Introduction to the Case Study

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy.stats import chi2_contingency
from scipy.stats import ttest_ind
from scipy.stats import f_oneway
```

```
df=pd.read_csv('/content/drive/MyDrive/Yulu.csv')
df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

10886 rows × 12 columns

```
df.shape

## There are 10886 rows and 12 columns present in the Dataset
```

```
(10886, 12)
```

```
df.info()

## No null values are present in any of the columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

Dependant variable in our problem statement is Count. hence we have to find what other variables are affecting this variable.

```
df.describe()
```

	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

```
df.describe(include='object')
```

	season	holiday	workingday	weather
count	10886	10886	10886	10886
unique	4	2	2	4
top	4	0	1	1

Converting Numerical to Categorical Columns

Datatype of following attributes needs to changed to proper data type

- datetime - to datetime
- season - to categorical
- holiday - to categorical
- workingday - to categorical
- weather - to categorical

```
df['datetime']=pd.to_datetime(df['datetime'])

cat_cols=['season','holiday','workingday','weather']

for cat in cat_cols:
  df[cat]=df[cat].astype('object')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

```
df.iloc[:, 1:].describe(include='all')
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	
count	10886.0	10886.0	10886.0	10886.0	10886.00000	10886.000000	10886.000000	10886.000000	108
unique	4.0	2.0	2.0	4.0	NaN	NaN	NaN	NaN	
top	4.0	0.0	1.0	1.0	NaN	NaN	NaN	NaN	
freq	2734.0	10575.0	7412.0	7192.0	NaN	NaN	NaN	NaN	
mean	NaN	NaN	NaN	NaN	20.23086	23.655084	61.886460	12.799395	
std	NaN	NaN	NaN	NaN	7.79159	8.474601	19.245033	8.164537	
min	NaN	NaN	NaN	NaN	0.82000	0.760000	0.000000	0.000000	
25%	NaN	NaN	NaN	NaN	13.94000	16.665000	47.000000	7.001500	
50%	NaN	NaN	NaN	NaN	20.50000	24.240000	62.000000	12.998000	
75%	NaN	NaN	NaN	NaN	26.24000	31.060000	77.000000	16.997900	
max	NaN	NaN	NaN	NaN	41.00000	45.455000	100.000000	56.996900	3

'Casual' and 'registered' have several outliers based on their mean and median values. Also the standard deviation for them is quite high which tells that there is high variance in the data of these attributes

```
df.isna().sum()
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

There are no missing values present in the data set

```
df['datetime'].max() , df['datetime'].min()
```

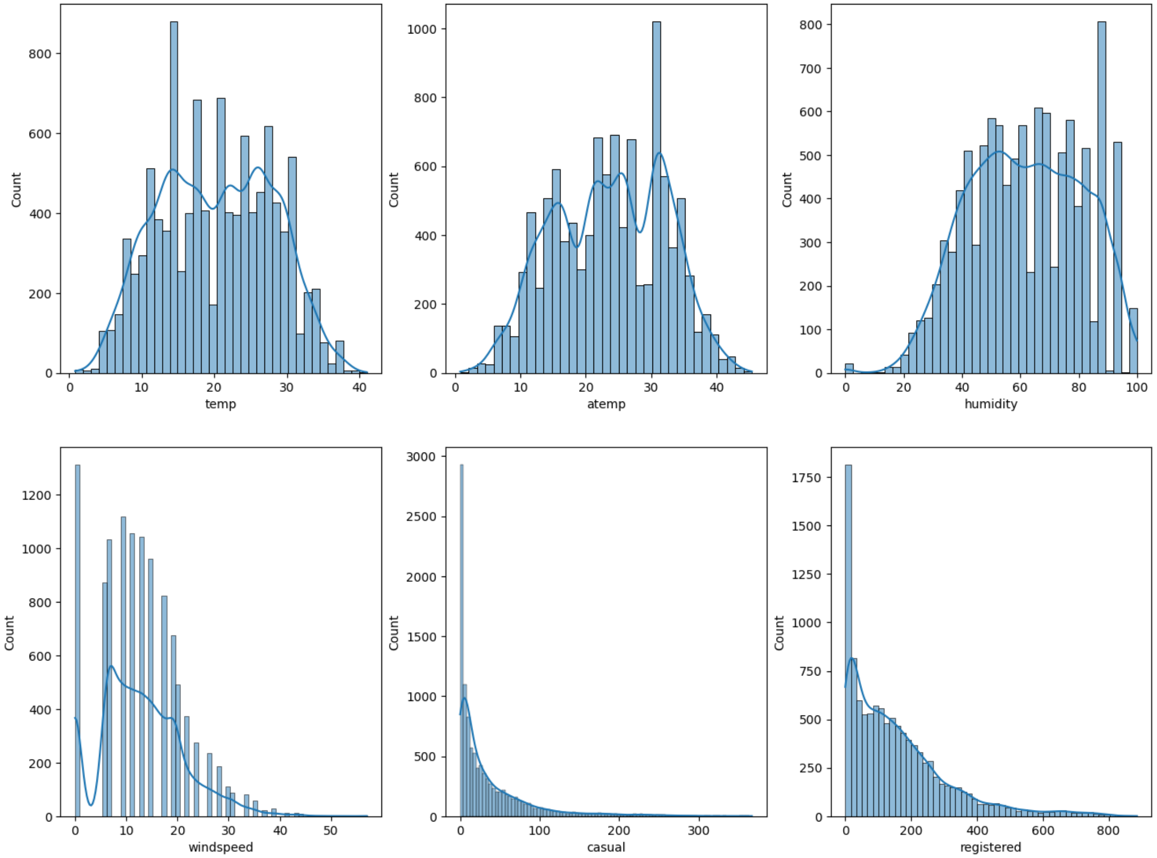
```
(Timestamp('2012-12-19 23:00:00'), Timestamp('2011-01-01 00:00:00'))
```

