Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm,t,binom,expon,chi2,chisquare,chi2_contingency,ttest_1samp,ttest_rel,ttest_ind,f,f_oneway
from scipy.stats import kruskal,levene,shapiro,kstest, probplot
from statsmodels.graphics.gofplots import qqplot
```

Import Dataset

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089 -O yulu_data.csv
```

```
--2023-10-16 09:52:46--  https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)... 18.172.139.94, 18.172.139.210, 18.172.139.46, ...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)|18.172.139.94|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 648353 (633K) [text/plain]
Saving to: ‘yulu_data.csv’

yulu_data.csv       100%[===================>] 633.16K  --.-KB/s    in 0.05s

2023-10-16 09:52:46 (11.6 MB/s) - ‘yulu_data.csv’ saved [648353/648353]
```

```
df = pd.read_csv('yulu_data.csv')
```

Analyse the Dataset

Analysing the structure & characteristics of the dataset

```
df.head()
```

	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	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
df.shape
```

```
(10886, 12)
```

```
df.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='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  object
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64  
 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: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

Changing the data types of columns

```
#changing to categorical type columns

for i in df.columns[1:5]:
  df[i] = df[i].astype('category')
```

```
#changing to date time type columns

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

```
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  category      
 2   holiday     10886 non-null  category      
 3   workingday  10886 non-null  category      
 4   weather     10886 non-null  category      
 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: category(4), datetime64[ns](1), float64(3), int64(4)
memory usage: 723.7 KB
```

Replacing the values of categorical columns

```
# replacing values of season columns
def season_category(x):
    if x == 1:
        return 'spring'
    elif x == 2:
        return 'summer'
    elif x == 3:
        return 'fall'
    else:
        return 'winter'
df['season'] = df['season'].apply(season_category)
```

```
#replacing values of holiday columns
df['holiday'].replace(1, 'holiday', inplace=True)
df['holiday'].replace(0, 'not holiday', inplace=True)
```

```
# replacing values of working day columns
df['workingday'].replace(1, 'working day', inplace=True)
df['workingday'].replace(0, 'holiday/weekend', inplace=True)
```

Statistical Summary

```
# date time
from_d = df.datetime.min().date()
to_d = df.datetime.max().date()
delta = to_d - from_d
```

```
print("The data is given from date:",from_d, "to date:", to_d,"and" )
print("Total", delta.days, "days data is given in the dataset.")
```

```
The data is given from date: 2011-01-01 to date: 2012-12-19 and
Total 718 days data is given in the dataset.
```

```
#Statistical Summary for categorical data
df.describe(include= 'category')
```

	season	holiday	workingday	weather
count	10886	10886	10886	10886
unique	4	2	2	4
top	winter	not holiday	working day	1
freq	2734	10575	7412	7192

```
#Statistical Summary for numerical data
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

```
#percentage of casual users in total users
(df['casual'].sum()/df['count'].sum())*100
```

```
18.8031413451893
```

```
#percentage od registered users in total users
(df['registered'].sum()/df['count'].sum())*100
```

```
81.1968586548107
```

Missing value & Duplicates

```
#Missing Values

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
```

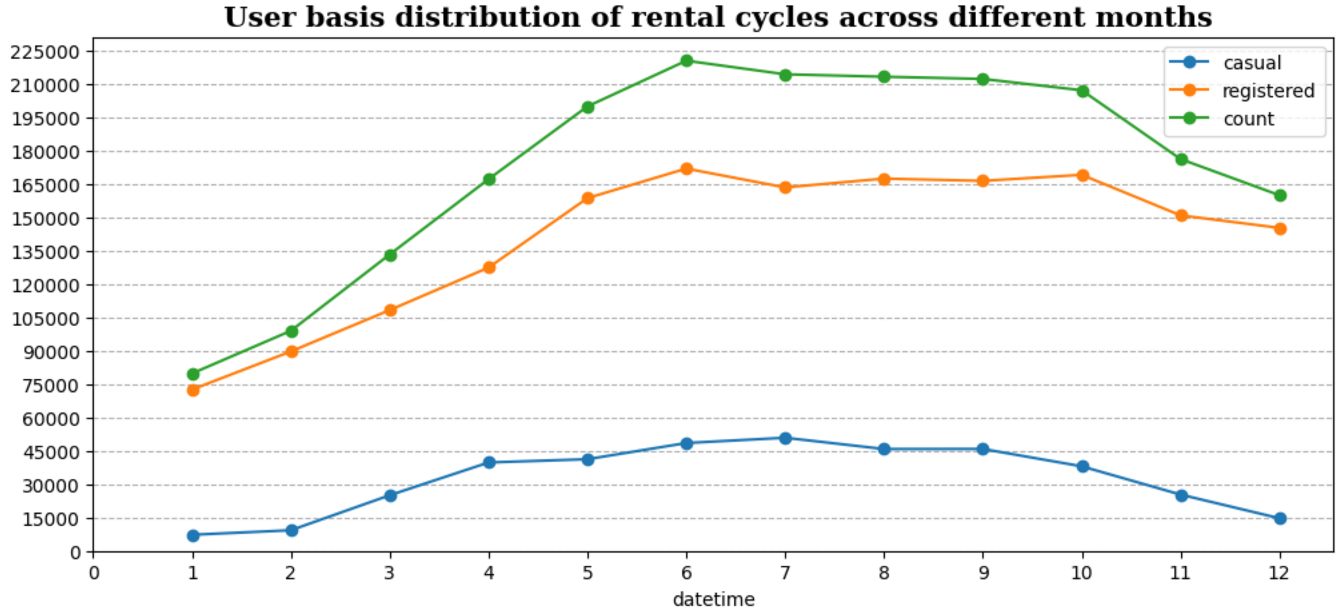
```
#Duplicate Values

df.duplicated().sum()
```

```
0
```

Date time column Analysis

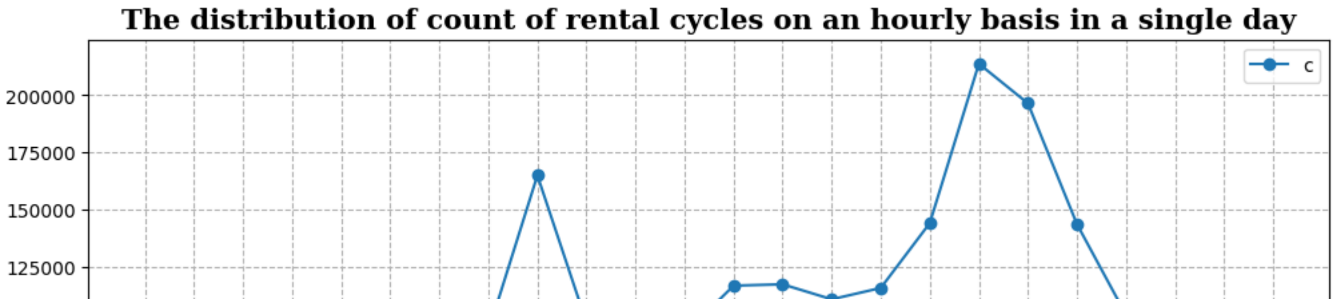
Monthly Basis Analysis

```
# The below code visualizes the trend of the monthly total values for the 'casual', 'registered',
    # and 'count' variables,  allowing for easy comparison and analysis of their patterns over time

plt.figure(figsize = (12, 5))

# plotting a lineplot on a monthly basis, and calculating the sum of 'casual', 'registered' and 'count' users for each month
df.groupby(by = df['datetime'].dt.month)['casual'].sum().plot(kind = 'line', legend = 'casual', marker = 'o')
df.groupby(by = df['datetime'].dt.month)['registered'].sum().plot(kind = 'line', legend = 'registered', marker = 'o')
df.groupby(by = df['datetime'].dt.month)['count'].sum().plot(kind = 'line', legend = 'count', marker = 'o')

plt.grid(axis = 'y', linestyle = '--')      # adding gridlines only along the y-axis
plt.yticks(np.arange(0, 230000,15000))
plt.xticks(np.arange(0,13,1))
plt.ylim(0,)
plt.xlim(0,)
plt.title(" User basis distribution of rental cycles across different months", font='serif', size=15, weight='bold')
plt.show()
```



Hourly Basis Analysis

```
# distribution of rental cycles on an hourly basis
plt.figure(figsize = (12, 5))
plt.title("The distribution of count of rental cycles on an hourly basis in a single day", font='serif',size=15, weight='bold')
df.groupby(by = df['datetime'].dt.hour)['count'].sum().plot(kind = 'line', marker = 'o')
plt.ylim(0,)
plt.xticks(np.arange(0, 24))
plt.legend('count')
plt.grid(axis = 'both', linestyle = '--')
plt.show()
```



Univariate Analysis

Continuous Variables

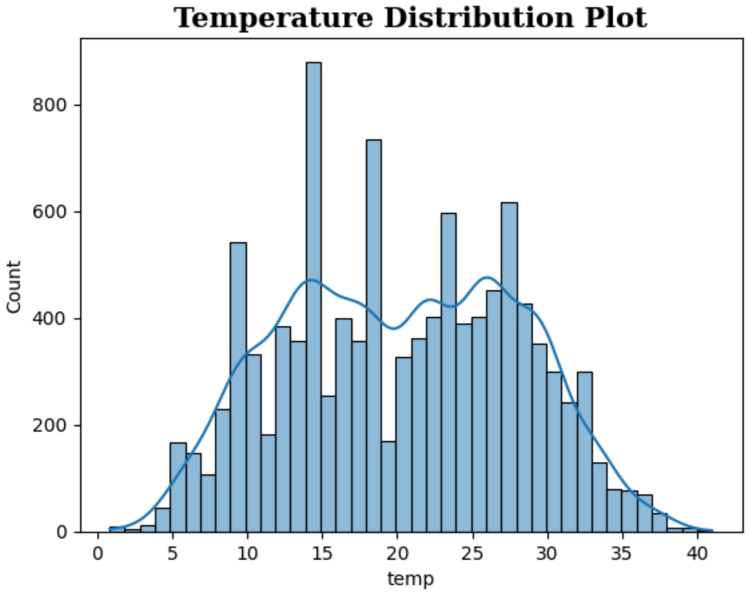
```
# Distribution for contineous variable
def distribution_plot(x):
  sns.histplot(x, kde = True, bins = 40)
  return print('mean:',x.mean(),'standard:', x.std())
```

```
# Cummalitive distribution plot for contineous variable
def cdf_plot(x):
  sns.histplot(x, kde= True, cumulative= True, stat='percent')
  plt.grid(axis='y', linestyle='--')
  plt.yticks(np.arange(0,101,10))
  return print('mean:',x.mean(),'standard:', x.std())
```

