Yulu Business Case Study

Introduction

Yulu, India's pioneering micro-mobility service provider, has embarked on a mission to revolutionize daily commutes by offering unique, sustainable transportation solutions. However, recent revenue setbacks have prompted Yulu to seek the expertise of a consulting company to delve into the factors influencing the demand for their shared electric cycles, specifically in the Indian market.
The company wants to know:
● Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
● How well those variables describe the electric cycle demands.

1. Data

The analysis was done on the data located at -
https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089

2. Libraries

Below are the libraries required for analysing and visualizing data

In [1]:
```
# libraries to analyze data
import numpy as np
import pandas as pd

# libraries to visualize data
import matplotlib.pyplot as plt
import seaborn as sns

# Misc libraries
import random

# scipy stats library
import scipy.stats as stats
```

3. Data loading and exploratory data analysis

Loading the data into Pandas dataframe for easily handling of data

In [2]:
```
# read the file into a pandas dataframe
df = pd.read_csv('yulu_data.csv')
# look at the datatypes of the columns
print('*************************************************')
print(df.info())
print('*************************************************\n')
print('*************************************************')
```

```python
print(f'Shape of the dataset is {df.shape}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of nan/null values in each column: \n{df.isna().sum()}')
print('**************************************************\n')
print('**************************************************')
print(f'Number of unique values in each column: \n{df.nunique()}')
print('**************************************************\n')
print('**************************************************')
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
print('**************************************************')
```

```
**************************************************
<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
None
**************************************************

**************************************************
Shape of the dataset is (10886, 12)
**************************************************

**************************************************
Number of nan/null values in each column: 
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
**************************************************

**************************************************
Number of unique values in each column: 
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
*************************************************

*************************************************
Duplicate entries:
False    10886
Name: count, dtype: int64
*************************************************
```

In [3]:

```
# look at the top 5 rows
df.head()
```

Out[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	cou
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	

Insight

- A quick look at the information of the data reveals that there are **10886 rows and 12 columns** implying information is available for 10886 different datetime. The information includes *season, holiday, workingday, weather, temp, atemp, humidity, windspeed, casual, registered and count*. The datatype of *datetime* is *object* which needs to be converted to *datetime64* datatype for easy handling of date and time.
- We can also infer that **there are no missing values or nulls** in the dataset.
- There are **10886 datetime entries, 4 seasons, 4 weathers**.
- There are **no duplicate entries**.
- It makes sense to convert datatype of *datetime* to *datetime64* from *object* and to convert *season, holiday, workingday and weather* to *category* datatype

In [4]:

```
df["datetime"] = pd.to_datetime(df["datetime"])
categorical_columns = ['season', 'holiday', 'workingday', 'weather']
for col in categorical_columns:
    df[col] = df[col].astype('category')
print(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
None
```

In [5]:
```
df.describe()
```

Out[5]:

	datetime	temp	atemp	humidity	windspeed	casual	registered
count	10886	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177
min	2011-01-01 00:00:00	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000
25%	2011-07-02 07:15:00	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000
50%	2012-01-01 20:30:00	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000
75%	2012-07-01 12:45:00	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000
max	2012-12-19 23:00:00	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000
std	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033

In [6]:
```
df.describe(include='category')
```

Out[6]:

	season	holiday	workingday	weather
count	10886	10886	10886	10886
unique	4	2	2	4
top	4	0	1	1
freq	2734	10575	7412	7192

Insight

- The **minimum timestamp** is **2011-01-01 00:00:00** and **maximum timestamp** is **2012-12-19 23:00:00** implying there is data for around 2 years.
- The maximum number of user for a given timestamp is 977.

4. Detailed Analysis

4.1. Detecting outliers

4.1.1. Outliers for every continuous variable

In [7]:

```
# helper function to detect outliers
def detectOutliers(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3-q1
    lower_outliers = df[df<(q1-1.5*iqr)]
    higher_outliers = df[df>(q3+1.5*iqr)]
    return lower_outliers, higher_outliers
```

In [8]:

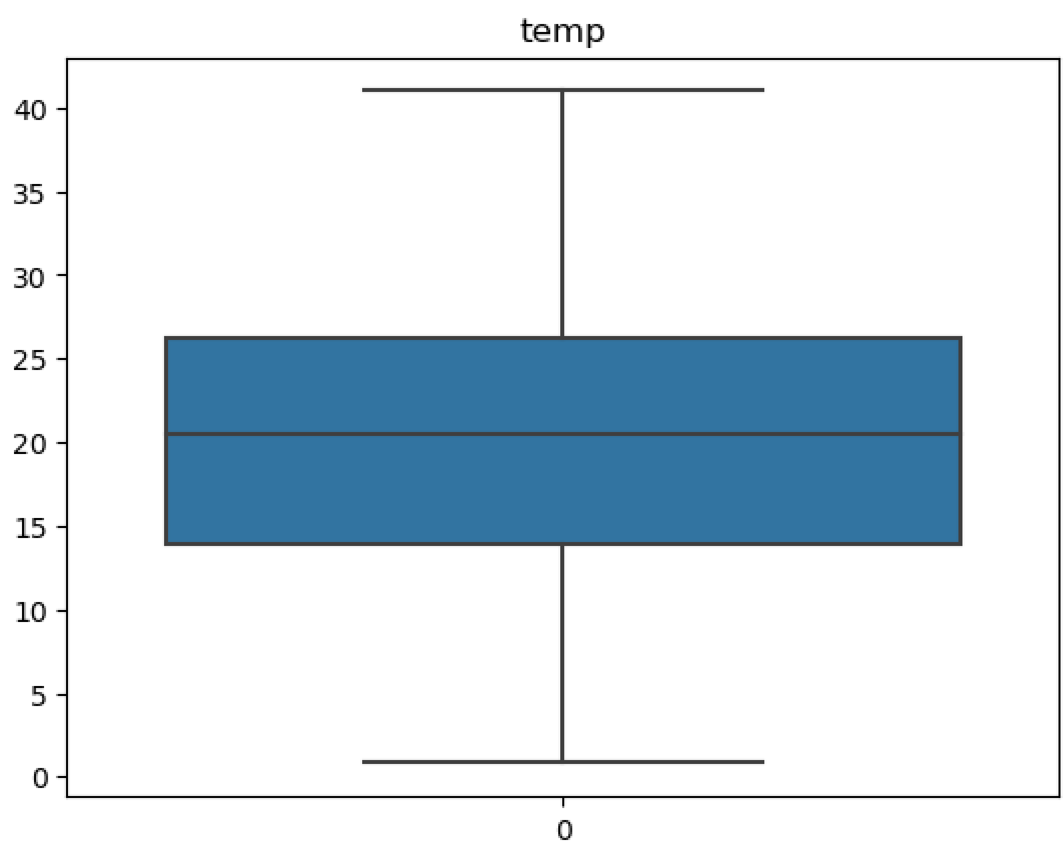
```
numerical_columns = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', '
column_outlier_dictionary = {}
for column in numerical_columns:
    print("***********************************")
    print(f'Outliers of \'{column}\' column are:')
    lower_outliers, higher_outliers = detectOutliers(df[column])
    print("Lower outliers:\n", lower_outliers)
    print("\nHigher outliers:\n", higher_outliers)
    print("***********************************")
    column_outlier_dictionary[column] = [lower_outliers, higher_outliers]
```

