Yulu

May 9, 2024

1 Problem Statement

We've received a dataset containing the count of rental bikes from Yulu, along with timestamps, weather conditions, and day types. Our task is to examine the factors influencing the company's revenue and provide insights into the conditions impacting it.

[1]:
```
# import all the required python libraries


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels
from scipy.special import comb
from scipy.stats import binom
from scipy.stats import norm,t
from scipy.stats import poisson, expon,geom, ttest_1samp,␣
 ↪ttest_ind,ttest_ind_from_stats
from scipy.stats import shapiro, levene, kruskal, chi2, chi2_contingency
from statsmodels.graphics.gofplots import qqplot
```

Loading our dataset

[2]:
```
#Loading of the dataset
df=pd.read_csv("bike_sharing.csv")
df
```

[2]:

	datetime	season	holiday	workingday	weather	temp	\
0	2011-01-01 00:00:00	1	0	0	1	9.84	
1	2011-01-01 01:00:00	1	0	0	1	9.02	
2	2011-01-01 02:00:00	1	0	0	1	9.02	
3	2011-01-01 03:00:00	1	0	0	1	9.84	
4	2011-01-01 04:00:00	1	0	0	1	9.84	
…	…	…	…	…	…	…	
10881	2012-12-19 19:00:00	4	0	1	1	15.58	
10882	2012-12-19 20:00:00	4	0	1	1	14.76	
10883	2012-12-19 21:00:00	4	0	1	1	13.94	

```
10884  2012-12-19 22:00:00       4        0           1        1  13.94
10885  2012-12-19 23:00:00       4        0           1        1  13.12

         atemp  humidity  windspeed  casual  registered  count
0       14.395        81     0.0000       3          13     16
1       13.635        80     0.0000       8          32     40
2       13.635        80     0.0000       5          27     32
3       14.395        75     0.0000       3          10     13
4       14.395        75     0.0000       0           1      1
…        …        …         …          …     …
10881   19.695        50    26.0027       7         329    336
10882   17.425        57    15.0013      10         231    241
10883   15.910        61    15.0013       4         164    168
10884   17.425        61     6.0032      12         117    129
10885   16.665        66     8.9981       4          84     88

[10886 rows x 12 columns]
```

2 Observation on shape of data, data types of all the attributes, conversion of categorical attributes to 'category', missing value detection, statistical summary:

[3]:
```
#Checking of the shape of the data
df.shape
```

[3]:
```
(10886, 12)
```

[4]:
```
#Checking number of unique values in attributes
df.nunique()
```

[4]:
```
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
```

[5]:
```
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
```

[6]:
```
# convert datetime from object to datetime category
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  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: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

[7]:
```
# convert categorical variable to category type
df["season"]=df["season"].astype("object")
df["holiday"]=df["holiday"].astype("object")
```

```
df["workingday"]=df["workingday"].astype("object")
df["weather"]=df["weather"].astype("object")
df.info()
```

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

[8]:
```
#Checking of missing values in the dataset
df.isnull().sum()
```

[8]:
```
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
```

[9]:
```
#Statistical summary of numeric variables in the dataset
df.describe()
```

[9]:
```
              temp         atemp      humidity     windspeed        casual  \
count  10886.00000  10886.000000  10886.000000  10886.000000  10886.000000
mean      20.23086     23.655084     61.886460     12.799395     36.021955
```

std	7.79159	8.474601	19.245033	8.164537	49.960477
min	0.82000	0.760000	0.000000	0.000000	0.000000
25%	13.94000	16.665000	47.000000	7.001500	4.000000
50%	20.50000	24.240000	62.000000	12.998000	17.000000
75%	26.24000	31.060000	77.000000	16.997900	49.000000
max	41.00000	45.455000	100.000000	56.996900	367.000000

	registered	count
count	10886.000000	10886.000000
mean	155.552177	191.574132
std	151.039033	181.144454
min	0.000000	1.000000
25%	36.000000	42.000000
50%	118.000000	145.000000
75%	222.000000	284.000000
max	886.000000	977.000000

[10]:
```
#Description of Object type data in dataset
df.describe(include="object")
```

[10]:

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

[11]:
```
#Description of datetime type data in dataset
df.describe(include="datetime")
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\3165184414.py:2:
FutureWarning: Treating datetime data as categorical rather than numeric in
`.describe` is deprecated and will be removed in a future version of pandas.
Specify `datetime_is_numeric=True` to silence this warning and adopt the future
behavior now.
  df.describe(include="datetime")
```

[11]:

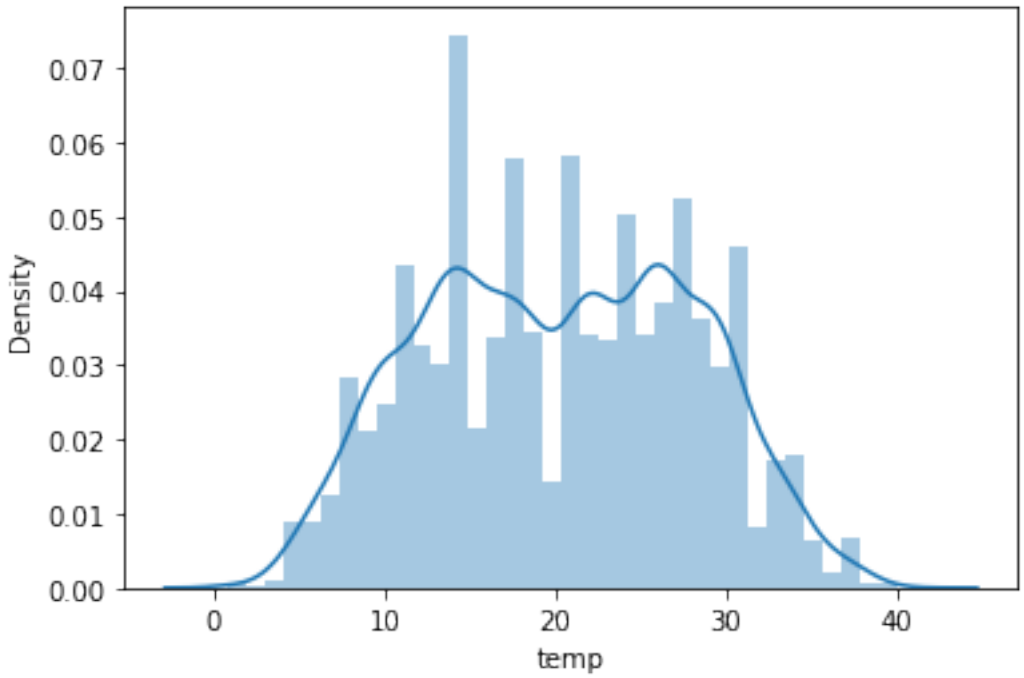
	datetime
count	10886
unique	10886
top	2011-01-01 00:00:00
freq	1
first	2011-01-01 00:00:00
last	2012-12-19 23:00:00

3 Univariate Analysis

[12]:
```
#distplot for temp attribute
sns.distplot(df["temp"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\977343850.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["temp"])
```

[12]: <AxesSubplot: xlabel='temp', ylabel='Density'>

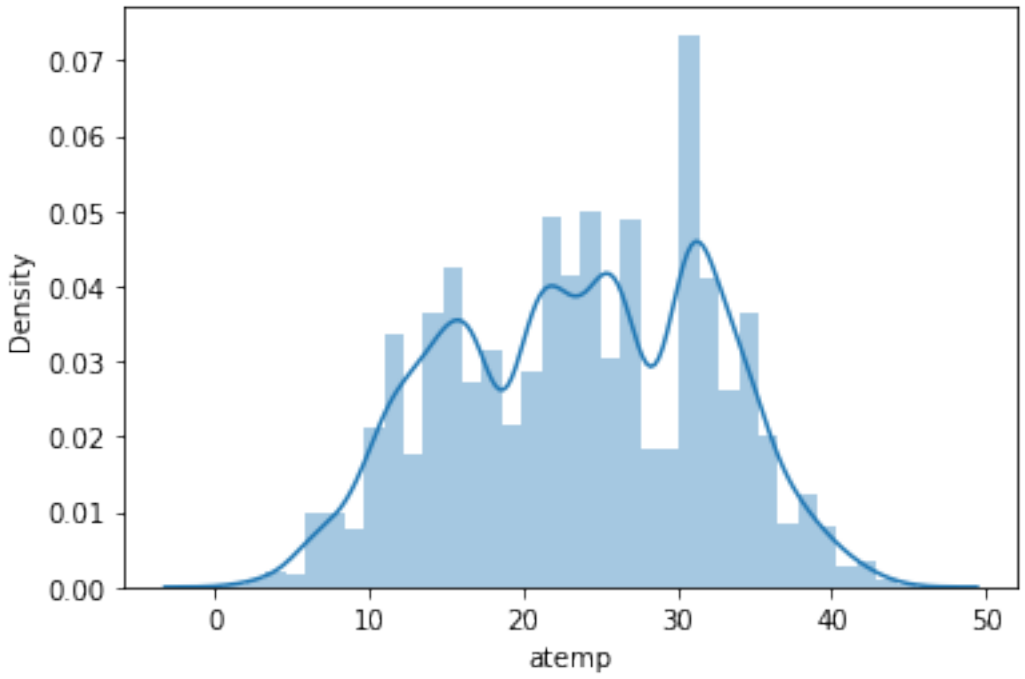


[13]:
```
#distplot for atemp attribute
sns.distplot(df["atemp"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\140082757.py:2:
UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["atemp"])
```

[13]: <AxesSubplot: xlabel='atemp', ylabel='Density'>

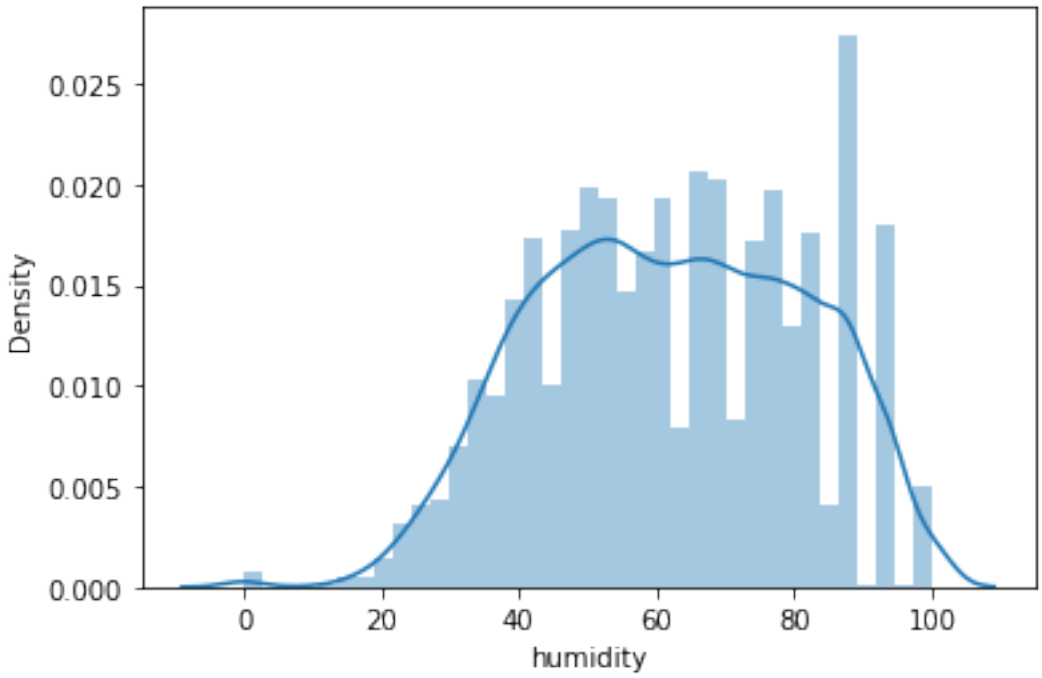


[14]:
```
#distplot for humidity attribute
sns.distplot(df["humidity"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\2949485006.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
```