Temperature

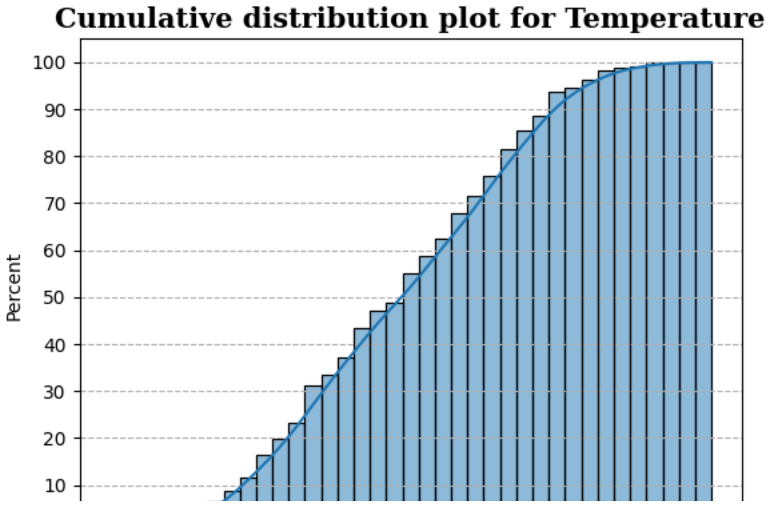
```
#temperature

distribution_plot(df['temp'])
plt.title('Temperature Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 20.23085981995223 standard: 7.791589843987567
```



```
#temperature
cdf_plot(df['temp'])
plt.title('Cumulative distribution plot for Temperature',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

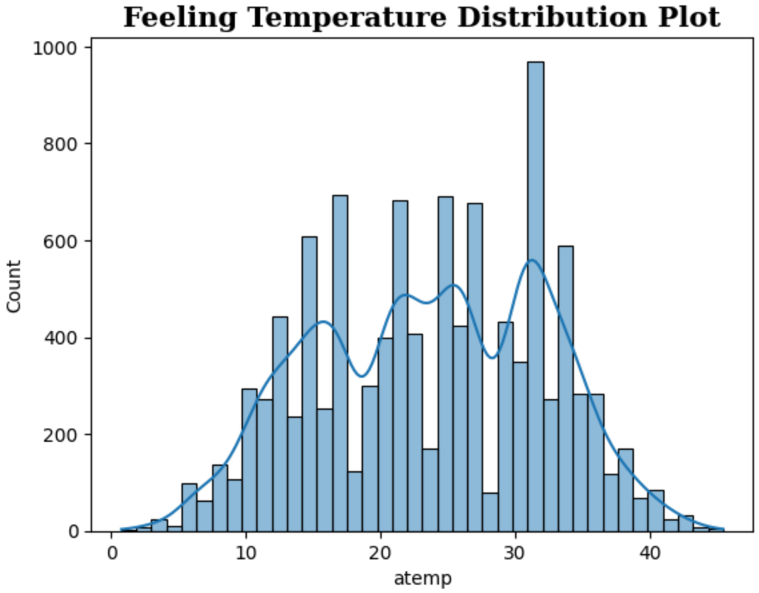
```
mean: 20.23085981995223 standard: 7.791589843987567
```



Feeling temperature

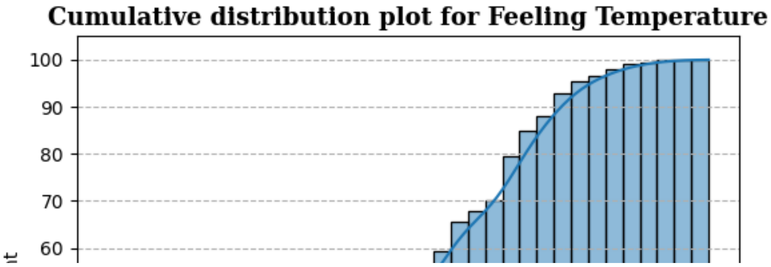
```
# feeling temperature

distribution_plot(df['atemp'])
plt.title('Feeling Temperature Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 23.655084052912 standard: 8.474600626484948
```



```
#feeling temperature

cdf_plot(df['atemp'])
plt.title('Cumulative distribution plot for Feeling Temperature',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

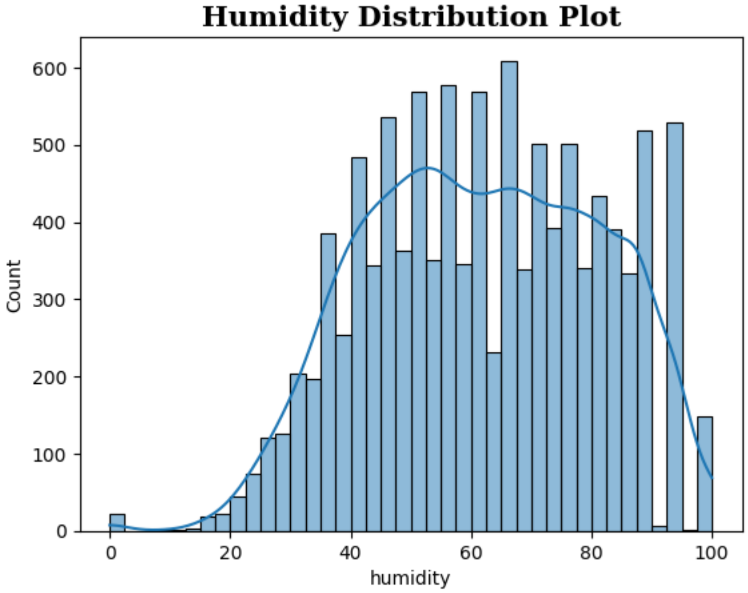
```
mean: 23.655084052912 standard: 8.474600626484948
```



Humidity

```
# humidity

distribution_plot(df['humidity'])
plt.title('Humidity Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

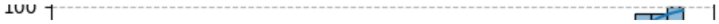
```
mean: 61.88645967297446 standard: 19.24503327739469
```



```
#humidity
cdf_plot(df['humidity'])
plt.title('Cumulative distribution plot for Humidity',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

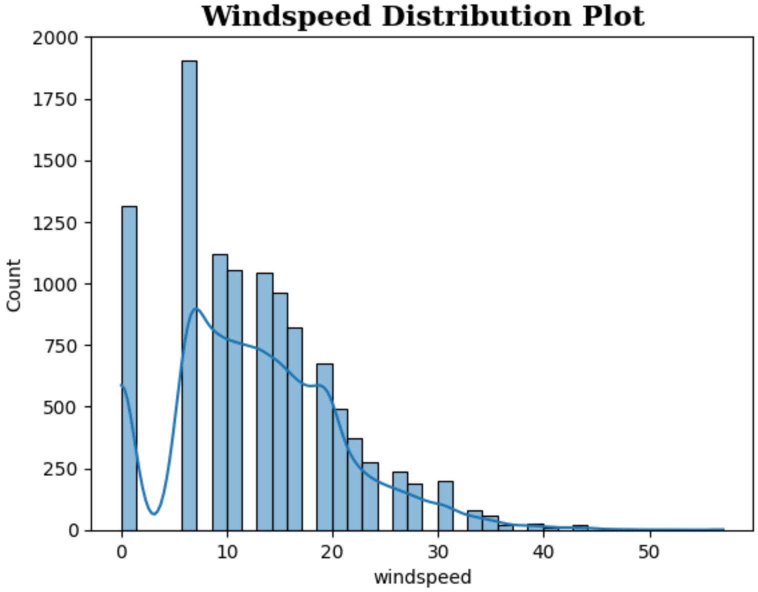
```
mean: 61.88645967297446 standard: 19.24503327739469
```

