2kxkrk6zk

March 24, 2024

##Problem Statement

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.

Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!

Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

[30]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import math
import random
import datetime as dt
import calendar
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm
from scipy.stats import binom
from scipy.stats import poisson
from scipy.stats import expon
from scipy.stats import boxcox
from scipy.stats import geom
import scipy.stats as stats
from scipy.stats import zscore
from scipy.stats import ttest_1samp
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel
from statsmodels.stats import weightstats as stests
import statsmodels.api as sm
```

```
from scipy.stats import chisquare
from scipy.stats import chi2
from scipy.stats import chi2_contingency
from scipy.stats import f_oneway
from statsmodels.graphics.gofplots import qqplot #Normality Test
from scipy.stats import shapiro #Normality Test
from scipy.stats import levene #Variance equality test
from scipy.stats import kruskal #Anova Alternative
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy.stats import pearsonr, spearmanr
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
!pip install --upgrade category_encoders
from category_encoders import TargetEncoder
from sklearn.preprocessing import MinMaxScaler
```

```
Requirement already satisfied: category_encoders in
/usr/local/lib/python3.10/dist-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.1)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2023.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.20.0->category_encoders) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn>=0.20.0->category_encoders) (3.3.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from
statsmodels>=0.9.0->category_encoders) (24.0)
```

Importing the data set

[31]:
```
df=pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089')
df.head()
```

[31]:

	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

Shape of data

[32]:
```
df.shape
```

[32]:
```
(10886, 12)
```

Total Rows

[33]:
```
len(df)
```

[33]:
```
10886
```

Checking the data info and the metrics of data

[34]:
```
print('DATA INFO\n')
print(df.info())
cat_cols=df.dtypes=='Object'
cat_cols = list(cat_cols[cat_cols].index)
print("Categories Coloumns= ", cat_cols)
num_cols=df.dtypes!='Object'
num_cols = list(num_cols[num_cols].index)
print("Numerical Coloumns= ", num_cols)
```

```
DATA 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
None
Categories Coloumns=  []
Numerical Coloumns=  ['datetime', 'season', 'holiday', 'workingday', 'weather',
'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
```

[35]:
```
print("DATA DESCRIPTION")

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

```
DATA DESCRIPTION
```

[35]:

	datetime	season	holiday	workingday	\
count	10886	10886.000000	10886.000000	10886.000000	
unique	10886	NaN	NaN	NaN	
top	2011-01-01 00:00:00	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	
mean	NaN	2.506614	0.028569	0.680875	
std	NaN	1.116174	0.166599	0.466159	
min	NaN	1.000000	0.000000	0.000000	
25%	NaN	2.000000	0.000000	0.000000	
50%	NaN	3.000000	0.000000	1.000000	
75%	NaN	4.000000	0.000000	1.000000	
max	NaN	4.000000	1.000000	1.000000	

	weather	temp	atemp	humidity	windspeed	\
count	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	
unique	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	
mean	1.418427	20.23086	23.655084	61.886460	12.799395	
std	0.633839	7.79159	8.474601	19.245033	8.164537	
min	1.000000	0.82000	0.760000	0.000000	0.000000	
25%	1.000000	13.94000	16.665000	47.000000	7.001500	
50%	1.000000	20.50000	24.240000	62.000000	12.998000	

```
75%          2.000000     26.24000     31.060000     77.000000     16.997900
max          4.000000     41.00000     45.455000    100.000000     56.996900

              casual    registered         count
count   10886.000000  10886.000000  10886.000000
unique           NaN           NaN           NaN
top              NaN           NaN           NaN
freq             NaN           NaN           NaN
mean       36.021955    155.552177    191.574132
std        49.960477    151.039033    181.144454
min         0.000000      0.000000      1.000000
25%         4.000000     36.000000     42.000000
50%        17.000000    118.000000    145.000000
75%        49.000000    222.000000    284.000000
max       367.000000    886.000000    977.000000
```

[36]:
```
print("CHECKING NULL VALUES")

print(df.isna().sum().sum())

print("No Null Values Detected")
```

```
CHECKING NULL VALUES
0
No Null Values Detected
```

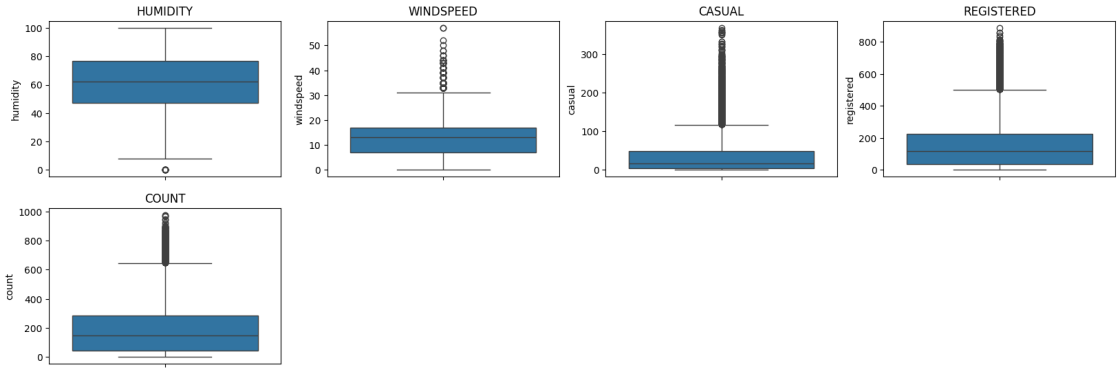
[37]:
```
print("CHECKING DUPLICATED ROWS")
print(df.duplicated().sum())
print("There are no duplicated rows")
```

```
CHECKING DUPLICATED ROWS
0
There are no duplicated rows
```

[38]:
```
print("CHECKING OUTLIERS")

plt.figure(figsize=(20,10))
plt.subplot(3,4,1)
plt.title('HUMIDITY')
sns.boxplot(data=df,y=df['humidity'])
plt.subplot(3,4,2)
plt.title('WINDSPEED')
sns.boxplot(data=df,y=df['windspeed'])
plt.subplot(3,4,3)
plt.title('CASUAL')
sns.boxplot(data=df,y=df['casual'])
plt.subplot(3,4,4)
```

```
plt.title('REGISTERED')
sns.boxplot(data=df,y=df['registered'])
plt.subplot(3,4,5)
plt.title('COUNT')
sns.boxplot(data=df, y=df['count'])

print("We can see that windspeed, casual, registered, count have outliers on␣
 ↪upper limit side and humidity has outliers on the lowwer limit side.")
```

CHECKING OUTLIERS
We can see that windspeed, casual, registered, count have outliers on upper limit side and humidity has outliers on the lowwer limit side.



