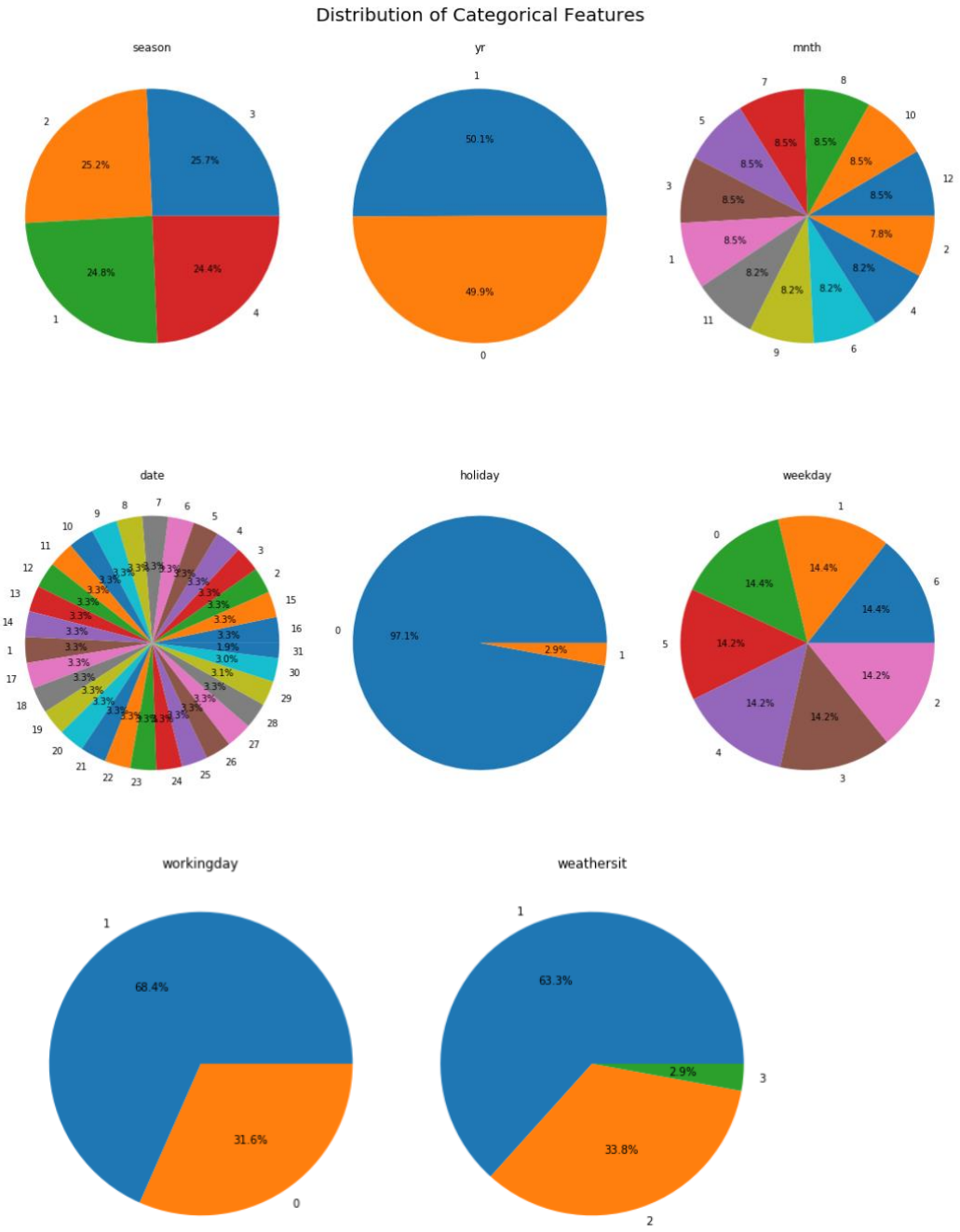
❖ Relationship between Continuous variables and Count




