```
#Downloading the dataset and stroing it.
!gdown https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv
To: /content/bike_sharing.csv
100% 648k/648k [00:00<00:00, 16.2MB/s]
```

```
#To list all the files
!ls
```

```
bike_sharing.csv  sample_data
```

```
#Importing all the required packages
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from scipy.stats import ttest_ind, f_oneway, levene, kruskal, chi2_contingency
```

```
#Reading the dataset and storing in the variable
df = pd.read_csv("bike_sharing.csv")
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

```
#Displaying the information of the Dataframe
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
```

```
#Converting the Datatype from object to datetime
df['datetime'] = pd.to_datetime(df['datetime'])
```

```
#Statistical Analysis of the provided dataset
df.describe()
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

```
#Checking is there any null values
df.isna().any()
```

```
datetime      False
season        False
holiday       False
workingday    False
weather       False
temp          False
atemp         False
humidity      False
windspeed     False
casual        False
registered    False
count         False
dtype: bool
```

There are no NULL values present in any of the column

```
#Checking the number of unique values
df.nunique()
```

```
datetime      10886
season            4
holiday           2
workingday        2
weather           4
temp             49
atemp            60
humidity         89
windspeed        28
casual          309
registered      731
count           822
dtype: int64
```

```
#Checking if there is any duplicate entries
df[df.duplicated()]
```

```
datetime season holiday workingday weather temp atemp humidity windspeed casual registered count
```

There are no duplicate values present in the given dataset

```
#Converting the season from integer to the respective season name
df['season'].replace(1, 'spring',inplace=True)
df['season'].replace(2, 'summer',inplace=True)
df['season'].replace(3, 'fall',inplace=True)
df['season'].replace(4, 'winter',inplace=True)
df['workingday'].replace(1, 'working',inplace=True)
df['workingday'].replace(0, 'off',inplace=True)
```

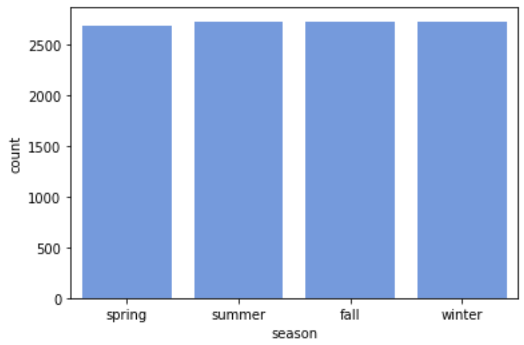
Converting the season and workingday filed from numbers to respective name, so that it will easy for plotting

```
#Counting the number of entires present for each season
df.groupby(['season']).size()
```

```
season
fall      2733
spring    2686
summer    2733
winter    2734
dtype: int64
```

```
sns.countplot(x = 'season', data = df, color='cornflowerblue')
```

```
<AxesSubplot:xlabel='season', ylabel='count'>
```

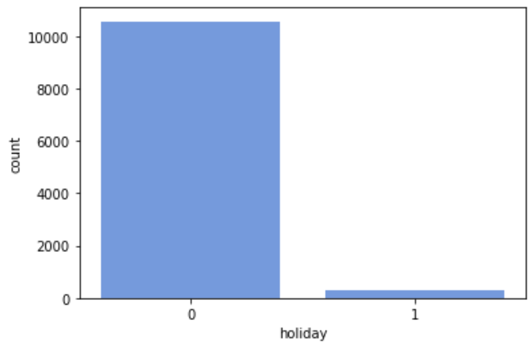


```
#Counting the number of entires present for holiday and NON-holiday
df.groupby(['holiday']).size()
```

```
holiday
0    10575
1      311
dtype: int64
```

```
sns.countplot(x = 'holiday', data = df, color='cornflowerblue')
```

```
<AxesSubplot:xlabel='holiday', ylabel='count'>
```

