BIKE RENTAL PREDICTION

By Hitesh Kumar
hiteshkumar111@outlook.com

Content

1. Introduction

1.1 Problem statement

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

1.2 Data

Our task is to build regression models which will predict the count of bike rented depending upon different environmental and seasonal settings .

Given below is a sample of the data set

instant	date	season	year	month	holiday	weekday	workingday	weathersit
1	2011-01-01	1	0	1	0	6	0	2
2	2011-01-02	1	0	1	0	0	0	2
3	2011-01-03	1	0	1	0	1	1	1
4	2011-01-04	1	0	1	0	2	1	1
5	2011-01-05	1	0	1	0	3	1	1

temp	atemp	humidity	windspeed	casual	registered	count
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

The details of data attributes in the dataset are as follow –

The data set consists of 731 observations recorded between the period of 2 Years, between 2011 and 2012. It has 15 variables or predictors and 1 target variable. The data fields in the given data file are enumerated below.

S.No	Variable Name	Description

1	instant	Record Index
2	dteday	Date
3	season	Season (1:springer, 2:summer, 3:fall, 4:winter)
4	yr	Year
5	mnth	Month
6	holiday	weather day is holiday or not (extracted fromHoliday Schedule)
7	weekday	Day of the week
8	workingday	If day is neither weekend nor holiday is 1, otherwise is 0.
9	weathersit	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
10	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)
11	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)
12	hum	Normalized humidity. The values are divided to 100 (max)
13	windspeed	Normalized wind speed. The values are divided to 67 (max)
14	casual	Count of casual users
15	registered	Count of registered users
16	cnt	Count of total rental bikes including both casual and registered

Variables and their data types ~

```
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
```

RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns)

2. Data Preparation

Here we are Going to create a clean and high quality data set for data modeling and exploratory analysis.

2.1 Data Preprocessing ~

In this step we are going to rename some variable names and then convert some data values to text for better understanding of analysis and modeling.

Here we are also going to change the data types of variables to categorical and numerical for better analysis.

Renaming Variable names ~

```
bike_rental = bike_rental.rename(columns = {'dteday':'date','yr': 'year',
'mnth':'month','hum':'humidity', 'cnt':'count'})
bike_rental.head()
```

Changing values from Numeric to text ~

```
seasons = {1:'Spring', 2:'Summer', 3:'Fall', 4:'Winter'}
weathers = {1:'Clear', 2:'Misty+Cloudy', 3:'Light Snow or Rain'}
weekday = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6:'Sun'}
years = {0: '2011', 1: '2012'}
months = {1: 'Jan', 2: 'Feb', 3:'Mar',
4:'Apr',5:'May',6:'June',7:'July',8:'Aug',9:'Sep',10:'Oct',11:'Nov',12:'Dec'}



bike_rental['season'] = bike_rental['season'].map(seasons)
bike_rental['weathersit'] = bike_rental['weathersit'].map(weathers)
bike_rental['weekday'] = bike_rental['weekday'].map(weekday)
bike_rental['year'] = bike_rental['year'].map(years)
bike_rental['month'] = bike_rental['month'].map(months)

bike_rental['holiday'] = np.where(bike_rental['holiday']==1, 'Holiday', 'No
Holiday')
bike_rental['workingday'] = np.where(bike_rental['workingday']==1, 'Working
Day', 'No Working Day')
```

Changing Variables Data types ~

```
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null float64
date          731 non-null datetime64[ns]
season        731 non-null category
year          731 non-null category
month         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
humidity      731 non-null float64
windspeed     731 non-null float64
casual        731 non-null float64
registered    731 non-null float64
count         731 non-null float64
dtypes: category(7), datetime64[ns](1), float64(8)
```

2.2 Missing Value Analysis

Here we will check missing values in our data. IF there are missing values we try to find out the reason behind those missing values.

Finding:

We have ZERO missing values.