```
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["humidity"])
```

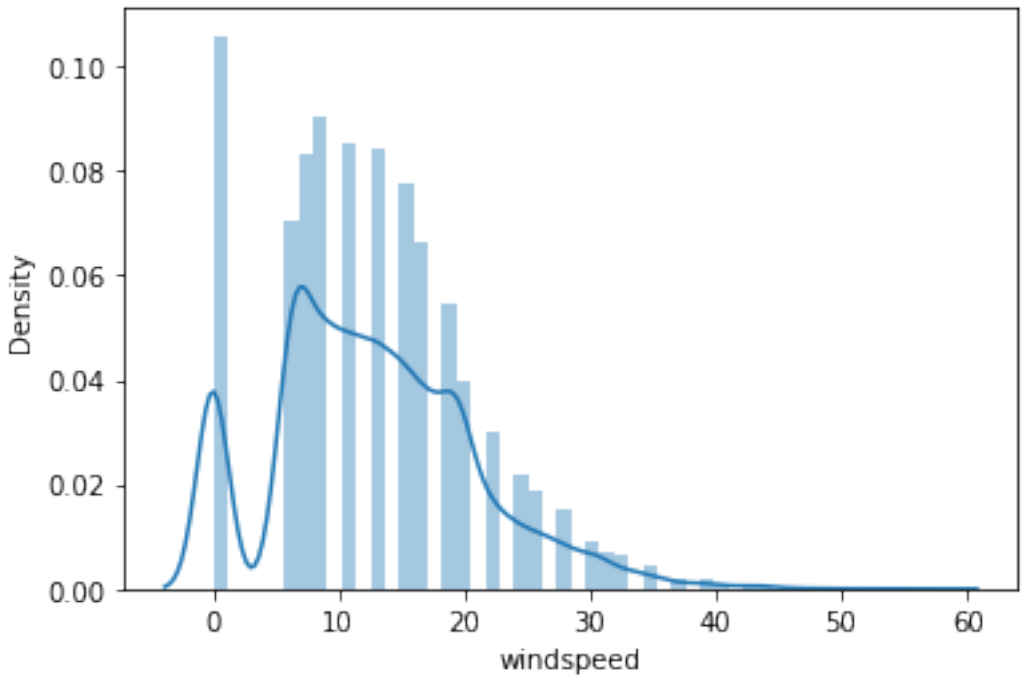
[14]: <AxesSubplot: xlabel='humidity', ylabel='Density'>



[15]:
```
#distplot for windspeed attribute
sns.distplot(df["windspeed"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\1122790581.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["windspeed"])
```

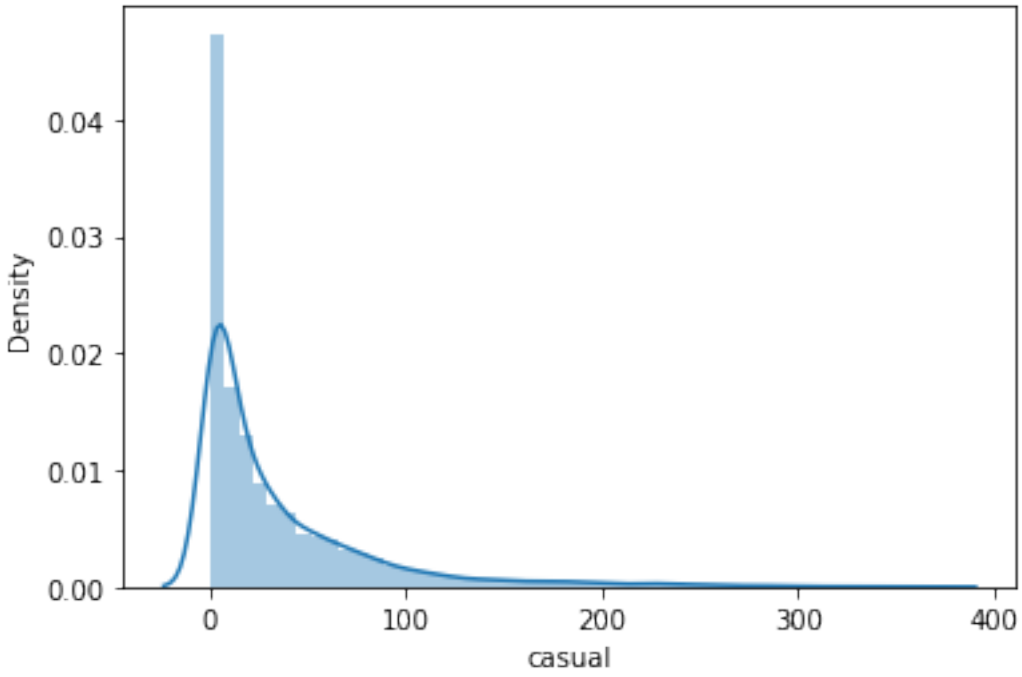
[15]: <AxesSubplot: xlabel='windspeed', ylabel='Density'>



[16]:
```
#distplot for count of casual users
sns.distplot(df["casual"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\1210671214.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["casual"])
```

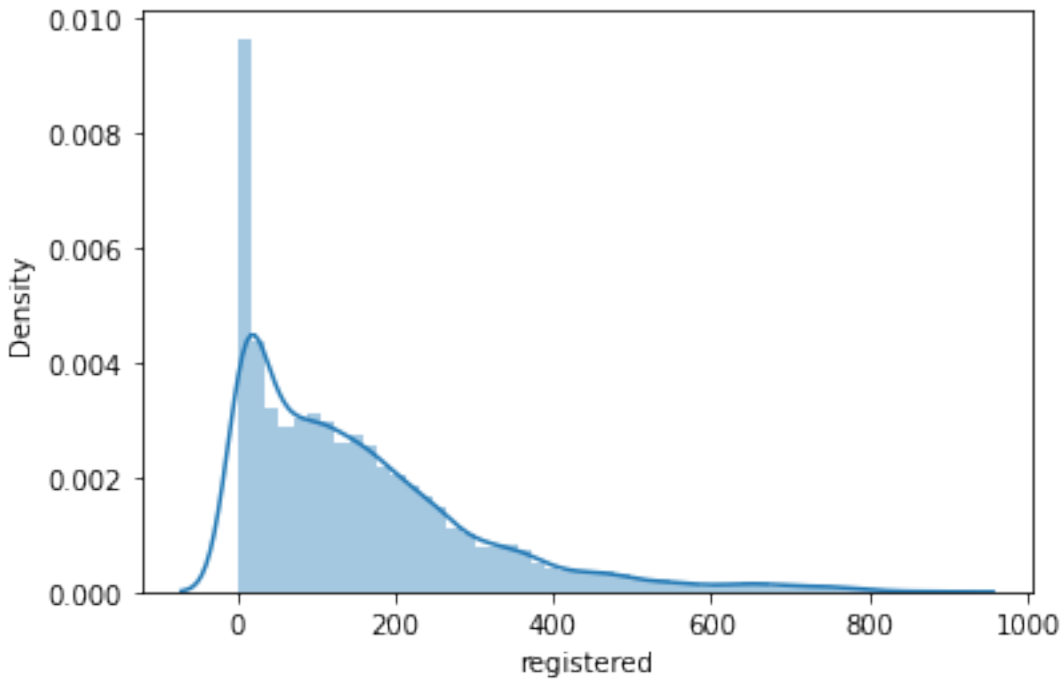
[16]: <AxesSubplot: xlabel='casual', ylabel='Density'>



[17]:
```
#distplot for count of registered users
sns.distplot(df["registered"])
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\2876350083.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["registered"])
```

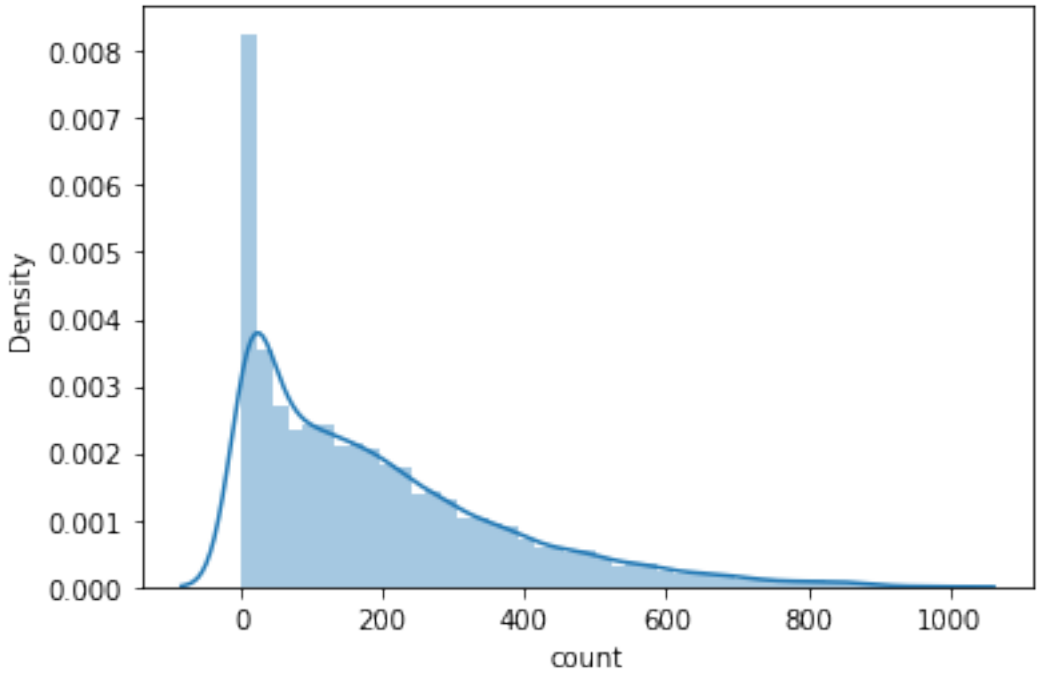
[17]: <AxesSubplot: xlabel='registered', ylabel='Density'>



[18]:
```
#distplot for total rental bikes including both casual and registered
sns.distplot(df["count"])
```

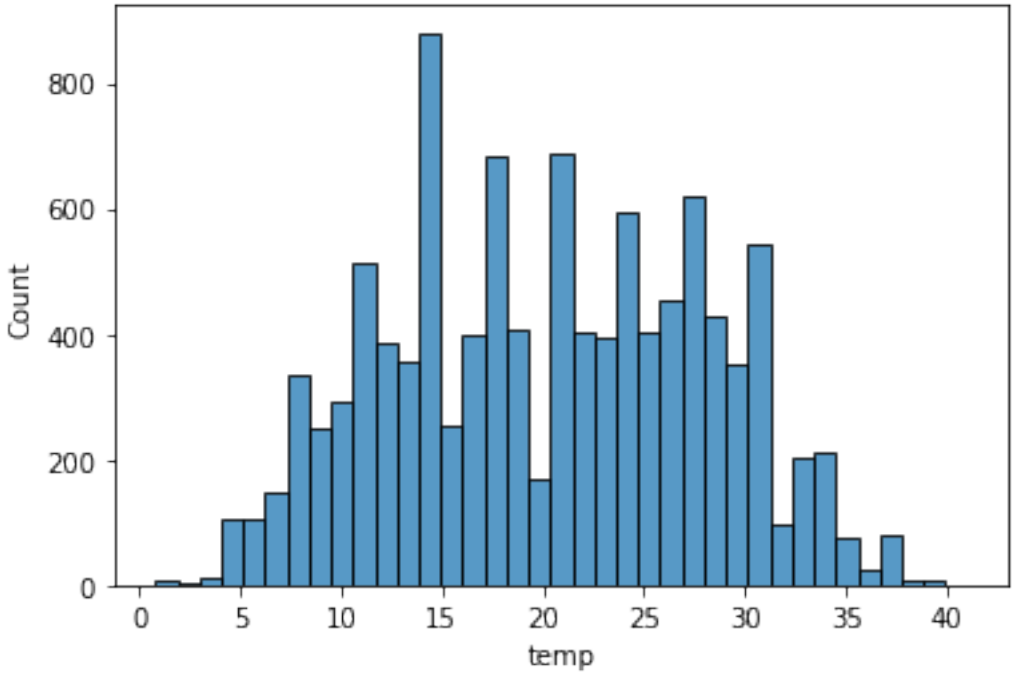
```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\1194791850.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df["count"])
```

[18]: <AxesSubplot: xlabel='count', ylabel='Density'>



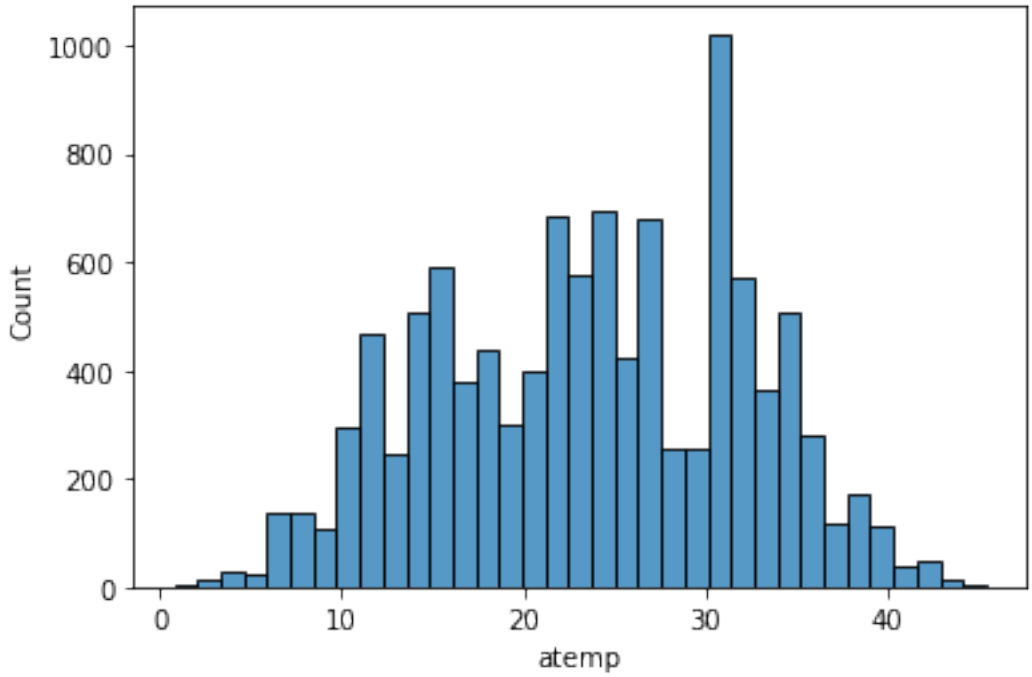
[19]:
```
#histplot for temp attribute
sns.histplot(df["temp"])
```

[19]: <AxesSubplot: xlabel='temp', ylabel='Count'>



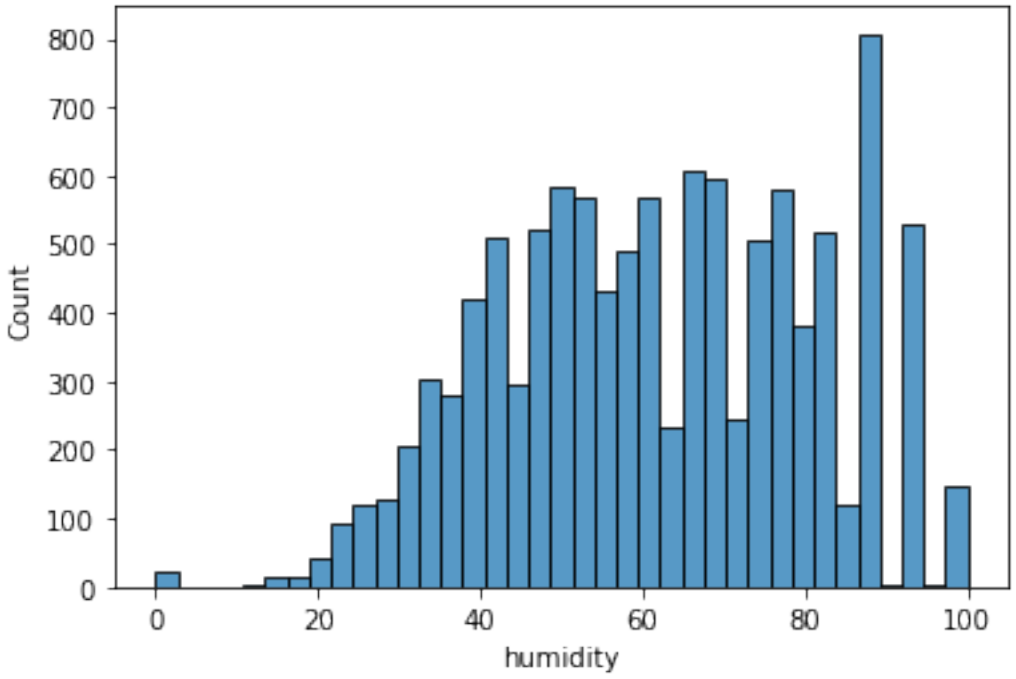
[20]:
```
#histplot for atemp attribute
sns.histplot(df["atemp"])
```

[20]: <AxesSubplot: xlabel='atemp', ylabel='Count'>



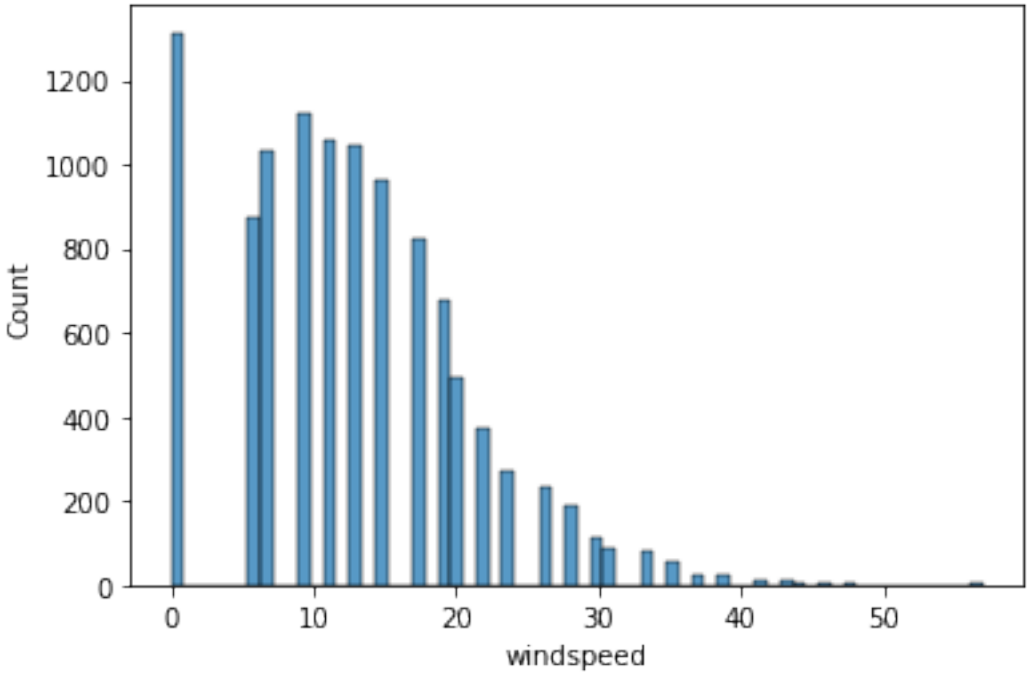
[21]:
```
#histplot for humidity attribute
sns.histplot(df["humidity"])
```

[21]: <AxesSubplot: xlabel='humidity', ylabel='Count'>



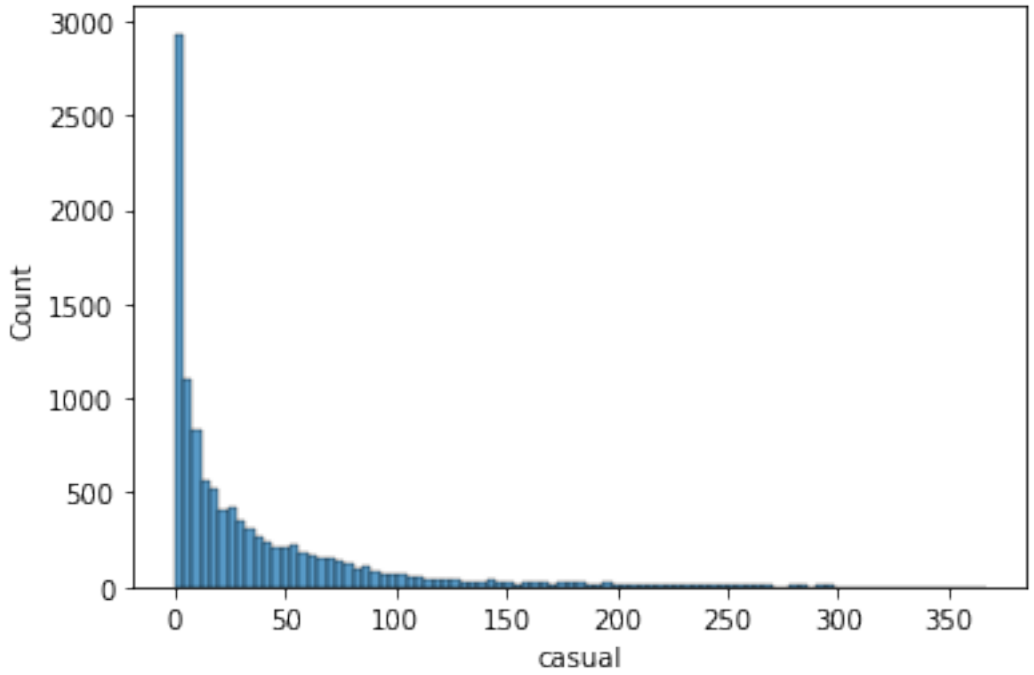
[22]:
```
#histplot for windspeed attribute
sns.histplot(df["windspeed"])
```

[22]: <AxesSubplot: xlabel='windspeed', ylabel='Count'>



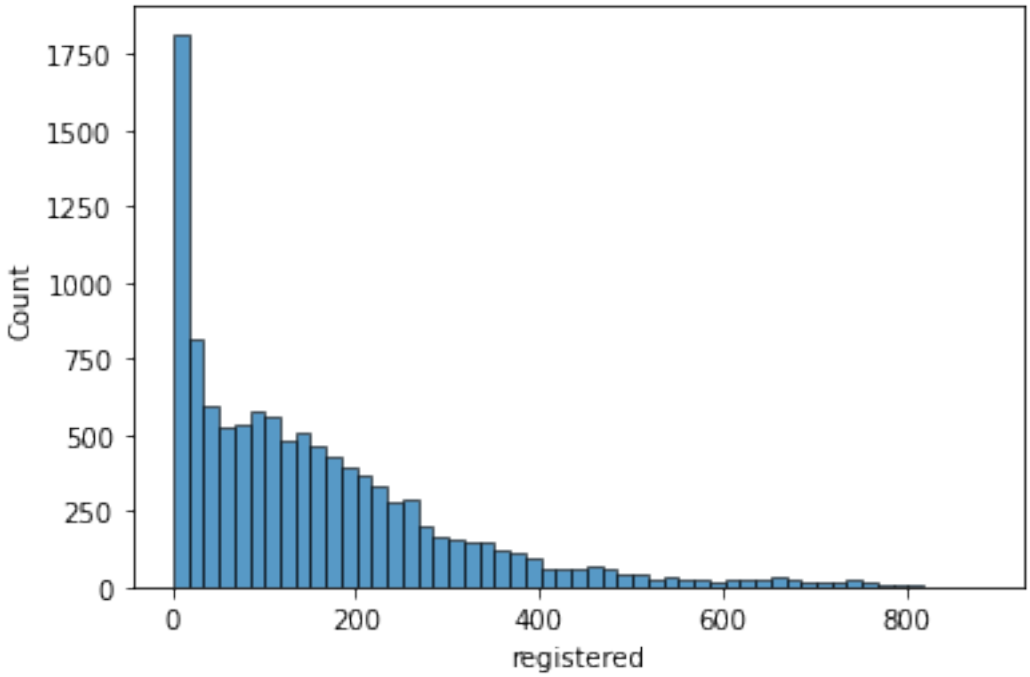
[23]:
```
#histplot for count of casual users
sns.histplot(df["casual"])
```

[23]: <AxesSubplot: xlabel='casual', ylabel='Count'>



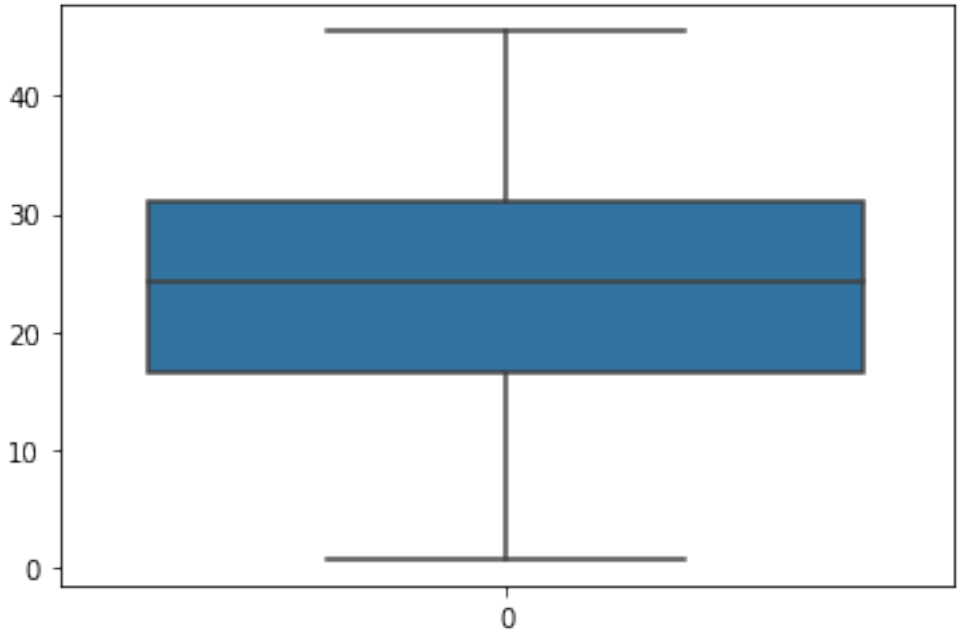
[24]:
```
#histplot for count of registered users
sns.histplot(df["registered"])
```

[24]: <AxesSubplot: xlabel='registered', ylabel='Count'>



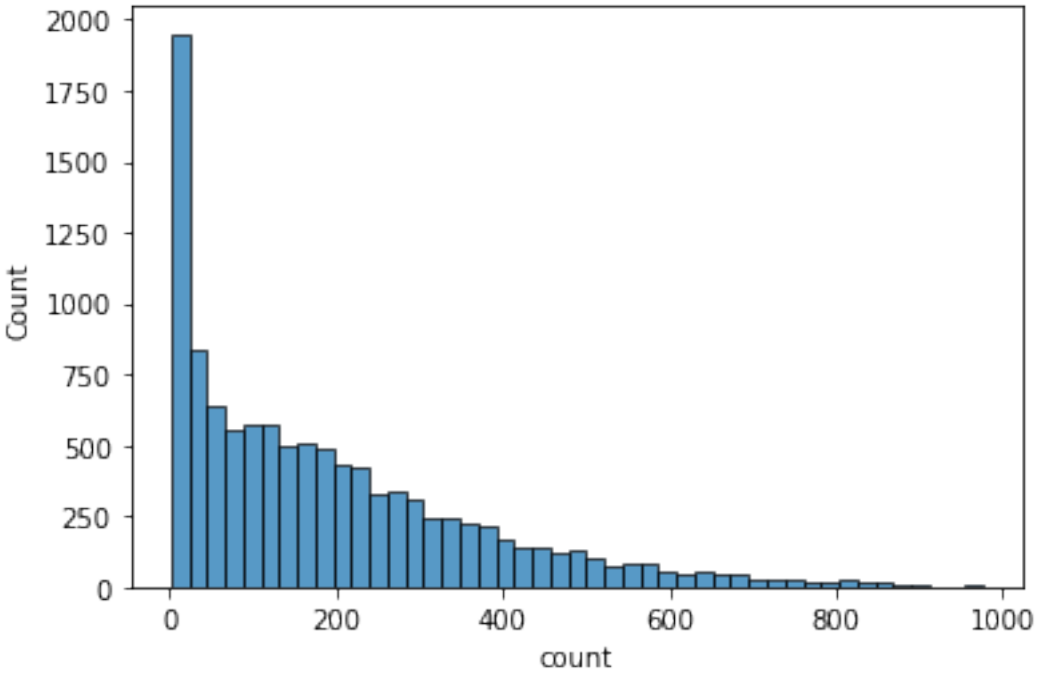
[27]:
```
#boxplot for atemp attribute
sns.boxplot(df["atemp"])
```

[27]: <AxesSubplot: >



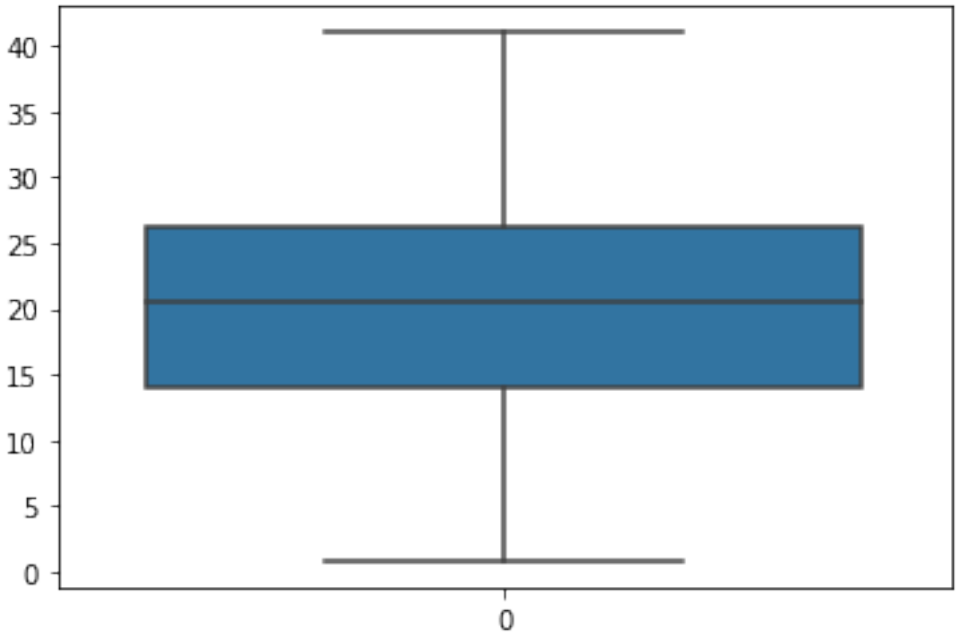
[25]:
```
#histplot for count of total rental bikes including both registered and casual
sns.histplot(df["count"])
```

[25]: <AxesSubplot: xlabel='count', ylabel='Count'>



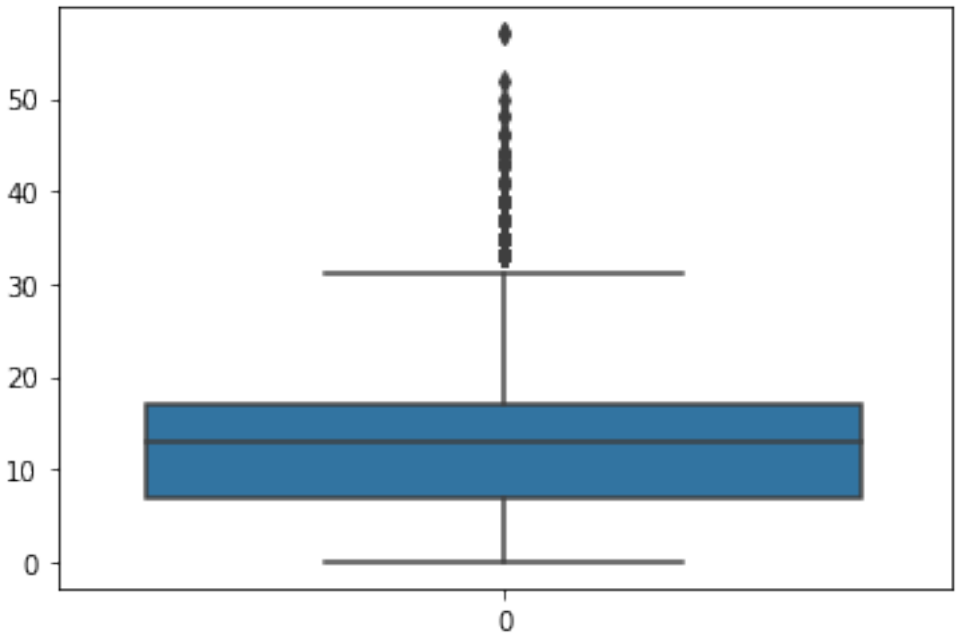
[26]:
```
#boxplot for temp attribute
sns.boxplot(df["temp"])
```

[26]: <AxesSubplot: >



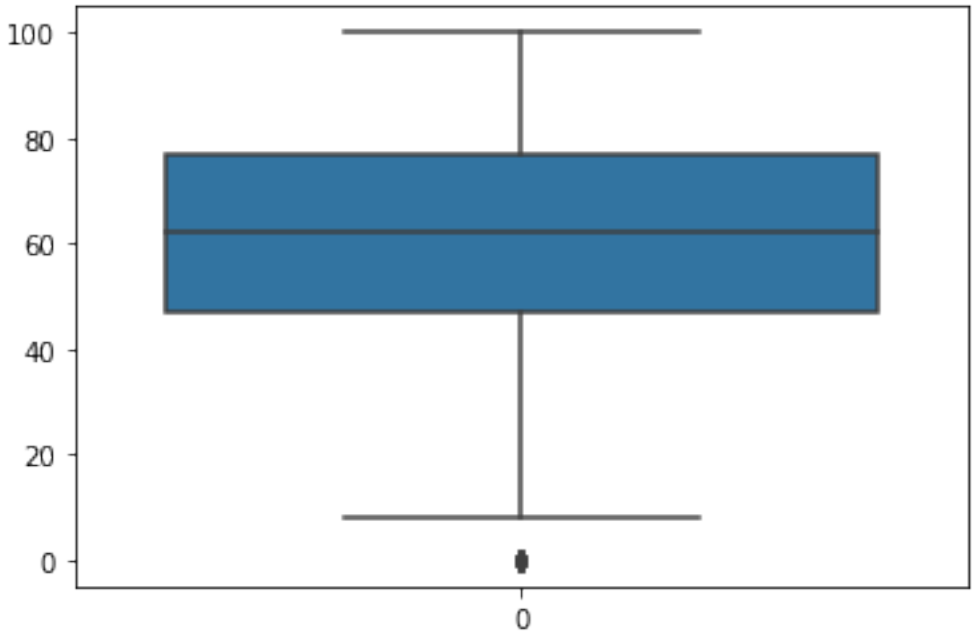
[28]:
```
#Boxplot for windspeed attribute
sns.boxplot(df["windspeed"])
```

[28]: <AxesSubplot: >



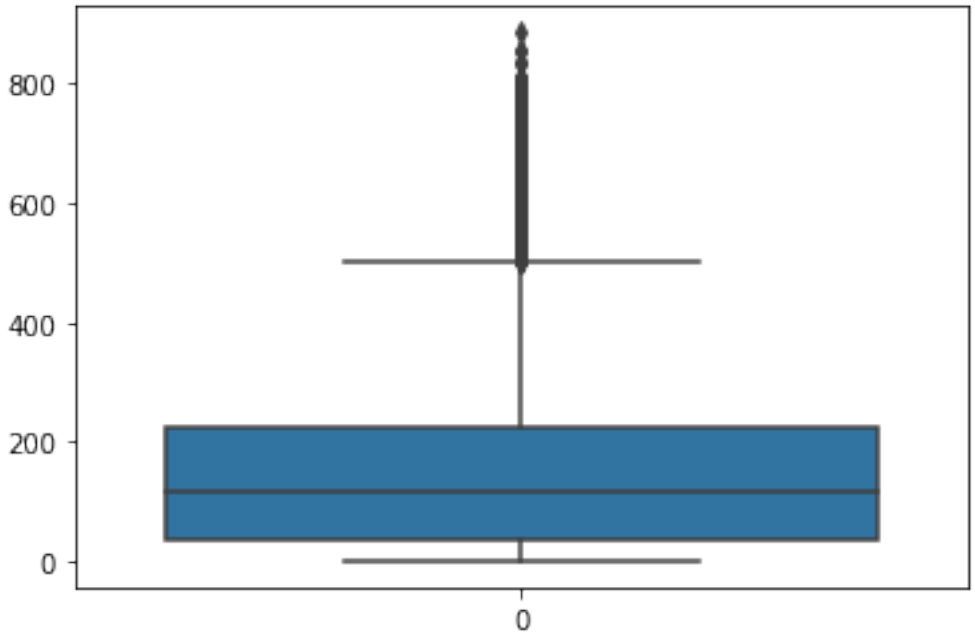
[29]:
```
#boxplot for humidity attribute
sns.boxplot(df["humidity"])
```

[29]: <AxesSubplot: >



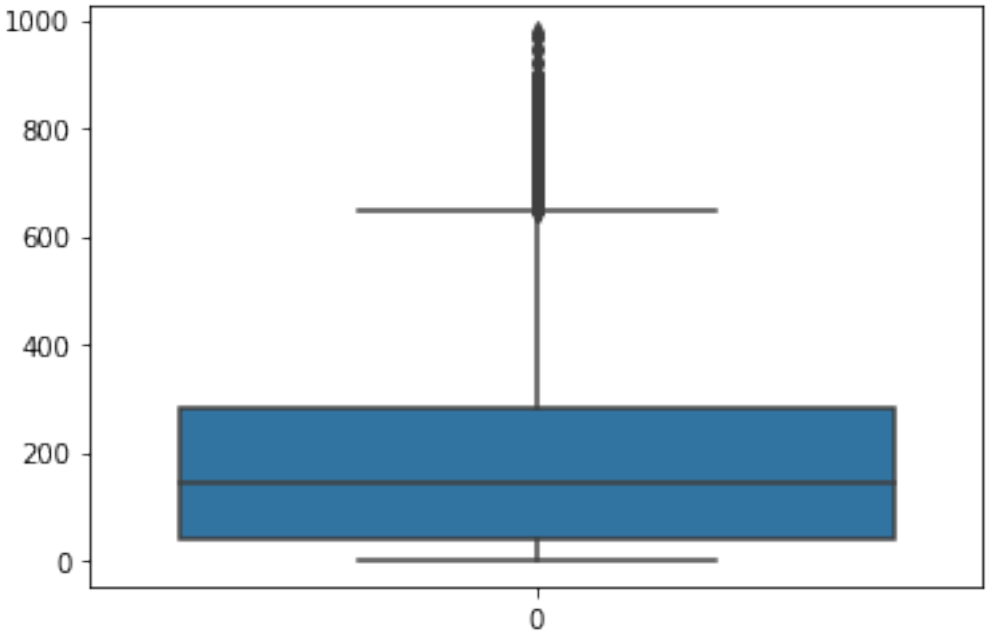
[30]:
```
#boxplot for count of registered users
sns.boxplot(df["registered"])
```

[30]: <AxesSubplot: >



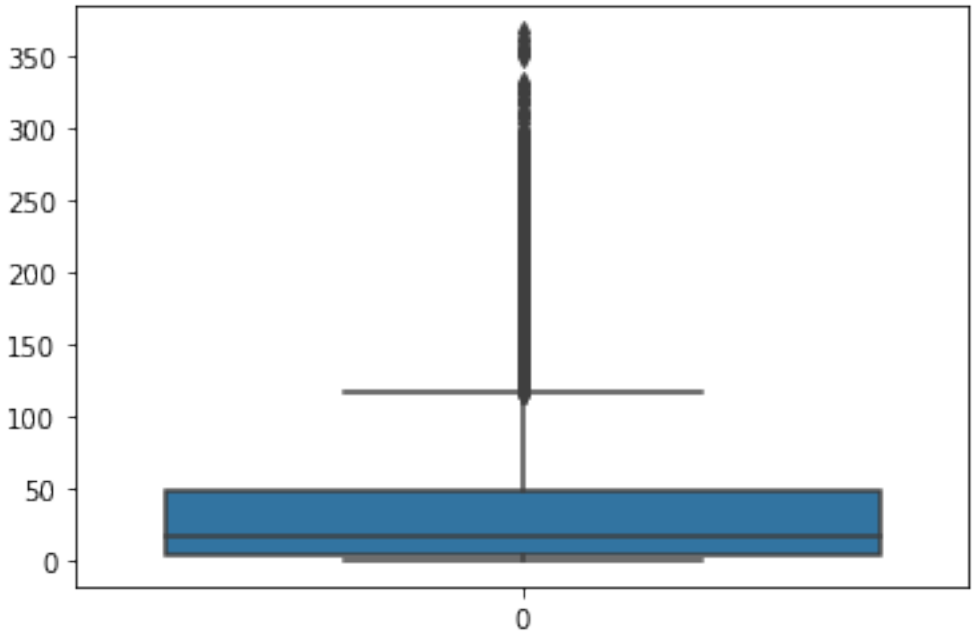
[32]:
```
#boxplot for count of total rental bikes including both casual and registered␣
↪users
sns.boxplot(df["count"])
```

[32]: <AxesSubplot: >



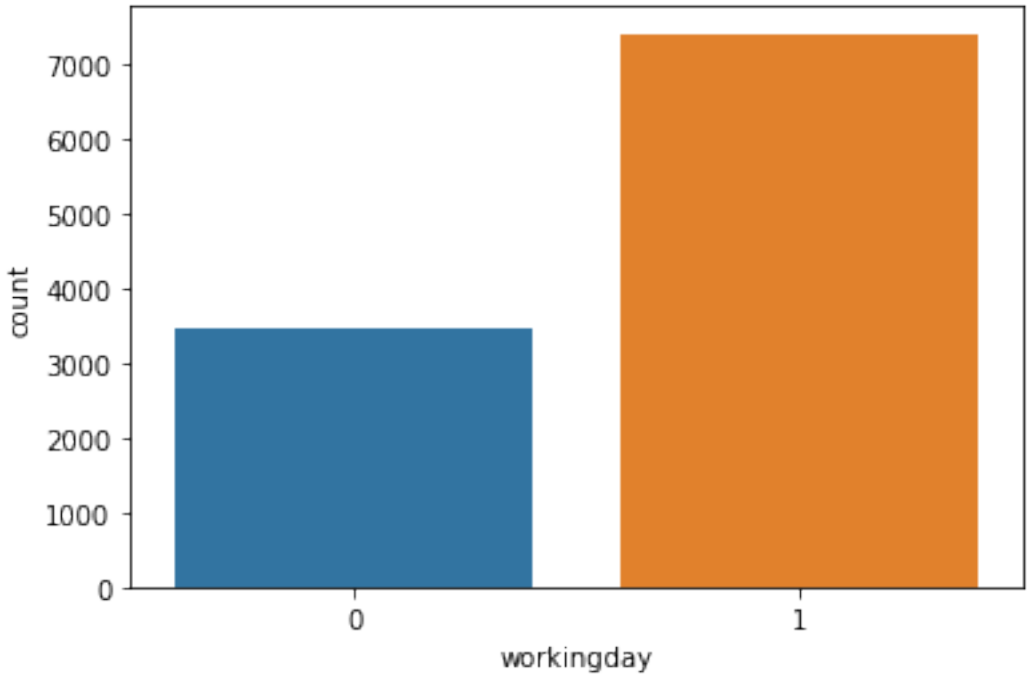
[31]:
```
#boxplot for count of casual users
sns.boxplot(df["casual"])
