

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
```

```
csv_path = "yulu dataset.txt"
df = pd.read_csv(csv_path, delimiter=",")
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
# no of rows amd columns in dataset
print(f'# rows: {df.shape[0]} \n# columns: {df.shape[1]}")
```

```
# rows: 10886
# columns: 12
```

```
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
```

```
df['datetime'] = pd.to_datetime(df['datetime'])
```

```
cat_cols= ['season', 'holiday', 'workingday', 'weather']
for col in cat_cols:
    df[col] = df[col].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
```

```
df.iloc[:, 1:].describe(include='all')
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered
<b>count</b>	10886.0	10886.0	10886.0	10886.0	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
<b>unique</b>	4.0	2.0	2.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN
<b>top</b>	4.0	0.0	1.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN
<b>freq</b>	2734.0	10575.0	7412.0	7192.0	NaN	NaN	NaN	NaN	NaN	NaN
<b>mean</b>	NaN	NaN	NaN	NaN	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177
<b>std</b>	NaN	NaN	NaN	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033
<b>min</b>	NaN	NaN	NaN	NaN	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000
<b>25%</b>	NaN	NaN	NaN	NaN	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000
<b>50%</b>	NaN	NaN	NaN	NaN	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000
<b>75%</b>	NaN	NaN	NaN	NaN	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000
<b>max</b>	NaN	NaN	NaN	NaN	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000

```
# detecting missing values in the dataset
df.isnull().sum()
```

```
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
```

```
# minimum datetime and maximum datetime
print(df['datetime'].min(), df['datetime'].max())
# number of unique values in each categorical columns
df[cat_cols].melt().groupby(['variable', 'value'])['value'].count()
```

```
2011-01-01 00:00:00 2012-12-19 23:00:00
```

	value	
variable	value	
holiday	0	10575
	1	311
season	1	2686
	2	2733
	3	2733
	4	2734
weather	1	7192
	2	2834
	3	859
	4	1
workingday	0	3474
	1	7412

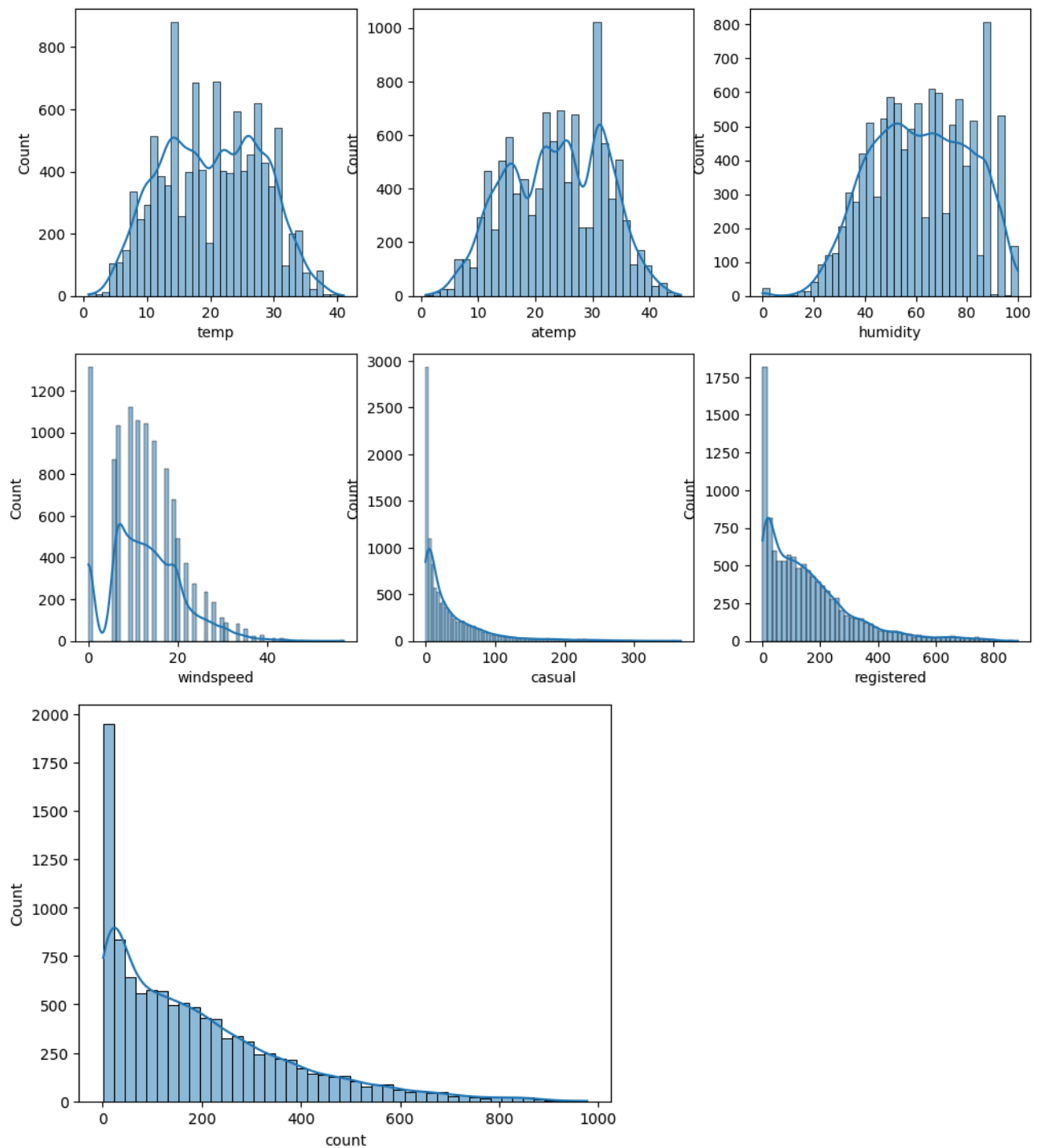
```
# understanding the distribution for numerical variables
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
```

```
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12, 8))
```

```
index = 0
for row in range(2):
    for col in range(3):
        sns.histplot(df[num_cols[index]], ax=axis[row, col], kde=True)
        index += 1
```

```
plt.show()
```

