YULU

- 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.

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

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import ttest_ind,f_oneway, levene, kruskal, shapiro, chi2_contingency
from statsmodels.graphics.gofplots import qqplot
```

In [2]:
```
!gdown 1-jH3nr8HNyHLkW-4fQ5C2CY94Oqxp3Xo
```

```
Downloading...
From: https://drive.google.com/uc?id=1-jH3nr8HNyHLkW-4fQ5C2CY94Oqxp3Xo
To: /content/bike_sharing.csv
100% 648k/648k [00:00<00:00, 20.2MB/s]
```

In [3]:
```
bike_df = pd.read_csv("bike_sharing.csv")
```

In [4]:
```
bike_df.shape
```

Out[4]:
```
(10886, 12)
```

In [5]:
```
bike_df.head()
```

Out[5]:

	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

In [6]:
```
bike_df.info()
```

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

Changing the requied columns

In [7]:
```
bike_df["datetime"] = pd.to_datetime(bike_df["datetime"])
```

In [8]:
```
bike_df["datetime"].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 10886 entries, 0 to 10885
Series name: datetime
Non-Null Count  Dtype
--------------  -----
10886 non-null  datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 85.2 KB
```

In [9]:
```
bike_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
```

In [10]:
```
bike_df.columns.tolist()
```

Out[10]:
```
['datetime',
 'season',
 'holiday',
 'workingday',
 'weather',
 'temp',
 'atemp',
 'humidity',
```

```
 'windspeed',
 'casual',
 'registered',
 'count']
```

In [11]:
```
bike_df.skew(numeric_only = True)
```

Out[11]:

	0
season	-0.007076
holiday	5.660517
workingday	-0.776163
weather	1.243484
temp	0.003691
atemp	-0.102560
humidity	-0.086335
windspeed	0.588767
casual	2.495748
registered	1.524805
count	1.242066

dtype: float64

Symmetrical Majority:

- The majority of the variables, including 'season' and 'temp', exhibit skewness values close to zero, suggesting relatively symmetrical distributions.

Positive Skewness Insights:

- Variables such as 'holiday', 'weather', 'windspeed', 'casual', 'registered', and 'count' demonstrate positive skewness, pointing to a concentration of lower values and a rightward skew in their distributions.

Negative Skewness Observations:

- In contrast, 'workingday', 'atemp', and 'humidity' exhibit negative skewness, implying a concentration of higher values and a leftward skew in their distributions.

In [12]:
```
bike_df["weather"].replace({1:"clear", 2:"cloudy", 3:"light_rains", 4:"thunderstroms"}, inplace=True)
bike_df["season"].replace({1:"spring", 2:"summer", 3:"fall", 4:"winter"}, inplace=True)
bike_df["holiday"].replace({1:"yes", 0:"no"}, inplace=True)
bike_df['workingday'] = bike_df['workingday'].replace({0:'No',1:'Yes'})
```

```
<ipython-input-12-43e38526f274>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thr
ough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  bike_df["weather"].replace({1:"clear", 2:"cloudy", 3:"light_rains", 4:"thunderstroms"}, inplace=True)
<ipython-input-12-43e38526f274>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thr
ough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  bike_df["season"].replace({1:"spring", 2:"summer", 3:"fall", 4:"winter"}, inplace=True)
<ipython-input-12-43e38526f274>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thr
ough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  bike_df["holiday"].replace({1:"yes", 0:"no"}, inplace=True)
```

In [13]:

```
bike_df["date"] = bike_df["datetime"].dt.date
bike_df["time"] = bike_df["datetime"].dt.time
bike_df["day"] = bike_df["datetime"].dt.day
bike_df["month"] = bike_df["datetime"].dt.month
bike_df["year"] = bike_df["datetime"].dt.year
bike_df["hour"] = bike_df["datetime"].dt.hour


# change of month
bike_df['month'] = bike_df['month'].replace({1: 'January',
                                   2: 'February',
                                   3: 'March',
                                   4: 'April',
                                   5: 'May',
                                   6: 'June',
                                   7: 'July',
                                   8: 'August',
                                   9: 'September',
                                   10: 'October',
                                   11: 'November',
                                   12: 'December'})
```

In [14]:

```
bike_df.head()
```

Out[14]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	date	time	day	mo
0	2011-01-01 00:00:00	spring	no	No	clear	9.84	14.395	81	0.0	3	13	16	2011-01-01	00:00:00	1	Janu
1	2011-01-01	spring	no	No	clear	9.02	13.635	80	0.0	8	32	40	2011-01-	01:00:00	1	Janu

	01:00:00												01			
2	2011-01-01 02:00:00	spring	no	No	clear	9.02	13.635	80	0.0	5	27	32	2011-01-01	02:00:00	1	Janu
3	2011-01-01 03:00:00	spring	no	No	clear	9.84	14.395	75	0.0	3	10	13	2011-01-01	03:00:00	1	Janu
4	2011-01-01 04:00:00	spring	no	No	clear	9.84	14.395	75	0.0	0	1	1	2011-01-01	04:00:00	1	Janu

In [15]:
```
def day_period(x):
    if x in (4,5,6,7):
        return 'EarlyMorning'
    elif x in (8,9,10,11):
        return 'Morning'
    elif x in (12,13,14,15,16):
        return 'Afternoon'
    elif x in (17,18,19):
        return 'Evening'
    elif x in (20,21,22,23):
        return 'Night'
    elif x in (0,1,2,3):
        return 'LateNight'
```

In [16]:
```
bike_df["day_period"] = bike_df["hour"].apply(day_period)
```

In [17]:
```
bike_df.isnull().sum()
```

Out[17]:

	0
datetime	0

season	0
holiday	0
workingday	0
weather	0
temp	0
atemp	0
humidity	0
windspeed	0
casual	0
registered	0
count	0
date	0
time	0
day	0
month	0
year	0
hour	0
day_period	0

dtype: int64

In [18]:
```
np.any(bike_df.duplicated())
```

```
Out[18]: False
```

```
In [19]: bike_df.describe(include="all").T
```

Out[19]:

	count	unique	top	freq	mean	min	25%	50%	75%	max	std
datetime	10886	NaN	NaN	NaN	2011-12-27 05:56:22.399411968	2011-01-01 00:00:00	2011-07-02 07:15:00	2012-01-01 20:30:00	2012-07-01 12:45:00	2012-12-19 23:00:00	NaN
season	10886	4	winter	2734	NaN	NaN	NaN	NaN	NaN	NaN	NaN
holiday	10886	2	no	10575	NaN	NaN	NaN	NaN	NaN	NaN	NaN
workingday	10886	2	Yes	7412	NaN	NaN	NaN	NaN	NaN	NaN	NaN
weather	10886	4	clear	7192	NaN	NaN	NaN	NaN	NaN	NaN	NaN
temp	10886.0	NaN	NaN	NaN	20.23086	0.82	13.94	20.5	26.24	41.0	7.79159
atemp	10886.0	NaN	NaN	NaN	23.655084	0.76	16.665	24.24	31.06	45.455	8.474601
humidity	10886.0	NaN	NaN	NaN	61.88646	0.0	47.0	62.0	77.0	100.0	19.245033
windspeed	10886.0	NaN	NaN	NaN	12.799395	0.0	7.0015	12.998	16.9979	56.9969	8.164537
casual	10886.0	NaN	NaN	NaN	36.021955	0.0	4.0	17.0	49.0	367.0	49.960477
registered	10886.0	NaN	NaN	NaN	155.552177	0.0	36.0	118.0	222.0	886.0	151.039033
count	10886.0	NaN	NaN	NaN	191.574132	1.0	42.0	145.0	284.0	977.0	181.144454
date	10886	456	2011-01-01	24	NaN	NaN	NaN	NaN	NaN	NaN	NaN
time	10886	24	12:00:00	456	NaN	NaN	NaN	NaN	NaN	NaN	NaN
day	10886.0	NaN	NaN	NaN	9.992559	1.0	5.0	10.0	15.0	19.0	5.476608

month	10886	12	May	912	NaN	NaN	NaN	NaN	NaN	NaN	NaN
year	10886.0	NaN	NaN	NaN	2011.501929	2011.0	2011.0	2012.0	2012.0	2012.0	0.500019
hour	10886.0	NaN	NaN	NaN	11.541613	0.0	6.0	12.0	18.0	23.0	6.915838
day_period	10886	6	Afternoon	2280	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- There are 10,886 records in the dataset
- There are 12 columns in teh dataset
- The highest recorded temperature is 41 in celsius
- The highest recorded windspeed is about 57 kmph

In [20]:
```
min_date = bike_df["datetime"].dt.date.min()
max_date = bike_df["datetime"].dt.date.max()
print(min_date)
print(max_date)
```

```
2011-01-01
2012-12-19
```

In [21]:
```
max_date-min_date
```

Out[21]:
```
datetime.timedelta(days=718)
```

- the dataset having dates between 01-01-2011 to 19-12-2012. there are 718 days.

In [22]:
```
col_list = bike_df.columns.tolist()
```

In [23]:
```
for col in col_list:
  unique_vals = bike_df[col].nunique()
  count = f"Total unique values in the {col} are --> {unique_vals}"
  print()
  print(count)
```

```
Total unique values in the datetime are --> 10886

Total unique values in the season are --> 4

Total unique values in the holiday are --> 2

Total unique values in the workingday are --> 2

Total unique values in the weather are --> 4

Total unique values in the temp are --> 49

Total unique values in the atemp are --> 60

Total unique values in the humidity are --> 89

Total unique values in the windspeed are --> 28

Total unique values in the casual are --> 309

Total unique values in the registered are --> 731

Total unique values in the count are --> 822

Total unique values in the date are --> 456

Total unique values in the time are --> 24

Total unique values in the day are --> 19

Total unique values in the month are --> 12

Total unique values in the year are --> 2

Total unique values in the hour are --> 24

Total unique values in the day_period are --> 6
```

Value counts of different columns

In [24]:
```
subset_col = ["season", "holiday", "workingday", "weather"]
for col in subset_col:
  value_counts = bike_df[col].value_counts().reset_index()
  print(f"value counts in the {col} are as follows")
  print(value_counts)
```

```
value counts in the season are as follows
   season  count
0  winter   2734
1  summer   2733
2    fall   2733
3  spring   2686
value counts in the holiday are as follows
  holiday  count
0      no  10575
1     yes    311
value counts in the workingday are as follows
  workingday  count
0        Yes   7412
1         No   3474
value counts in the weather are as follows
         weather  count
0          clear   7192
1         cloudy   2834
2    light_rains    859
3  thunderstroms      1
```

Proportion of the different columns

In [25]:
```
subset_col = ["season", "holiday", "workingday", "weather"]
for col in subset_col:
  value_counts = np.round(bike_df[col].value_counts(normalize=True)*100,2).reset_index()
  print(f"Proportion of {col} is follows")
  print(value_counts)
```

```
Proportion of season is follows
   season  proportion
0  winter       25.11
1  summer       25.11
2    fall       25.11
```

```
3  spring       24.67
Proportion of holiday is follows
  holiday  proportion
0      no       97.14
1     yes        2.86
Proportion of workingday is follows
  workingday  proportion
0        Yes       68.09
1         No       31.91
Proportion of weather is follows
         weather  proportion
0          clear       66.07
1         cloudy       26.03
2    light_rains        7.89
3  thunderstroms        0.01
```

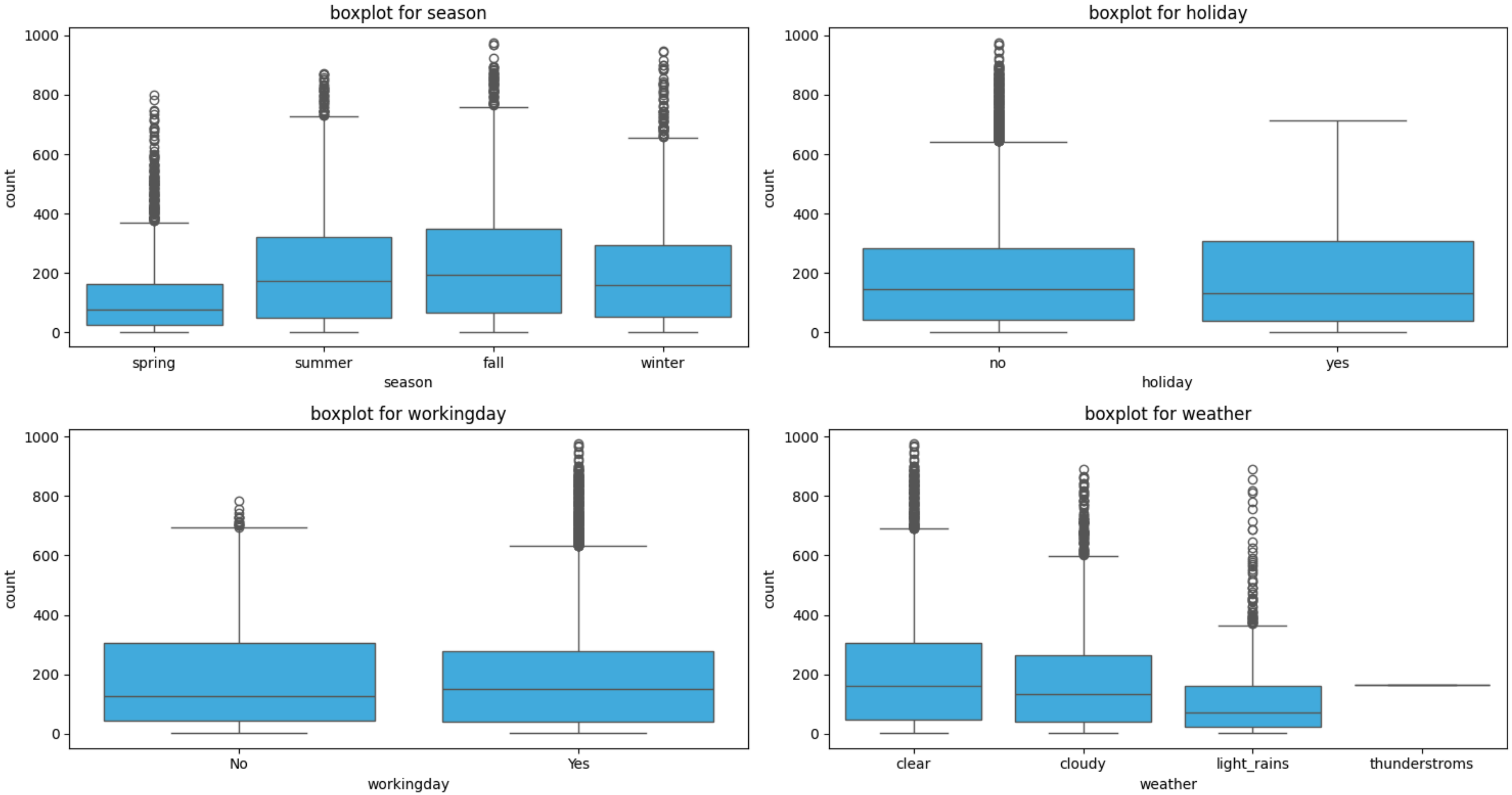
In [25]:

```

```

Outlier Detection

In [26]:

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

for i, column in enumerate(subset_col, 1):
  plt.subplot(2, 2, i)
  sns.boxplot(x=column, y="count", data = bike_df, color= "#29B6F6")
  plt.title(f"boxplot for {column}")
plt.tight_layout()
plt.show()
```



Outlier Analysis

Outliers in Different Seasons:

- In spring and winter, there are more unusual values in the data compared to other seasons.

Weather Outliers:

- Category 3 weather has a lot of unusual values, while category 4 weather doesn't have any.

Working Days vs. Holidays:

- On regular working days, there are more unusual values in the data than on holidays. This suggests some unexpected patterns during typical workdays that might need a closer look.

Univariate Analysis

In [27]:
```
col_list = ['season',
 'holiday',
 'workingday',
 'casual',
 'registered',
 'date',
 'time',
 'day',
 'month',
 'year',
 'hour',
 'day_period']
```

In [28]:
```
for col in col_list:
  value_counts = bike_df[col].value_counts().reset_index().sort_index()
  print(f"value counts in the {col} are as follows")
  print(value_counts)
```

```
value counts in the season are as follows
   season  count
0  winter   2734
1  summer   2733
2    fall   2733
3  spring   2686
value counts in the holiday are as follows
  holiday  count
0      no  10575
1     yes    311
```

```
value counts in the workingday are as follows
  workingday  count
0        Yes   7412
1         No   3474
value counts in the casual are as follows
     casual  count
0         0    986
1         1    667
2         2    487
3         3    438
4         4    354
..      ...    ...
304     332      1
305     361      1
306     356      1
307     331      1
308     304      1

[309 rows x 2 columns]
value counts in the registered are as follows
     registered  count
0             3    195
1             4    190
2             5    177
3             6    155
4             2    150
..          ...    ...
726         570      1
727         422      1
728         678      1
729         565      1
730         636      1

[731 rows x 2 columns]
value counts in the date are as follows
           date  count
0    2011-01-01     24
1    2012-04-18     24
2    2012-05-10     24
3    2012-05-09     24
4    2012-05-08     24
```

```
..           ...    ...
451  2011-01-12     22
452  2011-01-11     22
453  2011-01-03     22
454  2011-02-11     22
455  2011-01-18     12

[456 rows x 2 columns]
value counts in the time are as follows
        time  count
0   12:00:00    456
1   13:00:00    456
2   22:00:00    456
3   21:00:00    456
4   20:00:00    456
5   19:00:00    456
6   18:00:00    456
7   17:00:00    456
8   16:00:00    456
9   15:00:00    456
10  14:00:00    456
11  23:00:00    456
12  11:00:00    455
13  10:00:00    455
14  09:00:00    455
15  08:00:00    455
16  07:00:00    455
17  06:00:00    455
18  00:00:00    455
19  01:00:00    454
20  05:00:00    452
21  02:00:00    448
22  04:00:00    442
23  03:00:00    433
value counts in the day are as follows
    day  count
0     1    575
1     9    575
2    17    575
3     5    575
4    16    574
```

```
5    15    574
6    14    574
7    13    574
8    19    574
9     8    574
10    7    574
11    4    574
12    2    573
13   12    573
14    3    573
15    6    572
16   10    572
17   11    568
18   18    563
value counts in the month are as follows
        month  count
0         May    912
1        June    912
2        July    912
3      August    912
4    December    912
5     October    911
6    November    911
7       April    909
8   September    909
9    February    901
10      March    901
11    January    884
value counts in the year are as follows
   year  count
0  2012   5464
1  2011   5422
value counts in the hour are as follows
    hour  count
0     12    456
1     13    456
2     22    456
3     21    456
4     20    456
5     19    456
6     18    456
```

```
7     17    456
8     16    456
9     15    456
10    14    456
11    23    456
12    11    455
13    10    455
14     9    455
15     8    455
16     7    455
17     6    455
18     0    455
19     1    454
20     5    452
21     2    448
22     4    442
23     3    433
value counts in the day_period are as follows
     day_period  count
0     Afternoon   2280
1         Night   1824
2       Morning   1820
3  EarlyMorning   1804
4     LateNight   1790
5       Evening   1368
```

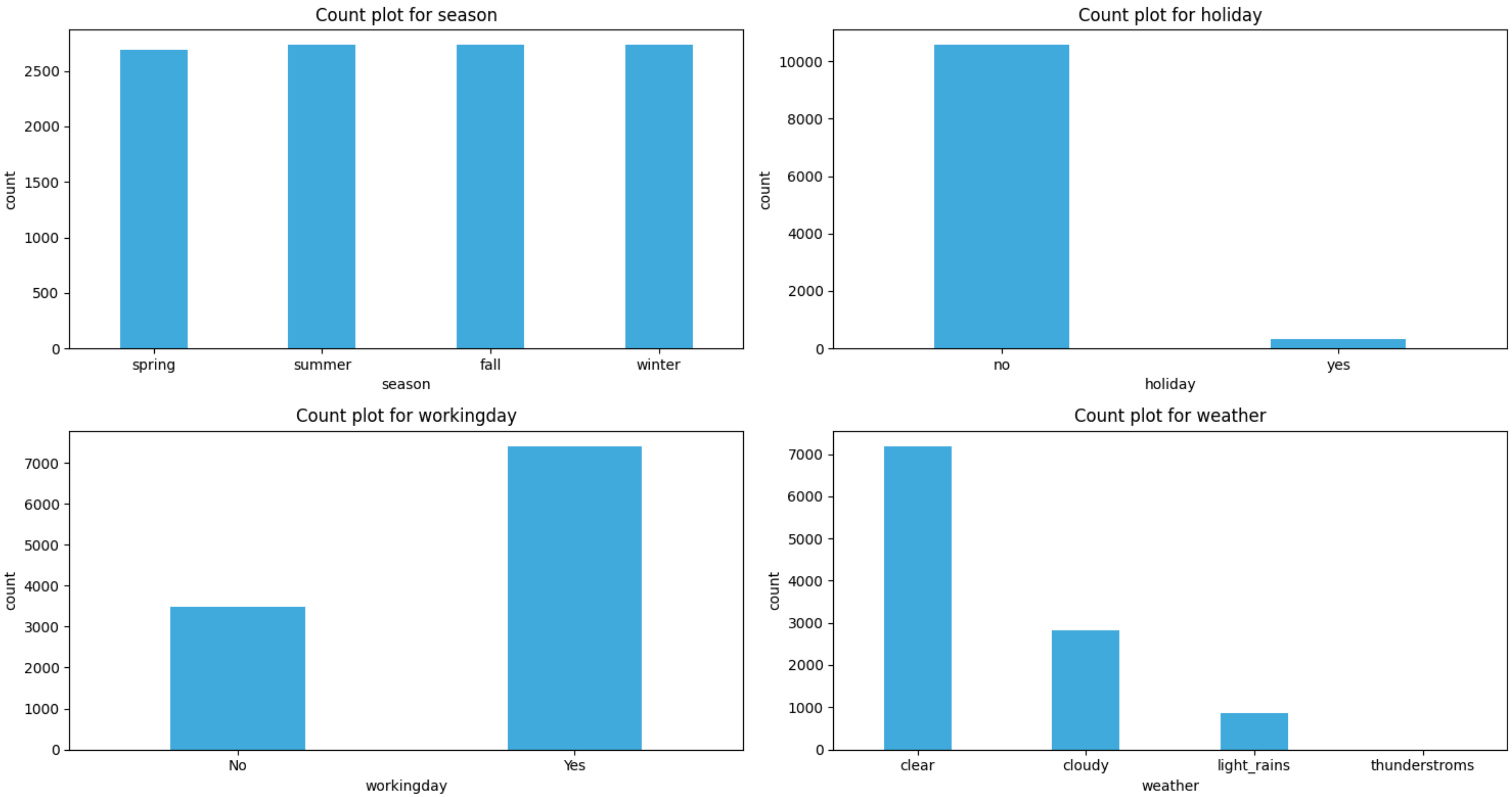
In [29]:

```
subset_col = ["season", "holiday", "workingday", "weather"]
values = {}

for col in col_list:
    value_counts = bike_df[col].value_counts()
    value_counts.columns = [col, f"{col}_count"]
    values[col] = value_counts
```

In [30]:

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

for i, column in enumerate(subset_col, 1):
  plt.subplot(2, 2, i)
  sns.countplot(x=column, data=bike_df, color="#29B6F6", width=0.4 )
  plt.title(f"Count plot for {column}")
```

```python
plt.tight_layout()
plt.show()
```



In [31]:

```python
values["season"]
```

Out[31]:

	count
season	
winter	2734

summer	2733
fall	2733
spring	2686

dtype: int64

In [32]:
```
values["holiday"]
```

Out[32]:

	count
holiday	
no	10575
yes	311

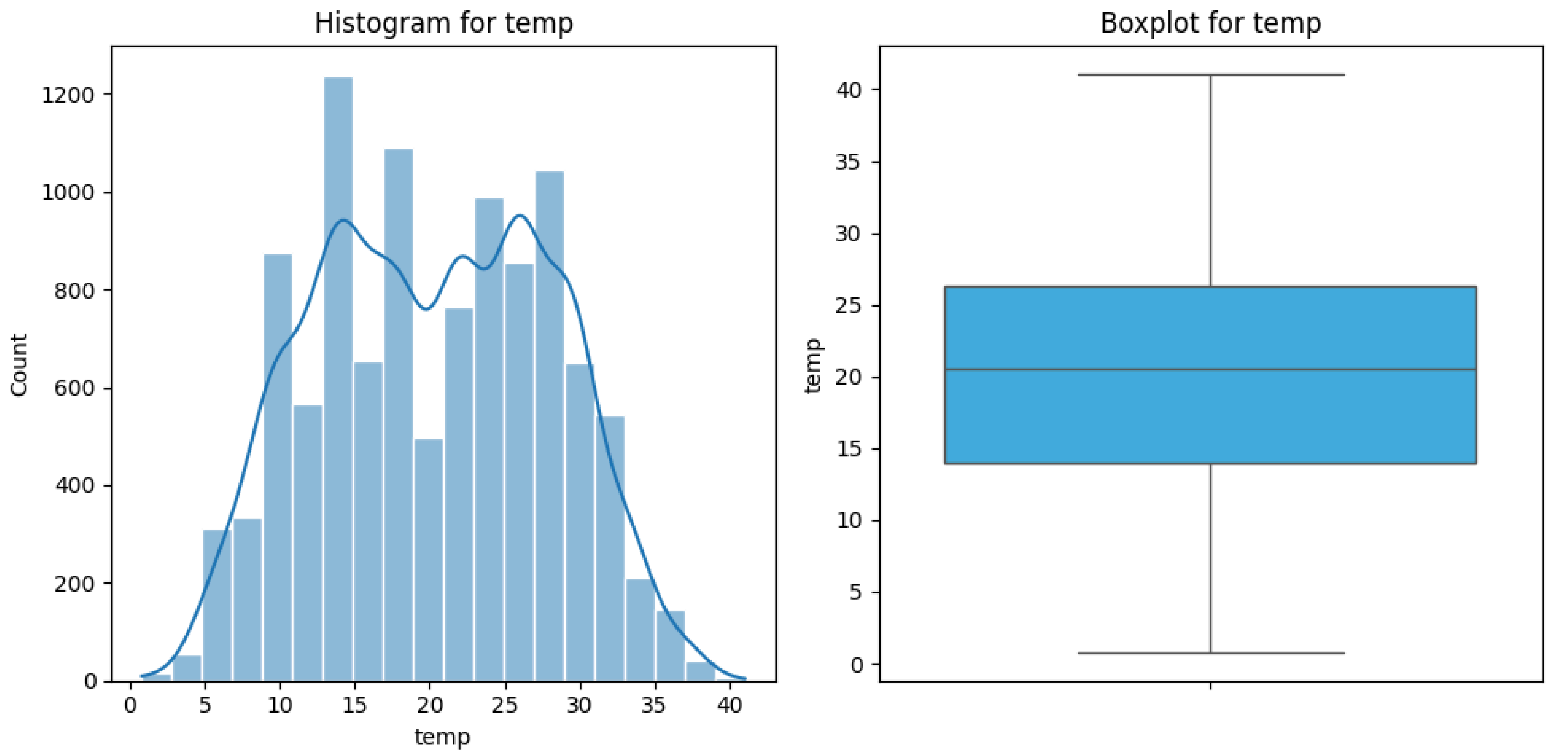
dtype: int64

In [33]:
```
# Function for histogram & boxplot on numerical columns

def hist_box(column):
    f, axs = plt.subplots(1, 2, figsize=(10, 5))
    sns.set(style="darkgrid")