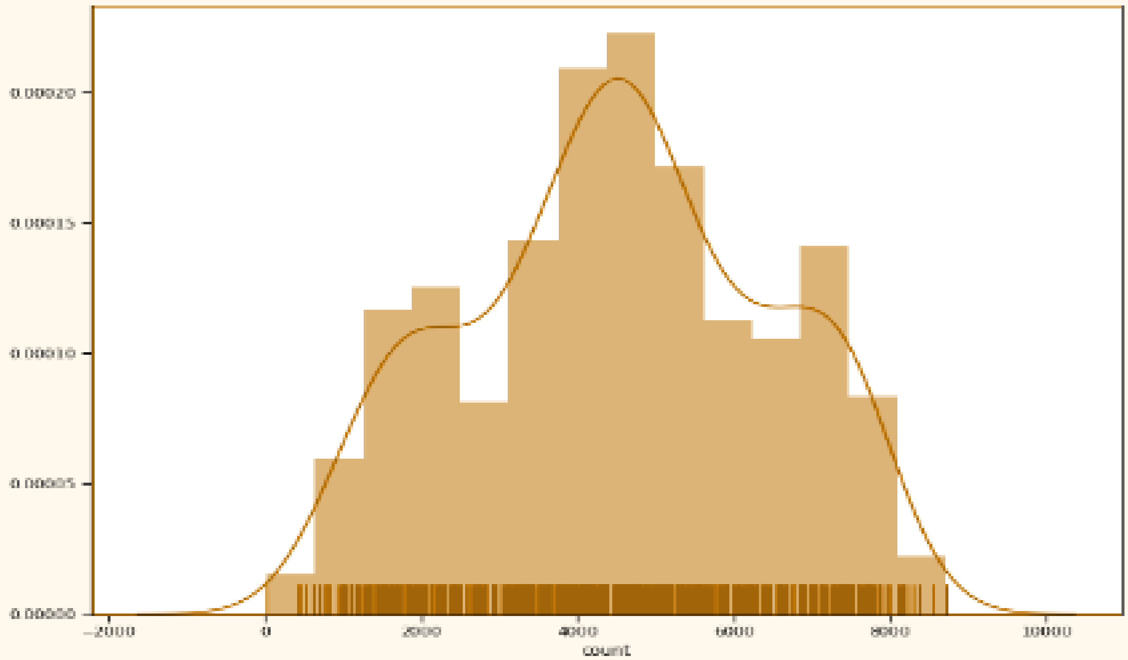
```
bike_rental.isnull().sum()

instant       0
date          0
season        0
year          0
month         0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

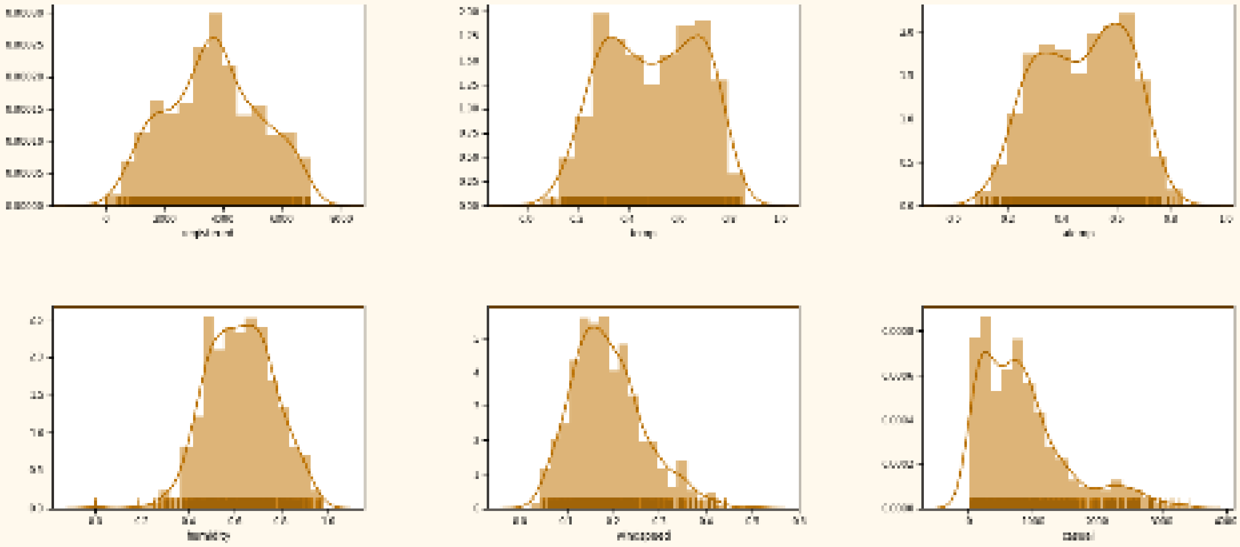
3. Exploratory Data Analysis (EDA)

3.1 Univariate Analysis

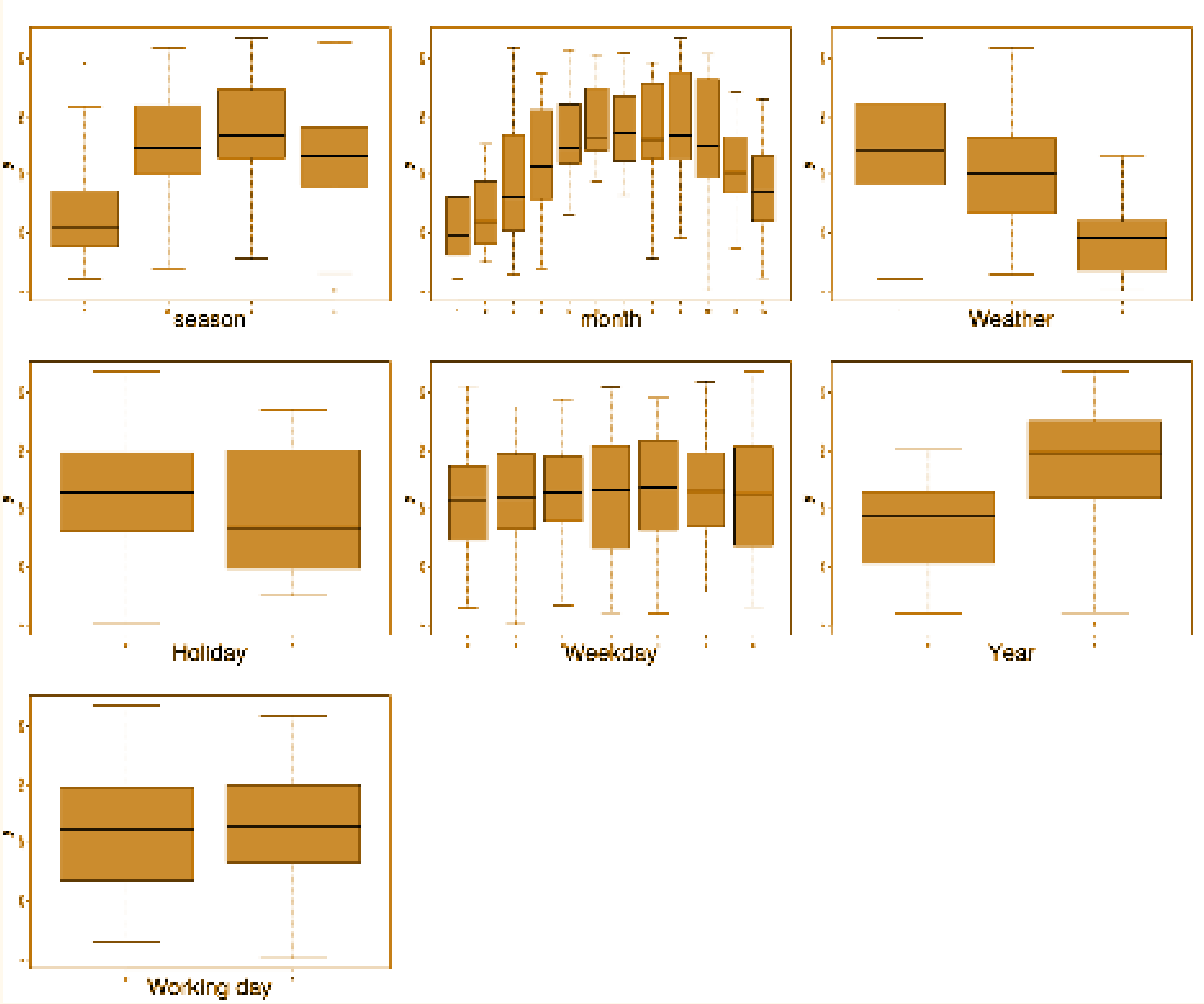
Shape of Target variable(count) ~



Shape of Other Numerical variables ~

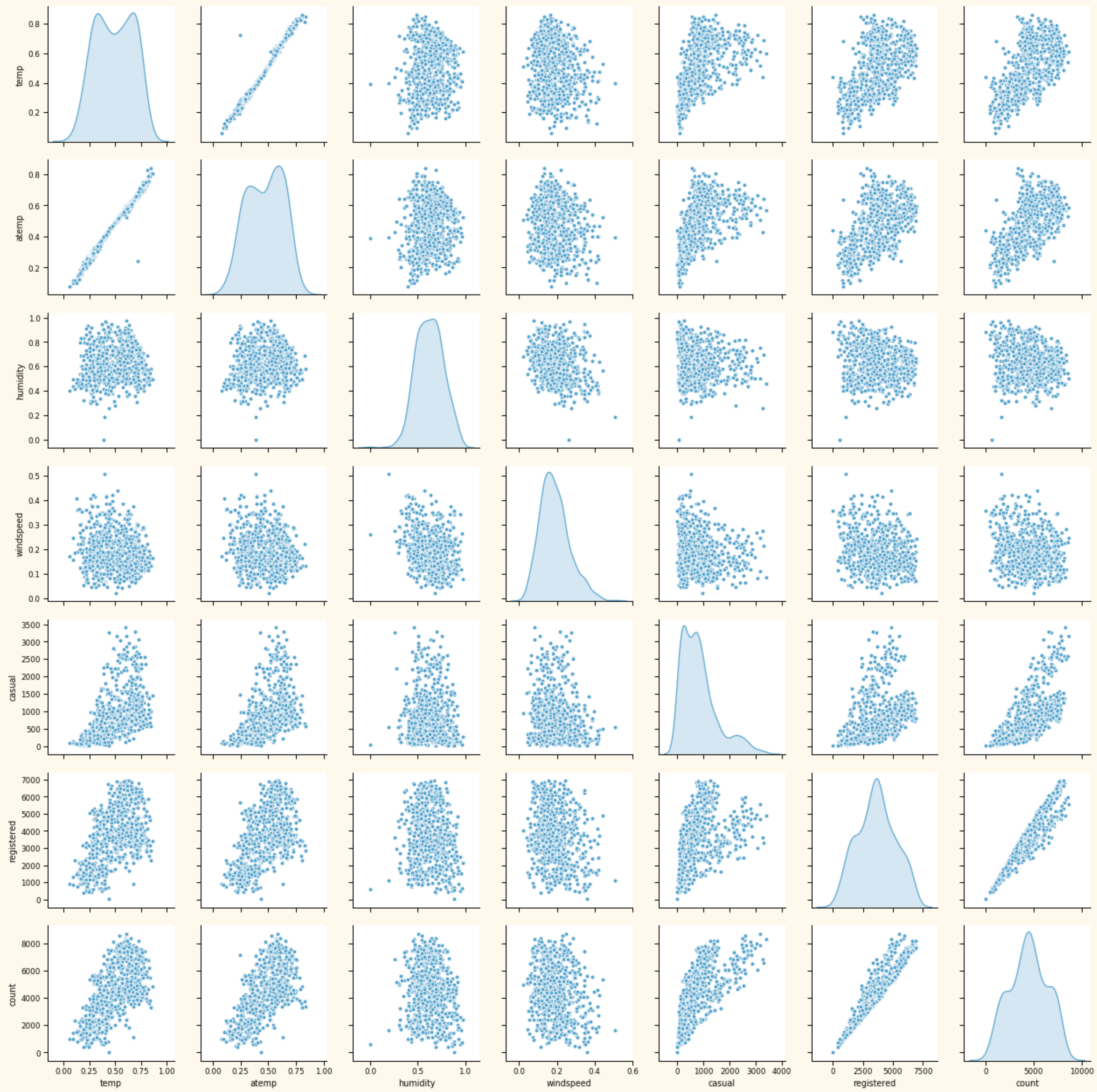


Categorical Variables~



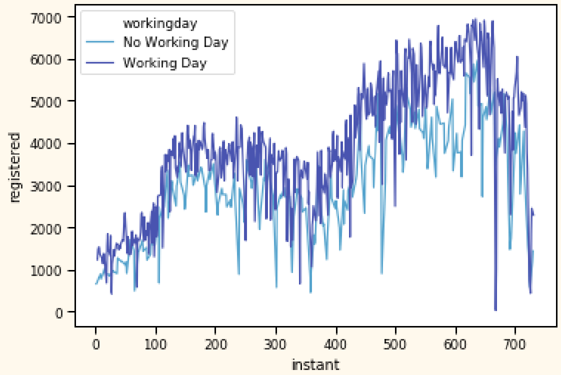
Multivariate Analysis

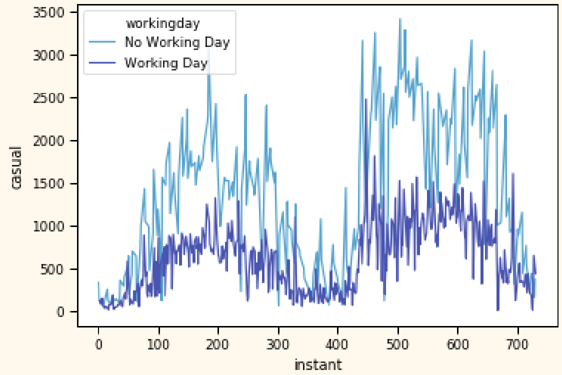
Scatter plot between all numerical variables



- It seems there is much correlation between temp and atemp, registred and count and to some extent in casual and count.

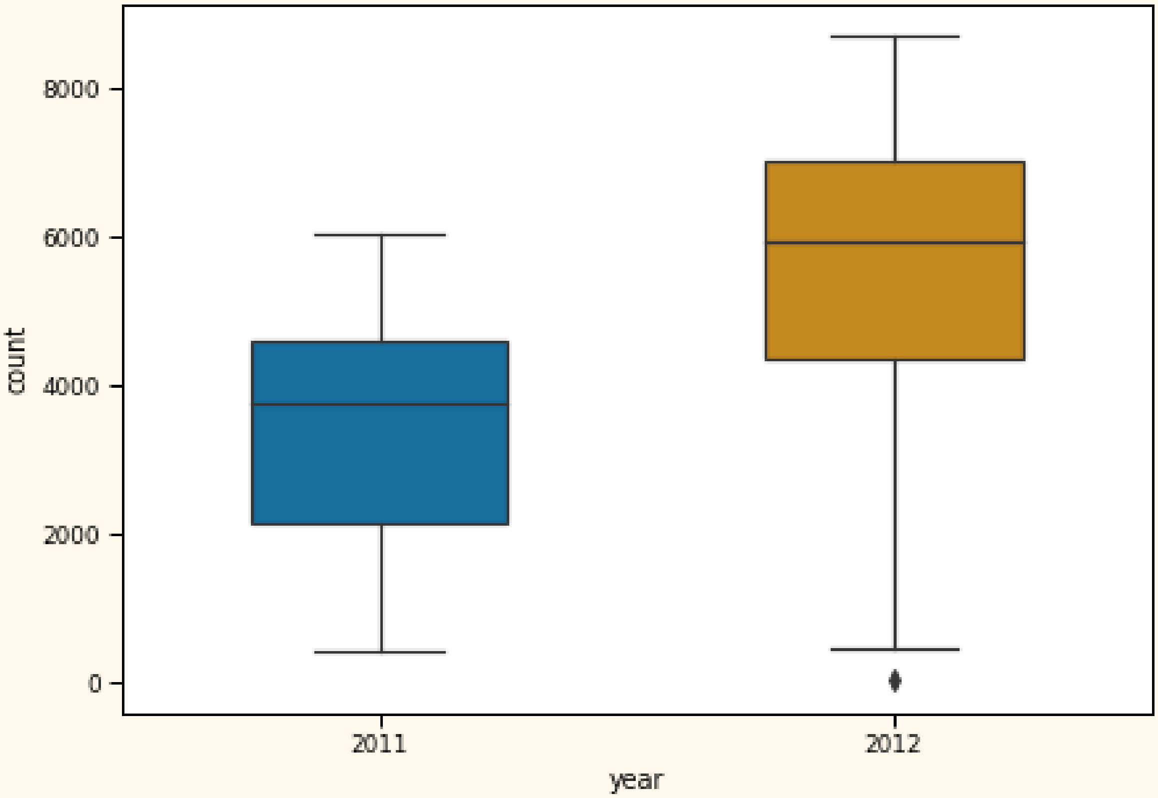
Working day relationship with registered and casual respectively ~





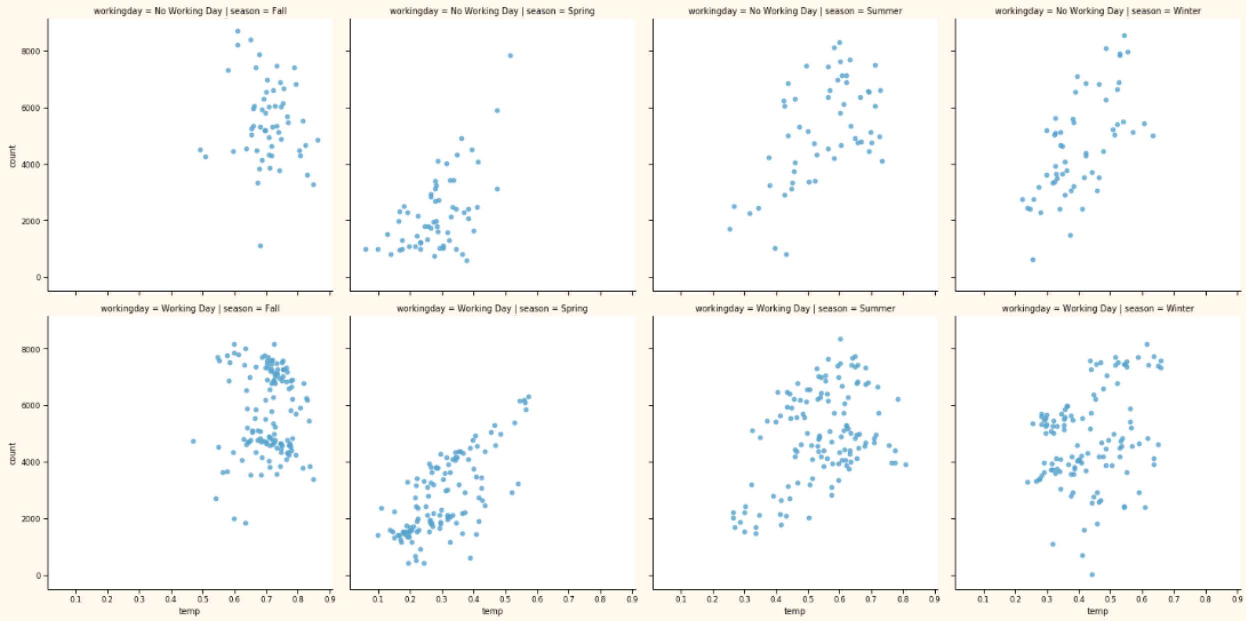
- A casual bike renter is most likely to rent a bike on a non-working day, whereas a registered user is more likely to rent it on a working day.

Change in rental count Year Wise~



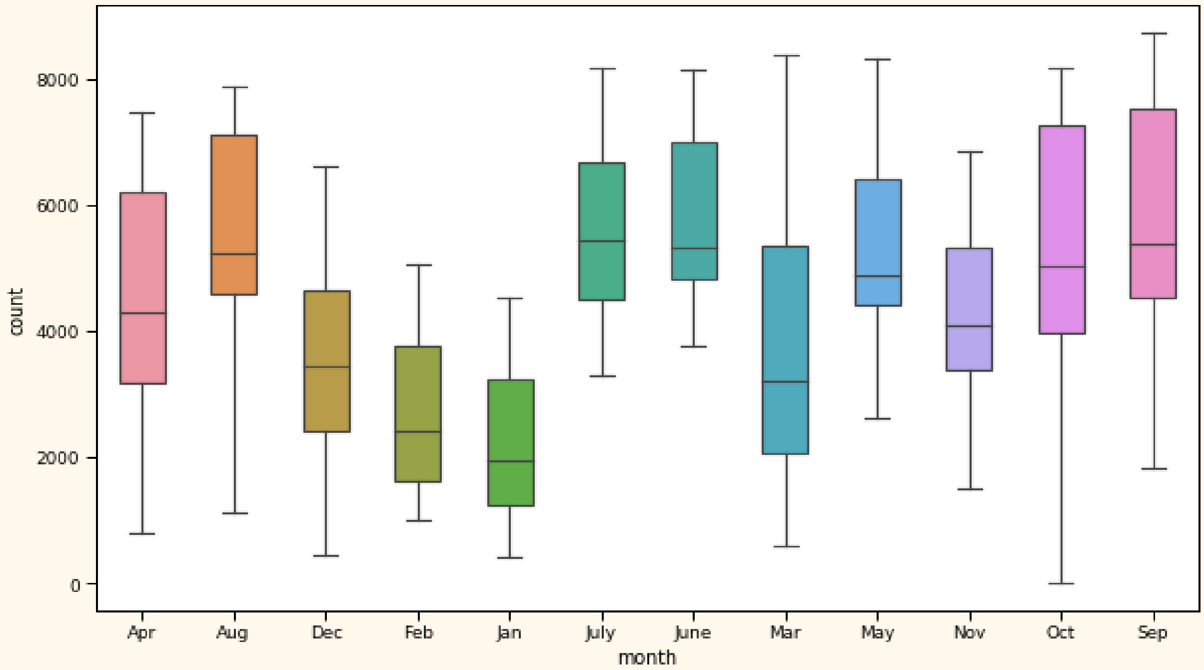
- There are more bikes rented in 2012 vs 2011.

Relationship between working day and seasons on Rental count

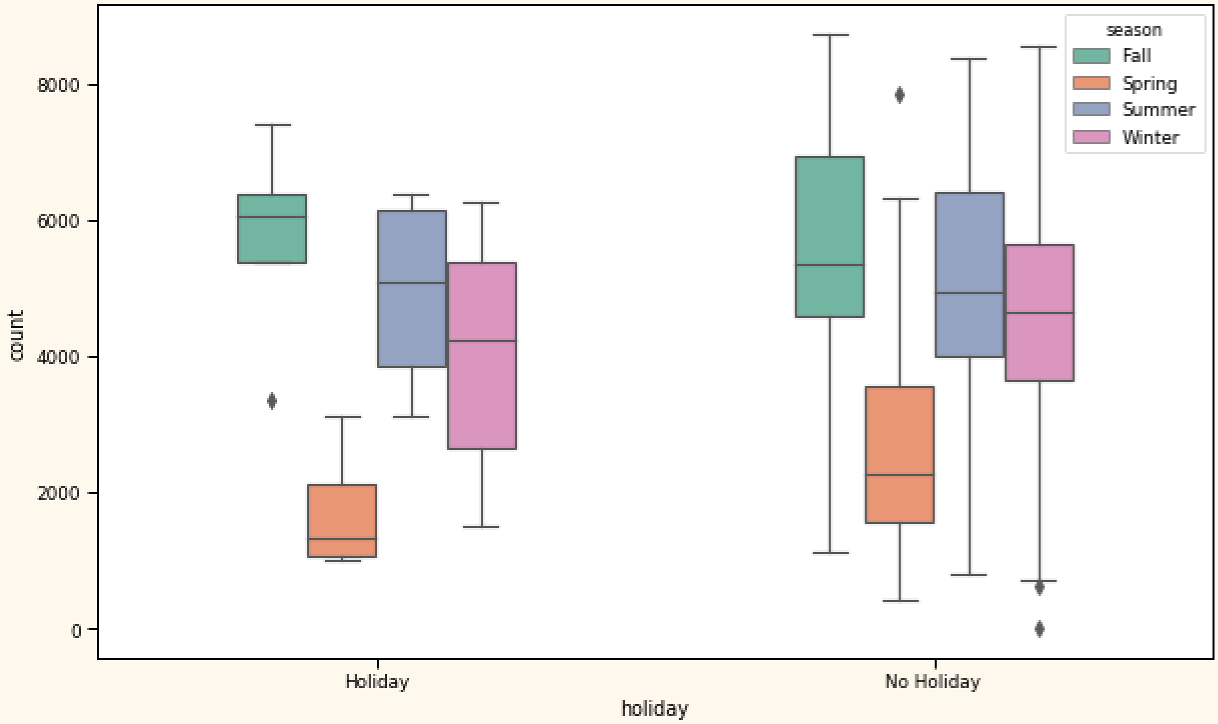


The pattern is quite similar here between working day and rental count but no. of count is higher on working day case irrespective of season.

Rental Monthwise ~

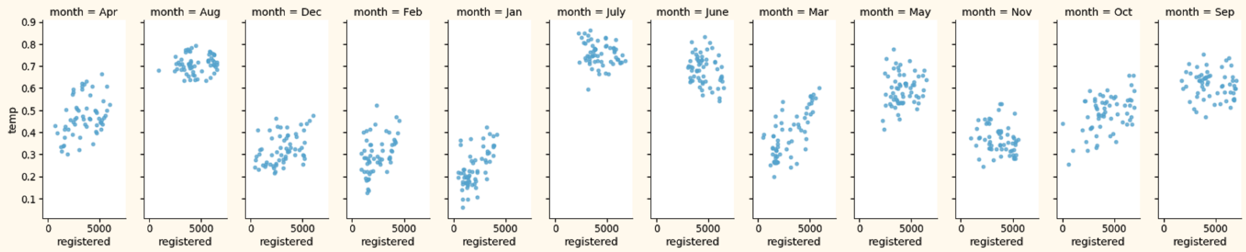


Hoiliday , Season and Count~

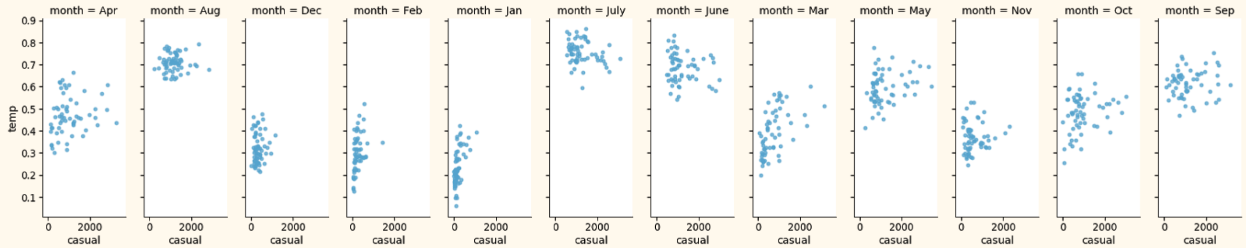


- It seems rental count is higher on non holidays irrespective of season.
- On holidays of winter and summer the bike rental count is quite high.

Relationship between casual and registered with temp over the all months~

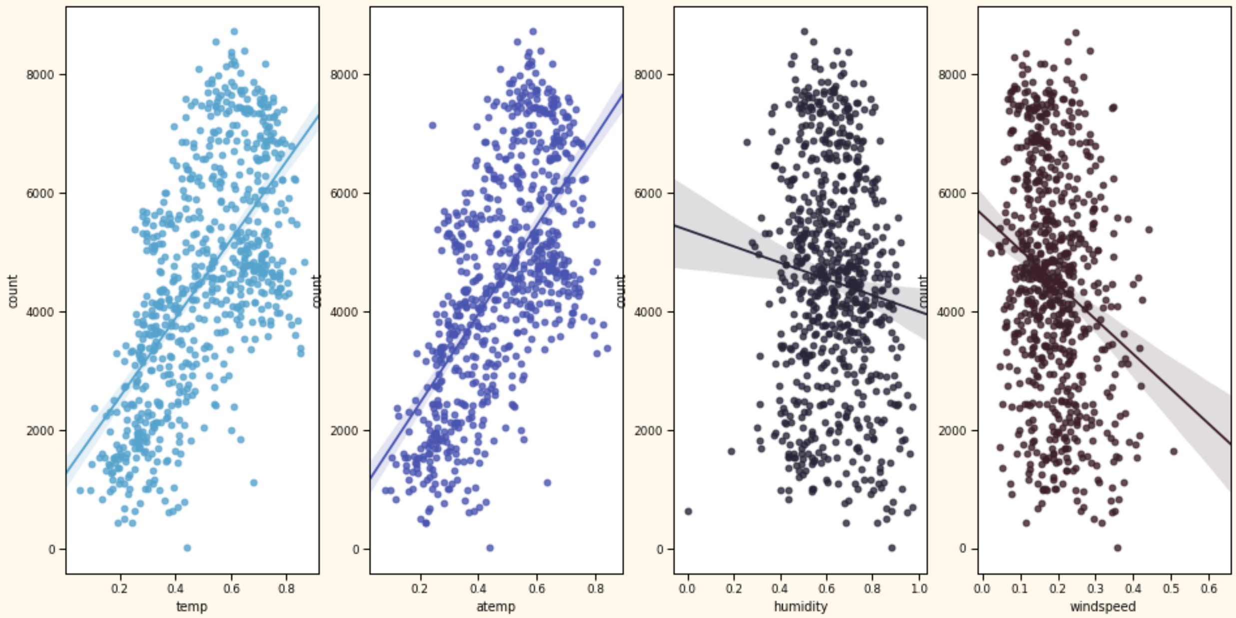


Registered vs temp



Casual vs temp

There is no particular pattern here.



- ‘Count’ and ‘Temperature’ have strong and positive relationship. It means that as the temperature rises, the bike demand also increases.
- ‘atemp’ and ‘Count’ have strong and positive relationship. It means that as the ambient temperature rises, demand for bikes also increases.
- Humidity’ has a negative linear relationship with ‘Count’. As humidity increases, count decreases.
- ‘Windspeed’ has a negative linear relationship with ‘Count’. With an increase in windspeed, bike count decreases.

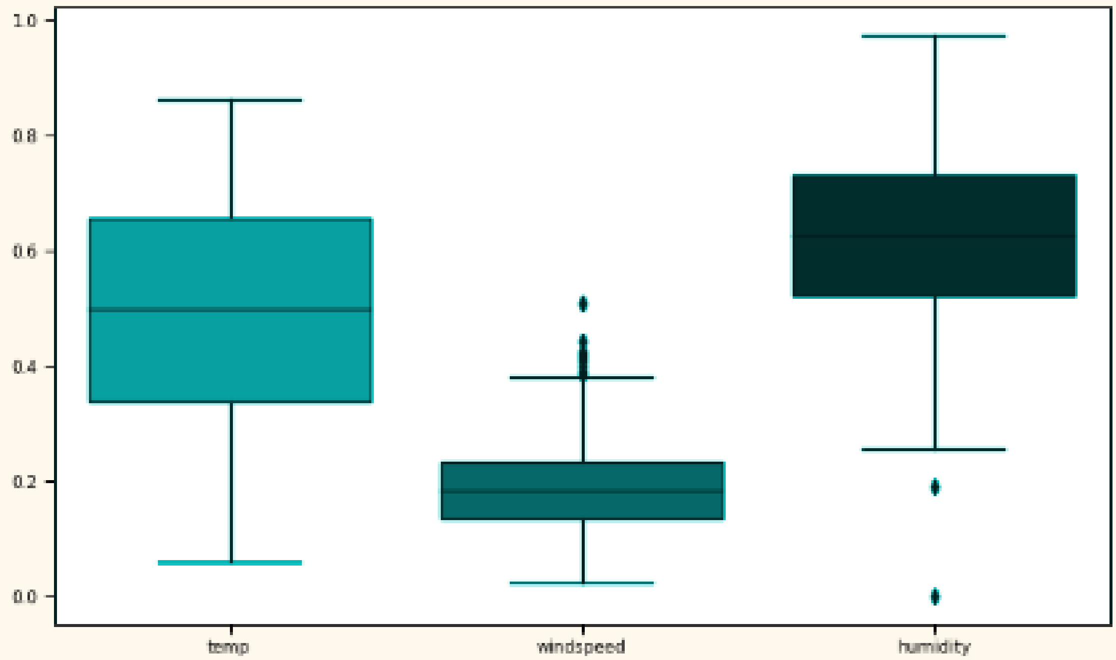
3.3 Outlier Analysis

In Outlier analysis we will try to find out data points which are inconsistent with other data points, these values can be extreme values which can be caused due to input error, system malfunctioning or could be a case of seasonality.

First we detected these values and then we'll take corrective measures.

Outlier Detection

Our Outliers in data is such as~



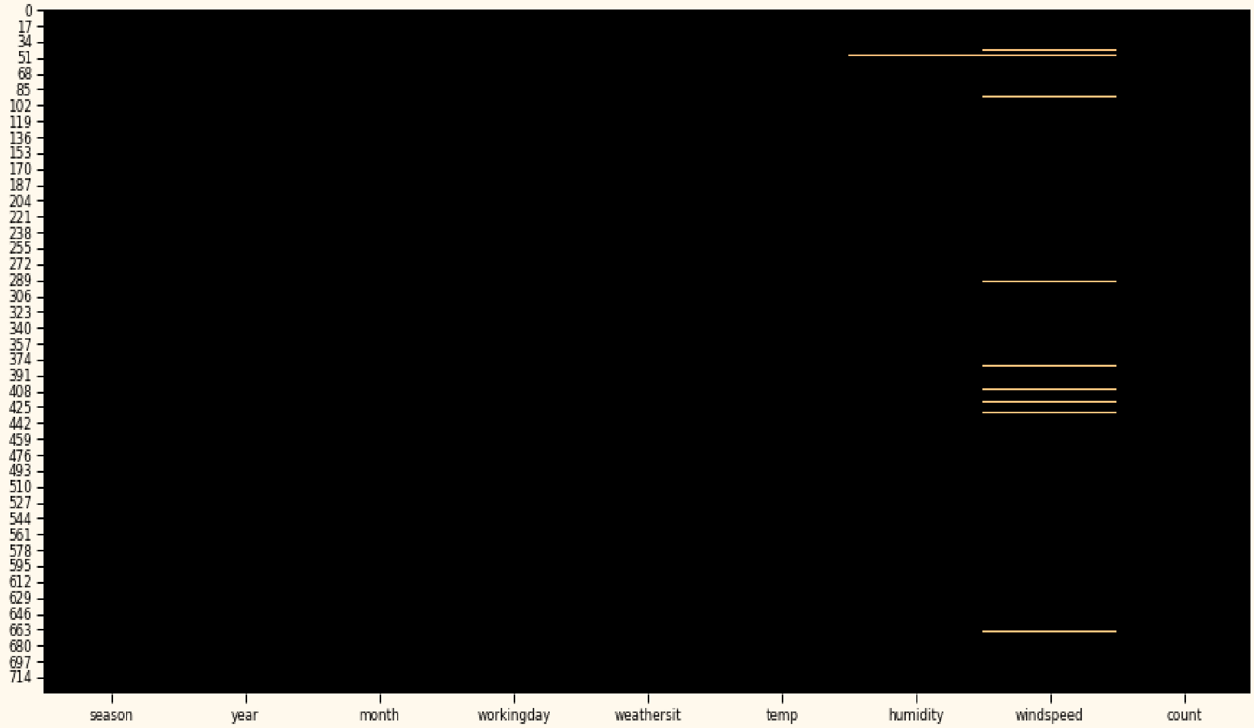
Treatment of Outliers

We'll remove these outliers and replace them with NaN's.