Windspeed



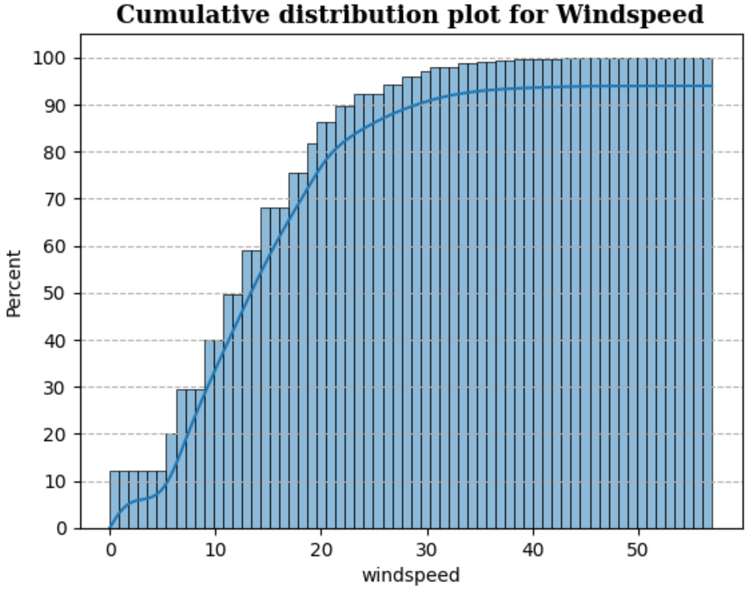
```
# Windspeed

distribution_plot(df['windspeed'])
plt.title('Windspeed Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 12.7993954069447 standard: 8.164537326838689
```



```
#Windspeed
cdf_plot(df['windspeed'])
plt.title('Cumulative distribution plot for Windspeed',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

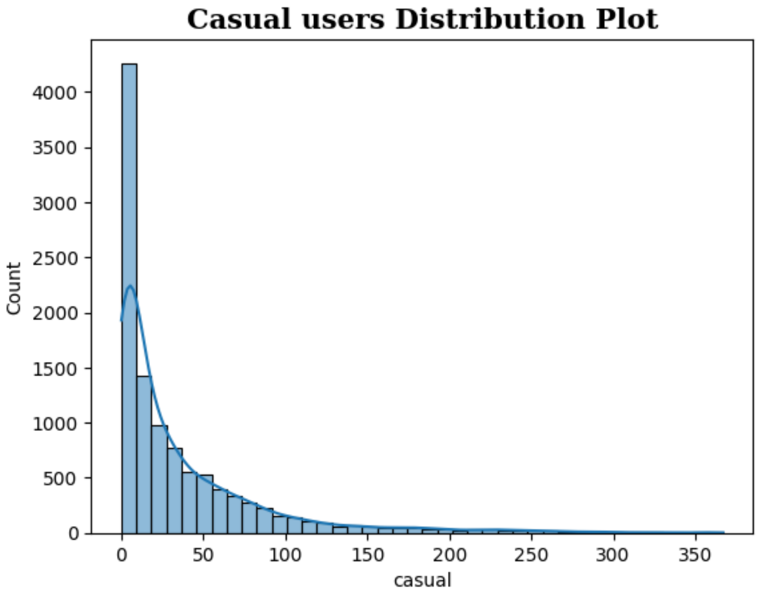
```
mean: 12.7993954069447 standard: 8.164537326838689
```



Casual Users

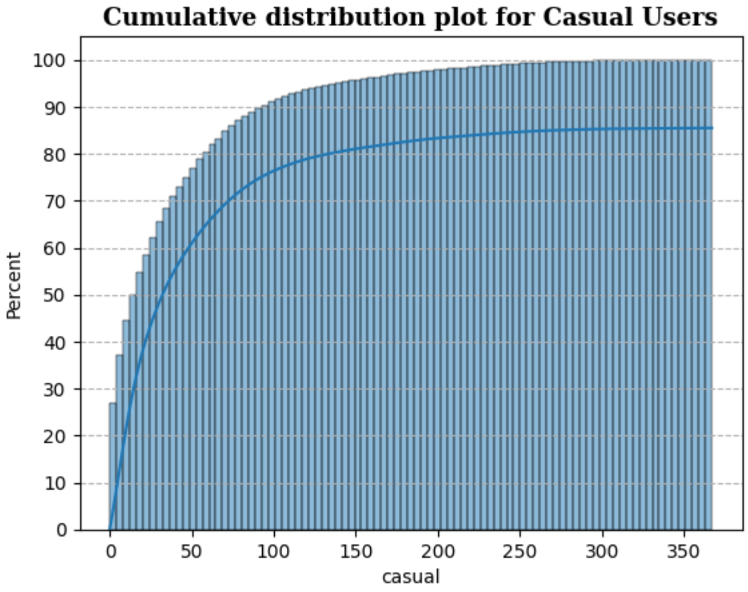
```
# casual users

distribution_plot(df['casual'])
plt.title('Casual users Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 36.02195480433584 standard: 49.960476572649526
```



```
#casual users
cdf_plot(df['casual'])
plt.title('Cumulative distribution plot for Casual Users',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

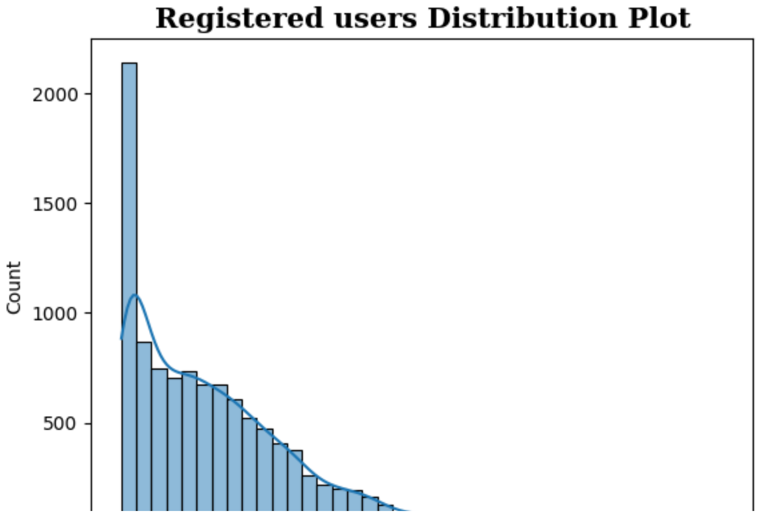
```
mean: 36.02195480433584 standard: 49.960476572649526
```



Registered Users

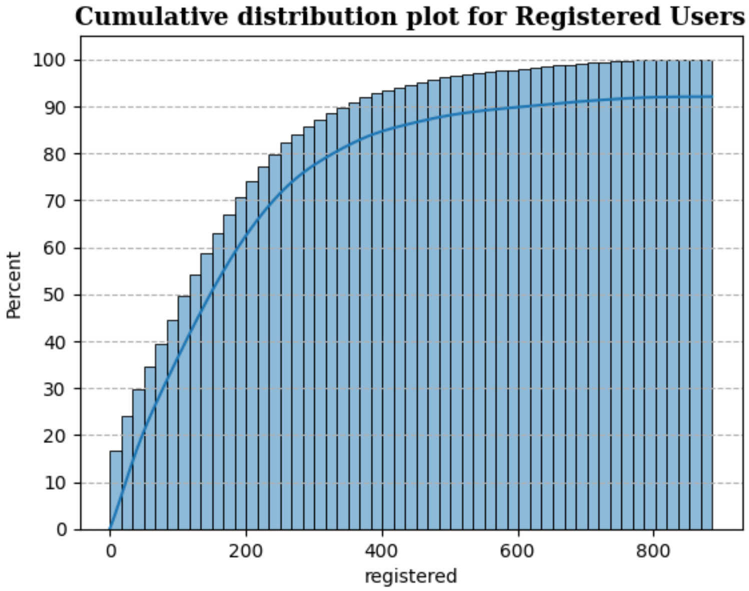
```
# registered users

distribution_plot(df['registered'])
plt.title('Registered users Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 155.5521771082124 standard: 151.03903308192454
```



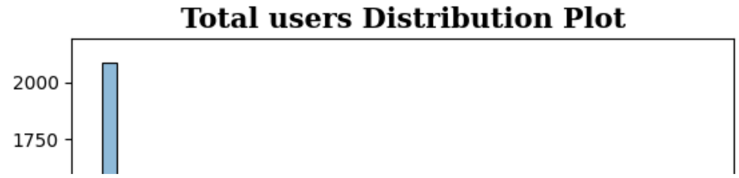
```
#registered users
cdf_plot(df['registered'])
plt.title('Cumulative distribution plot for Registered Users',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

```
mean: 155.5521771082124 standard: 151.03903308192454
```



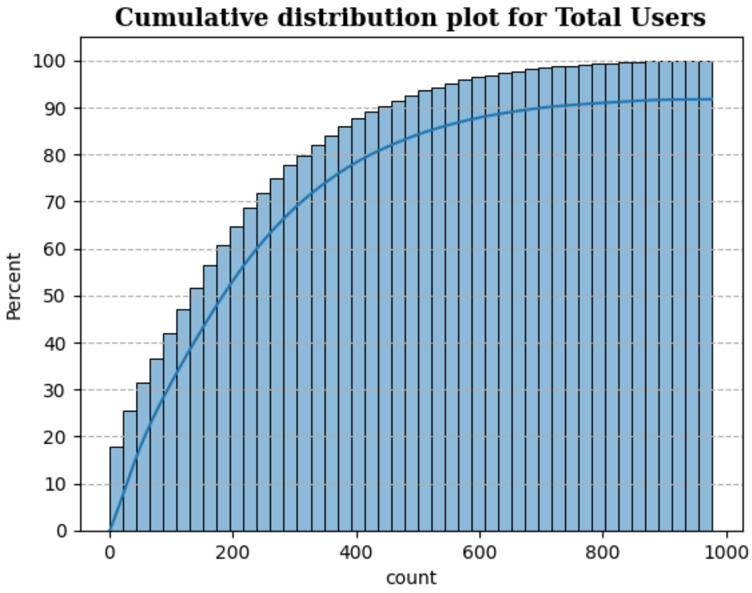
```
# Total users

distribution_plot(df['count'])
plt.title('Total users Distribution Plot',{'font':'serif', 'size':15,'weight':'bold'})
plt.show()
```

```
mean: 191.57413191254824 standard: 181.14445383028527
```



```
#Total users
cdf_plot(df['count'])
plt.title('Cumulative distribution plot for Total Users',{'font':'serif', 'size':13,'weight':'bold'})
plt.show()
```

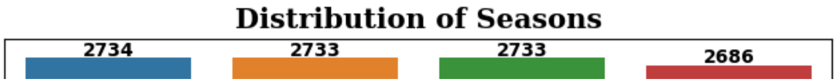
```
mean: 191.57413191254824 standard: 181.14445383028527
```



Categorical Variables

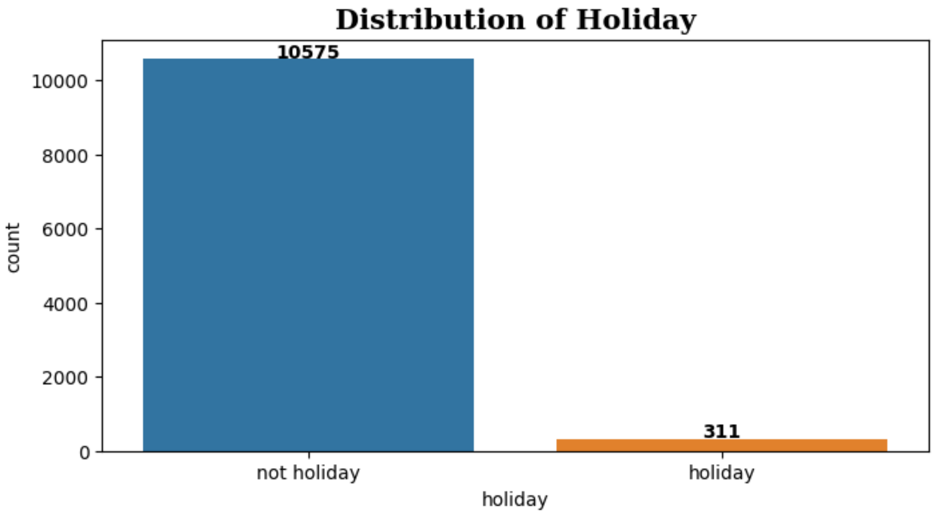
Seasons