```

[31]: <AxesSubplot: >



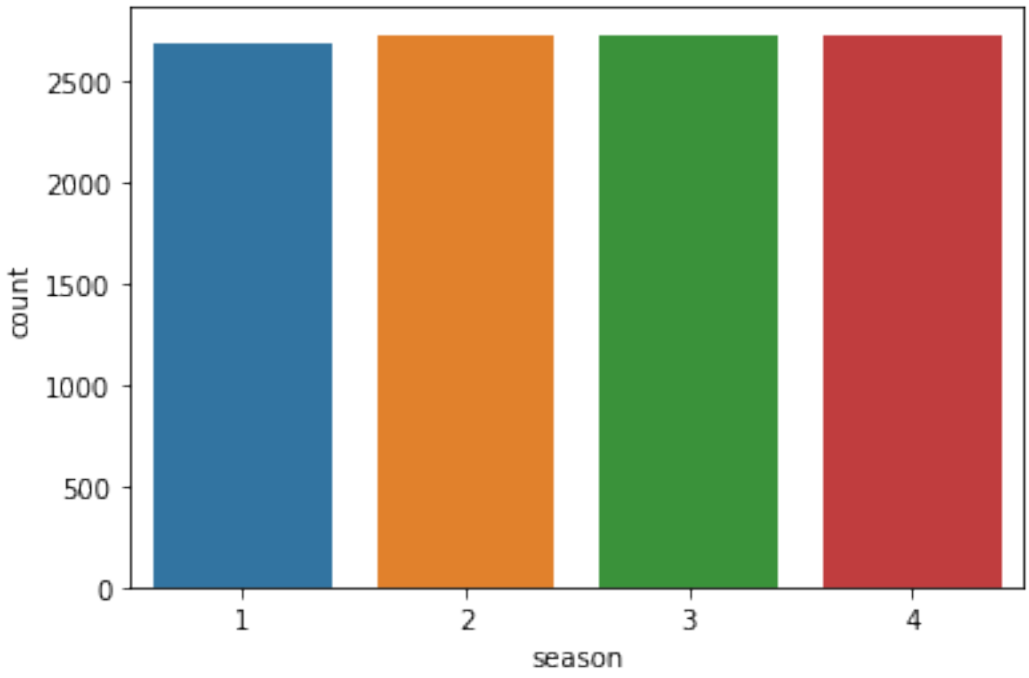
[35]:
```
#countplot for working days and non working days
sns.countplot(data=df,x="workingday")
```

[35]: <AxesSubplot: xlabel='workingday', ylabel='count'>



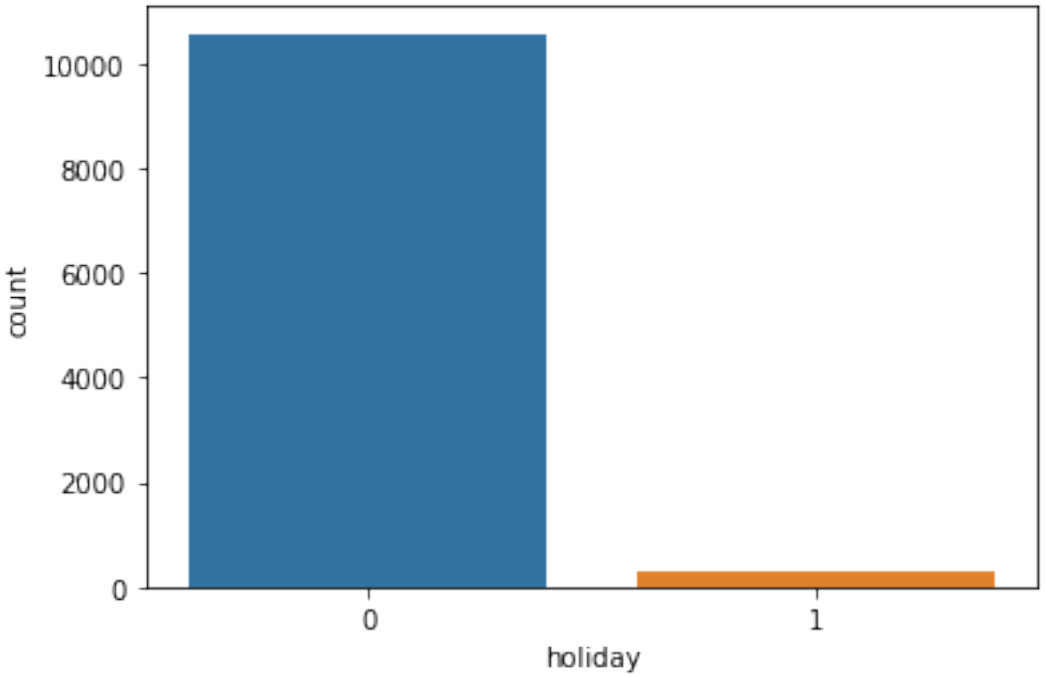
[33]:
```
#Countplot for seasons
sns.countplot(data=df,x="season")
```

[33]: <AxesSubplot: xlabel='season', ylabel='count'>



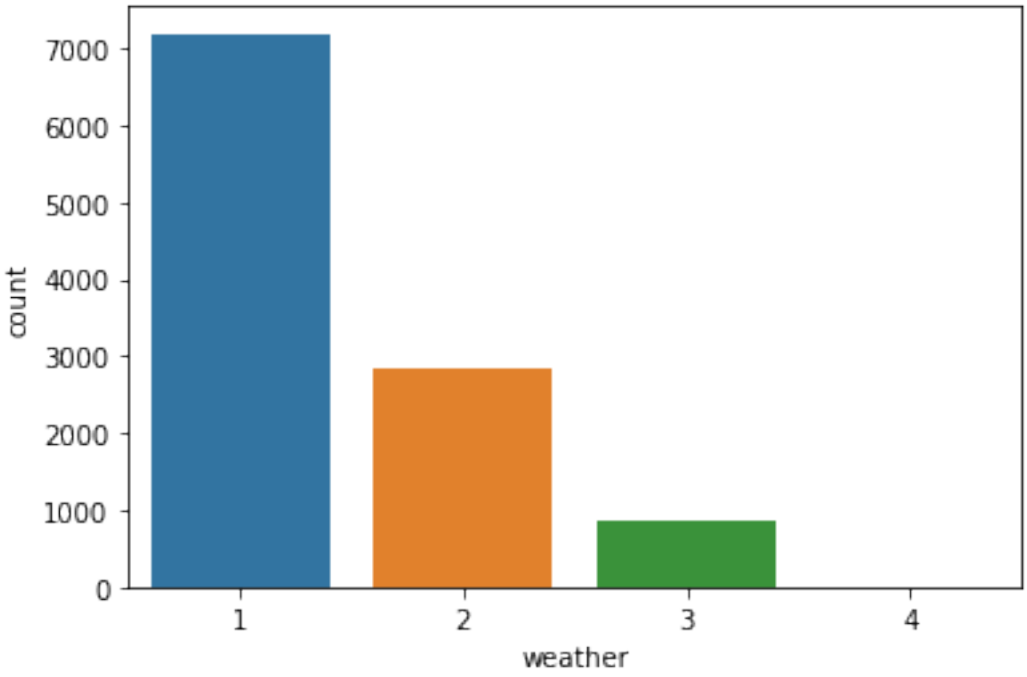
[34]:
```
#countplot for holidays categories
sns.countplot(data=df, x="holiday")
```

[34]: <AxesSubplot: xlabel='holiday', ylabel='count'>



[36]:
```
#countplot for weather types
sns.countplot(data=df,x="weather")
```

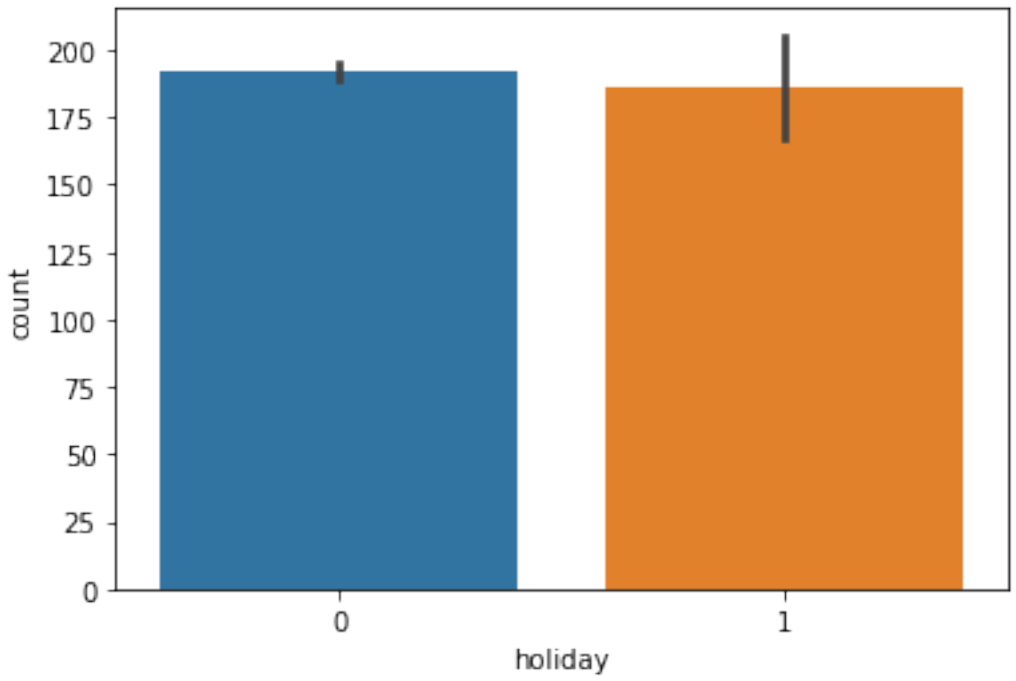
[36]: <AxesSubplot: xlabel='weather', ylabel='count'>



4 Bivariate Analysis

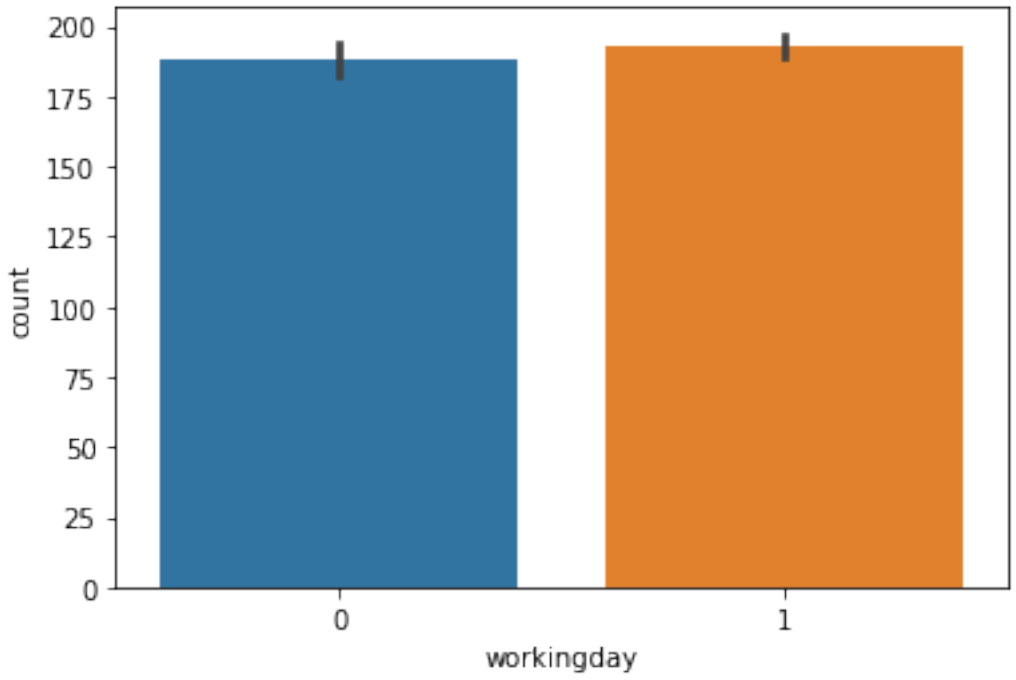
[38]:
```
#barplot between workingday and count to understand business based on holiday
sns.barplot(data=df, x="holiday", y="count")
```

[38]:
```
<AxesSubplot: xlabel='holiday', ylabel='count'>
```



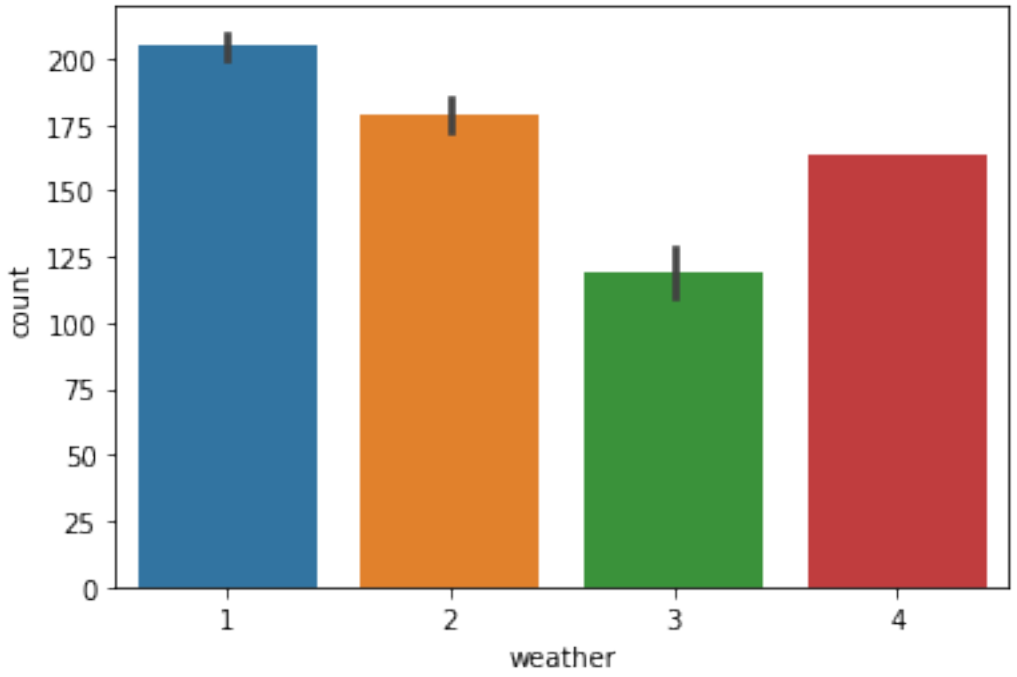
[37]:
```
#barplot between workingday and count to understand business based on workingday
sns.barplot(data=df, x="workingday", y="count")
```

[37]: <AxesSubplot: xlabel='workingday', ylabel='count'>



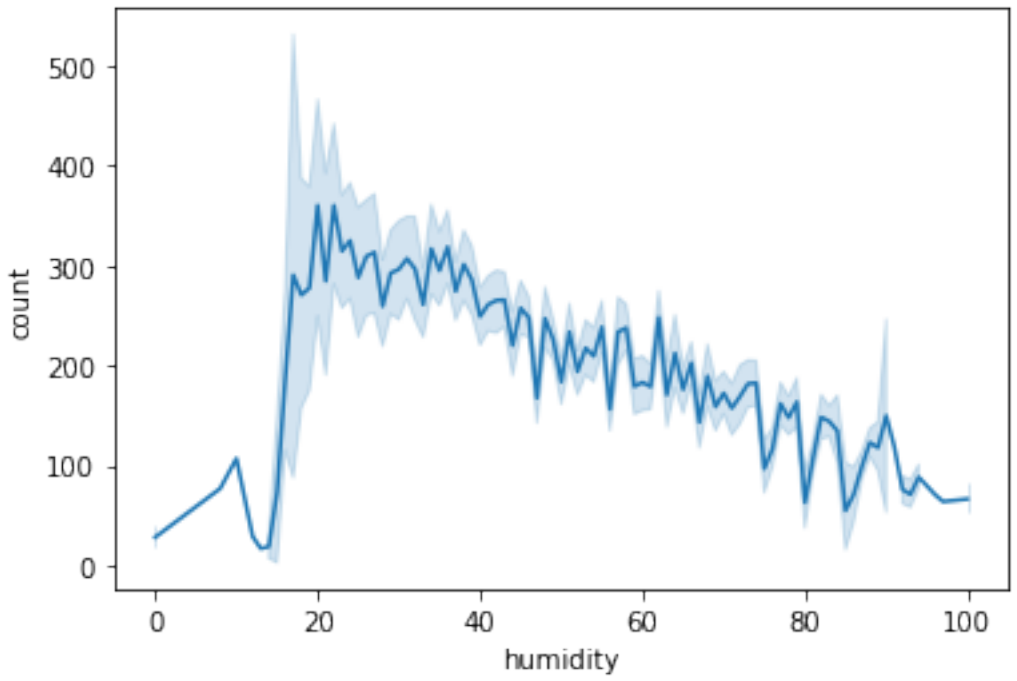
[40]:
```
#barplot between weather and count to understand business based on weather
sns.barplot(df, x="weather",y="count")
```

[40]: <AxesSubplot: xlabel='weather', ylabel='count'>



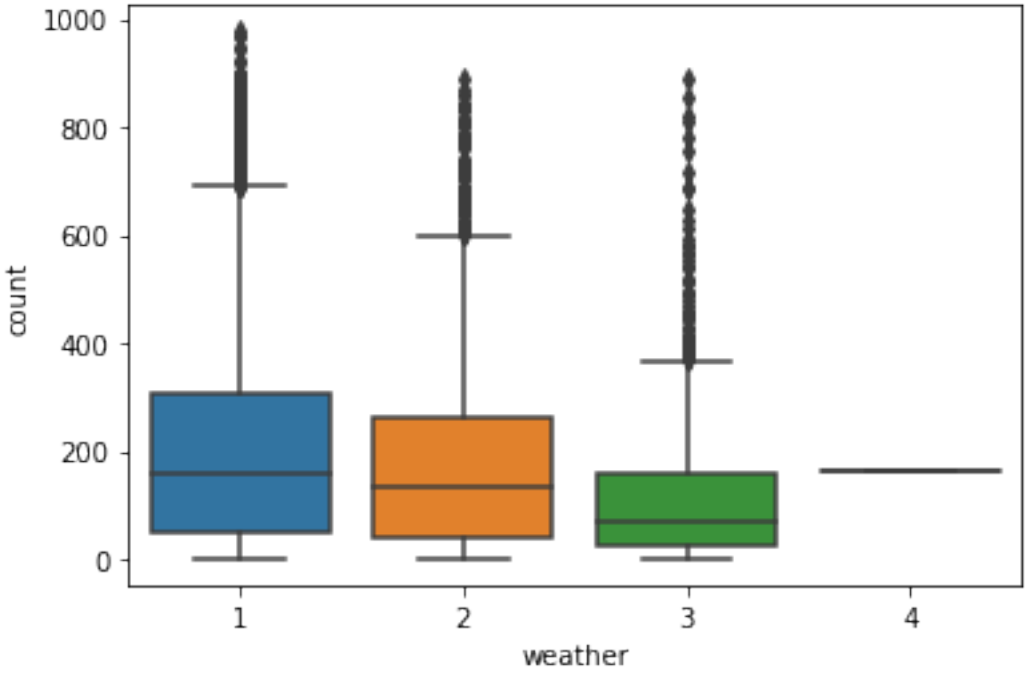
[41]:
```
#Lineplot between humidity and count
sns.lineplot(df, x="humidity",y="count")
```

[41]: <AxesSubplot: xlabel='humidity', ylabel='count'>



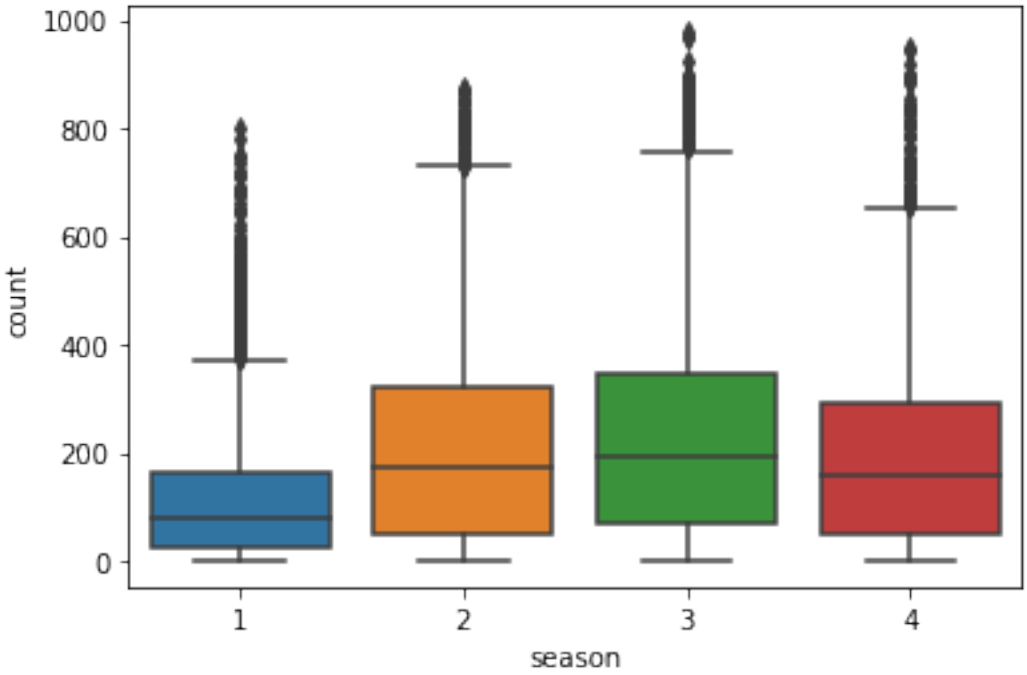
[42]:
```
#boxplot between weather and count to understand business based on weather
sns.boxplot(df, x="weather", y="count")
```

[42]: <AxesSubplot: xlabel='weather', ylabel='count'>



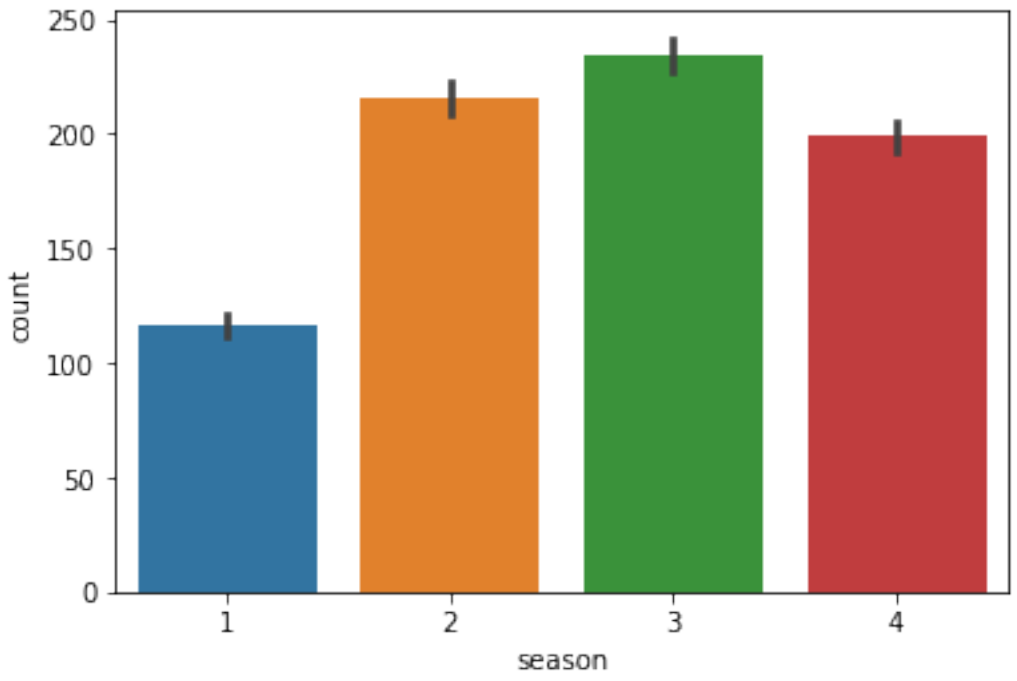
[43]:
```
#boxplot between season and count to understand business based on season
sns.boxplot(df,x="season", y="count")
```

[43]: <AxesSubplot: xlabel='season', ylabel='count'>



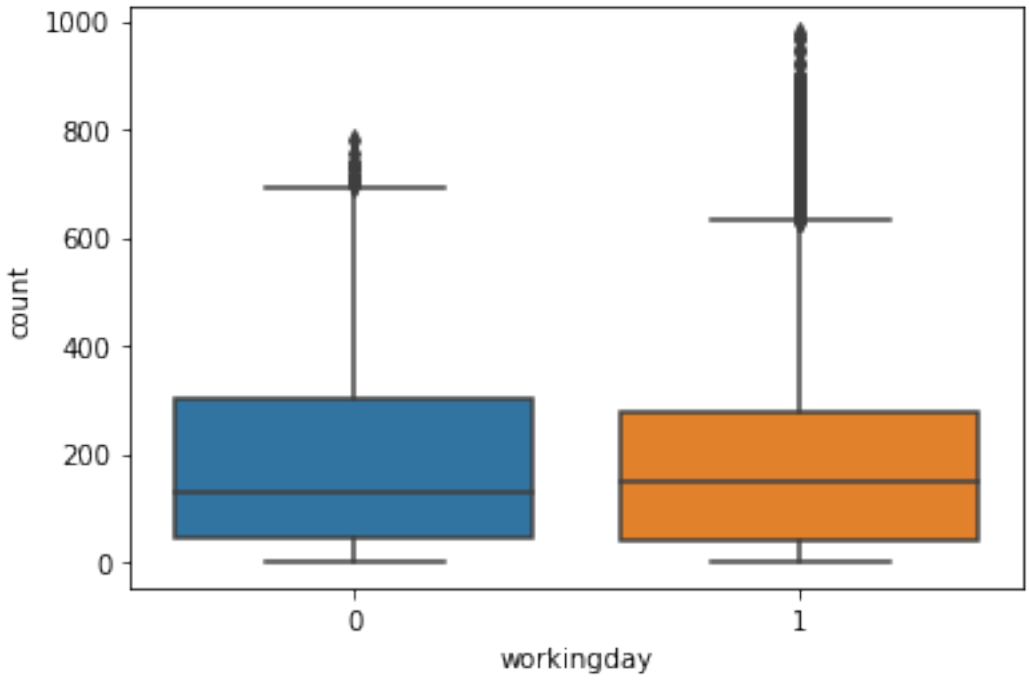
[39]:
```
#barplot between season and count to understand business based on season
sns.barplot(data=df, x="season", y="count")
```

[39]: <AxesSubplot: xlabel='season', ylabel='count'>



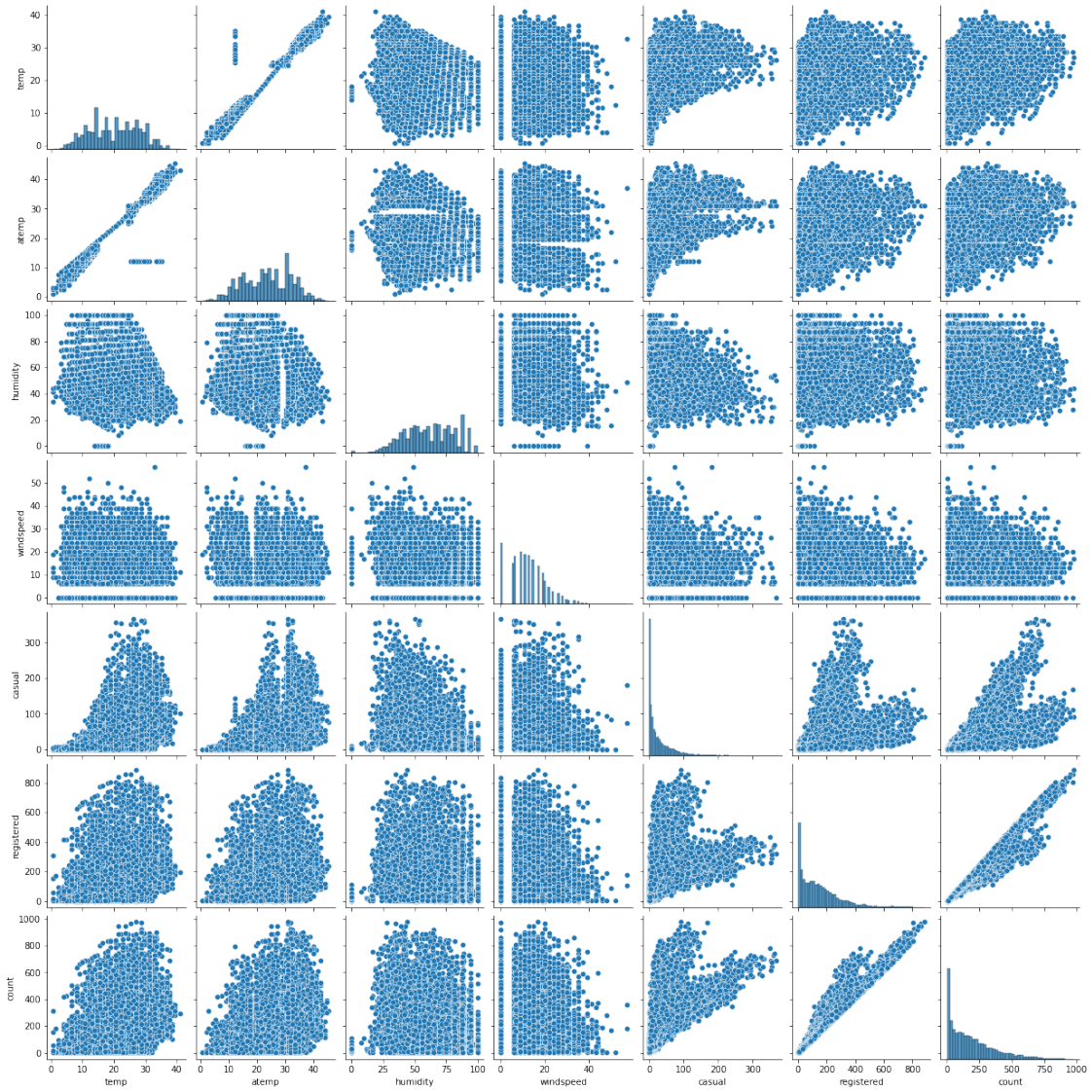
[44]:
```
#boxplot between workingday and count to understand business based on workingday
sns.boxplot(df,x="workingday", y="count")
```

[44]: <AxesSubplot: xlabel='workingday', ylabel='count'>



[45]:
```
#Pairplots in the dataframes having numeric datatype
sns.pairplot(df.loc[:,"temp":])
```

[45]: <seaborn.axisgrid.PairGrid at 0x2853e109db0>



[46]:
```
#Correlation between different attributes of dataframe
df.corr()
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\2466234702.py:2:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
```

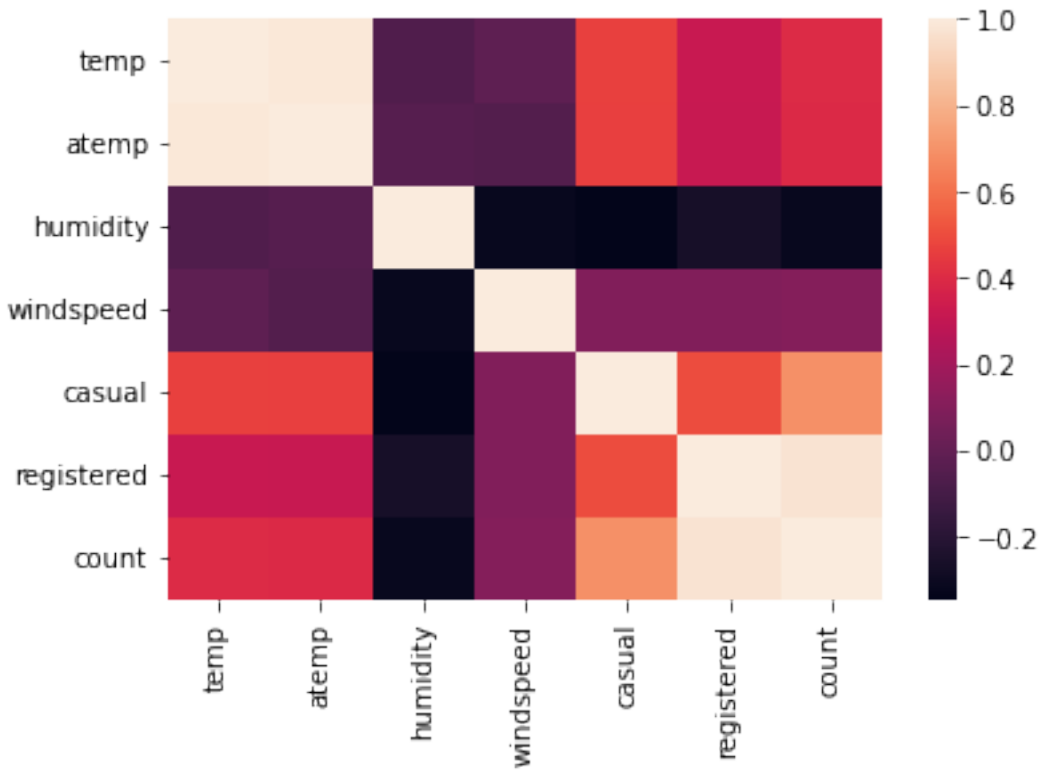
```
columns or specify the value of numeric_only to silence this warning.
  df.corr()
```

[46]:
```
                temp     atemp  humidity  windspeed    casual  registered  \
temp        1.000000  0.984948 -0.064949  -0.017852  0.467097    0.318571
atemp       0.984948  1.000000 -0.043536  -0.057473  0.462067    0.314635
humidity   -0.064949 -0.043536  1.000000  -0.318607 -0.348187   -0.265458
windspeed  -0.017852 -0.057473 -0.318607   1.000000  0.092276    0.091052
casual      0.467097  0.462067 -0.348187   0.092276  1.000000    0.497250
registered  0.318571  0.314635 -0.265458   0.091052  0.497250    1.000000
count       0.394454  0.389784 -0.317371   0.101369  0.690414    0.970948