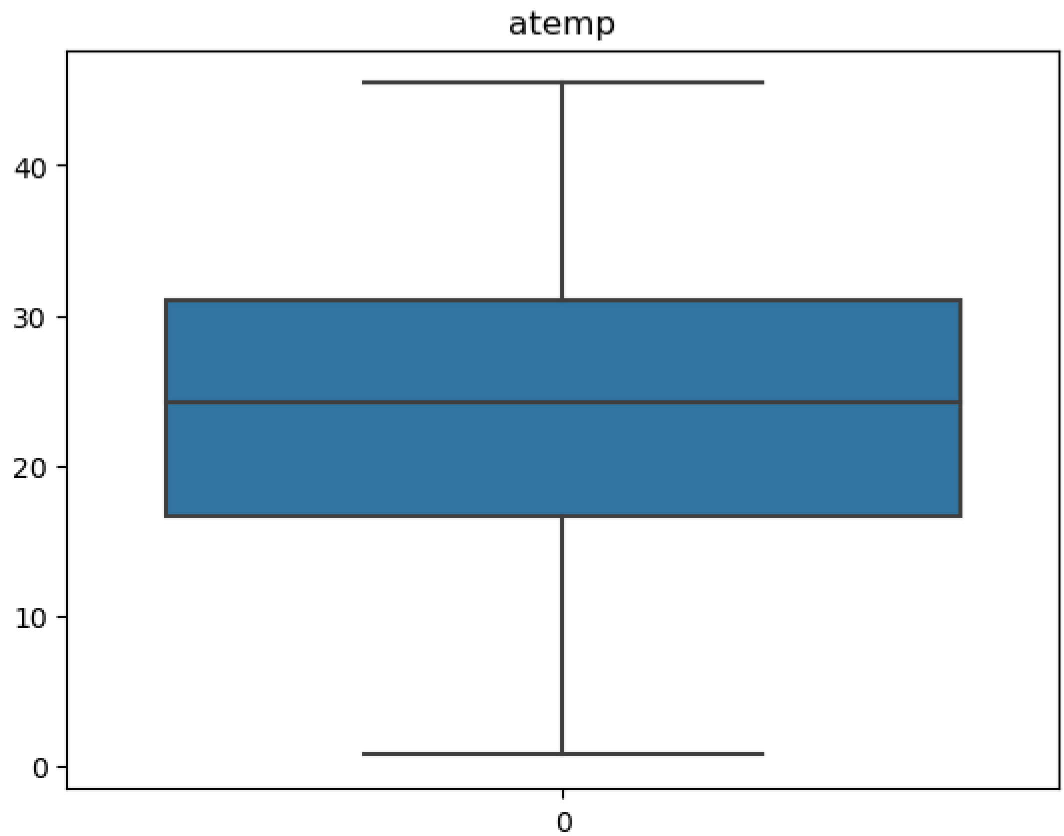
```
***********************************
Outliers of 'temp' column are:
Lower outliers:
 Series([], Name: temp, dtype: float64)

Higher outliers:
 Series([], Name: temp, dtype: float64)
***********************************
***********************************
Outliers of 'atemp' column are:
Lower outliers:
 Series([], Name: atemp, dtype: float64)

Higher outliers:
 Series([], Name: atemp, dtype: float64)
***********************************
***********************************
Outliers of 'humidity' column are:
Lower outliers:
 1091    0
1092    0
1093    0
1094    0
1095    0
1096    0
1097    0
1098    0
1099    0
1100    0
1101    0
1102    0
1103    0
1104    0
1105    0
1106    0
1107    0
```

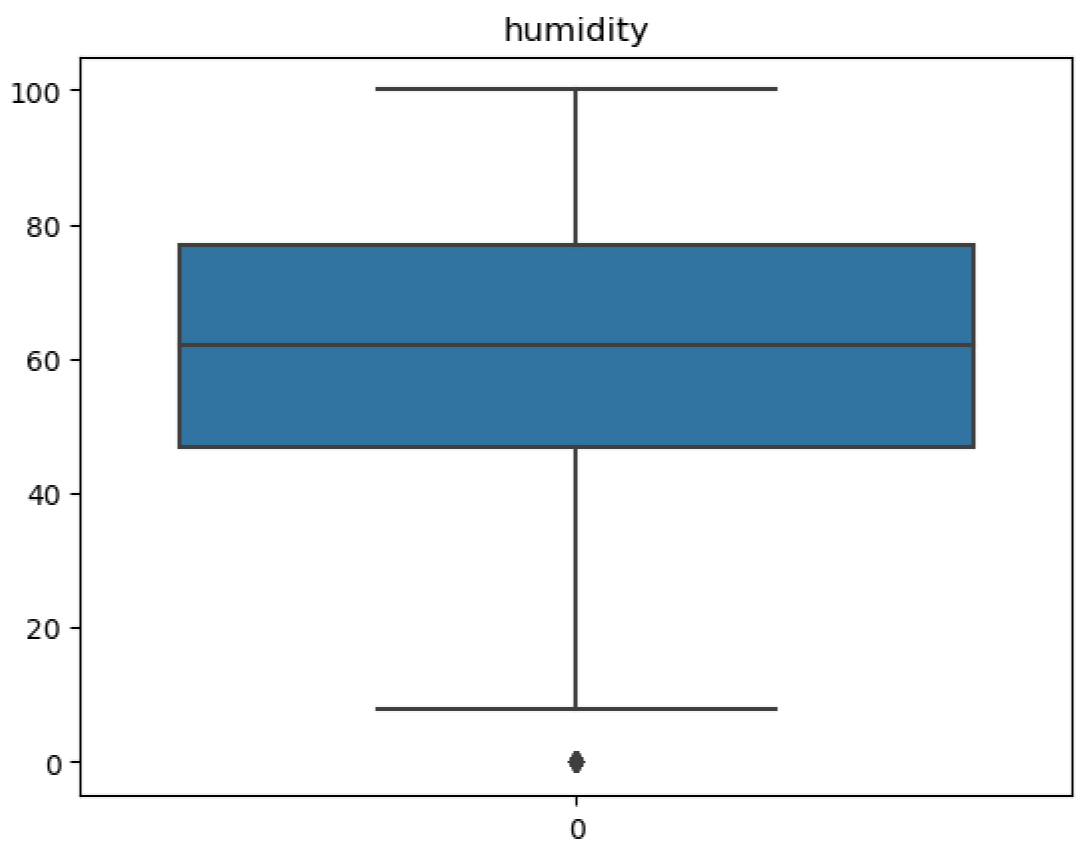
```
1108    0
1109    0
1110    0
1111    0
1112    0
Name: humidity, dtype: int64

Higher outliers:
 Series([], Name: humidity, dtype: int64)
************************************
************************************
Outliers of 'windspeed' column are:
Lower outliers:
 Series([], Name: windspeed, dtype: float64)

Higher outliers:
 175      32.9975
178      36.9974
194      35.0008
196      35.0008
265      39.0007
          ...
10013    32.9975
10154    32.9975
10263    43.0006
10540    32.9975
10853    32.9975
Name: windspeed, Length: 227, dtype: float64
************************************
************************************
Outliers of 'casual' column are:
Lower outliers:
 Series([], Name: casual, dtype: int64)

Higher outliers:
 1173     144
1174     149
1175     124
1311     126
1312     174
        ...
10610    122
10611    148
10612    164
10613    167
10614    139
Name: casual, Length: 749, dtype: int64
************************************
************************************
Outliers of 'registered' column are:
Lower outliers:
 Series([], Name: registered, dtype: int64)

Higher outliers:
 1987     539
2011     532
2059     540
2179     521
2371     516
        ...
10855    533
10856    512
10870    665
10879    536
10880    546
Name: registered, Length: 423, dtype: int64
```

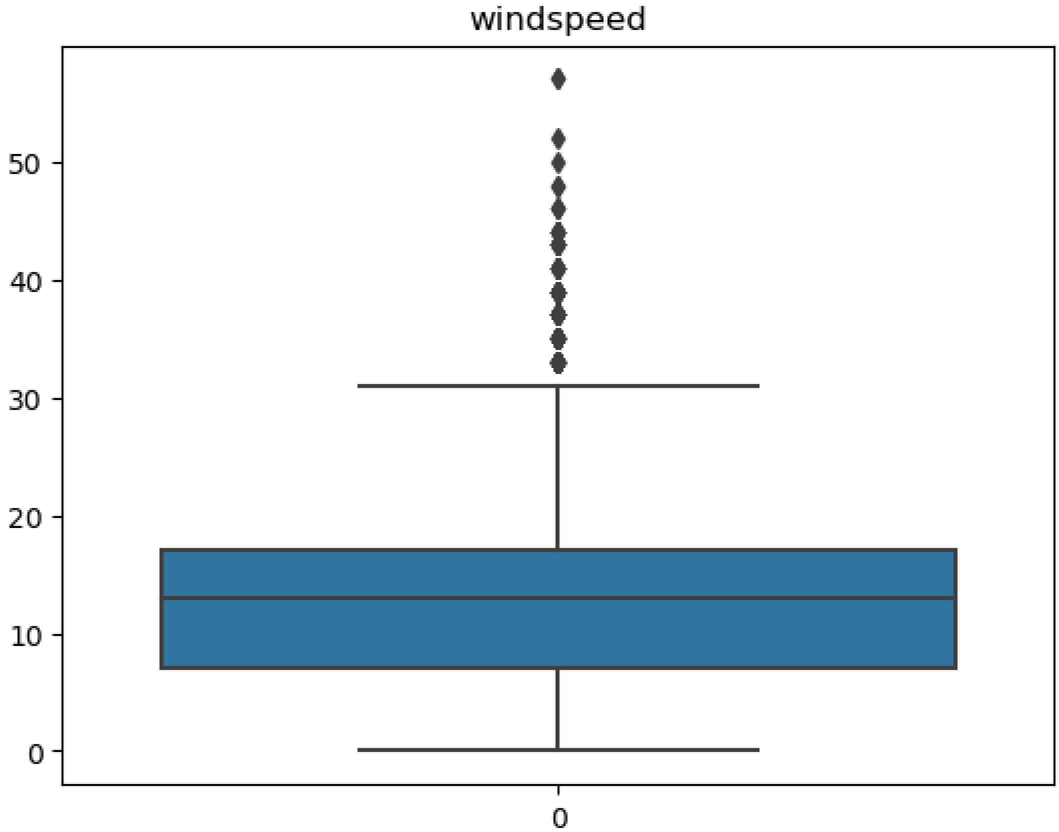
```
**********************************
**********************************
Outliers of 'count' column are:
Lower outliers:
 Series([], Name: count, dtype: int64)

Higher outliers:
 6611     712
6634     676
6635     734
6649     662
6658     782
        ...
10678    724
10702    688
10726    679
10846    662
10870    678
Name: count, Length: 300, dtype: int64
**********************************
```