```
fig = plt.figure(figsize=(8,4))

ht= df.season.value_counts(ascending=False)

sns.countplot(data=df, x='season', order= ht.index)
plt.title('Distribution of Seasons', {'font':'serif', 'size':15,'weight':'bold'})


for i in range(len(ht.index)):
  plt.text(i,ht[i]+20, ht[i], ha='center', va='baseline',fontsize=10, weight= 'bold')

plt.show()
```



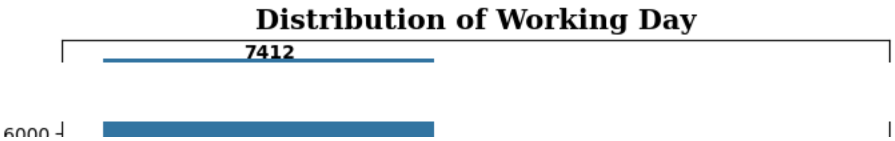
Holiday

```
fig = plt.figure(figsize=(8,4))

ht= df.holiday.value_counts(ascending=False)

sns.countplot(data=df, x='holiday', order= ht.index)
plt.title('Distribution of Holiday', {'font':'serif', 'size':15,'weight':'bold'})


for i in range(len(ht.index)):
  plt.text(i,ht[i]+50, ht[i], ha='center', va='baseline',fontsize=10, weight= 'bold')

plt.show()
```



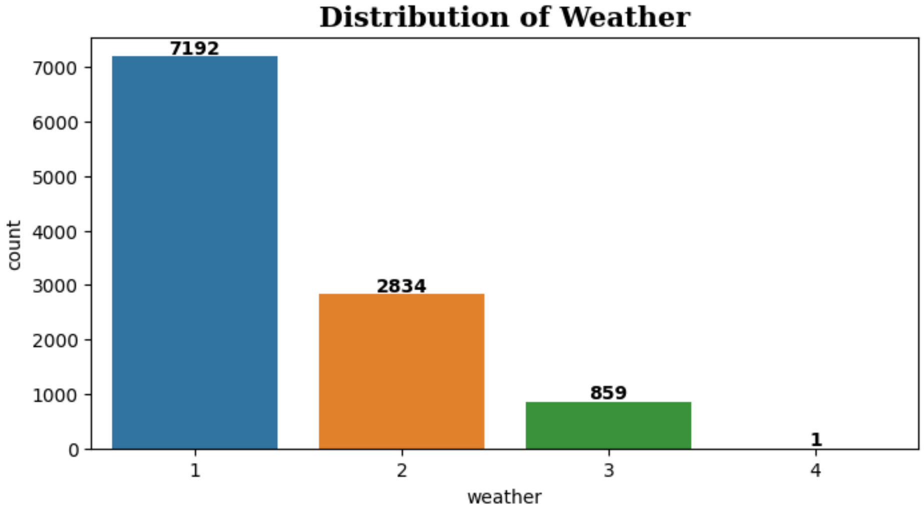
Working Day

```
fig = plt.figure(figsize=(8,4))

ht= df.workingday.value_counts(ascending=False)

sns.countplot(data=df, x='workingday', order= ht.index)
plt.title('Distribution of Working Day', {'font':'serif', 'size':15,'weight':'bold'})


for i in range(len(ht.index)):
  plt.text(i,ht[i]+20, ht[i], ha='center', va='baseline',fontsize=10, weight= 'bold')

plt.show()
```



Weather

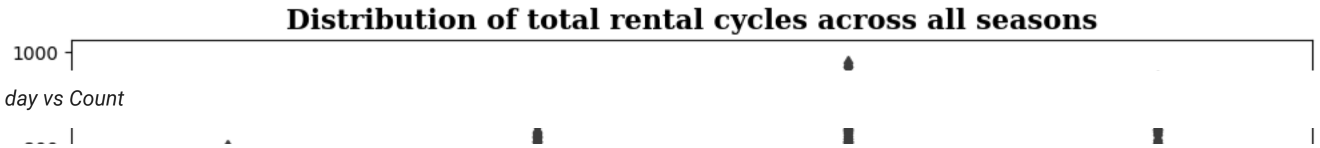
```
fig = plt.figure(figsize=(8,4))

ht= df.weather.value_counts(ascending=False)

sns.countplot(data=df, x='weather', order= ht.index)
plt.title('Distribution of Weather', {'font':'serif', 'size':15,'weight':'bold'})


for i in ht.index:
  plt.text(i-1,ht[i]+50, ht[i], ha='center', va='baseline',fontsize=10, weight= 'bold')

plt.show()
```



Bivariate Analysis for Important variables

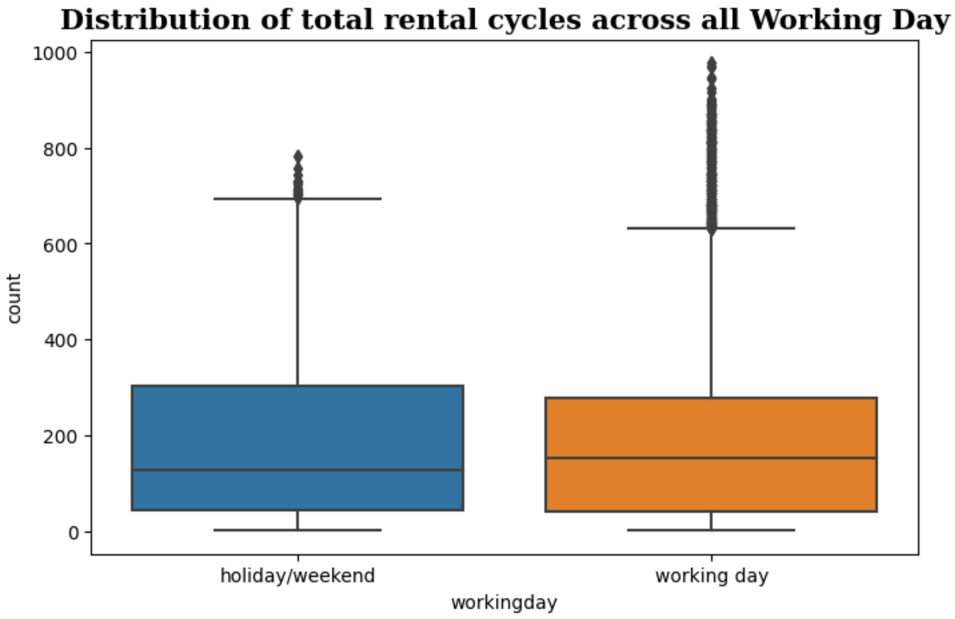
Season vs Count

```
fig= plt.figure(figsize=(12,5))
sns.boxplot(x='season', y='count', data=df)

plt.title('Distribution of total rental cycles across all seasons',{'font':'serif','size': 15, 'weight': 'bold'})

plt.show()
```



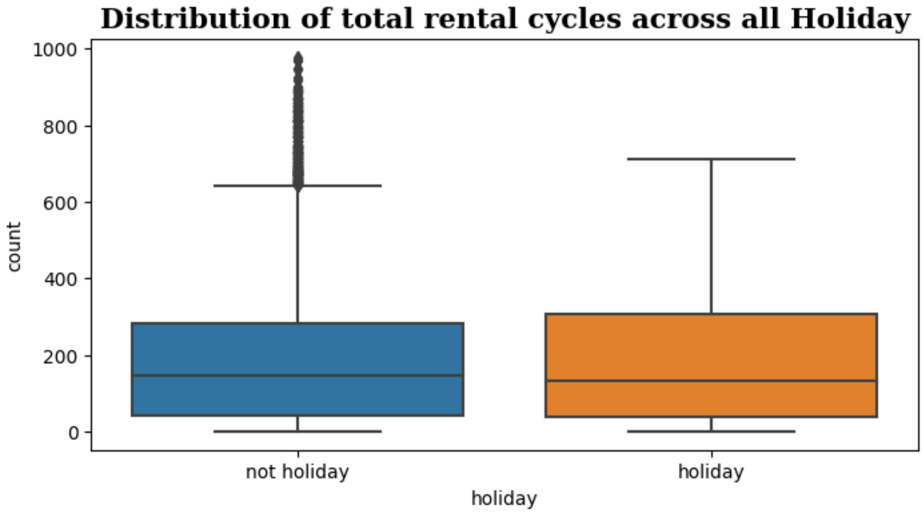
Working day vs Count

```
fig= plt.figure(figsize=(8,5))
sns.boxplot(x='workingday', y='count', data=df)

plt.title('Distribution of total rental cycles across all Working Day',{'font':'serif','size': 15, 'weight': 'bold'})