```
bike_rental.isnull().sum()
```

```
season         0
year           0
month          0
workingday     0
weathersit     0
temp           0
humidity       2
windspeed     13
count          0
dtype: int64
```

Now we have a total of 15 missing values.



Since the number of NA’s are very low we'll drop them.

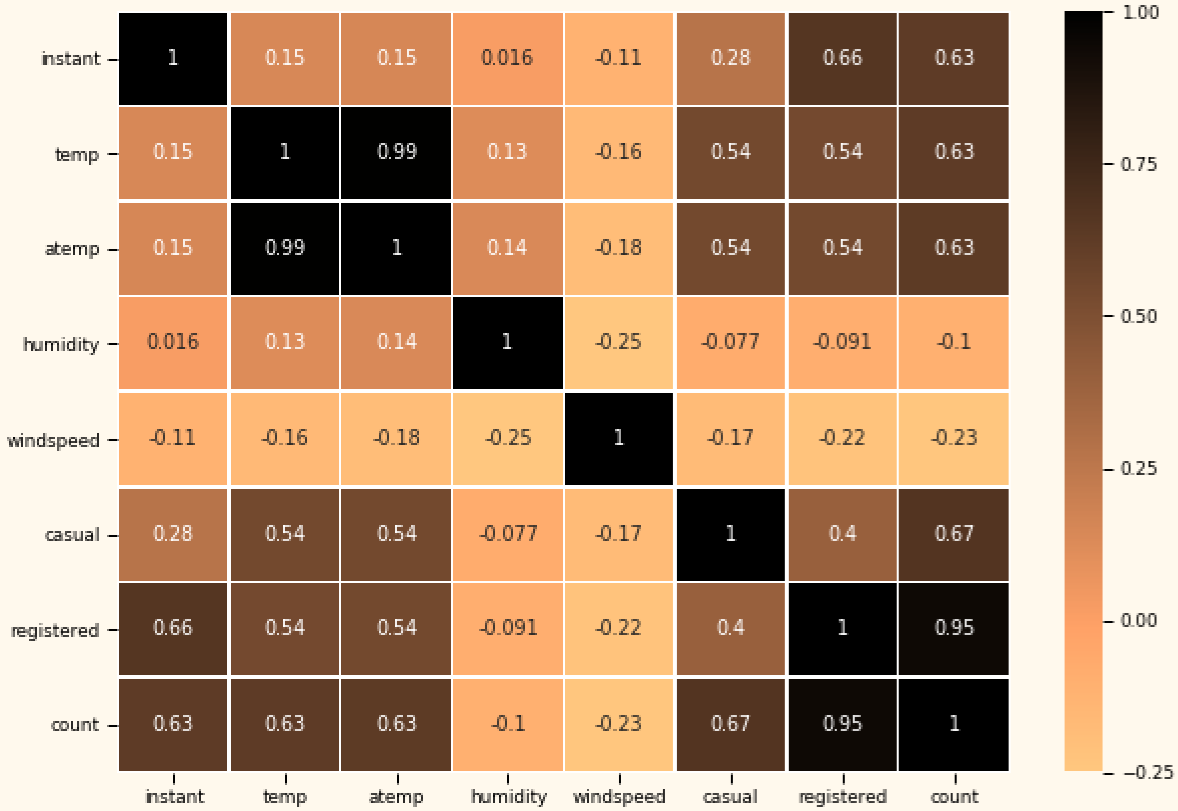
3.4 Feature selection

Not every feature is equally important some are useful some are not, so we will choose some feature and discard some. in order for our model to perform better. By performing feature selection we’ll assess the importance of all variables.

Correlation Analysis

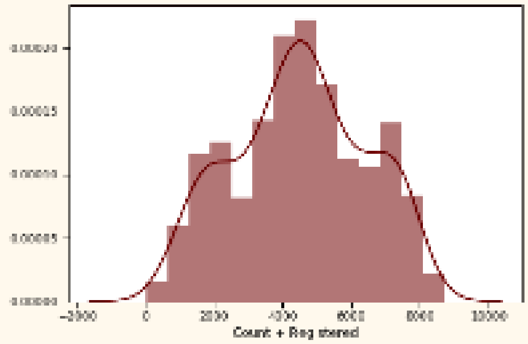
We will perform correlation analysis to see the correlation between all numeric variables. We’ll remove highly correlated variables because we want independent variables, ideally there should be no correlation between two independent variables.

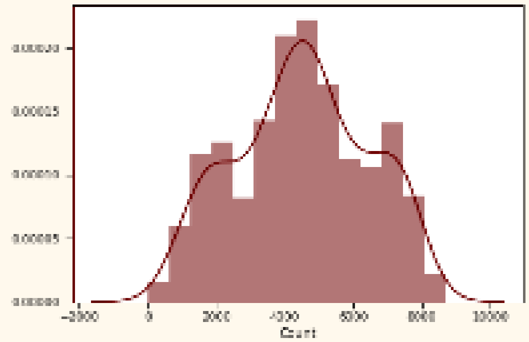
Here’s the heatmap of correlated numeric variables.



Findings~

- “temp” and “atemp” are most highly correlated variables, So we will remove “atemp” since both variables have same information.
- Casual and Registered highly correlated with count because “count” is sum of these variables.





Chi-square test

From this test will check the independence between categorical variables.

	season	year	month	holiday	weekday	workingday	weathersit
season	-	1.0	0.0	0.683	1.0	0.887	0.021
year	1.0	-	1.0	0.995	1.0	0.98	0.127
month	0.0	1.0	-	0.559	1.0	0.993	0.015
holiday	0.683	0.995	0.559	-	0.0	0.0	0.601
weekday	1.0	1.0	1.0	0.0	-	0.0	0.278
workingday	0.887	0.98	0.993	0.0	0.0	-	0.254
weathersit	0.021	0.127	0.015	0.601	0.278	0.254	-

Finding -

- From chi-square test, Removing weekday, holiday because they don't contribute much to the independent variable.
- Removing the instant variable, as it is index in datasets
- Removing date variable as we have to predict count on seasonal basis not date basis.

4. Modeling

Now we'll try to predict our target variable "count" which is a continuous variable, this will be a Regression problem and we will be using Supervised machine learning algorithms. We'll be using different models under Supervised Learning for Regression problems. On the basis of the performance of the model, the best model will be chosen for prediction of the count of our test . Before Doing modeling we will encode our variables, since all categorical variables are nominal so we will use one hot encoding.