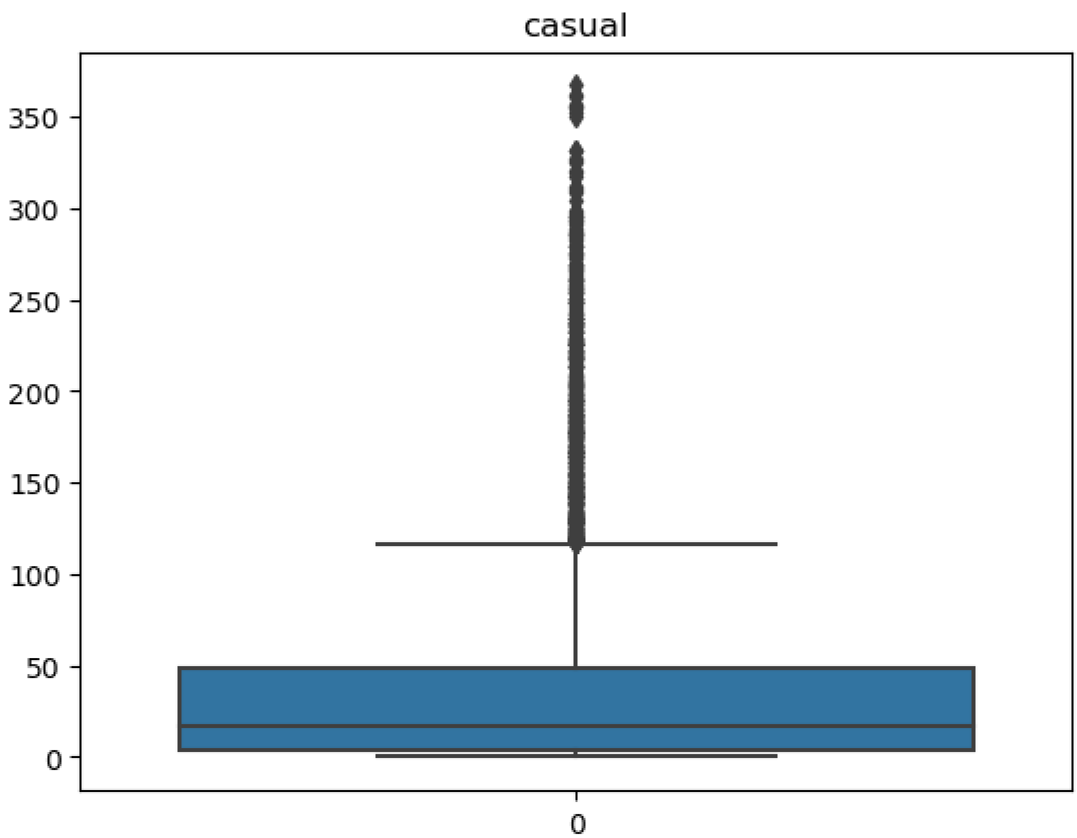
In [9]:
```
for column in numerical_columns:
    sns.boxplot(data=df[column])
    plt.title(column)
    plt.show()
```