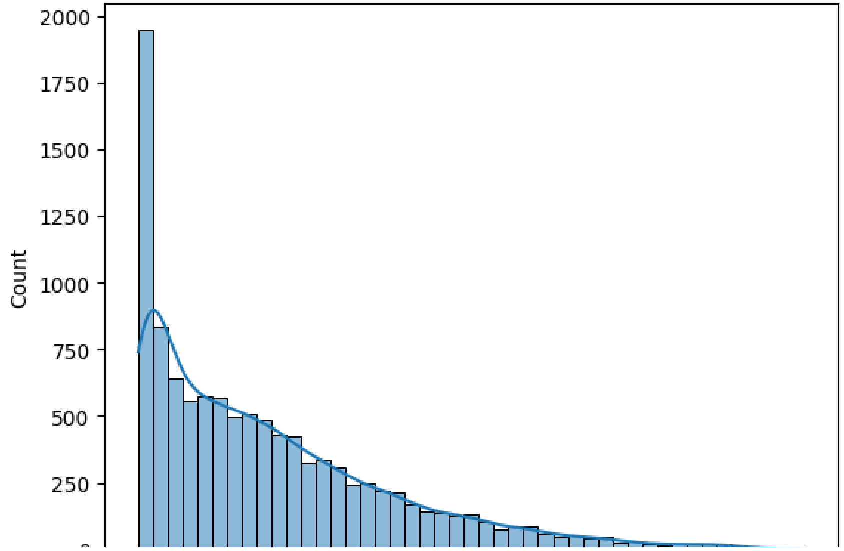
```
df[cat_cols].melt().groupby(['variable', 'value'])[['value']].count()
```

		value
variable	value	
holiday	0	10575
	1	311
season	1	2686
	2	2733
	3	2733
	4	2734
weather	1	7192
	2	2834
	3	859
	4	1
workingday	0	3474
	1	7412

Univariate Analysis

```
num_cols=['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']

fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16, 12))

index=0

for i in range (2):
  for j in range (3):
    sns.histplot(df[num_cols[index]],ax=axis[i,j],kde=True)
    index += 1

plt.show()
sns.histplot(df[num_cols[-1]],kde=True)
plt.show()
```