```
cat_names = ['season', 'year', 'month','weathersit']
bkr_enc = pd.get_dummies(train,columns=cat_names)
```

```

RangeIndex: 717 entries, 0 to 716
Data columns (total 26 columns):
workingday      717 non-null category
temp            717 non-null float64
```

```
humidity         717 non-null float64
windspeed        717 non-null float64
count            717 non-null float64
season_1         717 non-null uint8
season_2         717 non-null uint8
season_3         717 non-null uint8
season_4         717 non-null uint8
year_0           717 non-null uint8
year_1           717 non-null uint8
month_1          717 non-null uint8
month_2          717 non-null uint8
month_3          717 non-null uint8
month_4          717 non-null uint8
month_5          717 non-null uint8
month_6          717 non-null uint8
month_7          717 non-null uint8
month_8          717 non-null uint8
month_9          717 non-null uint8
month_10         717 non-null uint8
month_11         717 non-null uint8
month_12         717 non-null uint8
weathersit_1     717 non-null uint8
weathersit_2     717 non-null uint8
weathersit_3     717 non-null uint8
dtypes: category(1), float64(4), uint8(21)
```

Now we have 26 variables after one hot encoding, we used one hot encoding because we don't want to give our model any idea that one category weighs more than another, as all categorical variables are nominal not ordinal.

So now we apply different algorithms and choose the best model.

4.1 Multiple Linear Regression

Here our Linear regression with all variables only date and instant is removed.

```
lm(formula = count ~ ., data = training_set)

Residuals:
    Min       1Q  Median       3Q      Max
-4056.0  -339.0    84.6    452.1   2606.4

```

```
Coefficients: (1 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1474.06     295.84   4.983 8.81e-07 ***
season2       755.53     215.25   3.510 0.000491 ***
season3       763.96     251.07   3.043 0.002474 **
season4      1497.33     218.59   6.850 2.31e-11 ***
year1        2020.45      70.39  28.702  < 2e-16 ***
month2        293.31     174.32   1.683 0.093126 .
month3        612.93     203.88   3.006 0.002784 **
month4        810.74     303.56   2.671 0.007828 **
month5        801.25     324.39   2.470 0.013863 *
month6        584.60     345.84   1.690 0.091615 .
month7        102.39     383.45   0.267 0.789565
month8        379.53     370.92   1.023 0.306723
month9       1090.85     320.27   3.406 0.000715 ***
month10       721.81     295.33   2.444 0.014887 *
month11       -75.76     278.29  -0.272 0.785571
month12       138.00     226.82   0.608 0.543223
holiday1     -840.53     226.61  -3.709 0.000233 ***
weekday1      177.36     133.46   1.329 0.184513
weekday2      225.98     127.97   1.766 0.078055 .
weekday3      299.71     133.23   2.250 0.024936 *
weekday4      315.29     127.54   2.472 0.013783 *
weekday5      410.03     128.46   3.192 0.001508 **
weekday6      386.68     131.17   2.948 0.003357 **
workingday1       NA         NA      NA       NA
weathersit2  -426.73      92.81  -4.598 5.48e-06 ***
weathersit3 -1960.35     262.70  -7.462 4.11e-13 ***
temp         4595.88     500.87   9.176  < 2e-16 ***
humidity    -1415.93     363.67  -3.893 0.000113 ***
windspeed   -3072.56     508.87  -6.038 3.16e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 757.9 on 473 degrees of freedom
Multiple R-squared:  0.8507,    Adjusted R-squared:  0.8421
F-statistic: 99.79 on 27 and 473 DF,  p-value: < 2.2e-16
```

Summary of predicted vs actual count

```
[1] "summary of Predicted count values"

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
 -1311    3483    4809    4602    6033    7887
```

```
[2] "summary of actual count values"
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  506    3214    4582    4548    6049    8395
```

Metrices

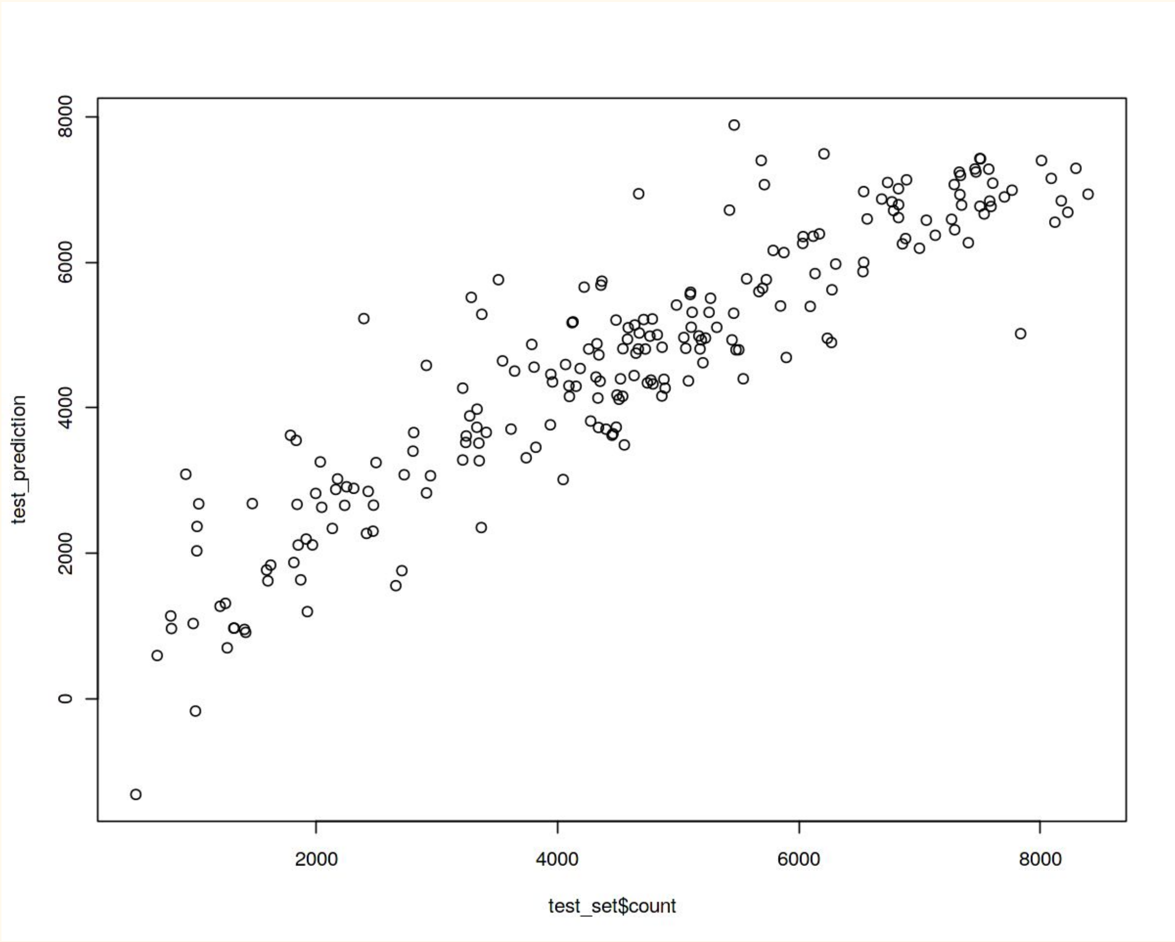
Mae	611.475678850163
Mse	672137.48949875
Rmse	819.839916995233
Mape	0.202556460709544

R-Squared Error

```
R2(test_set$count,test_prediction)
```

```
r-squared : 0.8314462
```

Plot Predicted vs actual count



Now we will remove features which we decided to discard during our feature selection. Here's the results of our new linear model without those variables.

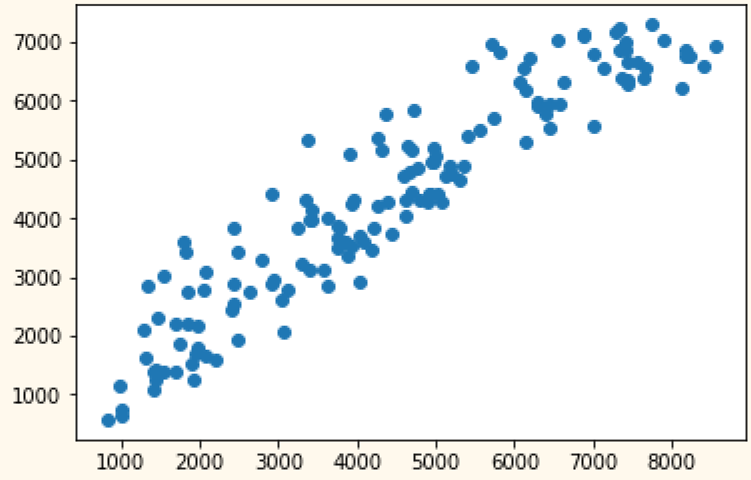
summary of predicted counts vs actual counts

	0		0
count	144.000000	**count**	144.000000
mean	4323.157767	**mean**	4395.312500
std	1822.685684	**std**	2071.799007
min	569.659141	**min**	822.000000
25%	2910.931872	**25%**	2598.750000
50%	4306.904012	**50%**	4300.000000
75%	5908.972984	**75%**	6121.750000
max	7301.344298	**max**	8555.000000

Metrics

```
R2 score : 0.87
Root Mean squared error: 747.40
Mean Absolute error: 586.44
MAPE: 17.03
Accuracy: 82.97
```

Plot b/w Predicted and actual count



Findings

We can see that the R-squared of first model was 0.831 means it captures only 83.1% of target variable while our newer model has R-squared 0.87 means it captures 87% of target variable, which signifies that our newer model with less variable is can explains more about our data.

4.2 Decision Trees

First Decision Tree

```
Regression tree:
rpart(formula = count ~ ., data = training_set, method = "anova")

Variables actually used in tree construction:
[1] humidity  month     season    temp      windspeed year

Root node error: 1819126781/501 = 3630992

n= 501

         CP nsplit rel error  xerror     xstd
1  0.394434      0   1.00000 1.00246 0.049340
2  0.216107      1   0.60557 0.65304 0.037452
3  0.073782      2   0.38946 0.43539 0.038328
4  0.038184      3   0.31568 0.34273 0.032678
5  0.036448      4   0.27749 0.31534 0.032816
6  0.022957      5   0.24105 0.27516 0.031823
7  0.015123      6   0.21809 0.27647 0.032007
8  0.011845      7   0.20297 0.28399 0.034841
9  0.010823      8   0.19112 0.28001 0.035097
10 0.010000      9   0.18030 0.26619 0.034280
```

Summary of predicted vs actual count

```
[1] "summary of Predicted count values"
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1758	3537	4526	4683	6876	6876