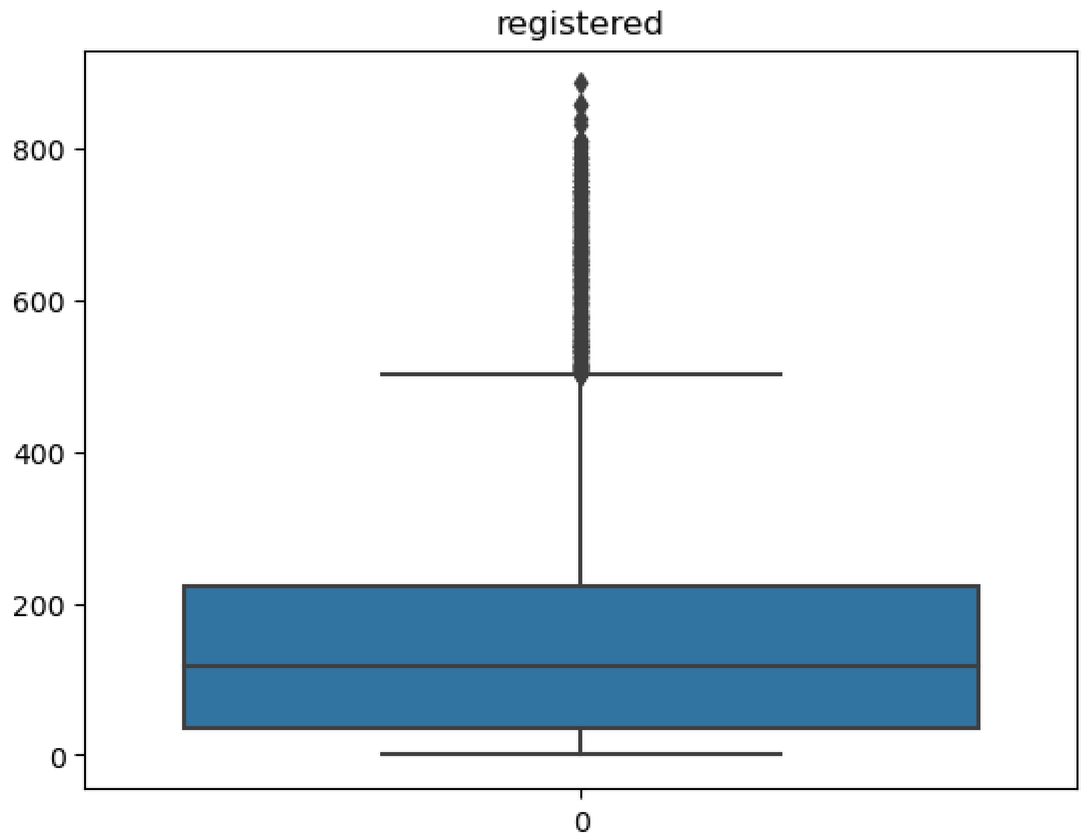


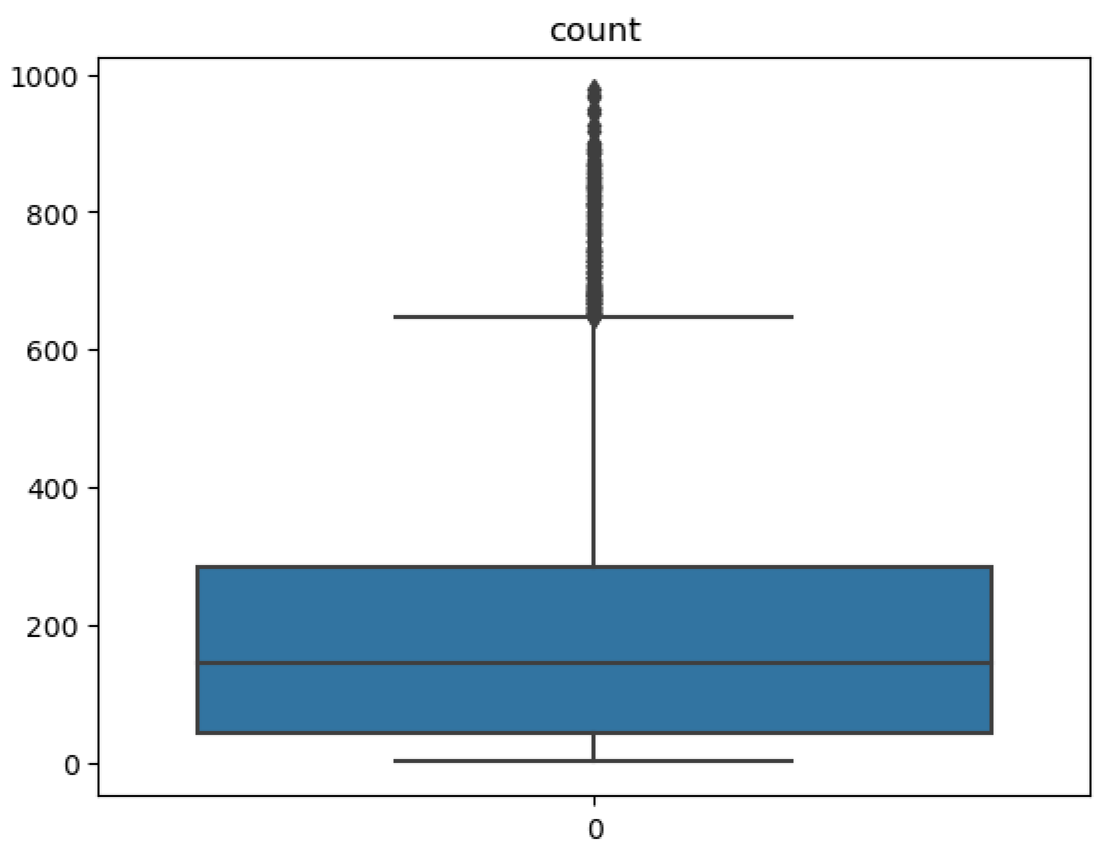

atemp
0
10
20
30
40
0


humidity
0
20
40
60
80
100
0


windspeed
50
40
30
20
10
0
0


casual
350
300
250
200
150
100
50
0
0





In [10]:
```
total_entries = df.shape[0]
for key, value in column_outlier_dictionary.items():
    total_outliers = len(value[0]) + len(value[1])
    outlier_percent = round((total_outliers/total_entries)*100, 2)
    print(f'The column \'{key}\' has {len(value[0]) + len(value[1])} outliers which is {
```

```
The column 'temp' has 0 outliers which is 0.0% of the data
The column 'atemp' has 0 outliers which is 0.0% of the data
The column 'humidity' has 22 outliers which is 0.2% of the data
The column 'windspeed' has 227 outliers which is 2.09% of the data
```

```
The column 'casual' has 749 outliers which is 6.88% of the data
The column 'registered' has 423 outliers which is 3.89% of the data
The column 'count' has 300 outliers which is 2.76% of the data
```

Insight

- There are no outliers in *temp* and *atemp* columns.
- There are 22 outliers in *humidity*, 227 in *windspeed*, 749 in *casual*, 423 in *registered* and 300 in *count* column.

4.1.2. Remove the outliers

In [11]:
```
if 0 :
    outlier_indices = []
    for key, value in column_outlier_dictionary.items():
        lower_outliers = value[0]
        higher_outliers = value[1]
        outlier_indices.extend(lower_outliers.index)
        outlier_indices.extend(higher_outliers.index)
    outlier_indices = list(set(outlier_indices))
    df.drop(outlier_indices, inplace=True)
    df.info()
```

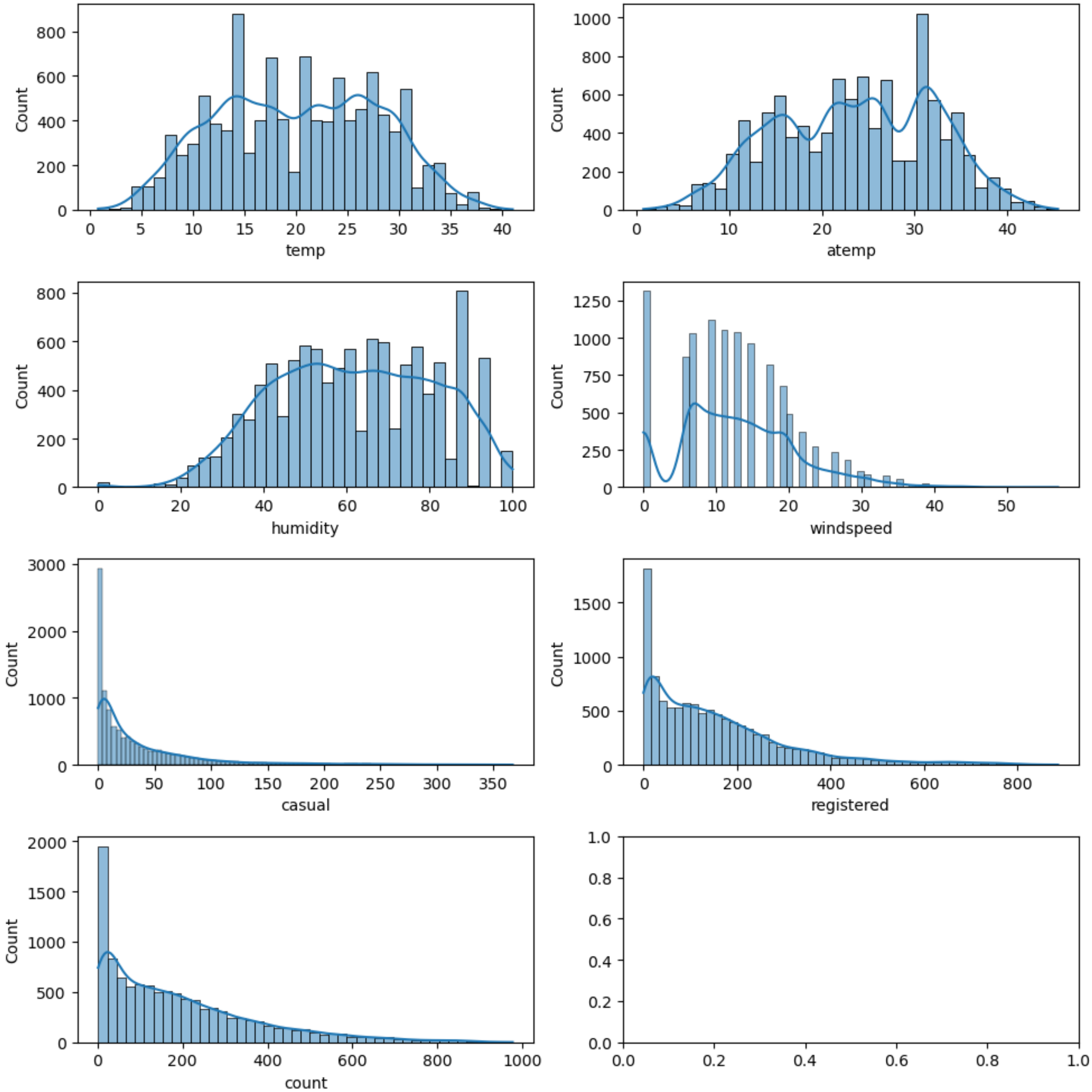
Insight

- Not removing any outliers

4.2. Univariate analysis

4.2.1. Numerical Variables

In [12]:
```
fig, axes = plt.subplots(nrows=4, ncols=2, sharex=False, sharey=False, figsize=(10, 10))

idx=0
for row in range(4):
    for col in range(2):
        if(idx < len(numerical_columns)):
            sns.histplot(ax=axes[row, col], data=df, x = numerical_columns[idx], kde=Tru
        idx += 1
plt.tight_layout()
plt.show()
```



