Bike Renting

Project by: Abhishek Pandey

Linkedin:
https://www.linkedin.com/in/abhishekpandey2505/

Email: abhishekpandey2505@gmail.com

Contents:

Chapter 1

Introduction

1.1 Problem Statement

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

1.2 Data

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)
hr: Hour (0 to 23)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
weathersit: (extracted fromFreemeteo)
1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + 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 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

	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

Chapter 2

Methodology

Pre-Processing:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. In this project we look at the distribution of categorical variables and continuous variables. We also look at the missing values in the data and the outliers present in the data

2.1>Exploratory data analysis

Here we are analyzing the data.

Just by looking at the data, we can make an initial hypothesis that is

:: **INITIAL HYPOTHSESIS** ::

Here are some of the hypothesis which we can initially make:

Season: We have four seasons here (1:springer, 2:summer, 3:fall, 4:winter) , we can't say how people will behave there ,so let's assume there are similar demands in all seasons.

Year: There must be increase in demand in next year (company must have gotten popular, just initial hypothesis).

Month: There must be more demand in months where condition is favorable.

Holiday: if people are using it for commute, the demand will be less on holiday.

weekday: if people are using it for commute, the demand will less on weekend.

Weathersit: if the situation is favorable, then the demand will be high (if it's raining, the demand will be less)

Temperature: If temp in that area is cold, then as temp increase people will come out and can use bikes as the day would be more favorable.

Humidity, windspeed: Here also the condition is needed as favorable.

Registered and Casual: The use of bikes by registered users would be more as compared to casual.

These are some hypothesis we can make just by trying to understand the business problem and looking at the factors which are influencing the business

2.1.1> Understanding the data

```
df.head()
```

	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

```
df.shape
```

```
(731, 16)
```

>> renaming the columns

```
#lets rename some columns

df = df.rename(columns = {'dteday':'dateday','mnth': 'month' , 'yr':'year','cnt':'count'})
```

>>checking the data types of column

```
#checking the data types of df
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dateday       731 non-null object
season        731 non-null int64
year          731 non-null int64
month         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
count         731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

>>changing the data types of columns

Because here some categorical columns have types as int.

```
#lets chnage the type of the columns
df['dateday'] = pd.to_datetime(df['dateday'],yearfirst=True)
df['season'] = df['season'].astype('category')
df['year'] = df['year'].astype('category')
df['month'] = df ['month'].astype('category')
df['holiday'] = df['holiday'].astype('category')
df['weekday'] = df['weekday'].astype('category')
df['workingday'] = df['workingday'].astype('category')
df['weathersit'] = df['weathersit'].astype('category')

df['temp'] = df['temp'].astype('float')
df['atemp'] = df['atemp'].astype('float')
df['hum'] = df['hum'].astype('float')
df['windspeed'] = df['windspeed'].astype('float')
df['casual'] = df['casual'].astype('float')
df['registered'] = df['registered'].astype('float')
df['count'] = df['count'].astype('float')
```

2.1.1>>UNIVARIATE ANALYSIS:

>> skewness of numerical columns

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.

Kurtosis is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. That is, data sets with high kurtosis tend to have heavy tails, or outliers. Data sets

with low kurtosis tend to have light tails, or lack of outliers.

```
#checking the skewness(symmetry) of the columns
df.skew()
```

```
temp          -0.054521
atemp         -0.131088
hum           -0.069783
windspeed      0.677345
casual         1.266454
registered     0.043659
count         -0.047353
dtype: float64
```

the skew limit is -1 to 1 for a distribution to be normal, so from numerical variables temp, atemp , hum, windspeed, casual , registered , count .
The casual (1.266) is highly skewed and others are moderately skewed .

So it means that there might be chance of having outliers in casual

>> kurtness of numerical columns

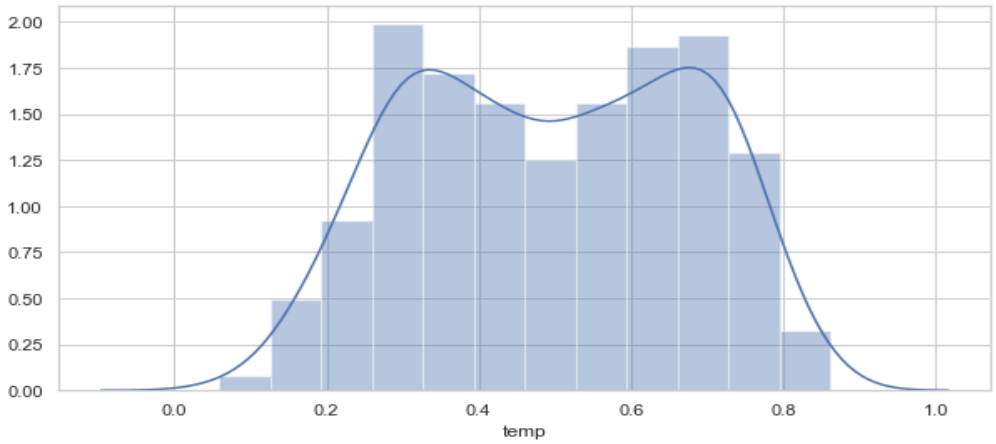
```
df.kurt()
```