                count
temp        0.394454
atemp       0.389784
humidity   -0.317371
windspeed   0.101369
casual      0.690414
registered  0.970948
count       1.000000
```

[47]:
```
# Heatmap based on correlation between attributes
sns.heatmap(data=df.corr())
```

```
C:\Users\Pipaliya\AppData\Local\Temp\ipykernel_16072\4070634115.py:2:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data=df.corr())
```

[47]:
```
<AxesSubplot: >
```



4.1 Dataset Information

datetime: It contains the date and time respondence to the given data. It ranges from 2011-01-01 to 2012-12-19.

season: It contains 4 values of the season that are spring, summer, fall, and winter

holiday: It gives whether a given day is a holiday or not.

working day: It gives whether the given day is a working day or a holiday or weekend

weather: It contains 4 different masked categories of weather

temp: It gives the temperature in Celsius at that moment, and its value ranges from 0.82 to 41.00.

atemp: It gives the feeling temperature in Celsius at that moment, and its value ranges from 0.76 to 45.45.

humidity: It gives the humidity at a given time, and its value ranges from 0.0 to 100.00

windspeed: It gives the values of windspeed at a given time, and its value ranges from 0.0 to 56.99

casual: It gives a count of casual users at a given time, and its value ranges from 0 to 367

registered: It gives a count of casual users at a given time, and its value ranges from 0 to 886

count: It gives a count of total rental bikes including both casual and registered, and its value ranges from 1 to 977.

Here Outliers are found by IQR method in casual, registered, and count columns, but as dropping or morphing of outliers may affect our statistical significance, so Its better to keep them in our data.

5 2 Sample T-Test

[48]:
```
#Filtering count based on working day
working_day_count= df.loc[df["workingday"]==1,"count"]
non_working_day_count=df.loc[df["workingday"]==0,"count"]
```

[49]:
```
#Mean and Standard Deviation of count during working day
working_day_count.mean(), working_day_count.std()
```

[49]: (193.01187263896384, 184.5136590421481)

[50]:
```
#Mean and Standard Deviation of count during Non-working day
non_working_day_count.mean(), non_working_day_count.std()
```

[50]: (188.50662061024755, 173.7240153250003)

Ho : mean of working day and non working day is same : mu1 = mu2

Ha : mean of working day is higher than non working day : mu1 > mu2

[51]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[52]:
```
#Let we do t-test for 2 samples and find test_statistics and p-value
test_statistic, p_value = ttest_ind(working_day_count,non_working_day_count,␣
 ↪alternative="greater")
test_statistic, p_value
```

[52]: (1.2096277376026694, 0.11322402113180674)

[53]:
```
#Decision based on p-value and significance level
if p_value < alpha:
    print("Reject Null Hypothesis Ho")
else:
    print("Fail to Reject Null Hypothesis Ho")
```