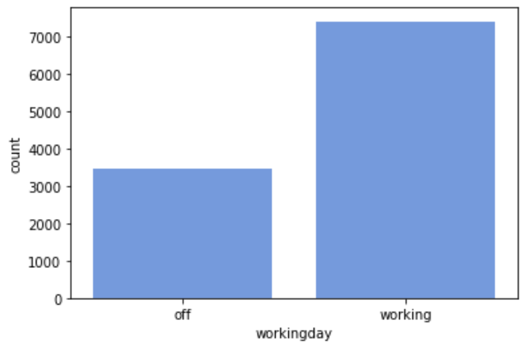


```
#Counting the number of entires present for workinday and non-workinday
df.groupby(['workingday']).size()
```

```
workingday
off        3474
working    7412
dtype: int64
```

```
sns.countplot(x = 'workingday', data = df, color='cornflowerblue')
```

```
<AxesSubplot:xlabel='workingday', ylabel='count'>
```

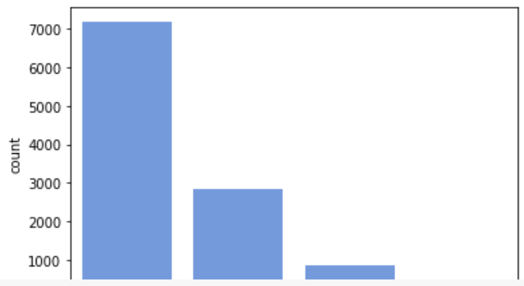


```
#Counting the number of entires present for each weather
df.groupby(['weather']).size()
```

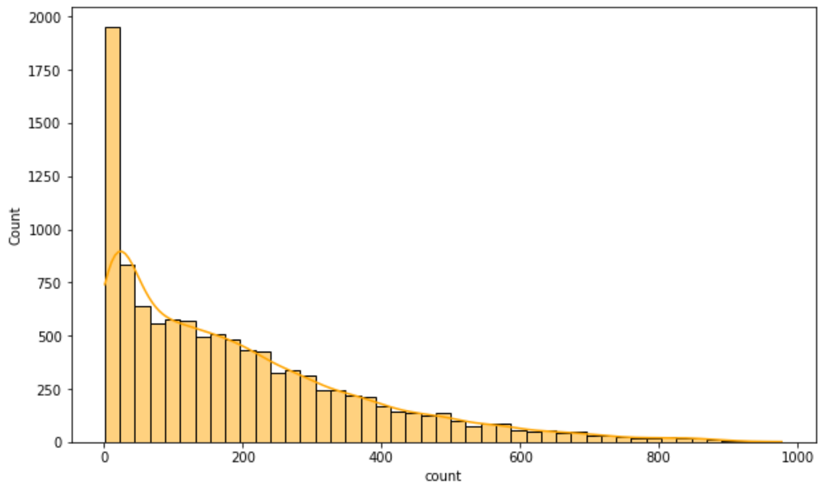
```
weather
1    7192
2    2834
3     859
4       1
dtype: int64
```

```
sns.countplot(x = 'weather', data = df, color='cornflowerblue')
```

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```

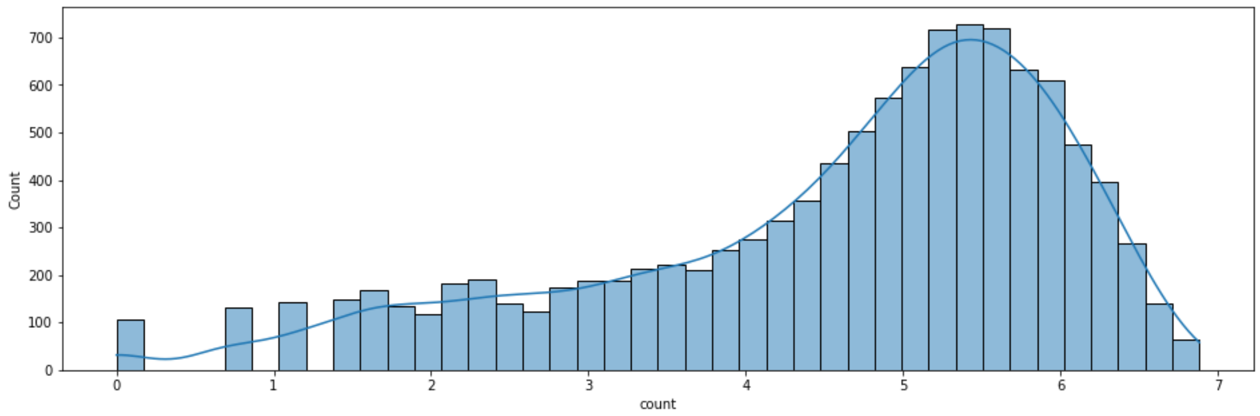


```
plt.figure(figsize=(10, 6))
sns.histplot(data=df,x=df['count'],kde=True,color='orange')
plt.show()
```