```
temp          -1.118864
atemp         -0.985131
hum           -0.064530
windspeed      0.410922
casual         1.322074
registered    -0.713097
count         -0.811922
dtype: float64
```

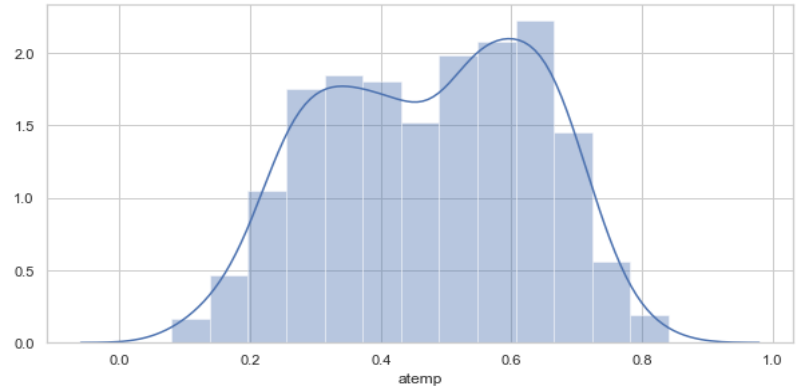
High kurtosis, rather than being a measure of "flatness," indicates a distribution that is more outlier-prone than is the normal distribution. Here we can expect outliers in casual.

>> Plotting all the numerical columns to see the graph

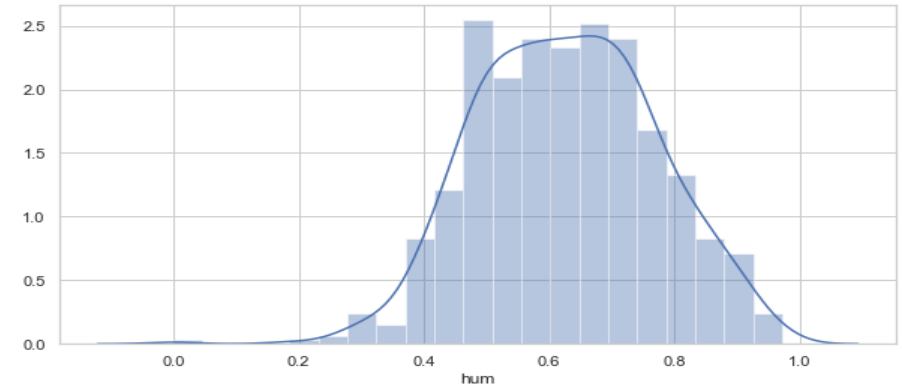
temp is close to normal distribution as it is showing symmetrical curve



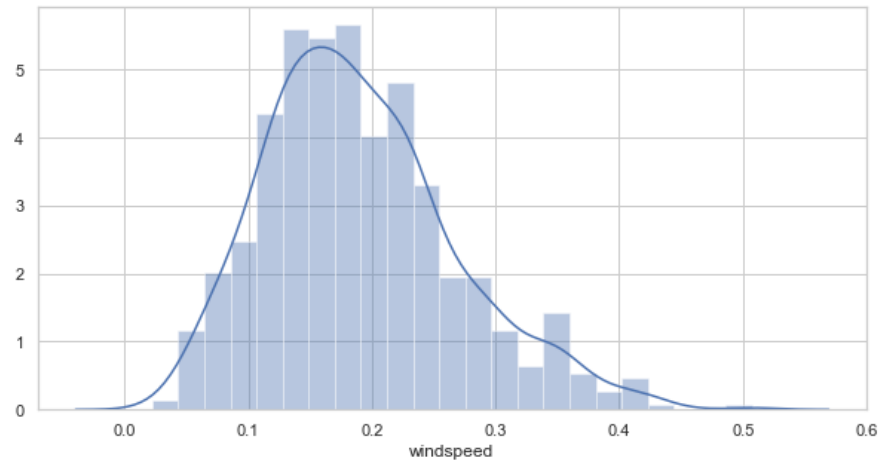
temp is close to normal distribution as it is showing symmetrical curve



#atemp is close to normal distribution as it is showing symmetrical curve

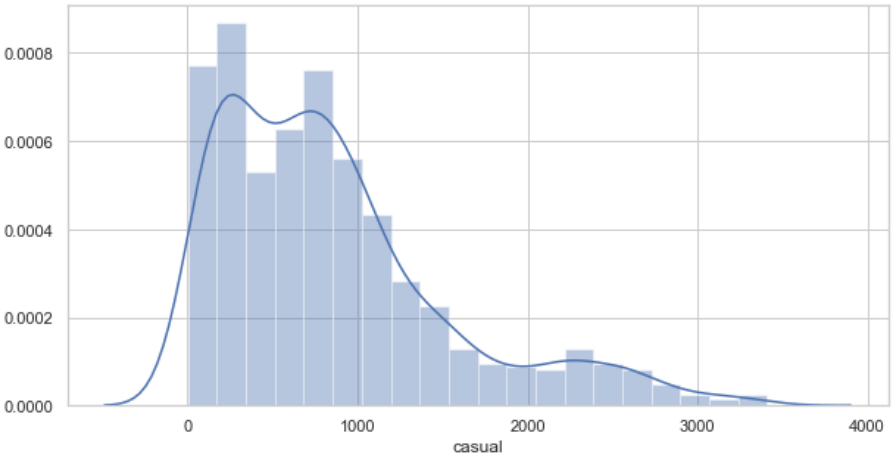


hum is close to normal distribution as it is showing symmetrical curve

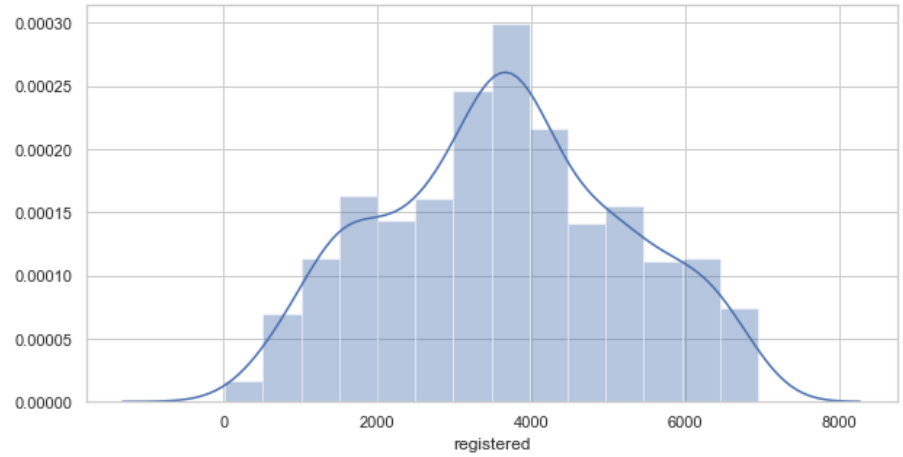


windspeed is close to normal distribution but it has some skewness as it somewhat right skewed

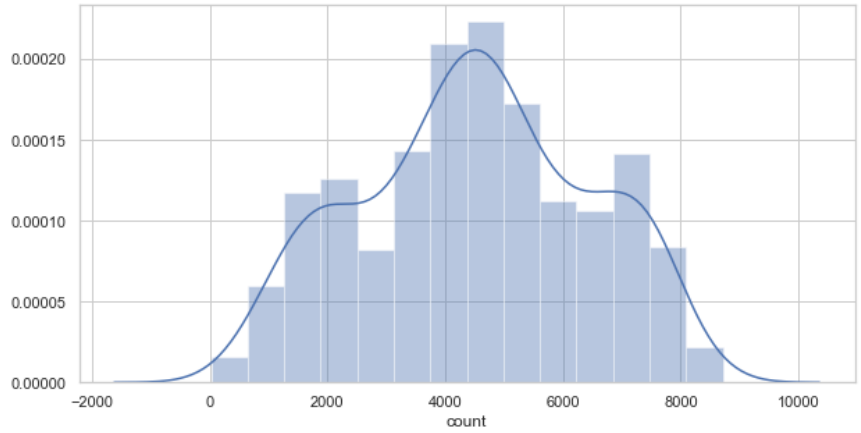
---



casual is not normally distributed as it not symmetrical at all



registered is close to normal distribution as it is showing symmetrical curve



count is close to normal distribution as it is showing symmetrical curve

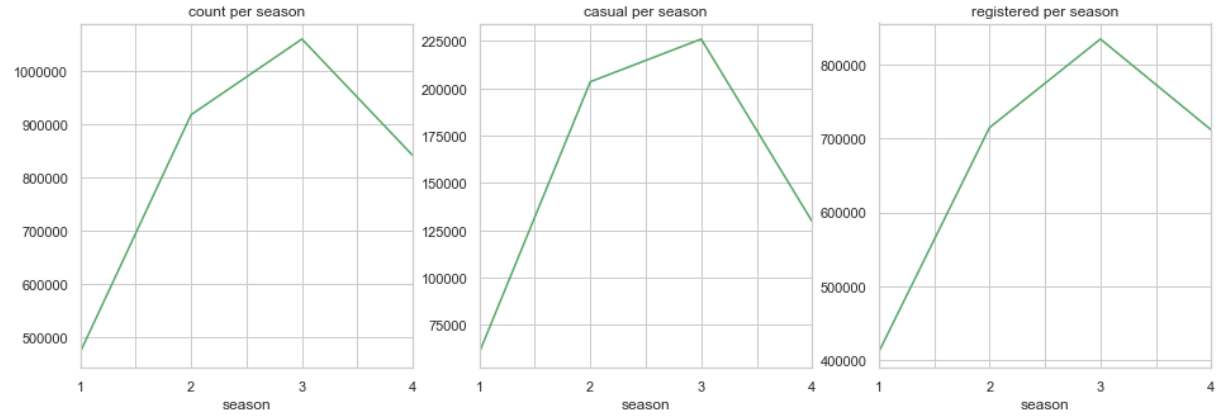
2.1.2>>BIVARIATE ANALYSIS:

>>CATEGORICAL COLUMNS WITH THE TARGET VARIABLE

cat_colnames = ['season', 'year','month','holiday', 'weekday','workingday','weathersit']

>> we are plotting the bike rented by casual users, registered user and total count of user with respect to each categorical column

>> Season

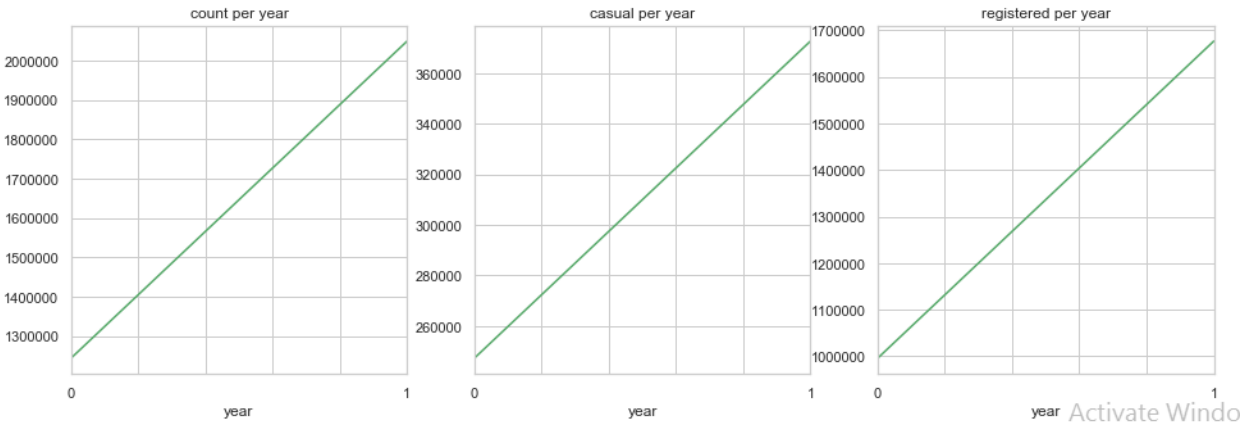


#casual user increases with season 2 and after that in season 3 it goes to top and it is minimum in season 1

#registered user also increases in season and after that goes to top in season 3

#count is sum of casual and registered, so it also shows same behavior, it also goes on top in season3

>>year

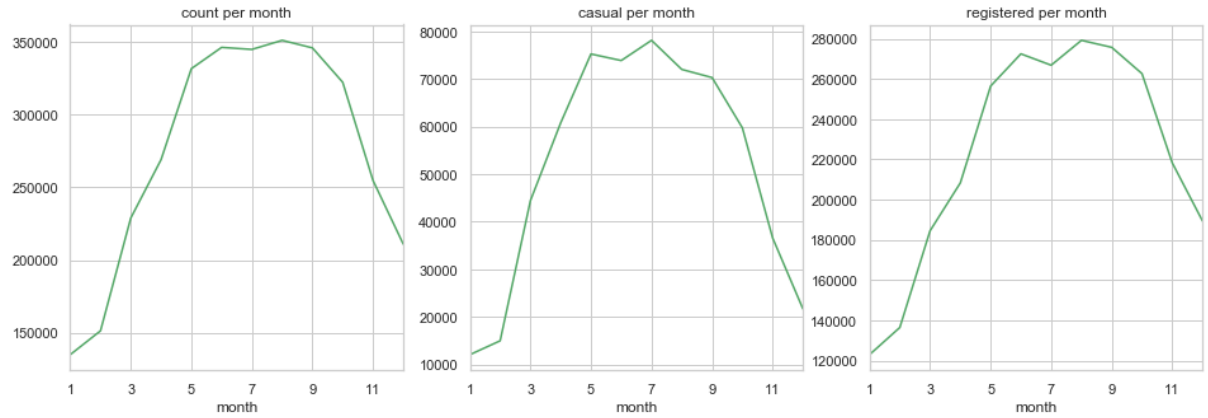


#casual user increases in year 1 (that would be year 2012)

#registered user also increases in year1

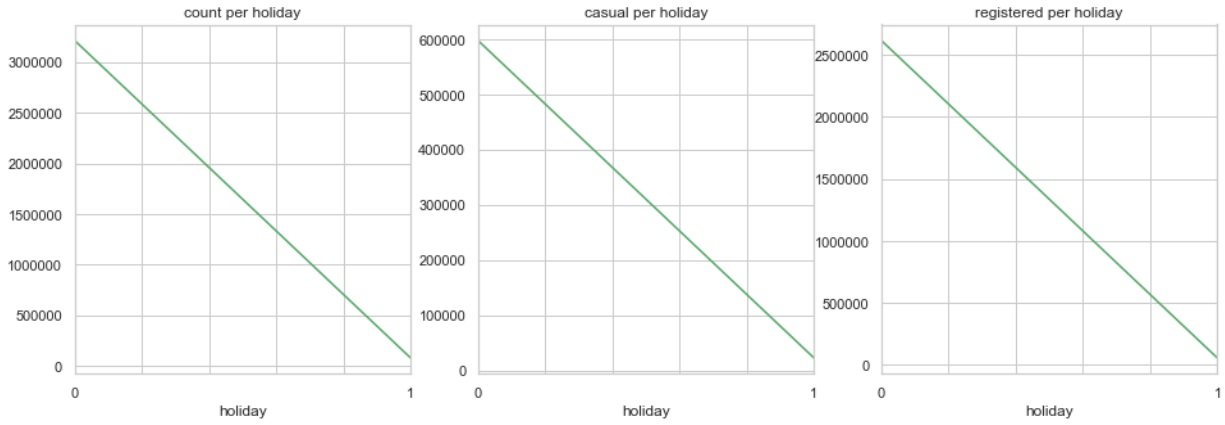
#count is sum of casual and registered so it also shows same behavior , means demand has increases in year 2012 as compared to year 0 (2011)

>> Month

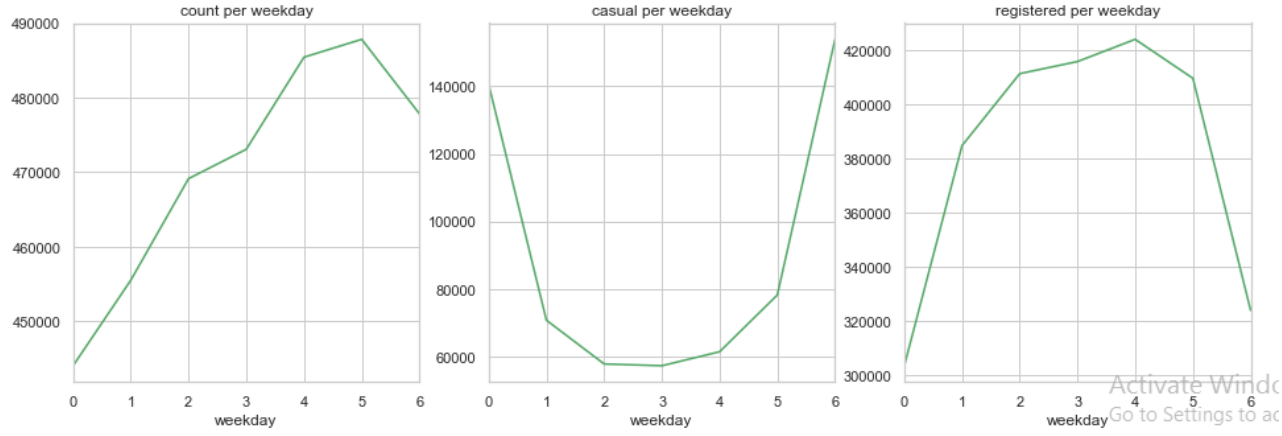