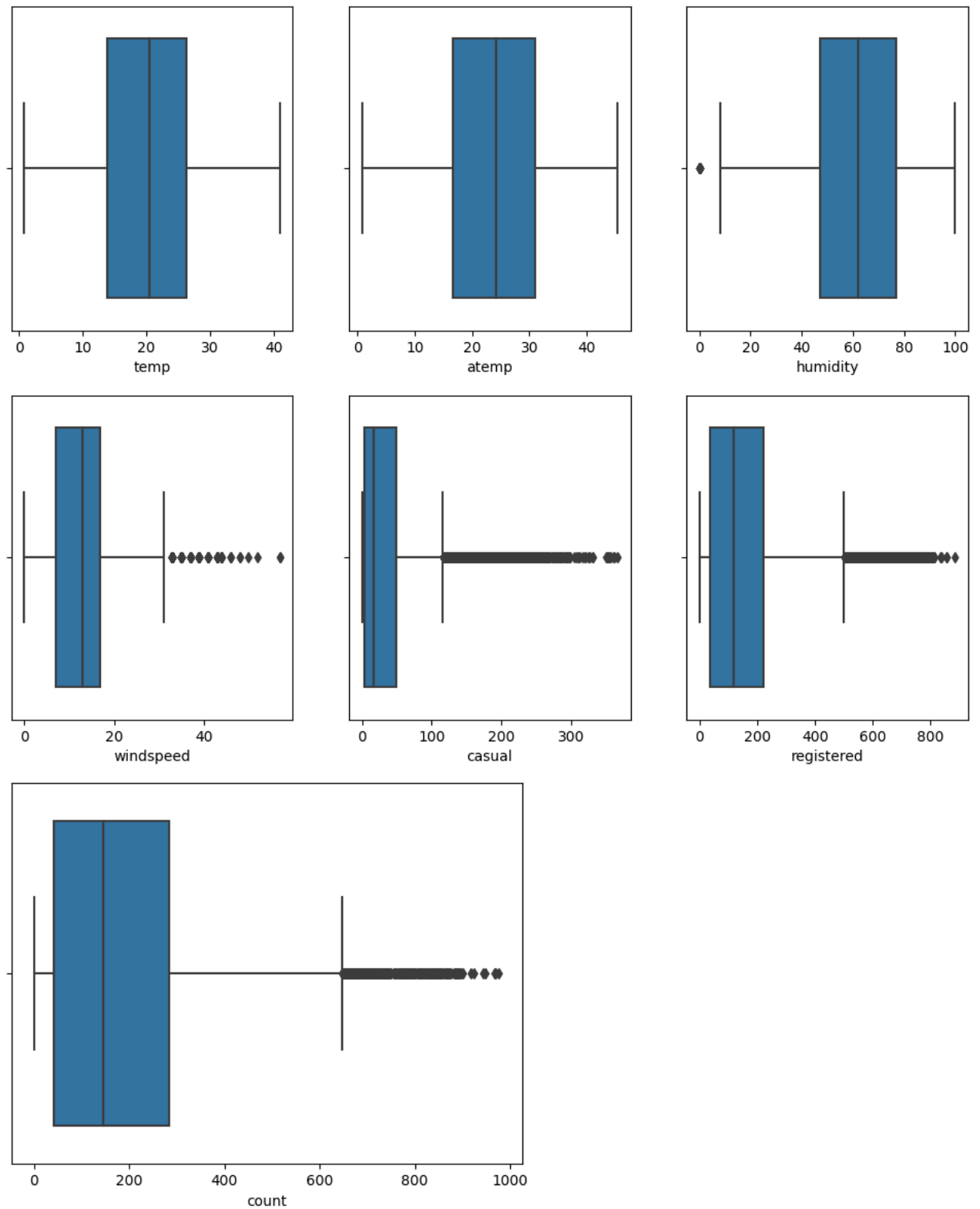
```
sns.histplot(df[num_cols[-1]], kde=True)
plt.show()
```



```
# plotting box plots to detect outliers in the data
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12, 9))

index = 0
for row in range(2):
    for col in range(3):
        sns.boxplot(x=df[num_cols[index]], ax=axis[row, col])
        index += 1

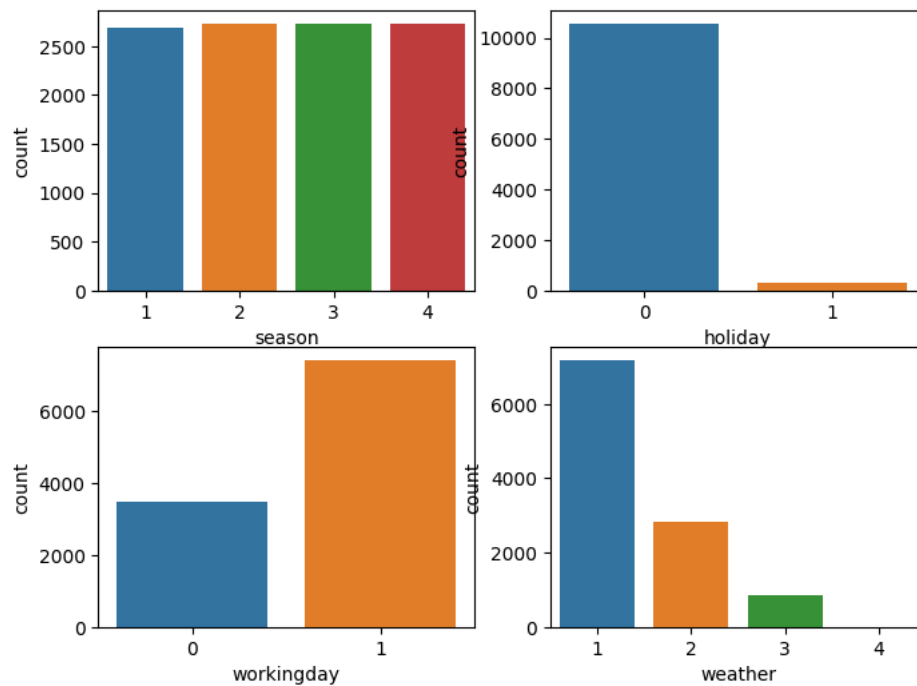
plt.show()
sns.boxplot(x=df[num_cols[-1]])
plt.show()
```



```
# countplot of each categorical column
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(8, 6))

index = 0
for row in range(2):
    for col in range(2):
        sns.countplot(data=df, x=cat_cols[index], ax=axis[row, col])
        index += 1