    # Histogram
    plt.subplot(1, 2, 1)
    sns.histplot(bike_df[column], bins=20, kde=True)
    plt.title(f'Histogram for {column}')

    # Boxplot
    plt.subplot(1, 2, 2)
    sns.boxplot(bike_df[column], color="#29B6F6")
    plt.title(f'Boxplot for {column}')

    tabular_data = bike_df[column].describe().reset_index()
```

```
    tabular_data.columns = ['Statistic', 'Value']
    display(tabular_data)

    plt.tight_layout()
    plt.show()
```

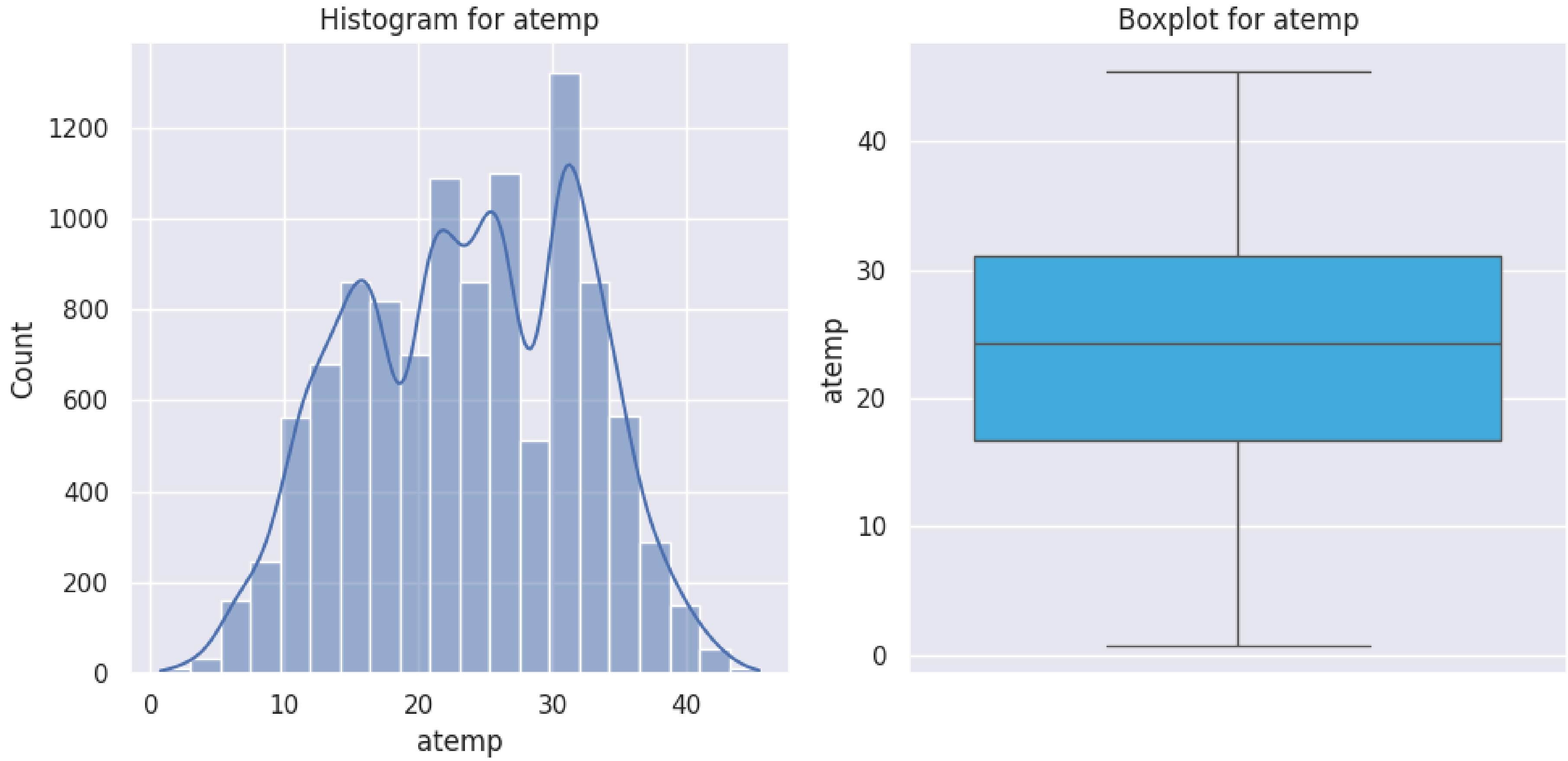
In [34]:
```
num_col = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']