#count increases in month 5 and after that month 10 it decreases (may be because of at that time winter comes)
#casual also increases and then drops after month 10
registered also increases from month 2 and then decreases after month

>> Holiday
#count decreases on holidays we can say that max bike is rented for commute and on holiday people stay at home
casual users also decrease on holiday
#registered users also decrease on holiday



>> Weekday



0 – Sunday (as the number of casual users increases on 0 and there after on 1 (Monday) it decreases)

1- Monday (as registered users use bikes on Monday)

2- Tuesday

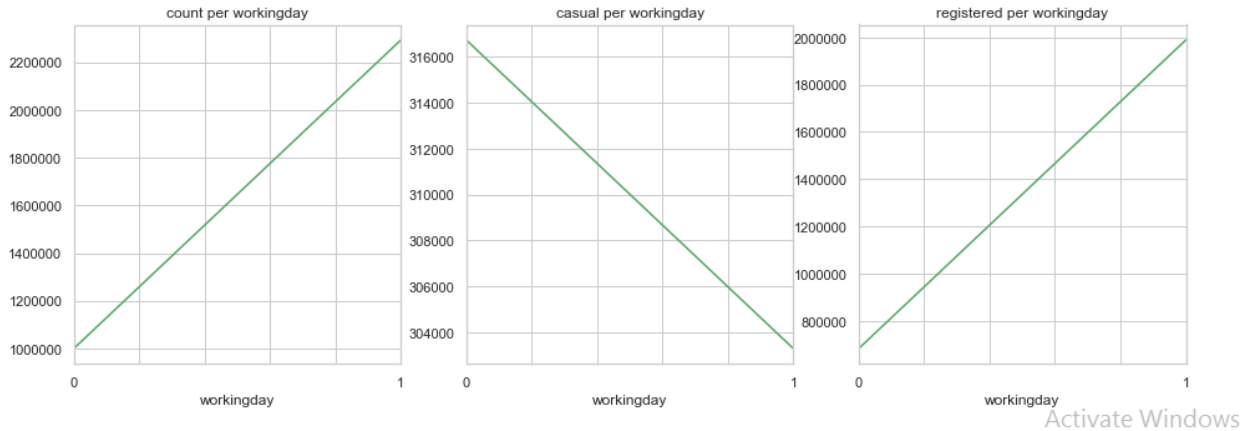
3- Wednesday

4- Thursday

5- Friday

6 – Saturday (as the casual starts to rent bikes on Saturday)

>>Working day

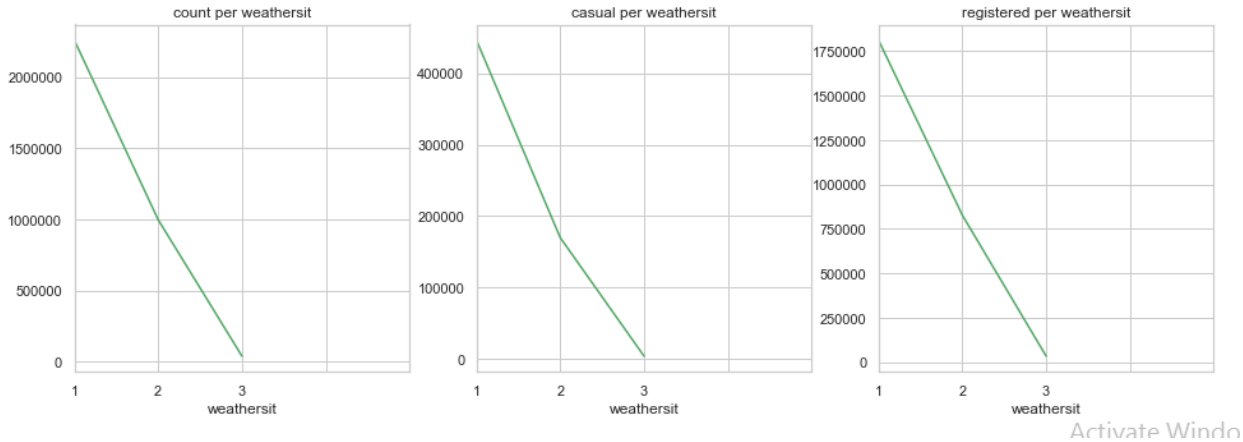


casual users don't rent bike on working days that means they don't use for commute, they use it occasionally

registered user rent it to go to office and therefore the on working day the number bike rented by registered users increases

overall count increases on working day as it is sum of registered and casual and we can clearly see that registered users has more impact on total count

>> weather situation

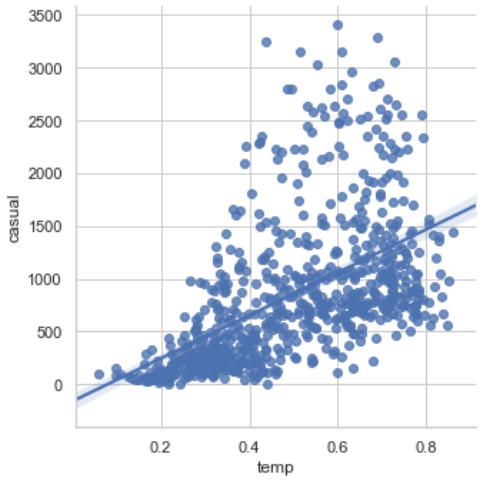


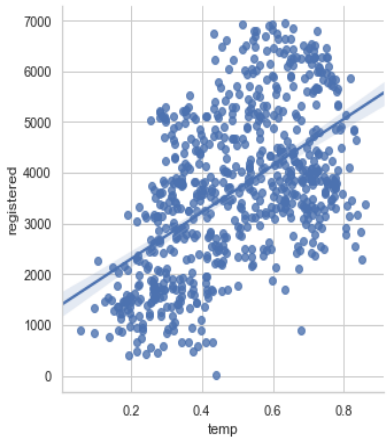
#count including registered and casual decreases in weather 3 and almost drops to zero in weather 4 because

#weather4 is heavy rain + thunderstorm

>>NUMERICAL COLUMNS WITH THE TARGET VARIABLE

>>temp

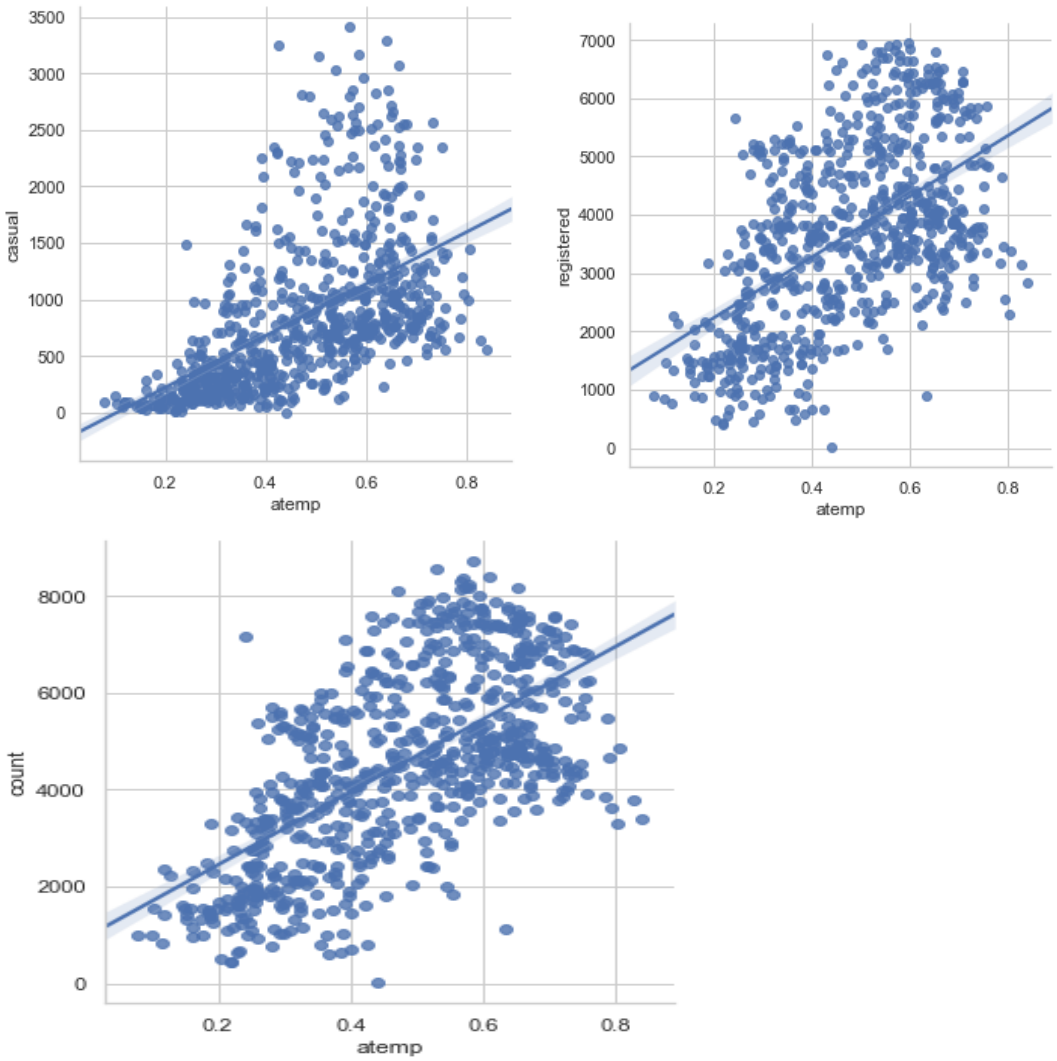




temp has good correlation with casual, registered and as the count is sum of casual and registered, the correlation seems good with count also

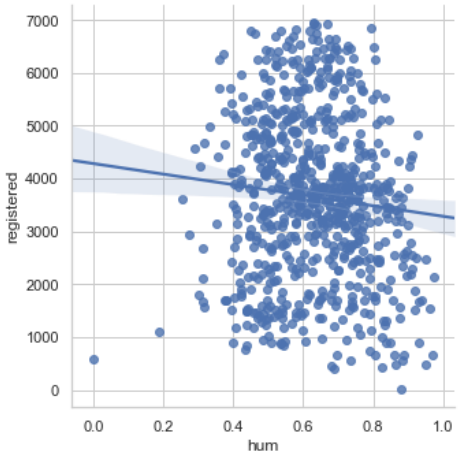
>>atemp

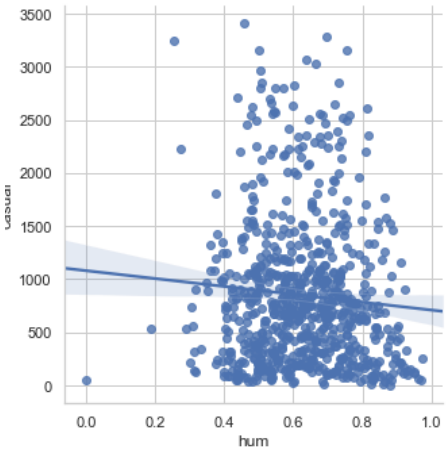
#atemp has good correlation with casual, registered and as the count is sum of casual and registered, the correlation seems good with count also

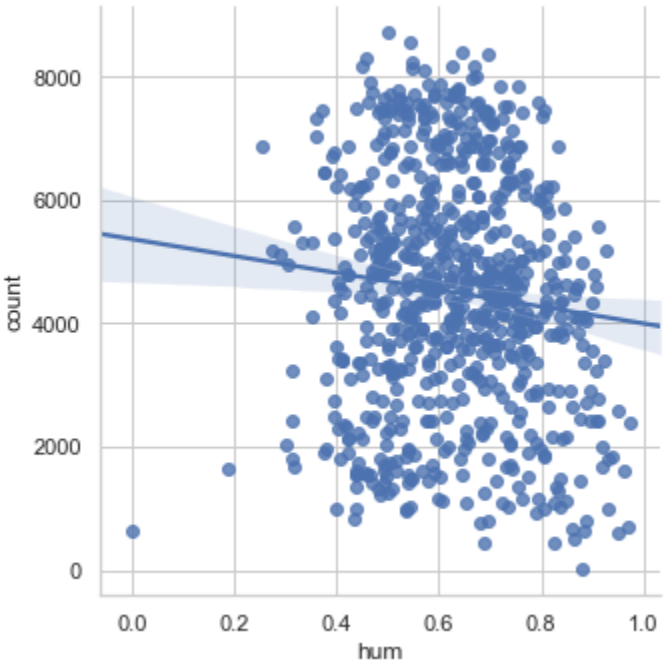


>>HUM

#it seems that hum has very less correlation with count, casual and registered

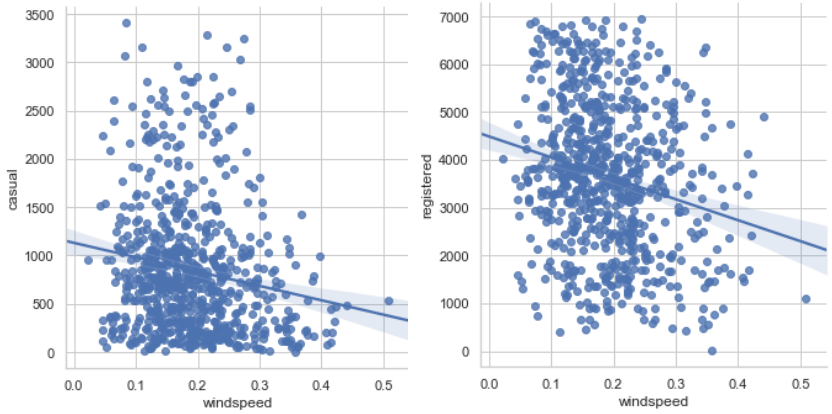


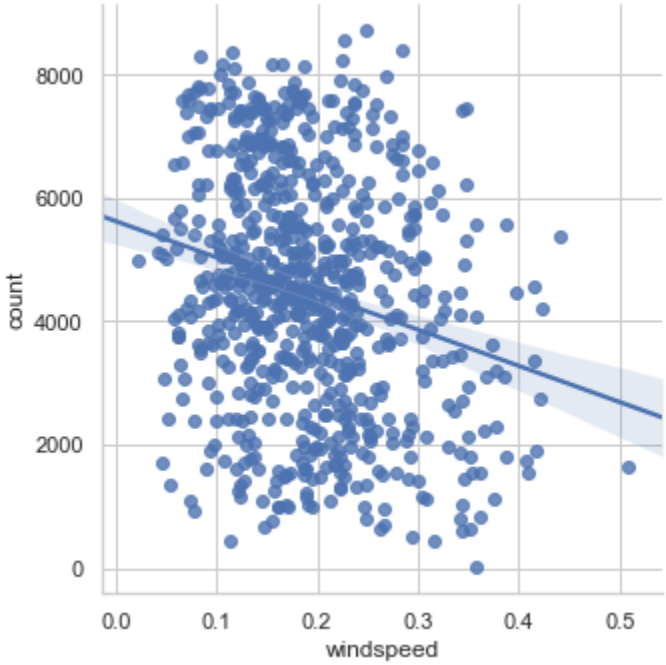




>>windspeed

#windspeed also has very less correlation with count, casual, registered





2.2>Missing value:

We will check if there is any missing value.

Here we got to know that we don't have any missing value in our data frame.

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