From the above plot we can see that the count values are right skewed

```
plt.figure(figsize=(16, 5))
sns.histplot(np.log(df['count']), kde=True)

plt.show()
```



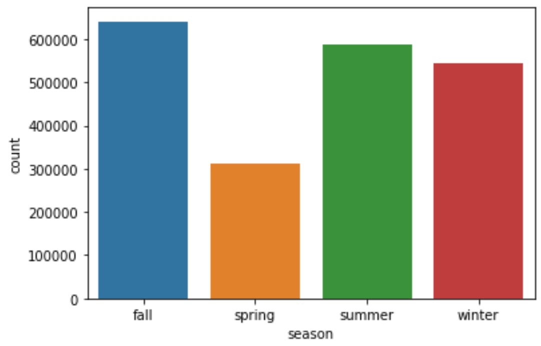
Log normal distribution is applied to the right skewed values

```
#Counting number of cycle counts in each season
df.groupby(['season'])['count'].sum().reset_index()
```

	season	count
0	fall	640662
1	spring	312498
2	summer	588282
3	winter	544034

```
sns.barplot(x='season', y='count', data = df.groupby(['season'])['count'].sum().reset_index())
```

```
<AxesSubplot:xlabel='season', ylabel='count'>
```

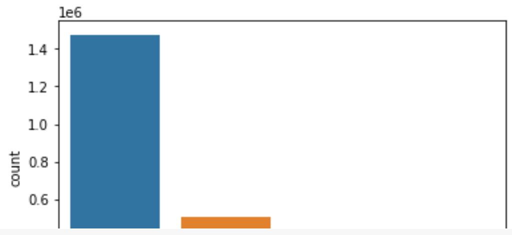


```
#Counting number of cycle counts in each weather
df.groupby(['weather'])['count'].sum().reset_index()
```

	weather	count
0	1	1476063
1	2	507160
2	3	102089
3	4	164

```
sns.barplot(x='weather', y='count', data = df.groupby(['weather'])['count'].sum().reset_index())
```

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```

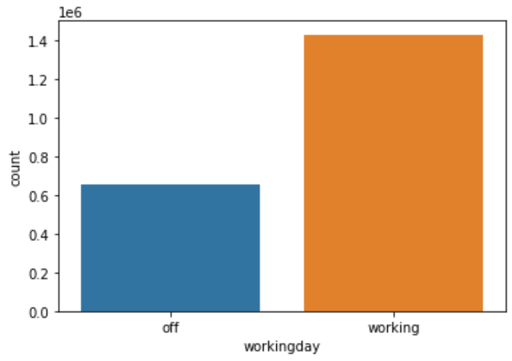


```
#Counting number of cycle counts in workingday
df.groupby(['workingday'])['count'].sum().reset_index()
```

	workingday	count
0	off	654872
1	working	1430604

```
sns.barplot(x='workingday', y='count', data = df.groupby(['workingday'])['count'].sum().reset_index())
```

```
<AxesSubplot:xlabel='workingday', ylabel='count'>
```

