




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df=pd.read_csv('/content/bike_sharing.txt')
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


Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)


```
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.columns
```

 `Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',  
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],  
 dtype='object')`

```
len(df.columns)
print(f'Total no of columns in yulu dataset are {len(df.columns)}')
```

 Total no of columns in yulu dataset are 12

Non graphical analysis : Value counts and number of unique attributes in a column

```
df['season'].nunique()
```

 4

```
df['season'].value_counts()
```



```
count
season
4    2734
2    2733
3    2733
1    2686
```

**dtype:** int64

```
df['holiday'].nunique()
```



```
2
```

```
df['holiday'].value_counts()
```



```
count
holiday
0    10575
1      311
```

**dtype:** int64

```
df['workingday'].nunique()
```



```
2
```

```
df['workingday'].value_counts()
```



```
count
workingday
1     7412
0     3474
```

**dtype:** int64

```
df['weather'].nunique()
```



```
4
```

```
df['weather'].value_counts()
```



```
count
weather
1     7192
2     2834
3      859
4         1
```

**dtype:** int64

```
df['temp'].nunique()
```



```
49
```

```
df['temp'].value_counts()
```



	count
temp	
14.76	467
26.24	453
28.70	427
13.94	413
18.86	406
22.14	403
25.42	403
16.40	400
22.96	395
27.06	394
24.60	390
12.30	385
21.32	362
17.22	356
13.12	356
29.52	353
10.66	332
18.04	328
20.50	327
30.34	299
9.84	294
15.58	255
9.02	248
31.16	242
8.20	229
27.88	224
23.78	203
32.80	202
11.48	181
19.68	170
6.56	146
33.62	130
5.74	107
7.38	106
31.98	98
34.44	80
35.26	76
4.92	60
36.90	46
4.10	44
37.72	34
36.08	23
3.28	11
0.82	7
38.54	7

```
39.36    6
2.46     5
1.64     2
41.00     1
```

```
dtype: int64
```

```
df['atemp'].nunique()
```

```
↔ 60
```

```
df['atemp'].value_counts()
```



	count
atemp	
31.060	671
25.760	423
22.725	406
20.455	400
26.515	395
16.665	381
25.000	365
33.335	364
21.210	356
30.305	350
15.150	338
21.970	328
24.240	327
17.425	314
31.820	299
34.850	283
27.275	282
32.575	272
11.365	271
14.395	269
29.545	257
19.695	255
15.910	254
12.880	247
13.635	237
34.090	224
12.120	195
28.790	175
23.485	170
10.605	166
35.605	159
9.850	127
18.180	123
36.365	123
37.120	118
9.090	107
37.880	97
28.030	80
7.575	75
38.635	74
6.060	73
39.395	67
6.820	63
8.335	63
18.940	45

```

40.150    45
40.910    39
5.305     25
42.425    24
41.665    23
3.790     16
4.545     11
3.030      7
43.940     7
2.275      7
43.180     7
44.695     3
0.760      2
1.515      1
45.455     1

```

**dtype:** int64

```
df['humidity'].nunique()
```

89

```
df['humidity'].value_counts()
```

```

count
humidity
88    368
94    324
83    316
87    289
70    259
...    ...
8       1
10      1
97       1
96       1
91       1

```

89 rows × 1 columns

**dtype:** int64

```
df['windspeed'].nunique()
```

28

```
df['windspeed'].value_counts()
```



	count
windspeed	
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

dtype: int64

```
df['casual'].nunique()
```



309

```
df['casual'].value_counts()
```



count

casual

0	986
1	667
2	487
3	438
4	354
...	...
332	1
361	1
356	1
331	1
304	1

309 rows × 1 columns

dtype: int64

```
df['registered'].nunique()
```



731

```
df['registered'].value_counts()
```



count

registered

3	195
4	190
5	177
6	155
2	150
...	...
570	1
422	1
678	1
565	1
636	1

731 rows × 1 columns

dtype: int64

```
df['count'].nunique()
```



822

```
df['count'].value_counts()
```





count	
count	
5	169
4	149
3	144
6	135
2	132
...	...
801	1
629	1
825	1
589	1
636	1

822 rows × 1 columns

dtype: int64

df.describe().T



	count	mean	std	min	25%	50%	75%	max	
season	10886.0	2.506614	1.116174	1.00	2.0000	3.000	4.0000	4.0000	
holiday	10886.0	0.028569	0.166599	0.00	0.0000	0.000	0.0000	1.0000	
workingday	10886.0	0.680875	0.466159	0.00	0.0000	1.000	1.0000	1.0000	
weather	10886.0	1.418427	0.633839	1.00	1.0000	1.000	2.0000	4.0000	
temp	10886.0	20.230860	7.791590	0.82	13.9400	20.500	26.2400	41.0000	
atemp	10886.0	23.655084	8.474601	0.76	16.6650	24.240	31.0600	45.4550	
humidity	10886.0	61.886460	19.245033	0.00	47.0000	62.000	77.0000	100.0000	
windspeed	10886.0	12.799395	8.164537	0.00	7.0015	12.998	16.9979	56.9969	
casual	10886.0	36.021955	49.960477	0.00	4.0000	17.000	49.0000	367.0000	
registered	10886.0	155.552177	151.039033	0.00	36.0000	118.000	222.0000	886.0000	
count	10886.0	191.574132	181.144454	1.00	42.0000	145.000	284.0000	977.0000	

Observations on Shape of Data: The dataset contains 10,886 entries and 12 columns.

Data Types of All the Attributes:

The dataset primarily consists of numeric (int64 and float64) and one object type (datetime) columns. The 'datetime' column is in object format and should ideally be converted to a datetime data type for better analysis.

Conversion of Categorical Attributes to 'Category':

- We haven't yet converted categorical variables like 'season', 'holiday', 'workingday', and 'weather' to 'category' data type. This conversion can optimize memory usage and is beneficial for certain types of analysis.

Missing Value Detection:

- The initial overview indicated that there are no missing values in any of the columns.

Columns Description:

- datetime: The date and time of bike rentals (currently in object format, may need conversion to datetime).
- season: Categorical (1: spring, 2: summer, 3: fall, 4: winter).
- holiday: Binary (0 or 1), indicating if the day is a holiday.
- workingday: Binary (0 or 1), indicating if the day is a working day.
- weather: Categorical, describing the weather conditions.

- temp: Numeric, temperature in Celsius.
- atemp: Numeric, 'feels like' temperature in Celsius.
- humidity: Numeric, percentage humidity.
- windspeed: Numeric, wind speed.
- casual: Numeric, count of casual users.
- registered: Numeric, count of registered users.
- count: Numeric, total count of rented bikes (casual + registered).

```
# Convert 'datetime' to datetime data type
df['datetime'] = pd.to_datetime(df['datetime'])

# Convert 'season', 'holiday', 'workingday', and 'weather' to 'category' data type
categorical_columns = ['season', 'holiday', 'workingday', 'weather']
df[categorical_columns] = df[categorical_columns].astype('category')

# Check for missing values
missing_values = df.isnull().sum()
```

```
df.dtypes,missing_values
```

```
(datetime      datetime64[ns]
 season         category
 holiday        category
 workingday      category
 weather         category
 temp           float64
 atemp          float64
 humidity        int64
 windspeed      float64
 casual         int64
 registered      int64
 count          int64
dtype: object,
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)
```

#### Data Types Conversion:

- The 'datetime' column has been converted to the datetime64 data type. The 'season', 'holiday', 'workingday', and 'weather' columns have been converted to category data types. Missing Value Detection:
- There are no missing values in the dataset. Each column has been checked, and none have missing or null values.

```
# renaming the count column to total users
df=df.rename(columns={'count':'Total_users'})

num_cols=['temp','atemp','humidity','windspeed','casual','registered','Total_users']

for i in range(len(num_cols)):
    ol_data=df[num_cols[i]].tolist()
    Q1=np.quantile(ol_data,0.25)
    Q2=np.median(ol_data)
    Q3=np.quantile(ol_data,0.75)
    min=np.min(ol_data)
    max=np.max(ol_data)
    IQR=Q3-Q1
    lower_bound=Q1-(1.5*IQR)
    upper_bound=Q3+(1.5*IQR)
```

```

lower_outliers = [k for k in ol_data if k < lower_bound]
upper_outliers = [k for k in ol_data if k > upper_bound]

# calculating outlier percentage
upper_outlier_percentage=(len(upper_outliers)/len(df)*100)
lower_outlier_percentage=(len(lower_outliers)/len(df)*100)

# printing outlier statistics
print(f'outlier detection of column {num_cols[i]}')
print('-'*45)
print(f'Q1: {Q1}')
print(f'Q2: {Q2}')
print(f'Q3: {Q3}')
print(f'IQR: {IQR}')
print(f'min: {min}')
print(f'max: {max}')
print(f'lower bound: {lower_bound}')
print(f'upper bound: {upper_bound}')
print(f'lower outlier percentage: {lower_outlier_percentage}')
print(f'upper outlier percentage: {upper_outlier_percentage}')
print(f'Total outlier percentage : {lower_outlier_percentage+upper_outlier_percentage}')
print('-'*45)

```

```

outlier detection of column Total_users
-----
Q1: 42.0
Q2: 145.0
Q3: 284.0
IQR: 242.0
min: 1
max: 977
lower bound: -321.0
upper bound: 647.0
lower outlier percentage: 2.75583318023149
upper outlier percentage: 2.75583318023149
Total outlier percentage : 5.51166636046298
-----

```

```

df['season']=df['season'].map(str)
season_map={'1':'spring','2':'summer','3':'fall','4':'winter'}
df['season']=df['season'].map(lambda x: season_map[x])

df['holiday']=df['holiday'].map(str)
holiday_map={'0':'no','1':'yes'}
df['holiday']=df['holiday'].map(lambda x: holiday_map[x])

df['workingday']=df['workingday'].map(str)
workingday_map={'0':'no','1':'yes'}
df['workingday']=df['workingday'].map(lambda x: workingday_map[x])

df['weather']=df['weather'].map(str)
weather_map={'1':'clear','2':'cloudy','3':'light_rain','4':'heavy_rain'}
df['weather']=df['weather'].map(lambda x: weather_map[x])