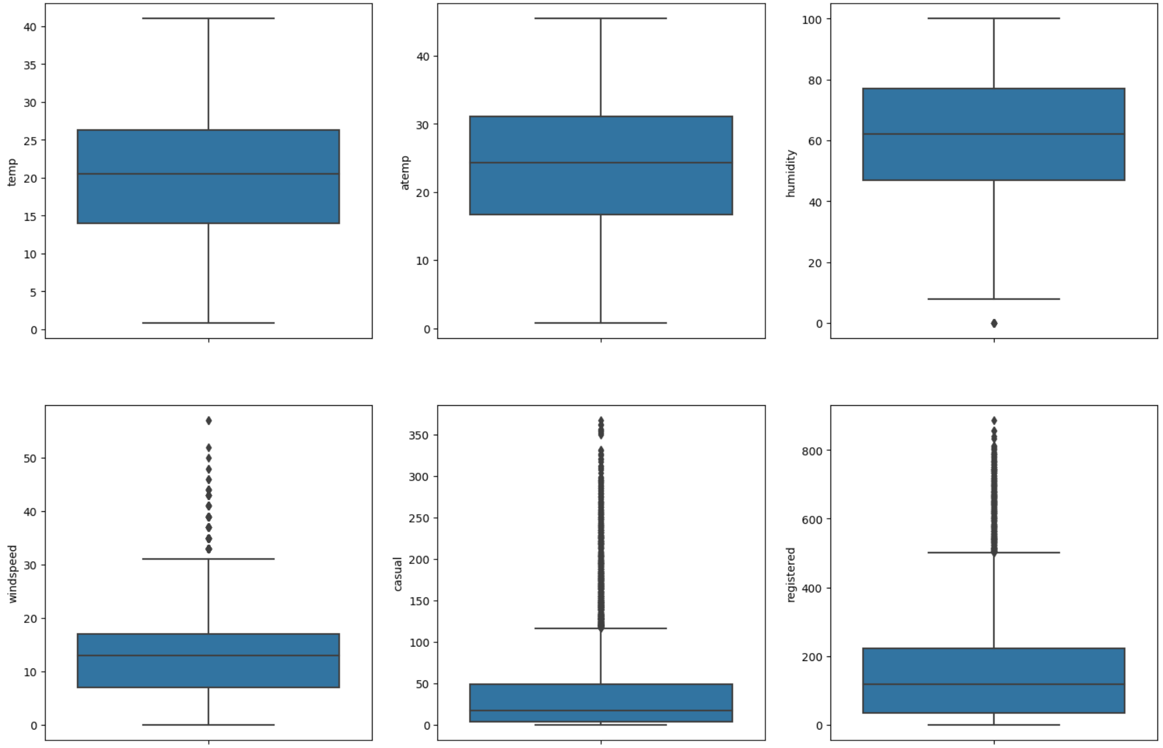

Count
temp
atemp
humidity
windspeed
casual
registered

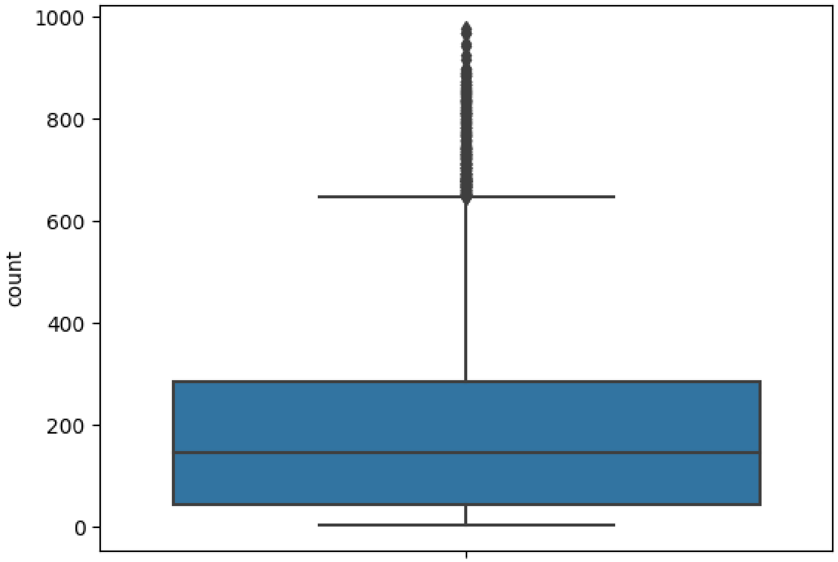

Count

- casual, registered and count somewhat looks like Log Normal Distrinution
- temp, atemp and humidity looks like they follows the Normal Distribution
- windspeed follows the binomial distribution