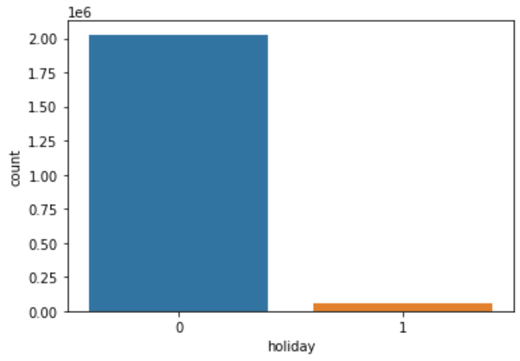


```
#Counting number of cycle counts in holidays
df.groupby(['holiday'])['count'].sum().reset_index()
```

	holiday	count
0	0	2027668
1	1	57808

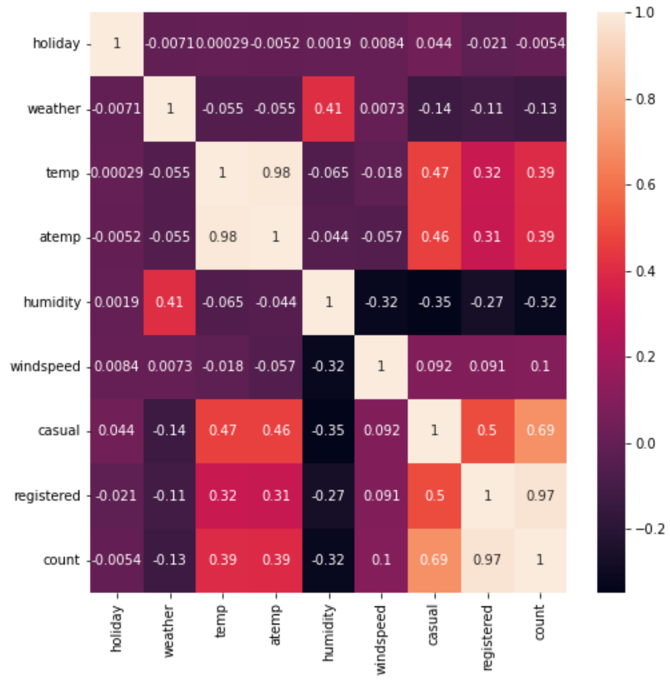
```
sns.barplot(x='holiday', y='count', data = df.groupby(['holiday'])['count'].sum().reset_index())
```

```
<AxesSubplot:xlabel='holiday', ylabel='count'>
```



```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```



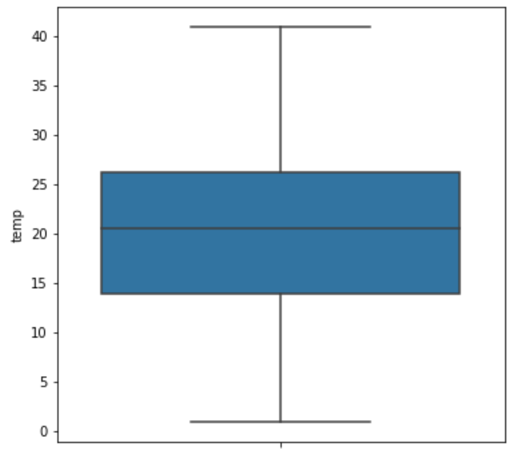
```
#Average temprature, actual temprature and windspeed in each season and in each weather condition
df.groupby(['season','weather'])['temp','atemp','windspeed'].mean().reset_index()
```

```
<ipython-input-135-ae9210b288a5>:2: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a lis
  df.groupby(['season','weather'])['temp','atemp','windspeed'].mean().reset_index()
```

	season	weather	temp	atemp	windspeed
0	fall	1	29.227264	33.044816	11.241711
1	fall	2	28.048344	31.772434	11.409224
2	fall	3	26.788040	29.984497	14.402239
3	spring	1	12.539147	15.135455	15.735428