[39]:
```
q1= df.quantile(.05)
q2= df.quantile(.95)
print(q1,q2)

def clip_array(arr, min_val, max_val):
  return arr[(arr >= min_val) & (arr <= max_val)]

# Example usage
min_val = q1
max_val = q2

clipped_arr = clip_array(df, min_val, max_val)
print(clipped_arr)  # Output: [5 7]
```

```
season         1.00
holiday        0.00
workingday     0.00
weather        1.00
temp           8.20
atemp          9.85
humidity      31.00
```

```
windspeed      0.00
casual         0.00
registered     4.00
count          5.00
Name: 0.05, dtype: float64 season          4.0000
holiday         0.0000
workingday      1.0000
weather         3.0000
temp           32.8000
atemp          36.3650
humidity       93.0000
windspeed      27.9993
casual        141.0000
registered    464.0000
count         563.7500
Name: 0.95, dtype: float64
      datetime  season  holiday  workingday  weather   temp   atemp  humidity  \
0          NaN       1      0.0           0      1.0   9.84  14.395      81.0
1          NaN       1      0.0           0      1.0   9.02  13.635      80.0
2          NaN       1      0.0           0      1.0   9.02  13.635      80.0
3          NaN       1      0.0           0      1.0   9.84  14.395      75.0
4          NaN       1      0.0           0      1.0   9.84  14.395      75.0
…        …     …       …          …       …    …     …       …
10881      NaN       4      0.0           1      1.0  15.58  19.695      50.0
10882      NaN       4      0.0           1      1.0  14.76  17.425      57.0
10883      NaN       4      0.0           1      1.0  13.94  15.910      61.0
10884      NaN       4      0.0           1      1.0  13.94  17.425      61.0
10885      NaN       4      0.0           1      1.0  13.12  16.665      66.0

       windspeed  casual  registered  count
0         0.0000     3.0        13.0   16.0
1         0.0000     8.0        32.0   40.0
2         0.0000     5.0        27.0   32.0
3         0.0000     3.0        10.0   13.0
4         0.0000     0.0         NaN    NaN
…           …     …          …    …
10881    26.0027     7.0       329.0  336.0
10882    15.0013    10.0       231.0  241.0
10883    15.0013     4.0       164.0  168.0
10884     6.0032    12.0       117.0  129.0
10885     8.9981     4.0        84.0   88.0

[10886 rows x 12 columns]

<ipython-input-39-a23a1e5bdebf>:1: FutureWarning: The default value of
numeric_only in DataFrame.quantile 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.
```

```
  q1= df.quantile(.05)
<ipython-input-39-a23a1e5bdebf>:2: FutureWarning: The default value of
numeric_only in DataFrame.quantile 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.
  q2= df.quantile(.95)
<ipython-input-39-a23a1e5bdebf>:6: FutureWarning: Automatic reindexing on
DataFrame vs Series comparisons is deprecated and will raise ValueError in a
future version. Do `left, right = left.align(right, axis=1, copy=False)` before
e.g. `left == right`
  return arr[(arr >= min_val) & (arr <= max_val)]
```

Column Profiling:

---

datetime: datetime

season: season (1: spring, 2: summer, 3: fall, 4: winter)

holiday: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)

workingday: if day is neither weekend nor holiday is 1, otherwise is 0.

weather:

1: Clear, Few clouds, partly cloudy, partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp: temperature in Celsius

atemp: feeling temperature in Celsius

humidity: humidity

windspeed: wind speed

casual: count of casual users

registered: count of registered users

count: count of total rental bikes including both casual and registered

[40]:
```
print("If the sum of casual and registered users is equal to the final count or␣
  ↪not")
(df['count']==df['casual']+df['registered']).all()
```

```
If the sum of casual and registered users is equal to the final count or not
```

[40]:
```
True
```

[41]:
```
print(df['workingday'].value_counts())
```

```
1    7412
0    3474
Name: workingday, dtype: int64
```

[42]:
```
print(df['holiday'].value_counts())
```

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

[43]:
```
print(df['season'].value_counts())
```

```
4    2734
2    2733
3    2733
1    2686
Name: season, dtype: int64
```

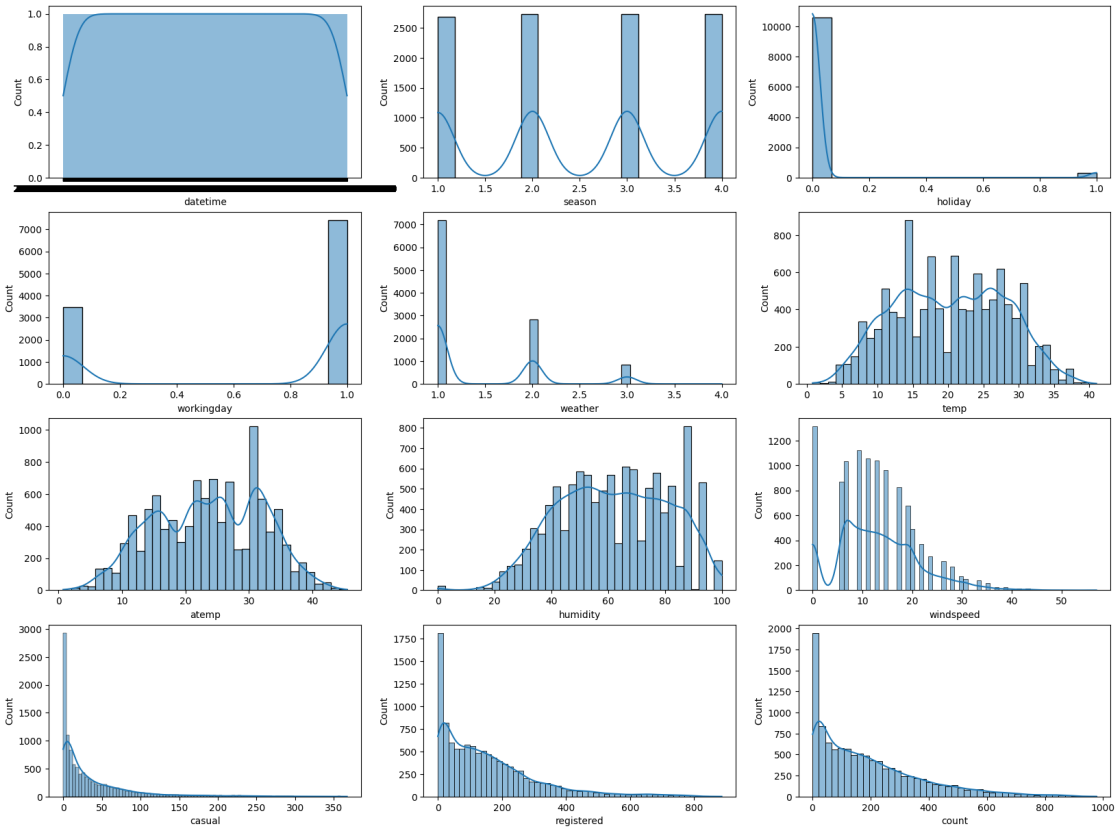
[44]:
```
print(df['windspeed'].value_counts())
```

```
0.0000     1313
8.9981     1120
11.0014    1057
12.9980    1042
7.0015     1034
15.0013     961
6.0032      872
16.9979     824
19.0012     676
19.9995     492
22.0028     372
23.9994     274
26.0027     235
27.9993     187
30.0026     111
31.0009      89
32.9975      80
35.0008      58
39.0007      27
36.9974      22
43.0006      12
40.9973      11
43.9989       8
46.0022       3
56.9969       2
47.9988       2
51.9987       1
50.0021       1
```

```
Name: windspeed, dtype: int64
```