```

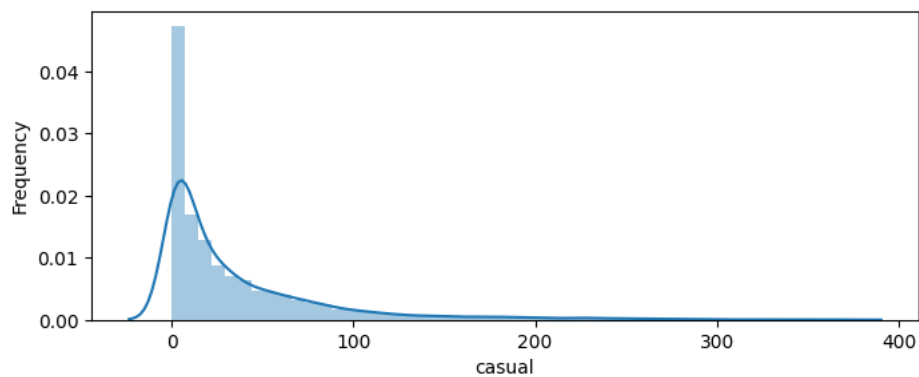
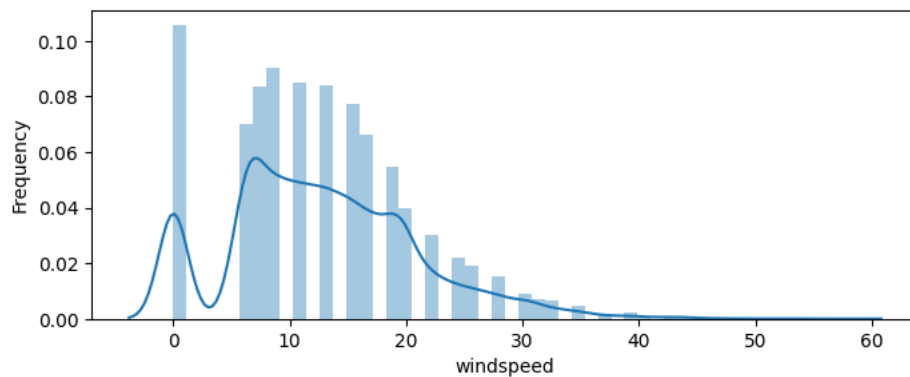
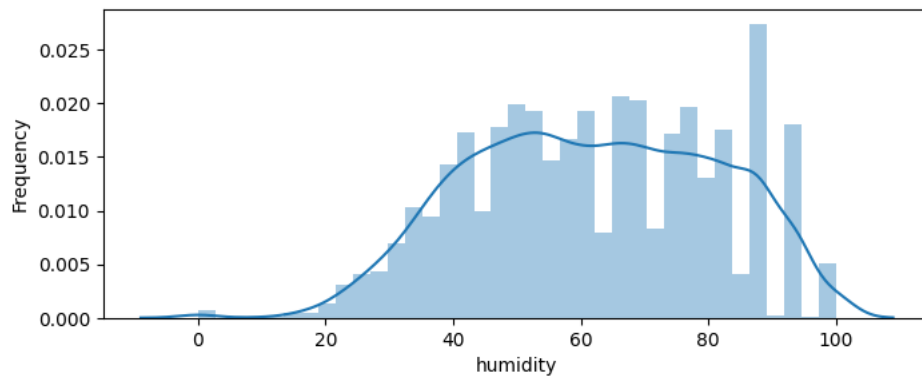
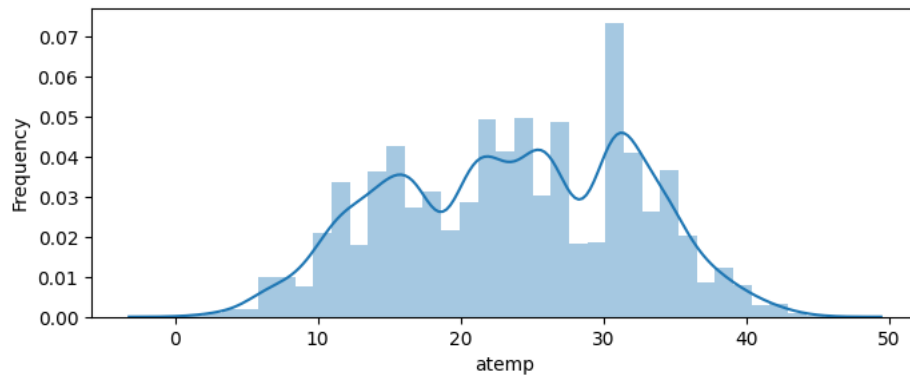
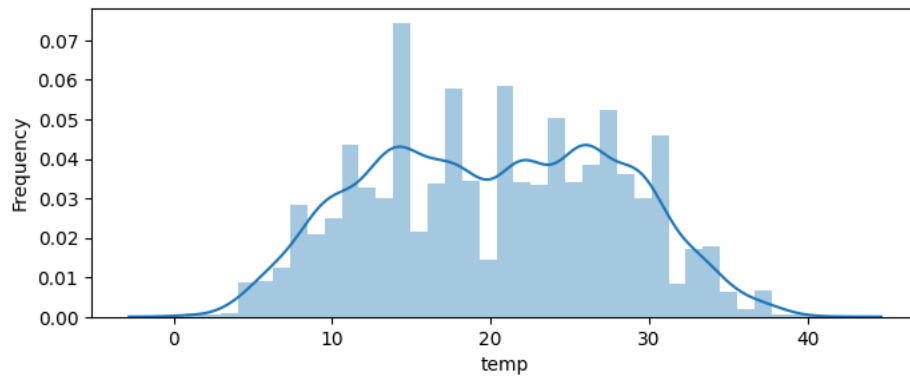
## Univariate analysis

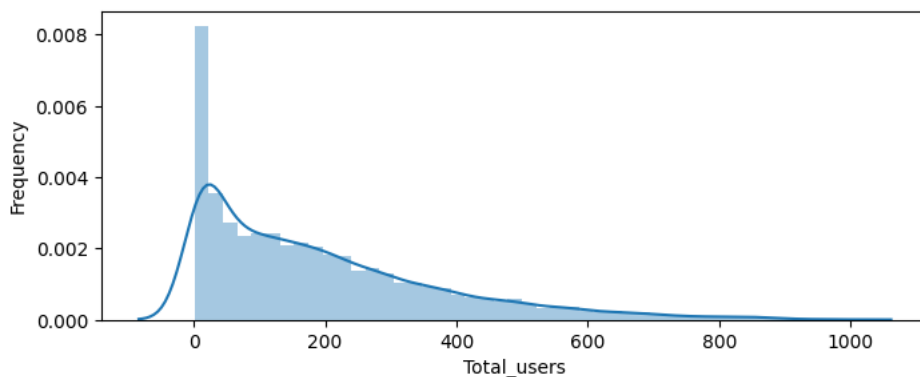
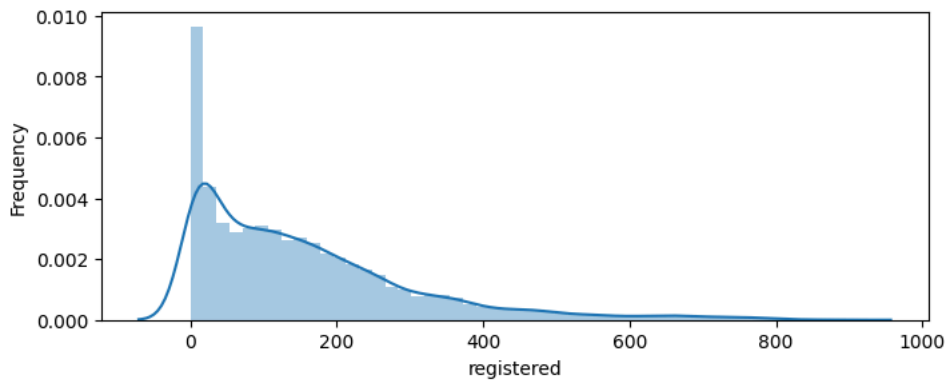
```

# plotting distribution plots for continuous variables
num_cols=['temp','atemp','humidity','windspeed','casual','registered','Total_users']

for i in num_cols:
    plt.figure(figsize=(8,3))
    sns.distplot(df[i],kde=True)
    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.show()

```





#### Continuous Variables:

1. Temperature (temp) and 'Feels Like' Temperature (atemp): Both show similar distributions, indicating a reasonable spread across different temperatures.
2. Humidity (humidity): The distribution is somewhat uniform but with a slight increase in frequency at higher humidity levels.
3. Wind Speed (windspeed): Most of the data points are clustered at lower wind speeds, indicating that high wind speeds are less common.
4. Casual (casual): The distribution is right-skewed, suggesting that there are many instances with few casual users and fewer instances with a large number of casual users.
5. Registered (registered): Similar to 'casual', but with a slightly less pronounced right skew. This indicates a higher overall usage by registered users.
6. Total Count (count): Reflects the combined effect of 'casual' and 'registered', showing a right-skewed distribution indicating more instances of lower total usage.

```
# plots for categorical variables
cat_cols=['season','holiday','workingday','weather']
```

```
for i in cat_cols:
    plt.figure(figsize=(6,3))
    sns.countplot(x=df[i])
    plt.xlabel(i)
    plt.ylabel('count')
    plt.show()
```

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

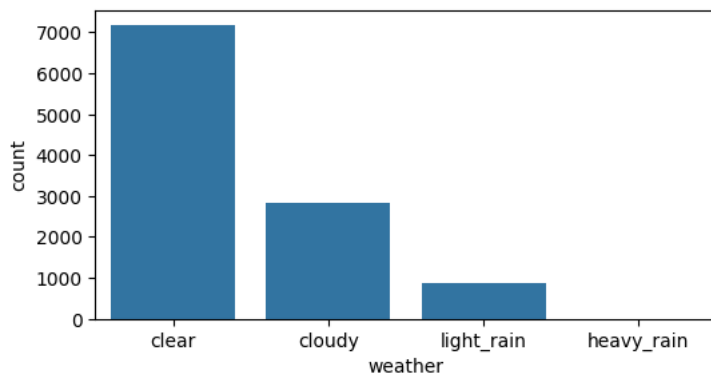
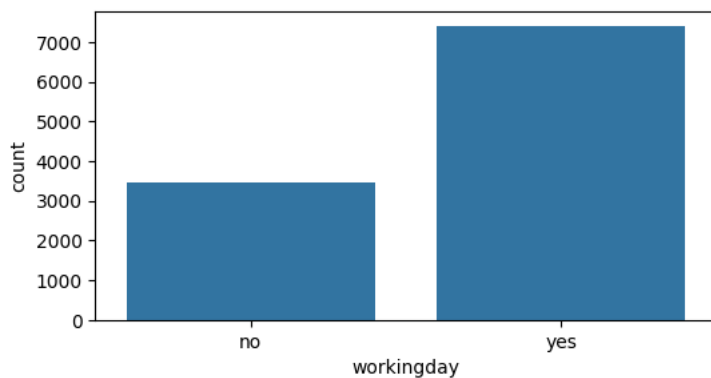
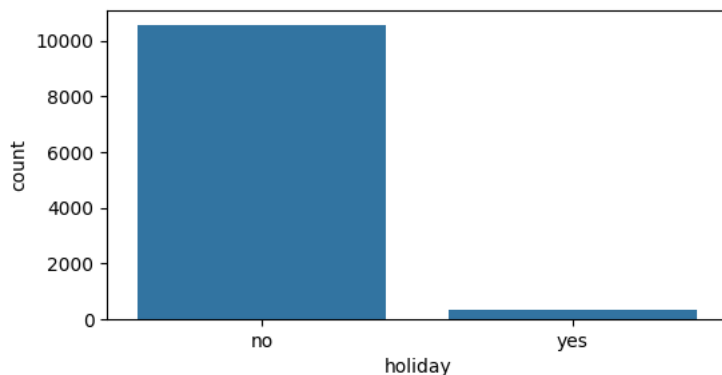
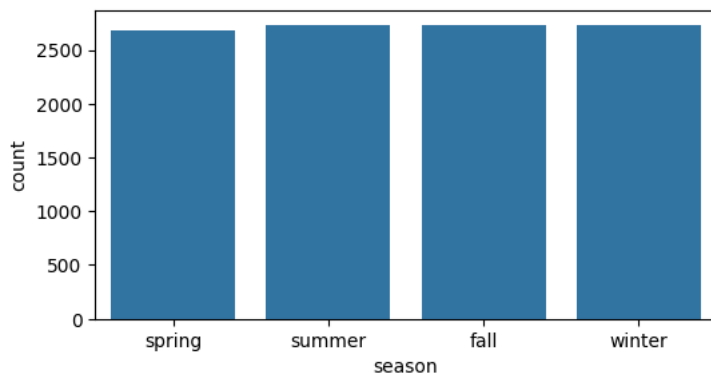