```
Fail to Reject Null Hypothesis Ho
```

We have considered a confidence level of 95% in the Test.

The 2 Sample T-Test between the count attributes of the working day and the non-working day has been carried out and We found from the 2 Sample T-test that the means of both samples have no statistically significant difference.

5.1 ANOVA Test

[54]:
```
#Filtering count based on weather category
weather_1 = df.loc[df["weather"]==1,"count"]
weather_2 = df.loc[df["weather"]==2,"count"]
weather_3 = df.loc[df["weather"]==3,"count"]
weather_4 = df.loc[df["weather"]==4,"count"]
```

[55]:
```
weather_4 #Only single value is there with weather category 4 so, We will not␣
 ↪consider this category for ANOVA Test
```

[55]:
```
5631    164
Name: count, dtype: int64
```

We will do shapiro Test for checking whether our sample follows Gaussian Distribution or not

Null and Alternate Hypothesis for Shapiro Test

H0: The sample follows Gaussian Distribution

Ha: The sample does not follow Gaussian Distribution

[56]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[57]:
```
#p-value calculation
test_statistics, p_value = shapiro(weather_1)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

```
C:\Users\Pipaliya\AppData\Local\Programs\Python\Python310\lib\site-
packages\scipy\stats\_morestats.py:1816: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

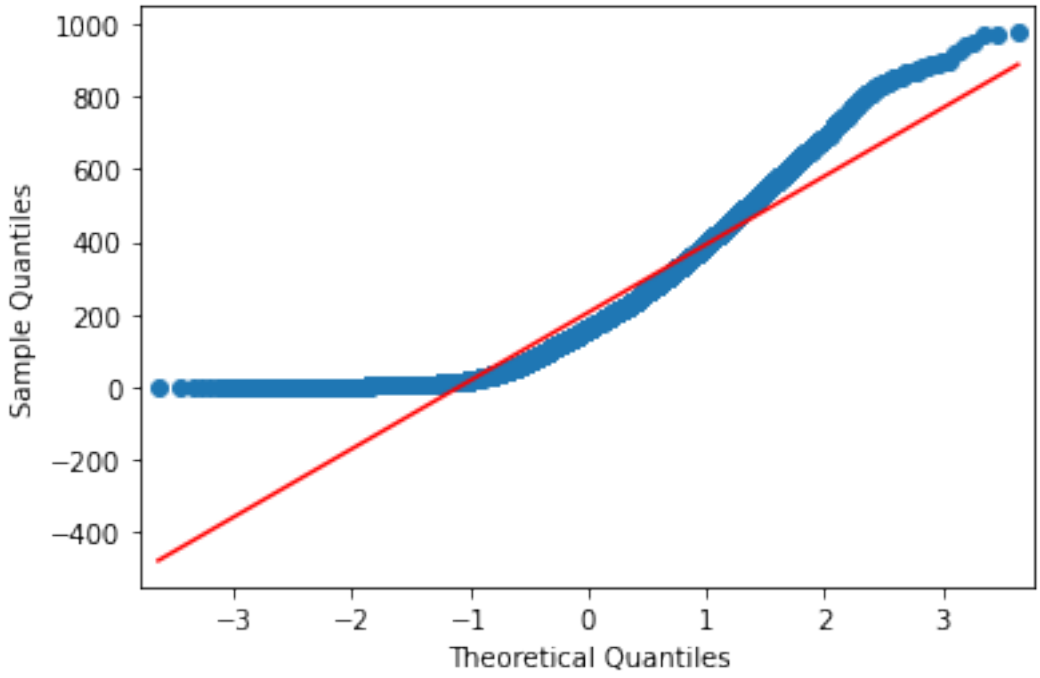
[58]:
```
#p-value calculation
test_statistics, p_value = shapiro(weather_2)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