Outlier Detection

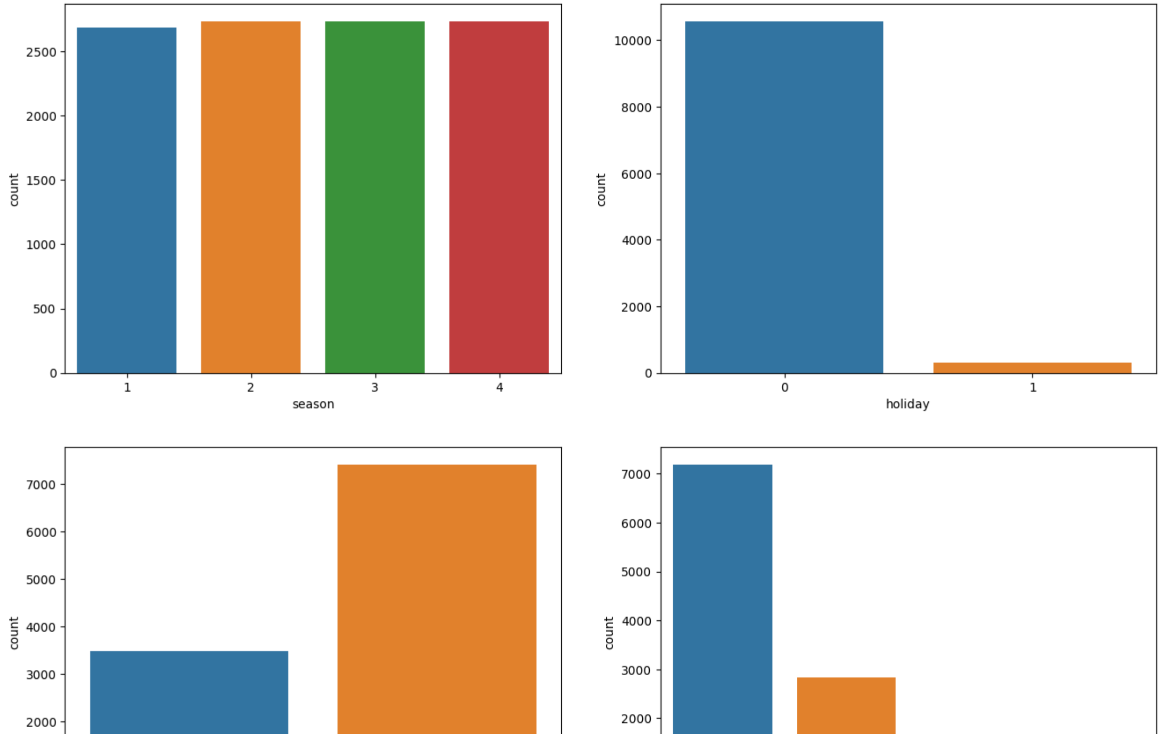
```
## Plotting a boxplot to check for outliers

fig,axis= plt.subplots(nrows=2,ncols=3,figsize=(18,12))

index=0

for i in range (2):
  for j in range (3):
    sns.boxplot(y=df[num_cols[index]],ax=axis[i,j])
    index += 1
plt.show()
sns.boxplot(y=df[num_cols[-1]])
plt.show()
```


temp
atemp
humidity
windspeed
casual
registered


count

Windspeed, casual, registered and count show to have a lot of outliers in the data

```
# Creating countplots for all categorical variables

fig,axis= plt.subplots(nrows=2,ncols=2,figsize=(16,12))
index=0

for i in range (2):
  for j in range (2):
    sns.countplot(x=df[cat_cols[index]],ax=axis[i,j])
    index +=1

plt.show()
```



For the above analysis the data looks quite common as there are:

- Equal number of days across each season
- More working days than there are holidays
- Weather is mostly Clear, Few clouds, partly cloudy, partly cloudy