10000
8000
6000
4000
2000
0
−2000
cnt
0.00
0.25
0.50
0.75
1.00
temp


10000
8000
6000
4000
2000
0
−2000
cnt
0.0
0.2
0.4
0.6
0.8
1.0
atemp

❖ Distribution of Categorical variables



3. Feature Selection

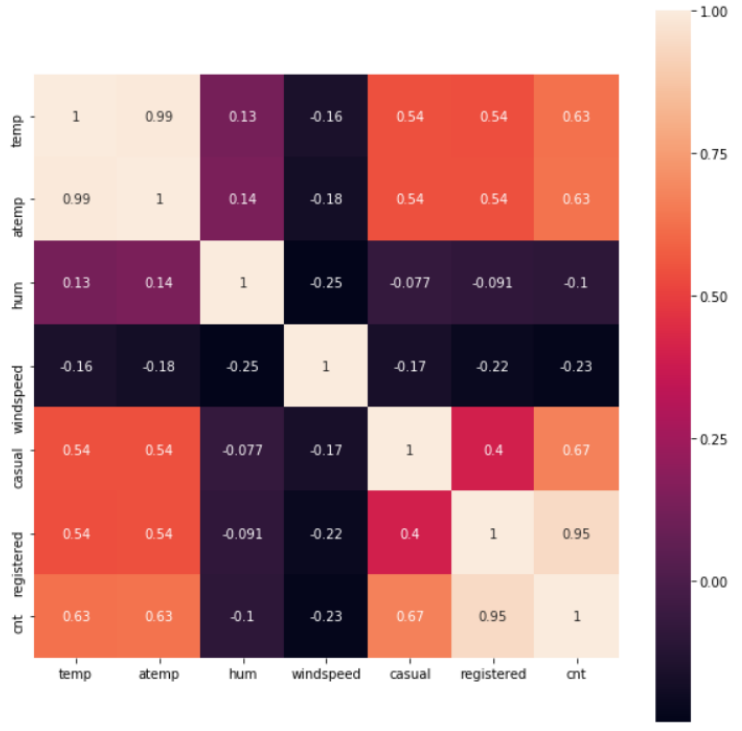
Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. Selecting subset of relevant columns for the model construction is known as Feature Selection. We cannot use all the features because some features may be carrying the same information or irrelevant information which can increase overhead. To reduce overhead we adopt feature selection technique to extract meaningful features out of data. This in turn helps us to avoid the problem of multi collinearity.

Correlation table for continuous variables:

```
data.loc[:, continuous_col].corr()
```

	temp	atemp	hum	windspeed	casual	registered	cnt
temp	1.000000	0.991702	0.126963	-0.157944	0.543285	0.540012	0.627494
atemp	0.991702	1.000000	0.139988	-0.183643	0.543864	0.544192	0.631066
hum	0.126963	0.139988	1.000000	-0.248489	-0.077008	-0.091089	-0.100659
windspeed	-0.157944	-0.183643	-0.248489	1.000000	-0.167613	-0.217449	-0.234545
casual	0.543285	0.543864	-0.077008	-0.167613	1.000000	0.395282	0.672804
registered	0.540012	0.544192	-0.091089	-0.217449	0.395282	1.000000	0.945517
cnt	0.627494	0.631066	-0.100659	-0.234545	0.672804	0.945517	1.000000

Correlation Plot:

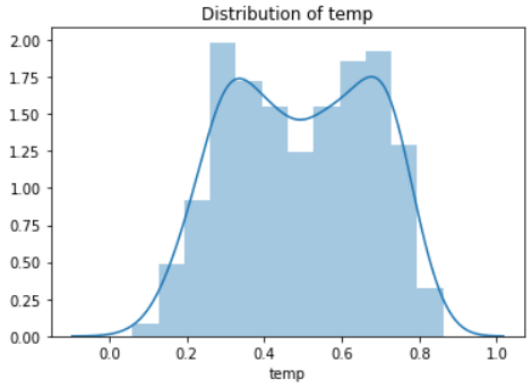


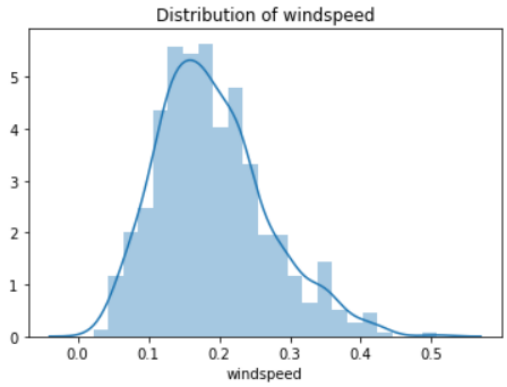
- ➔ From the plot we can see “temp” and “atemp” are higly correlated
- ➔ Count and Registered are highly correlated
- ➔ So atemp, hum and registered will be removed from the data
- ➔ Target variable is sum of “casual” and “registered”. Therefore casual will also be removed
- ➔

4. Feature Scaling

Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

<u>Plotting Distribution plot to check if data is normally distributed:</u>





- ➔ All continuous variables are normally distributed. Hence there is no need for Feature Scaling.

5. Modelling

5.1 Choosing the ML Models

Now, the exploratory analysis is done and Feature Engineering is also done. Now the data is ready for model building. 4 Algorithm will be used to build model:

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting

i) <u>Linear Regression Algorithm:</u>

Linear regression is one of the most commonly used predictive modelling techniques. The aim of linear regression is to find a mathematical equation for a continuous response variable Y as a function of one or more X variable(s). So that you can use this regression model to predict the Y when only the X is known.

Y= a1 X1 + a2X2 + ……. + anXn

ii) Decision Tree Algorithm:

A decision tree is a supervised machine learning model used to predict a target by learning decision rules from features. ... Based on the features in our training set, the decision tree model learns a series of questions to infer the class labels of the samples.

iii) <u>Random Forest Model:</u>

Random Forest is an ensemble machine learning algorithm that uses 'bagging' technique.

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.[1][2] Random decision forests correct for decision trees' habit of overfitting to their training set.

iv) **Gradient Boosting Model:**

Gradient Boost is an ensemble machine learning algorithm that uses 'boosting' technique.

Whereas random forests build an ensemble of deep independent trees, GBMs build an ensemble of shallow and weak successive trees with each tree learning and improving on the previous. When combined, these many weak successive trees produce a powerful "committee" that are often hard to beat with other algorithms.

All the above model, I am building with feature engineered dataset.

5.2 Choosing the Performance Measures for the Models

We are working on regression problem, so I think the best performance matrix could be

- R-SQAURED
- RMSE
- MSE
- MAE

5.3 Building the ML predictive Models

Let's start building models. Follow the following steps:

i) **Importing training and testing dataset**

I have already created train and test dataset so that I can use the same data in both R and Python and I will import it now.

```
# Importing train and test data used in R so as to use the same train and test data in both R and Python
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

```
y_train = train['cnt'].copy()
y_test = test['cnt'].copy()

x_train = train.drop(columns="cnt")
x_test = test.drop(columns="cnt")
```

ii) **Create function to measure evaluation metrics**

```
# Creating function that returns the evaluation metrics
def metrics(t, p):
    mae = mean_absolute_error(t,p)
    #mse = ((t-p)**2).mean()
    mse = mean_squared_error(t,p)
    rmse = np.sqrt(mse)
    rsqr = r2_score(t,p)

    print('RMSE      :',rmse)
    print('MSE       :',mse)
    print('MAE       :',mae)
    print('R²        :',rsqr)
```

iii) Build, Train & Predict with Linear Regression Model

```
# Linear Regression
lin_reg_model = LinearRegression().fit(x_train, y_train)
lin_reg_predict = lin_reg_model.predict(x_test)
metrics(y_test, lin_reg_predict)
```