[59]:
```
#p-value calculation
test_statistics, p_value = shapiro(weather_3)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

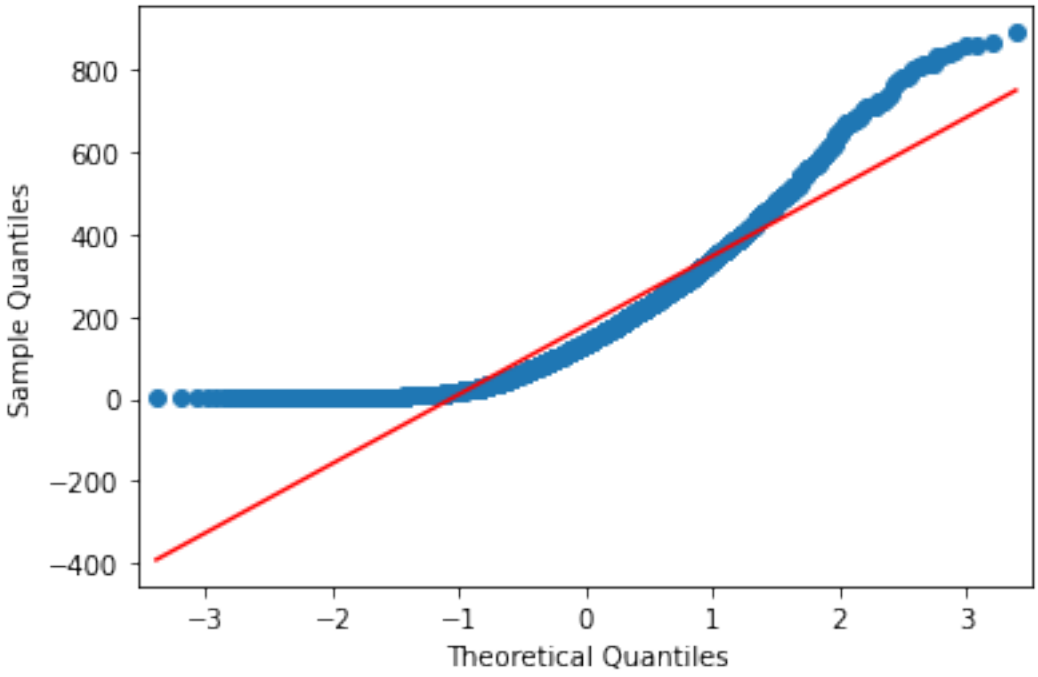
```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