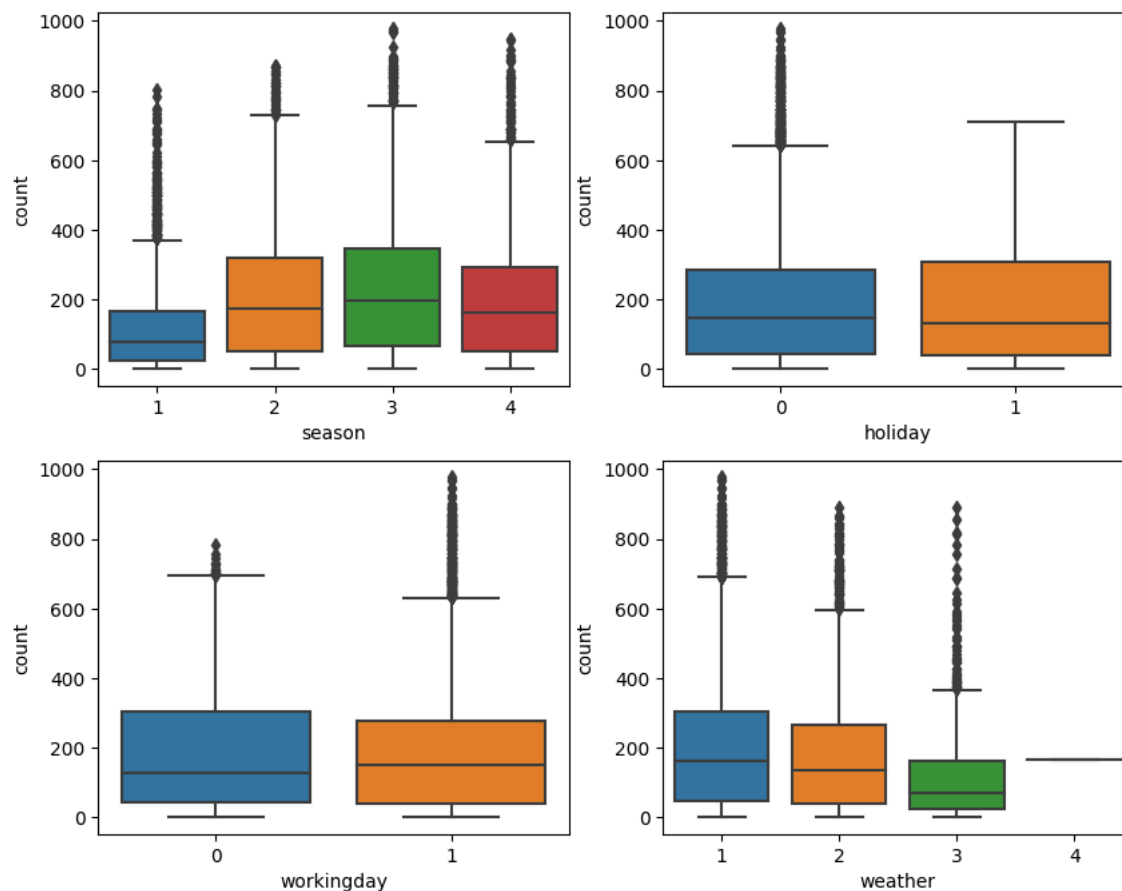
plt.show()
```



```
# plotting categorical variables against count using boxplots
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

index = 0
for row in range(2):
    for col in range(2):
        sns.boxplot(data=df, x=cat_cols[index], y='count', ax=axis[row, col])
        index += 1

plt.show()
```



```
# plotting numerical variables against count using scatterplot
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12, 6))

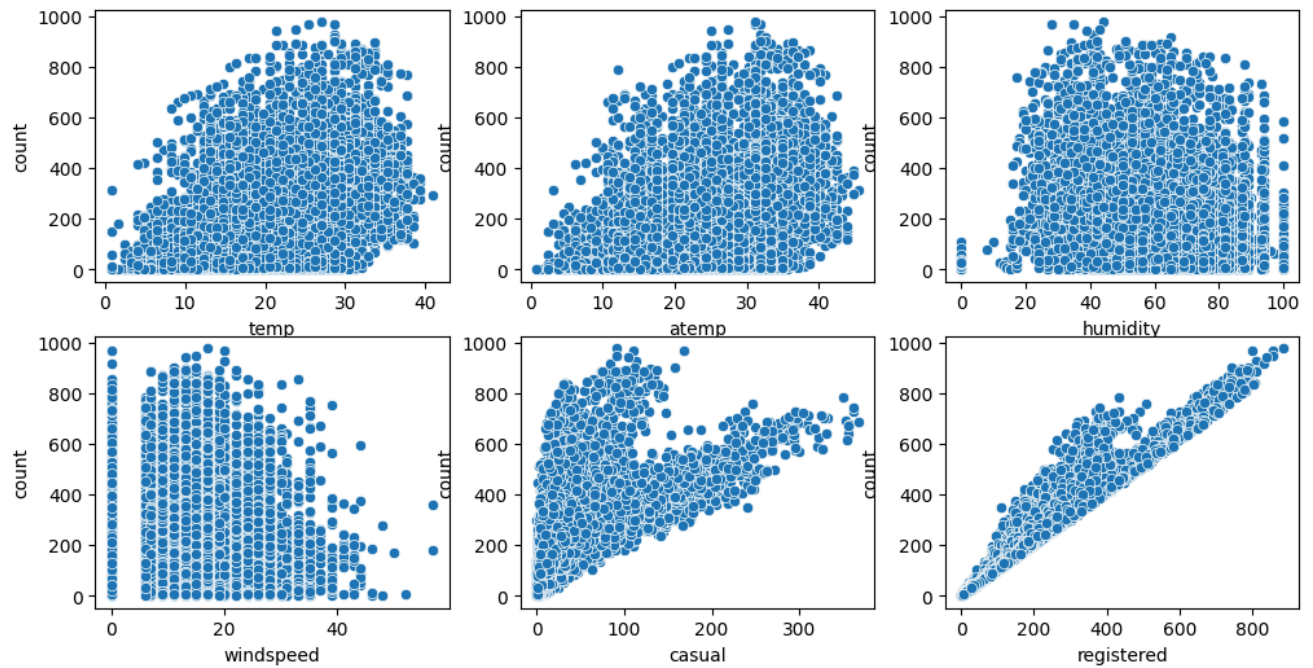
index = 0
```

```

for row in range(2):
    for col in range(3):
        sns.scatterplot(data=df, x=num_cols[index], y='count', ax=axis[row, col])
        index += 1

plt.show()