Figure size 640x480 with 4 Axes

#### Categorical Variables:

1. Season (season): The dataset seems to have a relatively balanced distribution across different seasons.
2. Holiday (holiday): There are significantly fewer days classified as holidays compared to non-holidays.
3. Working Day (workingday): More days are classified as working days than non-working days.
4. Weather (weather): Most days fall into the first two weather categories (clear/few clouds and mist/cloudy), with very few instances of the more severe weather conditions (categories 3 and 4).

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

## ✓ Bivariate Analysis: Exploring relationships between important categorical variables and 'count'

```
# Relationship between 'workingday' and 'Total_users'

plt.figure(figsize=(10, 6))
sns.boxplot(x='workingday', y='Total_users', data=df)
plt.title('Count of Bikes vs Working Day', fontsize=14)
plt.xlabel('Working Day (0: No, 1: Yes)')
plt.ylabel('Bike Count')
plt.show()

# Relationship between 'season' and 'count'

plt.figure(figsize=(10, 6))
sns.boxplot(x='season', y='Total_users', data=df)
plt.title('Count of Bikes vs Season', fontsize=14)
plt.xlabel('Season')
plt.ylabel('Bike Count')

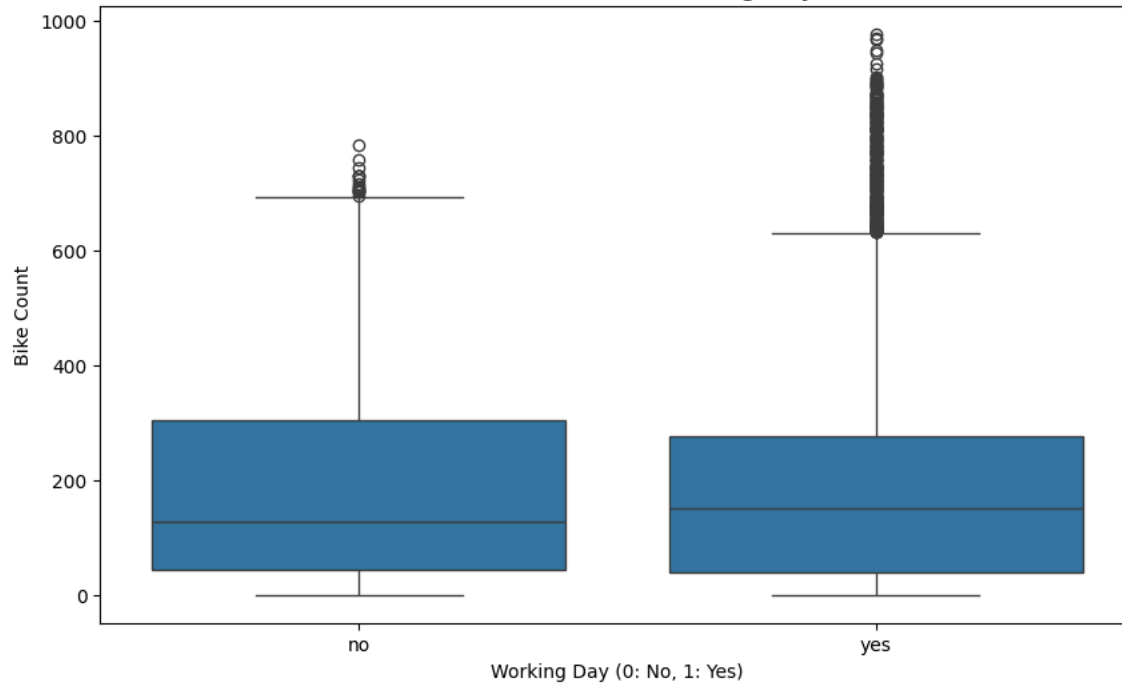
plt.show()

# Relationship between 'weather' and 'count'

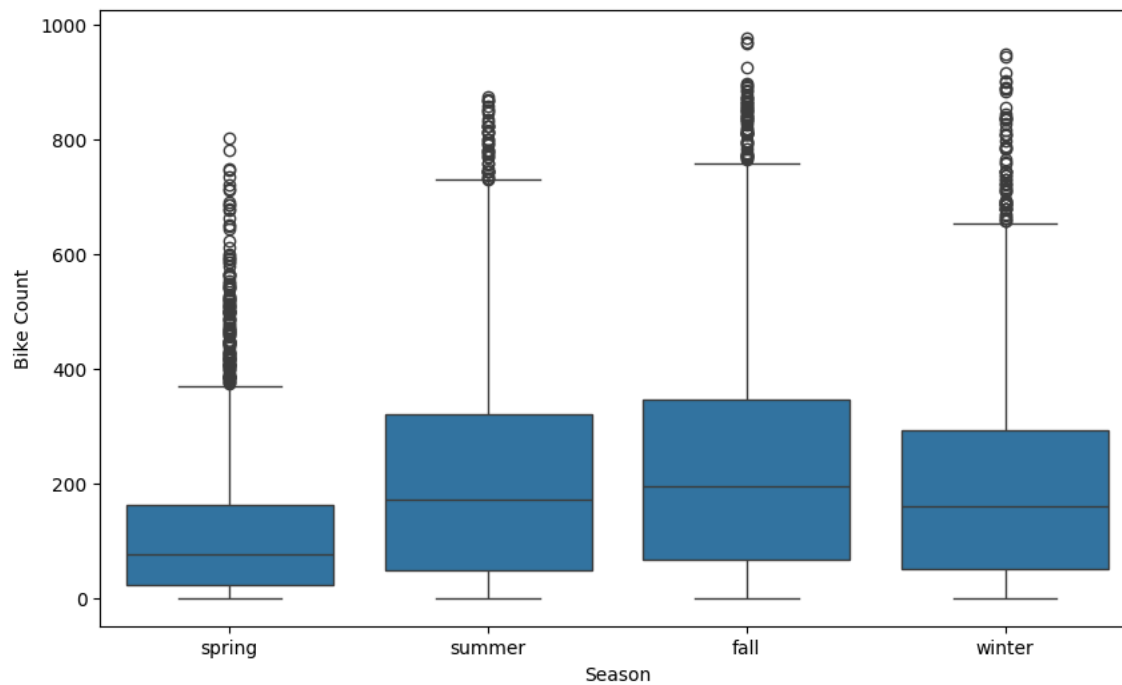
plt.figure(figsize=(10, 6))
sns.boxplot(x='weather', y='Total_users', data=df)
plt.title('Count of Bikes vs Weather', fontsize=14)
plt.xlabel('Weather')
plt.ylabel('Bike Count')
plt.show()
```