```
dateday       0
season        0
year          0
month         0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

2.3>Outlier Analysis:

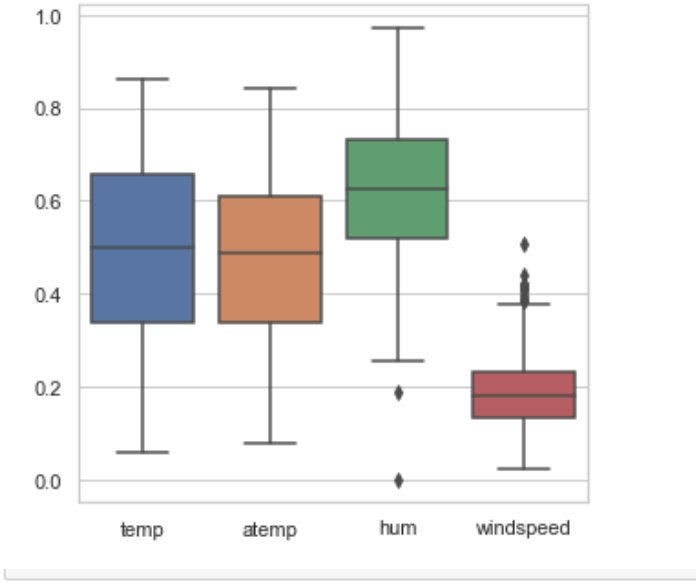
An outlier may also be explained as a piece of data or observation that deviates drastically from the given norm or average of the data set. An outlier may be caused simply by chance, but it may also indicate measurement error or that the given data set has a heavy-tailed distribution.

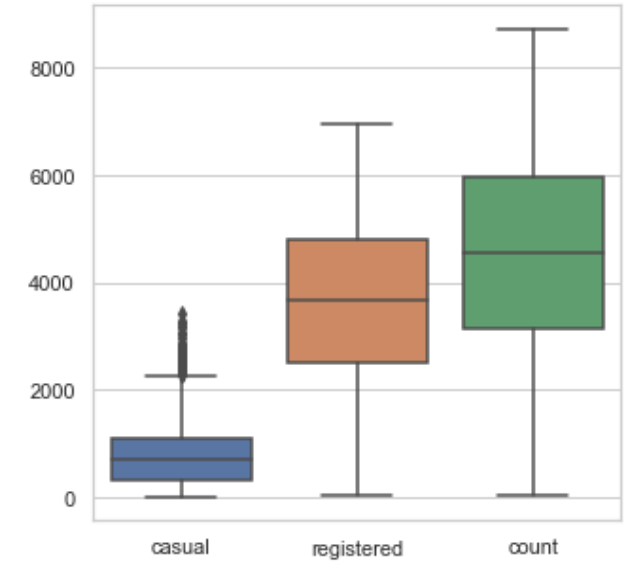
Here we have to methods to treat outliers which depends on the graph of the column, If the data we have is of normal type then we have to use the Z score method else for skwed type we have to use Numerical outlier.
We have made num_colnames (for names of column which are numeric)

num_colnames = ['temp','atemp','hum','windspeed','casual','registered']
cat_colnames = ['season', 'year','month','holiday', 'weekday','workingday','weathersit']

2.4>Box Plot of numerical variable to see outliers

>> plotting the box plot of numerical columns





We need to remove these outliers, after removal we can proceed further. We will use quartile method to replace.

Here we will replace the values that is above than the upper boundary and less the minimum boundary, we will replace them with Nan.

Upper boundary: q75 + 1.5*IQR
Lower boundary: q25 – 1.5*IQR

Then the Nan values would be considered as missing value and we will treat missing values with the appropriate method. It can be by mean, median, interpolate nearest, interpolate linear, interpolate cubic.

As we have seen that the maximum number of outliers are in casual. Let's make a particular value in casual column as Nan, and lets try which method will work best.

We are making copies of the df to check which method works.

```
df_m1 = df.copy()
df_m2 = df.copy()
df_m3 = df.copy()
df_m4 = df.copy()
df_m5 = df.copy()
```

We are making 1st row and 12th column whose value is 131 as Nan.

```
#the value was 131
df_m1.iloc[1,12] = np.nan
df_m2.iloc[1,12] = np.nan
df_m3.iloc[1,12] = np.nan
df_m4.iloc[1,12] = np.nan
df_m5.iloc[1,12] = np.nan
```

Now we will try different methods:

1>> mean

```
#using mean method
df_m1['casual'] = df_m1['casual'].fillna(df_m1['casual'].mean())
df_m1.iloc[1,12]
```

```
732.8862973760932
```

We are getting the replacement as 732 which is very far away from 131.

2>>median

```
#using median method
df_m2['casual'] = df_m2['casual'].fillna(df_m2['casual'].median())
df_m2.iloc[1,12]
```

```
675.0
```

We are getting the replacement as 675 which is very far away from 131.

3>> interpolate (Nearest)

```
#using interpolate method to replace the outliers(NEAREST)
df_m3['casual'] = df_m3['casual'].interpolate(method = 'nearest', limit_direction = 'both')
df_m3.iloc[1,12]
```

```
331.0
```

We are getting the replacement as 331 which is very far away from 131.

4>> Interpolate (Linear)

```
#using interpolate method to replace the outliers(LINEAR)
df_m4['casual'] = df_m4['casual'].interpolate(method = 'linear', limit_direction = 'both')
df_m4.iloc[1,12]
```

```
225.5
```

We are getting the replacement as 225 which is very far away from 131.

5>> Interpolate(Cubic)

```
#using interpolate method to replace the outliers(Cubic)
df_m5['casual'] = df_m5['casual'].interpolate(method = 'cubic', limit_direction = 'both')
df_m5.iloc[1,12]
```

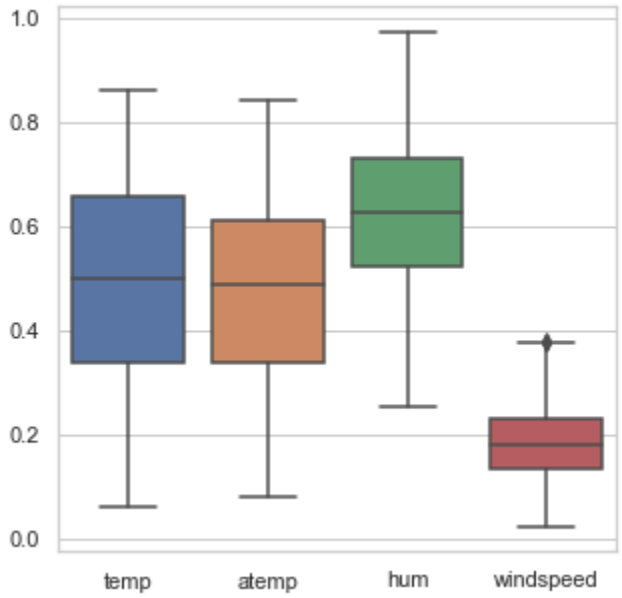
```
169.9172898670703
```

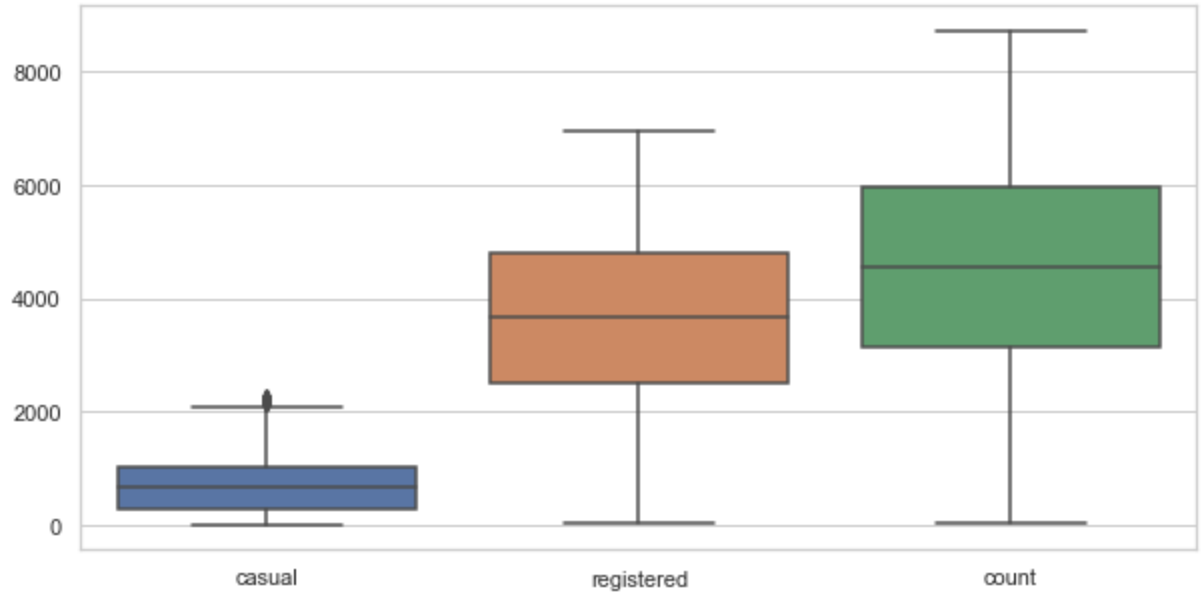
We are getting the replacement as 169 that is near 131. We will use this to replace our missing value created by replacing the outlier with Nan.

We are using the interpolate (cubic) on columns hum , windspeed , casual

```
#using interpolate method to replace the outliers
df['hum'] = df['hum'].interpolate(method ='cubic', limit_direction = 'both')
df['windspeed'] = df['windspeed'].interpolate(method ='cubic', limit_direction = 'both')
df['casual'] = df['casual'].interpolate(method ='cubic', limit_direction = 'both')
```

We are checking the outliers again:





We are getting outliers again in casual and windspeed.

So basically, what is happening is, after replacement of those Nan value with interpolate. Let's suppose the nan is getting replaced with x. So now for the column, there would be a new minimum value, a maximum value. So, when box plot plots the plot, this x value is acting as outlier.

Case 1: We tried to use the interpolate method in hope to treat those outliers.
Result: we tried that but, when we are again plotting the box plot, we are again getting the outlier.

Case 2:
We can try al linear model to predict the values according to the column
Result: In this case also, we are getting the outlier in casual column.

One thought might be to again treat the outlier which are getting created, but we can think in the direction that we have removed the outlier once, and may be the outlier we are getting aren't actually the outliers(because boxplot is making the calculation and it is saying that this value is outlier , but may be it isn't the case).

So we will go with these outlier only.
making a copy of the df , we are naming it bike_data

2.5>Correlation among variables

2.5.1>Feature selection (Correlation in continuous variable)

Feature selection is used to determine which features are of really of use because if there are two variable which are highly correlated then means they both give same information, so we need to get rid of one.

	temp	atemp	hum	windspeed	casual	registered	count
temp	1	0.991702	0.123431	0.150256	0.594834	0.540012	0.627494
atemp	0.991702	1	0.137013	0.176047	0.594466	0.544192	0.631066
hum	0.123431	0.137013	1	0.214651	0.0816912	0.11201	0.121642
windspeed	0.150256	0.176047	0.214651	1	0.193678	0.211365	0.226815
casual	0.594834	0.594466	0.0816912	0.193678	1	0.415295	0.654747
registered	0.540012	0.544192	0.11201	0.211365	0.415295	1	0.945517
count	0.627494	0.631066	0.121642	0.226815	0.654747	0.945517	1

temp and atemp is having max correlation (0.991702). So, we need to drop one from these two. we will drop the atemp (as this is the temp felt which might change for person to person and it in future the temp felt might not be available). Its, better to have as input as temp because it will be available, and it will not change person to person.
#we are dropping the "hum" as it has very LESS correlation with count.
#casual: they are the bikes randomly rented
registered: They are the user which have registered
We are dropping both the columns casual and registered because we have the count as the sum of casual and registered. and in future when we have to predict on the basis environment factors.
Nobody would have the casual and registered count.
And there is one more issue about the data leakage: the two columns "casual" and "registered", if we take both, then we will get 99 percent accuracy and that is data leakage.

here I am keeping the count variable as I need to predict the total number of the count for an environment factor.