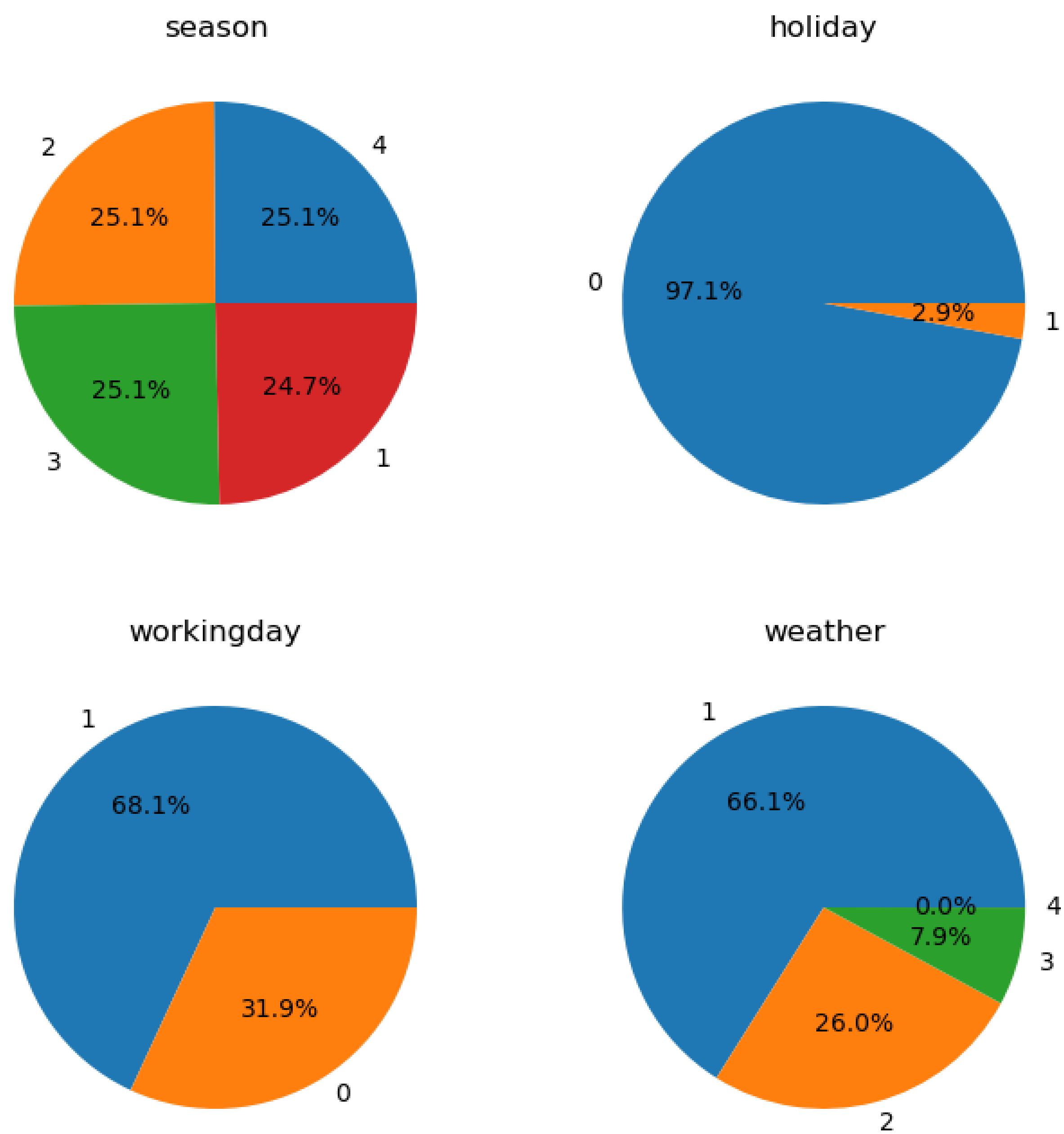
Insight

- *temp*, *atemp* and *humidity* seems to follow normal distribution.
- *windspeed*, *casual*, *registered* and *count* seems to follow log-normal distribution

4.2.2. Categorical Variables

In [13]:
```
# 'season', 'holiday', 'workingday', 'weather'
categorical_columns = ["Gender", "Age", "Occupation", "City_Category", "Stay_In_Current_
plt.figure(figsize=(8,8))
plt.subplot(2,2,1)
data = df["season"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("season")
plt.subplot(2,2,2)
data = df["holiday"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
```

```
plt.title("holiday")
plt.subplot(2,2,3)
data = df["workingday"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("workingday")
plt.subplot(2,2,4)
data = df["weather"].value_counts()
plt.pie(data.values, labels = data.index, autopct='%.1f%%')
plt.title("weather")
plt.show()
```



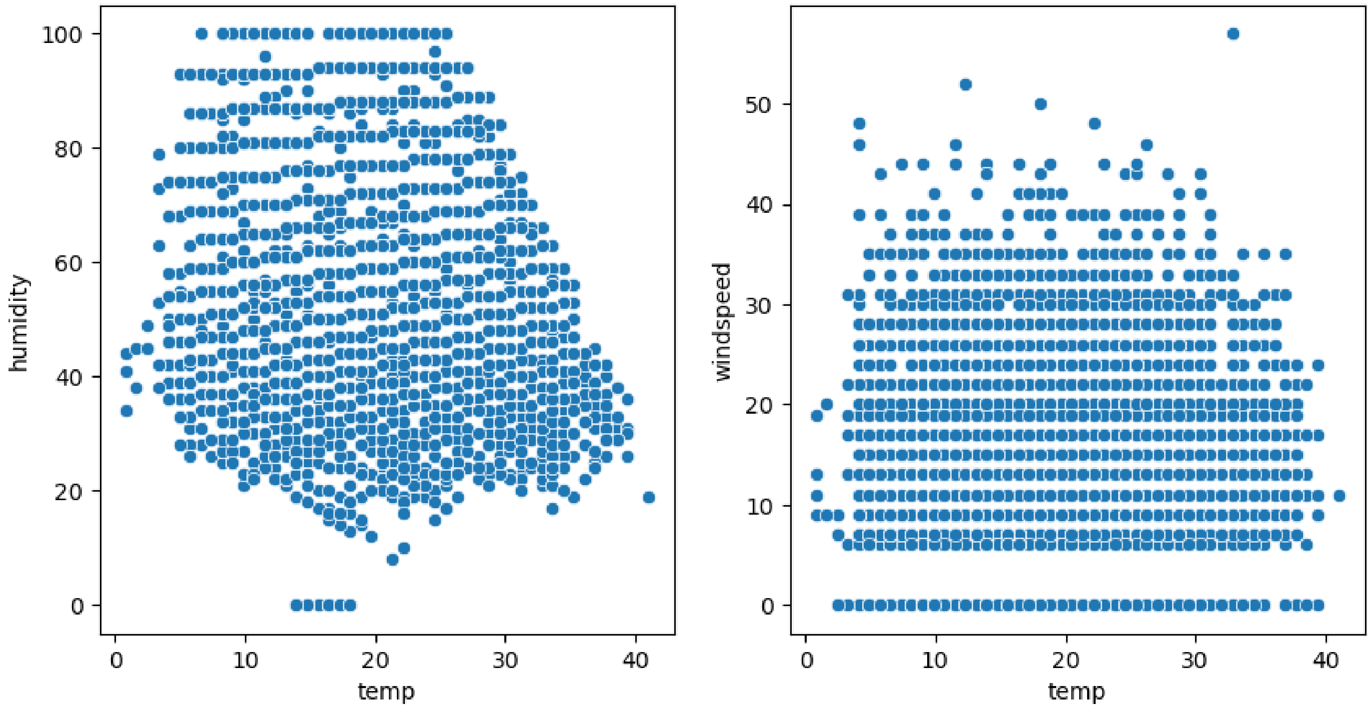
Insight

- Bikes have been rented almost equally for all seasons(26% in spring, 24% in summer, 24% in fall and 26% in winter)
- As expected the bikes have been rented out more during *workingday* - 71%
- People prefer to rent bike during *Clear, Few clouds, partly cloudy* weather - 65%