plt.show()
```



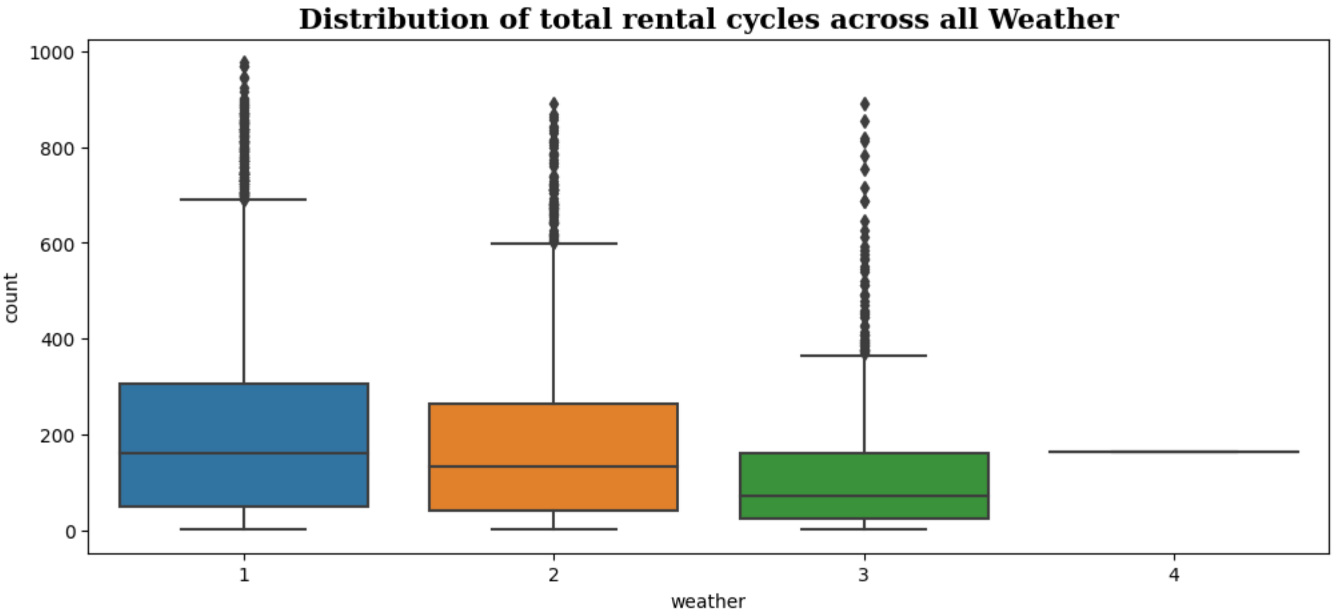
Holiday vs Count

```
fig= plt.figure(figsize=(8,4))
sns.boxplot(x='holiday', y='count', data=df)

plt.title('Distribution of total rental cycles across all Holiday',{'font':'serif','size': 15, 'weight': 'bold'})

plt.show()
```



Weather vs. Count

```
fig= plt.figure(figsize=(12,5))
sns.boxplot(x='weather', y='count', data=df)

plt.title('Distribution of total rental cycles across all Weather',{'font':'serif','size': 15, 'weight': 'bold'})

plt.show()
```



Multivariate Analysis

Heat map

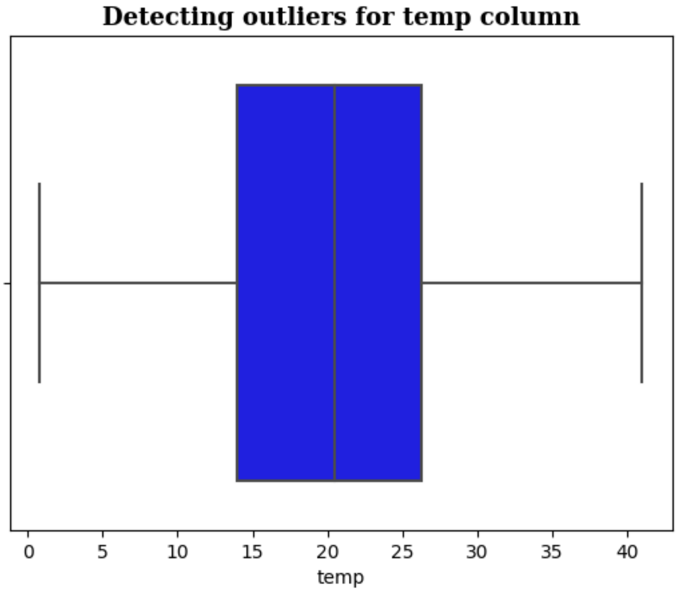
```
plt.figure(figsize = (10, 5))

corr_data = df.corr()
sns.heatmap(data = corr_data, cmap = 'Greens', annot = True, vmin = -1, vmax = 1)

plt.title('Correlation between diffirent numerial variable using heat map', font='serif', size=15, weight='bold')
plt.xticks(rotation=45)

plt.show()
```

```
<ipython-input-243-c484e71e2085>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versic
  corr_data = df.corr()
```

Outliers Detection

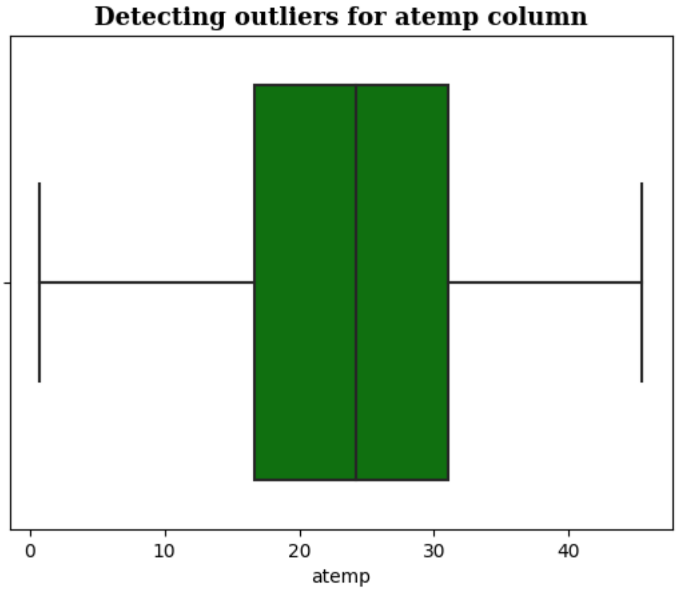
Temperature

```
# temperature

sns.boxplot(x= df['temp'], color='b')
plt.title('Detecting outliers for temp column', font='serif', size=13, weight= 'bold')
plt.show()
```



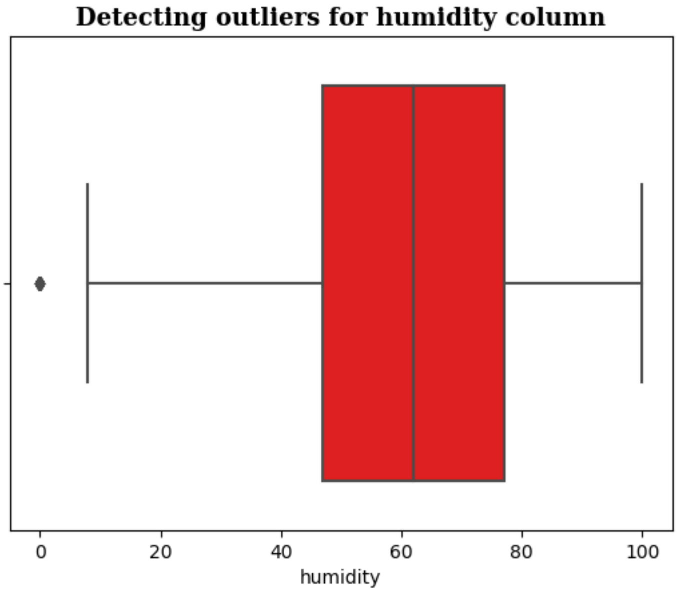
Feeling Temperature

```
# feeling temperature

sns.boxplot(x= df['atemp'], color='g')
plt.title('Detecting outliers for atemp column', font='serif', size=13, weight= 'bold')
plt.show()
```



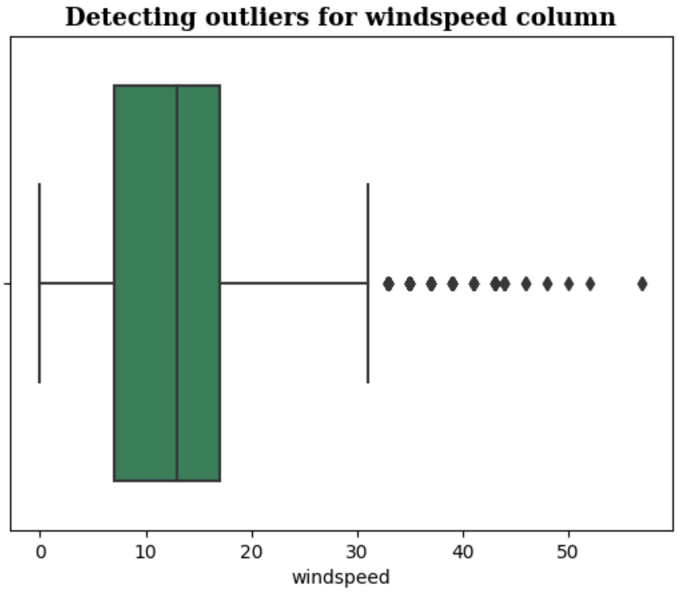
Humidity

```
# humidity

sns.boxplot(x= df['humidity'], color='r')
plt.title('Detecting outliers for humidity column', font='serif', size=13, weight= 'bold')
plt.show()
```



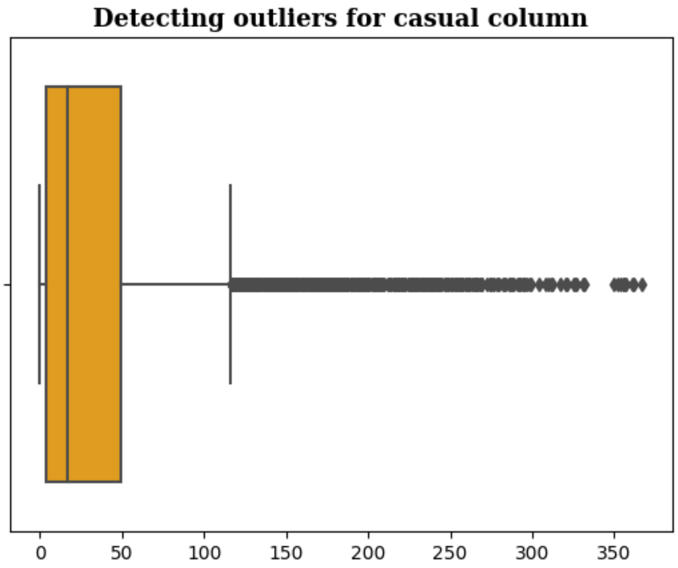
Wind Speed