[45]:
```
# Create subplots
print("UNIVARIATE ANALYSIS")
n_cols = 3
n_rows = (len(df.columns) + n_cols - 1) // n_cols
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(20, 15))

# Plot distplots for each column
for i, col in enumerate(df.columns):
    row = i // n_cols
    col_index = i % n_cols
    sns.histplot(df[col], ax=axes[row, col_index], kde=True)
```

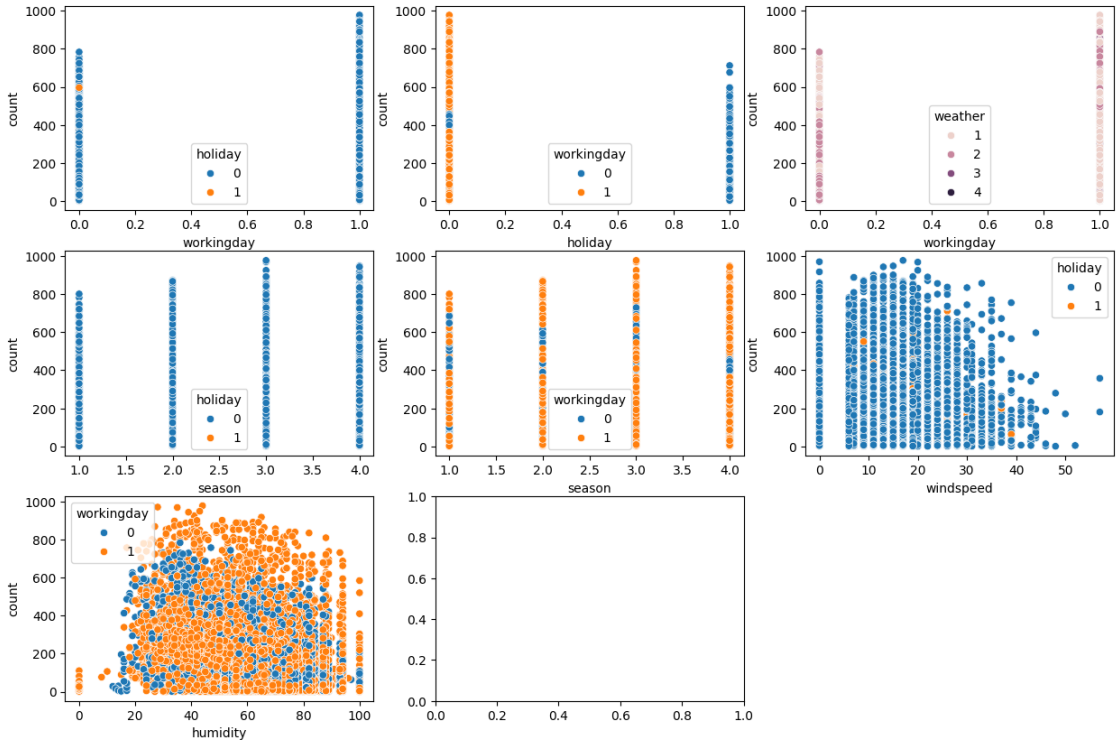
```
UNIVARIATE ANALYSIS
```



[46]:
```
plt.figure(figsize=(15,10))
plt.subplot(331)
sns.scatterplot(x=df['workingday'], y=df['count'], hue=df['holiday'])
plt.subplot(332)
sns.scatterplot(x=df['holiday'], y=df['count'], hue=df['workingday'])
```

```
plt.subplot(333)
sns.scatterplot(x=df['workingday'], y=df['count'], hue=df['weather'])
plt.subplot(334)
sns.scatterplot(x=df['season'], y=df['count'], hue=df['holiday'])
plt.subplot(335)
sns.scatterplot(x=df['season'], y=df['count'], hue=df['workingday'])
plt.subplot(336)
sns.scatterplot(x=df['windspeed'], y=df['count'], hue=df['holiday'])
plt.subplot(337)
sns.scatterplot(x=df['humidity'], y=df['count'], hue=df['workingday'])
plt.subplot(338)
```