```



```

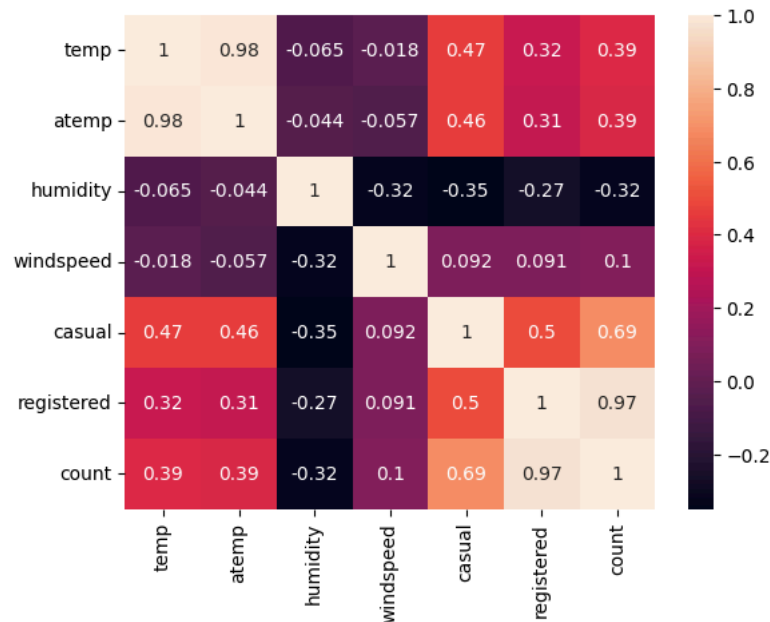
# understanding the correlation between count and numerical variables
df.corr()['count']
sns.heatmap(df.corr(), annot=True)
plt.show()