```
# windspeed

sns.boxplot(x= df['windspeed'], color='seagreen')
plt.title('Detecting outliers for windspeed column', font='serif', size=13, weight= 'bold')
plt.show()
```



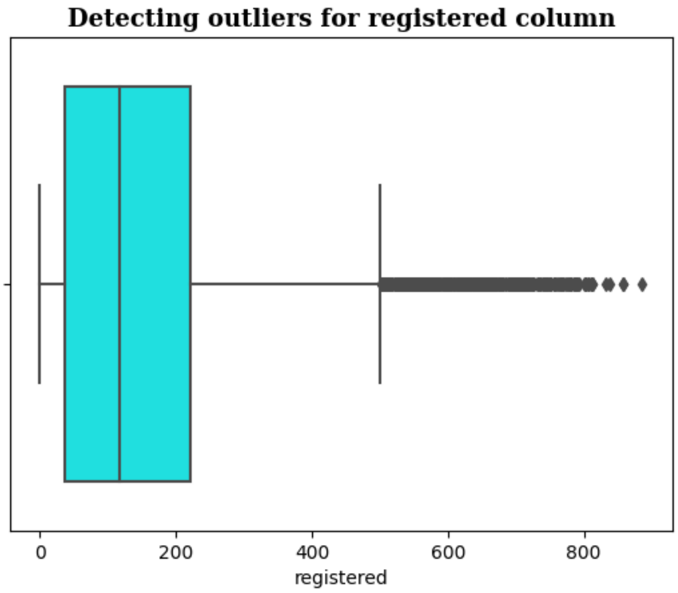
Casual Users

```
# casual users

sns.boxplot(x= df['casual'], color='orange')
plt.title('Detecting outliers for casual column', font='serif', size=13, weight= 'bold')
plt.show()
```



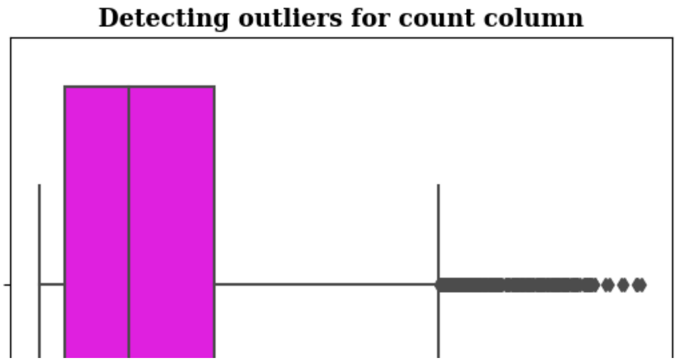
Registered Users

```
# registered

sns.boxplot(x= df['registered'], color='cyan')
plt.title('Detecting outliers for registered column', font='serif', size=13, weight= 'bold')
plt.show()
```



Total Users

```
# count

sns.boxplot(x= df['count'], color='magenta')
plt.title('Detecting outliers for count column', font='serif', size=13, weight= 'bold')
plt.show()
```



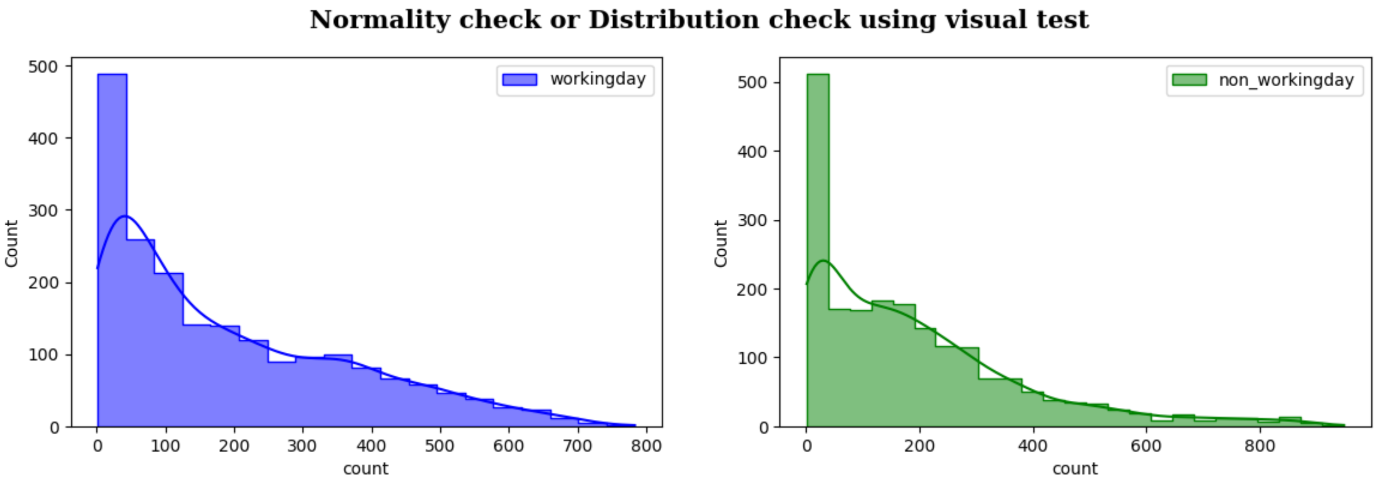
Hypothesis Testing

Working Day vs. count

Question: Is there any effect of Working Day on the number of electric cycles rented?

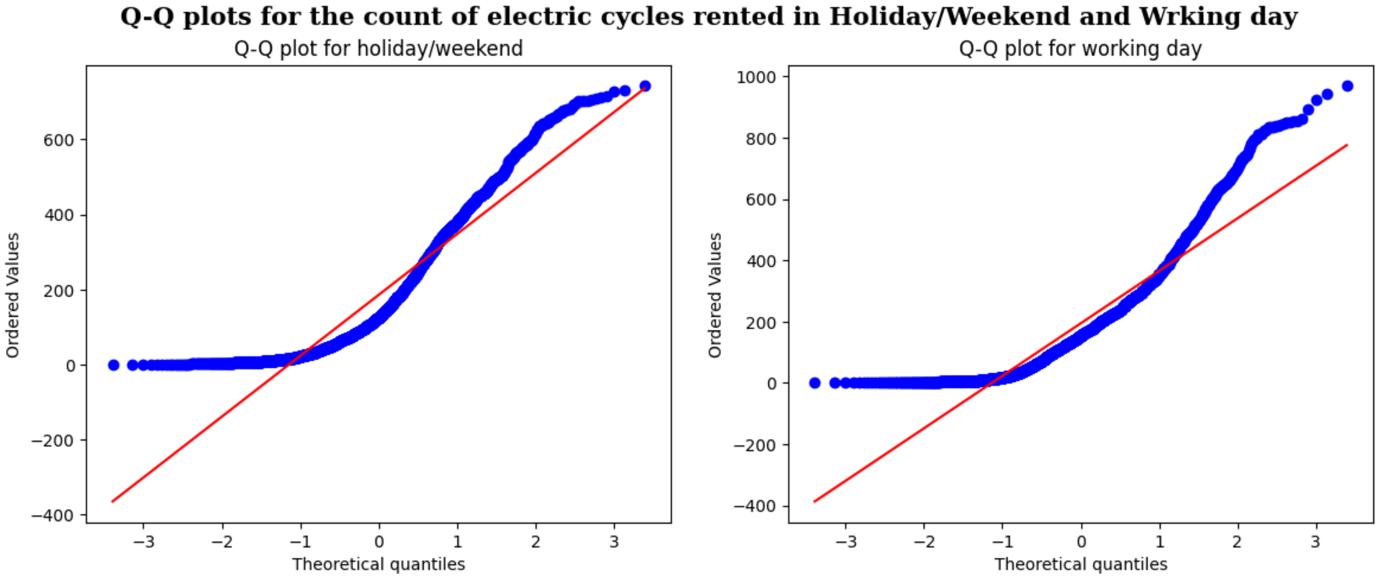
Normality Check or Distribution Check using Histogram or visual

```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (14, 4))

plt.subplot(1, 2, 1)
sns.histplot(df.loc[df['workingday'] == 'holiday/weekend', 'count'].sample(2000),
             element = 'step', color = 'blue', kde = True, label = 'workingday')
plt.legend()

plt.subplot(1, 2, 2)
sns.histplot(df.loc[df['workingday'] == 'working day', 'count'].sample(2000),
             element = 'step', color = 'green', kde = True, label = 'non_workingday')
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=15, weight='bold')
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 5))
plt.suptitle('Q-Q plots for the count of electric cycles rented in Holiday/Weekend and Wrking day', font='serif', size=15, weight='bold')
plt.subplot(1, 2, 1)
probplot(df.loc[df['workingday'] == 'holiday/weekend', 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('Q-Q plot for holiday/weekend')

plt.subplot(1, 2, 2)
probplot(df.loc[df['workingday'] == 'working day', 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('Q-Q plot for working day')
plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for holiday/weekend)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['workingday'] == 'holiday/weekend', 'count'].sample(2000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 1.0499888587374573e-36
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for working day)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['workingday'] == 'working day', 'count'].sample(2000))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 4.1607656683200763e-38
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance
```

```
workingday_sample =df.loc[df['workingday'] == 'working day', 'count'].sample(2000)
non_workingday_sample = df.loc[df['workingday'] == 'holiday/weekend', 'count'].sample(2000)

alpha = 0.05
test_stat, p_value = levene(workingday_sample, non_workingday_sample)
print('p-value', p_value)
if p_value < alpha:
    print('reject Ho: The samples do not have  Homogenous Variance')
else:
    print('Fail to Reject Ho: The samples have Homogenous Variance ')
```

```
p-value 0.8511943598404806
Fail to Reject Ho: The samples have Homogenous Variance
```

Calculate Statistics by ks-test

```
#ks-test
working_day= df.loc[df['workingday'] == 'holiday/weekend']['count']
non_working_day = df.loc[df['workingday'] == 'working day']['count']

ks_stat,p_value = kstest(working_day, non_working_day)

print('ks test statistic result is:', ks_stat)
print('P value is:', p_value)
```

```
ks test statistic result is: 0.05570196737090361
P value is: 8.003959300341833e-07
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): Working Day does not have any effect on the number of electric cycles rented.
# Alternative Hypothesis (Ha): Working Day has effect on the number of electric cycles rented.