```
plt.figure(figsize=(6,6))
sns.boxplot(y="temp",data=df)
```

```
<AxesSubplot:ylabel='temp'>
```

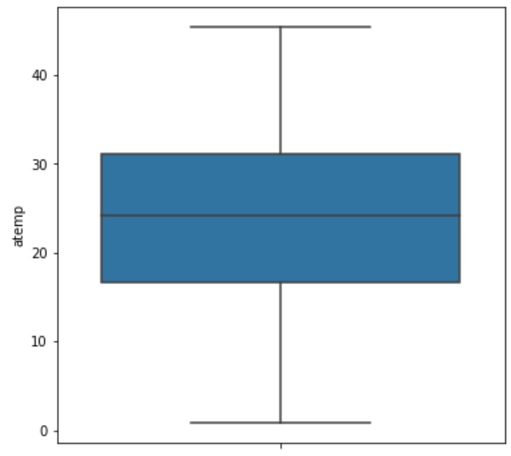


```
p_25 = np.percentile(df["temp"], 25)
p_50 = np.percentile(df["temp"], 50)
p_75 = np.percentile(df["temp"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["temp"] > right_whis])
print("Recorded Temprature's outliers: ", num_outliers)

print("Recorded Temprature's std: ", df["temp"].std())
```

```
First Quartile:  13.94
Second Quartile:  20.5
Third Quartile:  26.24
IQR:  12.299999999999999
Left:  0
Right:  44.69
Recorded Temprature's outliers:  0
Recorded Temprature's std:  7.791589843987567
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="atemp",data=df)
```

```
<AxesSubplot:ylabel='atemp'>
```

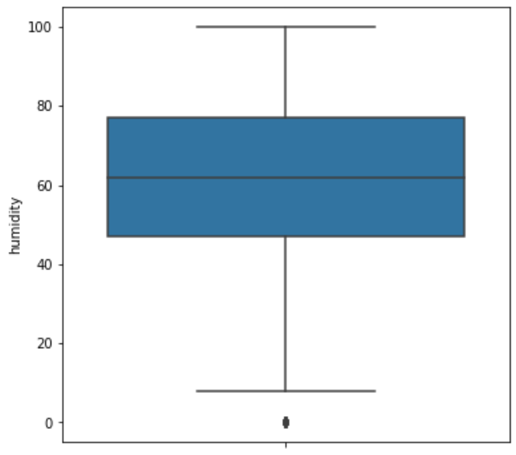


```
p_25 = np.percentile(df["atemp"], 25)
p_50 = np.percentile(df["atemp"], 50)
p_75 = np.percentile(df["atemp"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["atemp"] > right_whis])
print("Actual Temprature's outliers: ", num_outliers)

print("Actual Temprature's std: ", df["atemp"].std())
```

```
First Quartile:  16.665
Second Quartile:  24.24
Third Quartile:  31.06
IQR:  14.395
Left:  0
Right:  52.6525
Actual Temprature's outliers:  0
Actual Temprature's std:  8.474600626484948
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="humidity",data=df)
```

```
<AxesSubplot:ylabel='humidity'>
```

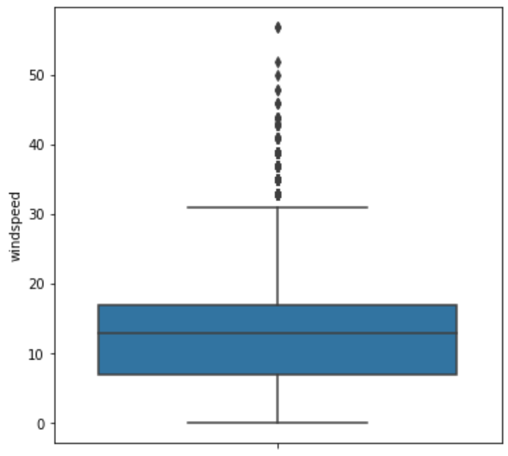


```
p_25 = np.percentile(df["humidity"], 25)
p_50 = np.percentile(df["humidity"], 50)
p_75 = np.percentile(df["humidity"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["humidity"] > right_whis])
print("humidity's outliers: ", num_outliers)

print("humidity's std: ", df["humidity"].std())
```

```
First Quartile:  47.0
Second Quartile:  62.0
Third Quartile:  77.0
IQR:  30.0
Left:  2.0
Right:  122.0
humidity's outliers:  0
humidity's std:  19.24503327739469
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="windspeed",data=df)
```