```
RMSE      : 939.7447126560853
MSE       : 883120.1249650683
MAE       : 702.111822726758
R²        : 0.747306818134555
```

iv) Build, Train & Predict with Decision tree Model

```
# Decision Tree
dt_model = DecisionTreeRegressor().fit(x_train, y_train)
dt_predict = dt_model.predict(x_test)
metrics(y_test, dt_predict)
```

```
RMSE      : 1022.1258450188781
MSE       : 1044741.2430555555
MAE       : 659.0069444444445
R²        : 0.7010610657930471
```

v) Build, Train & Predict with Random Forest Model

```
# Random Forest
rf_model = RandomForestRegressor(n_estimators=500).fit(x_train, y_train)
rf_predict = rf_model.predict(x_test)
metrics(y_test, rf_predict)
```

```
RMSE      : 671.23817312126
MSE       : 450560.68505516666
MAE       : 470.9858055555556
R²        : 0.8710779995704795
```

vi) Build, Train $ Predict with Gradient Boosting

```
gb_model = GradientBoostingRegressor(loss='ls', learning_rate=0.1,
                                     n_estimators=300, subsample=1.0,
                                     criterion='friedman_mse',
                                     min_samples_split=2,
                                     min_samples_leaf=1,
                                     min_weight_fraction_leaf=0.0,
                                     max_depth=3,
                                     min_impurity_decrease=0.0,
                                     min_impurity_split=None,
                                     init=None, random_state=1,
                                     max_features=None, alpha=0.9,
                                     verbose=0, max_leaf_nodes=100,
                                     warm_start=False, presort='auto')
gb_model.fit(x_train, y_train)
gb_predict = gb_model.predict(x_test)
metrics(y_test, gb_predict)
```

```
RMSE      : 655.9114465781079
MSE       : 430219.8257521861
MAE       : 465.4918722627291
R²        : 0.8768982683129123
```

5.3.1 Building the Performance Matrix dataframe

Now when all the models are built, let's call the function to measure performance for each model and store in a dataframe

```
def model_eval_metric(model,X_test,Y_test,Y_predict):
    r_squared = model.score(X_test, Y_test)
    mse = mean_squared_error(Y_predict, Y_test)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(Y_predict, Y_test)
    return r_squared,mse,rmse, mae
```

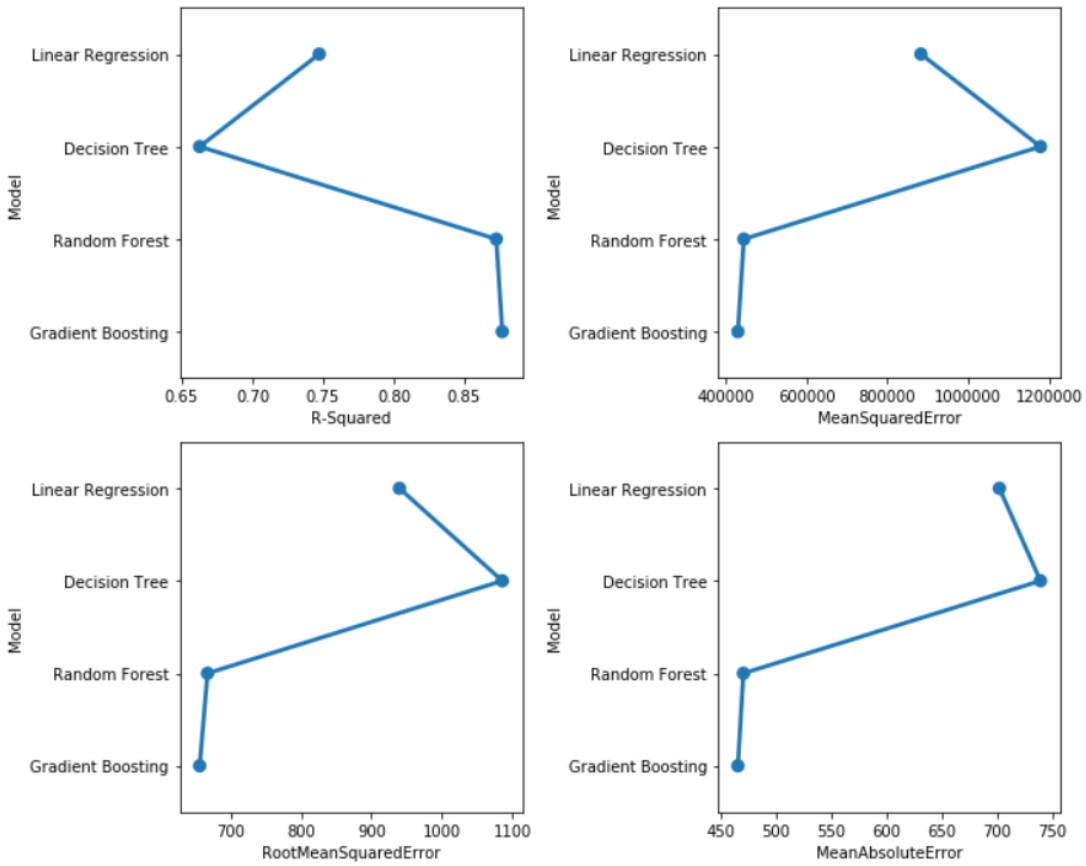
```
eval_metric=[]
ml_algo=['Linear Regression','Decision Tree','Random Forest','Gradient Boosting']
ml_models= [lin_reg_model,dt_model,rf_model,gb_model]
predictions =[lin_reg_predict,dt_predict,rf_predict,gb_predict]
i=0
for mod in ml_algo:
    R_SQR,MSE,RMSE,MAE = model_eval_metric(ml_models[i],x_test,y_test,predictions[i])
    eval_metric.append([mod,R_SQR,MSE,RMSE,MAE])
    i=i+1
model_performance=pd.DataFrame(eval_metric,columns =['Model','R-Squared','MeanSquaredError','RootMeanSquaredError',
                                                     'MeanAbsoluteError'])
```