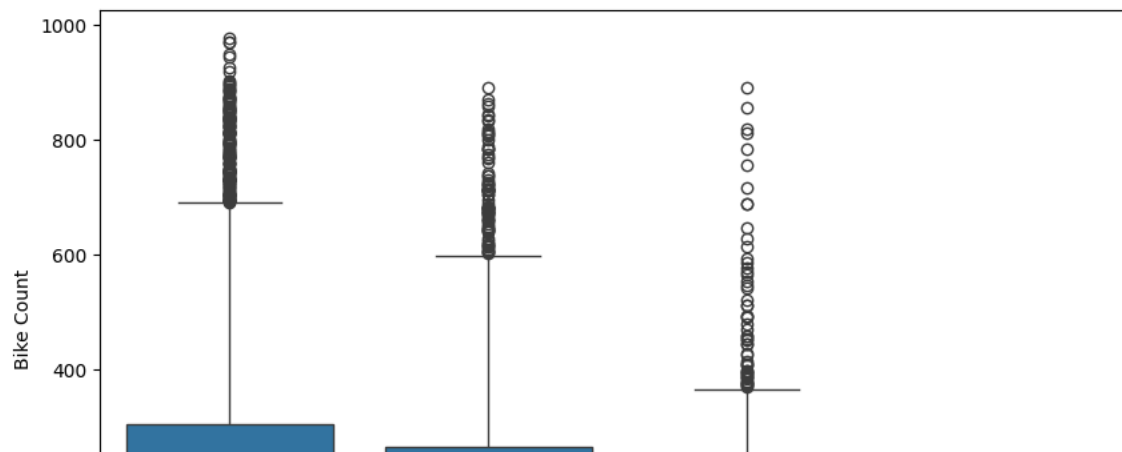
Count of Bikes vs Working Day



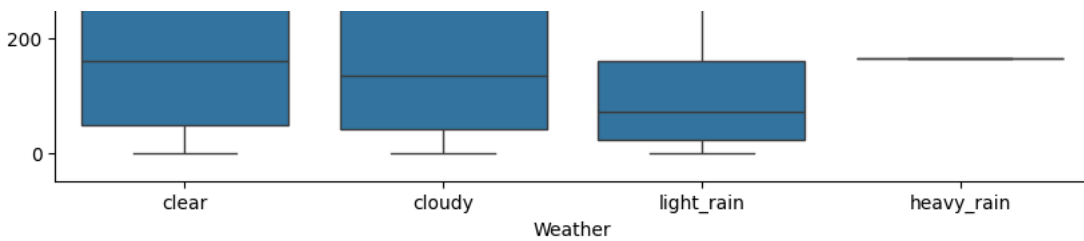
Count of Bikes vs Season



Count of Bikes vs Weather







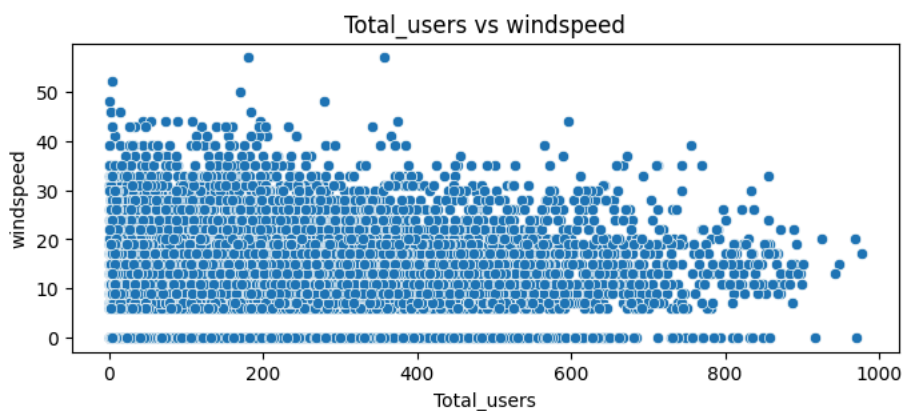
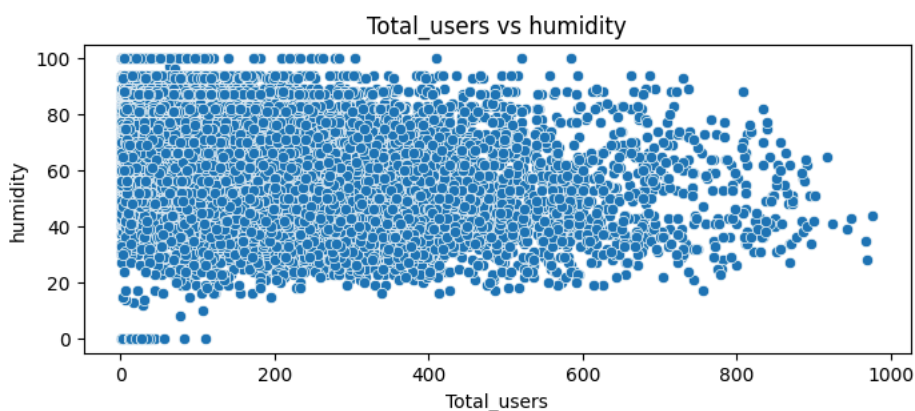
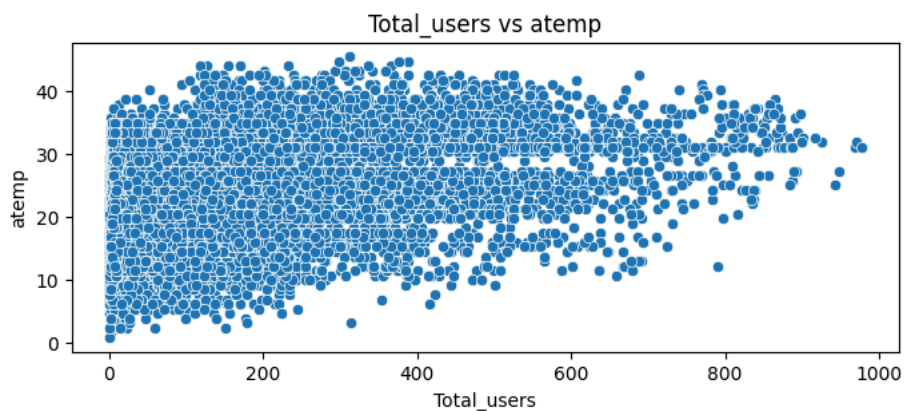
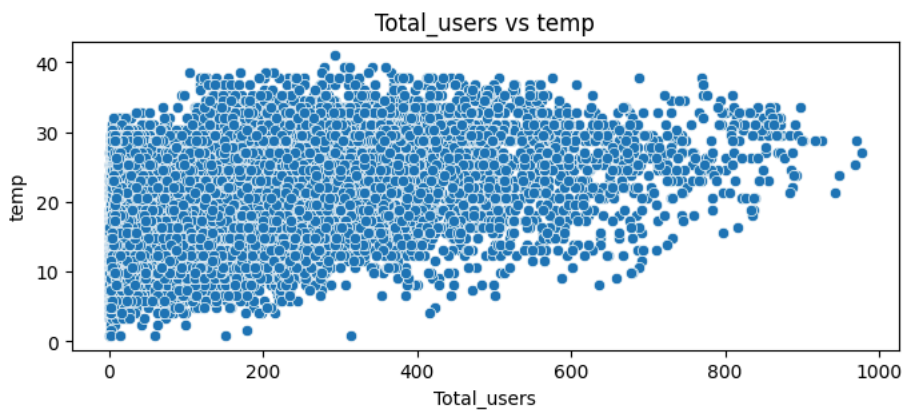
#### Summary:

The EDA revealed valuable insights into the factors influencing bike rental demand. Seasonal changes, weather conditions, and whether a day is a working day significantly impact bike usage. The data also indicated potential areas for further investigation, such as the reasons behind the outliers in bike rental counts.

#### Relation between 2 continuous variables

```
num_cols=['temp','atemp','humidity','windspeed']

for i in num_cols:
    plt.figure(figsize=(8,3))
    sns.scatterplot(x=df['Total_users'],y=df[i])
    plt.title(f'Total_users vs {i}')
    plt.xlabel('Total_users')
    plt.ylabel(i)
    plt.show()
```



```
# correlation chart
corr=df.corr(numeric_only=True)
display(corr)

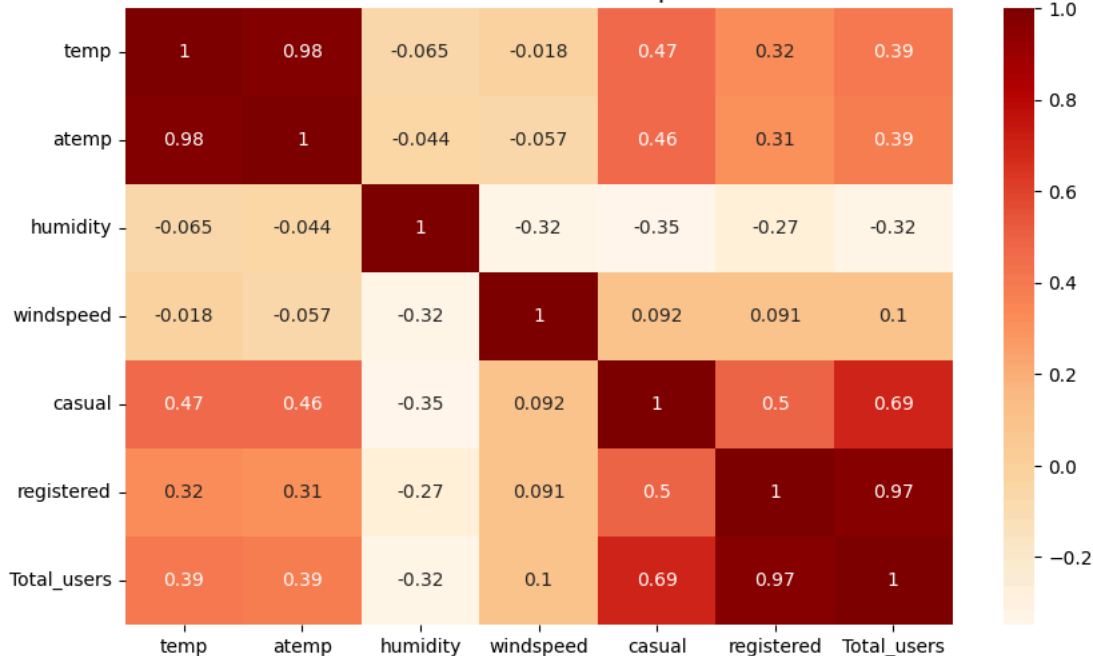
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='OrRd')
plt.title('Correlation Heatmap', fontsize=14)
plt.show()
```



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



Correlation Heatmap



Next steps:

[Generate code with corr](#)[View recommended plots](#)[New interactive sheet](#)

Temperature (temp and atemp): Rentals increase with temperature up to a certain point but drop at extremely high temperatures. Indicates a preference for cycling in mild to warm weather.

Humidity: Rentals are lower at very high humidity levels, suggesting discomfort during muggy conditions.

Wind Speed: No strong correlation with rentals, but very high wind speeds may slightly reduce demand.

### \*Hypothesis Testing \*

```
df.head(2)
```



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



Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

To conduct a 2-sample t-test to evaluate if there's a significant difference in the number of electric cycles rented on working days versus non-working days, we'll follow these steps:

### 1. Understand the Variables:

- **workingday**: Categorical variable indicating if the day is a working day (yes) or not (no).
- **total\_riders**: Numerical variable representing the number of electric cycles rented.

\*2. *Hypotheses*: \*

- **Null Hypothesis ( $H_0$ )**: The mean number of electric cycles rented is the same for working days and non-working days.
- **Alternative Hypothesis ( $H_a$ )**: The mean number of electric cycles rented differs between workingdays and non-working days.

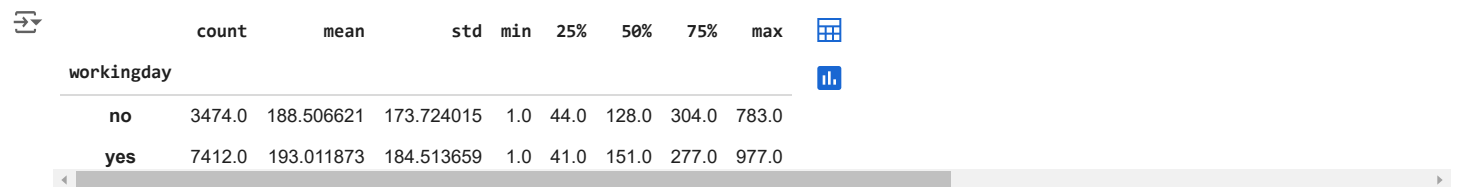
**3. Partition Data**: Divide total\_riders into two groups based on the workingday column: one group for working days and one for non-working days.

#### 4. Check Assumptions:

- **Normality**: Use a normality test (e.g., Shapiro-Wilk test) or visual analysis using Q-Q plot.
  - **Equal Variance**: Use Levene's test.