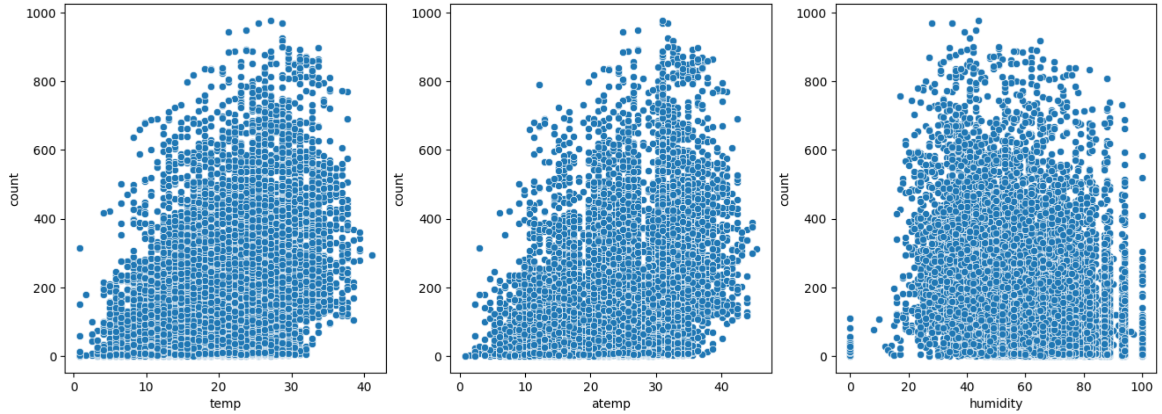
Bivariate Analysis

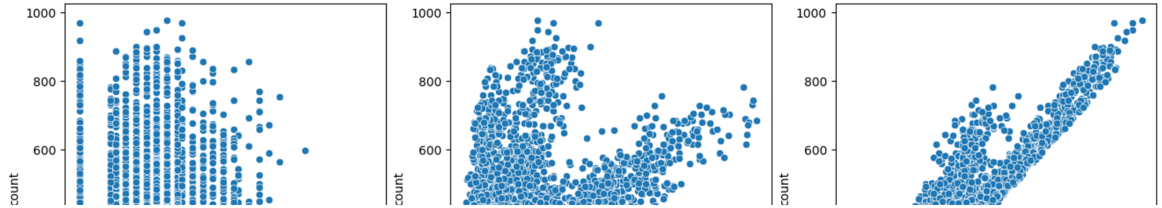
```
## Plotting Categorical Vs Count variables

fig,axis= plt.subplots(nrows=2,ncols=2, figsize=(16,12))

index=0

for i in range (2):
  for j in range (2):
    sns.boxplot(x=df[cat_cols[index]], y=df['count'],ax=axis[i,j])
    index += 1

plt.show()
```

The following can be concluded from the bivaraite analysis of categorical variables in comparision to the 'count' :

- In summer and fall seasons more bikes are rented as compared to other seasons.
- Whenever its a holiday more bikes are rented.
- It is also clear from the workingday also that whenever day is holiday or weekend, slightly more bikes were rented.
- Whenever there is rain, thunderstorm, snow or fog, there were less bikes were rented.


```
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16, 12))

index = 0
for row in range(2):
    for col in range(3):
        sns.scatterplot(data=df, x=num_cols[index], y='count', ax=axis[row, col])
        index += 1

plt.show()
```





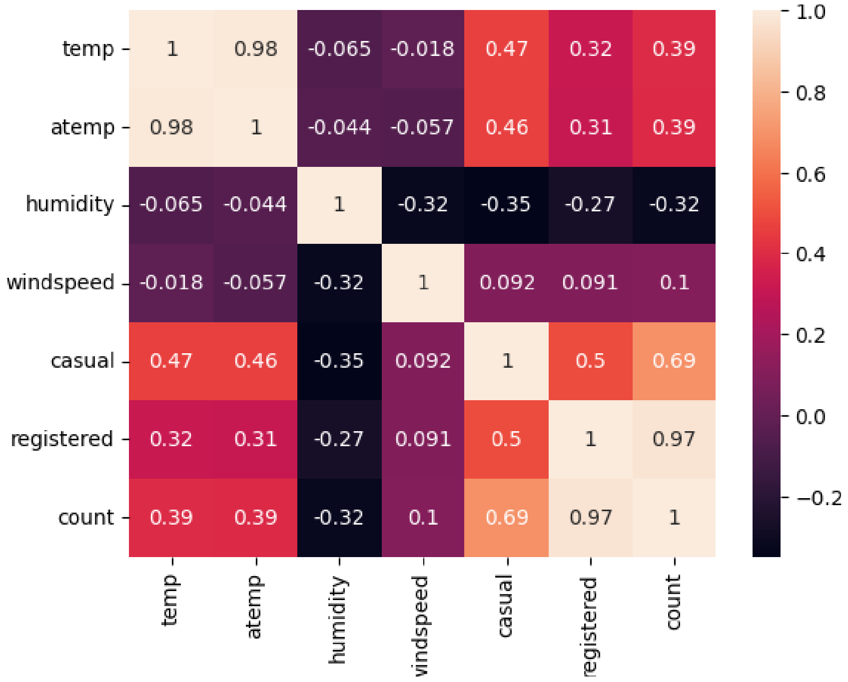
- Whenever the humidity is less than 20, number of bikes rented is very very low.
- Whenever the temperature is less than 10, number of bikes rented is less.
- Whenever the windspeed is greater than 35, number of bikes rented is less.

```
## Understanding the correlation between the count and numerical variables

df.corr()['count']
```

```
<ipython-input-17-05ab75d8c625>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only
  df.corr()['count']
temp          0.394454
atemp         0.389784
humidity     -0.317371
windspeed     0.101369
casual        0.690414
registered    0.970948
count         1.000000
Name: count, dtype: float64
```

```
sns.heatmap(df.corr(),annot=True)
plt.show()
```

```
<ipython-input-18-f6412ee67fb3>:1: FutureWarning: The default value of numeric_only in DataFrame.corr i
  sns.heatmap(df.corr(),annot=True)
```



Assumptions Test for Anova

Anova assumes normality and also it assumes variances is same across all groups.