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: Working Day has effect on the number of electric cycles rented.')
else:
  print('Accept Ho: Working Day does not have any effect on the number of electric cycles rented.')
```

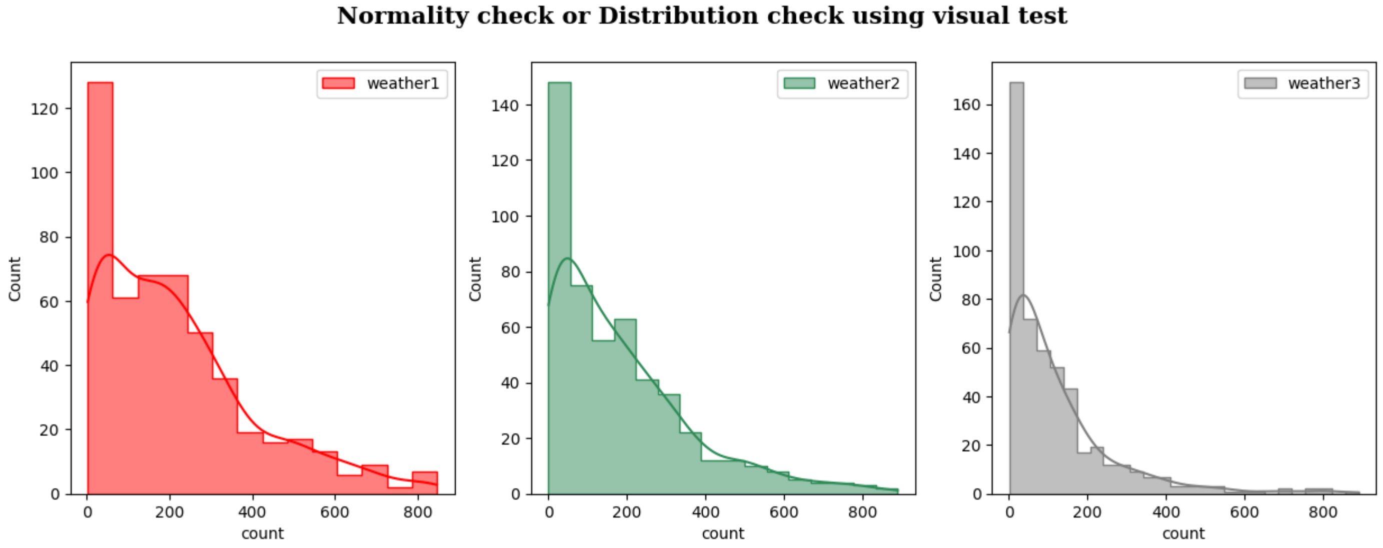
```
Reject Ho: Working Day has effect on the number of electric cycles rented.
```

Weather vs. Count

Question: Is the number of cycles rented is similar or different in different weather?

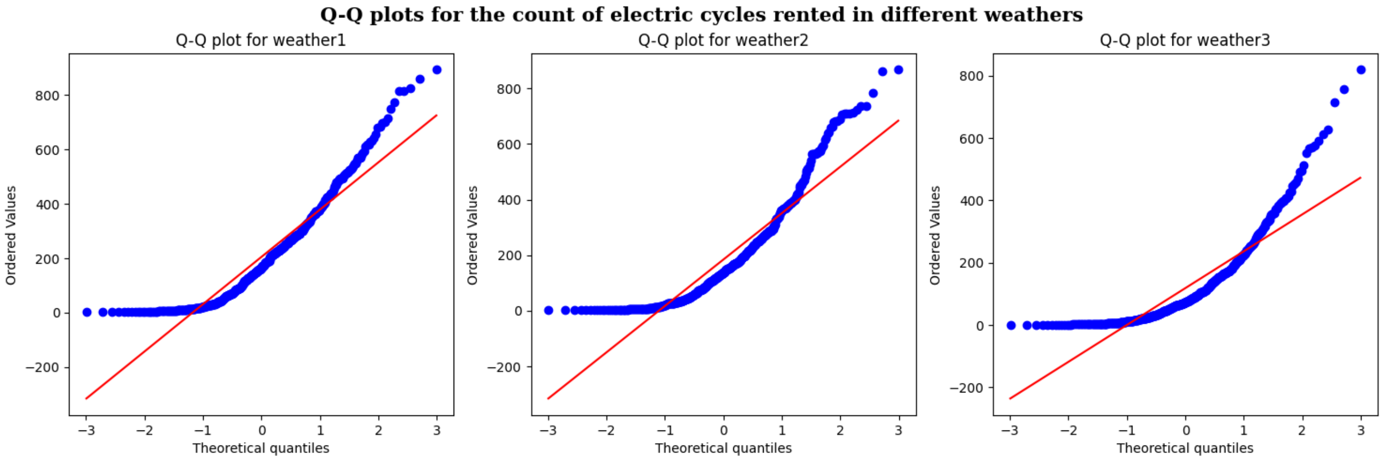
Normality Check or Distribution Check using Histogram or visual

```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (15, 5))

plt.subplot(1, 3, 1)
sns.histplot(df.loc[df['weather'] == 1, 'count'].sample(500), element = 'step',
                    color = 'red', kde = True, label = 'weather1')
plt.legend()

plt.subplot(1, 3, 2)
sns.histplot(df.loc[df['weather'] == 2, 'count'].sample(500),
             element = 'step', color = 'seagreen', kde = True, label = 'weather2')
plt.legend()

plt.subplot(1, 3, 3)
sns.histplot(df.loc[df['weather'] == 3, 'count'].sample(500),
             element = 'step', color = 'grey', kde = True, label = 'weather3')
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=15, weight='bold')
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (18, 5))
plt.suptitle('Q-Q plots for the count of electric cycles rented in different weathers', font='serif', size=15, weight='bold')

plt.subplot(1, 3, 1)
probplot(df.loc[df['weather'] == 1, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for weather1')

plt.subplot(1, 3, 2)
probplot(df.loc[df['weather'] == 2, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for weather2')

plt.subplot(1, 3, 3)
probplot(df.loc[df['weather'] == 3, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for weather3')

plt.show()
```



Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for weather1)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.
```

```
alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['weather'] == 1, 'count'].sample(500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 7.992033790030142e-18
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for weather 2)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['weather'] == 2, 'count'].sample(500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 2.1635124763765527e-19
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for weather 3)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['weather'] == 3, 'count'].sample(500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 3.18837560544863e-25
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance
weather1_sample = df.loc[df['weather'] == 1, 'count'].sample(500)
weather2_sample = df.loc[df['weather'] == 2, 'count'].sample(500)
weather3_sample = df.loc[df['weather'] == 3, 'count'].sample(500)

alpha = 0.05
test_stat, p_value = levene(weather1_sample, weather2_sample, weather3_sample)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho: The samples do not have  Homogenous Variance')
else:
    print('Fail to Reject Ho: The samples have Homogenous Variance ')
```

```
p-value 4.6063916616692275e-14
Reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by kruskal-wallis test

```
# Ho : Mean no. of cycles rented is same for different weather
# Ha : Mean no. of cycles rented is different for different weather

test_stat, p_value = kruskal(df.loc[df['weather'] == 1, 'count'],
                             df.loc[df['weather'] == 2, 'count'],
                             df.loc[df['weather'] == 3, 'count'])
print('kruskal test Statistic result is:', test_stat)
print('P value is:', p_value)
```

```
kruskal test Statistic result is: 204.95566833068537
P value is: 3.122066178659941e-45
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): Mean of cycle rented is same for different weathers.
# Alternative Hypothesis (Ha): Mean of cycle rented is different for different weathers.
# significance level(alpha): 0.05