**5. Conduct the T-Test**: Use a 2-sample independent T-test assuming equal or unequal variances based on the variance test result. Interpret Results:

```
df.groupby('workingday')['Total_users'].describe()
```



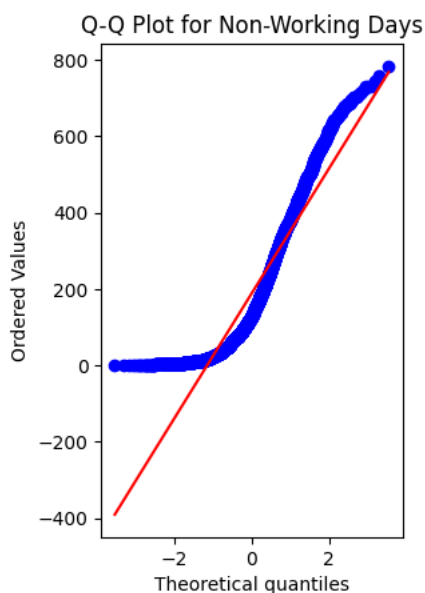
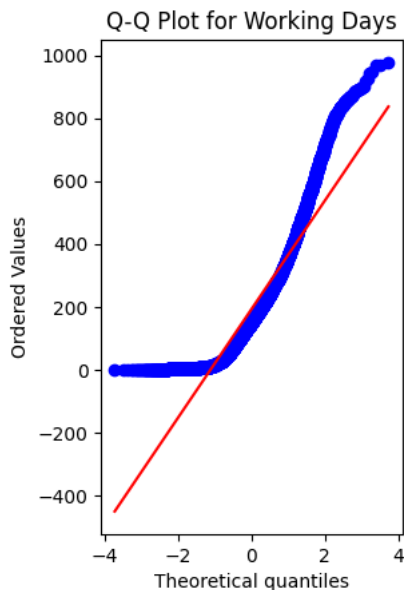
	count	mean	std	min	25%	50%	75%	max	
workingday									
no	3474.0	188.506621	173.724015	1.0	44.0	128.0	304.0	783.0	
yes	7412.0	193.011873	184.513659	1.0	41.0	151.0	277.0	977.0	

```
# Partition data into working day and non-working day groups
workingday_data=df[df['workingday']=='yes']['Total_users']
non_workingday_data=df[df['workingday']=='no']['Total_users']
```

```
# importing qq plot library
```

```
import scipy.stats as stats
plt.subplot(1,2,1)
stats.probplot(workingday_data, dist='norm', plot=plt)
plt.title('Q-Q Plot for Working Days')
plt.show()

plt.subplot(1,2,2)
stats.probplot(non_workingday_data, dist='norm', plot=plt)
plt.title('Q-Q Plot for Non-Working Days')
plt.show()
```



```
# importing library to perform shapiro wilk test to check data normality
```

```
from scipy.stats import shapiro
```

```
shapiro_working=shapiro(workingday_data)
shapiro_non_working=shapiro(non_workingday_data)
```

```
print('shapiro_working',shapiro_working)
print('shapiro_non_working',shapiro_non_working)
```

```
shapiro_working ShapiroResult(statistic=0.8702545795617624, pvalue=2.2521124830019574e-61)
shapiro_non_working ShapiroResult(statistic=0.885211755076074, pvalue=4.4728547627911074e-45)
```

```
if shapiro_working[1]>0.05 and shapiro_non_working[1]>0.05:
    print('data is normal')
else:
    print('data is not normal')
```

```
data is not normal
```

Double-click (or enter) to edit

Insights:

1. Hence  $p\_value$  is  $< 0.05$  significance value we can say that data is normally distributed.
2. QQ plot suggests that actual distribution deviates from perfect normal distribution line.

```
# check for equal variance
from scipy.stats import levene
stat, p_value = levene(workingday_data, non_workingday_data)
print('stat', stat)
print('p_value', p_value)
```

```
stat 0.004972848886504472
p_value 0.9437823280916695
```

```
if p_value > 0.05:
    print('data has equal variance')
else:
    print('data does not have equal variance')
```

```
data has equal variance
```

To perform the t test the data should be normally distributed but in qq plot the actual distribution is deviating from the perfect normal distribution. we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples. But we will try to compare both tests

```
# importing library to perform Mann-Whitney test
```

```
from scipy.stats import mannwhitneyu
```

```
stat, p_value = mannwhitneyu(workingday_data, non_workingday_data)
print('stat', stat)
print('p_value', p_value)
```

```
if p_value > 0.05:
    print('Fail to reject the null hypothesis:')
    print('There is no significant difference in the number of electric cycles rented on working days and non-working days.')
else:
    print('Reject the null hypothesis:')
    print('There is a significant difference in the number of electric cycles rented on working days and non-working days.')
```

```
stat 12868495.5
p_value 0.9679139953914079
Fail to reject the null hypothesis:
There is no significant difference in the number of electric cycles rented on working days and non-working days.
```

## Perform 2-sample T-test

```
from scipy.stats import ttest_ind
```

```
t_stat, p_value = ttest_ind(workingday_data, non_workingday_data)
print('t_stat', t_stat)
print('p_value', p_value)
```

```
if p_value < 0.05:
    print('Reject the null hypothesis : There is a significant difference between cycles rented on working_day and non_working_day')
else:
    print('Fail to reject the null hypothesis : There is no significant difference between cycles rented on working_day and non_working_day')
```

```
t_stat 1.2096277376026694
p_value 0.22644804226361348
Fail to reject the null hypothesis : There is no significant difference between cycles rented on working_day and non_working_day
```

Insights:

In both t test and MannWhitney-u-Rank test we find out that the number of bikes rented is same on working and non working days.

ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season (10 points)

The one-way ANOVA compares the means between the groups you are interested in and determines whether any of those means are statistically significantly different from each other.

STEP-1 : Set up Null Hypothesis Null Hypothesis( $H_0$ ) - The mean of bikes rented is same for across various Seasons.

Alternate Hypothesis( $H_a$ ) - The mean number of bikes rented is different for across various seasons.

STEP-2 : Checking for basic assumptions for the hypothesis

- Normality check using QQ Plot. If the distribution is not normal, use BOX-COX transform to transform it to normal distribution.
- Homogeneity of Variances using Levene's test

Each observations are independent

STEP-3: Dene Test statistics; Distribution of T under  $H_0$ . The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

STEP-4: Decide the kind of test. We will be performing right tailed t-test becuz of the data right skewness

STEP-5: Compute the p-value and x value of alpha. we will be computing the anova-test p-value using the `f_oneway` function using `scipy.stats`. We set our alpha to be 0.05

STEP-6: Compare p-value and alpha. Based on p-value, we will accept or reject  $H_0$ .  $p\text{-val} > \alpha$  : Accept  $H_0$   $p\text{-val} < \alpha$  : Reject  $H_0$

To properly conduct an ANOVA (Analysis of Variance), it's important to check the assumptions underlying the test to ensure its validity. The main assumptions are:

#### 1. Independence of Observations:


Each observation (e.g., total number of cycles rented) must be independent of others. This assumption is typically addressed during data collection, so we generally do not test it directly. However, you should ensure that there are no dependencies between the observations, like repeated measures from the same subjects.

2. Normality: The data within each group (i.e., for each category of weather and season) should be approximately normally distributed. You can test normality using the Shapiro-Wilk test or visual checks like QQ plots.

#### 3. Homogeneity of Variance :

The variances of the groups should be roughly equal. You can test for this assumption using the Levene's test

```
df['weather'].value_counts()
```



	count
weather	
clear	7192
cloudy	2834
light_rain	859
heavy_rain	1

dtype: int64

```
from scipy.stats import shapiro, probplot
import matplotlib.pyplot as plt
import numpy as np
```

```
# Check normality for each group in 'weather'
for weather_group in df['weather'].unique():
    group_data = df[df['weather'] == weather_group]['Total_users']

    # Skip Shapiro-Wilk test and QQ plot if less than 3 data points
    if len(group_data) < 3:
        print(f'Not enough data for weather group: {weather_group} to perform shapiro wilk test and qq plot')
        continue

    # Shapiro-Wilk test for normality
    stat, p_value = shapiro(group_data)
    print(f'shapiro_stat for {weather_group}')
    print(f'stat: {stat}')
    print(f'p_value: {p_value}')

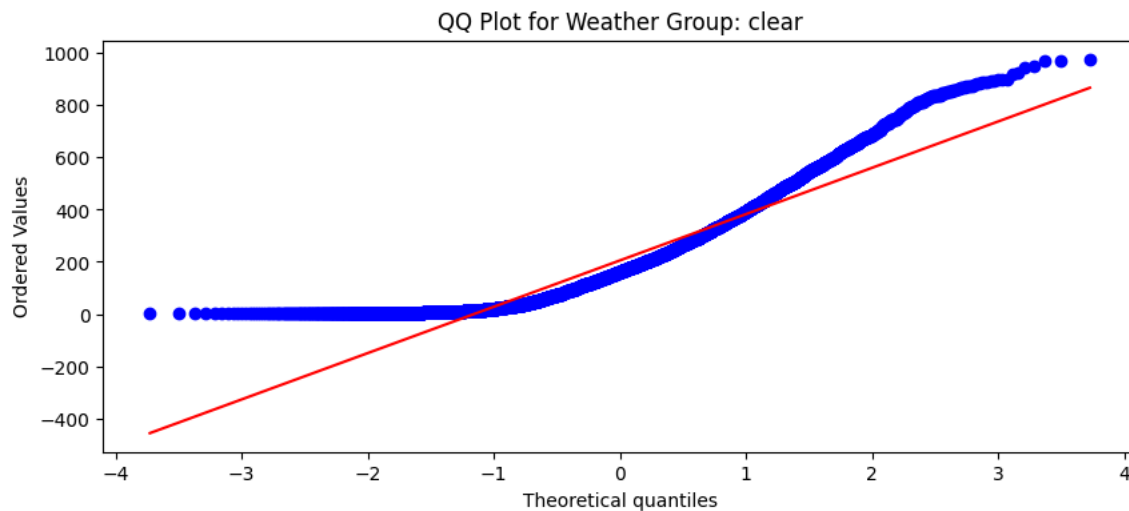
    if p_value > 0.05:
```

```
    print('Data is normally distributed')
else:
    print('Data is not normally distributed')

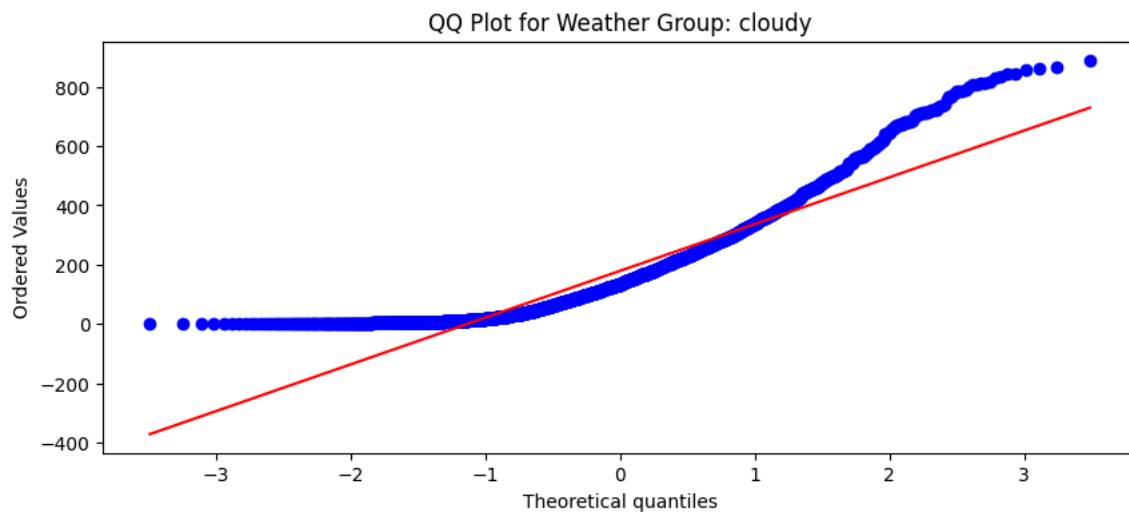
# QQ plot for visual check of normality
plt.figure(figsize=(10,4))
probplot(group_data,dist='norm',plot=plt)
plt.title(f'QQ Plot for Weather Group: {weather_group}')
plt.show()
```