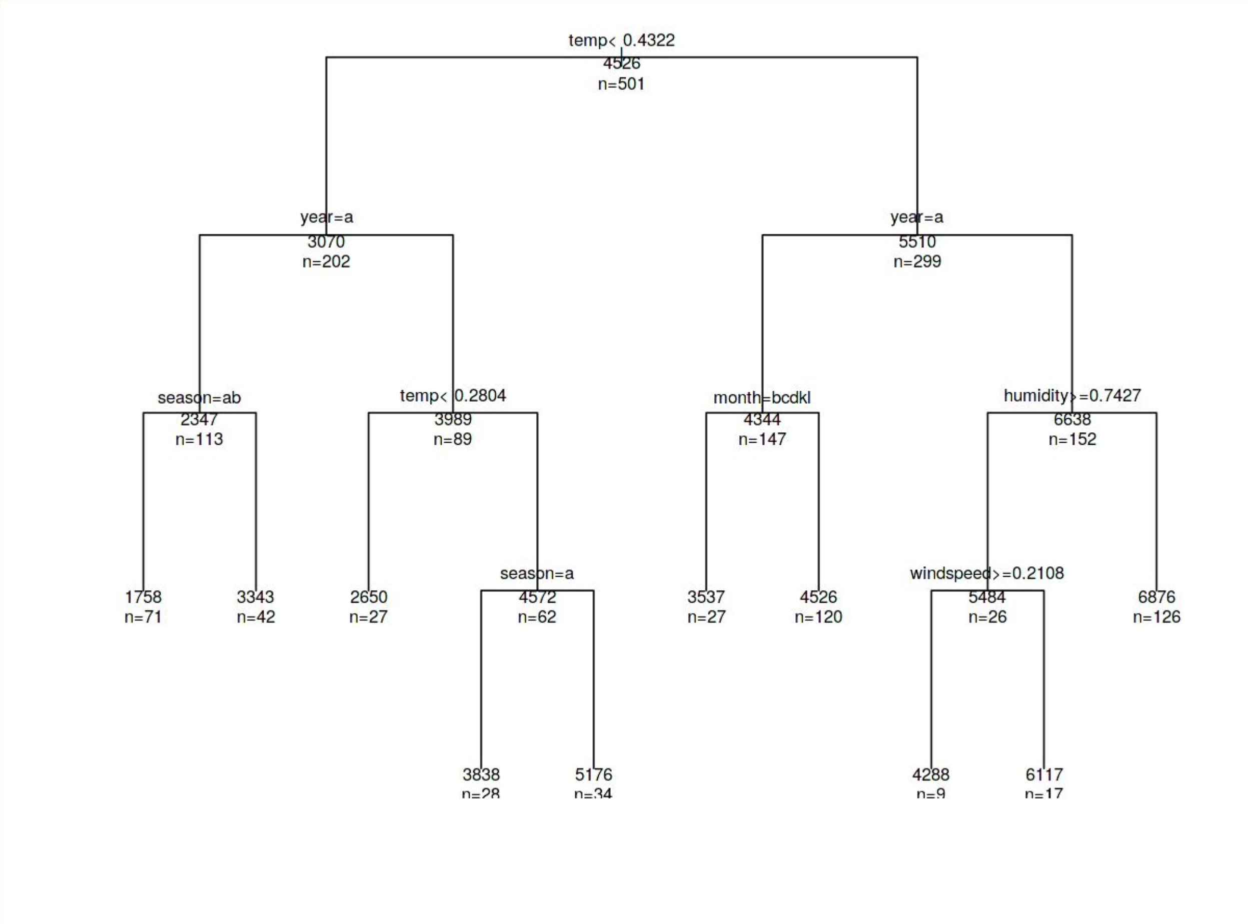
```
[2] "summary of actual count values"
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
506	3214	4582	4548	6049	8395

Variable importance

temp	month	year	season	humidity	windspeed	weekday	workingday.
29	24	18	17	5	3	2	1

Regression Trees~




```
cat("r-squared :", R2(test_set$count,predictions_dt))
```

```
 r-squared : 0.7920457
```

Metrics

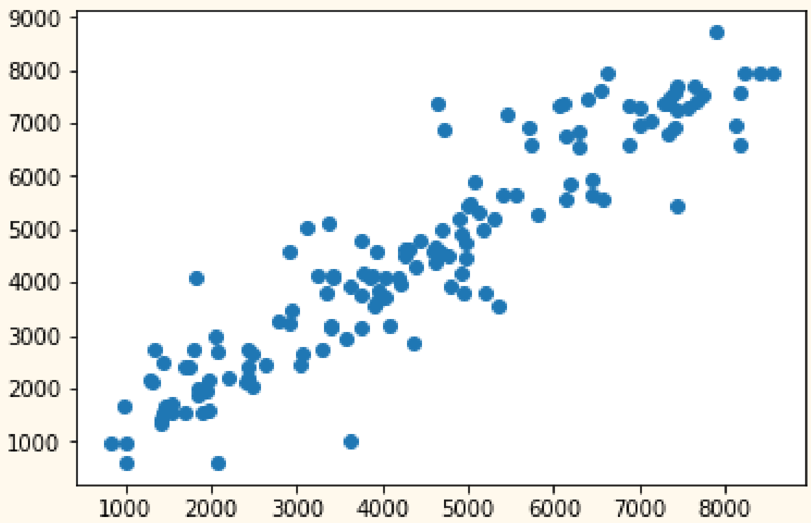
MAE	682.436371933286
MSE	844300.917110984
RMSE	918.858485900296
MAPE	0.241935747658705

Second Decision tree