for column in num_col:
    hist_box(column)
```

	Statistic	Value
0	count	10886.00000
1	mean	20.23086
2	std	7.79159
3	min	0.82000
4	25%	13.94000
5	50%	20.50000
6	75%	26.24000
7	max	41.00000



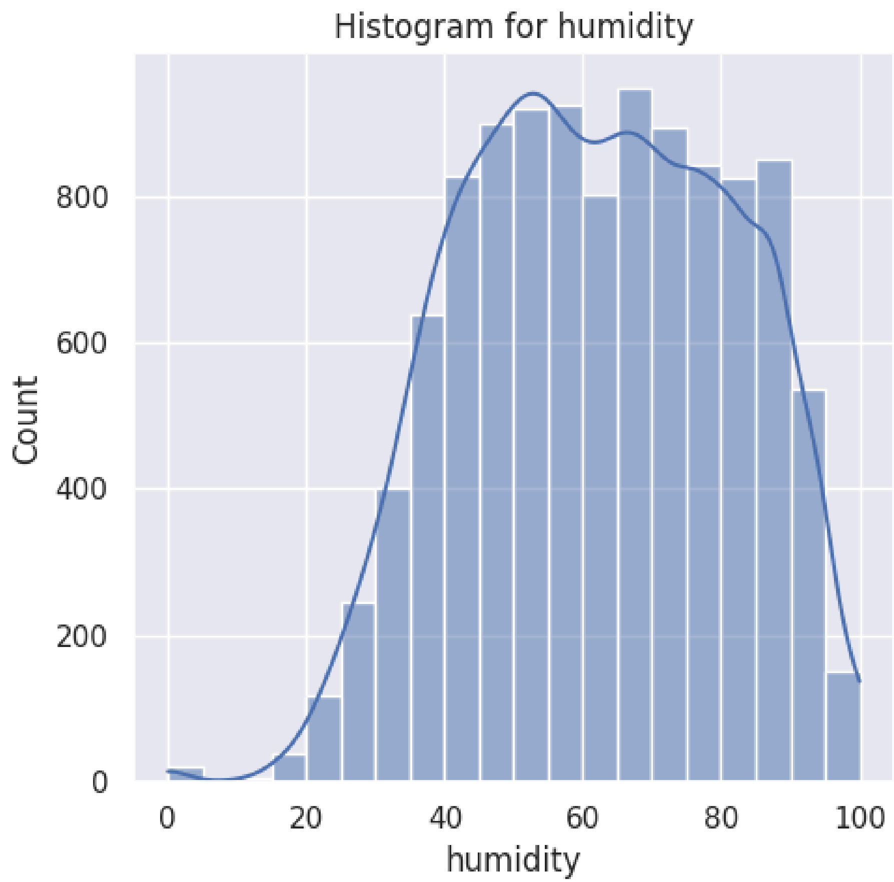
	Statistic	Value
0	count	10886.000000
1	mean	23.655084
2	std	8.474601
3	min	0.760000
4	25%	16.665000
5	50%	24.240000

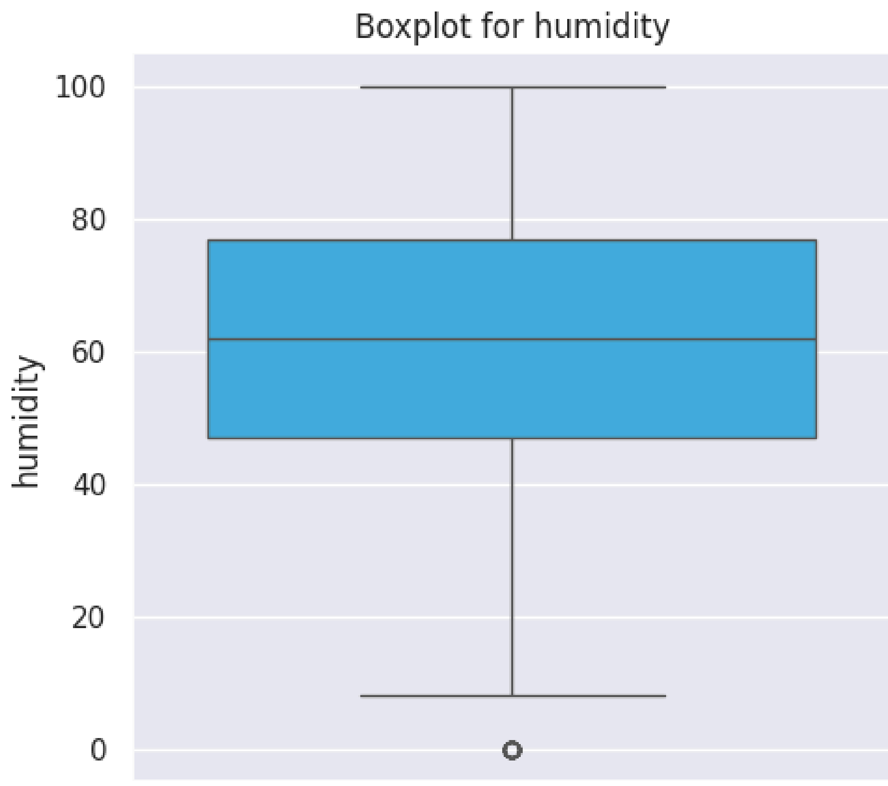
6	75%	31.060000
7	max	45.455000



	Statistic	Value
0	count	10886.000000
1	mean	61.886460
2	std	19.245033

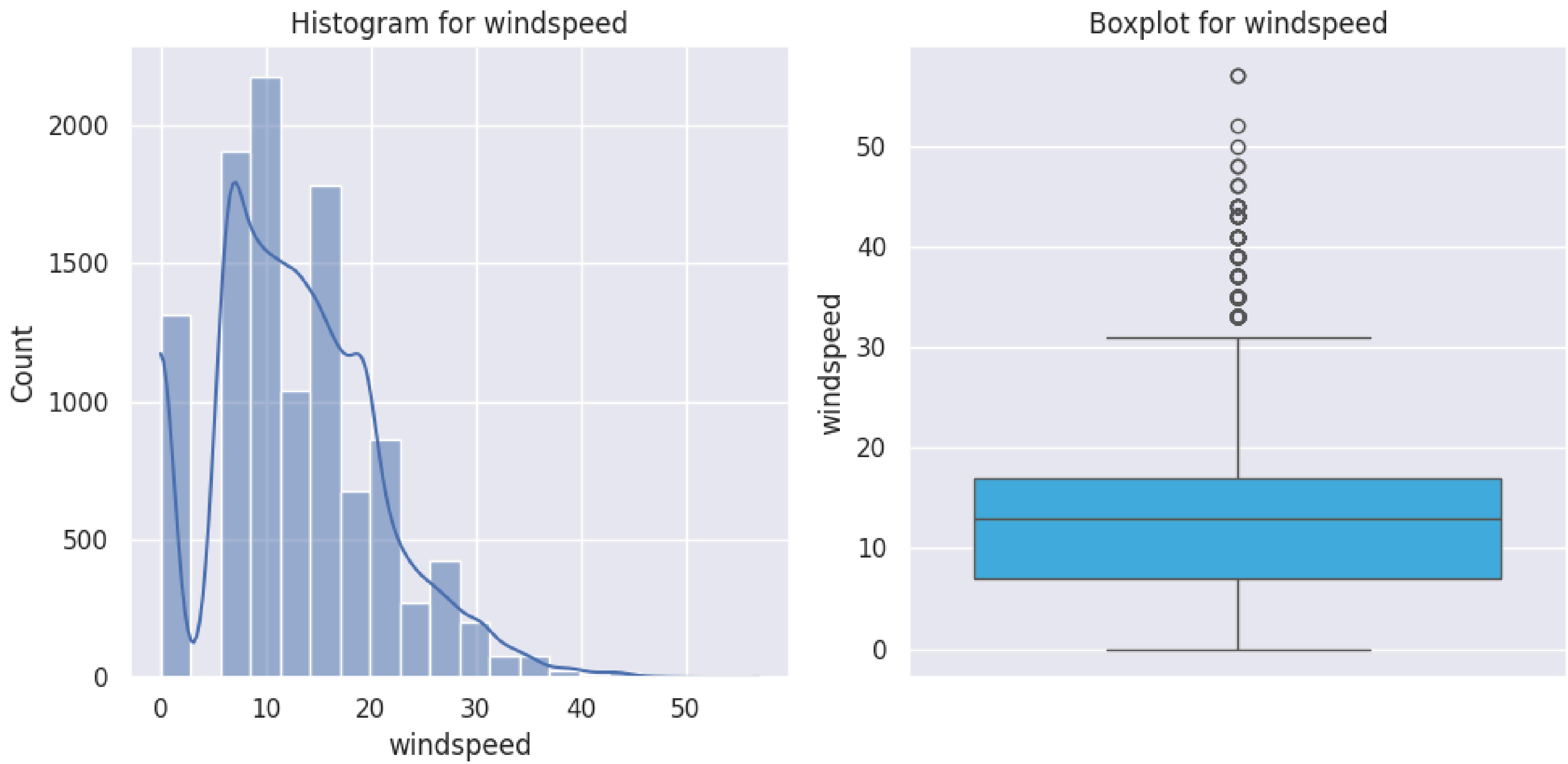
3	min	0.000000
4	25%	47.000000
5	50%	62.000000
6	75%	77.000000
7	max	100.000000





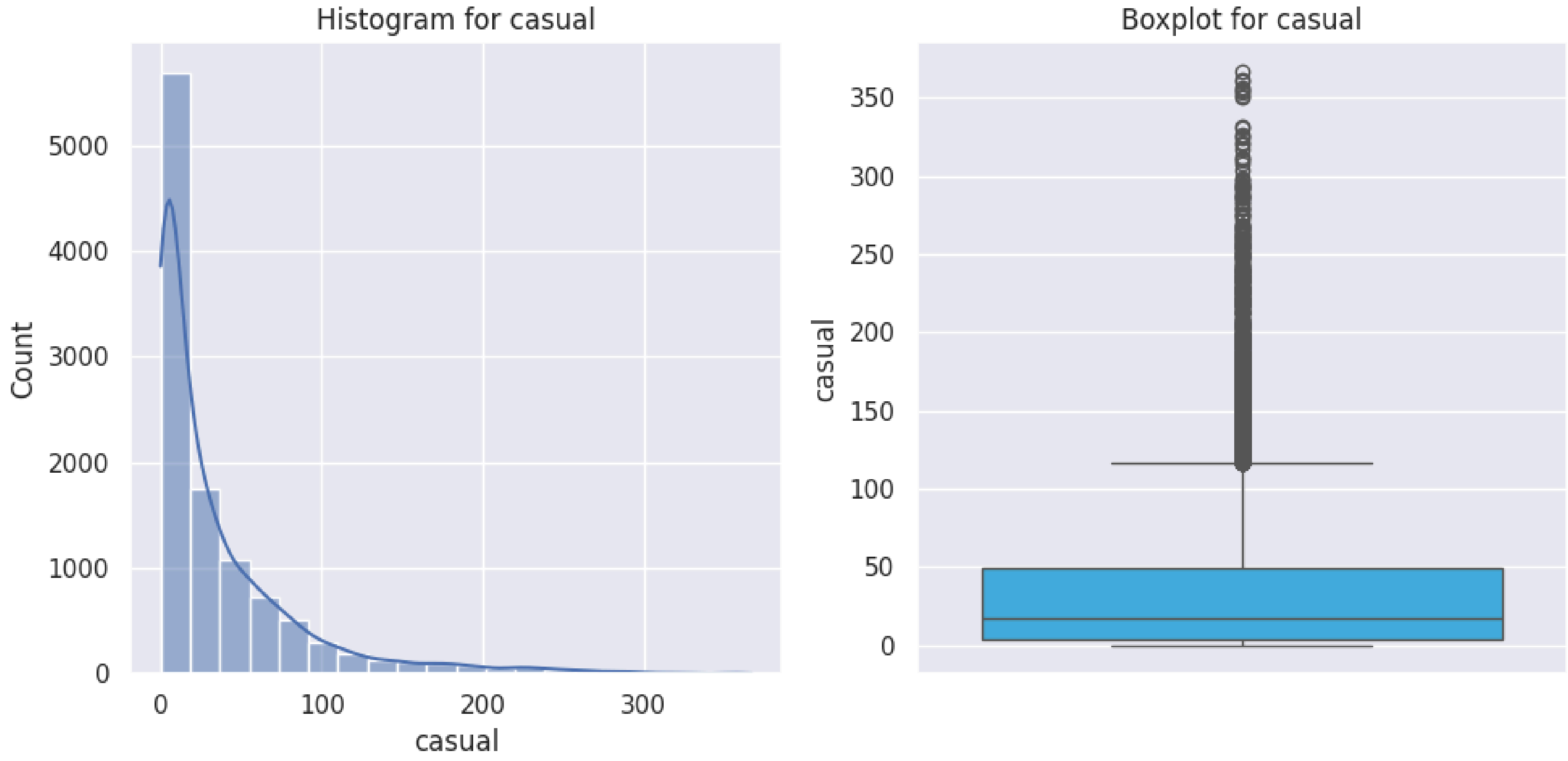
Statistic	Value

0	count	10886.000000
1	mean	12.799395
2	std	8.164537
3	min	0.000000
4	25%	7.001500
5	50%	12.998000
6	75%	16.997900
7	max	56.996900



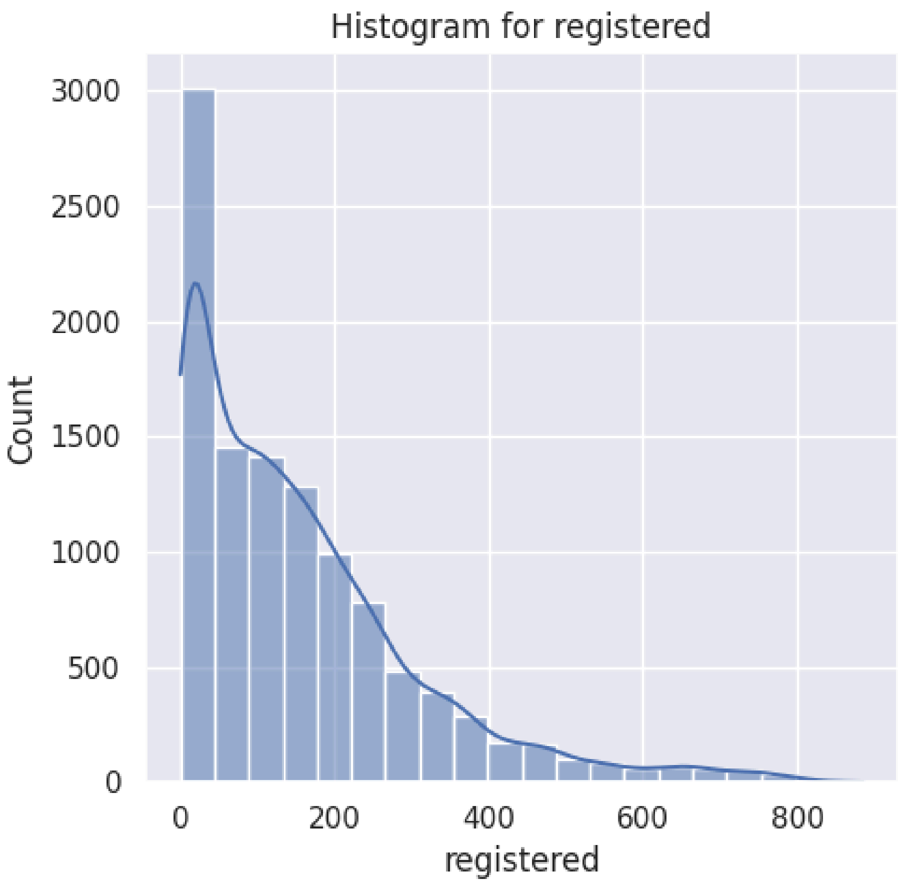
	Statistic	Value
0	count	10886.000000
1	mean	36.021955
2	std	49.960477
3	min	0.000000
4	25%	4.000000
5	50%	17.000000

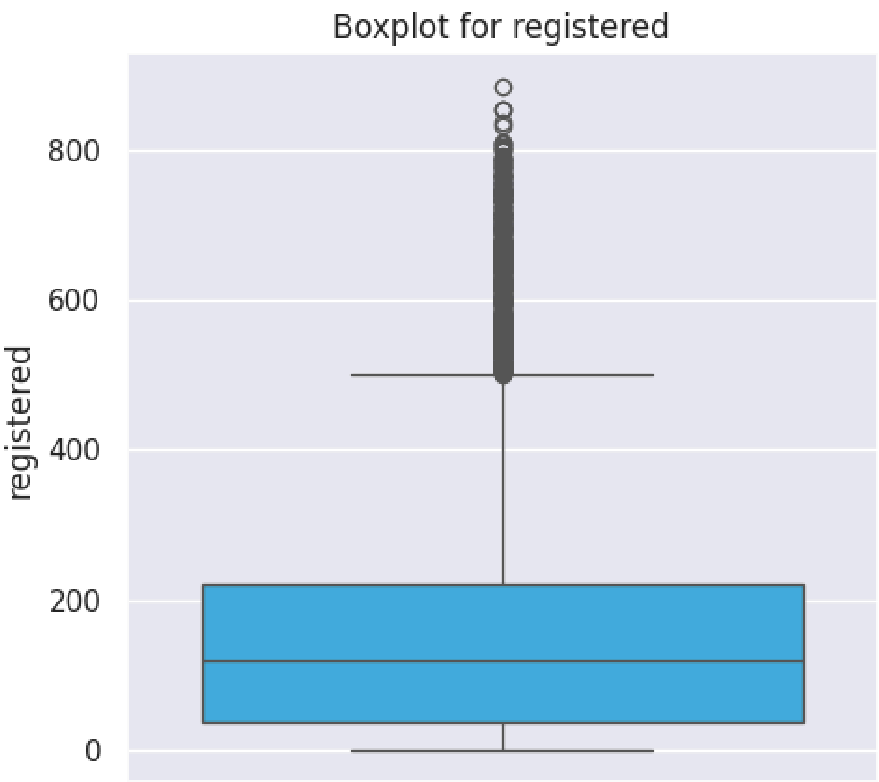
6	75%	49.000000
7	max	367.000000



	Statistic	Value
0	count	10886.000000
1	mean	155.552177
2	std	151.039033

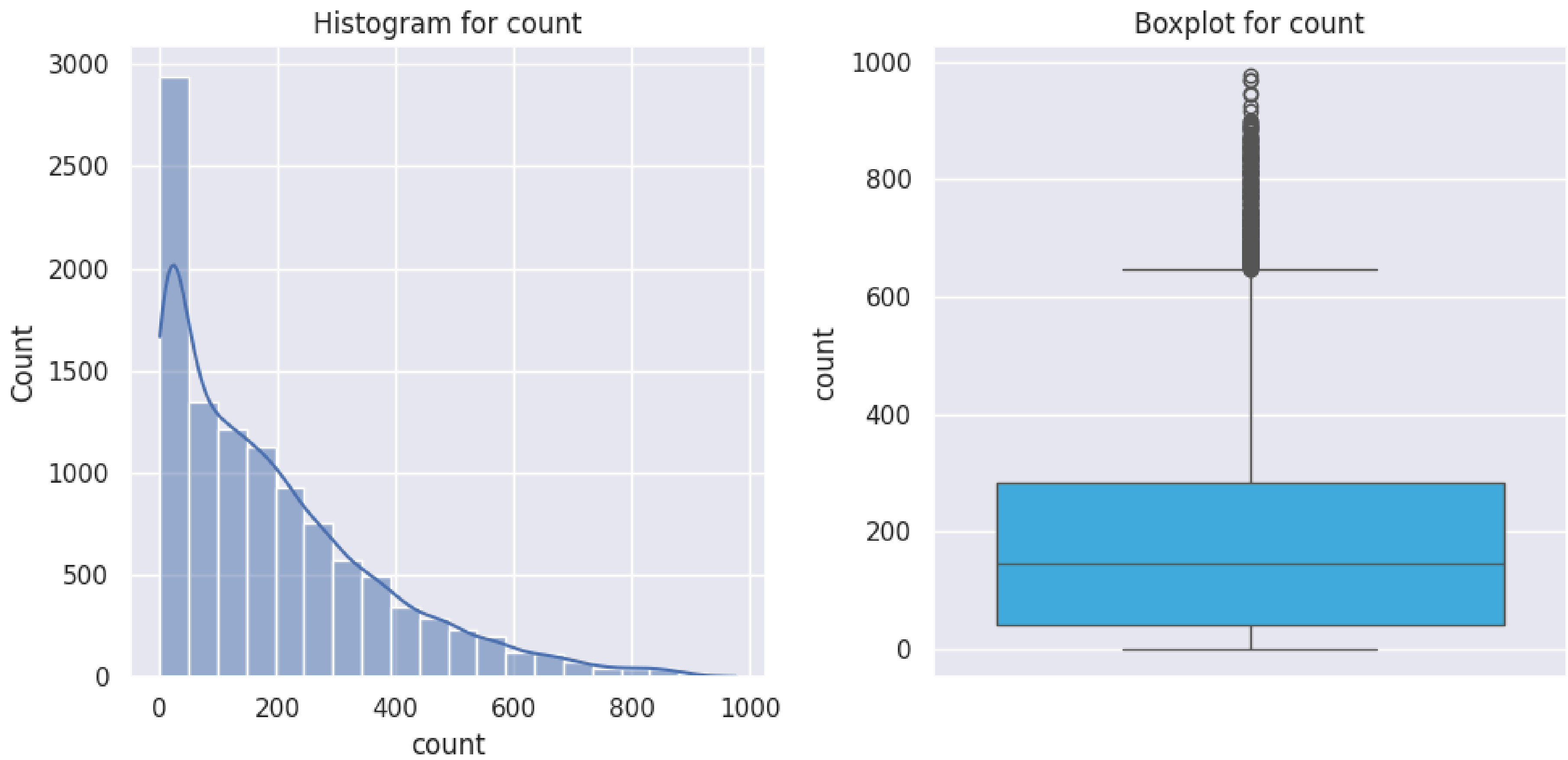
3	min	0.000000
4	25%	36.000000
5	50%	118.000000
6	75%	222.000000
7	max	886.000000





Statistic	Value

0	count	10886.000000
1	mean	191.574132
2	std	181.144454
3	min	1.000000
4	25%	42.000000
5	50%	145.000000
6	75%	284.000000
7	max	977.000000



Numerical column analysis

Temp:

- The 'temp' column shows a diverse temperature range (0.82 to 41.0), with a median of 20.5 and moderate variability around the mean of approximately 20.23 degrees Celsius.

Atemp

- The 'atemp' column displays a wide range of apparent temperatures (0.76 to 45.455), with a mean of approximately 23.66 and moderate variability around the median of 24.24.

Humidity

- The 'humidity' column depicts a range of humidity values (0 to 100), with an average around 61.89. The distribution shows moderate variability, from 47 at the 25th percentile to 77 at the 75th percentile, indicating diverse humidity levels in the dataset.

WindSpeed

- The 'windspeed' column displays a range of wind speeds from 0 to 56.9979, with a mean of approximately 12.80.

Casual

- The 'casual' column demonstrates a broad range of casual bike rental counts, with values spanning from 0 to 367. The distribution is positively skewed, as indicated by the mean (36.02) being less than the median (17.0).

Registered

- The 'registered' column showcases a diverse range of registered bike rental counts, ranging from 0 to 886. The distribution is positively skewed, evidenced by the mean (155.55) being less than the median (118.0).

Count

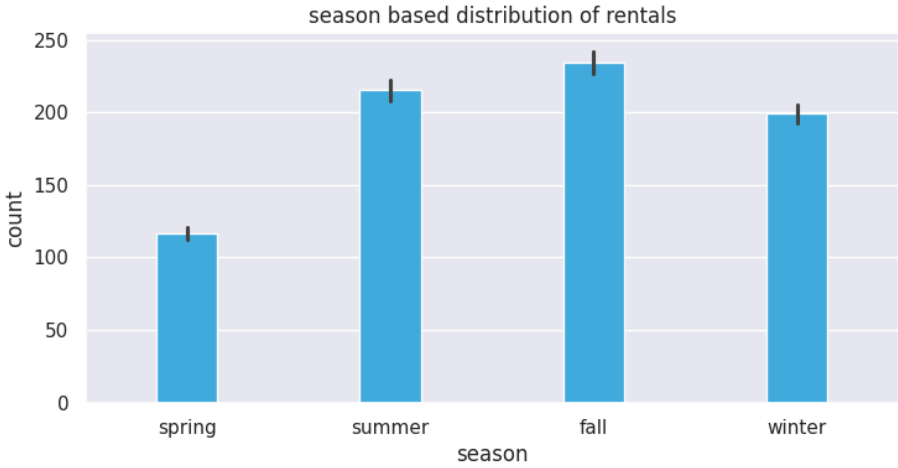
- The 'count' column reveals a wide range of total bike rental counts, varying from 1 to 977. The distribution is positively skewed, with a mean (191.57) greater than the median (145.0), indicating a concentration of lower values

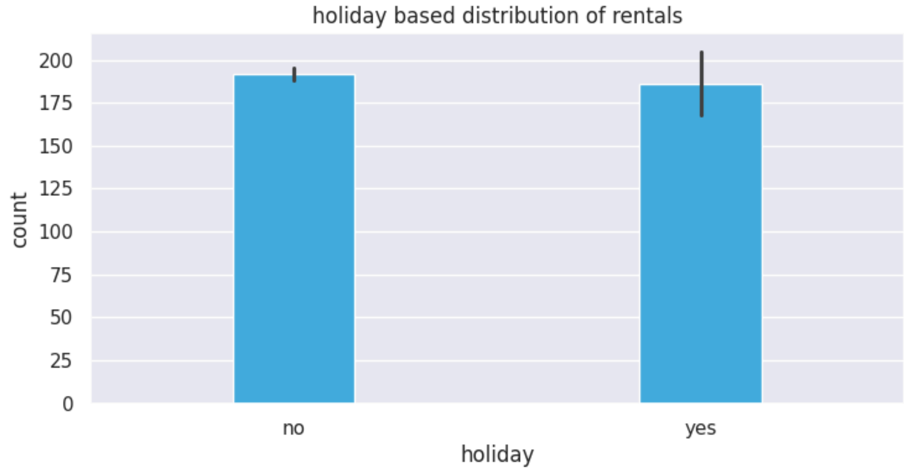
Bivariate Analysis

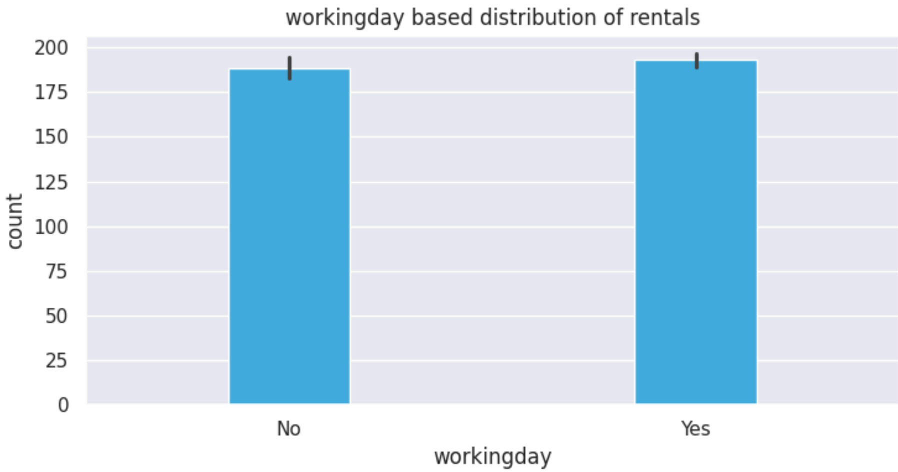
In [35]:
```
subset_col
```

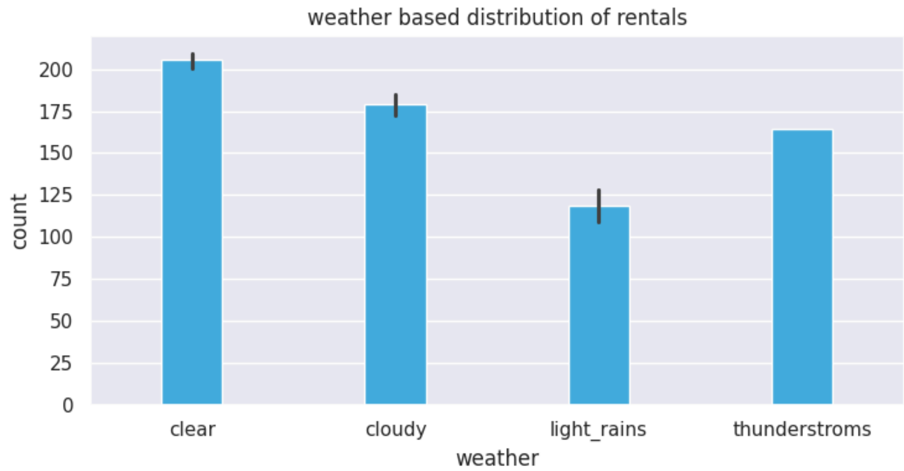
Out[35]:
```
['season', 'holiday', 'workingday', 'weather']
```

In [36]:
```
plt.figure(figsize=(15, 8))