```
model_performance
```

	Model	R-Squared	MeanSquaredError	RootMeanSquaredError	MeanAbsoluteError
0	Linear Regression	0.747307	8.831201e+05	939.744713	702.111823
1	Decision Tree	0.662645	1.178998e+06	1085.816750	739.277778
2	Random Forest	0.872688	4.449355e+05	667.034839	470.469014
3	Gradient Boosting	0.876898	4.302198e+05	655.911447	465.491872

5.3.2 Visualize the Performance of the all the models

```
# Plotting Model performance in each Metrics
fig, saxis = plt.subplots(2, 2,figsize=(16,12))
a=sns.pointplot(y='Model', x='R-Squared', rotate =90,data=model_performance, markers='o', linestyles='-', dodge=False,
                join=True,ax = saxis[0,0])
sns.pointplot(y='Model', x='MeanSquaredError', data=model_performance, markers='o', linestyles='-', dodge=False,
              join=True,ax = saxis[0,1])
sns.pointplot(y='Model', x='RootMeanSquaredError', data=model_performance, markers='o', linestyles='-', dodge=False,
              join=True,ax = saxis[1,0])
sns.pointplot(y='Model', x='MeanAbsoluteError', data=model_performance, markers='o', linestyles='-', dodge=False,
              join=True,ax = saxis[1,1])
plt.tight_layout()
```



From plot we can tell:

- Gradient Boosting has highest R-Squared value
- Gradient Boosting has lowest MSE value
- Gradient Boosting has lowest RMSE value
- Gradient Boosting ahs lowest MAE Value

It is evident from the results above that Gradient Boosting Model gives the best performance of all. Hence, Gradient Boosting model is the model our final model. Let's create sample output file using this model.

5.3.3 Generating Sample Output

Generate a sample output file with actual count and model predicted count.