```
summary of predicted counts vs actual counts
```

	0		0
count	144.000000	count	144.000000
mean	4481.090278	mean	4395.312500
std	2067.971412	std	2071.799007
min	605.000000	min	822.000000
25%	2729.000000	25%	2598.750000
50%	4335.000000	50%	4300.000000
75%	6545.250000	75%	6121.750000
max	8714.000000	max	8555.000000

Plot Predictions vs actual count~



Metrics

R2 score : 0.85
Root Mean squared error: 794.08
Mean Absolute error: 565.86
MAPE: 16.56
Accuracy: 83.44

4.3 KNN

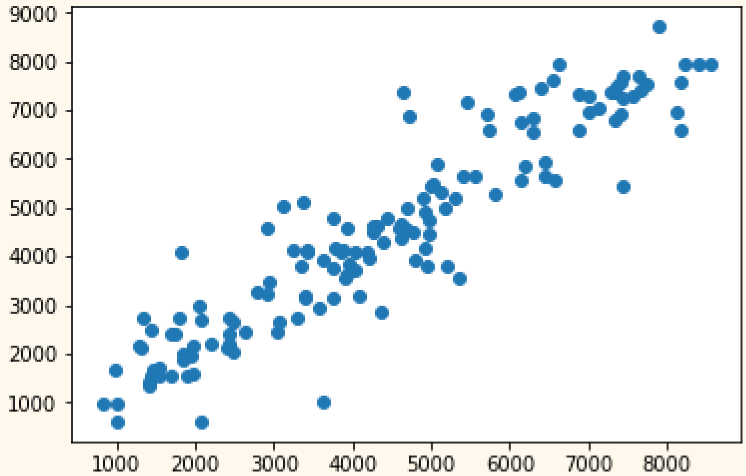
summary of predicted counts vs actual counts

	0		0
count	144.000000	count	144.000000
mean	4348.609375	mean	4395.312500
std	1914.225811	std	2071.799007
min	1067.250000	min	822.000000
25%	2770.500000	25%	2598.750000
50%	4339.750000	50%	4300.000000
75%	5813.562500	75%	6121.750000
max	7850.750000	max	8555.000000

Metrics

R2 score : 0.88
Root Mean squared error: 725.48
Mean Absolute error: 547.76
MAPE: 15.70
Accuracy: 84.30

Plot predicted vs actual count



4.4 Random Forest

Random forest is one of the most popular and most powerful machine learning algorithm. Random forest or random decision forest are an ensemble learning method for classification, regression and other tasks that operates by constructing multitude of decision trees at training time and outputting the class that is the mode of the classes(classification) and mean prediction(regression) of the individual trees .

Random Forest model 1

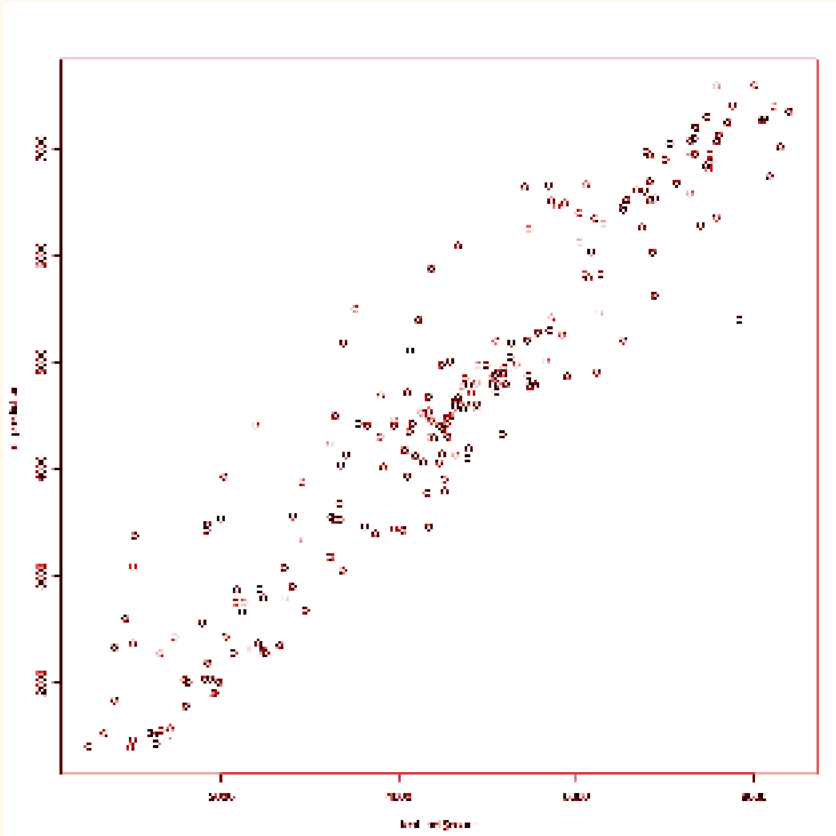
```
randomForest(formula = count ~ ., data = training_set, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 466703.3
                    % Var explained: 87.15
```

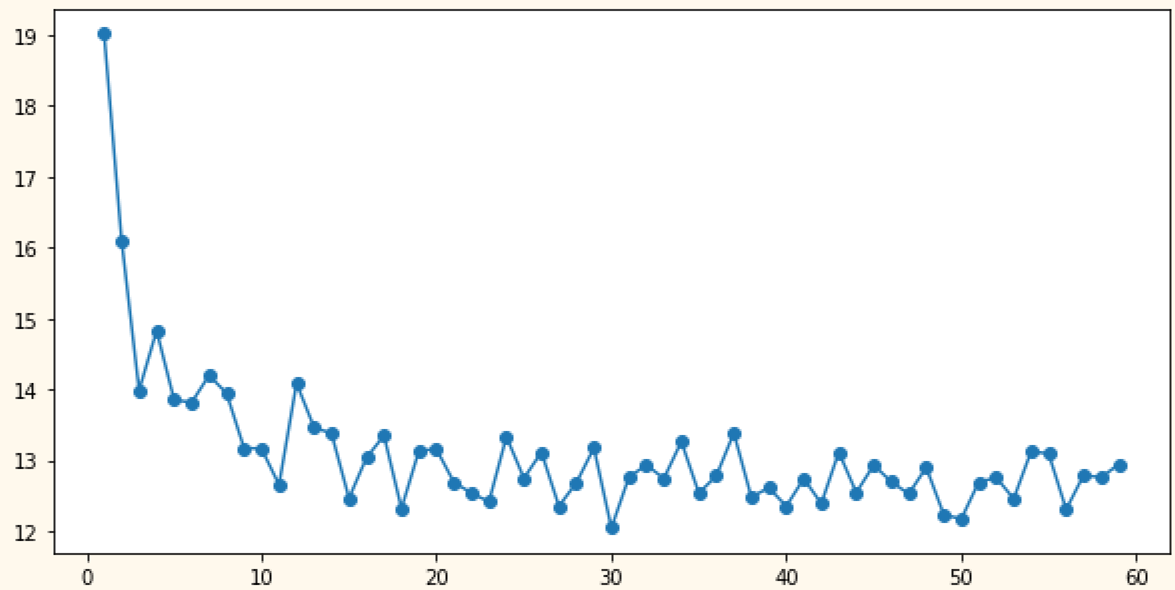
Metrics Model1

Mae	512.50486596291
Mse	494678.113566289
Rmse	703.333572045505
mape	0.1875750062866
R2	0.8903481

Predicted vs Actual count



Error rate vs N_estimators



New Random Forest model with n_estimators = 30

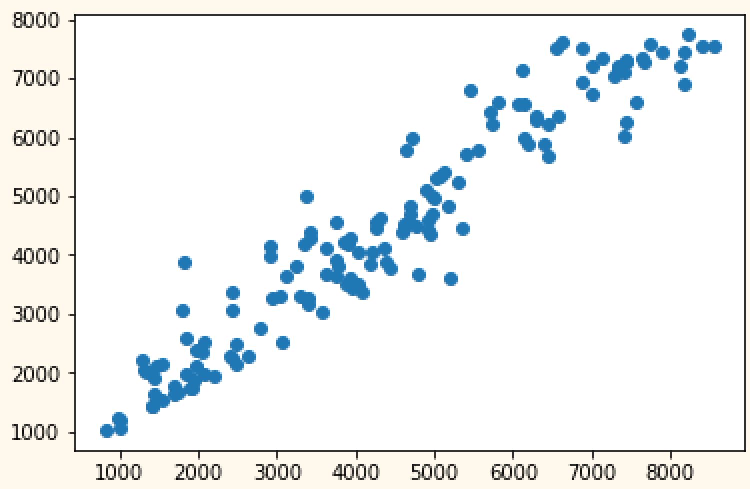
```
summary of predicted counts vs actual counts
```

	0		0
count	144.000000	**count**	144.000000
mean	4446.132176	**mean**	4395.312500
std	1919.906868	**std**	2071.799007
min	1022.833333	**min**	822.000000
25%	3048.450000	**25%**	2598.750000
50%	4330.083333	**50%**	4300.000000
75%	6225.766667	**75%**	6121.750000
max	7744.100000	**max**	8555.000000

Metrics model2

R2 score : 0.92
Root Mean squared error: 597.46
Mean Absolute error: 448.47
MAPE: 12.95
Accuracy: 87.05

Predicted vs Actual values



It seems this newer model is doing much better and is more explanatory of data.

5. Model Evaluation

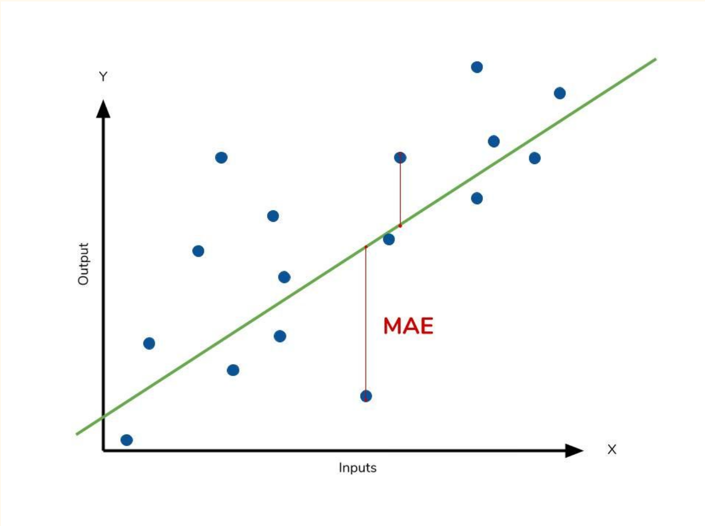
Since we Have developed different models for predicting target variables, we need to decide which one to choose. For this purpose we will create an error matrix which will be composed of different models and different error metrics.

Error Metrics to be used~

As our problem is a regression problem we will be using Regression Matrix evaluation. Following are the measure of error used in regression:

Mean Absolute Error:

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.



$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

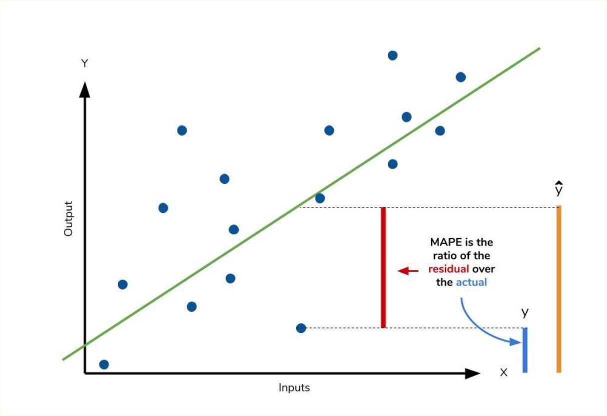
Root Mean Squared Error:

RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Mean Absolute Percentage Error(MAPE):

The mean absolute percentage error (MAPE) is the percentage equivalent of MAE. The equation looks just like that of MAE, but with adjustments to convert everything into percentages.Just as MAE is the average magnitude of error produced by your model, the MAPE is how far the model's predictions are off from their corresponding outputs on average.



$$MAPE = \frac{100\%}{n} \sum \left| \frac{y - \widehat{y}}{y} \right|$$

R-Squared Error(R2):

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

R-squared = Explained variation / Total variation

Accuracy:

This signifies how accurately our model is able to predict the target variable. In is obtained in percentage and calculated by subtracting Mean Absolute Percentage Error form 100.

100- MAPE

5.1 ERROR MATRIX

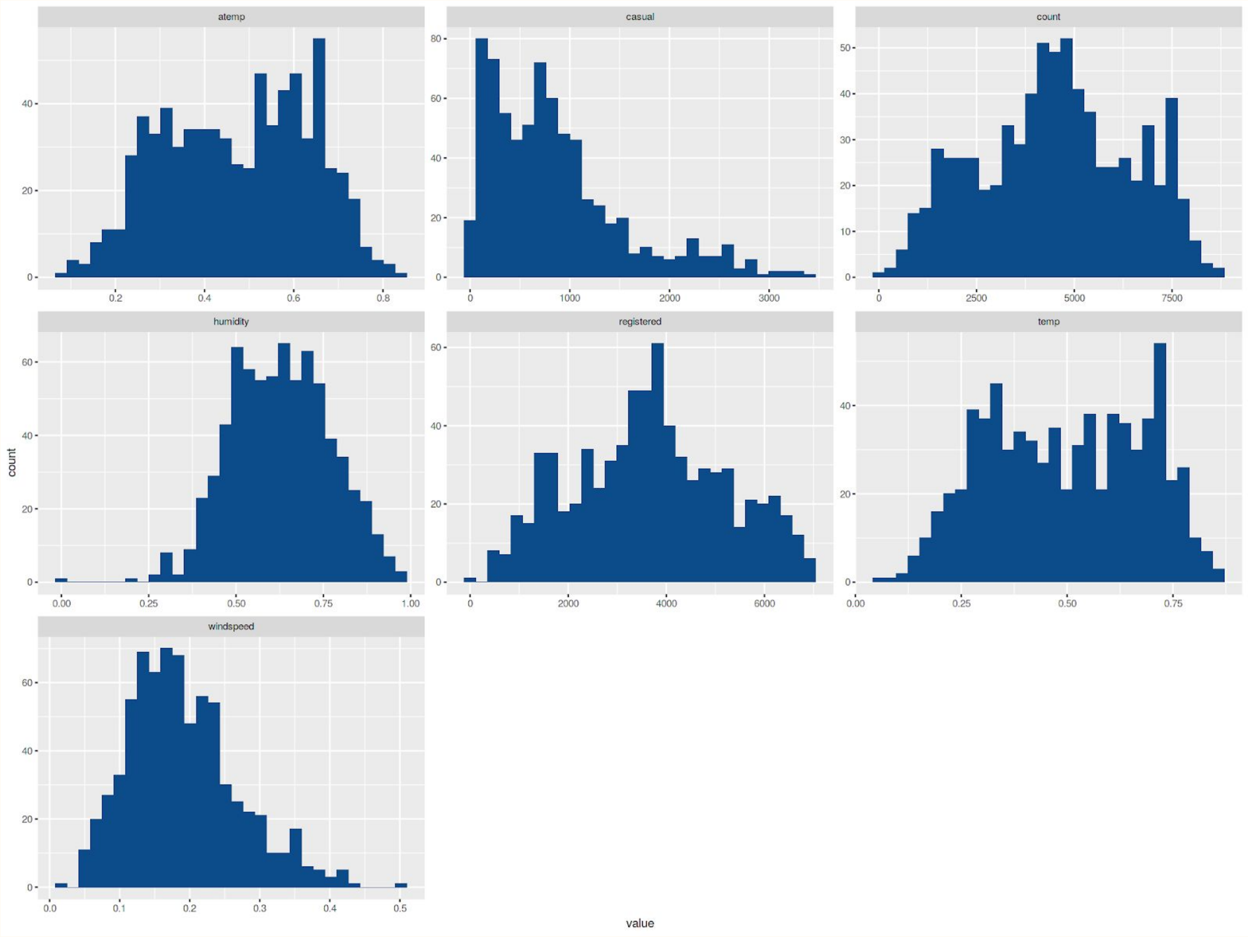
Error Metrics → Model ↓	MAE	RMSE	MAPE	R2	Accuracy
Linear Regression	586.44	747.40	17.03	0.87	82.97
Decision tree	565.86	794.08	16.56	0.85	83.44
KNN	547.56	725.48	15.70	0.88	84.30
Random Forest	448.47	597.46	12.95	0.92	87.05

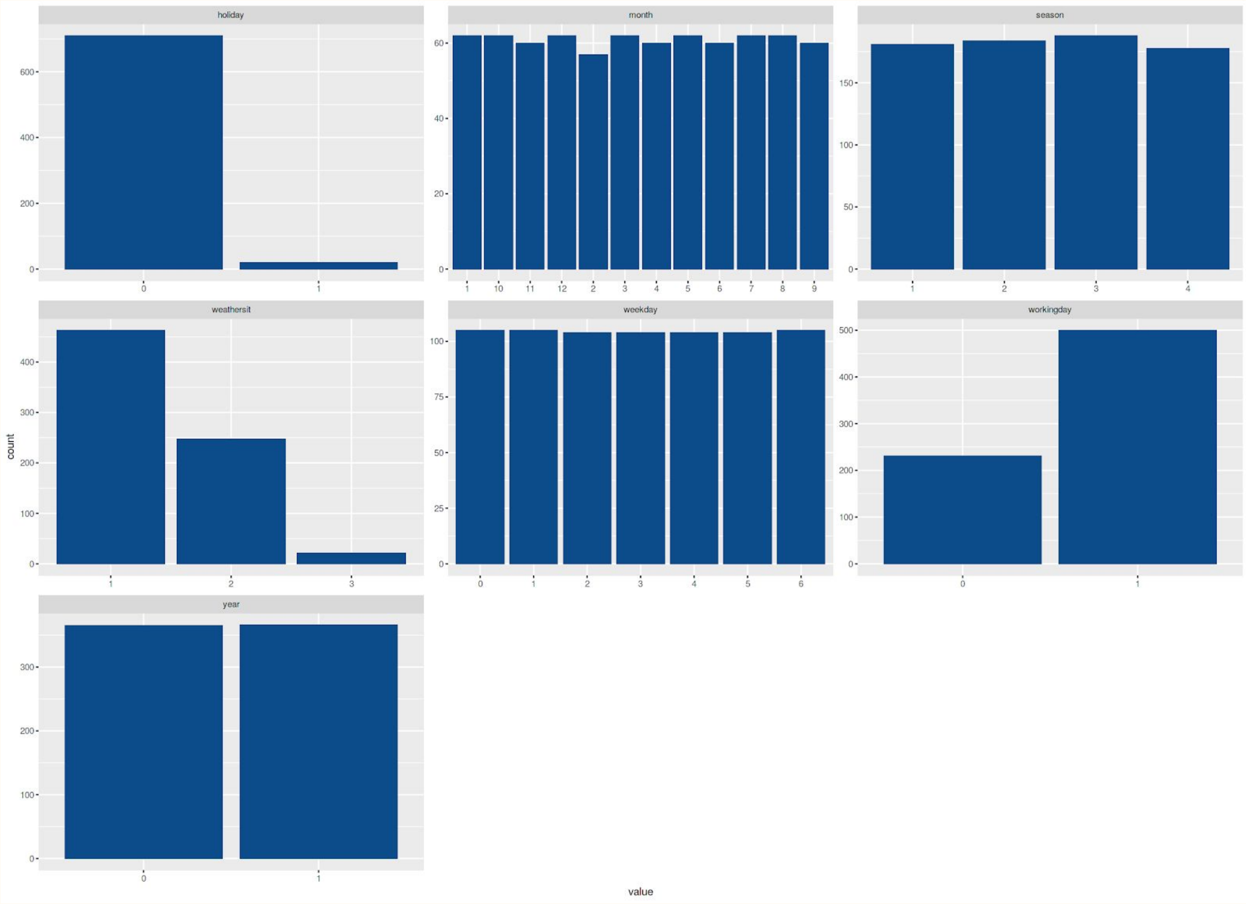
5.2 Model Selection

We will be selecting Random forest as we can see from the Error matrix that it is doing well in every Error metric.

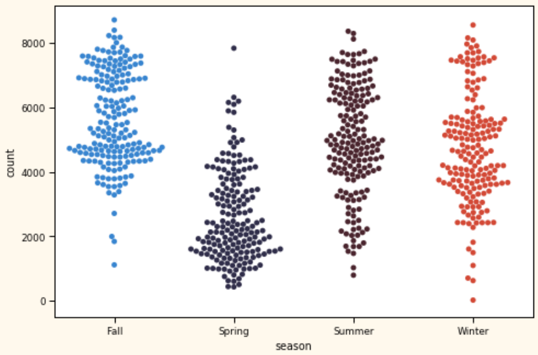
Error Metrics → Model ↓	MAE	RMSE	MAPE	R2	Accuracy
Random Forest	448.47	597.46	12.95	0.92	87.05

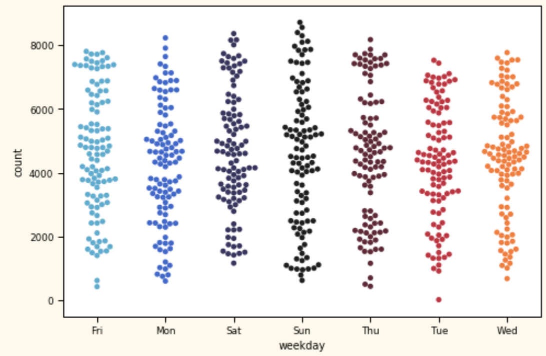
Appendix A - Extra Figures

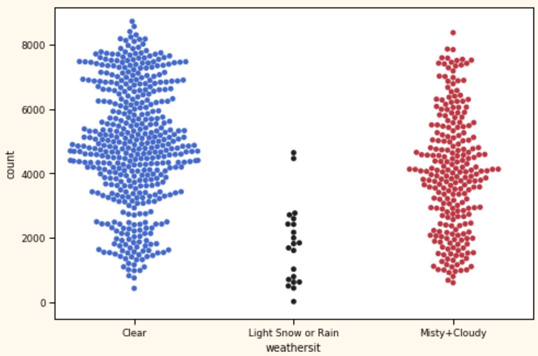


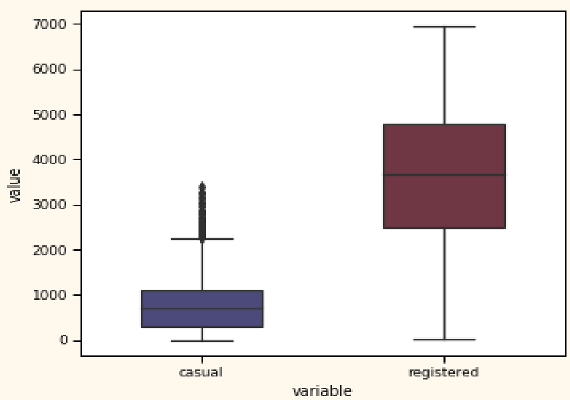


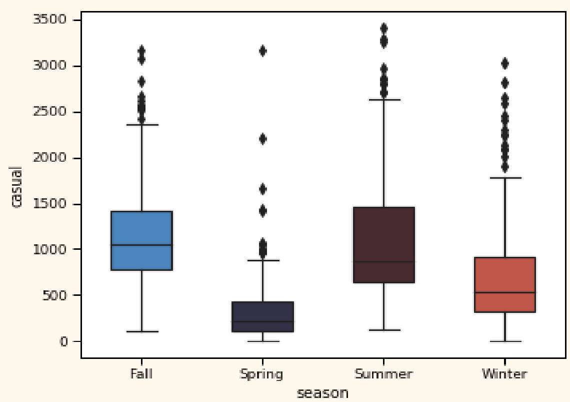
holiday
month
season
weathersit
weekday
workingday
year
count
value

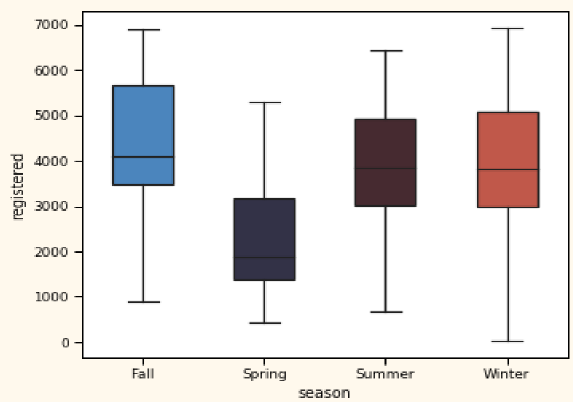

count
0
2000
4000
6000
8000
Fall
Spring
Summer
Winter
season

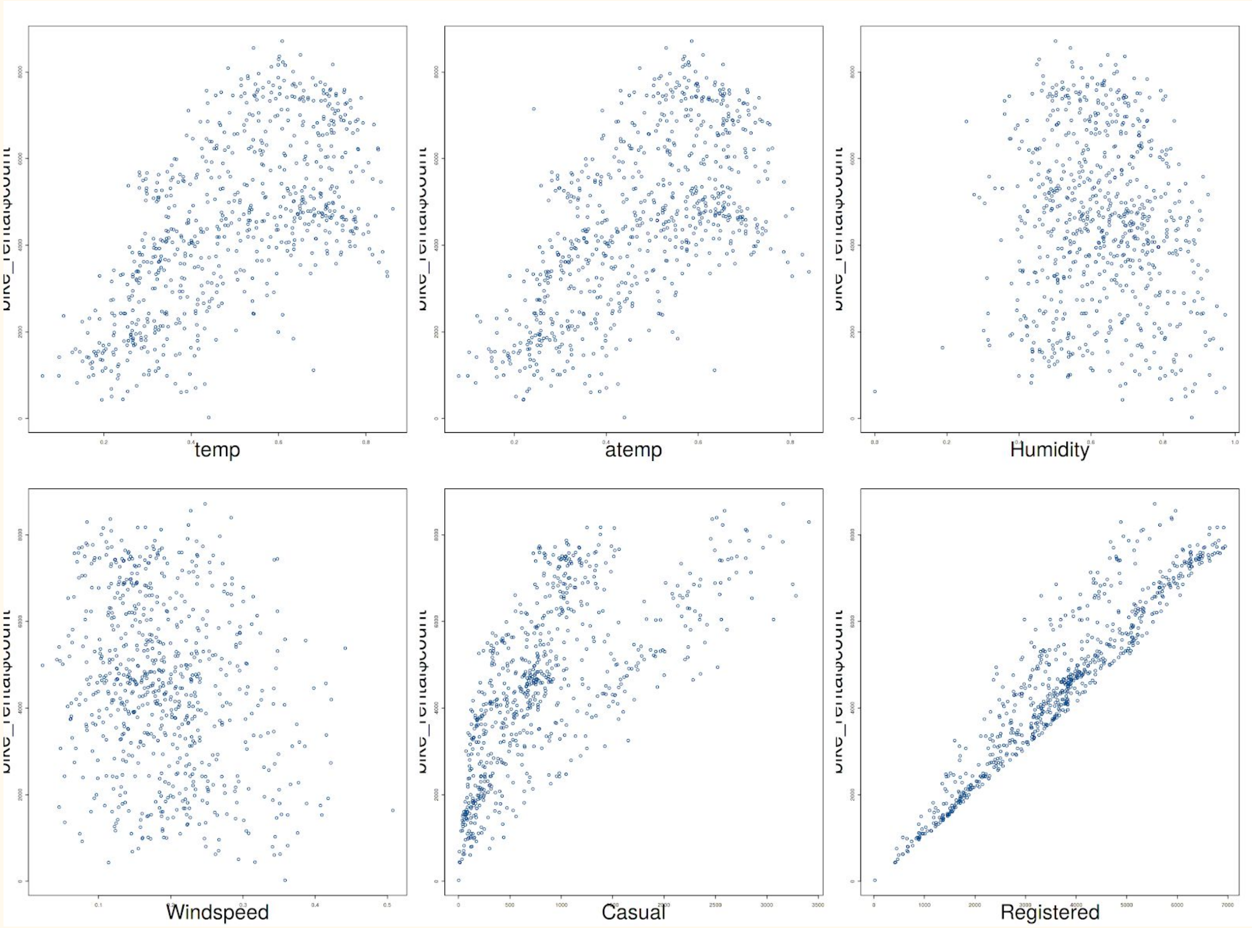

count
0
2000
4000
6000
8000
Fri
Mon
Sat
Sun
Thu
Tue
Wed
weekday

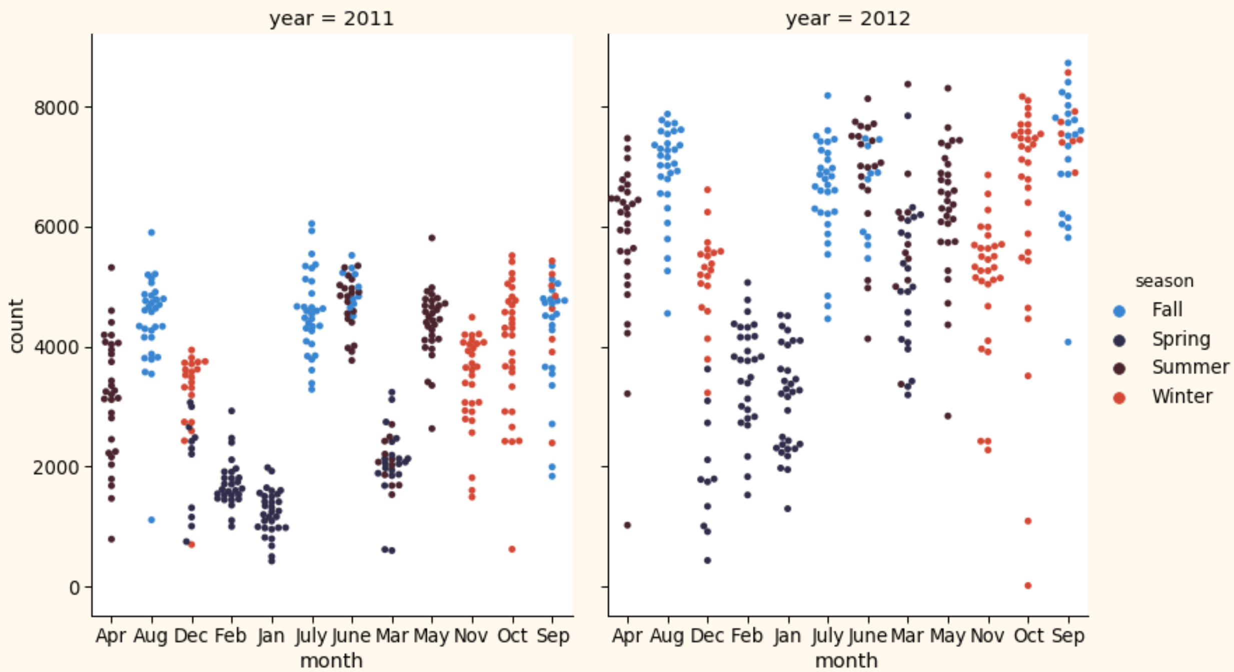

count
0
2000
4000
6000
8000
Clear
Light Snow or Rain
Misty+Cloudy
weathersit

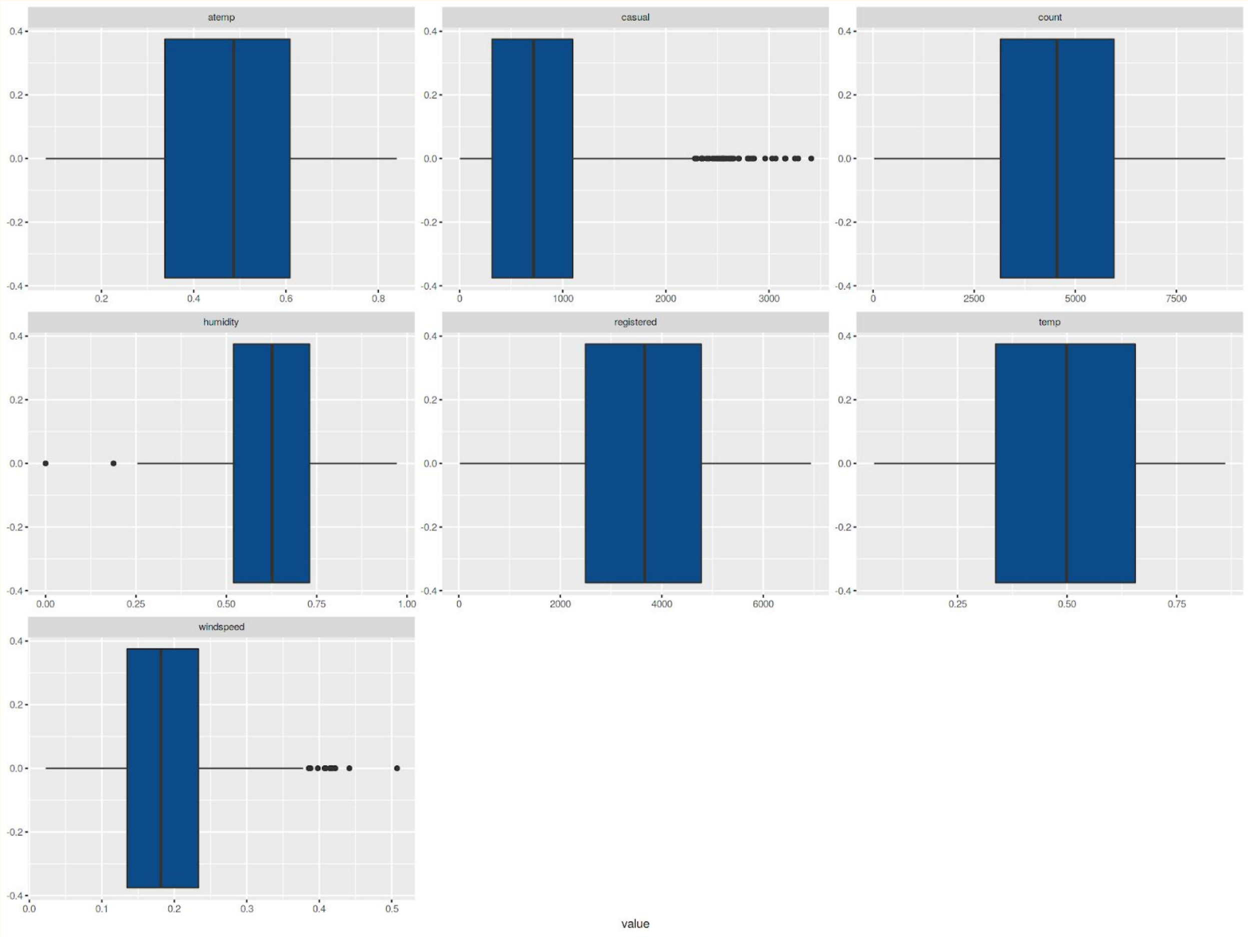

value
0
1000
2000
3000
4000
5000
6000
7000
casual
registered
variable

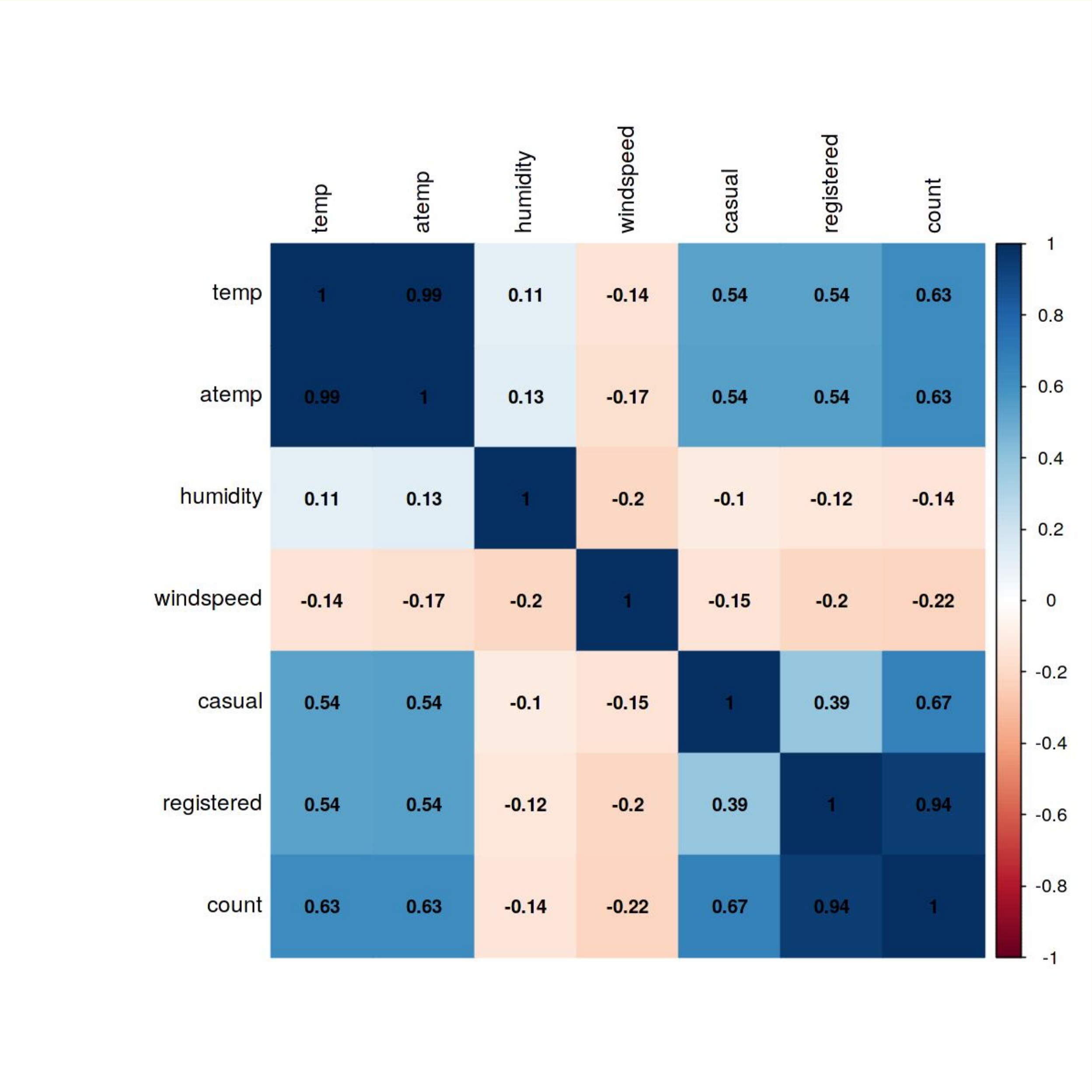

casual
0
500
1000
1500
2000
2500
3000
3500
Fall
Spring
Summer
Winter
season


registered
0
1000
2000
3000
4000
5000
6000
7000
Fall
Spring
Summer
Winter
season


temp
atemp
Humidity
Windspeed
Casual
Registered


year = 2011
year = 2012
count
8000
6000
4000
2000
0
Apr Aug Dec Feb Jan July June Mar May Nov Oct Sep
month
season
Fall
Spring
Summer
Winter


atemp
casual
count
humidity
registered
temp
windspeed
value


temp
atemp
humidity
windspeed
casual
registered
count
1
0.99
0.11
-0.14
0.54
0.54
0.63
0.13
-0.17
-0.2
-0.1
-0.12
-0.15
-0.22
0.39
0.67
0.94
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1

Appendix B - R Code