2.5.2> Chi-square test

We also need to see the relation between the categorical columns. For that we need to use chi-square test. and if there is any other column which zero other than diagonal elements, it means our null hypothesis is false.

```
from scipy.stats import chi2_contingency
#for catagorical variables we will perform Chisquare test of independence
for i in cat_colnames:
    print(i)
    chi2, p, dof, ex = chi2_contingency(pd.crosstab(bike_data['count'], bike_data[i]))
    print(p)
```

```
season
0.5440596338517547
year
0.367724209978971
month
0.49177634472851
holiday
0.6781463865579803
weekday
0.4102128310764939
workingday
0.45435914436398794
weathersit
0.6407439686474243
```

Here we are taking all independent variables
One by one and getting their p value with count.
Null hypothesis: both are independent

Alternate hypothesis: there is relation

here we are dropping the column having max value of p that is holiday. (as this will not be independent variable with count and we don't want that out variables are independent with target variable)

Chapter 3

Modeling

After doing these pre-processing we need to train our model so that we can predict the outcome (absenteeism hours) in future. Here we need to split our data and then train our model.

3.1> Splitting data: we need to divide the data into train (80 percent) and test (20 percent).

Model selection: we need to decide which model we need to use for our data. The target variable in our model is a continuous variable (Absenteeism time in hour). So, the models that we choose are Decision Tree and Random Forest, Linear Regression, OLS(python). The error metric chosen for the given problem statement is mean_absolute_error and Rmse.

3.2> Decision Tree:

```
#Decision tree for regression
# Train the model using the training sets
fit_DT = DecisionTreeRegressor(max_depth=2).fit(x_train, y_train)
# make the predictions by the model
predictions_DT = fit_DT.predict(x_test)
# data frame for actual and predicted values
df_dt = pd.DataFrame({'actual': y_test, 'pred': predictions_DT})
print(df_dt.head())
#Calculate MAE
def MAE(y_true, y_pred):
    mae = np.mean(np.abs(( y_true - y_pred) / y_true))
    return mae
mae_dt=MAE(y_test, predictions_DT)
#RMSE
from sklearn.metrics import r2_score
from math import sqrt

def RSQUARE(y_true, y_pred):
    rsquare = r2_score(y_true, y_pred)
    return rsquare
rsquare_dt =RSQUARE(y_test, predictions_DT)

# Calculate and display accuracy
accuracy_MAPE_dt = 100 - np.mean(mae_dt)*100

# errors and accuracy
print("MEAN ABSOLUTE ERROR:"+str(mae_dt))
print('Accuracy:', round(accuracy_MAPE_dt, 2), '%.')
print("RSQUARE:"+str(rsquare_dt))
```

R-squared is a statistical measure of how close the data are to the fitted regression line it is the percentage of the response variable variation that is explained by a linear model.

MAE is the sum of absolute differences between our target and predicted variables. So it measures the average magnitude of errors in a set of predictions, without considering their directions.

Decision tree builds regression is in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision

tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

	Mae(Accuracy)	R sq
Python	0.27375526781(72.62 %.)	0.58915674274835
R	0.18194 (81.8051)	0.848048

3.3>Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

```
#Random forest for regression
#Import libraries for RF
from sklearn.ensemble import RandomForestRegressor
# Train the model using the training sets
RFmodel = RandomForestRegressor(n_estimators = 200).fit(x_train, y_train)
# make the predictions by the model
RF_Predictions = RFmodel.predict(x_test).round(0)
# data frame for actual and predicted values
df_RF = pd.DataFrame({'actual': y_test, 'pred': RF_Predictions})
print(df_dt.head())
# Calculate and display accuracy
mae=MAE(y_test,RF_Predictions)
accuracy = 100 - np.mean(mae*100)

rsquare_dt =RSQUARE(y_test, RF_Predictions)
print("MEAN ABSOLUTE ERROR:"+str(mae))
print('Accuracy:', round(accuracy, 2), '%.')
print('RSQUARE:', round(rsquare_dt, 2))
```

	Mae(Accuracy)	R sq
Python	0.170605801775227 (82.94)	0.83
R	0.128146 (87.1854)	0.863417

3.4>Linear: The technique uses statistical calculations to plot a trend line in a set of data points. The trend line could be anything from the number of people diagnosed with skin cancer to the financial performance of a company. Linear regression shows a relationship between an independent variable and a dependent variable being studied.

```
# linear
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression().fit(x_train, y_train)

#predict
lr_prediction = lr_model.predict(x_test)

df_lr = pd.DataFrame({'actual':y_test, 'prediction':lr_prediction})

# Calculate and display accuracy
mae_lr=MAE(y_test,lr_prediction)
accuracy_lr = 100 - np.mean(mae_lr*100)
print("MEAN ABSOLUTE ERROR:"+str(mae_lr))
print('Accuracy:', round(accuracy_lr, 2), '%.')


rsquare_lr =RSQUARE(y_test, lr_prediction)
print("RSQUARE:"+str(rsquare_lr))
```

.

	Mae(Accuracy)	R sq
Python	0.1920944479(80.79)	0.75117273049489
R	0.1738101 (82.6189)	0.8295169

3.5>OLS method : Ordinary least squares (OLS) regression is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable; the method estimates the relationship by minimizing the sum of the squares in the difference between the observed and predicted values of the dependent variable configured as a straight line

```
#OLS
#Import libraries for LR
import statsmodels.api as sm
# Train the model using the training sets
model = sm.OLS(y_train,x_train.astype(float)).fit()
#Summary of model
model.summary()
# make the predictions by the model
predictions_OLS = model.predict(x_test.astype(float)).round(0)
# data frame for actual and predicted values
df_OLS = pd.DataFrame({'actual': y_test, 'pred': predictions_OLS})

# Calculate and display accuracy
mae_ols=MAE(y_test,predictions_OLS)
accuracy_ols = 100 - np.mean(mae_ols*100)
print("MEAN ABSOLUTE ERROR:"+str(mae_ols))
print('Accuracy:', round(accuracy_lr, 2), '%.')


rsquare_ols =RSQUARE(y_test, predictions_OLS)
print("r2score:"+str(rsquare_ols))


model.summary()
```

Dep. Variable:	count	**R-squared:**	0.968
Model:	OLS	**Adj. R-squared:**	0.968
Method:	Least Squares	**F-statistic:**	1641.
Date:	Wed, 26 Jun 2019	**Prob (F-statistic):**	9.50e-317
Time:	15:34:29	**Log-Likelihood:**	-3592.7
No. Observations:	438	**AIC:**	7201.
Df Residuals:	430	**BIC:**	7234.
Df Model:	8		
Covariance Type:	nonrobust		

	Mae(Accuracy)	R sq
Python	0.1948175499018 (80.94)	0.738862547496

	coef	std err	t	P>|t|	[0.025	0.975]
season	554.4669	68.451	8.100	0.000	419.927	689.007
year	2111.0588	84.298	25.043	0.000	1945.371	2276.747
month	-32.2170	20.750	-1.553	0.121	-73.000	8.566
weekday	114.5806	21.078	5.436	0.000	73.153	156.009
workingday	256.3359	90.125	2.844	0.005	79.195	433.477
weathersit	-612.8623	70.188	-8.732	0.000	-750.816	-474.908
temp	5762.4073	229.491	25.110	0.000	5311.344	6213.471
windspeed	-1363.6652	493.967	-2.761	0.006	-2334.555	-392.775

Chapter 4

Conclusion

4.1> Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. We need to decide our model on basis of *Predictive performance* as the criteria to compare and evaluate models

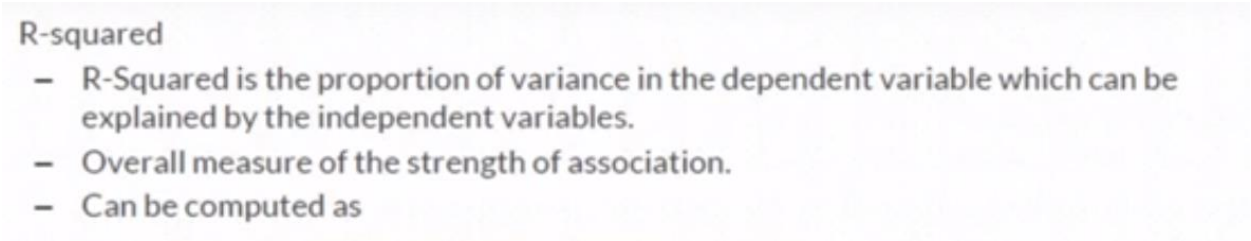
Predictive performance can be measured by comparing Predictions of the models with real values of the target variables and calculating some average error measure.

4.1.1> Mean Absolute Error (MAE) and R square

MAE is one of the error measures used to calculate the predictive performance of the

model. We will apply this measure to our models that we have generated in the previous section.

```
def MAE(y_true, y_pred):

   mae = np.mean(np.abs(( y_true - y_pred) / y_true))

   return mae
```



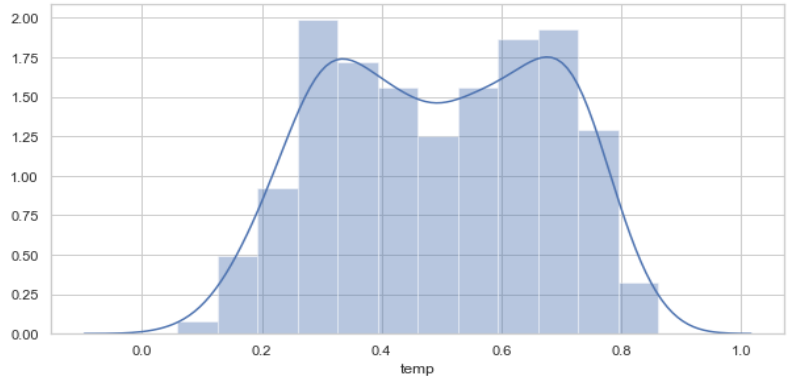

```
def R2 =function( y_true, y_pred ) {

  rss =sum((y_pred - y_true) ^ 2)  ## residual sum of squares

  tss =sum((y_true - mean(y_true)) ^ 2)  ## total sum of squares

  rsq =1 - rss/tss }
```

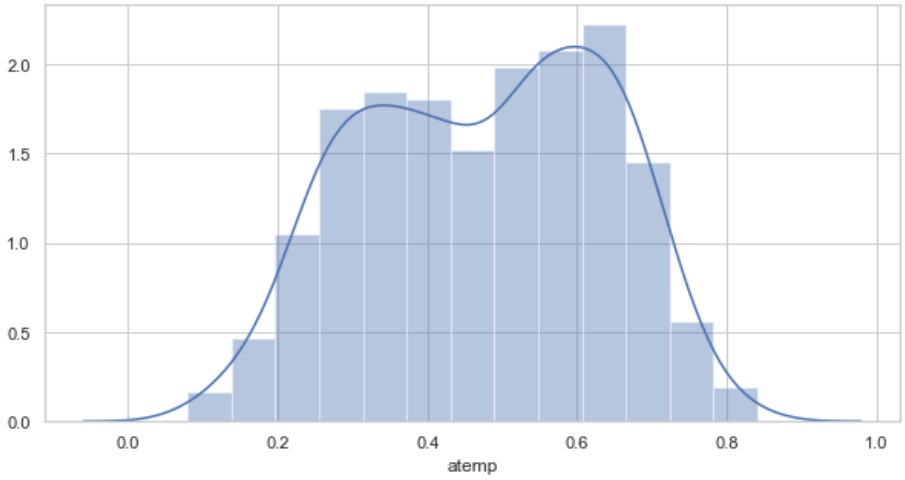
4.1.2> Model Selection

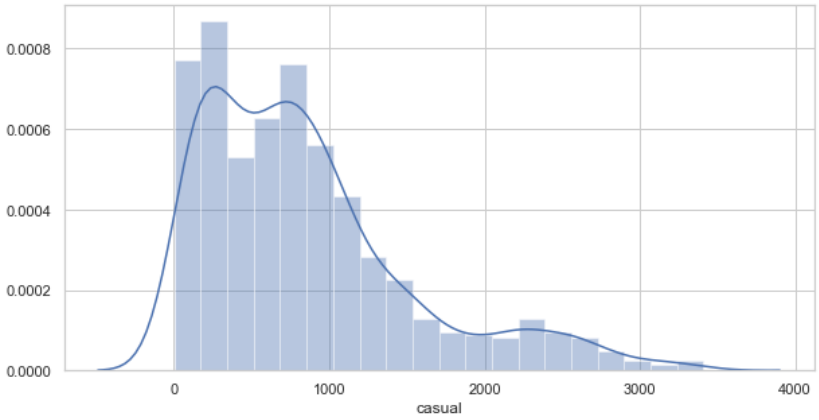
We can see that all the models perform well on the test data, However Random forest is working well and giving more accuracy comparative to them. Therefore, we can select either of the two models without any loss of information.