```

```

<ipython-input-22-b0729b22659f>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future
df.corr()['count']
<ipython-input-22-b0729b22659f>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future
sns.heatmap(df.corr(), annot=True)

```



```

data_table = pd.crosstab(df['season'], df['weather'])
print("Observed values:")
data_table

```

Observed values:

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

```
val = stats.chi2_contingency(data_table)
print(val)
expected_values = val[3]
print(expected_values)
nrows, ncols = 4, 4
dof = (nrows-1)*(ncols-1)
print("degrees of freedom: ", dof)
alpha = 0.05
```

```
chi_sqr = sum([(o-e)**2/e for o, e in zip(data_table.values, expected_values)])
chi_sqr_statistic = chi_sqr[0] + chi_sqr[1]
print("chi-square test statistic: ", chi_sqr_statistic)
```

```
critical_val = stats.chi2.ppf(q=1-alpha, df=dof)
print(f"critical value: {critical_val}")
```

```
p_val = 1-stats.chi2.cdf(x=chi_sqr_statistic, df=dof)
print(f"p-value: {p_val}")
```

```
if p_val <= alpha:
    print("\nSince p-value is less than the alpha 0.05, We reject the Null Hypothesis. Meaning that\
Weather is dependent on the season.")
else:
    print("Since p-value is greater than the alpha 0.05, We do not reject the Null Hypothesis")
```

```
Chi2ContingencyResult(statistic=49.158655596893624, pvalue=1.549925073686492e-07, dof=9, expected_freq=array([[1.77454639e+03, 6
[1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-01],
[1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-01],
[1.80625831e+03, 7.11754180e+02, 2.15736359e+02, 2.51148264e-01]])))
[[1.77454639e+03 6.99258130e+02 2.11948742e+02 2.46738931e-01]
[1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
[1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
[1.80625831e+03 7.11754180e+02 2.15736359e+02 2.51148264e-01]]
degrees of freedom: 9
chi-square test statistic: 44.09441248632364
critical value: 16.918977604620448
p-value: 1.3560001579371317e-06
```

Since p-value is less than the alpha 0.05, We reject the Null Hypothesis. Meaning that Weather is dependent on the season.

```
data_group1 = df[df['workingday']==0]['count'].values
data_group2 = df[df['workingday']==1]['count'].values
```

```
print(np.var(data_group1), np.var(data_group2))
np.var(data_group2)// np.var(data_group1)
```

```
30171.346098942427 34040.69710674686
1.0
```

```
stats.ttest_ind(a=data_group1, b=data_group2, equal_var=True)
```

```
Ttest_indResult(statistic=-1.2096277376026694, pvalue=0.22644804226361348)
```