```
rm(list = ls())

library(purrr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(caret)
library(rpart)
library(MASS)
library(DMwR)

bike_rental = read.csv("../input/bikerental/day.csv")

head(bike_rental)
dim(bike_rental)
str(bike_rental)
dim(bike_rental)
summary(bike_rental)

# %% [markdown]
# # Pre-Processing

# %% [markdown]
#          Removing the instant variable, as it is index in datasets
#          Removing date variable as we have to predict count on seasonal basis not date basis-
#
bike_rental= subset(bike_rental,select=-(instant))
bike_rental= subset(bike_rental,select=-(dteday))
names(bike_rental)

names(bike_rental)[names(bike_rental) == "yr"] <- "year"
names(bike_rental)[names(bike_rental) == "mnth"] <- "month"
names(bike_rental)[names(bike_rental) == "hum"] <- "humidity"
names(bike_rental)[names(bike_rental) == "cnt"] <- "count"

head(bike_rental)

# %% [markdown]
#    ### Missing values

```

```

sum(is.na(bike_rental))

# %% [markdown]
#     Changing data types of variable

bike_rental[,1:7]= lapply (bike_rental[, 1:7], as.factor)
bike_rental[,8:14]= lapply (bike_rental[, 8:14], as.numeric)

# %% [markdown]
# # EDA

fig <- function(width, heigth){
     options(repr.plot.width = width, repr.plot.height = heigth)
}

fig(16,12)
bike_rental %>%
  keep(is.numeric) %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_histogram(bins = 30L, fill = "#0c4c8a")

fig(18,13)
bike_rental %>%
  keep(is.factor) %>%
  gather() %>%
  ggplot(aes(value)) +
    facet_wrap(~ key, scales = "free") +
    geom_bar( fill = "#0c4c8a")


fig(24,20)
par(mfrow=c(3,3))

plot(x = bike_rental$season, y = bike_rental$count,cex.lab=4, xlab =
"season",col = "grey")
plot(x = bike_rental$month, y = bike_rental$count,cex.lab=4, xlab = "month",col
= "grey")
plot(x = bike_rental$weathersit, y = bike_rental$count,cex.lab=4, xlab =
"Weather",col = "grey")
plot(x = bike_rental$holiday, y = bike_rental$count,cex.lab=4, xlab =
"Holiday",col = "grey")
```

```
plot(x = bike_rental$weekday, y = bike_rental$count,cex.lab=4, xlab = "Weekday",col = "grey")
plot(x = bike_rental$year, y = bike_rental$count,cex.lab=4, xlab = "Year",col = "grey")
plot(x = bike_rental$workingday, y = bike_rental$count,cex.lab=4, xlab = "Working day",col = "grey")
#Double Click to magnify

fig(27,20)
par(mfrow=c(2,3))

plot(x = bike_rental$temp, y = bike_rental$count,cex.lab=4, xlab = "temp",col = "#0c4c8a")
plot(x = bike_rental$atemp, y = bike_rental$count,cex.lab=4, xlab = "atemp",col = "#0c4c8a")
plot(x = bike_rental$humidity, y = bike_rental$count, cex.lab=4 , xlab="Humidity",col = "#0c4c8a")
plot(x = bike_rental$windspeed, y = bike_rental$count, cex.lab=4, xlab = "Windspeed",col = "#0c4c8a")
plot(x = bike_rental$casual, y = bike_rental$count, cex.lab=4, xlab = "Casual",col = "#0c4c8a")
plot(x = bike_rental$registered, y = bike_rental$count, cex.lab=4, xlab = "Registered",col = "#0c4c8a")


# %% [markdown]
# # Outliers
fig(16,12)
  bike_rental %>%
    keep(is.numeric) %>%
    gather() %>%
    ggplot(aes(value)) +
      facet_wrap(~ key, scales = "free") +
      geom_boxplot(bins = 30L, fill = "#0c4c8a")

  # Outlier Removal
  outlier_variables = c("humidity","windspeed")
  for(i in outlier_variables){
    val = bike_rental[,i][bike_rental[,i] %in% boxplot.stats(bike_rental[,i])$out]
    print(length(val))
    bike_rental[,i][bike_rental[,i] %in% val] = NA
  }

```

```


# Checking Missing data - after outlier
apply(bike_rental, 2, function(x) {sum(is.na(x))})
bike_rental = drop_na(bike_rental)

# %% [markdown]
# # Correlation analysis
library(corrplot)
fig(10,10)
par(mfrow = c(1, 1))
num_vars <- names(bike_rental[,8:14])
numVarDataset <- bike_rental[, num_vars]
corr <- cor(numVarDataset)
corrplot(
  corr,
  method = "color",
  rect.col = "black",
  tl.col = "black",
  addCoef.col = "black",
  number.digits = 2,
  number.cex = 1,
  tl.cex = 1.2,
  cl.cex = 1,
)
install.packages("GoodmanKruskal")
library("GoodmanKruskal")
fig(10,10)
factor_index = sapply(bike_rental,is.factor)
factor_data = bike_rental[,factor_index]
plot(GKtauDataframe( factor_data), corrColors = 'blue')

factor_index = sapply(bike_rental,is.factor)
factor_data = bike_rental[,factor_index]

for (i in 1:7)
{
  print(names(factor_data)[i])
  print(chisq.test(table(factor_data$season,factor_data[,i])))
}



bike_rental= subset(bike_rental,select=-(casual))
```

```r
bike_rental= subset(bike_rental,select=-(registered))
bike_rental = subset(bike_rental, select = -(atemp))
names(bike_rental)
library(caTools)
library(rcompanion)
library(mlr)
library(MASS)
library(Metrics)
library(randomForest)
set.seed(654)
split <- sample.split(bike_rental$count, SplitRatio = 0.70)
training_set <- subset(bike_rental, split == TRUE)
test_set <- subset(bike_rental, split == FALSE)

# %% [markdown]
# ### Linear Model
model_lr <- lm(count ~ ., data = training_set)

summary(model_lr)

# Apply prediction on test set
test_prediction <- predict(model_lr, newdata = test_set)

summary(test_prediction)
print("summary of actual count values")
summary(test_set$count)

cat("r-squared :", R2(test_set$count,test_prediction)
regr.eval(test_set$count,test_prediction)

fig(10,8)
plot(test_set$count ,test_prediction)

# %% [markdown]
# ### Decision Tree Model
model_dt = rpart(count ~ ., data=training_set, method = "anova")

# summary on trained model
summary(model_dt)

#Prediction on test_data
predictions_dt = predict(model_dt, test_set)
```

```
printcp(model_dt)
plot(model_dt, uniform=TRUE,
   main="Regression Tree  ")
text(model_dt, use.n=TRUE, all=TRUE, cex=.8)
summary(predictions_dt)
print("summary of actual count values")
summary(test_set$count)


cat("r-squared :", R2(test_set$count,predictions_dt))


regr.eval(test_set$count,predictions_dt)


plot(test_set$count ,predictions_dt)

# %% [markdown]
# ### Random Forest
model_rf <- randomForest(count ~.,
                           data = training_set, importance = TRUE)
print(model_rf)

varImpPlot(model_rf)

rf_prediction <- predict(model_rf,test_set)
cat("r-squared :", R2(test_set$count,rf_prediction))
regr.eval(test_set$count,rf_prediction)

tuned_randomForest <-  randomForest(count ~.,
                                    data = training_set,ntree = 30, mtry = 6,
importance = TRUE)
tuned_randomForest
rf1_prediction <- predict(tuned_randomForest,test_set)

cat("r-squared :", R2(test_set$count,rf1_prediction))
regr.eval(test_set$count,rf1_prediction)

fig(18,9)
par(mfrow=c(1,2))
plot(test_set$count ,rf_prediction)
plot(test_set$count ,rf1_prediction)

```

Appendix C - Python Code

```
#!/usr/bin/env python
# coding: utf-8

# <h1 align="center" style="background-color:MediumSeaGreen; font-size:48px" ><br> Bike Rental Modeling <br></h1>

# In[2]:


import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
from sklearn.linear_model import LinearRegression

get_ipython().run_line_magic('matplotlib', 'inline')


# In[3]:


train = pd.read_csv('/home/woodman/Downloads/bike_rental_clean1.csv')


# In[4]:


train = train.drop(["Unnamed: 0"], axis = 1)


# In[5]:


train.shape


# In[6]:


train.info()


# In[7]:


train.head()


# In[8]:

```

```python

# Converting into proper datatype
train['season'] = train.season.astype('category')
train['month'] = train.month.astype('category')
train['workingday'] = train.workingday.astype('category')
train['weathersit'] = train.weathersit.astype('category')
train['year'] = train.year.astype('category')
train.dtypes


# Encoding Categorical variables since all the categorical variables are nominal.

# In[9]:


cat_names = ['season', 'year', 'month','weathersit']

bkr_enc = pd.get_dummies(train,columns=cat_names)


# In[11]:


bkr_enc.info()


# In[12]:


bkr_enc.columns


# In[13]:


#Calculate MAPE
def MAPE(y_true, y_pred):
    mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
    return mape


# In[14]:


bkr_enc.info()


# In[15]:


Y = bkr_enc["count"].values
X = bkr_enc.drop("count", axis = 1).values
```

```python

#Splitting the data in train and testing

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 1)


# In[16]:


model_lr = LinearRegression().fit(X_train , y_train)


# In[17]:


# predict on train data
pred_train_LR = model_lr.predict(X_train)


# In[18]:


pred_test_LR = model_lr.predict(X_test)


# In[34]:


from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:inline"'),raw=True)


# In[37]:


print("summary of predicted counts vs actual counts")
df_describe_predicted = pd.DataFrame(pred_test_LR)
df_describe_actual = pd.DataFrame(y_test)
display_side_by_side(df_describe_predicted.describe(), df_describe_actual.describe())


# In[21]:


# Results on Training data
print("R2 score : %.2f" % r2_score(y_train, pred_train_LR))
print("Root Mean Squared error: %.2f" % np.sqrt(mean_squared_error(y_train, pred_train_LR)))
```

```python
# Results on Training data
print("R2 score : %.2f" % r2_score(y_train, pred_train_LR))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_train, pred_train_LR)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_train, pred_train_LR))
print("MAPE: %.2f" % MAPE(y_train, pred_train_LR))
print("Accuracy: %.2f" % (100-MAPE(y_train, pred_train_LR)))


# In[23]:


#results on Testing data
print("R2 score : %.2f" % r2_score(y_test, pred_test_LR))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, pred_test_LR)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, pred_test_LR))
print("MAPE: %.2f" % MAPE(y_test, pred_test_LR))
print("Accuracy: %.2f" % (100-MAPE(y_test, pred_test_LR)))


# In[38]:


plt.scatter(y_test,pred_test_LR)


# ## Decision Tree

# In[39]:


train.columns


# In[45]:


model_dt = DecisionTreeRegressor()


# In[46]:


model_dt.fit(X_train,y_train)


# In[49]:


predictions_dt = model_dt.predict(X_test)
```

```python
print("summary of predicted counts vs actual counts")
df_describe_predicted = pd.DataFrame(predictions_dt)
df_describe_actual = pd.DataFrame(y_test)
display_side_by_side(df_describe_predicted.describe(), df_describe_actual.describe())


# In[50]:


# metrics on test
print("R2 score : %.2f" % r2_score(y_test, predictions_dt))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions_dt)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions_dt))
print("MAPE: %.2f" % MAPE(y_test, predictions_dt))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions_dt)))


# In[63]:


plt.scatter(y_test,predictions_dt)


# # KNN

# In[64]:


from sklearn.neighbors import KNeighborsRegressor


# In[65]:


# function to increment n_neighbors one by one and calculate error rate..

error_rate=[]
for i in range(1,40):
    knn =KNeighborsRegressor(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i=knn.predict(X_test)
    error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)


# In[66]:


# plot of error rate by the function
plt.figure(figsize=(8,4))
plt.plot(range(1,40),error_rate,marker='o')
```

```
model_knn = KNeighborsRegressor(n_neighbors = 4)


# In[70]:


model_knn.fit(X_train, y_train)


# In[71]:


predictions_knn = model_knn.predict(X_test)


# In[85]:


print("summary of predicted counts vs actual counts")
df_describe_predicted = pd.DataFrame(predictions_knn)
df_describe_actual = pd.DataFrame(y_test)
display_side_by_side(df_describe_predicted.describe(), df_describe_actual.describe())


# In[72]:


print("R2 score : %.2f" % r2_score(y_test, predictions_knn))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions_knn)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions_knn))
print("MAPE: %.2f" % MAPE(y_test, predictions_knn))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions_knn)))


# In[73]:


plt.scatter(y_test,predictions)


# ## Random Forest

# In[55]:


from sklearn.ensemble import RandomForestRegressor


# In[116]:


# function to increment n_estimators one by one and calculate error rate..
error_rate=[]
```

```
# function to increment n_estimators one by one and calculate error rate..
error_rate=[]
for i in range(1,60):
    ran =RandomForestRegressor(n_estimators=i)
    ran.fit(X_train,y_train)
    pred_i=ran.predict(X_test)
    error_rate.append(np.mean(np.abs((y_test - pred_i) / y_test))*100)


# In[117]:


# plot of error rate
plt.figure(figsize=(10,5))
plt.plot(range(1,60),error_rate,marker='o')


# In[118]:


model_rf = RandomForestRegressor(n_estimators =30).fit(X_train,y_train)


# In[119]:


predictions_rf = model_rf.predict(X_test)


# In[120]:


print("summary of predicted counts vs actual counts")
df_describe_predicted = pd.DataFrame(predictions_rf)
df_describe_actual = pd.DataFrame(y_test)
display_side_by_side(df_describe_predicted.describe(), df_describe_actual.describe())


# In[121]:


print("R2 score : %.2f" % r2_score(y_test, predictions_rf))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, predictions_rf)))
print("Mean Absolute error: %.2f" % mean_absolute_error(y_test, predictions_rf))
print("MAPE: %.2f" % MAPE(y_test, predictions_rf))
print("Accuracy: %.2f" % (100-MAPE(y_test, predictions_rf)))


# In[122]:


plt.scatter(y_test,predictions_rf)
```

References

https://edwisor.com/
https://stackoverflow.com