Normality test

Shapiro-Wilk's test

We will test the

- Null hypothesis: Count follows normal distribution
- Alternative hypothesis: Count doesn't follow normal distribution

```
from scipy.stats import shapiro
# find the p-value
w, p_value = shapiro(df['count'])
print('The p-value is', p_value)
```

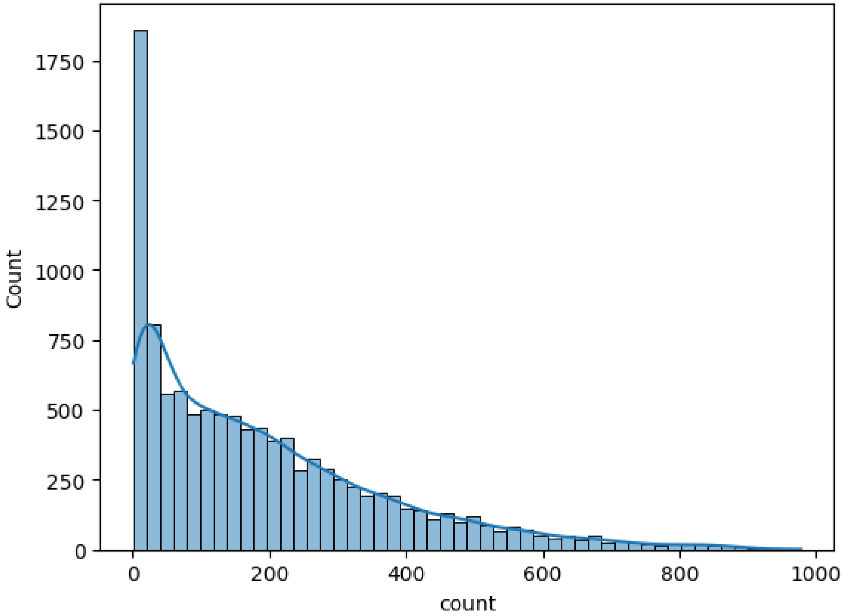
```
The p-value is 0.0
/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py:1816: UserWarning: p-value may not be accurate for N > 5000.
```

```
        warnings.warn("p-value may not be accurate for N > 5000.")
```

As mentioned on documentation of scipy, when no of data points>5000, this test fails. Let's move to another test

```
# Normality test using Distplot

sns.histplot(df['count'],bins=50,kde=True)
```

```
<Axes: xlabel='count', ylabel='Count'>
```



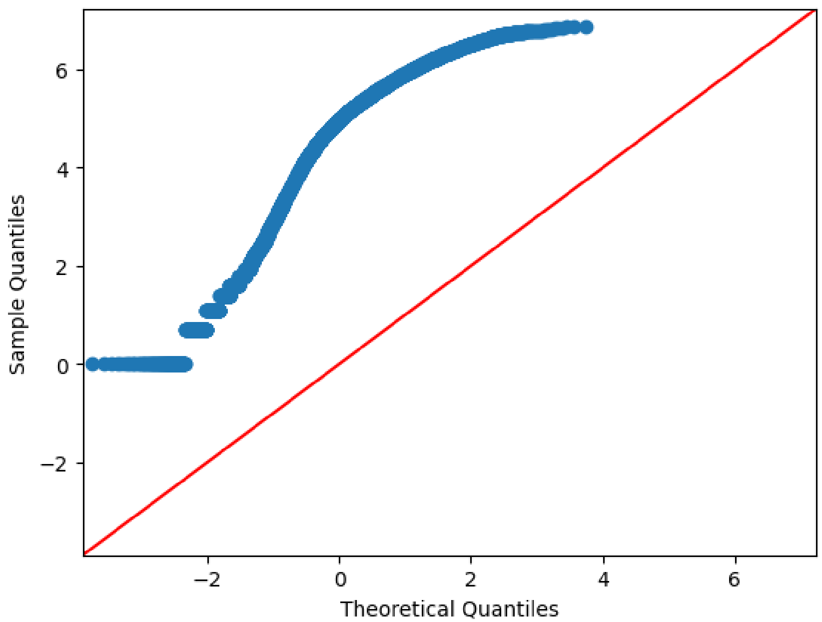

```
df['log_count']=np.log(df['count'])
sns.histplot(df['log_count'],bins=50,kde=True)
```

```
<Axes: xlabel='log_count', ylabel='Count'>
```