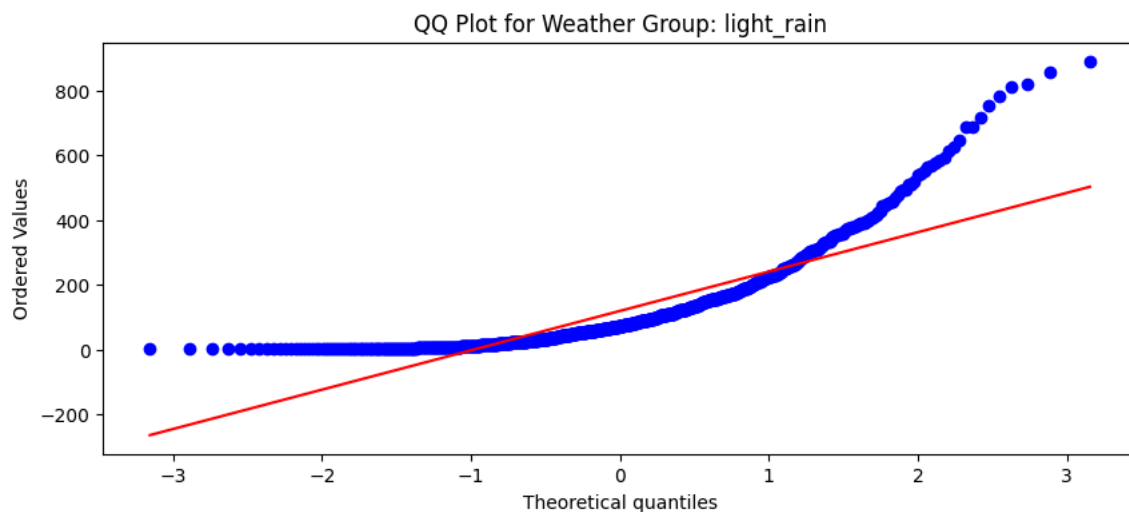
```
shapiro_stat for clear  
stat: 0.8909259459740138  
p_value: 1.5964921477006555e-57  
Data is not normally distributed
```



```
shapiro_stat for cloudy  
stat: 0.8767694973495206  
p_value: 9.777839106111785e-43  
Data is not normally distributed
```



```
shapiro_stat for light_rain  
stat: 0.7674327906035717  
p_value: 3.875893017396149e-33  
Data is not normally distributed
```



Not enough data for weather group: heavy\_rain to perform shapiro wilk test and qq plot

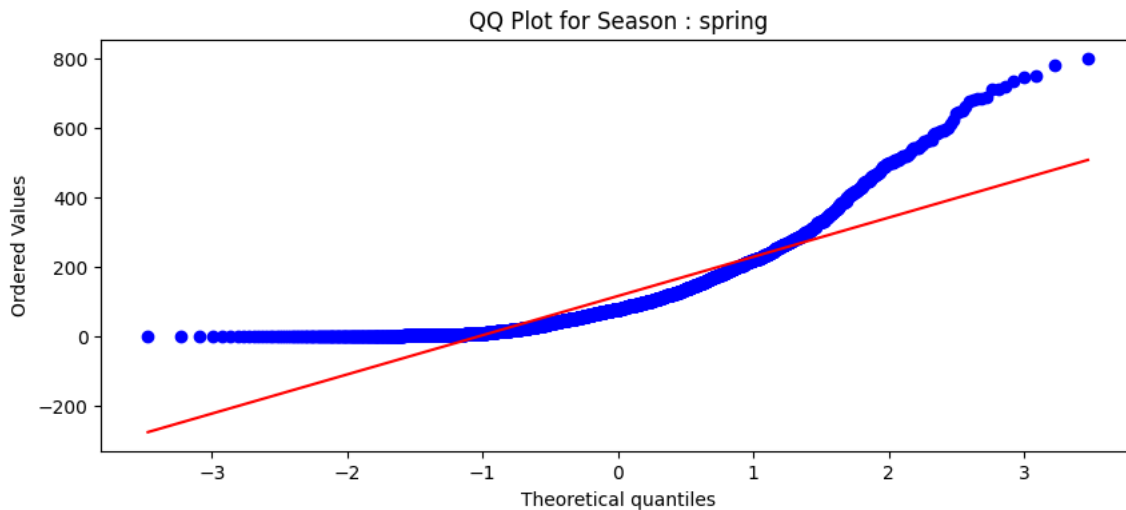
```
# Check normality for each group in 'season'
for season_group in df['season'].unique():
    group_season = df[df['season'] == season_group]['Total_users']

    # Shapiro-Wilk test for normality
    stat, p_value = shapiro(group_season)
    print(f'shapiro_stat for {season_group}')
    print(f'stat: {stat},p_value: {p_value}')

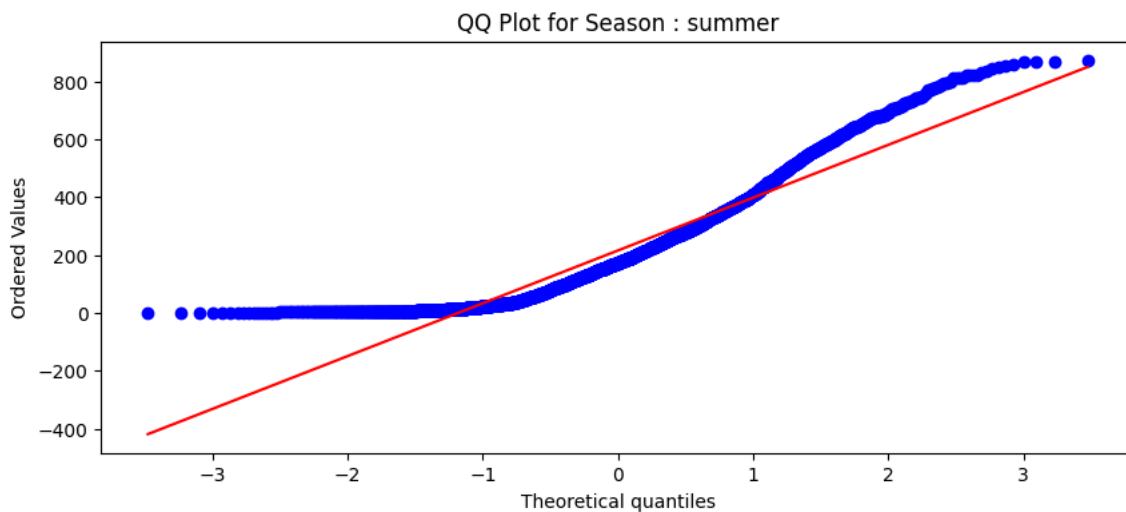
    if p_value > 0.05:
        print('Data is normally distributed')
    else:
        print('Data is not normally distributed')

    # QQ plot for visual check of normality
    plt.figure(figsize=(10,4))
    probplot(group_season,dist='norm',plot=plt)
    plt.title(f'QQ Plot for Season : {season_group}')
    plt.show()
```

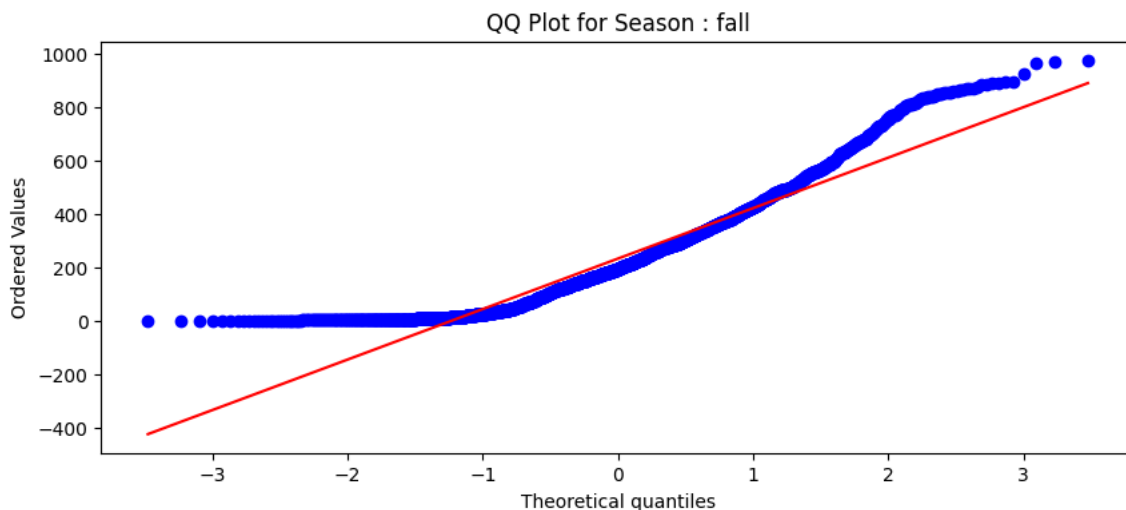
```
shapiro_stat for spring
stat: 0.8087378401253588,p_value: 8.749584618867662e-49
Data is not normally distributed
```



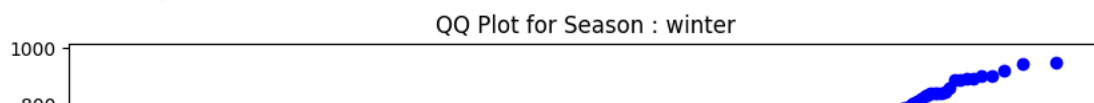
```
shapiro_stat for summer
stat: 0.9004818080893252,p_value: 6.039374406270491e-39
Data is not normally distributed
```