4.3. Bivariate analysis

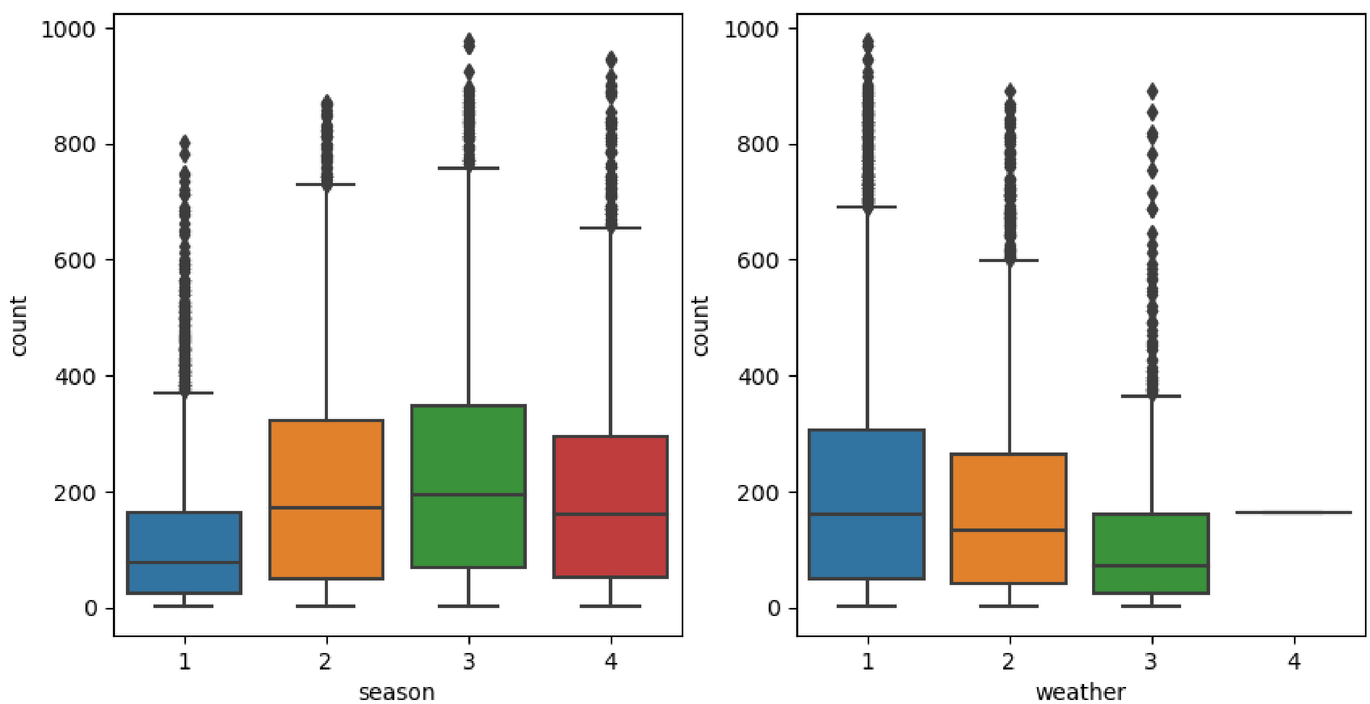
4.3.1. Numerical Variables

In [14]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=False, figsize=(10, 5))
sns.scatterplot(ax=axes[0], data= df, x='temp', y='humidity')
sns.scatterplot(ax=axes[1], data= df, x='temp', y='windspeed')
plt.show()
```



4.3.2. Categorical Variables

In [15]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=False, figsize=(10, 5))
sns.boxplot(ax = axes[0], data= df, x='season', y='count')
sns.boxplot(ax = axes[1], data= df, x='weather', y='count')
plt.show()
```

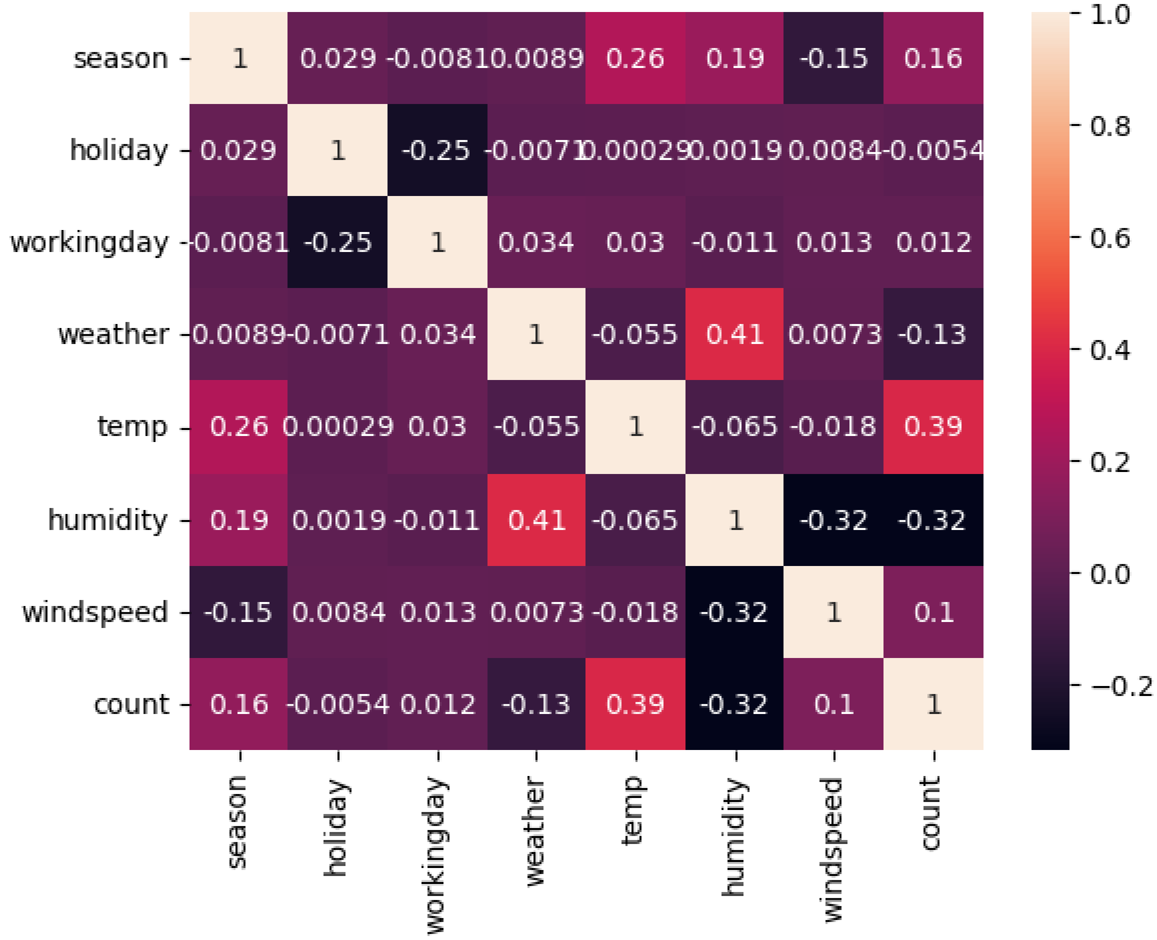


Insight

- Comparitavelu, more bikes are rented during summer, fall and winter.
- Almost no bikes are rented during Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog

Relationship between dependent variable "Count" and independent variables

In [16]:
```
reduced_df = df[['season', 'holiday', 'workingday', 'weather', 'temp', 'humidity', 'wind
sns.heatmap(reduced_df.corr(), annot=True)
plt.show()
```



Insight

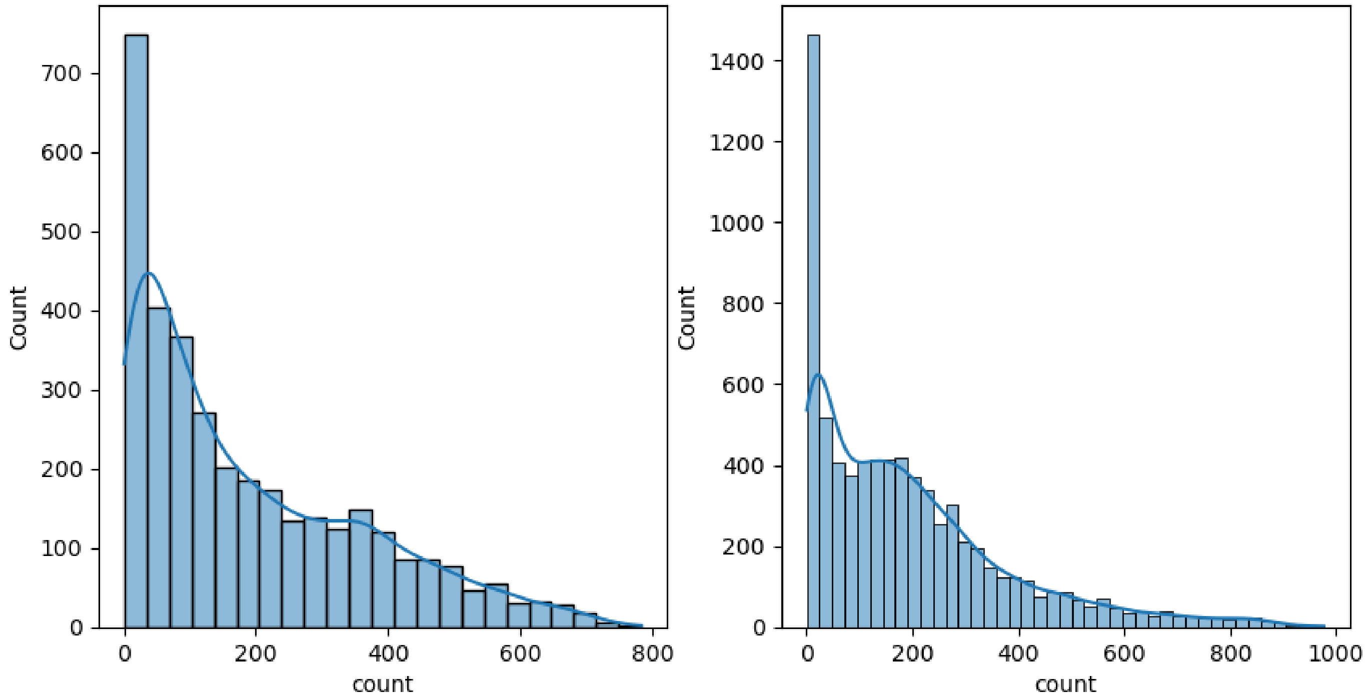
- As expected humidity and weather are correlated
- It is interesteting to see no. of bikes rides being related to temperature