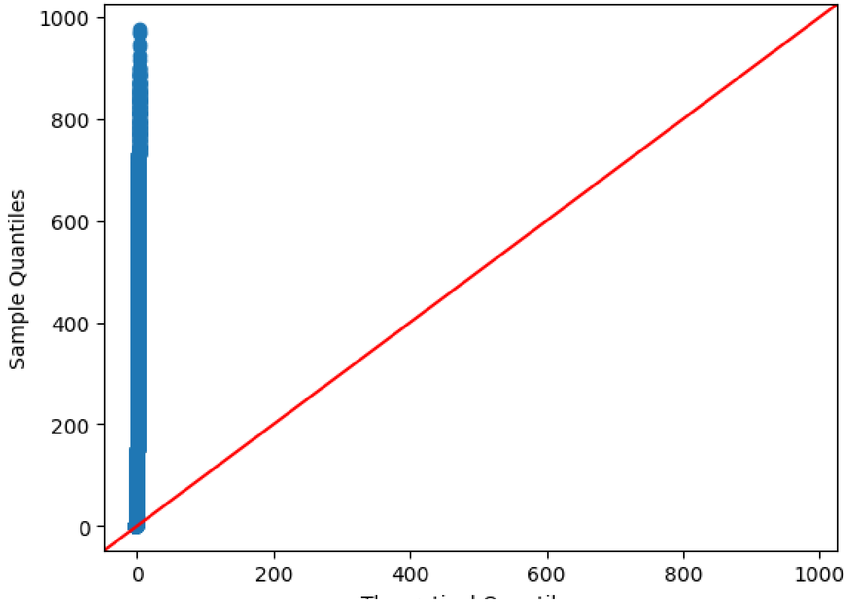
Another test that can be done to check the Normality is the Q-Q Plot

```
import numpy as np
import statsmodels.api as sm
```

```
sm.qqplot(df['log_count'], line ='45')
plt.show()
```




```
sm.qqplot(df['count'], line ='45')
plt.show()
```



None of the above two q-q plots follows a normal distribution but the log_count is comparitively better hence we will proceed with log_count as we have huge data.

Hypothesis Testing -1

Null Hypothesis (H0): Weather is independent of the season

Alternate Hypothesis (H1): Weather is not independent of the season

Significance level (alpha): 0.05

We will use chi-square test to test hypyothesis defined above.

```
data_tab= pd.crosstab(df['season'],df['weather'])
print('Observed values:')
data_tab
```

Observed values:

weather	1	2	3	4
season				

Let us not include weather is equal to 4 in our Hypothesis.

```
data= df[df.weather !=4]
data
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	regis
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	
...	...	...	...	...	...	...	...	...	...	...	
	2012-12-										

```
contingency = pd.crosstab(data['season'],data['weather'])
contingency
```

weather	1	2	3
season			
1	1759	715	211
2	1801	708	224
3	1930	604	199
4	1702	807	225

```
chi2, pval, dof, exp_freq = chi2_contingency(contingency)
print("chi-square statistic: {} , Pvalue: {} , Degree of freedom: {} ,expected frequency:{} ".format(chi2, pval, dof, exp_freq))
```

```
chi-square statistic: 46.10145731073249 , Pvalue: 2.8260014509929343e-08 , Degree of freedom: 6 ,expected frequency:[[1774.04869086  699.06201194  211.8892972 ]
 [1805.76352779  711.55920992  215.67726229]
 [1805.76352779  711.55920992  215.67726229]
 [1806.42425356  711.81956821  215.75617823]]
```

P-value is very low. Null Hypothesis is rejected. Hence weather and season are dependant on each other.

Hypothesis Testing 2

- H0- Working day has no affect on the number of cycles rented
- Ha- Working day has an affect on the number of cycles rented
- Signifance level = 0.05
- We will use a 2 sample T-Test to test the hypothesis stated

```
data_group1 = df[df['workingday']==0]['count'].values
data_group2 = df[df['workingday']==1]['count'].values
```

```
np.var(data_group1), np.var(data_group2)
```

```
(30171.346098942427, 34040.69710674686)
```

Before conducting the two-sample T-Test we need to find if the given data groups have the same variance. If the ratio of the larger data groups to the small data group is less than 4:1 then we can consider that the given data groups have equal variance.

Here, the ratio is 34040.70 / 30171.35 which is less than 4:1