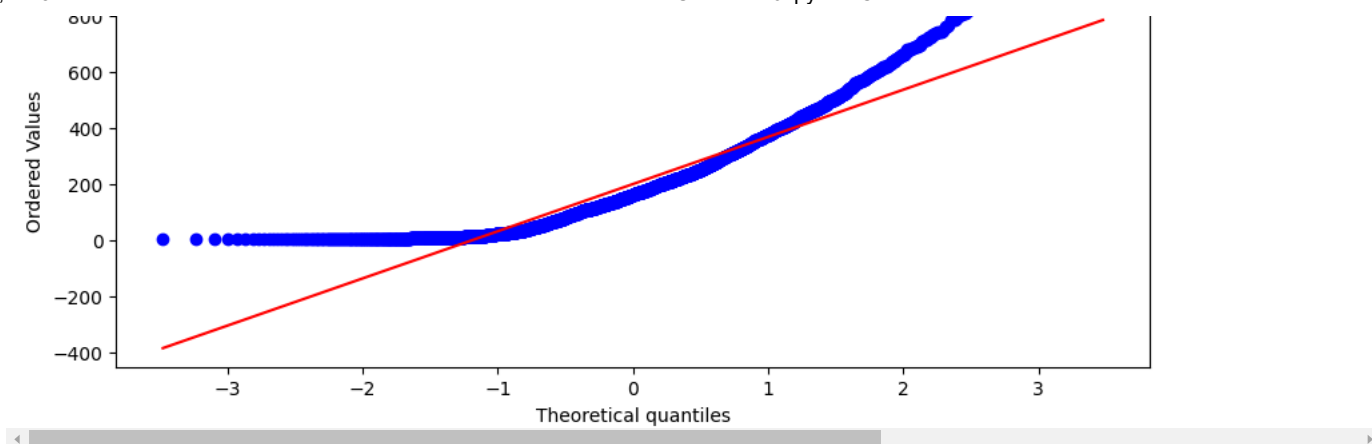


```
shapiro_stat for fall
stat: 0.9148166372899196,p_value: 1.043680518918597e-36
Data is not normally distributed
```



```
shapiro_stat for winter
stat: 0.8954637482095505,p_value: 1.1299244409282836e-39
Data is not normally distributed
```





```
# Levene's test for homogeneity of variance (weather)
group_weather=[df[df['weather']==group]['Total_users'] for group in df['weather'].unique()]
stat,p_value=levene(*group_weather)
print('stat',stat)
print('p_value',p_value)
```

```
if p_value>0.05:
    print('data has equal variance')
else:
    print('data does not have equal variance')
```

```
stat 54.85106195954556
p_value 3.504937946833238e-35
data does not have equal variance
```

```
# Levene's test for homogeneity of variance (season)
group_season=[df[df['season']==group]['Total_users'] for group in df['season'].unique()]
stat,p_value=levene(*group_season)
print('stat',stat)
print('p_value',p_value)
```

```
if p_value>0.05:
    print('data has equal variance')
else:
    print('data does not have equal variance')
```

```
stat 187.7706624026276
p_value 1.0147116860043298e-118
data does not have equal variance
```

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

Since, the data is not normally distributed and violates the assumption of homogeneity of variances, using ANOVA is not appropriate because ANOVA assumes normality and equal variances.

- In such cases, we can apply non-parametric tests, which do not have these assumptions.
- For our case, the Kruskal-Wallis H-test is a suitable non-parametric alternative to ANOVA
- It tests whether there is a significant difference between two or more independent groups.

We will perform both test to just check

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

### 1. Kruskal-Wallis test for weather group

```
from scipy.stats import kruskal
group_weather=[df[df['weather']==group]['Total_users'] for group in df['weather'].unique()]
stat,p_value=kruskal(*group_weather)
print('stat',stat,'p_value',p_value)
```

```

if p_value>0.05:
    print('Fail to reject the null hypothesis:')
    print('There is no significant difference in the number of electric cycles rented on working days and non-working days.')
else:
    print('Reject the null hypothesis:')
    print('There is a significant difference in the number of electric cycles rented on working days and non-working days.')

stat 205.00216514479087 p_value 3.501611300708679e-44
Reject the null hypothesis:
There is a significant difference in the number of electric cycles rented on working days and non-working days.

```

### 1. Kruskal-Wallis test for season group

```

season_group=[df[df['season']==group]['Total_users'] for group in df['season'].unique()]
stat,p_value=kruskal(*season_group)
print('stat',stat,'p_value',p_value)

if p_value>0.05:
    print('Fail to reject the null hypothesis:')
    print('There is no significant difference in the number of electric cycles rented on working days and non-working days.')
else:
    print('Reject the null hypothesis:')
    print('There is a significant difference in the number of electric cycles rented on working days and non-working days.')

stat 699.6668548181988 p_value 2.479008372608633e-151
Reject the null hypothesis:
There is a significant difference in the number of electric cycles rented on working days and non-working days.

```

```

# Post-hoc analysis (Mann-Whitney U test) if Kruskal-Wallis test is significant
import itertools

```

```

group_pairs=list(itertools.combinations(df['weather'].unique(),2))

```

```

for group1,group2 in group_pairs:
    group1_data=df[df['weather']==group1]['Total_users']
    group2_data=df[df['weather']==group2]['Total_users']

```

```

# Mann-Whitney U test
u_stat, p_value = mannwhitneyu(group1_data, group2_data)
print(f'Mann-Whitney U test for {group1} and {group2}:')
print(f'U-statistic: {u_stat}, p-value: {p_value}')

```

```

if p_value < 0.05:
    print(f'Reject the null hypothesis: There is a significant difference between {group1} and {group2}')
else:
    print(f'Fail to reject the null hypothesis: There is no significant difference between {group1} and {group2}')

```

```

Mann-Whitney U test for clear and cloudy:
U-statistic: 10906755.0, p-value: 4.154401456746316e-08
Reject the null hypothesis: There is a significant difference between clear and cloudy
Mann-Whitney U test for clear and light_rain:
U-statistic: 3980116.0, p-value: 1.4378281276950748e-43
Reject the null hypothesis: There is a significant difference between clear and light_rain
Mann-Whitney U test for clear and heavy_rain:
U-statistic: 3556.5, p-value: 0.9850147504225071
Fail to reject the null hypothesis: There is no significant difference between clear and heavy_rain
Mann-Whitney U test for cloudy and light_rain:
U-statistic: 1497645.0, p-value: 1.251432299071393e-24
Reject the null hypothesis: There is a significant difference between cloudy and light_rain
Mann-Whitney U test for cloudy and heavy_rain:
U-statistic: 1241.5, p-value: 0.8306748017102786
Fail to reject the null hypothesis: There is no significant difference between cloudy and heavy_rain
Mann-Whitney U test for light_rain and heavy_rain:
U-statistic: 208.0, p-value: 0.37333987515235556
Fail to reject the null hypothesis: There is no significant difference between light_rain and heavy_rain

```

```

# kruskal wallis test for season
season_group_pairs=list(itertools.combinations(df['season'].unique(),2))

```

```

for group1,group2 in season_group_pairs:
    group1_data=df[df['season']==group1]['Total_users']
    group2_data=df[df['season']==group2]['Total_users']

```

```

# Mann-Whitney U test
u_stat, p_value = mannwhitneyu(group1_data, group2_data)
print(f'Mann-Whitney U test for {group1} and {group2}:')

```

```

if p_value < 0.05:
    print(f'Reject the null hypothesis: There is a significant difference between {group1} and {group2}')
else:
    print(f'Fail to reject the null hypothesis: There is no significant difference between {group1} and {group2}')

```

```

→ Mann-Whitney U test for spring and summer:
Reject the null hypothesis: There is a significant difference between spring and summer
Mann-Whitney U test for spring and fall:
Reject the null hypothesis: There is a significant difference between spring and fall
Mann-Whitney U test for spring and winter:
Reject the null hypothesis: There is a significant difference between spring and winter
Mann-Whitney U test for summer and fall:
Reject the null hypothesis: There is a significant difference between summer and fall
Mann-Whitney U test for summer and winter:
Reject the null hypothesis: There is a significant difference between summer and winter
Mann-Whitney U test for fall and winter:
Reject the null hypothesis: There is a significant difference between fall and winter

```

## one-way ANOVA

Before doing Anova test lets transform data into normality using BOX-COX transformation

```

import numpy as np
import pandas as pd
from scipy.stats import boxcox

# Create a function to apply Box-Cox transformation
def boxcox_transformation(data):
    # Check if data is constant
    if len(np.unique(data)) == 1:
        raise ValueError("Data must not be constant.")

    # Check if data contains non-positive values and adjust it
    if (data <= 0).any():
        # Add a constant to make data strictly positive
        data = data + abs(data.min()) + 1

    # Apply the Box-Cox transformation
    transformed_data, lambda_value = boxcox(data)

    # Return transformed data and lambda value
    return transformed_data, lambda_value

# Initialize containers for transformed data and lambda values
transformed_weather_data = {}
lambda_values_weather = {}

# Apply Box-Cox transformation for 'Total_users' by weather and handle constant data
for weather_condition in df['weather'].unique():
    data_w = df[df['weather'] == weather_condition]['Total_users']

    try:
        # Apply the Box-Cox transformation
        transformed_data, lambda_value = boxcox_transformation(data_w)

        # Store results
        transformed_weather_data[weather_condition] = transformed_data
        lambda_values_weather[weather_condition] = lambda_value

        # Print the results for the current weather condition
        print(f"Weather: {weather_condition} - Box-Cox lambda: {lambda_value:.4f}")

    except ValueError as e:
        # Handle cases where the data is constant or invalid for Box-Cox transformation
        print(f"Weather: {weather_condition} - {e}")

# Apply Box-Cox transformation for 'Total_users' by season (same as weather)
transformed_season_data = {}
lambda_values_season = {}

for season_condition in df['season'].unique():
    data_s = df[df['season'] == season_condition]['Total_users']

    try:
        # Apply the Box-Cox transformation