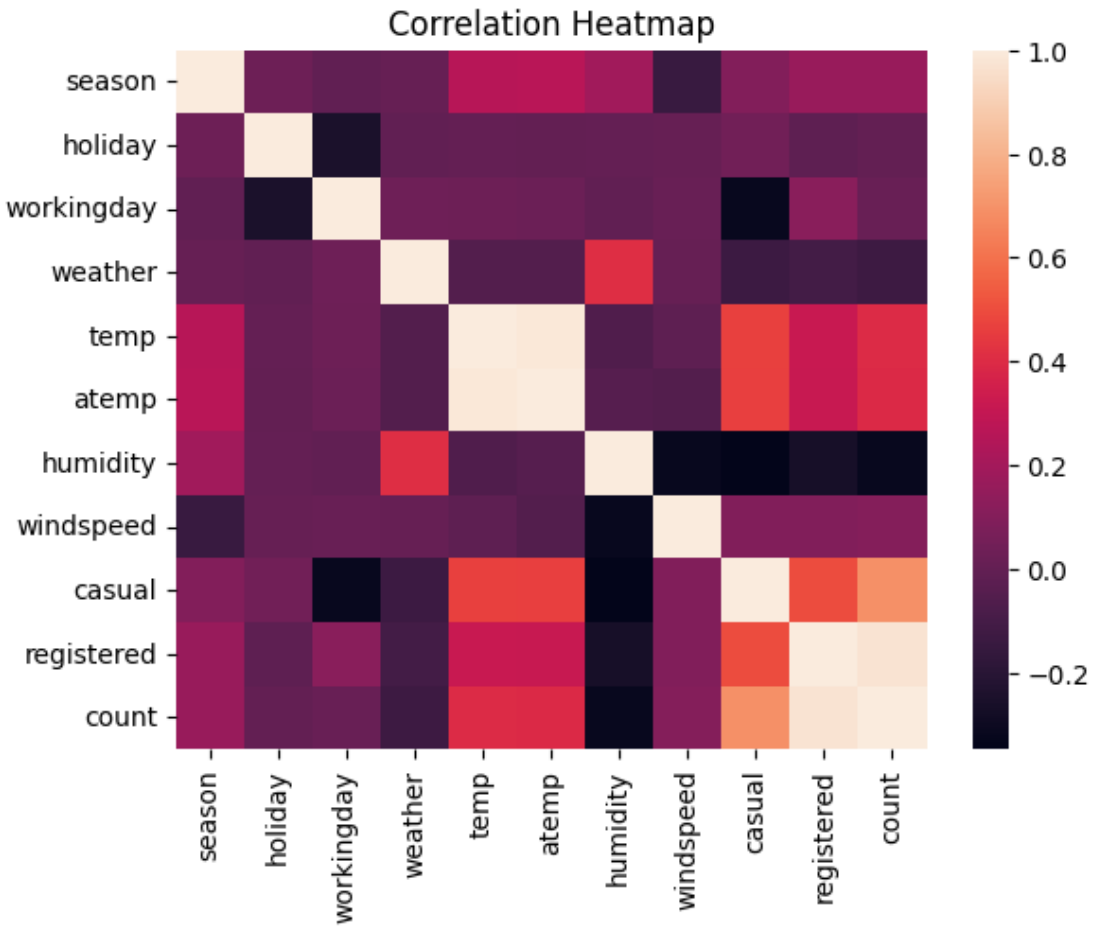
[46]: <Axes: >



[47]:
```
df_corr=df.corr()
sns.heatmap(df_corr)
plt.title('Correlation Heatmap')
```

```
<ipython-input-47-b82dcb1993f3>:1: 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=df.corr()
```

[47]: Text(0.5, 1.0, 'Correlation Heatmap')



[48]:
```
print("DROPPING HIGHLY CORRELATED COLOUMN atemp")
df=df.drop(['atemp'], axis=1)
```

```
DROPPING HIGHLY CORRELATED COLOUMN atemp
```

[49]:
```
print("ADDING A NEW FEATURE TO CHECK IF CURRENT DAT IS WEEKDAY OR NOT")
df['datetime'] = pd.to_datetime(df['datetime'])
def is_weekday(date):
    return date.weekday() < 5
df['is_weekday'] = df['datetime'].apply(lambda x: is_weekday(x))
```

```
ADDING A NEW FEATURE TO CHECK IF CURRENT DAT IS WEEKDAY OR NOT
```

Test 1- Checking if there any significant difference between the no. of bike rides on Weekdays and Weekends?

Test Name- 2 Sample T-test

Significance Level- 5%

NULL HYPOTHESIS(H0)- There is no difference in the sale of bikes on weekdays and weekends

ALTERNATE HYPOTHESIS(H1)- There is a difference in the sale of bikes on weekdays and weekends

[50]:
```
df.groupby('is_weekday')['count'].mean()
```

[50]:
```
is_weekday
False    188.765096
True     192.724589
Name: count, dtype: float64
```

[51]:
```
weekday_sales=df[df['is_weekday']==True]['count']
print(len(weekday_sales))
weekend_sales=df[df['is_weekday']==False]['count']
print(len(weekend_sales))
```

```
7723
3163
```

[52]:
```
ttest_stat, pval= ttest_ind(weekday_sales, weekend_sales)
print("Ttest Statitic- ", ttest_stat)
print("P-Value- ", pval)
siglvl=.05
if(pval>siglvl):
    print("We cannot reject the null hypothesis")
if(pval<siglvl):
    print("We can reject the null hypothesis, and accept the alternate␣
 ↪hypothesis")
```

```
Ttest Statitic-  1.0354386367292092
P-Value-  0.30048711429228286
We cannot reject the null hypothesis
```

Result- Since the PValue is not less than the assumed 5% significance level se cannot reject the Null hypothesis i.e. There is not significant difference in the sales on weekdays and weekends.

Test 2- Checking if the demand of bicycles on rent is the same for different Weather conditions?

Test name- One way ANOVA

Significance level- 5%

NULL HYPOTHESIS(H0)- The demand for the bicycles is same for all the weathers

ALTERNATE HYPOTHESIS (H1)- The demand varies as the weather change

[53]:
```
df.groupby("weather")['count'].mean()
```

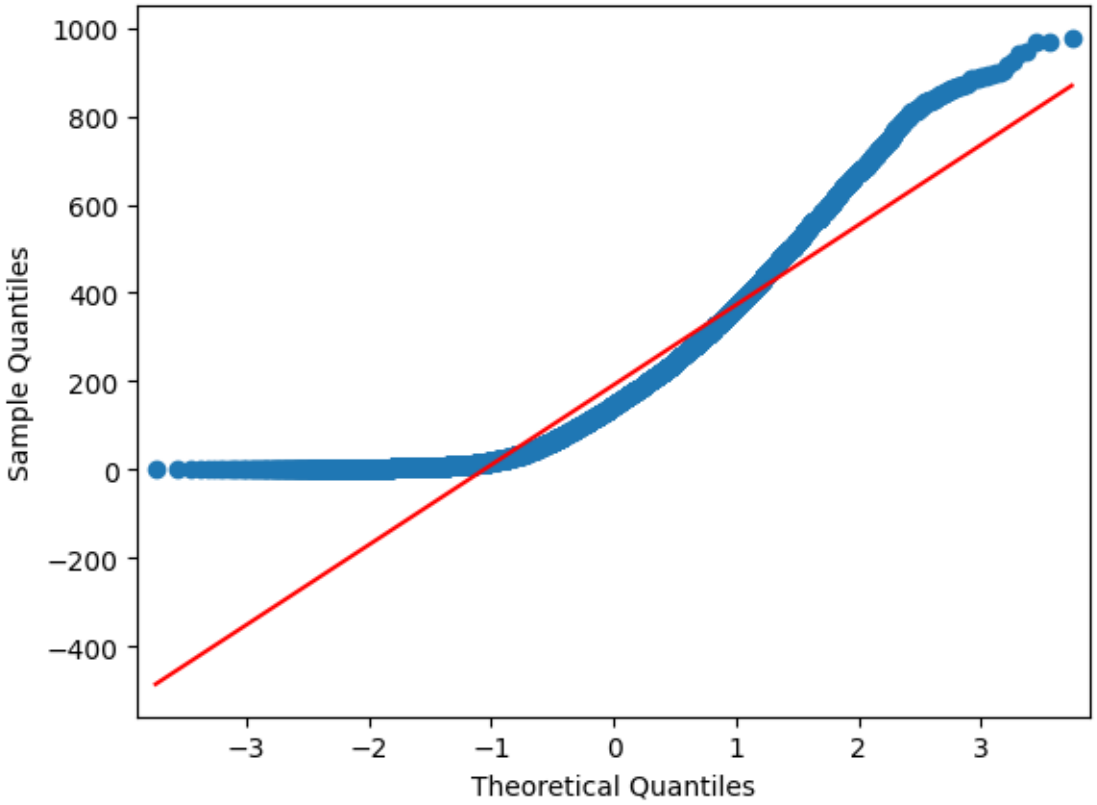
[53]:
```
weather
1    205.236791
2    178.955540
3    118.846333
4    164.000000
Name: count, dtype: float64
```