```
t_statistic, p_value = ttest_ind(data_group1,data_group2,alternative='two-sided')
alpha=0.05
print(p_value)
if p_value < alpha:
  print('We reject the Null Hypothesis and the working day has an effect on count')
else :
  print('We do not reject the Null Hypothesis and the working day has no effect on the count variable ')
```

```
0.22644804226361348
We do not reject the Null Hypothesis and the working day has no effect on the count variable
```

Hypothesis Testing 3- Anova

Checking whether the weather and season have an effect on the count variable

- NULL HYPOTHESIS: μ1 = μ2 = μ3 = μ4 The mean Count in every season is same.
- ALTERNATIVE HYPOTHESIS: Atleast one of mean of count is not same

Anova and all parametric tests assume-

- Normality:- Values in each sampled groups are assumed to be drawn from normally distributed populations. We can use normal probability plot or Q-Q plot to check normality.
- Homogeneity of variance:- All the c group variances are equal, that is $\sigma_1^2 = \sigma_2^2 = \sigma_3^2 = \ldots = \sigma_c^2$.

Test for Equality of Variances

Levene’s test

We will test the

Null hypothesis : All the count variances are equal

against the

Alternative hypothesis : At least one variance is different from the rest.

```
df.groupby('season')['log_count'].describe()
```

season	count	mean	std	min	25%	50%	75%	max
1	2686.0	3.984206	1.539737	0.0	3.178054	4.356709	5.099866	6.685861
2	2733.0	4.703267	1.462172	0.0	3.891820	5.147494	5.771441	6.771936
3	2733.0	4.860311	1.378662	0.0	4.219508	5.273000	5.849325	6.884487
4	2734.0	4.652650	1.421134	0.0	3.931826	5.081404	5.683580	6.854355

We see that the STD across all the seasons is the same hence it should not have much variance. Let's check for the same statistically

```
from scipy.stats import levene

t_stat,p_value= levene(
    df[df['season']==1]['log_count'],
    df[df['season']==2]['log_count'],
    df[df['season']==3]['log_count'],
    df[df['season']==4]['log_count']
)

print('The p value is : ', p_value)

if p_value< 0.05:
  print('Reject the null hypothesis :At least one variance is different from the rest. ')
```

```
else:
  print('Accept the null Hypothesis: Variances are the same across')
```

```
The p value is :  2.3678125658230693e-06
Reject the null hypothesis :At least one variance is different from the rest.
```

```
from scipy.stats import f_oneway
# find the p-value
test_stat, p_value = f_oneway(df[df['season']==1]['log_count'].sample(2686),
 df[df['season']==2]['log_count'].sample(2686),
 df[df['season']==3]['log_count'].sample(2686),
 df[df['season']==4]['log_count'].sample(2686))
# print the p-value
print('The p-value is', p_value)
```

```
The p-value is 1.449081485998964e-120
```

P-value is low. So Null hypothesis is False.

The mean Count in every season is not same.i.e. season effect count.

proved statistically.

```
gp_1= df[df['weather']==1]['count'].values
gp_2= df[df['weather']==2]['count'].values
gp_3= df[df['weather']==3]['count'].values
gp_4= df[df['weather']==4]['count'].values

gp_5= df[df['season']==1]['count'].values
gp_6= df[df['season']==2]['count'].values
gp_7= df[df['season']==3]['count'].values
gp_8= df[df['season']==4]['count'].values

# Performing an anova test on the same

f_oneway(gp_1,gp_2,gp_3,gp_4,gp_5,gp_6,gp_7,gp_8)
```

```
F_onewayResult(statistic=127.96661249562491, pvalue=2.8074771742434642e-185)
```

Since p_value is less than alpha = 0.05 we reject the null hypothesis which states that :

- Number of cycles rented is different in different weather and seasons

Key Takeaways

- Less Number of people take vehicle in spring season and more no bookings happen in Fall season

- Less number of people hire cycle when it is slow rain and more number of people hire when weather is clear.
- Above pictures tells us that Count is linearly related to temp and inversely related to humidity & windspeed.
- It is also clear from the workingday that whenever day is holiday or weekend, slightly more bikes were rented.
- Whenever there is rain, thunderstorm, snow or fog, there were less bikes were rented.

Recommendations

- With a significance level of 0.05, workingday has no effect on the number of bikes being rented.
- In summer and fall seasons the company should have more bikes in stock to be rented. Because the demand in these seasons is higher as compared to other seasons.
- In very low humid days, company should have less bikes in the stock to be rented.
- Whenever temprature is less than 10 or in very cold days, company should have less bikes.