5. Tests

5.1. Is there any significant difference between the no. of bike rides on Weekdays and Weekends

- **Null Hypothesis(H0)** :There is no difference between the no. of bike rides on Weekdays and Weekends
- **Alternate Hypothesis(H1)** :There is difference between the no. of bike rides on Weekdays and Weekends
- **Significance level** : 5%

In [17]:
```
g1 = df[df['workingday'] == 0]['count']
g2 = df[df['workingday'] == 1]['count']
fig, axes = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=False, figsize=(10, 5))
sns.histplot(ax=axes[0], x=g1, kde=True)
sns.histplot(ax=axes[1], x=g2, kde=True)
plt.show()
```



Test for normal distribution - Shapiro-Wilk test

In [18]:
```
print("Group1:")
test_stat, p_value = stats.shapiro(g1)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group2:")
test_stat, p_value = stats.shapiro(g2)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')
```

```
Group1:
p-value:  4.203895392974451e-45
The sample doe not follow normal distribution
Group2:
p-value:  0.0
The sample doe not follow normal distribution
```

```
C:\ProgramData\anaconda3\Lib\site-packages\scipy\stats\_morestats.py:1882: UserWarning:
p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

Test for equale variance - Lavene's test

In [19]:
```
test_stat, p_value = stats.levene(g1, g2)
print('p-value', p_value)
if p_value > 0.05:
    print('The samples have homogenous variance')
```

```
else:
    print('The samples do not have homogenous variance')
```

```
p-value 0.9437823280916695
The samples have homogenous variance
```

Since the data doesnt follow normal distribution, the Z, T and ANOVA tests cannot be applied. I will apply the Mann-Whitney U-Test for two independent samples

In [20]:
```
test_stat, p_value = stats.mannwhitneyu(g1, g2)
print('p-value', p_value)
if p_value > 0.05:
    print('Fail to reject Null Hypothesis')
else:
    print('Reject Null Hypothesis')
```

```
p-value 0.9679139953914079
Fail to reject Null Hypothesis
```

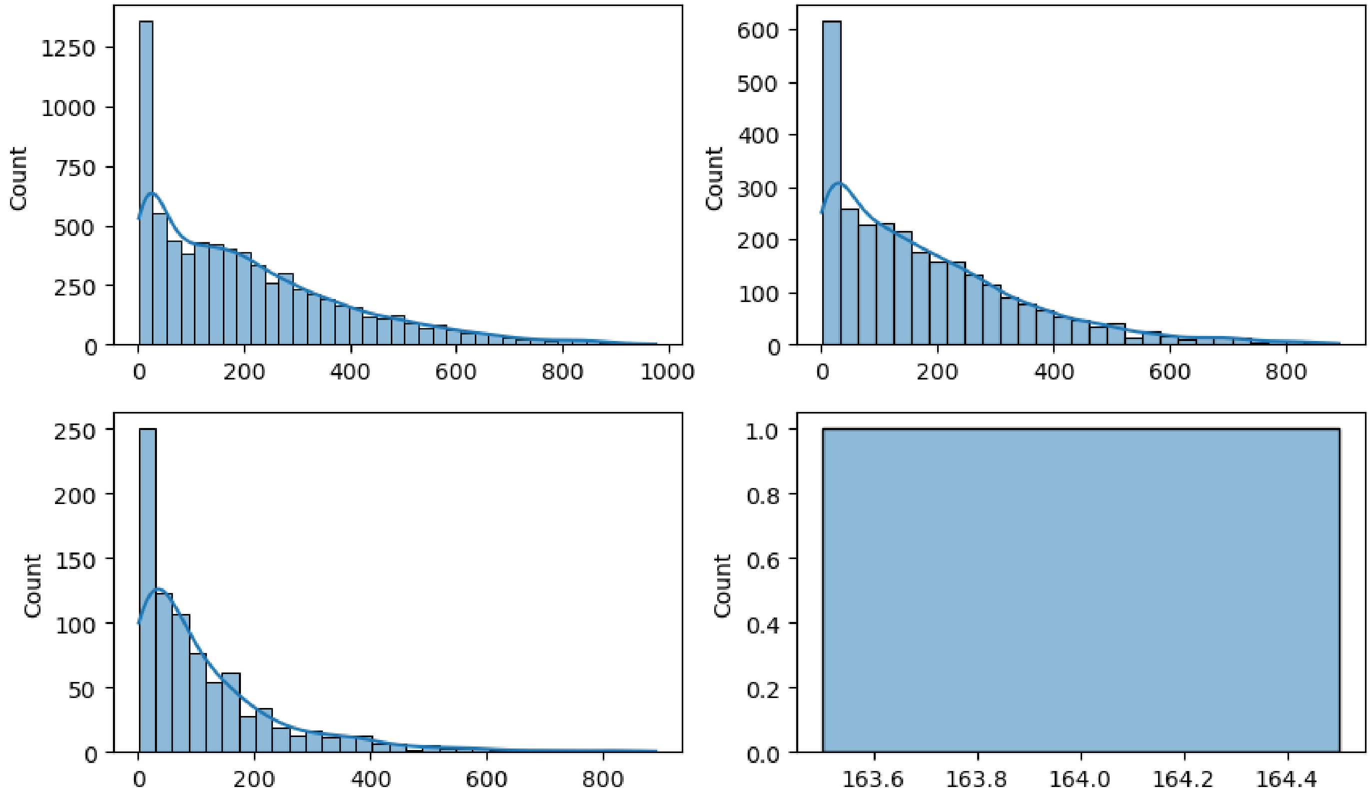
Insight

- There is no significant differnce between the no. of bike rides on Weekdays and Weekends

5.2. Is the demand of bicycles on rent the same for different Weather conditions?

- **Null Hypothesis(H0)** : The demand for bicycles on rent is the same for different weather conditions.
- **Alternate Hypothesis(H1)** : The demand for bicycles on rent is different for different weather conditions.
- **Significance level** : 5%

In [21]:
```
g1 = df[df['weather'] == 1]['count'].values
g2 = df[df['weather'] == 2]['count'].values
g3 = df[df['weather'] == 3]['count'].values
g4 = df[df['weather'] == 4]['count'].values
fig, axes = plt.subplots(nrows=2, ncols=2, sharex=False, sharey=False, figsize=(10, 6))
sns.histplot(ax=axes[0, 0], x=g1, kde=True)
sns.histplot(ax=axes[0, 1], x=g2, kde=True)
sns.histplot(ax=axes[1, 0], x=g3, kde=True)
sns.histplot(ax=axes[1, 1], x=g4, kde=True)
plt.show()
```



Test for normal distribution - Shapiro-Wilk test

In [22]:

```
print("Group1:")
test_stat, p_value = stats.shapiro(g1)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group2:")
test_stat, p_value = stats.shapiro(g2)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group3:")
test_stat, p_value = stats.shapiro(g3)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')
```

```
Group1:
p-value:  0.0
The sample doe not follow normal distribution
Group2:
p-value:  9.781063280987223e-43
The sample doe not follow normal distribution
Group3:
p-value:  3.876090133422781e-33
The sample doe not follow normal distribution
```

```
C:\ProgramData\anaconda3\Lib\site-packages\scipy\stats\_morestats.py:1882: UserWarning:
p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

Test for equale variance - Lavene's test

In [23]:
```
test_stat, p_value = stats.levene(g1, g2, g3)
print('p-value', p_value)
if p_value > 0.05:
    print('The samples have homogenous variance')
else:
    print('The samples do not have homogenous variance')
```

```
p-value 6.198278710731511e-36
The samples do not have homogenous variance
```

Note : Group4 is not used as it has only one data point.
Since the data doesnt follow normal distribution and doesnt show homogenous variances between groups, the Z, T and ANOVA tests cannot be applied. Since there are multiple groups, I will test using Kruskal-Wallis Test.