[54]:
```
weather1=df[df['weather']==1]['count']
weather2=df[df['weather']==2]['count']
weather3=df[df['weather']==3]['count']
weather4=df[df['weather']==4]['count']
```

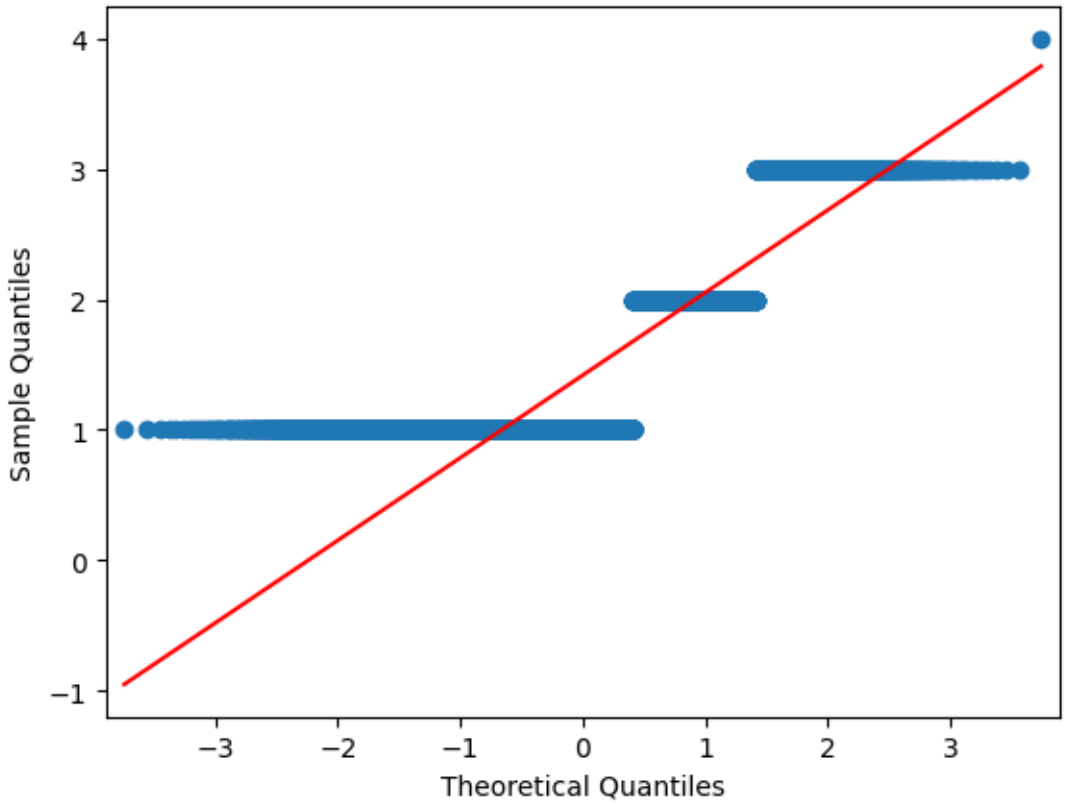
Checking the assumptions of the test.

[55]:
```
print("Normality Test")
qqplot(df['count'], line="s")
plt.show()
```

```
Normality Test
```



[56]:
```
qqplot(df['weather'], line='s')
plt.show()
```

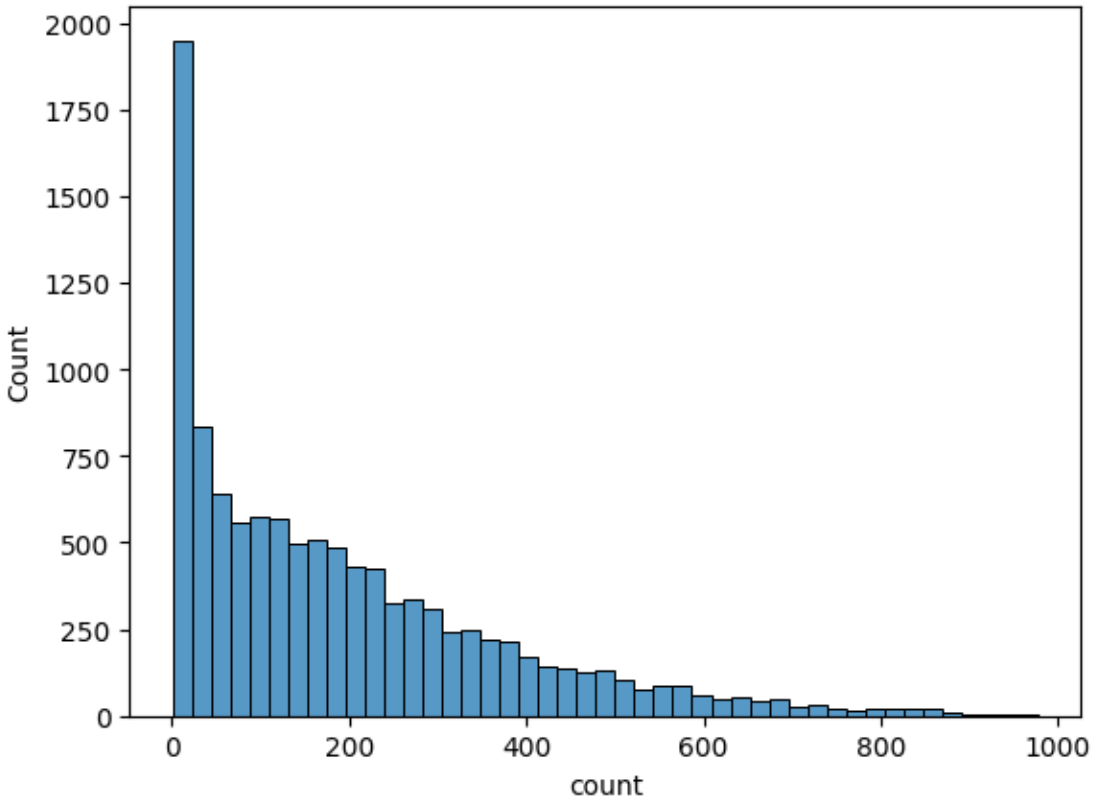


[57]:
```
np.random.seed(23)
count=df['count']
weather=df['weather']
count_subset = count.sample(100)
# H0: data is gaussian
# H1: data is NOT gaussian
test_stat, p_val = shapiro(count_subset)

print(p_val)

np.random.seed(23)
weather=df['weather']
weather_subset = weather.sample(100)
# H0: data is gaussian
# H1: data is NOT gaussian
test_stat, p_val = shapiro(weather_subset)

print(p_val)
```

```
2.4483579608158834e-09
4.1655338846226936e-14
```