[60]:
```
#Let's check for normality based on q-q plot
qqplot(weather_1,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```

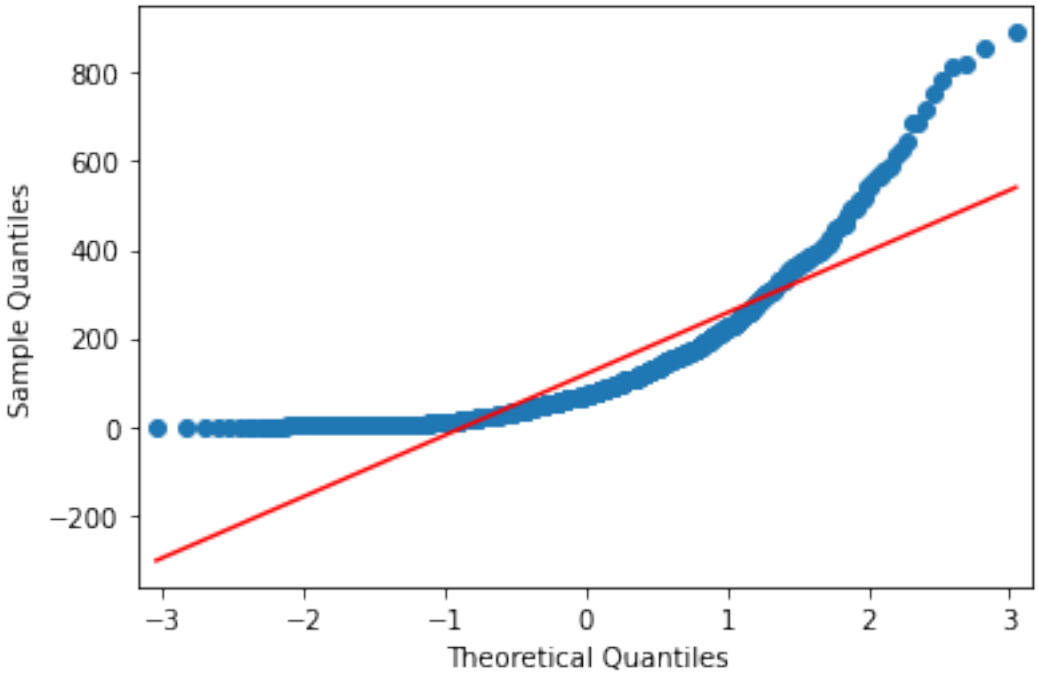


[61]:
```
#Let's check for normality based on q-q plot
qqplot(weather_2,line="s")
plt.show()
```

```
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



[62]:
```
#Let's check for normality based on q-q plot
qqplot(weather_3,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



We will do levene test to check whether variance of the samples are same or not

Null Hypothesis and Alternate Hypothesis for Levene Test

H0: Variances of the samples are same

Ha: Variances of the samples are not same

[63]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[64]:
```
#p-value calculation
test_statistics, p_value=levene(weather_1,weather_2, weather_3)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Variances of the samples are not same")
else:
    print("Fail to Reject Null Hypothesis, Variances of the samples are same")
```

```
p-value: 0.0
Reject Null Hypotheis, Variances of the samples are not same
```

As we have done shapiro and Q-Q Plot for checking Normality and Levene Test for checking Variance.

We have found that Samples do not follow Gaussian Distribution and do not have similar variance. So we will go for Kruskal-Wallis Test

Null and Alternate Hypothesis for Kruskal Wallis Test

H0: mean of total rental bikes of different weathers are same

Ha: mean of total rental bikes of different weathers are not same

[65]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[66]:
```
#p-value calculation
test_statistics,p_value=kruskal(weather_1,weather_2,weather_3)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, mean of total rental bikes of different␣
 ↪weathers are not same")
else:
    print("Fail to Reject Null Hypothesis, mean of total rental bikes of␣
 ↪different weathers are same")
```

```
p-value: 0.0
Reject Null Hypotheis, mean of total rental bikes of different weathers are not
same
```

[67]:
```
#Filtering count based on weather category
season_1 = df.loc[df["season"]==1,"count"]
season_2 = df.loc[df["season"]==2,"count"]
season_3 = df.loc[df["season"]==3,"count"]
season_4 = df.loc[df["season"]==4,"count"]
```

We will do shapiro Test for checking whether our sample follows Gaussian Distribution or not

Null and Alternate Hypothesis for Shapiro Test

H0: The sample follows Gaussian Distribution

Ha: The sample does not follow Gaussian Distribution

[68]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[69]:
```
#p-value calculation
test_statistics, p_value = shapiro(season_1)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