In [24]:
```
test_stat, p_value = stats.kruskal(g1, g2, g3)
print('p-value', p_value)
if p_value > 0.05:
    print('Fail to reject Null Hypothesis')
else:
    print('Reject Null Hypothesis')
```

```
p-value 3.122066178659941e-45
Reject Null Hypothesis
```

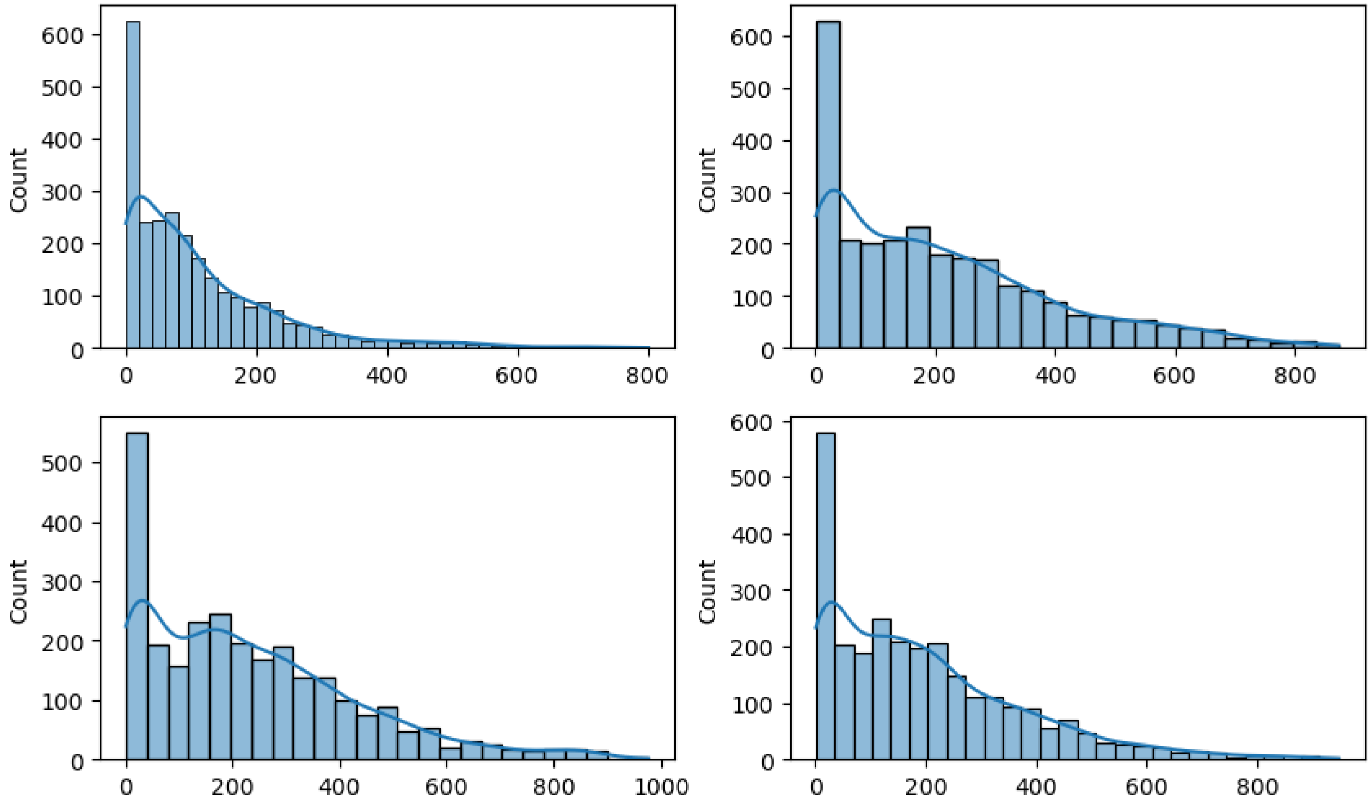
Insight

- The demand for bicycles on rent is different for different weather conditions.

5.3. Is the demand of bicycles on rent the same for different Seasons?

- **Null Hypothesis(H0)** : The demand for bicycles on rent is the same for different seasons.
- **Alternate Hypothesis(H1)** : The demand for bicycles on rent is different for different seasons.
- **Significance level** : 5%

In [25]:
```
g1 = df[df['season'] == 1]['count'].values
g2 = df[df['season'] == 2]['count'].values
g3 = df[df['season'] == 3]['count'].values
g4 = df[df['season'] == 4]['count'].values
fig, axes = plt.subplots(nrows=2, ncols=2, sharex=False, sharey=False, figsize=(10, 6))
sns.histplot(ax=axes[0, 0], x=g1, kde=True)
sns.histplot(ax=axes[0, 1], x=g2, kde=True)
sns.histplot(ax=axes[1, 0], x=g3, kde=True)
sns.histplot(ax=axes[1, 1], x=g4, kde=True)
plt.show()
```



Test for normal distribution - Shapiro-Wilk test

In [26]:
```python
print("Group1:")
test_stat, p_value = stats.shapiro(g1)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group2:")
test_stat, p_value = stats.shapiro(g2)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group3:")
test_stat, p_value = stats.shapiro(g3)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')

print("Group4:")
test_stat, p_value = stats.shapiro(g4)
print('p-value: ', p_value)
if p_value > 0.05:
    print('The sample follows normal distribution')
else:
    print('The sample doe not follow normal distribution')
```

```
Group1:
p-value:  0.0
The sample doe not follow normal distribution
Group2:
```

```
p-value:  6.039093315091269e-39
The sample doe not follow normal distribution
Group3:
p-value:  1.043458045587339e-36
The sample doe not follow normal distribution
Group4:
p-value:  1.1301682309549298e-39
The sample doe not follow normal distribution
```

Test for equale variance - Lavene's test

In [27]:
```python
test_stat, p_value = stats.levene(g1, g2, g3, g4)
print('p-value', p_value)
if p_value > 0.05:
    print('The samples have homogenous variance')
else:
    print('The samples do not have homogenous variance')
```

```
p-value 1.0147116860043298e-118
The samples do not have homogenous variance
```

Since the data doesnt follow normal distribution and doesnt show homogenous variances between groups, the Z, T and ANOVA tests cannot be applied. Since there are multiple groups, I will test using Kruskal-Wallis Test.

In [28]:
```python
test_stat, p_value = stats.kruskal(g1, g2, g3, g4)
print('p-value', p_value)
if p_value > 0.05:
    print('Fail to reject Null Hypothesis')
else:
    print('Reject Null Hypothesis')
```

```
p-value 2.479008372608633e-151
Reject Null Hypothesis
```

Insight

- The demand for bicycles on rent is different for different seasons.

5.4. Are the weather conditions significantly different during different seasons?

- **Null Hypothesis(H0)** : The weather conditions are same during different seasons.
- **Alternate Hypothesis(H1)** : The weather conditions are significantly differet during different seasons.
- **Significance level** : 5%

  As the groups are categorical, I will use Chi-Square test

In [29]:
```python
cross = pd.crosstab(index = df['weather'],
                    columns = df['season'],
                    values = df['count'],
                    aggfunc = np.sum)
print(cross)
```

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

Since the last row, weather 4 has 0 for 3 out of 4 columns, I will not consider weather 4

In [30]:
```
cross = pd.crosstab(index = df[df['weather'] != 4]['weather'],
                    columns = df['season'],
                    values = df['count'],
                    aggfunc = np.sum).to_numpy()[:3, :]
print(cross)
```

```
[[223009 426350 470116 356588]
 [ 76406 134177 139386 157191]
 [ 12919  27755  31160  30255]]
```

In [31]:
```
test_stat, p_value, _, _ = stats.chi2_contingency(cross)
print('p-value', p_value)
if p_value > 0.05:
    print('Fail to reject Null Hypothesis')
else:
    print('Reject Null Hypothesis')
```

```
p-value 0.0
Reject Null Hypothesis
```

Insight

- The weather conditions are significantly differet during different seasons.

6. Recommendation

- Yulu should promote rentals during summer, fall and winter as the demand is high during these months. It should attract people during spring by providing discounts.
- Yulu should promote rentals during good weather conditions and provide discounts during extreme weather to keep the users renting bikes
- Yulu should run campaings promoting the advantages of using electric bikes and how they care for the environment
- Yulu can efficiently maintain its inventory based on the season, weather and temperature.

In [ ]:
```

```