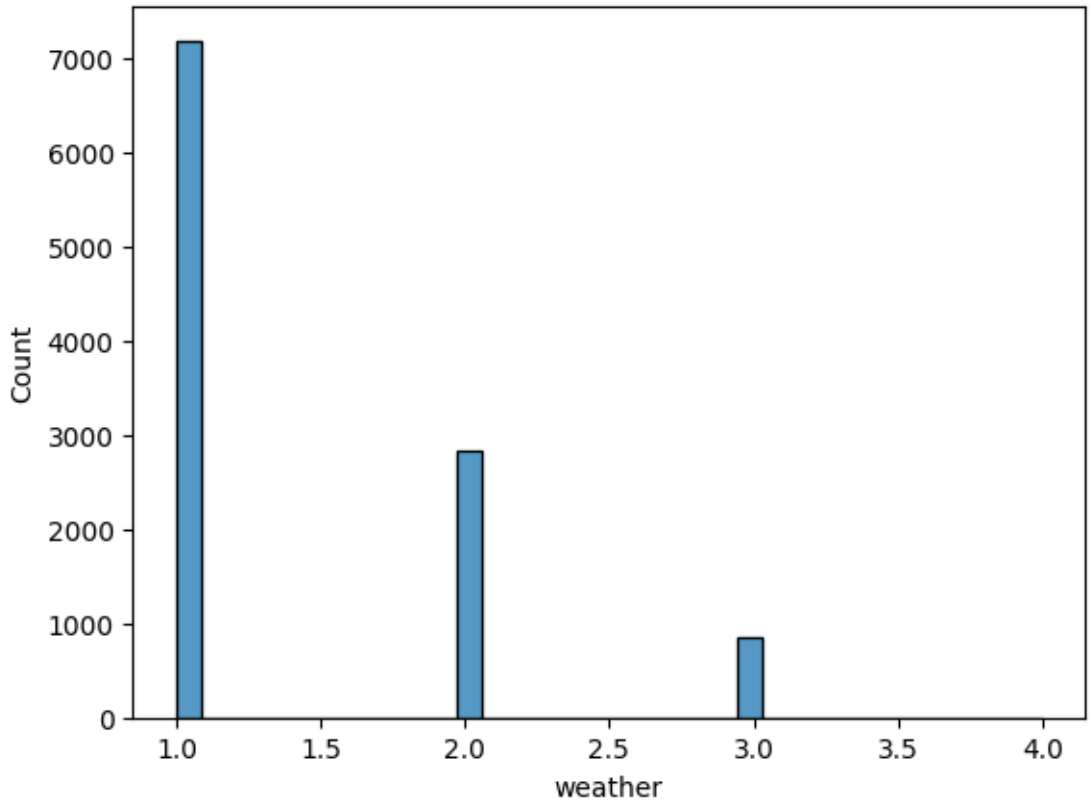
[58]:
```
sns.histplot(count)
```

[58]:
```
<Axes: xlabel='count', ylabel='Count'>
```



[59]:
```
sns.histplot(weather)
```

[59]:
```
<Axes: xlabel='weather', ylabel='Count'>
```



[60]:
```
print('Levene\'s Test')
levene(weather1, weather2, weather3, weather4)
```

```
Levene's Test
```

[60]:
```
LeveneResult(statistic=54.85106195954556, pvalue=3.504937946833238e-35)
```

[61]:
```
fstats, p_val= kruskal(weather1, weather2, weather3, weather4)
print("F Statitic- ", fstats)
print("P-Value- ", p_val)
siglvl=.05
if(p_val>siglvl):
    print("We cannot reject the null hypothesis")
if(p_val<siglvl):
    print("We can reject the null hypothesis, and accept the alternate␣
 ↪hypothesis")
```

```
F Statitic-  205.00216514479087
P-Value-  3.501611300708679e-44
We can reject the null hypothesis, and accept the alternate hypothesis
```

Result- We can clearly see that there is a significant variation in the demand accross the Weather

heighest being in Weather 1 and lowest being in Weather 3. We have conducted a ONE WAY ANOVA to verify this.

We have distributed the datasets for all the weather in 4 different categories. Then checked the assumptions needed for the One way ANOVA . We can see that the normality , equality of variance are failing in both the shapario, levenes test and we have verified it visually using qqplot and histplot.

So not having the conducive situtations for the ANOVA test we are going to use kruskal wallis test.

In kruskal wallis test we can see that the pvalue is less than .05 so we can reject the null hypothesis and can see that there is a clear differnce in the demand of bicycle in different weather.

Test 3- Checking if the demand of bicycles on rent is the same for different Seasons?

Test Name- One way ANOVA

Significant Level- 5%

NULL HYPOTHESIS (HO)- There is no difference in the demand accross all the seasons.

ALTERNATE HYPOTHESIS (H1)- There is significant difference in the demands accross all the seasons.

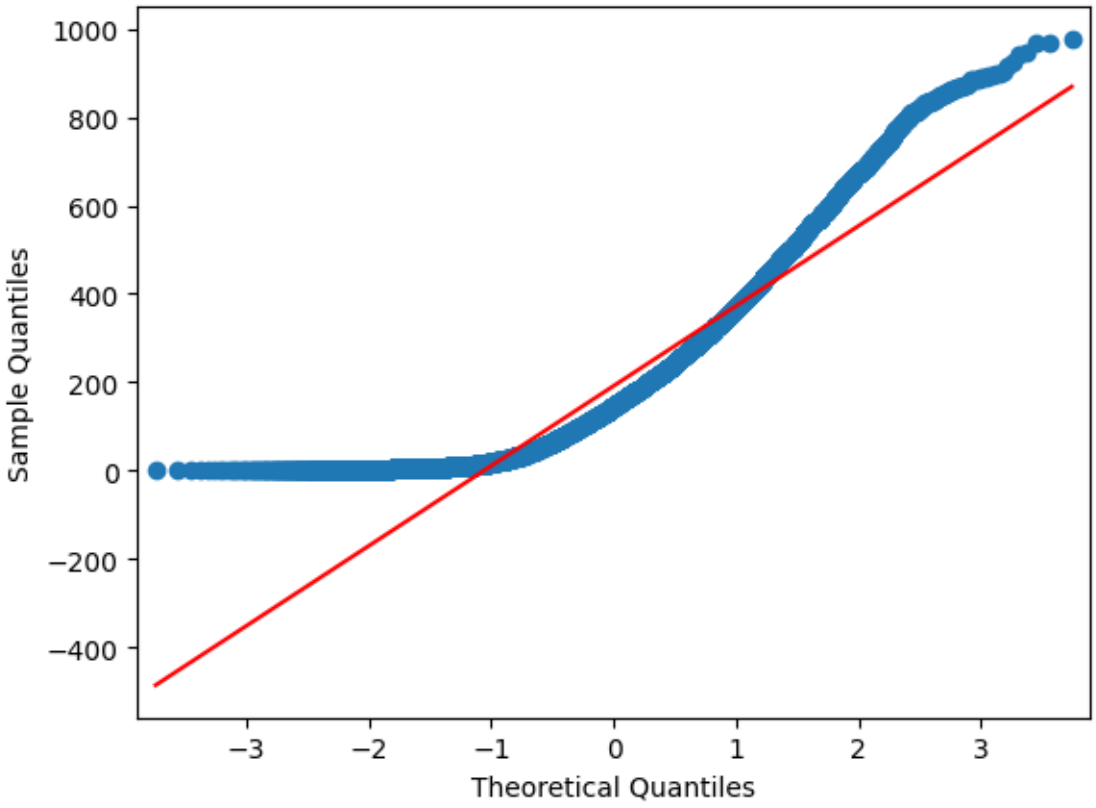
[62]:
```
df.groupby("season")['count'].mean()
```