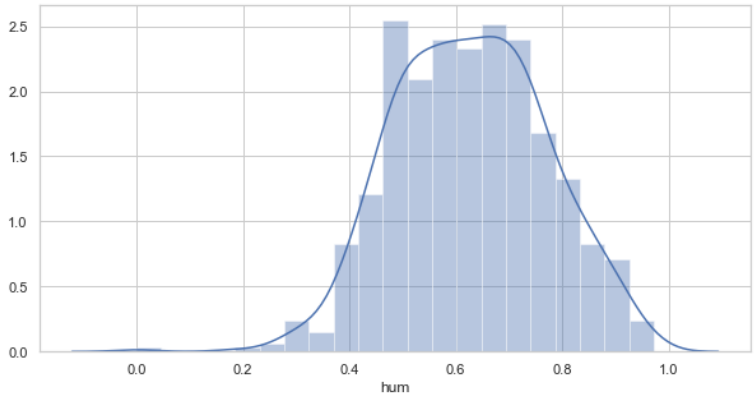
Chapter 5

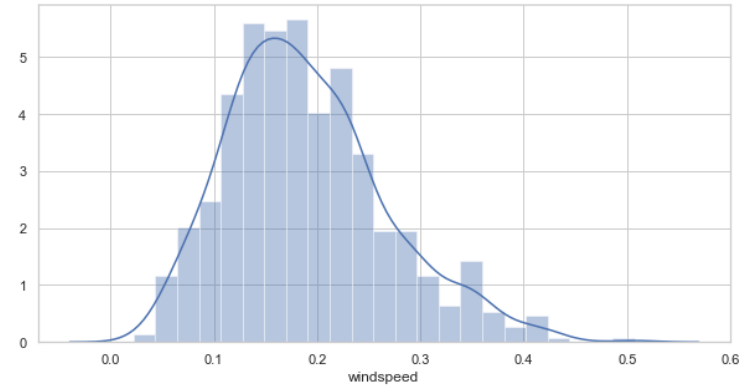
5.1>>Appendix A - Extra Figures

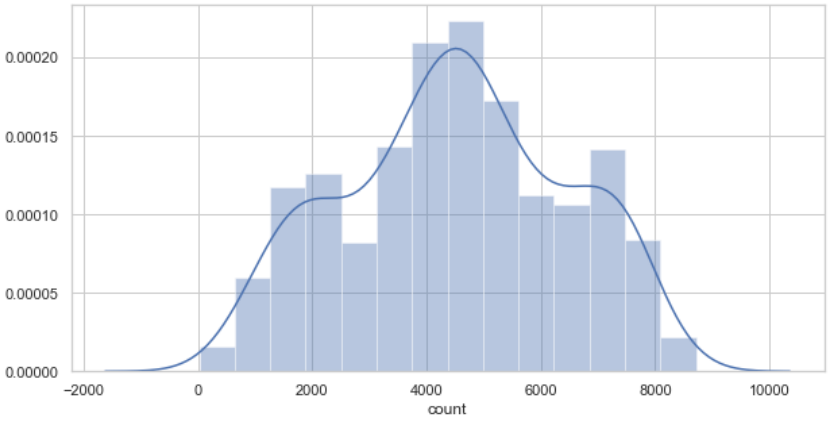


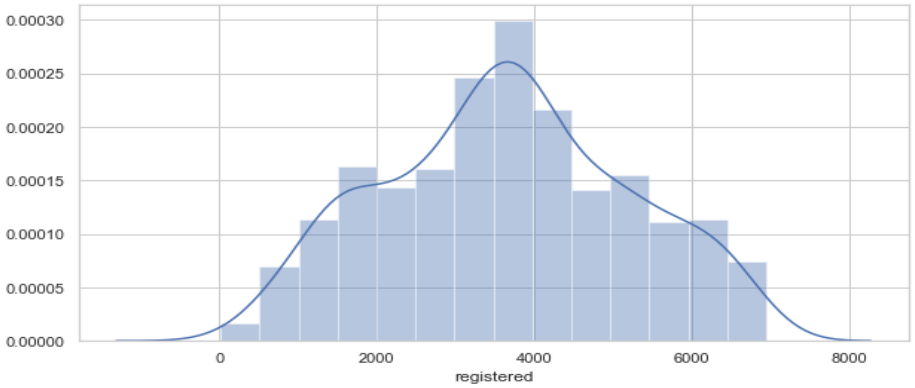


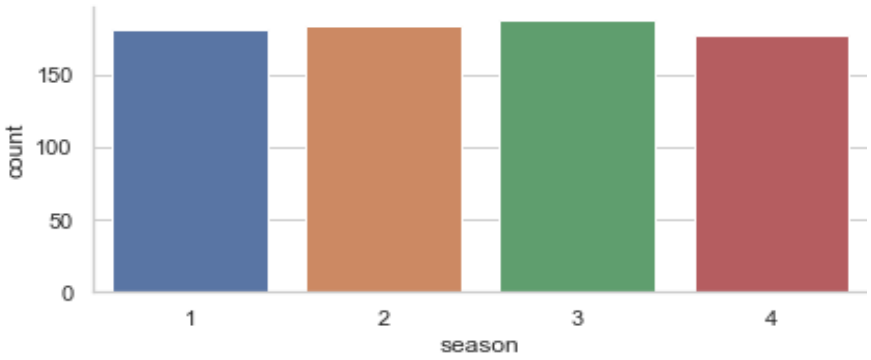


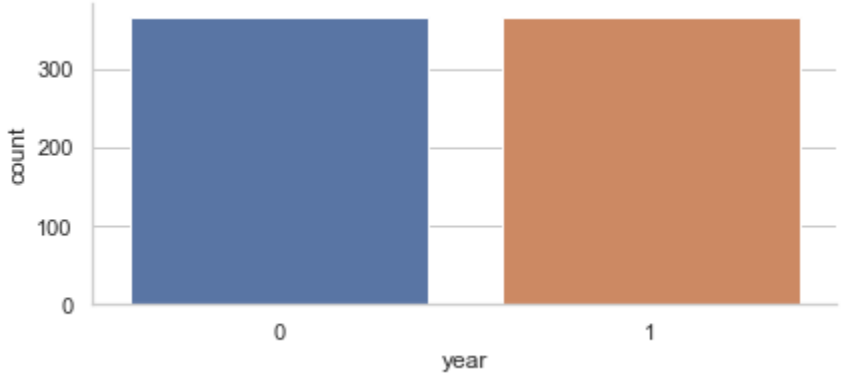

2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
hum

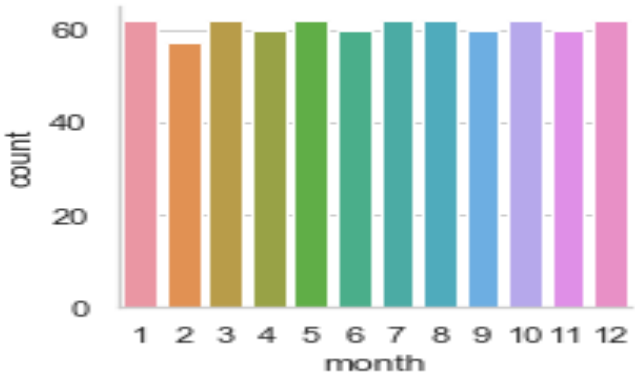

5
4
3
2
1
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
windspeed

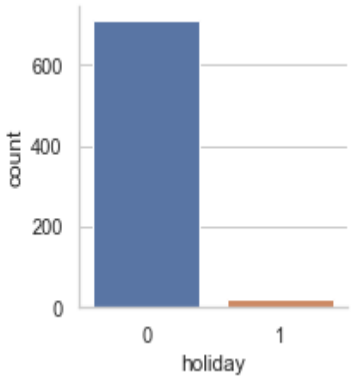

0.00020
0.00015
0.00010
0.00005
0.00000
−2000
0
2000
4000
6000
8000
10000
count

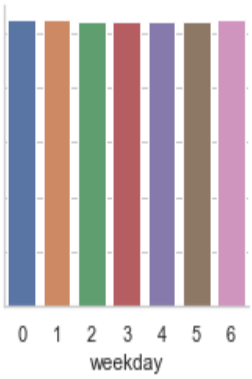

0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
2000
4000
6000
8000
registered