for i, column in enumerate(subset_col, 1):
  plt.subplot(2, 2, i)
  sns.barplot(data= bike_df, x=column, y = "count",color="#29B6F8", width = 0.3)
  plt.title(f'{column} based distribution of rentals')

plt.tight_layout()
plt.show()
```









In [37]:

```
# corrrelation analysis

correlation_matrix = bike_df[["atemp", "temp", "humidity", "windspeed", "casual", "registered", "count"]].corr()
correlation_df = pd.DataFrame(correlation_matrix)
correlation_df
```

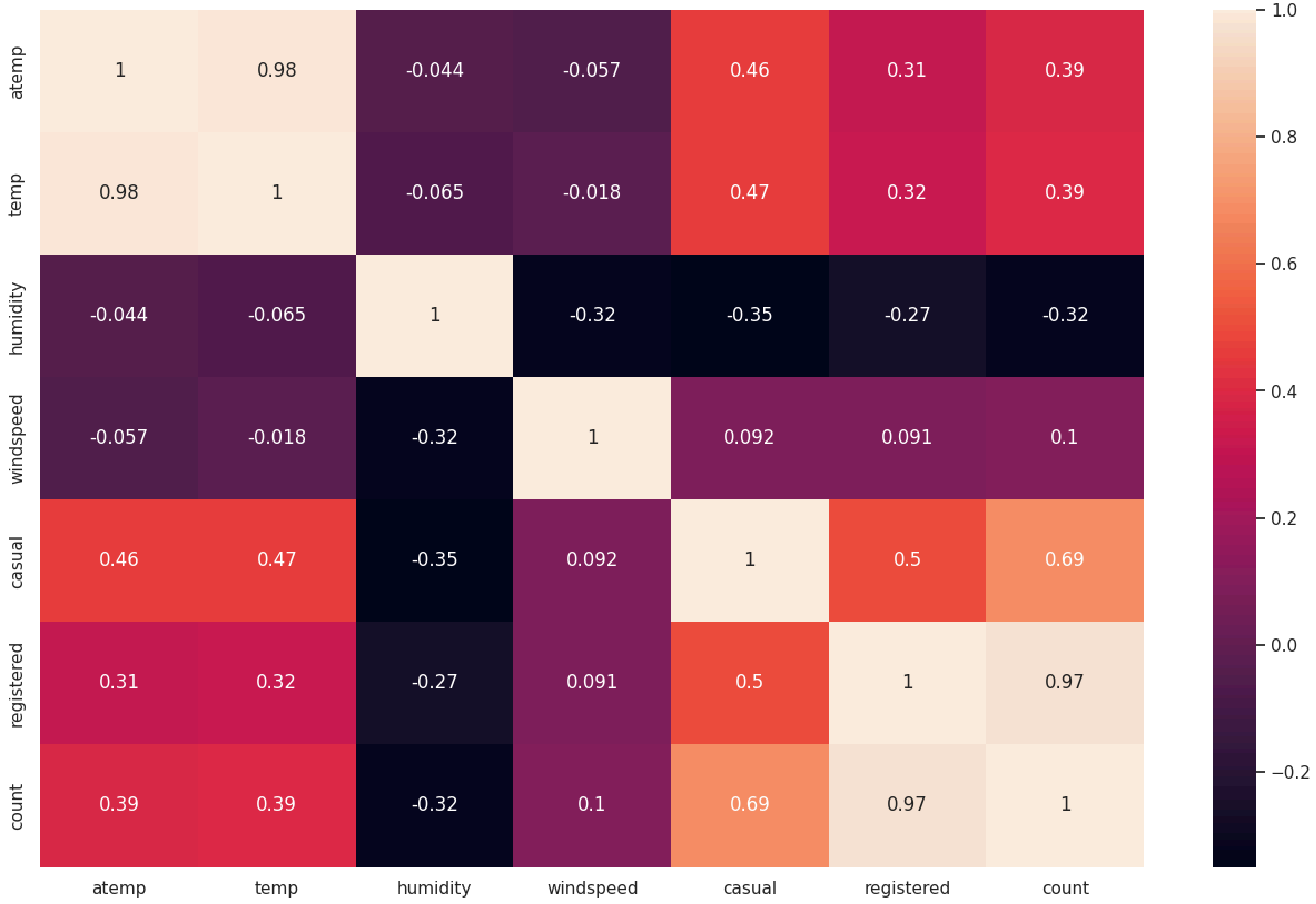
Out[37]:

	atemp	temp	humidity	windspeed	casual	registered	count
atemp	1.000000	0.984948	-0.043536	-0.057473	0.462067	0.314635	0.389784

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

In [38]:

```
# correlation chart

plt.figure(figsize = (16, 10))
sns.heatmap(correlation_matrix, annot = True)
plt.show()
```



Correlation Analysis

Atemp:

- Strong positive correlation with 'temp' (0.98), indicating a close relationship.

Moderate positive correlation with 'casual' (0.46) and 'registered' (0.31). Positive correlation with 'count' (0.39), suggesting a relationship with overall bike rentals.

Temp (Temperature):

- Highly correlated with 'atemp' (0.98), indicating a strong connection.

Moderate positive correlation with 'casual' (0.47) and 'registered' (0.32). Positive correlation with 'count' (0.39), showing a relationship with overall bike rentals.

Humidity:

- Weak negative correlation with 'atemp' (-0.04) and 'temp' (-0.06).

Moderate negative correlation with 'casual' (-0.35), 'registered' (-0.27), and 'count' (-0.32). Indicates a tendency for fewer bike rentals during higher humidity.

Windspeed:

- Weak negative correlation with 'atemp' (-0.06) and 'temp' (-0.02).

Weak positive correlation with 'casual' (0.09), 'registered' (0.09), and 'count' (0.10). Suggests a subtle influence on bike rentals with increasing wind speed.

Casual (Casual Bike Rentals):

- Strong positive correlation with 'atemp' (0.46) and 'temp' (0.47).
- Moderate negative correlation with 'humidity' (-0.35) and positive - correlation with 'windspeed' (0.09).
- Highly correlated with 'registered' (0.50) and 'count' (0.69), indicating a significant impact on overall rentals.

Registered (Registered Bike Rentals):

- Positive correlation with 'atemp' (0.31) and 'temp' (0.32).
- Negative correlation with 'humidity' (-0.27) and positive correlation with 'windspeed' (0.09).
- Highly correlated with 'casual' (0.50) and 'count' (0.97), emphasizing a substantial impact on overall rentals.

Count (Total Bike Rentals):

- Positive correlation with 'atemp' (0.39), 'temp' (0.39), and 'casual' (0.69).
- Negative correlation with 'humidity' (-0.32).
- Highly correlated with 'registered' (0.97), emphasizing the joint impact of casual and registered rentals on the overall count.

In [39]:
```
monthly_count = bike_df.groupby('month')['count'].sum().reset_index()

monthly_count = monthly_count.sort_values(by='count', ascending=False)

monthly_count
```

Out[39]:

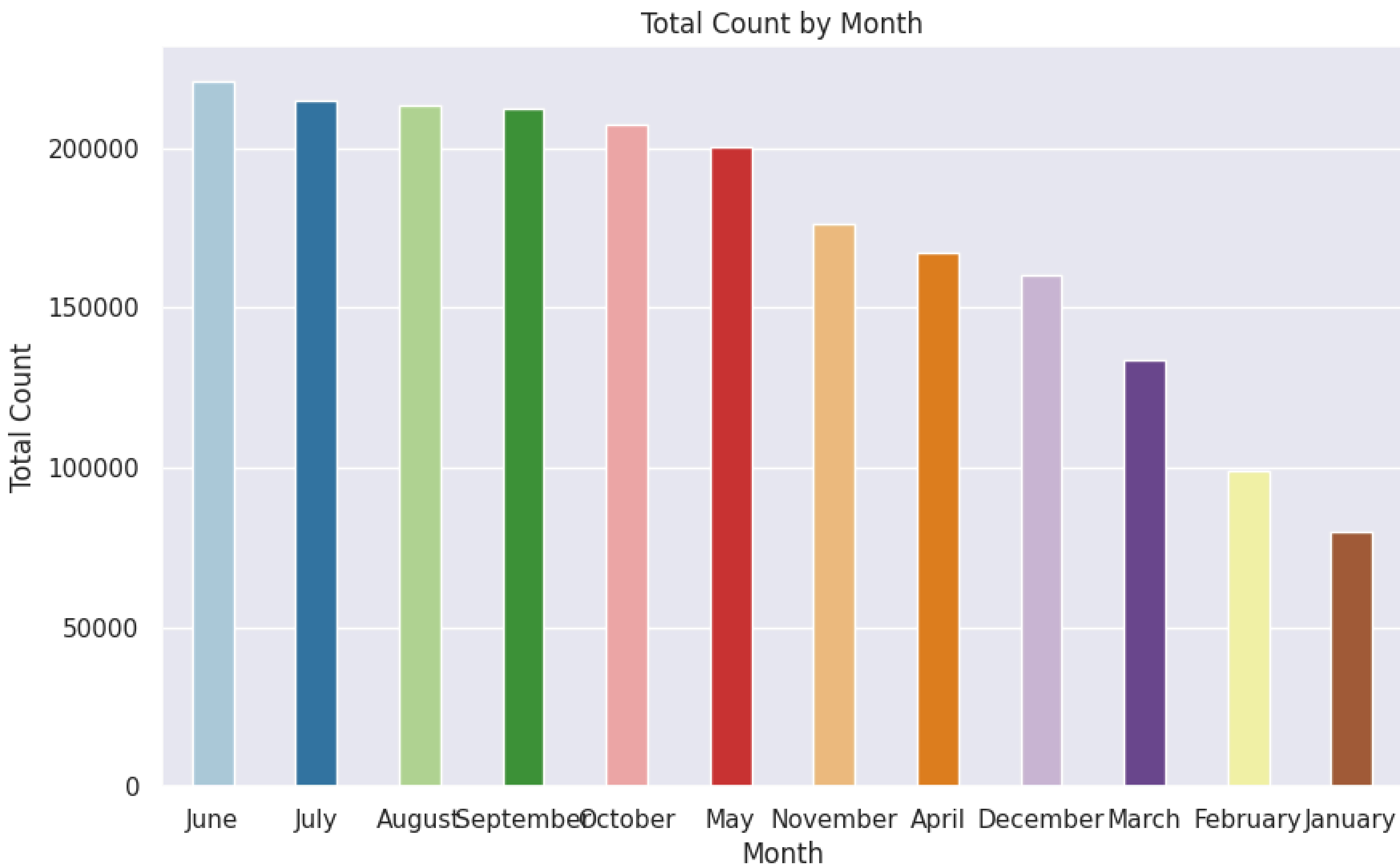
	month	count
6	June	220733
5	July	214617
1	August	213516

11	September	212529
10	October	207434
8	May	200147
9	November	176440
0	April	167402
2	December	160160
7	March	133501
3	February	99113
4	January	79884

In [40]:

```
# rentals on monthly counts

plt.figure(figsize=(10, 6))
sns.barplot(x='month', y='count', data=monthly_count, palette='Paired', width = 0.4)

plt.title('Total Count by Month')
plt.xlabel('Month')
plt.ylabel('Total Count')
plt.show()
```

```
<ipython-input-40-a98c35e383bc>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `
hue` and set `legend=False` for the same effect.

  sns.barplot(x='month', y='count', data=monthly_count, palette='Paired', width = 0.4)
```



Monthly analysis on rentals

Peak Rental Months:

- June stands out as the peak month for bike rentals, with the highest count of 220,733, followed closely by July and August.

Seasonal Trend:

- Summer months (June, July, August) show higher bike rental counts, consistent with favorable weather conditions.

Off-Peak Rental Months:

- January, February, and March have notably lower bike rental counts, indicating potential off-peak periods, possibly influenced by colder weather or fewer outdoor activities.

Hypothesis Testing

Demand of bicycles on rent is the same on Weekdays & Weekends

- Since we have two independent samples, we can go with Two Sample Independent T-Test.
- Assumptions of Two Sample Independent T-Test :
- The data should be normall distributed
- variances of the two groups are equal
- Let the Confidence interval be 95%, so siginificance (alpha) is 0.05

To check if the data is normal, we will go with Wilkin-ShapiroTest.

The test hypothesis for the Wilkin-Shapiro test are:

- Ho: Data is normally distributed

- Ha: Data is not normally distributed.

In [41]:
```
np.random.seed(41)

df_subset = bike_df.sample(100)["count"]

test_stat, p_val = shapiro(df_subset)

p_val
```

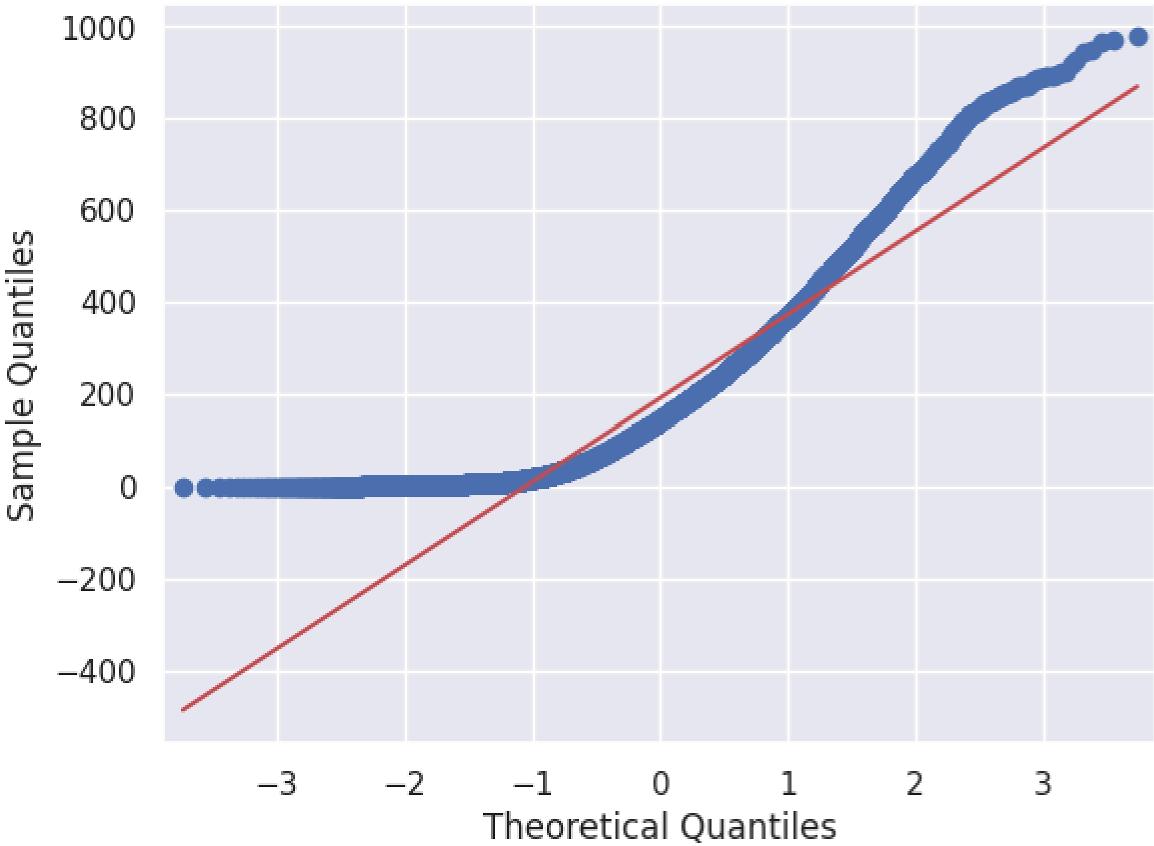
Out[41]: 2.6341210395843134e-07

- Hence the p_values is lesser than the significance level, Null hypothesis can be rejected.

- Therefore, the Data is not normally distributed.

QQ Plot analysis

In [43]:
```
qqplot(bike_df['count'], line = 's')
plt.show()
```



To check if the variances of two groups are equal. We will perform Levene's test

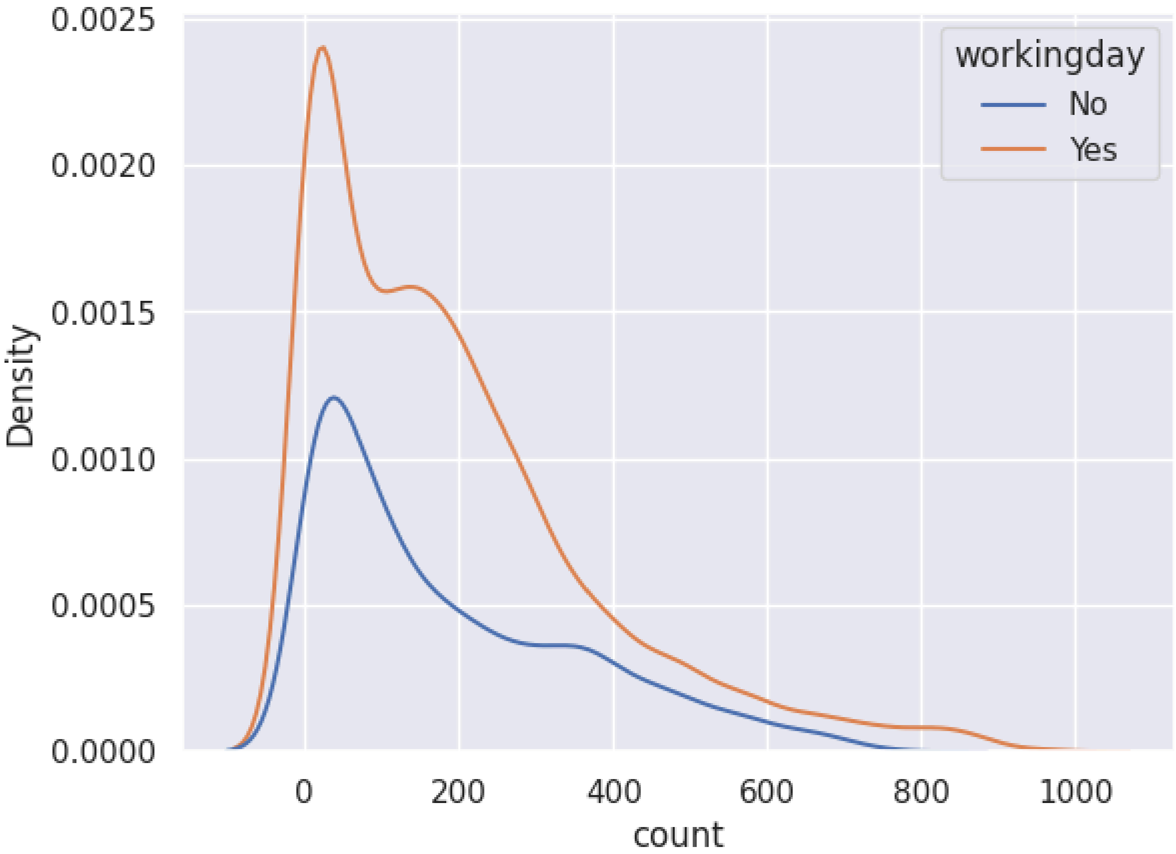
The Test hypotheses for Levene's test are:

- Ho: The variances are equal.
- Ha: The variances are not equal.

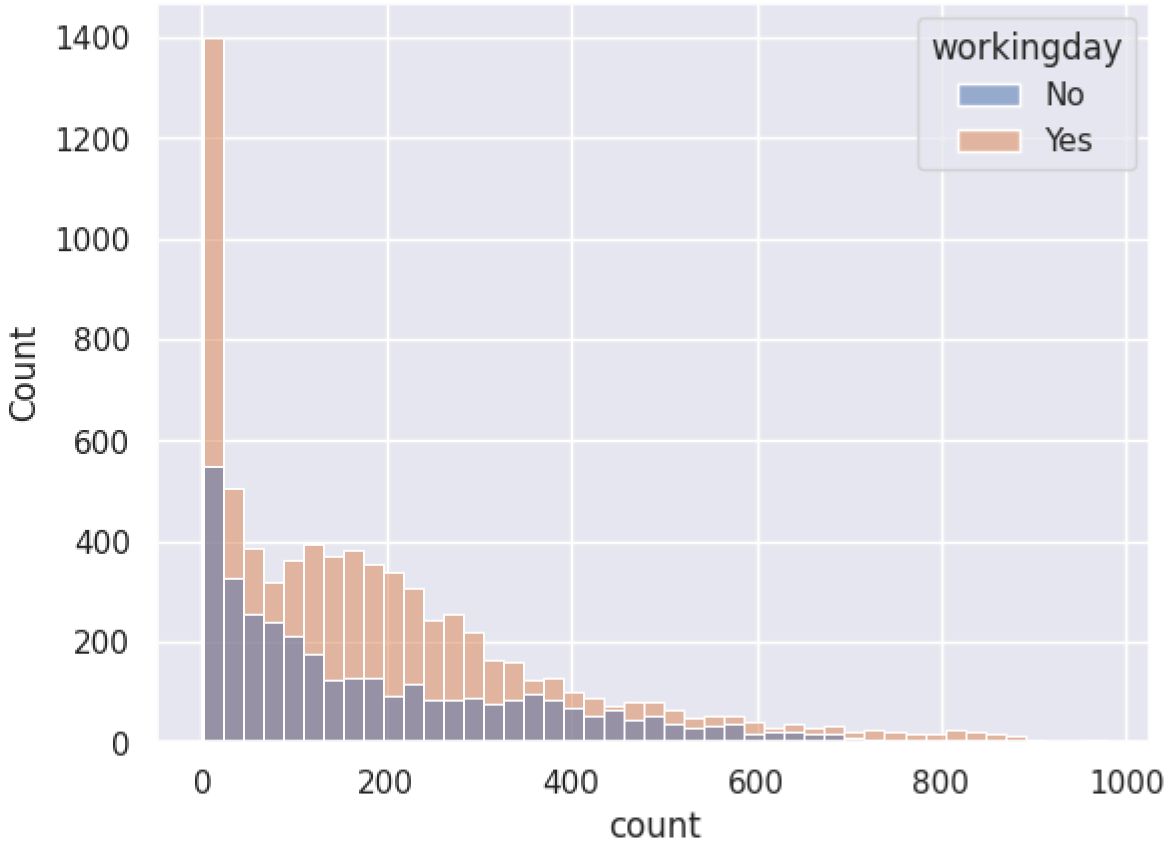
In [44]:
```
working_day = bike_df[bike_df['workingday'] == 'Yes']['count']