[62]:
```
season
1    116.343261
2    215.251372
3    234.417124
4    198.988296
Name: count, dtype: float64
```

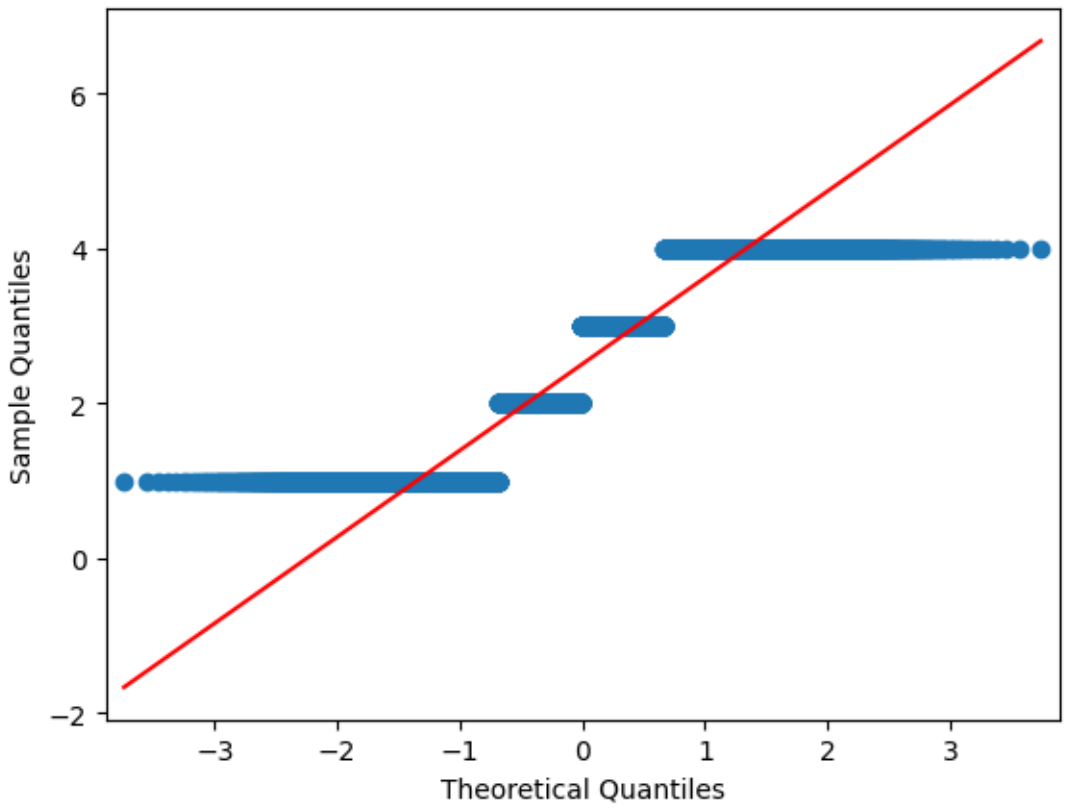
[63]:
```
season1=df[df['season']==1]['count']
season2=df[df['season']==2]['count']
season3=df[df['season']==3]['count']
season4=df[df['season']==4]['count']
```

[64]:
```
print("Normality Test")
qqplot(df['count'], line="s")
plt.show()
```

```
Normality Test
```