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: Mean of cycle rented is different for different weathers.')
else:
  print('Accept Ho: Mean of cycle rented is same for different weathers.')
```

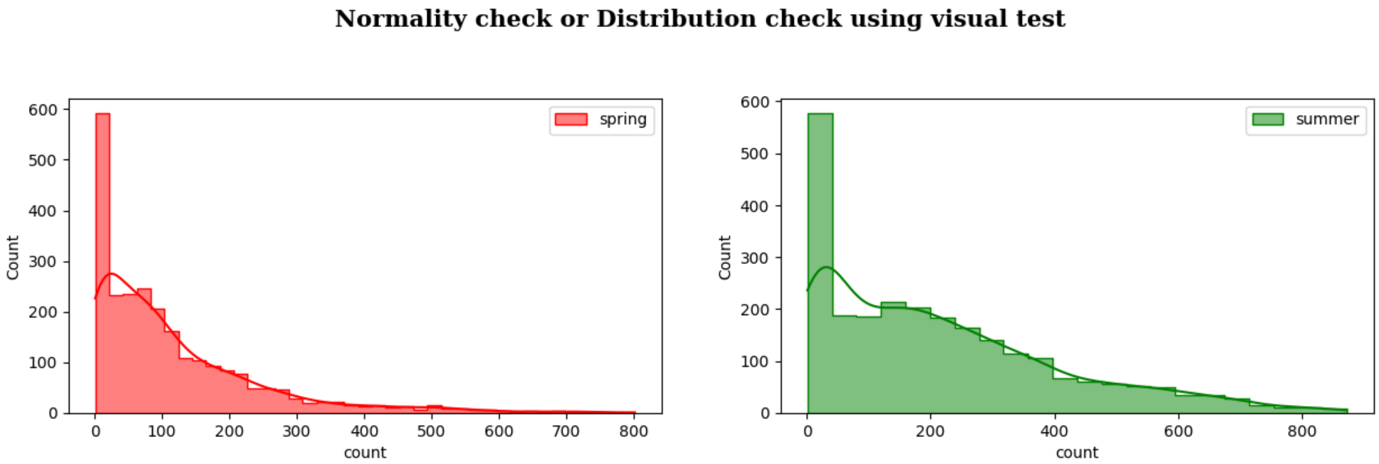
```
Reject Ho: Mean of cycle rented is different for different weathers.
```

Season vs. Count

Question: Is the number of cycles rented is similar or different in different seasons?

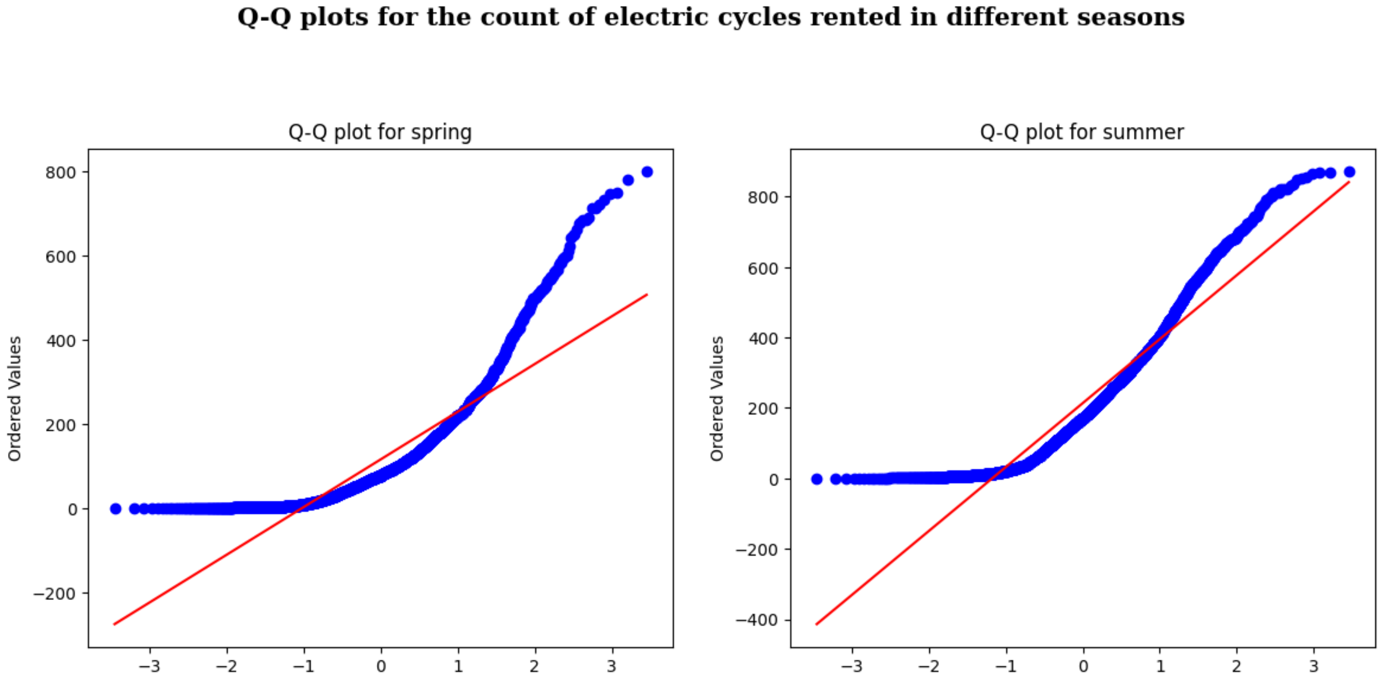
Normality Check or Distribution Check using Histogram or visual

```
# Distribution check or Normality check by Visual Tests

plt.figure(figsize = (15, 8))

plt.subplot(2, 2, 1)
sns.histplot(df.loc[df['season'] == 'spring', 'count'].sample(2500), element = 'step',
                    color = 'red', kde = True, label = 'spring')
plt.legend()

plt.subplot(2, 2, 2)
sns.histplot(df.loc[df['season'] == 'summer', 'count'].sample(2500), element = 'step',
                    color = 'green', kde = True, label = 'summer')
plt.legend()

plt.subplot(2, 2, 3)
sns.histplot(df.loc[df['season'] == 'fall', 'count'].sample(2500),
             element = 'step', color = 'seagreen', kde = True, label = 'fall')
plt.legend()

plt.subplot(2, 2, 4)
sns.histplot(df.loc[df['season'] == 'winter', 'count'].sample(2500),
             element = 'step', color = 'grey', kde = True, label = 'winter')
plt.legend()

plt.suptitle('Normality check or Distribution check using visual test', font='serif', size=15, weight='bold')
plt.show()
```



Normality Check or Distribution Check using Q-Q plot

```
# Distribution check or Normality check by Q-Q plot

plt.figure(figsize = (14, 12))
plt.suptitle('Q-Q plots for the count of electric cycles rented in different seasons', font='serif', size=15, weight='bold')

plt.subplot(2, 2, 1)
probplot(df.loc[df['season'] == 'spring', 'count'].sample(2500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for spring')

plt.subplot(2, 2, 2)
probplot(df.loc[df['season'] == 'summer', 'count'].sample(2500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for summer')

plt.subplot(2, 2, 3)
probplot(df.loc[df['season'] == 'fall', 'count'].sample(2500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for fall')

plt.subplot(2, 2, 4)
probplot(df.loc[df['season'] == 'winter', 'count'].sample(2500), plot = plt, dist = 'norm')
plt.title('Q-Q plot for winter')

plt.show()
```



Q-Q plots for the count of electric cycles rented in different seasons

Normality Check or Distribution Check using Shapiro-Wilk test

```
#Normality Check using Shapiro-Wilk test(for spring)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['season'] == 'spring', 'count'].sample(2500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 0.0
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for summer)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['season'] == 'summer', 'count'].sample(2500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 2.1093447587168263e-37
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for fall)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['season'] == 'fall', 'count'].sample(2500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 4.051901763621503e-35
Reject Ho. The sample does not follow normal distribution
```

```
#Normality Check using Shapiro-Wilk test(for winter)

# Ho: The sample follows normal distribution.
# Ha: The sample does not follow normal distribution.

alpha = 0.05
test_stat, p_value = shapiro(df.loc[df['season'] == 'winter', 'count'].sample(2500))
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho. The sample does not follow normal distribution')
else:
    print('Fail to reject Ho. The sample follows normal distribution')
```

```
p-value 1.7419123623079432e-38
Reject Ho. The sample does not follow normal distribution
```

Variance Check using Levene's test

```
# Ho - Varience is Equal. Homogenous Variance
# Ha - Varience is Not Equal. Non Homogenous Variance
spring_sample = df.loc[df['season'] == 'spring', 'count'].sample(2500)
summer_sample = df.loc[df['season'] == 'summer', 'count'].sample(2500)
fall_sample = df.loc[df['season'] == 'fall', 'count'].sample(2500)
winter_sample = df.loc[df['season'] == 'winter', 'count'].sample(2500)

alpha = 0.05
test_stat, p_value = levene(spring_sample, summer_sample, fall_sample, winter_sample)
print('p-value', p_value)
if p_value < alpha:
    print('Reject Ho: The samples do not have  Homogenous Variance')
else:
    print('Fail to Reject Ho: The samples have Homogenous Variance ')
```

```
p-value 9.245693730759443e-109
Reject Ho: The samples do not have  Homogenous Variance
```

Calculate Statistics by kruskal-wallis test

```
# Ho : Mean no. of cycles rented is same for different seasons
# Ha : Mean no. of cycles rented is different for different seasons

test_stat, p_value = kruskal(df.loc[df['season'] == 'spring', 'count'],
                             df.loc[df['season'] == 'summer', 'count'],
                             df.loc[df['season'] == 'fall', 'count'],
                             df.loc[df['season'] == 'winter', 'count'])
print('kruskal test Statistic result is:', test_stat)
print('P value is:', p_value)
```

```
kruskal test Statistic result is: 699.6668548181988
P value is: 2.479008372608633e-151
```

Decision to accept or reject null hypothesis.

```
# Null Hypothesis (Ho): Mean of cycle rented is same for different seasons.
# Alternative Hypothesis (Ha): Mean of cycle rented is different for different seasons.
# significance level(alpha): 0.05

alpha = 0.05
if p_value < alpha:
  print('Reject Ho: Mean of cycle rented is different for different seasons.')
else:
  print('Accept Ho: Mean of cycle rented is same for different seasons.')
```

```
Reject Ho: Mean of cycle rented is different for different seasons.
```

Weather vs. Season

Question: Is weather dependent on the season?

Create a Contigency table

```
contigency_table = pd.crosstab(index = df['season'],
                          columns = df['weather'],
                          values = df['count'],
                          aggfunc = 'sum')
contigency_table
```

weather	1	2	3	4
season				
spring	223009	76406	12919	164
summer	426350	134177	27755	0
fall	470116	139386	31160	0
winter	356588	157191	30255	0

Chi-square Test

```
chi_test_stat, p_value, dof, expected = chi2_contingency(observed = contigency_table)
print('Test Statistic =', chi_test_stat)
print('P value =', p_value)
```

```
Test Statistic = 11769.559450959445
P value = 0.0
```

Decision to accept or reject null hypothesis

```
# Null Hypothesis (Ho): Weather is not dependent on season.
# Alternative Hypothesis (Ha): Weather is dependent on season.
# Significance level (alpha): 0.05

alpha=0.05
if p_value < alpha:
    print('Reject Ho: Weather is dependent on season.')
else:
    print('Failed to reject Ho: Weather is not dependent on season.')
```

```
Reject Ho: Weather is dependent on season.
```