```
<AxesSubplot:ylabel='windspeed'>
```

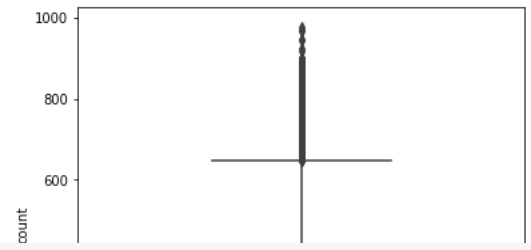


```
p_25 = np.percentile(df["windspeed"], 25)
p_50 = np.percentile(df["windspeed"], 50)
p_75 = np.percentile(df["windspeed"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["windspeed"] > right_whis])
print("windspeed's outliers: ", num_outliers)

print("windspeed's std: ", df["windspeed"].std())
```

```
First Quartile:  7.0015
Second Quartile:  12.998
Third Quartile:  16.9979
IQR:  9.996400000000001
Left:  0
Right:  31.992500000000003
windspeed's outliers:  227
windspeed's std:  8.164537326838689
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="count",data=df)
```

```
<AxesSubplot:ylabel='count'>
```

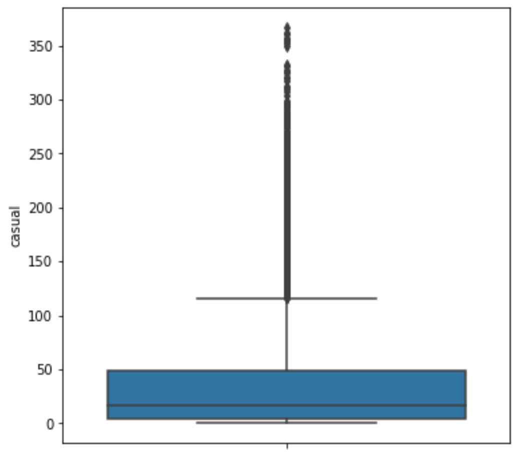


```
p_25 = np.percentile(df["count"], 25)
p_50 = np.percentile(df["count"], 50)
p_75 = np.percentile(df["count"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["count"] > right_whis])
print("cycle count's outliers: ", num_outliers)

print("cycle count's std: ", df["count"].std())
```

```
First Quartile:  42.0
Second Quartile:  145.0
Third Quartile:  284.0
IQR:  242.0
Left:  0
Right:  647.0
cycle count's outliers:  300
cycle count's std:  181.14445383028527
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="casual",data=df)
```

```
<AxesSubplot:ylabel='casual'>
```

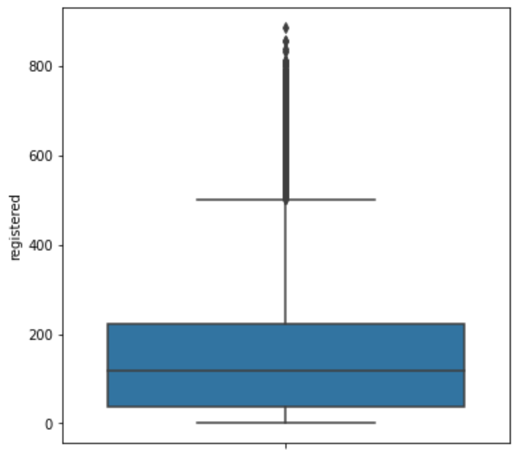


```
p_25 = np.percentile(df["casual"], 25)
p_50 = np.percentile(df["casual"], 50)
p_75 = np.percentile(df["casual"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["casual"] > right_whis])
print("casual customer's outliers: ", num_outliers)

print("casual customer's std: ", df["casual"].std())
```

```
First Quartile:  4.0
Second Quartile:  17.0
Third Quartile:  49.0
IQR:  45.0
Left:  0
Right:  116.5
casual customer's outliers:  749
casual customer's std:  49.960476572649526
```