[65]:
```
qqplot(df['season'], line='s')
plt.show()
```

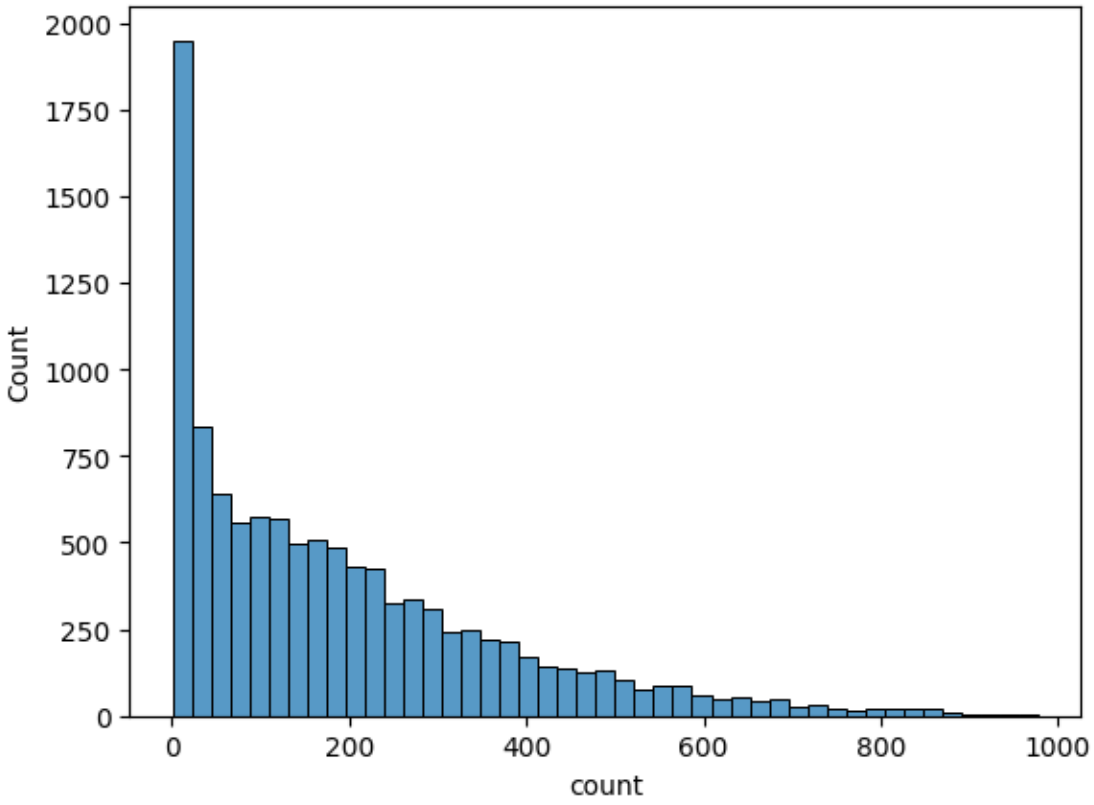


[66]:
```
np.random.seed(23)
count=df['count']
season=df['season']
count_subset = count.sample(100)
# H0: data is gaussian
# H1: data is NOT gaussian
test_stat, p_val = shapiro(count_subset)

print(p_val)

np.random.seed(23)
season=df['season']
season_subset = season.sample(100)
# H0: data is gaussian
# H1: data is NOT gaussian
test_stat, p_val = shapiro(season_subset)

print(p_val)
```

```
2.4483579608158834e-09
1.5530936536833906e-08
```

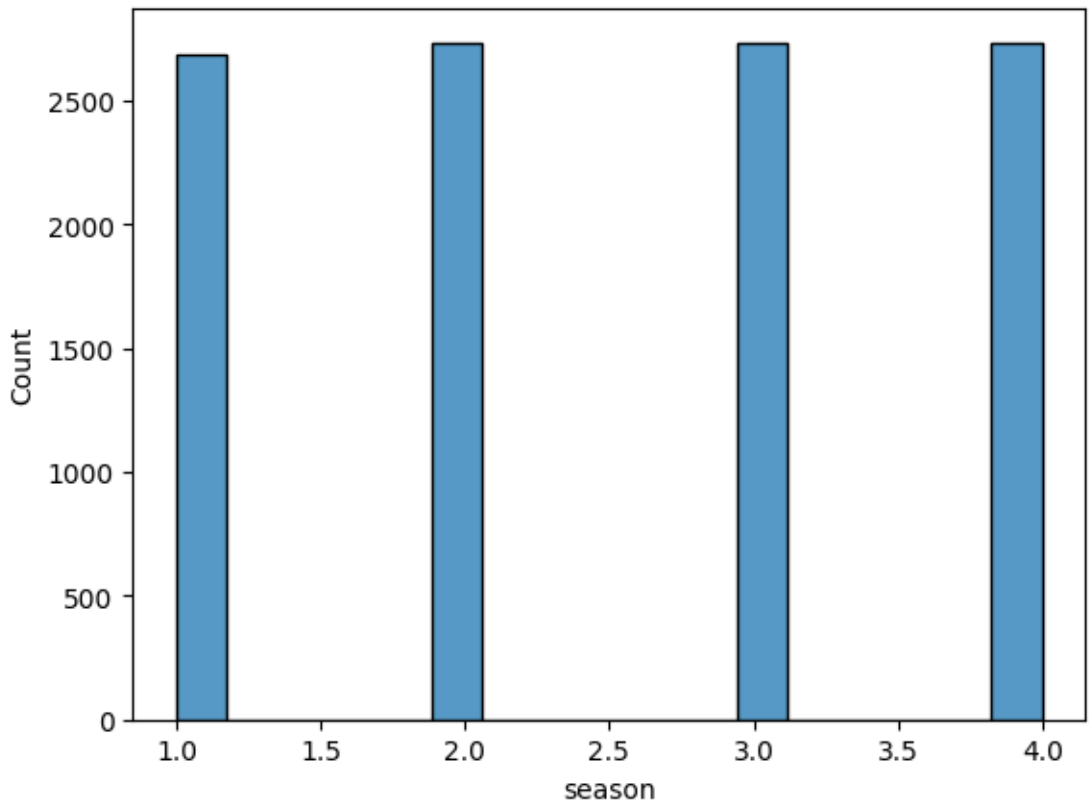
[67]:
```
sns.histplot(count)
```

[67]: <Axes: xlabel='count', ylabel='Count'>



[68]:
```
sns.histplot(season)
```

[68]: <Axes: xlabel='season', ylabel='Count'>



[69]:
```
print('Levene\'s Test')
levene(season1, season2, season3, season4)
```

```
Levene's Test
```

[69]:
```
LeveneResult(statistic=187.7706624026276, pvalue=1.0147116860043298e-118)
```

[70]:
```
fstats, p_val= kruskal(season1, season2, season3, season4)
print("F Statitic- ", fstats)
print("P-Value- ", p_val)
siglvl=.05
if(p_val>siglvl):
    print("We cannot reject the null hypothesis")
if(p_val<siglvl):
    print("We can reject the null hypothesis, and accept the alternate␣
 ↪hypothesis")
```

```
F Statitic-  699.6668548181988
P-Value-  2.479008372608633e-151
We can reject the null hypothesis, and accept the alternate hypothesis
```

Result- We can clearly see that there is a significant variation in the demand accross the seasons

heighest being in Season 1 and lowest being in season 3. We have conducted a ONE WAY ANOVA to verify this.

We have distributed the datasets for all the season in 4 different categories. Then checked the assumptions needed for the One way ANOVA . We can see that the normality , equality of variance are failing in both the shapario, levenes test and we have verified it visually using qqplot and histplot.

So not having the conducive situtations for the ANOVA test we are going to use kruskal wallis test.

In kruskal wallis test we can see that the pvalue is less than .05 so we can reject the null hypothesis and can see that there is a clear differnce in the demand of bicycle in different seasons.

Test 4- Checking if the Weather conditions are significantly different during different Seasons?

Test name- Chi 2 contingency

Significant Level- 5%

NULL HYPOTHESIS (H0)- There is no difference in the waether of different seasons.

ALTERNATE HYPOTHESIS (H1)- There is significant difference in the weather of sifferent seasons.

[71]:
```
season_weather = pd.crosstab(index=df['season'],columns=df['weather'])
chi_stat, p_value, dof, exp_freq = chi2_contingency(season_weather)
print("Chi Statitic- ", chi_stat)
print("P-Value- ", p_value)

siglvl=.05

if(p_value>siglvl):
    print("We cannot reject the null hypothesis")
if(p_value<siglvl):
    print("We can reject the null hypothesis, and accept the alternate␣
 ↪hypothesis")
```

```
Chi Statitic-  49.158655596893624
P-Value-  1.549925073686492e-07
We can reject the null hypothesis, and accept the alternate hypothesis
```

Result- We can see that the p value for the above test is less than .05 so we can reject the null hypothesis and accept that the weather in all seasons is significantly different.