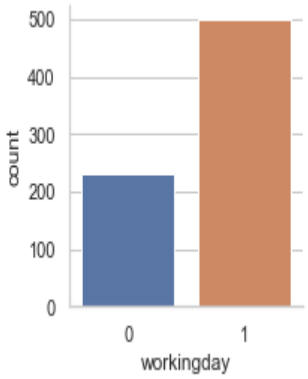

150
100
50
0
count
1
2
3
4
season

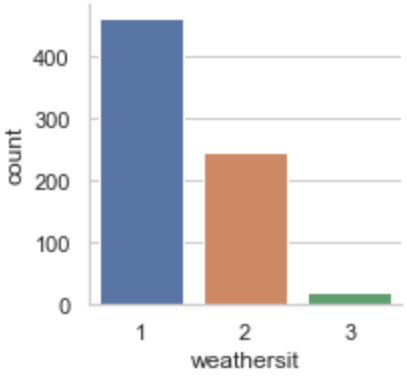

300
200
100
0
count
0
1
year

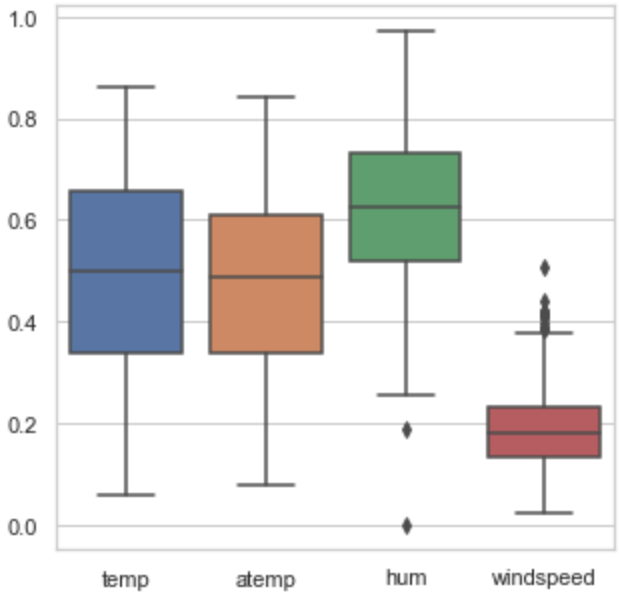

60
40
20
0
count
1 2 3 4 5 6 7 8 9 10 11 12
month

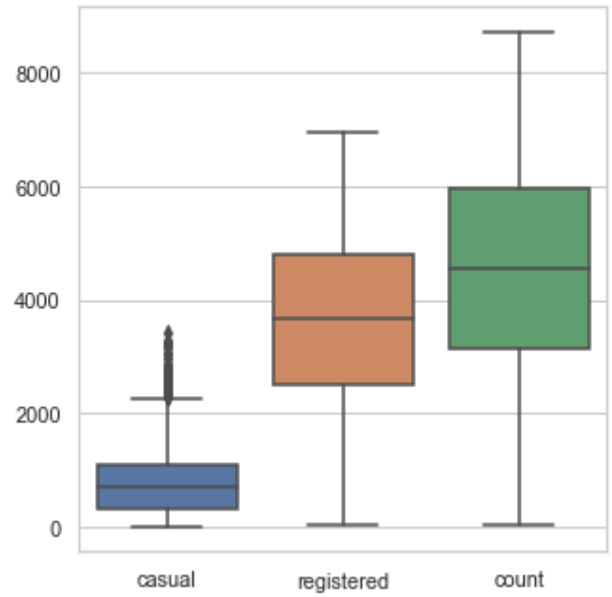

count
600
400
200
0
0
1
holiday

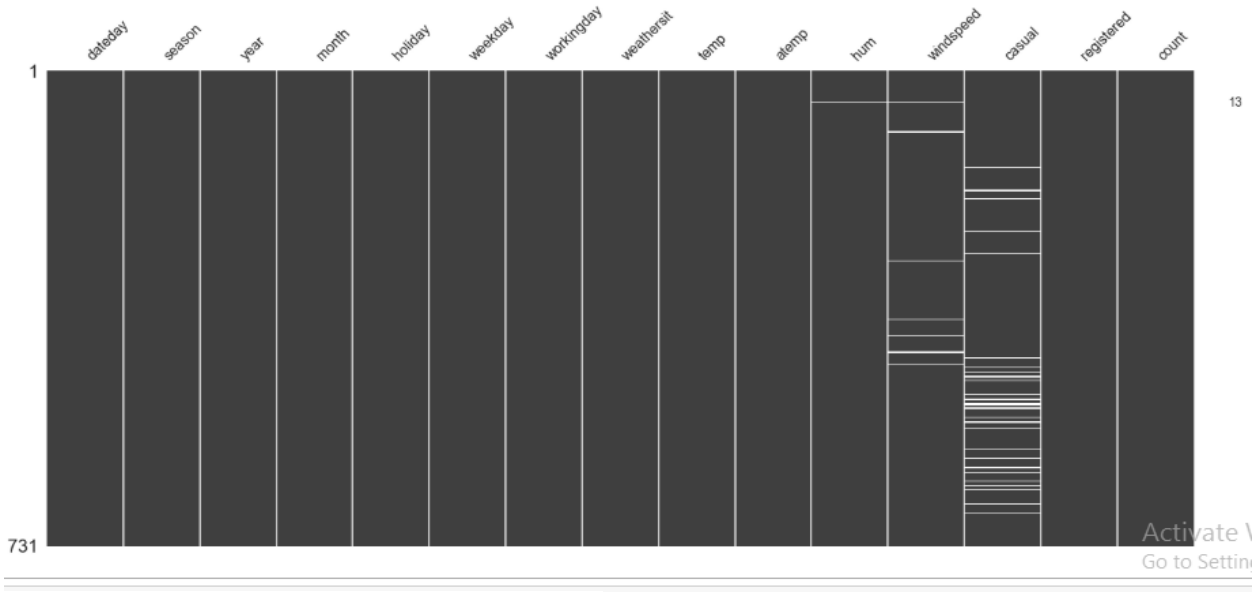

0 1 2 3 4 5 6
weekday

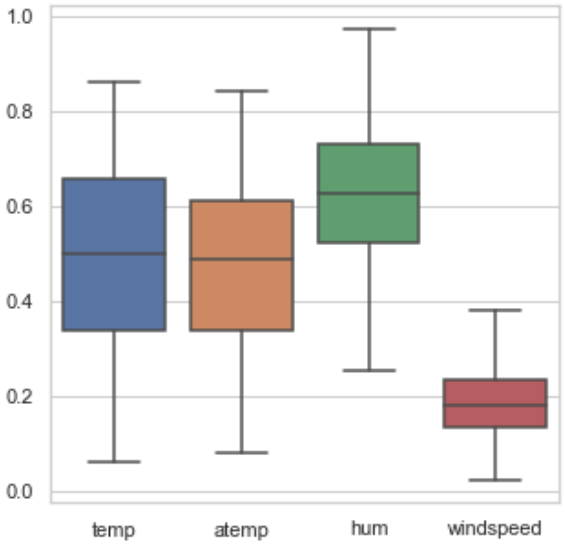

count
500
400
300
200
100
0
0
1
workingday

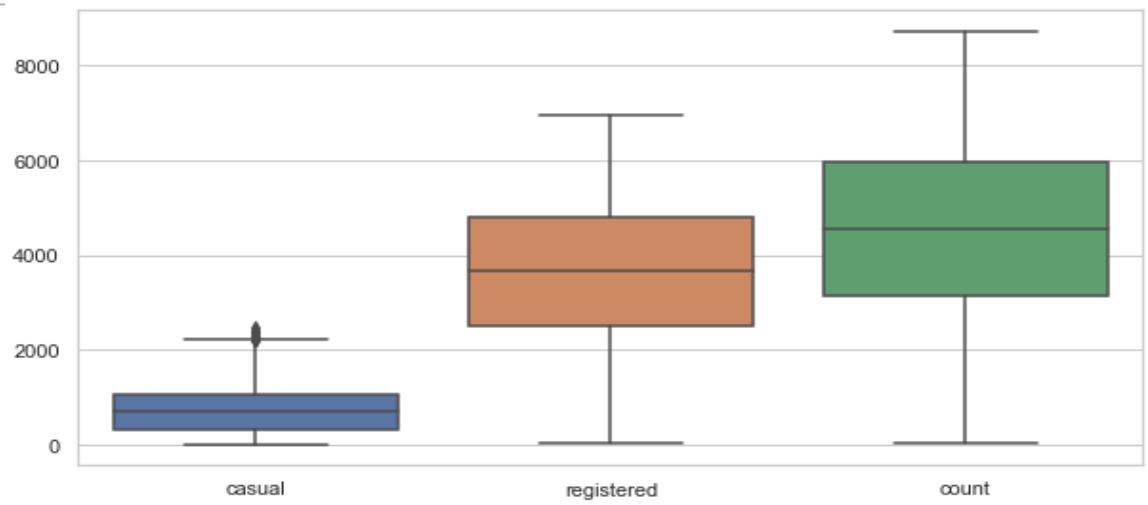

count
400
300
200
100
0
1
2
3
weathersit


1.0
0.8
0.6
0.4
0.2
0.0
temp
atemp
hum
windspeed


8000
6000
4000
2000
0
casual
registered
count


dateday
season
year
month
holiday
weekday
workingday
weathersit
temp
atemp
hum
windspeed
casual
registered
count
1
731
13

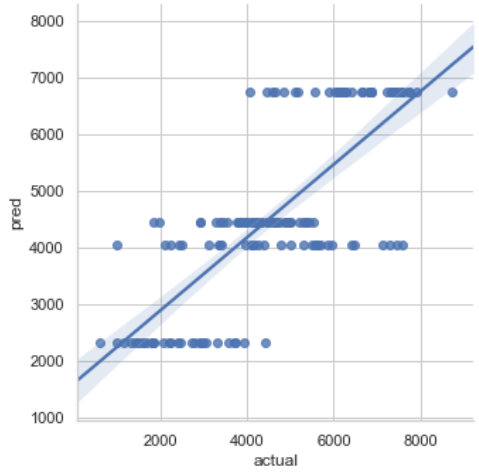

1.0
0.8
0.6
0.4
0.2
0.0
temp
atemp
hum
windspeed


8000
6000
4000
2000
0
casual
registered
count

	temp	atemp	hum	windspeed	casual	registered	count
temp	1	0.991702	0.123431	0.150256	0.594834	0.540012	0.627494
atemp	0.991702	1	0.137013	0.176047	0.594466	0.544192	0.631066
hum	0.123431	0.137013	1	0.214651	0.0816912	0.11201	0.121642
windspeed	0.150256	0.176047	0.214651	1	0.193678	0.211365	0.226815
casual	0.594834	0.594466	0.0816912	0.193678	1	0.415295	0.654747
registered	0.540012	0.544192	0.11201	0.211365	0.415295	1	0.945517
count	0.627494	0.631066	0.121642	0.226815	0.654747	0.945517	1

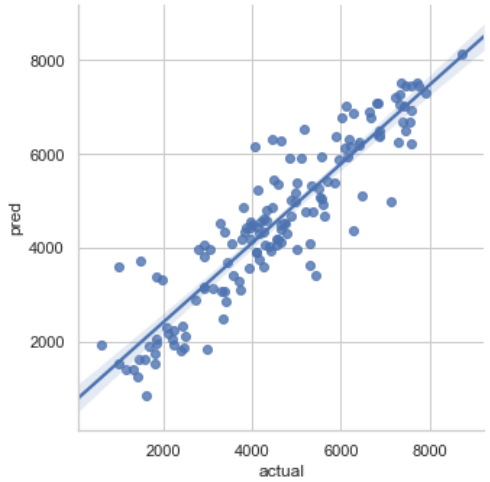
```
sns.lmplot(x='actual', y='pred', data = df_dt ,fit_reg = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x1bcf116acf8>
```



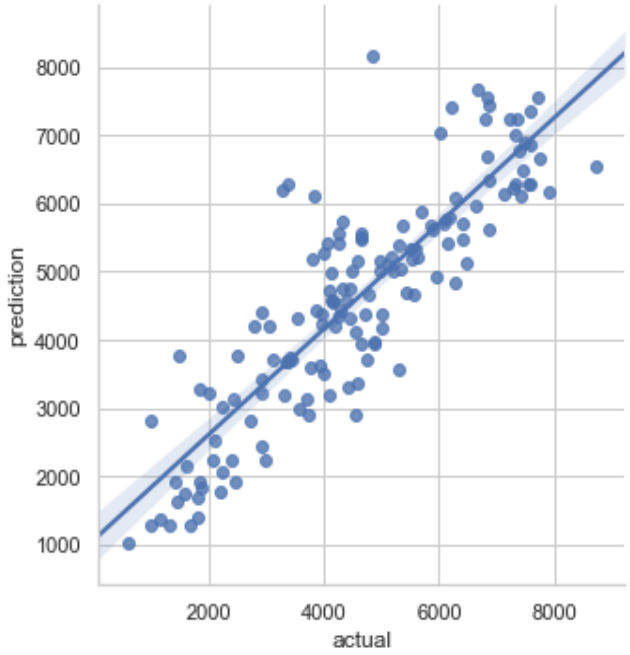

```
# df_RF.plot.scatter(x='actual', y='pred')
sns.lmplot(x='actual', y='pred', data = df_RF ,fit_reg = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x1bcf55f2400>
```



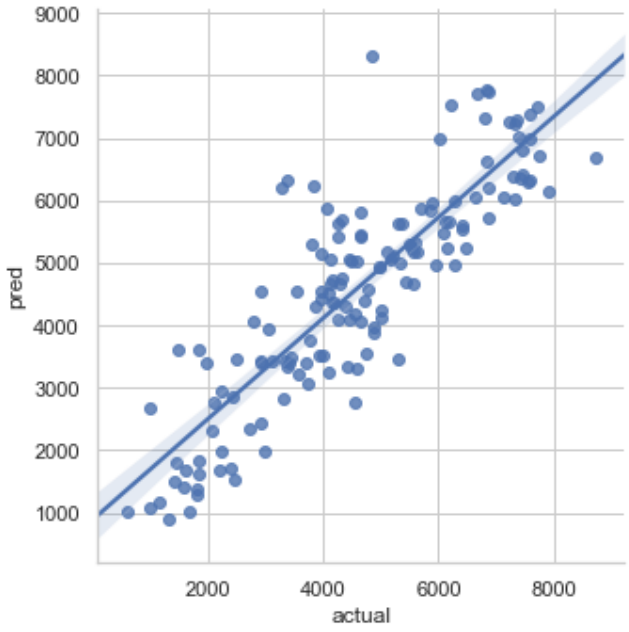
```
sns.lmplot(x='actual', y='prediction', data = df_lr ,fit_reg = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x1bcf55f2748>
```




```
sns.lmplot(x='actual', y='pred', data = df_OLS ,fit_reg = True)
```

```
<seaborn.axisgrid.FacetGrid at 0x1bcf721a9e8>
```