```
plt.figure(figsize=(6,6))
sns.boxplot(y="registered",data=df)
```

```
<AxesSubplot:ylabel='registered'>
```



```
p_25 = np.percentile(df["registered"], 25)
p_50 = np.percentile(df["registered"], 50)
p_75 = np.percentile(df["registered"], 75)

print("First Quartile: ", p_25) # p = 25%
print("Second Quartile: ", p_50)# p = 50%
print("Third Quartile: ", p_75) # p = 75%

print("IQR: ", p_75 - p_25)

left_whis  = max(p_25 - 1.5 * (p_75 - p_25), 0)
right_whis = p_75 + 1.5 * (p_75 - p_25)

print("Left: ", left_whis)
print("Right: ", right_whis)

num_outliers = len(df[df["registered"] > right_whis])
print("registered customer's outliers: ", num_outliers)

print("registered customer's std: ", df["registered"].std())
```

```
    First Quartile:  36.0
    Second Quartile:  118.0
    Third Quartile:  222.0
    IQR:  186.0
    Left:  0
    Right:  501.0
    registered customer's outliers:  423
    registered customer's std:  151.03903308192454
```

2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented:

```
workingday_count = df[df['workingday'] == "working"]['count']
non_workingday_count = df[df['workingday'] == "off"]['count']
```

```
workingday_count = np.array(workingday_count)
non_workingday_count = np.array(non_workingday_count)
```

```
t_stat, p_value = ttest_ind(workingday_count, non_workingday_count, alternative="greater")
```

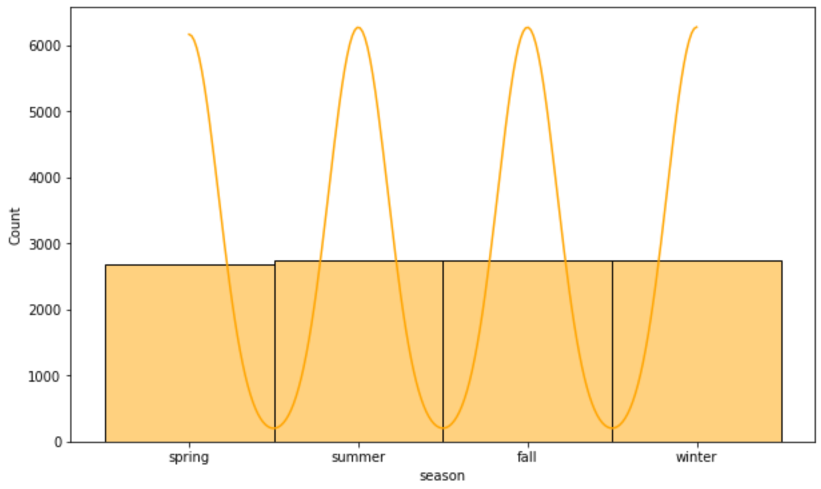
```
alpha = 0.01
H0 = "Working Day has no effect with the number of cycles rented"
Ha = "Working Day has effect with the number of cycles rented"
if(p_value < alpha):
  print("Working day has effect on the number of cycles rented")
  print("H0 --> \""+H0+"\" is Rejected ")
else:
  print("Working Day has no effect with the number of cycles rented")
  print("H0 --> \""+H0+"\" is Not Rejected ")
```

```
    Working Day has no effect with the number of cycles rented
    H0 --> "Working Day has no effect with the number of cycles rented" is Not Rejected
```

As pvalue is less than 0.01, Working day has effect on the number of cycles rented

ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season

```
plt.figure(figsize=(10, 6))
sns.histplot(data=df,x=df['season'],kde=True,color='orange')
plt.show()
```



As per the above graph the season are following the gaussian distribution, hence we can have the ANOVA test to check the weather the No. of cycles rented similar or different in different seasons

```
df["weather"].unique()
```

```
    array([1, 2, 3, 4])
```

```
weather_1 = df[df['weather'] == 1]['count']
weather_2 = df[df['weather'] == 2]['count']
weather_3 = df[df['weather'] == 3]['count']
weather_4 = df[df['weather'] == 4]['count']
```