holiday = bike_df[bike_df['workingday'] == 'No']['count']

levene_stat, p_val = levene(working_day, holiday)

p_val
```

Out[44]:
```
0.9437823280916695
```

In [47]:
```
sns.kdeplot(data = bike_df, x = 'count', hue = 'workingday')
plt.show()
```



In [49]:
```
sns.histplot(data = bike_df, x = 'count', hue = 'workingday')
plt.show()
```



- Hence the p_values is greater than the significance level, Null hypothesis can be accepted.
- Therefore, the variances are approximately equal.

Despite the data is not normally distributed according to both the Wilkin-ShapiroTest and qq-plot

It is important to highlight that the variances between the two groups are equal

So we can proceed with the Two Sample Independent T-Test.

The hypothesis for the t-test are:

- Ho: There is no significant difference between working and non-working days.
- Ha: There is a significant difference between working and non-working days.

In [51]:
```
ttest_stat, p_val = ttest_ind(working_day, holiday)

p_val

#Hence the p_values is greater than the significance level, Null hypothesis can be accepted.

#Therefore, There is no significant difference on bike rentals between working and non-working days.
```

Out[51]: 0.22644804226361348

In [52]:
```
kruskal_stat, p_val = kruskal(working_day, holiday)

p_val
```

Out[52]: 0.9679113872727798

- Hence the p_values is greater than the significance level, Null hypothesis can be accepted.
- Therefore, There is no significant difference on bike rentals between working and non-working days.

Demand of bicycles on rent is the same for different Weather conditions

Since we have more than two categories now, so will use ANOVA here.

Assumptions for ANOVA are:

The population data should be normally distributed- The data is not normal as verified by Wilkin-Shapiro test and the qqplot.

The data points must be independent- This condition is satisfied.

Approximately equal variance within groups- This will be verified using Levene's test.

In [53]:
```
# skewness of weather

bike_df.groupby('weather')['count'].skew()
```

Out[53]:

weather	count
clear	1.139857
cloudy	1.294444
light_rains	2.187137
thunderstroms	NaN

dtype: float64

In [55]:
```
# kurtosis test of weather

bike_df.groupby('weather')['count'].apply(lambda x: x.kurtosis())
```

Out[55]:

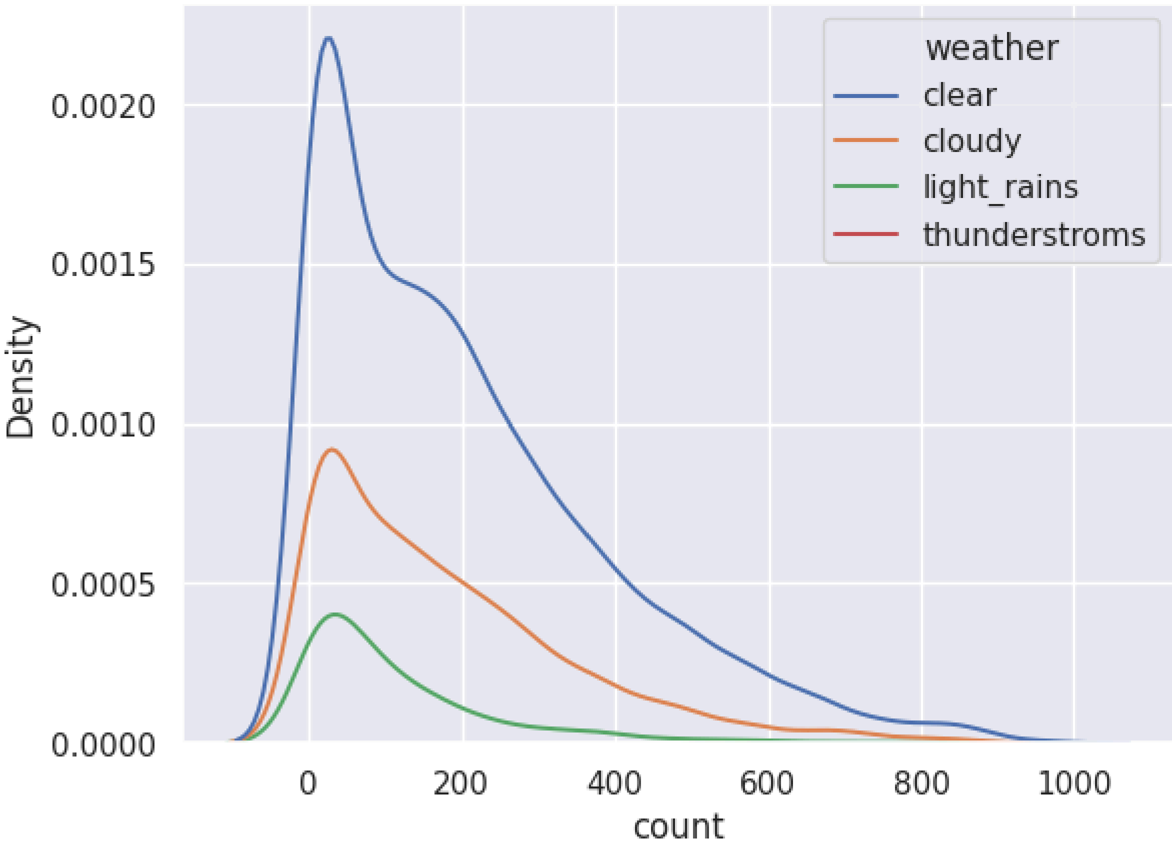
weather	count
clear	0.964720

cloudy	1.588430
light_rains	6.003054
thunderstroms	NaN

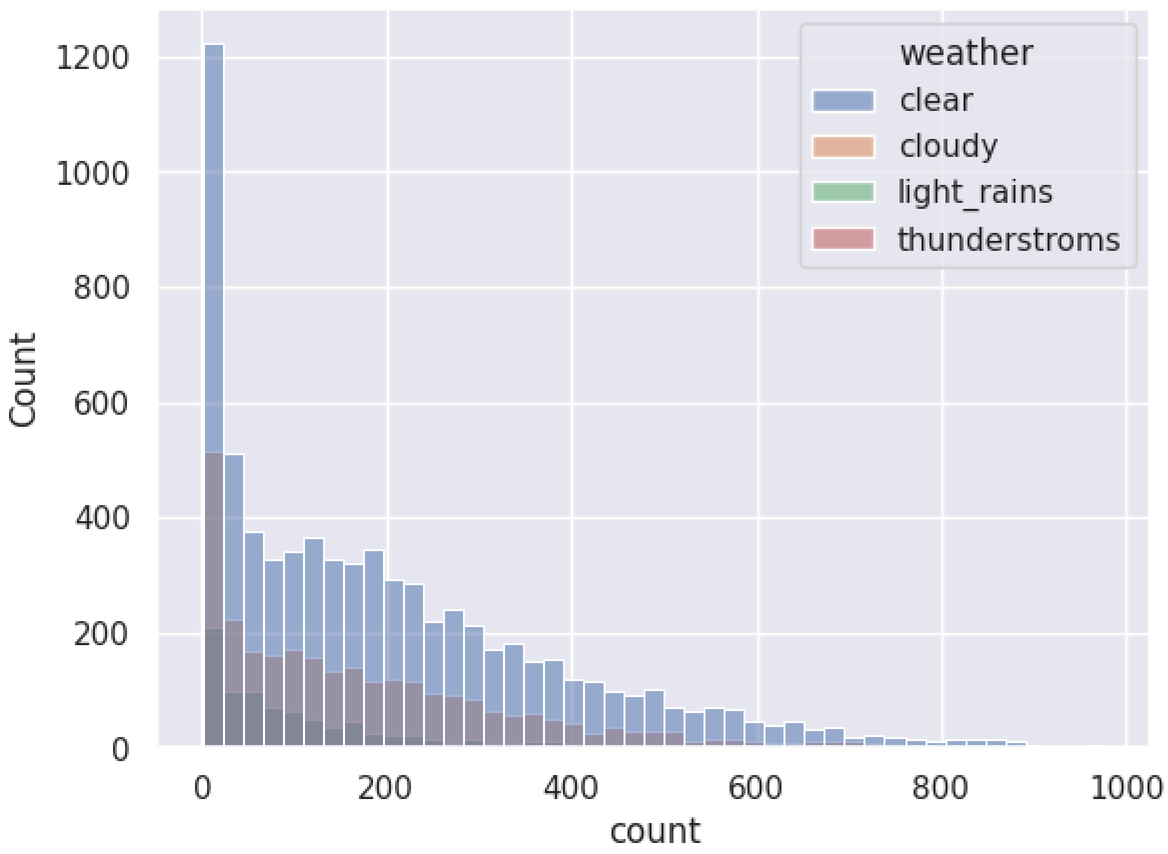
dtype: float64

In [57]:
```
sns.kdeplot(data = bike_df, x = 'count', hue = 'weather')
plt.show()
```

```
<ipython-input-57-e52da3c0f946>:1: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singul
ar=False` to disable this warning.
  sns.kdeplot(data = bike_df, x = 'count', hue = 'weather')
```



In [59]:
```
sns.histplot(data = bike_df, x = 'count', hue = 'weather')
plt.show()
```



The Test hypothesis for Levene's test are:

- Ho: The variances are equal.
- Ha: The variances are not equal.

In [62]:
```
bike_df["weather"].unique()
```

Out[62]:
```
array(['clear', 'cloudy', 'light_rains', 'thunderstroms'], dtype=object)
```

In [63]:
```
weather1 = bike_df[bike_df['weather'] == "clear"]['count']
weather2 = bike_df[bike_df['weather'] == "cloudy"]['count']
weather3 = bike_df[bike_df['weather'] == "light_rains"]['count']
```