```
data_predicted = x_test
data_predicted["Actual Count"] = y_test
data_predicted["Predicted Count"] = gb_predict
data_predicted["Predicted Count"] = round(data_predicted["Predicted Count"])
data_predicted.head(10)
```

	season	yr	mnth	date	holiday	weekday	workingday	weathersit	temp	windspeed	Actual Count	Predicted Count
0	1	0	1	1	0	6	0	2	0.344167	0.160446	985	1234.0
1	1	0	1	5	0	3	1	1	0.226957	0.186900	1600	1777.0
2	1	0	1	12	0	3	1	1	0.172727	0.304627	1162	1105.0
3	1	0	1	13	0	4	1	1	0.165000	0.301000	1406	1421.0
4	1	0	1	18	0	2	1	2	0.216667	0.146775	683	1472.0
5	1	0	1	31	0	1	1	2	0.180833	0.187192	1501	956.0
6	1	0	2	2	0	3	1	2	0.260000	0.264308	1526	1506.0
7	1	0	2	6	0	0	0	1	0.285833	0.141800	1623	1898.0
8	1	0	2	8	0	2	1	1	0.220833	0.361950	1530	1481.0
9	1	0	2	13	0	0	0	1	0.316522	0.260883	1589	1837.0

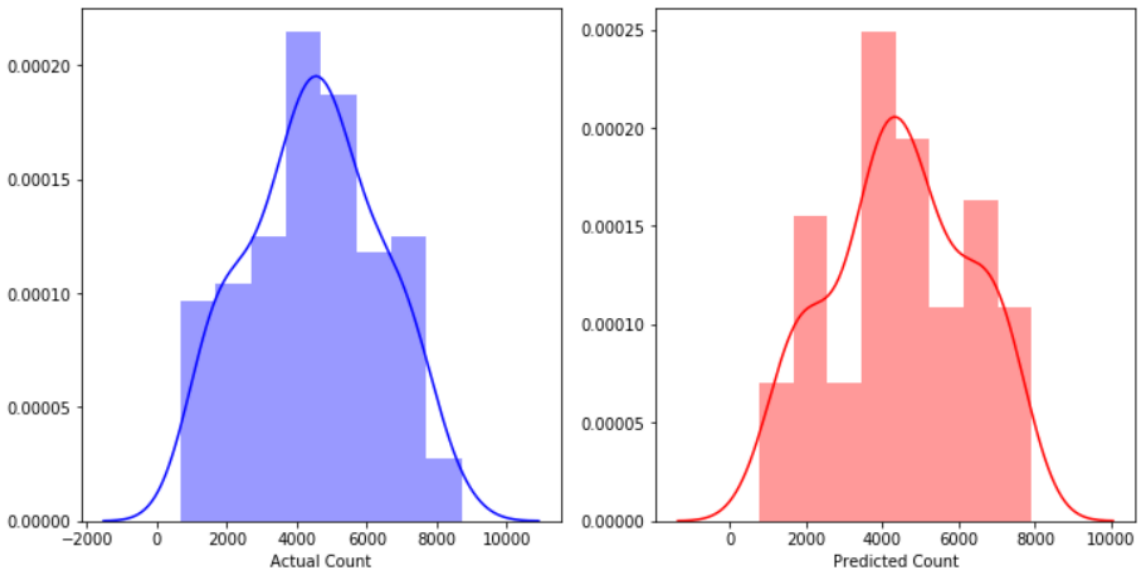
6. CONCLUSION

So, we have build a good model with around 87.6% of R-Squared value(Goodness of fit) using Gradient Boosting ML algorithm(which uses boosting technique)

Let's see the distribution of actual values and predicted values.

```
#-----Plotting the distributions of 'ActualCount' and 'PredictedCount'
fig, saxis = plt.subplots(ncols=2 ,figsize=(12,6))
sns.distplot(data_predicted["Actual Count"],color ='b', ax = saxis[0])
sns.distplot(data_predicted["Predicted Count"],color ='r', ax = saxis[1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25fc1cabfc8>
```



Both predicted values and actual counts have almost similar distribution. This model is ready for deployment now.

7. PYTHON CODE

8. RCODE