```
stat, p_value = levene(weather_1, weather_2, weather_3, weather_4)
alpha = 0.05
H0 = "Weather has no effect with the number of cycles rented"
Ha = "Weather has effect with the number of cycles rented"
if(p_value < alpha):
  print("Anova test can't be performed, try kruskal")
else:
  print("Anova test can be performed")
```

```
    Anova test can't be performed, try kruskal
```

Since the leven test pvalue < 0.05 we will not be able to perform anova test

```
stat, p_value = kruskal(weather_1, weather_2, weather_3,weather_4)
alpha = 0.05
H0 = "Weather has no effect with the number of cycles rented"
Ha = "Weather has effect with the number of cycles rented"
if(p_value < alpha):
  print("Anova test can't be performed, and we reject the H0")
  print("H0 --> \""+H0+"\" is Rejected ")
else:
  print("Anova test can be performed")
```

```
Anova test can't be performed, and we reject the H0
H0 --> "Weather has no effect with the number of cycles rented" is Rejected
```

Since the pvalue for kruskal is < 0.05 hence we can reject the null hyporthesis and weather has effect with the number of cycles rented

```
df["season"].unique()
```

```
array(['spring', 'summer', 'fall', 'winter'], dtype=object)
```

```
fall = df[df['season'] == "fall"]['count']
spring = df[df['season']=='spring']['count']
summer = df[df['season']=='summer']['count']
winter = df[df['season']=='winter']['count']
```

```
stat, p_value = levene(summer, winter, spring, fall)
alpha = 0.05
H0 = "Season has no effect with the number of cycles rented"
Ha = "Season has effect with the number of cycles rented"
if(p_value < alpha):
  print("Anova test can't be performed, try kruskal")
else:
  print("Anova test can be performed")
```

```
Anova test can't be performed, try kruskal
```

Since the leven test pvalue < 0.05 we will not be able to perform anova test

```
stat, p_value = kruskal(summer, winter, spring, fall)
alpha = 0.05
H0 = "Season has no effect with the number of cycles rented"
Ha = "Season has effect with the number of cycles rented"
if(p_value < alpha):
  print("Anova test can't be performed, and we reject the H0")
  print("H0 --> \""+H0+"\" is Rejected ")
else:
  print("Anova test can be performed")
```

```
Anova test can't be performed, and we reject the H0
H0 --> "Season has no effect with the number of cycles rented" is Rejected
```

Since the pvalue for kruskal is < 0.05 hence we can reject the null hyporthesis and Season has effect with the number of cycles rented

Chi-square test to check if Weather is dependent on the season

```
pd.crosstab(df["season"],df["weather"], margins=True)
```

weather	1	2	3	4	All
season					
fall	1930	604	199	0	2733
spring	1759	715	211	1	2686
summer	1801	708	224	0	2733
winter	1702	807	225	0	2734
All	7192	2834	859	1	10886

```
#Since there is only one value for weather 4 we can remove that column
weather = df[df['weather']!= 4]
```

```
pd.crosstab(df["season"],weather["weather"], margins=True)
```

weather	1	2	3	All
season				
fall	1930	604	199	2733
spring	1759	715	211	2685
summer	1801	708	224	2733
winter	1702	807	225	2734
All	7192	2834	859	10885

```
val = pd.crosstab(index=df["season"], columns=weather["weather"]).values
print(val)
chi2_contingency(val)
# alpha = 0.01
# H0 = "Weather/ season are indepenbdent"
# Ha = "Weather/ season are depenbdent"
```

```
[[1930  604  199]
 [1759  715  211]
 [1801  708  224]
 [1702  807  225]]
Chi2ContingencyResult(statistic=46.10145731073249, pvalue=2.8260014509929343e-08, dof=6, expected_freq=array([[1805.76352779,  711.55920992,  215.67726229],
       [1774.04869086,  699.06201194,  211.8892972 ],
       [1805.76352779,  711.55920992,  215.67726229],
       [1806.42425356,  711.81956821,  215.75617823]]))
```

The test statiscs comes out to be 46.10 and pvalue = 2.8260014509929343e-08 hence we can reject the null hypothesis and can conclude that season ans weather are dependent on each other

✓ 0s completed at 11:24 PM ● ×