5.2>R Code

```
#Clean the environment
rm(list = ls())

#Seting the  working directory
setwd("D:/Rfiles")

#Loading the librarires which would be needed
libraries = c("rpart.plot","plyr","dplyr",
"ggplot2","rpart",'propagate',"DMwR","randomForest","usdm","DataCombine")
lapply(X = libraries,require, character.only = TRUE)
rm(libraries)

#Read the csv file and making insant as index column
df = read.csv(file = "bikedata.csv", header = T,row.names="instant")


######EXPLORE THE DATA

#number of rows and columns
dim(df)

#Observe top 5 rows
head(df)

#Structure of variables
str(df)

#renaming the column
colnames(df)[colnames(df)=="dteday"] <- "dateday"
colnames(df)[colnames(df)=="mnth"] <- "month"
colnames(df)[colnames(df)=="yr"] <- "year"
```

```
colnames(df)[colnames(df)=="cnt"] <- "count"


#Transform data types

#making factor
df$season = as.factor(df$season)
df$year = as.factor(df$year)
df$month = as.factor(df$month)
df$holiday = as.factor(df$holiday)
df$weekday = as.factor(df$weekday)
df$workingday = as.factor(df$workingday)
df$weathersit = as.factor(df$weathersit)
#for numeric

df$temp = as.numeric(df$temp)
df$atemp = as.numeric(df$atemp)
df$hum = as.numeric(df$hum)
df$windspeed = as.numeric(df$windspeed)
df$casual = as.numeric(df$casual)
df$registered = as.numeric(df$registered)
df$count = as.numeric(df$count)


#checking the skewness and kurtosis of the numerical columns
skewness(df$temp)
skewness(df$atemp)
skewness(df$hum)
skewness(df$windspeed)
skewness(df$casual)
skewness(df$registered)
skewness(df$count)
```

```
kurtosis(df$temp)
kurtosis(df$atemp)
kurtosis(df$hum)
kurtosis(df$windspeed)
kurtosis(df$casual)
kurtosis(df$registered)
kurtosis(df$count)

# the skew limit is -1 to 1 , so from numercial variables temp , atemp , hum ,
windspeed, casual , registered , cnt .
#so thia casual (1.2 ) is highy skewed and others are modererly skewed .
#so it means that there might be chnaces of having outliers in casual


##########################################  analysis
# numerical data
#numeric_index = sapply(df, is.numeric)
#numeric_data = df[,numeric_index]
#Check the distribution of numerical data using histogram
hist1 = ggplot(data = df, aes(x =temp)) + ggtitle("Distribution of : temp") +
geom_histogram(bins = 25)
hist2 = ggplot(data = df, aes(x =atemp)) + ggtitle("Distribution of: atemp") +
geom_histogram(bins = 25)
hist3 = ggplot(data = df, aes(x =hum)) + ggtitle("Distribution of: hum") +
geom_histogram(bins = 25)
hist4 = ggplot(data = df, aes(x =windspeed)) + ggtitle("Distribution of : windspeed")
+ geom_histogram(bins = 25)
hist5 = ggplot(data = df, aes(x =casual)) + ggtitle("Distribution of : casual") +
geom_histogram(bins = 25)
hist6 = ggplot(data = df, aes(x =registered)) + ggtitle("Distribution of :
registered") + geom_histogram(bins = 25)

#making a grid
```

```
gridExtra::grid.arrange(hist1,hist2,ncol=1)
gridExtra::grid.arrange(hist3,hist4,ncol=1)
gridExtra::grid.arrange(hist5,hist6,ncol=1)

#Distribution of factor data using bar plot
bar1 = ggplot(data = df, aes(x = season)) + geom_bar() + ggtitle("season") +
theme_dark()
bar2 = ggplot(data = df, aes(x = year)) + geom_bar() +  ggtitle("year") +
theme_dark()
bar3 = ggplot(data = df, aes(x = month)) + geom_bar() + ggtitle("Month") +
theme_dark()
bar4 = ggplot(data = df, aes(x = holiday)) + geom_bar() + ggtitle("holiday") +
theme_dark()
bar5 = ggplot(data = df, aes(x = weekday)) + geom_bar() + ggtitle("weekday") +
theme_dark()
bar6 = ggplot(data = df, aes(x = workingday)) + geom_bar() + ggtitle("workingday")
+ theme_dark()
bar7 = ggplot(data = df, aes(x = weathersit)) + geom_bar() + ggtitle("weathersit") +
theme_dark()

#making a grid
gridExtra::grid.arrange(bar1,bar2,ncol=1)
gridExtra::grid.arrange(bar3,bar4,ncol=1)
gridExtra::grid.arrange(bar5,bar6,bar7,ncol=2)



#######MISSING VALUE ANALYSIS
#Get number of missing values(use of sapply)
sapply(df,function(x){sum(is.na(x))})

# we dont  have any missing value here
```

```
##########################################outliers

#Get the data with only numeric columns
numeric_index = sapply(df, is.numeric)
numeric_data = df[,numeric_index]

#Get the data with only factor columns
factor_data = df[,!numeric_index]

#Check for outliers using boxplots

for(i in 1:ncol(numeric_data)) {
  assign(paste0("box",i), ggplot(data = df, aes_string(y = numeric_data[,i]))
         +stat_boxplot(geom = "errorbar", width = 1)
         +geom_boxplot(outlier.colour = "red", fill = "grey", outlier.size = 1)
         +labs(y = colnames(numeric_data[i]))
         +ggtitle(paste("Boxplot: ",colnames(numeric_data[i]))))
}


#Arrange the plots in grids
gridExtra::grid.arrange(box1,box2,ncol=2)
gridExtra::grid.arrange(box3,box4,ncol=2)
gridExtra::grid.arrange(box5,box6,ncol=2)
gridExtra::grid.arrange(box7,ncol=1)

#boxplot(numeric_data)
# we have outliers in windspeed , casual column
# we will Replace all outlier data with NA
#but first we need to check which method to use for replacing the NA


#CHECKING WHICH METHOD TO USE
```

```
#Create missing value and impute using mean, median and knn
df1 =df
df2= df
df3= df


# we are using casual becase it has max missing value ,

df1[["casual"]][2]
df2[["casual"]][2]
df3[["casual"]][2]


# here the value we have choosen to remove is 131

df1[["casual"]][2] =NA
df2[["casual"]][2]= NA
df3[["casual"]][2] = NA

# checking for different values

#mean

df1[["casual"]][2] = mean(df1$casual, na.rm = T)
df1[["casual"]][2] #the value we got is 849.1589


#median


df2[["casual"]][2] = median(df1$casual, na.rm = T)
df2[["casual"]][2] #the value we got is 721
```

```
#KNN
df3 = knnImputation(data = df3, k = 3)
df3[["casual"]][2]  # we got the value 349  , so we going ahead with KNN



#Replacing all outlier data with NA
numeric_columns = colnames(numeric_data)

for(i in numeric_columns){
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(paste(i,length(val)))
  df[,i][df[,i] %in% val] = NA
}

# KNN INPUTATION METHOD TO REPLACE THE OUTLIERS on DF

df[,9:15] = knnImputation(df[ ,9:15], k = 3)

# CHECKING BOXPLOT ----------------------------AGAIN--------
for(i in 1:ncol(numeric_data)) {
  assign(paste0("box_after",i), ggplot(data = df, aes_string(y = numeric_data[,i]))
         +stat_boxplot(geom = "errorbar", width = 1)
         +geom_boxplot(outlier.colour = "red", fill = "grey", outlier.size = 1)
         +labs(y = colnames(numeric_data[i]))
         +ggtitle(paste("Boxplot: ",colnames(numeric_data[i]))))
}

#Arrange the plots in grids
gridExtra::grid.arrange(box_after1,box2,ncol=2)
gridExtra::grid.arrange(box3,box4,ncol=2)
```

```
gridExtra::grid.arrange(box5,box6,ncol=2)
gridExtra::grid.arrange(box7,ncol=1)


#again outliers on hum and windspeed

val_windspeed = df$windspeed[df$windspeed %in%
boxplot.stats(df$windspeed)$out]
df = df[which(!df$windspeed%in%val_windspeed),]


val_hum = df$hum[df$hum %in% boxplot.stats(df$hum)$out]
df = df[which(!df$hum%in%val_hum),]


# CASUAL STILL HAS OUTLIER BUT WINDSPEED  and hum HASNT
numeric_index = sapply(df, is.numeric)
numeric_data = df[,numeric_index]


########FEATURE SELECTION

#usdm library  # done for the checking correlation
#Check for multicollinearity
vifcor(numeric_data)

#Check for multicollinearity using corelation graph
corrdf = cor(numeric_data,numeric_data, method = "spearman" )

# dropping the casual and registerd
to_drop <- c("casual","registered","atemp")
df=df[ ,  -which(names(df) %in% to_drop)]
```

```
# removing the date column as it wont be used in model
df=df[ ,  -which(names(df) %in% "dateday")]


##########DECISION TREE


#Splitting the  data (80-20 percent)
set.seed(1)
train_index = sample(1:nrow(df), 0.8*nrow(df))
train = df[train_index,]
test = df[-train_index,]


#Build decsion tree using rpart
dt_model = rpart(count~ ., data = train, method = "anova")
# here we can try any method other than anova ,
#one of "anova", "poisson", "class" or "exp".
#If method is missing then the routine tries to make an intelligent guess.

#Ploting the tree
rpart.plot(dt_model)


#Perdict for test cases

dt_predictions = predict(dt_model, test[,-11])

#dataframe_pred_dt= data.frame((dt_predictions))

#Create data frame for actual and predicted values
```

```
df_pred_dt = data.frame("actual"= test[,11], "dt_pred"=dt_predictions)


# analyse relationship between actual and predicted count
ggplot(df_pred_dt, aes(x= actual ,y=dt_predictions)) +
  geom_point()+
  geom_smooth()

################ ERROR AND R2 SQ ########################



#calculate MAE
MAE = function(y_true, y_pred){
  mean(abs((y_true - y_pred)/y_true))
}

#R2
R2 =function(y_true, y_pred){
  rss =sum((y_pred - y_true) ^ 2)  ## residual sum of squares
  tss =sum((y_true - mean(y_true)) ^ 2)  ## total sum of squares
  rsq =1 - rss/tss

}



#accuracy and error(decision tree)

error_dt = MAE(test[,11], dt_predictions)
print("accuracy of decision tree ")
error_dt_per = (error_dt*100)
print(100-(error_dt_per))
```

```
#81.8051

r2_value_dt = R2(test[,11], dt_predictions)
print("R2 sq value for  decision tree ")
print(r2_value_dt)


##########RANDOM FOREST


#Training the model using training data
rf_model = randomForest(count~., data = train, ntree = 500)

#Predict the test cases
rf_predictions = predict(rf_model, test[,-11])


#Create data frame for actual and predicted values
df_pred_rf = data.frame("actual"= test[,11], "rf_predictions"=rf_predictions)

# analyse relationship between actual and predicted count
ggplot(df_pred_rf, aes(x= actual ,y=rf_predictions)) +
  geom_point()+
  geom_smooth()



#errors and accuracy
error_rf= MAE(test[,11], rf_predictions)
error_rf_per = error_rf*100
print(error_rf)
print("accuracy of Random forest ")
```

```
print(100-(error_rf_per))


r2_value_rf = R2(test[,11], rf_predictions)
print("R2 sq value for  random forest ")
print(r2_value_rf)




##########LINEAR REGRESSION
#check multicollearity
library(usdm)
#converting multilevel categorical variable into ineger again
df$season=as.integer(df$season)
df$month=as.integer(df$month)
df$year=as.integer(df$year)
df$holiday=as.integer(df$holiday)
df$weekday=as.integer(df$weekday)
df$workingday=as.integer(df$workingday)
df$weathersit=as.integer(df$weathersit)

vif(df[,1:11])

vifcor(df[,1:11], th = 0.9)

# develop Linear Regression  model

set.seed(1)
train_index = sample(1:nrow(df), 0.8*nrow(df))
train_lm = df[train_index,]
test_lm = df[-train_index,]
```

```
#run regression model
lm_model = lm(count ~., data = train_lm)

#Summary of the model
summary(lm_model)


# observe the  residuals and   coefficients  of the linear regression model
# Predict  the Test data

#Predict
lm_predictions = predict(lm_model, test_lm[,-11])


#Create data frame for actual and predicted values
df_pred_lm = data.frame("actual"= test[,11], "lm_predictions"=lm_predictions)

# analyse relationship between actual and predicted count
ggplot(df_pred_lm, aes(x= actual ,y=lm_predictions)) +
  geom_point()+
  geom_smooth()

# Evaluate Linear Regression Model
#MAE

error_lm = MAE(test_lm[,11], lm_predictions)
error_lm_per = error_lm*100
print("accuracy  of linear model")
print(100-(error_lm_per))


r2_value_lm = R2(test[,11], lm_predictions)
print("R2 sq value for linear model ")
```

```
print(r2_value_lm)



################
print("accuracy of decision tree ")
print(100-(error_dt_per))
print("R2 sq value for  decision tree ")
print(r2_value_dt)

#  acc : 73.67317   r2 : 0.7303338

print("accuracy of Random forest ")
print(100-(error_rf_per))

print("R2 sq value for  random forest ")
print(r2_value_rf)

#Acc :83.2571  R2 :0.863417


#
print("accuracy  of linear model")
print(100-(error_lm_per))

print("R2 sq value for linear model ")
print(r2_value_lm)

#acc :  R2 :0.7813869
```

write.csv(df_pred_rf, file = ' R bike outout .csv', row.names = FALSE, quote=FALSE)

5.3 >> References

https://stackoverflow.com

https://towardsdatascience.com/