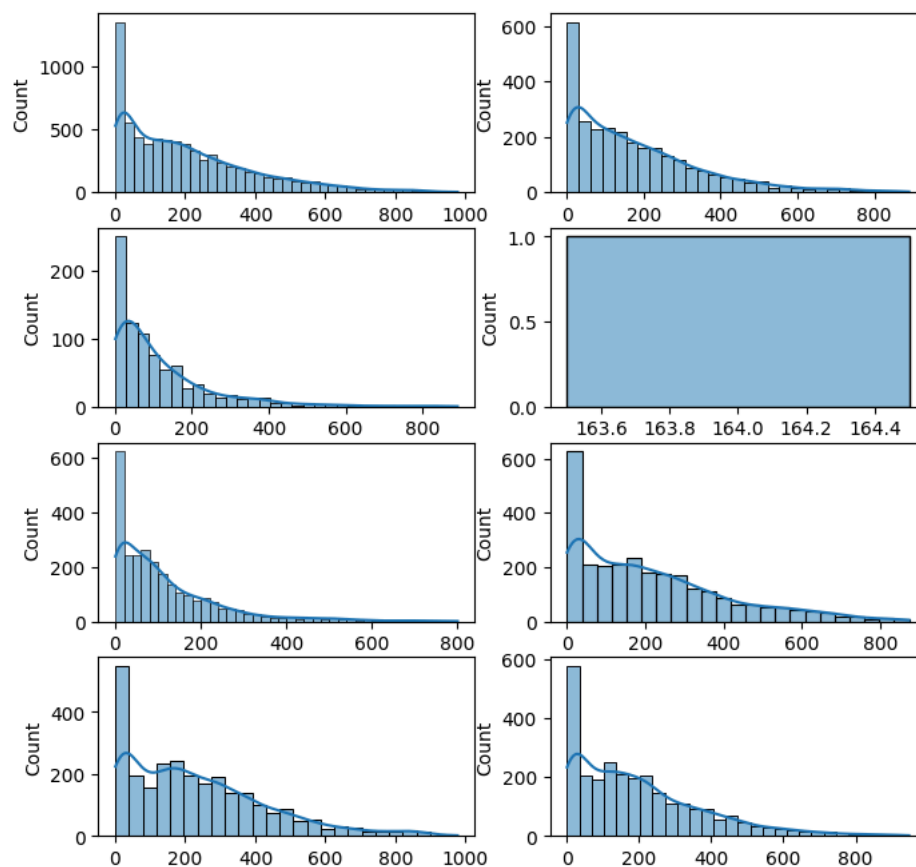
```
# defining the data groups for the ANOVA
from statsmodels.graphics.gofplots import qqplot
gp1 = df[df['weather']==1]['count'].values
gp2 = df[df['weather']==2]['count'].values
gp3 = df[df['weather']==3]['count'].values
gp4 = df[df['weather']==4]['count'].values

gp5 = df[df['season']==1]['count'].values
gp6 = df[df['season']==2]['count'].values
gp7 = df[df['season']==3]['count'].values
gp8 = df[df['season']==4]['count'].values
groups=[gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8]
```

```
fig, axis = plt.subplots(nrows=4, ncols=2, figsize=(8, 8))

index = 0
for row in range(4):
    for col in range(2):
        sns.histplot(groups[index], ax=axis[row, col], kde=True)
        index += 1

plt.show()
```