```
weather4 = bike_df[bike_df['weather'] == "thunderstroms"]['count']

levene_stat, p_val = levene(weather1, weather2, weather3,weather4)

p_val
```

Out[63]: 3.504937946833238e-35

hence the p_values is smaller than the significance level, Null hypothesis can be rejected.

Therefore, the variances are not equal.

- Two of the three conditions of ANOVA are not met, We will still perform ANOVA.
- Then We will also perform Kruskal's test and compare the results.

In case of any discrepancies, Kruskal's test results will be considered, since data does not met conditions of ANOVA.

The hypothesis for ANOVA are:

- Ho: There is no significant difference between demand of bicycles for different Weather conditions.
- Ha: There is a significant difference between demand of bicycles for different Weather conditions.

In [64]:
```
anova_stat, p_val = f_oneway(weather1, weather2, weather3,weather4)

p_val
```

Out[64]: 5.482069475935669e-42

- Hence the p_values is smaller than the significance level, Null hypothesis can be rejected.
- Therefore, There is a significant difference between demand of bicycles for different Weather conditions.

Kruskal Test on weather

In [65]:
```
kruskal_stat, p_val = kruskal(weather1, weather2, weather3,weather4)

p_val
```

Out[65]:
```
3.501611300708679e-44
```

Again the p_values is smaller than the significance level, Null hypothesis can be rejected.

Therefore, we can conclude that there is a significant difference between demand of bicycles for different Weather conditions.

Demand of bicycles on rent is the same for different Seasons

Here also we have more than two categories now, so will use ANOVA here.

Assumptions for ANOVA are:

1. The population data should be normally distributed- The data is not normal as verified by **Wilkin-Shapiro test and the qqplot**
2. The data points must be independent- This condition is satisfied.
3. Approximately equal variance within groups- This will be verified using **Levene's test.**

In [66]:
```
# skewness of seasons

bike_df.groupby('season')['count'].skew()
```

Out[66]:

	count
season	

fall	0.991495
spring	1.888056
summer	1.003264
winter	1.172117

dtype: float64

In [68]:
```
# kurtosis test of seasons

bike_df.groupby('weather')['count'].apply(lambda x: x.kurtosis())
```

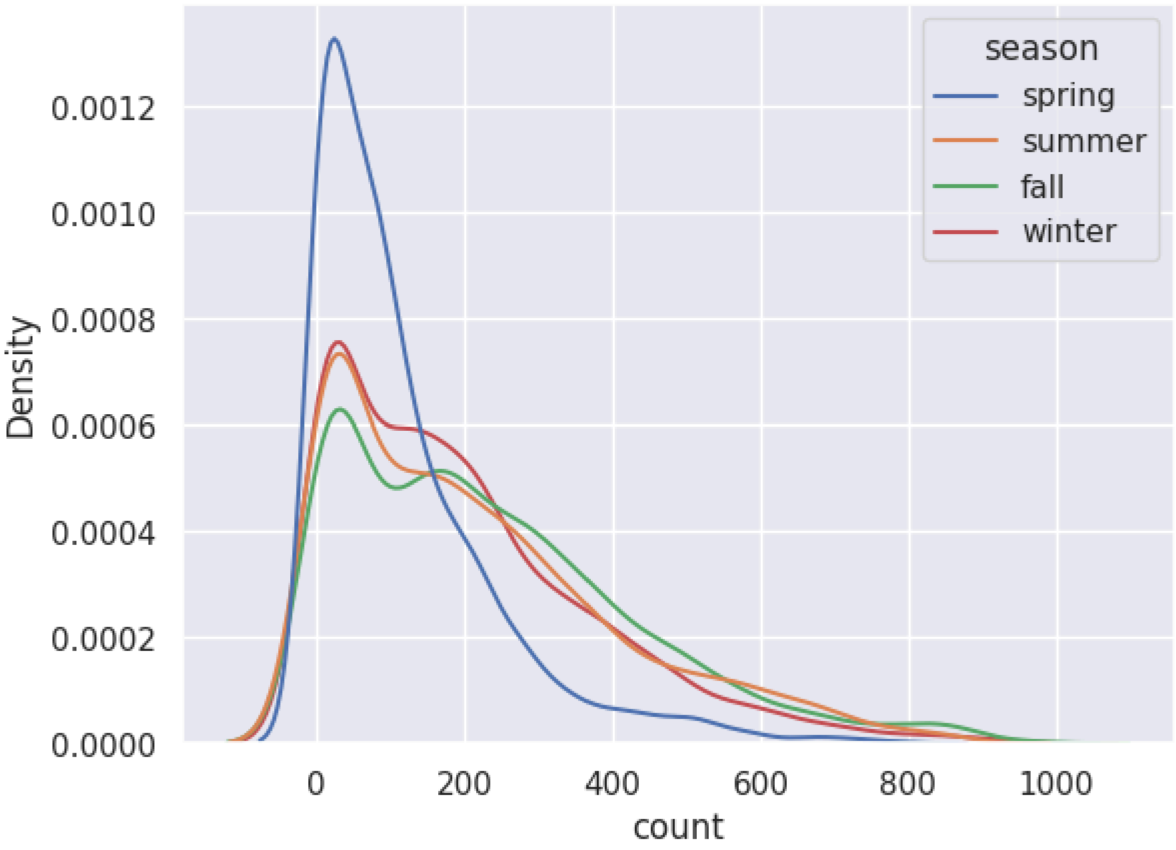
Out[68]:

	count
weather	
clear	0.964720
cloudy	1.588430
light_rains	6.003054
thunderstroms	NaN

dtype: float64

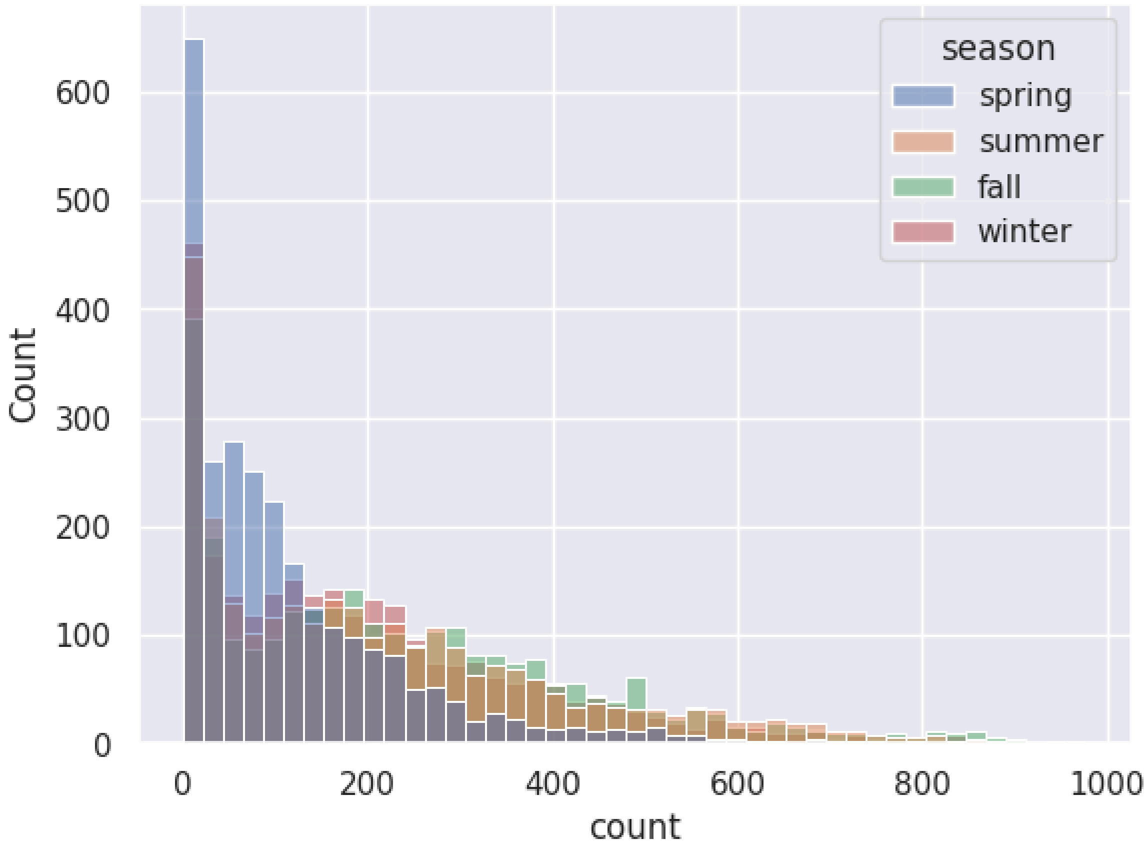
In [69]:
```
sns.kdeplot(data = bike_df, x = 'count', hue = 'season')
```

Out[69]:
```
<Axes: xlabel='count', ylabel='Density'>
```



In [70]:
```
sns.histplot(data = bike_df, x = 'count', hue = 'season')
```

Out[70]:
```
<Axes: xlabel='count', ylabel='Count'>
```



The Test hypothesis for Levene's test are:

- Ho: The variances are equal.
- Ha: The variances are not equal.

In [74]:
```
bike_df["season"].unique()
```

Out[74]: `array(['spring', 'summer', 'fall', 'winter'], dtype=object)`

In [75]:
```
spring = bike_df[bike_df['season'] == 'spring']['count']
summer = bike_df[bike_df['season'] == 'summer']['count']
```

```
fall = bike_df[bike_df['season'] == 'fall']['count']
winter = bike_df[bike_df['season'] == 'winter']['count']

levene_stat, p_val = levene(spring,summer,fall,winter)

p_val
```

Out[75]: 1.0147116860043298e-118

Hence the p_values is smaller than the significance level, Null hypothesis can be rejected.

Therefore, the variances are not equal.

As like before, we still use both ANOVA and Kruskal's test, comparing the results.

If discrepancies arise, we'll rely on **Kruskal's test**, Since data does not met the conditions for ANOVA.

The hypothesis for ANOVA are:

- **Ho: There is no significant difference between demand of bicycles for different Seasons**
- **Ha: There is a significant difference between demand of bicycles for different Seasons.**

In [76]:
```
anova_stat, p_val = f_oneway(spring ,summer, fall, winter)

p_val
```

Out[76]: 6.164843386499654e-149

Hence the p_values is smaller than the significance level, Null hypothesis can be rejected.

Therefore, There is a significant difference between demand of bicycles for different Seasons.

Kruskal Test on season

In [77]:
```
kruskal_stat, p_val = kruskal(spring ,summer, fall, winter)

p_val
```

Out[77]: 2.479008372608633e-151

Again the p_values is smaller than the significance level, Null hypothesis can be rejected.

Therefore, we can conclude that there is a significant difference between demand of bicycles for different Seasons.

Analysis of Weather Conditions Across Seasons using Chi-square Test

The hypothesis for the chi-square test are:

Ho: Season and Weather are independent of each other.

Ha: Season and Weather are dependent on each other.

In [78]:
```
contingency_table = pd.crosstab(bike_df['weather'], bike_df['season'])

contingency_table
```

Out[78]:

season	fall	spring	summer	winter
weather				
clear	1930	1759	1801	1702
cloudy	604	715	708	807
light_rains	199	211	224	225
thunderstroms	0	1	0	0

In [79]:
```
chi2_contingency(contingency_table)
```

Out[79]:
```
Chi2ContingencyResult(statistic=49.15865559689363, pvalue=1.5499250736864862e-07, dof=9, expected_freq=array([[1.80
559765e+03, 1.77454639e+03, 1.80559765e+03, 1.80625831e+03],
       [7.11493845e+02, 6.99258130e+02, 7.11493845e+02, 7.11754180e+02],
       [2.15657450e+02, 2.11948742e+02, 2.15657450e+02, 2.15736359e+02],
       [2.51056403e-01, 2.46738931e-01, 2.51056403e-01, 2.51148264e-01]]))
```

Hence the p_values(1.5499250736864862e-07) is smaller than the significance level, Null hypothesis can be rejected.

Therefore, we can conclude that Season and Weather are dependent on each other.

Strategic Recommendations for Yulu's Profitable Growth

Optimize Bike Distribution in Peak Months:

- Concentrate bike deployment efforts during peak months, especially in June, July, and August, to meet increased demand and capitalize on favorable weather conditions.

Seasonal Marketing Strategies:

- Tailor marketing efforts to leverage the seasonal trend, promoting Yulu's services more aggressively during summer months to attract a larger user base.

Enhance User Engagement in Off-Peak Months:

- Implement targeted promotional campaigns or discounts during off-peak months (e.g., January to March) to encourage increased bike rentals and maintain consistent revenue flow.

Weather-Responsive Pricing:

- Consider implementing dynamic pricing strategies that respond to weather conditions. For example, adjusting rental rates during extreme weather days to optimize revenue.

Diversify Revenue Streams:

- Explore additional revenue streams, such as partnerships, sponsorships, or offering premium membership services with added benefits, to diversify income sources and boost overall profitability.

Optimize Bike Deployment on Working Days:

- Given the lack of significant differences in bike rentals between working and non-working days, consider adjusting bike deployment strategies to ensure optimal resource allocation throughout the week.

Combine Season and Weather Plans:

- Plan bike availability based on both the season and the weather to make sure people have the bikes they need when they want them. For example, have more bikes available on sunny days in the summer.

In [ ]:

```
```