[70]:
```
#p-value calculation
test_statistics, p_value = shapiro(season_2)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

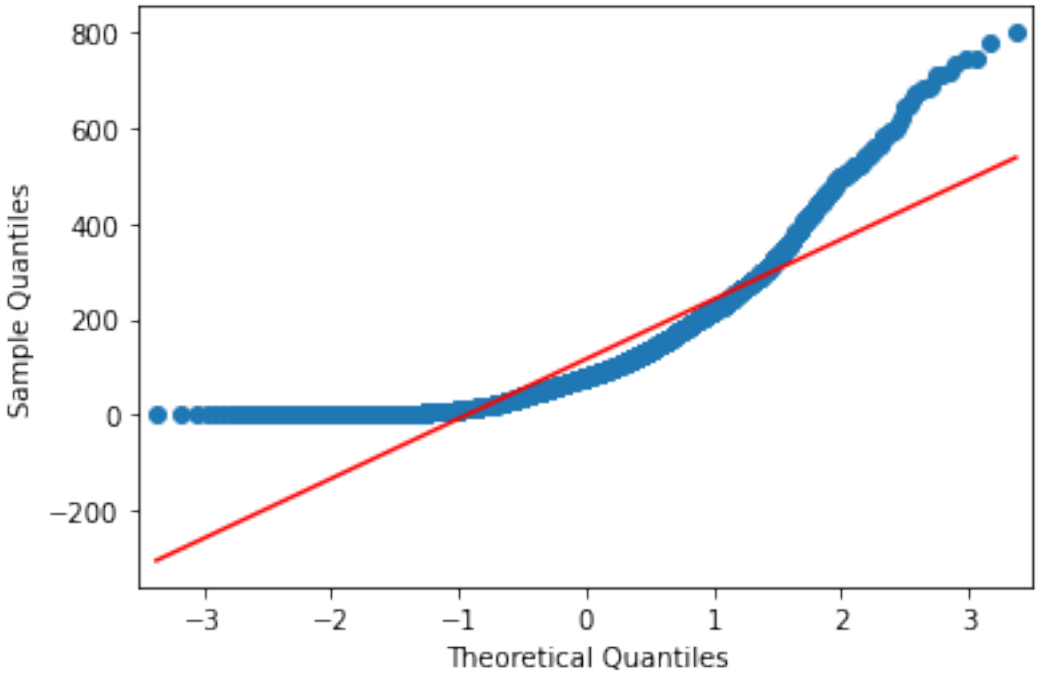
[71]:
```
#p-value calculation
test_statistics, p_value = shapiro(season_3)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

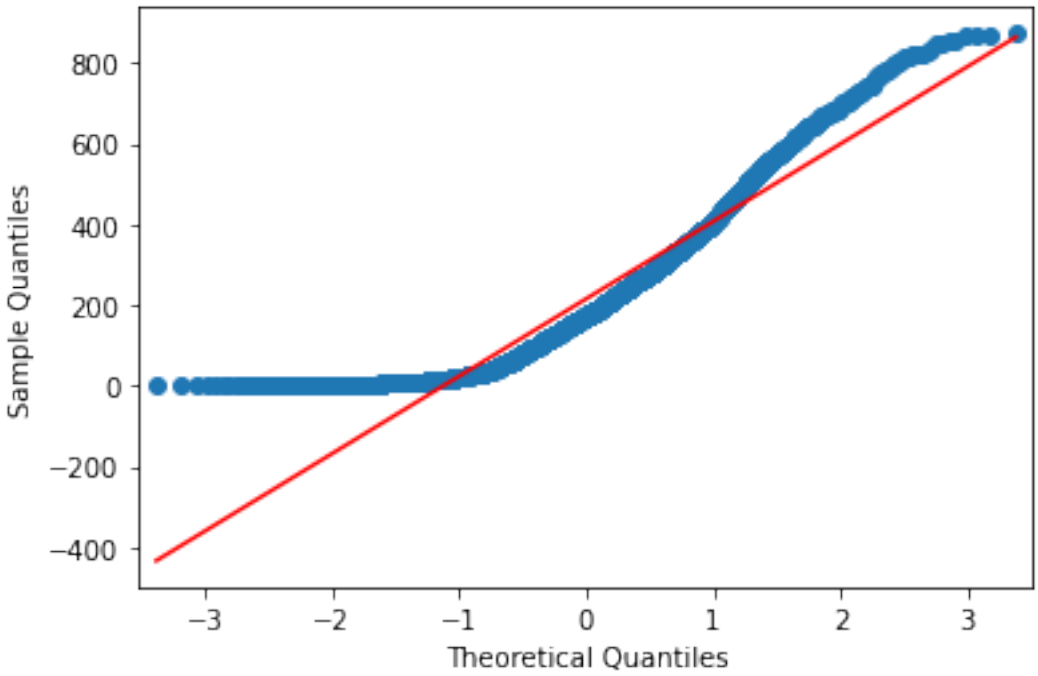
[72]:
```
#p-value calculation
test_statistics, p_value = shapiro(season_4)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Sample does not follow Gaussian Distribution")
else:
    print("Fail to Reject Null Hypothesis, Sample follows Gaussian␣
 ↪Distribution")
```

```
p-value: 0.0
Reject Null Hypotheis, Sample does not follow Gaussian Distribution
```

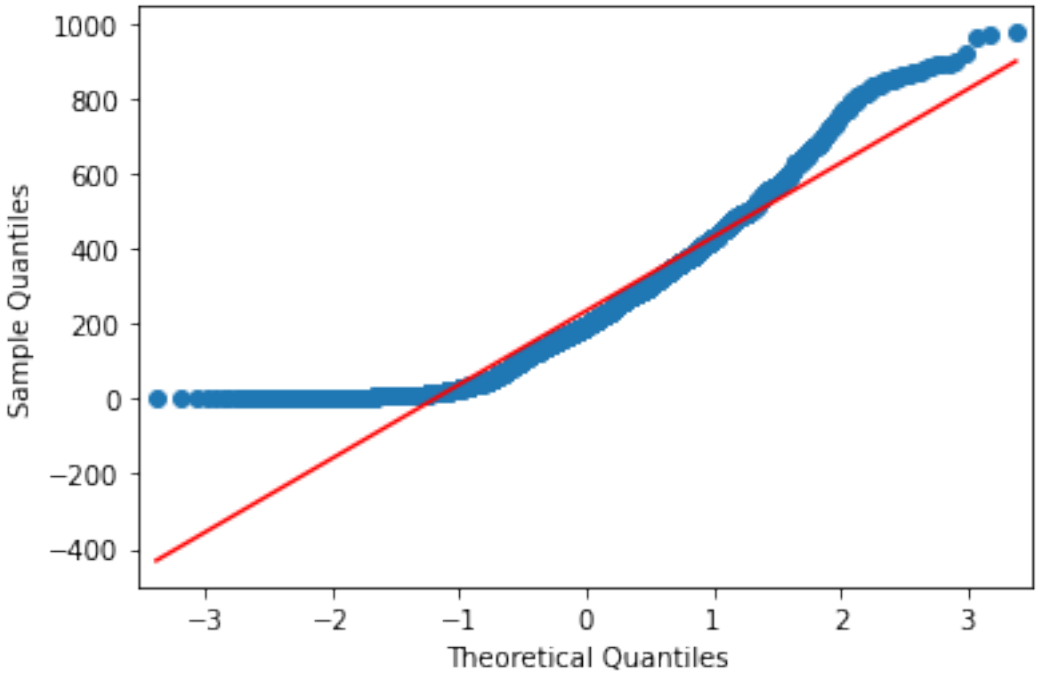
[73]:
```
#Let's check for normality based on q-q plot
qqplot(season_1,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



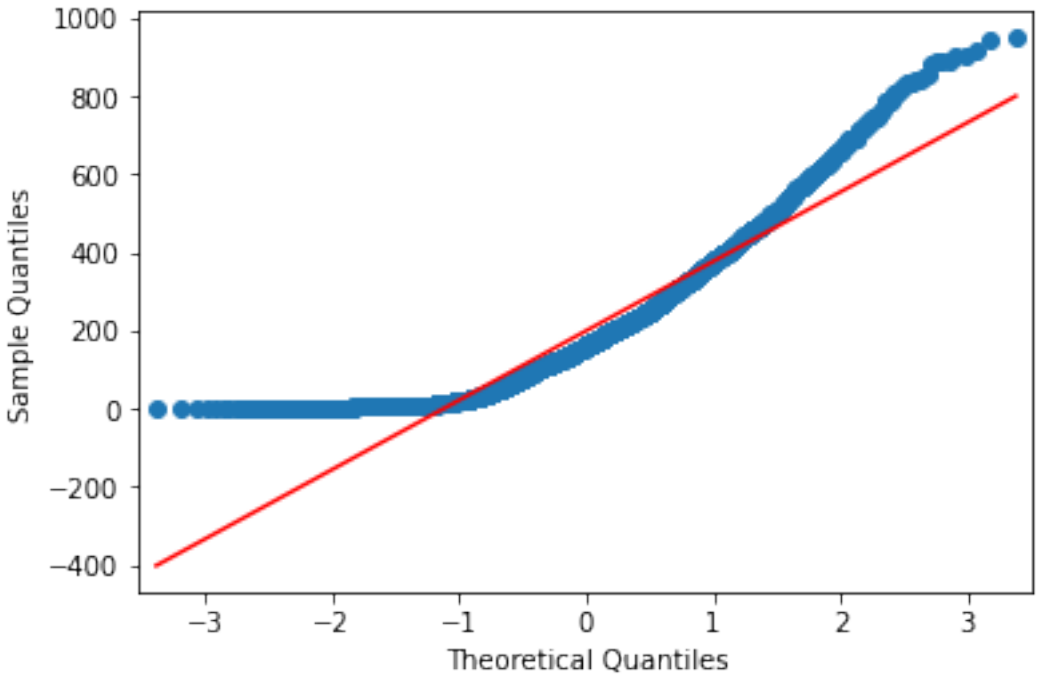
[74]:
```
#Let's check for normality based on q-q plot
qqplot(season_2,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



[75]:
```
#Let's check for normality based on q-q plot
qqplot(season_3,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



[76]:
```
#Let's check for normality based on q-q plot
qqplot(season_4,line="s")
plt.show()
#Here Plot not matching with straight line so based on that we can say that␣
 ↪sample does not follow normal distribution
```



We will do levene test to check whether variance of the samples are same or not

Null Hypothesis and Alternate Hypothesis for Levene Test

H0: Variances of the samples are same

Ha: Variances of the samples are not same

[77]:
```
#Let us set siginificance level 0.05, confidence level 95%
alpha=0.05
```

[78]:
```
#p-value calculation
test_statistics, p_value=levene(season_1, season_2, season_3, season_4)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Variances of the samples are not same")
else:
    print("Fail to Reject Null Hypothesis, Variances of the samples are same")
```

```
p-value: 0.0
Reject Null Hypotheis, Variances of the samples are not same
```

As we have done shapiro and Q-Q Plot for checking Normality and Levene Test for checking Variance.

We have found that Samples do not follow Gaussian Distribution and do not have similar variance. So we will go for Kruskal-Wallis Test

Null and Alternate Hypothesis for Kruskal Wallis Test

H0: mean of total rental bikes of different seasons are same

Ha: mean of total rental bikes of different seasons are not same

[79]:
```
# calculate the p value
test_statistics,p_value=kruskal(season_1, season_2, season_3, season_4)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, mean of total rental bikes of different␣
 ↪seasons are not same")
else:
    print("Fail to Reject Null Hypothesis, mean of total rental bikes of␣
 ↪different seasons are same")
```

```
p-value: 0.0
Reject Null Hypotheis, mean of total rental bikes of different seasons are not
same
```

We've utilized a confidence level of 95% in our analysis. Assumption tests, such as the Shapiro-Wilk test, Q-Q plot, and Levene test, were conducted in the Jupyter Notebook.

Since the samples did not meet the normality and variance assumptions, we proceeded with the Kruskal-Wallis Test.

According to the results of the Kruskal-Wallis Test, there is a statistically significant difference in the means of total rental bikes across different weather conditions and seasons.

6 Chi-square Test

[80]:
```
# Creating Contingency table between categorical attributes weather and season
ws= pd.crosstab(df["weather"], df["season"])
ws
```

[80]:

season	1	2	3	4
weather				
1	1759	1801	1930	1702
2	715	708	604	807
3	211	224	199	225
4	1	0	0	0

[81]:
```
# Here in our contingency table there is value count of 1 and 0 for weather␣
 ↪type 4
# We cant do chi-square test as minimum frequency to run chi-square test is 5
ws.loc[1:3,:]
```

[81]:

season	1	2	3	4
weather				

```
1         1759  1801  1930  1702
2          715   708   604   807
3          211   224   199   225
```

Here For Chi-Square Test between weather and Season

Null and Alternate Hypothesis

H0: Seasons and weather are independent

Ha: Seasons and weather are dependent on each other

[82]:
```
# Let us set siginificance level 0.05
alpha=0.05
```

[83]:
```
# calculate the p value
test_statistics,p_value, dof, exp=chi2_contingency(ws)
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Seasons and weather are dependent on each␣
 ↪other")
else:
    print("Fail to Reject Null Hypothesis, Seasons and weather are independent")
```

```
p-value: 0.0
Reject Null Hypotheis, Seasons and weather are dependent on each other
```

[84]:
```
#calculate the p value
test_statistics,p_value, dof, exp=chi2_contingency(ws.loc[1:3,:])
print("p-value:", round(p_value,4))
if p_value < alpha:
    print("Reject Null Hypotheis, Seasons and weather are dependent on each␣
 ↪other")
else:
    print("Fail to Reject Null Hypothesis, Seasons and weather are independent")
```

```
p-value: 0.0
Reject Null Hypotheis, Seasons and weather are dependent on each other
```

We have considered a confidence level of 95% in the Test.

From the Chi-Square Test, We can say that weather and season are depended on each other.

6.1 Business Insights

Weathers and seasons are dependent on each other.

Total rental bikes are dependent on the weather. The mean value for the total rental bikes for the weather 1st category is high compared to others.

Total rental bikes are also dependent on the seasons. the mean value for total rental bikes for fall is higher compared to other, during spring there is the lowest number of users.

There is no statistical difference in the mean of the total rental bikes on working days and non-working days

Most days in the city are of 1st category weather.

Temperature and total rental bikes are correlated and humidity and total rental bikes are negatively correlated.

6.2 Recommendations

During spring, Yulu should provide some discounts and offers to increase the use of rental bikes.

During weather of rain, The mean of total rental bikes is lower than others. As Yulu provides bike services, customers can't use it in rainy times. so Yulu should provide some roofs or cab services during this weather.

As humidity increases the total number of rental bikes decreases, so, Yulu should provide benefits during these humid days.

Yulu can increase the use of rental bikes by providing some city tour offers, events, or campaigns during non-working days.

As mostly there is clear weather, Yulu should focus on the increase in total rental bikes during clear weather days.