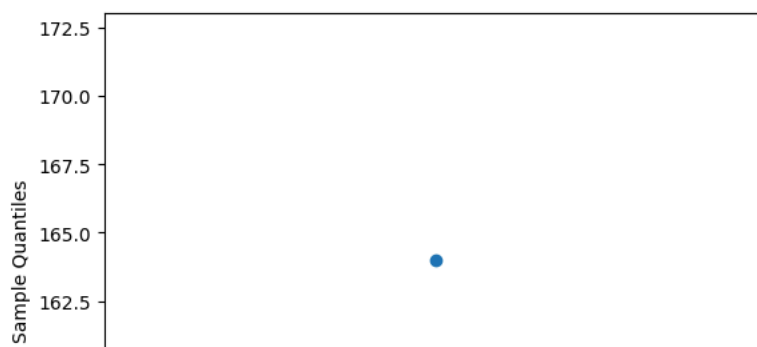
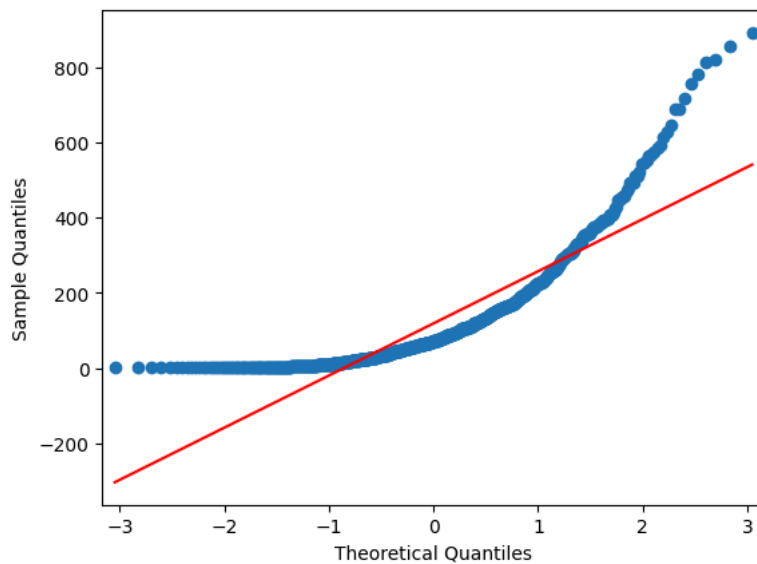
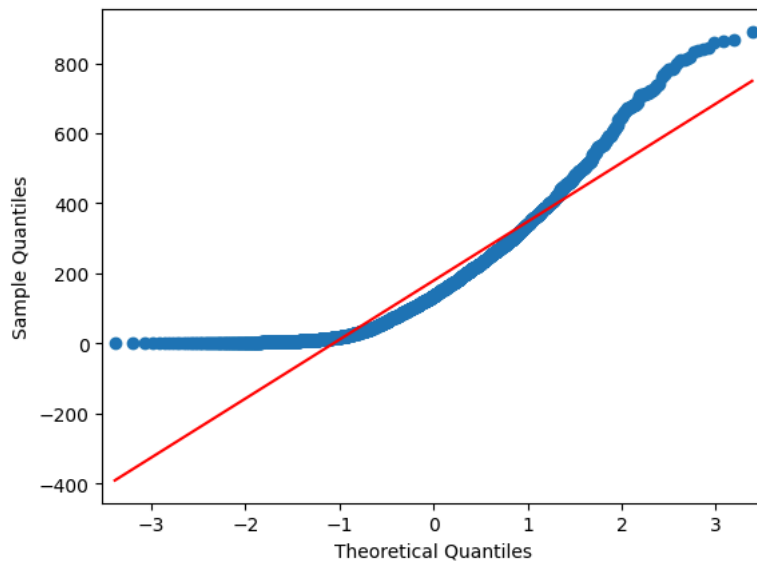
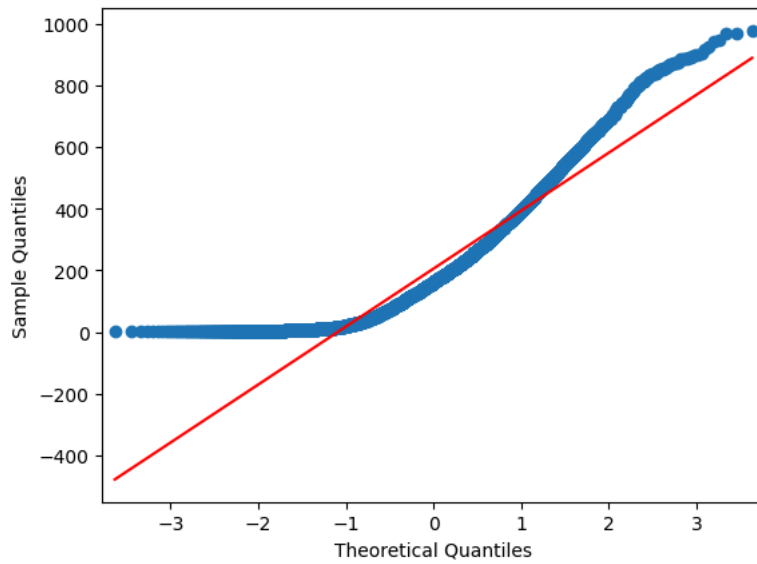


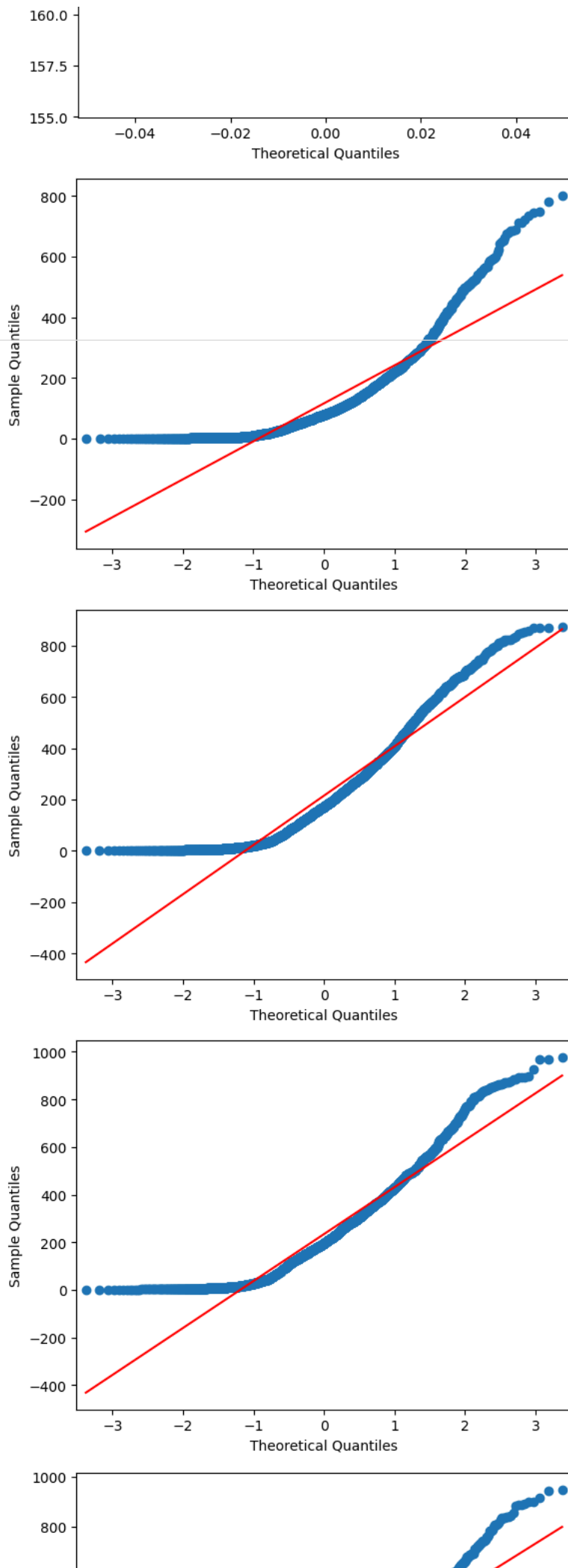
```
index = 0
for row in range(4):
    for col in range(2):
        qqplot(groups[index], line="s")
        index += 1

plt.show()
```









```
#Null Hypothesis: Variances is similar in different weather and season.

#Alternate Hypothesis: Variances is not similar in different weather and season.

#Significance level (alpha): 0.05
levene_stat, p_value = stats.levene(gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8)
print(p_value)
if p_value < 0.05:
    print("Reject the Null hypothesis.Variances are not equal")
else:
    print("Fail to Reject the Null hypothesis.Variances are equal")
```

```
3.46353188897594e-148
Reject the Null hypothesis.Variances are not equal
```

```
#assumptions of ANOVA don't hold, we need Kruskal Wallis
kruskal_stat, p_value = stats.kruskal(gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8)
print("p_value==", p_value)
if p_value<0.05:
    print("Since p-value is less than 0.05, we reject the null hypothesis")
```

```
p_value=== 4.614440933900297e-191
Since p-value is less than 0.05, we reject the null hypothesis
```

## Insights:

- Ride demand strongly depends on weather and season. Usage is highest during pleasant and dry conditions and lowest during rainy or extremely cold seasons.
- Temperature has a strong positive correlation with ride count. As temperature increases (up to a comfortable level), the number of rides also increases.
- Humidity and windspeed negatively affect ride demand. Riders avoid using bikes in humid or windy weather due to discomfort.
- Weekday and weekend ride counts are statistically similar. Indicates that people use Yulu for both commuting and leisure, ensuring consistent demand throughout the week.
- Variability in rides differs across weather and seasons. Ride counts fluctuate significantly between different weather and seasonal conditions.
- Significant differences exist in average rides across seasons and weather types. Demand is not uniform — some months and weather conditions generate much higher rentals.

## Recommendations:

- Plan fleet allocation based on season and weather forecasts. Increase the number of bikes during dry, warm seasons and reduce during monsoons or cold months.
- Introduce dynamic pricing. Offer discounts during low-demand weather conditions and higher prices when demand peaks to balance usage.
- Schedule maintenance during low-demand periods. Perform bike servicing and battery replacements in rainy or cold seasons when rides are fewer.
- Maintain consistent weekday and weekend operations. Since usage is steady, ensure bike availability across all days to maximize revenue.
- Use predictive analytics for demand forecasting. Build models using weather, temperature, and season data to forecast future ride counts and optimize resource deployment.
- Enhance customer engagement. Notify users about favorable weather conditions or offer incentives for off-peak usage to increase ride frequency.
- Adopt sustainability-focused campaigns. Promote Yulu as an eco-friendly commuting option during high-demand seasons to boost ridership and brand image.

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.