```

```

transformed_data, lambda_value = boxcox_transformation(data_s)

# Store results
transformed_season_data[season_condition] = transformed_data
lambda_values_season[season_condition] = lambda_value

# Print the results for the current season condition
print(f"Season: {season_condition} - Box-Cox lambda: {lambda_value:.4f}")

except ValueError as e:
    # Handle cases where the data is constant or invalid for Box-Cox transformation
    print(f"Season: {season_condition} - {e}")

→ Weather: clear - Box-Cox lambda: 0.3382
Weather: cloudy - Box-Cox lambda: 0.3177
Weather: light_rain - Box-Cox lambda: 0.2074
Weather: heavy_rain - Data must not be constant.
Season: spring - Box-Cox lambda: 0.2633
Season: summer - Box-Cox lambda: 0.3492
Season: fall - Box-Cox lambda: 0.3934
Season: winter - Box-Cox lambda: 0.3532

# shapiro wilk test on weather transformed data to check data normality
from scipy.stats import shapiro,probplot

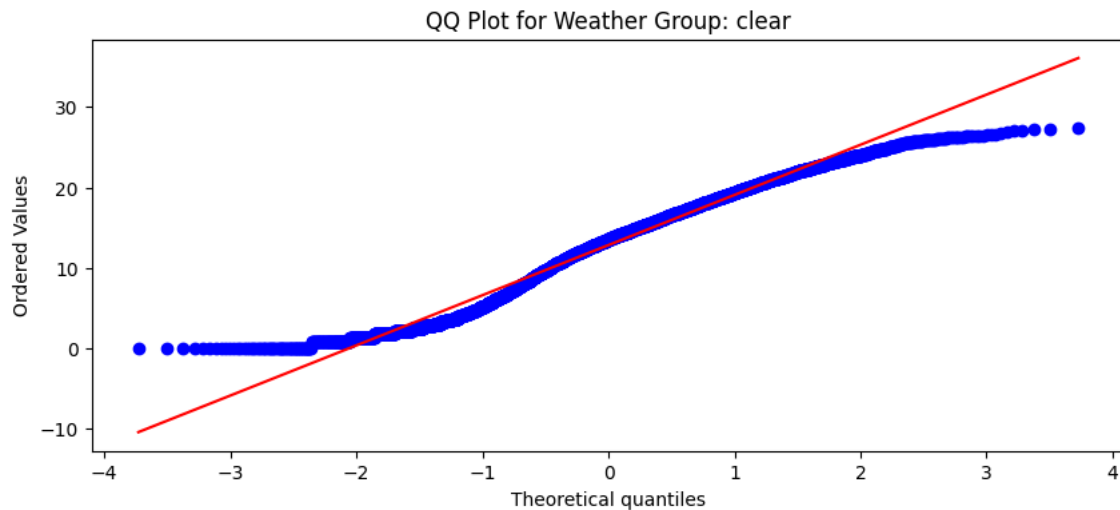
for weather_condition, transformed_data in transformed_weather_data.items():
    # Perform Shapiro-Wilk test
    stat,p_value=shapiro(transformed_weather_data[weather_condition])
    print(f'shapiro_stat for {weather_condition}')
    print(f'stat: {stat},p_value: {p_value}')

    if p_value > 0.05:
        print('Data is normally distributed')
    else:
        print('Data is not normally distributed')

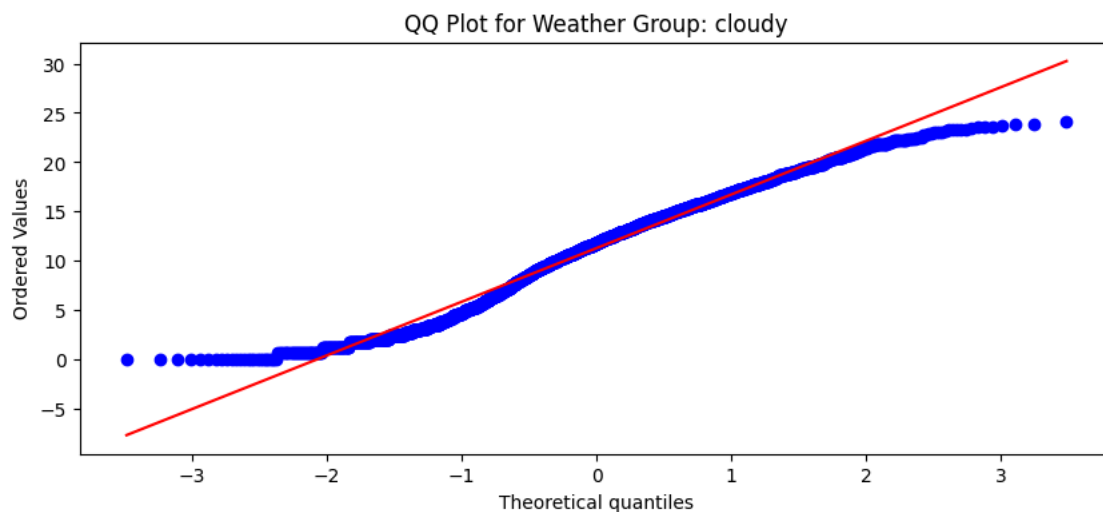
# Q-Q plot to check data normality
plt.figure(figsize=(10,4))
probplot(transformed_weather_data[weather_condition],dist='norm',plot=plt)
plt.title(f'QQ Plot for Weather Group: {weather_condition}')
plt.show()
# to get tight layout
plt.tight_layout()

```

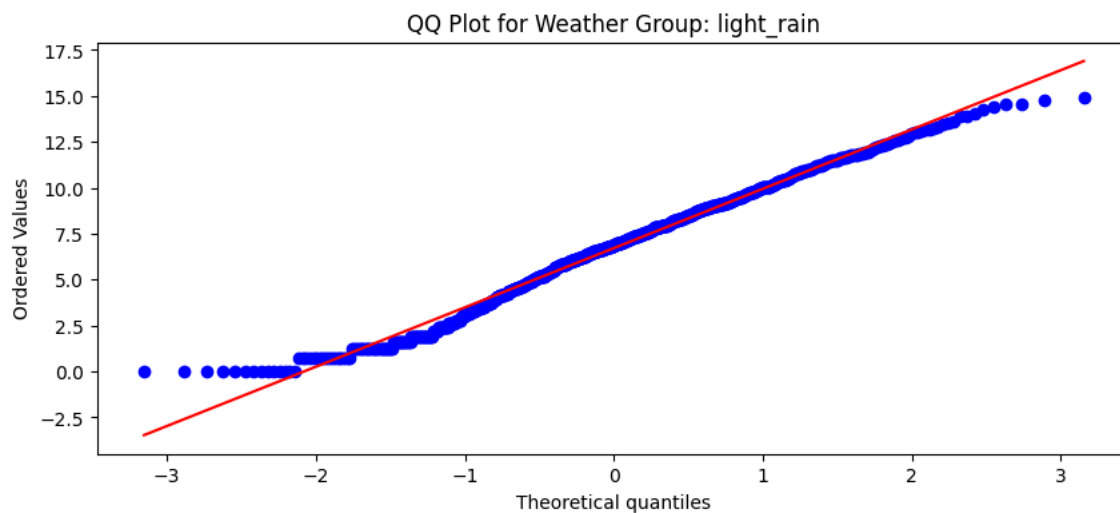
```
shapiro_stat for clear
stat: 0.9771627462003276, p_value: 2.0663779794427112e-32
Data is not normally distributed
```



```
shapiro_stat for cloudy
stat: 0.9802164862657785, p_value: 1.9245396515414659e-19
Data is not normally distributed
<Figure size 640x480 with 0 Axes>
```



```
shapiro_stat for light_rain
stat: 0.987789103522957, p_value: 1.4117548371138875e-06
Data is not normally distributed
<Figure size 640x480 with 0 Axes>
```



<Figure size 640x480 with 0 Axes>

```
# shapiro wilk test on weather transformed data to check data normality
from scipy.stats import shapiro, probplot
```



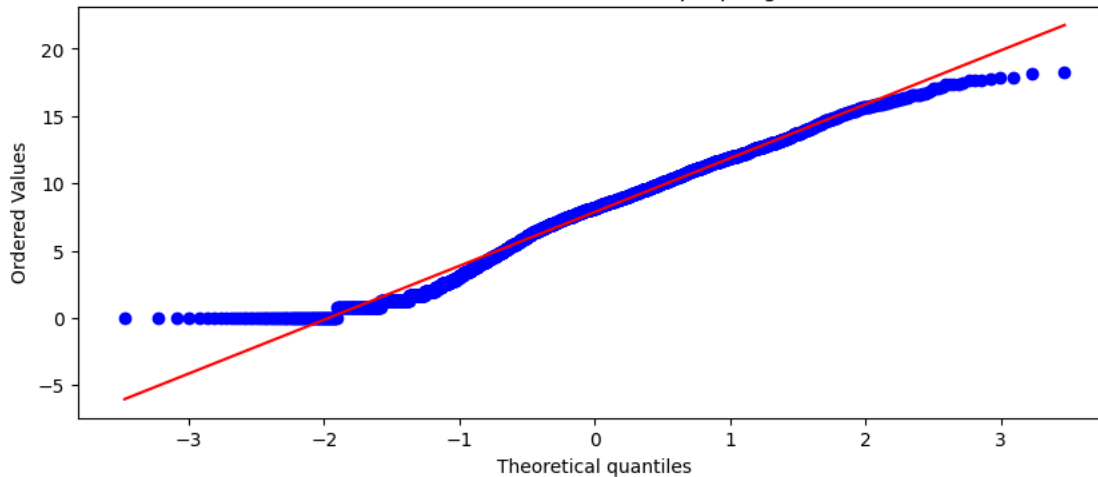
```
for season_condition, transformed_data in transformed_season_data.items():
    # Perform Shapiro-Wilk test
    stat,p_value=shapiro(transformed_season_data[season_condition])
    print(f'shapiro_stat for {season_condition}')
    print(f'stat: {stat},p_value: {p_value}')

    if p_value > 0.05:
        print('Data is normally distributed')
    else:
        print('Data is not normally distributed')

    # Q-Q plot to check data normality
    plt.figure(figsize=(10,4))
    probplot(transformed_season_data[season_condition],dist='norm',plot=plt)
    plt.title(f'QQ Plot for season Group: {season_condition}')
    plt.show()
    # to get tight layout
    plt.tight_layout()
```

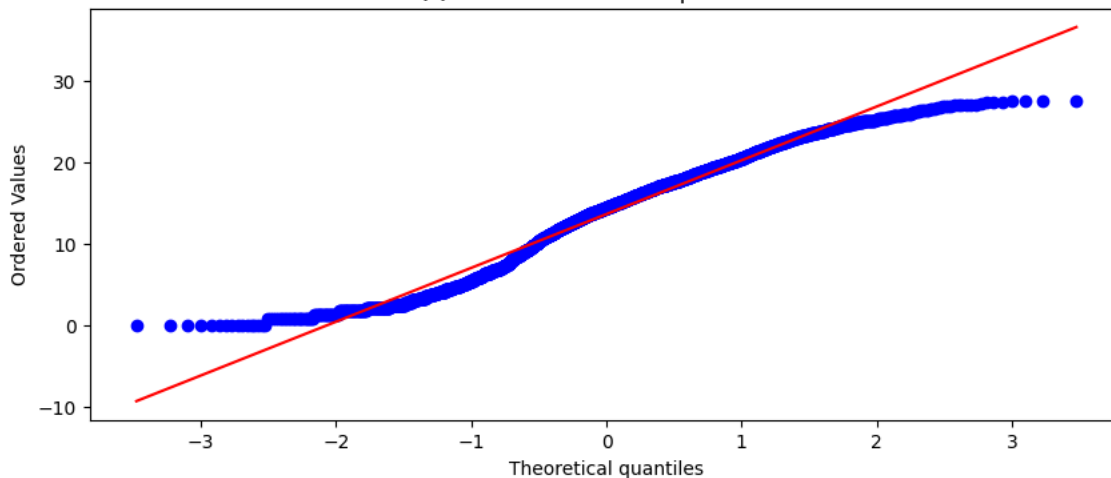
```
shapiro_stat for spring
stat: 0.9828613151293278,p_value: 1.7152265938758798e-17
Data is not normally distributed
```

QQ Plot for season Group: spring



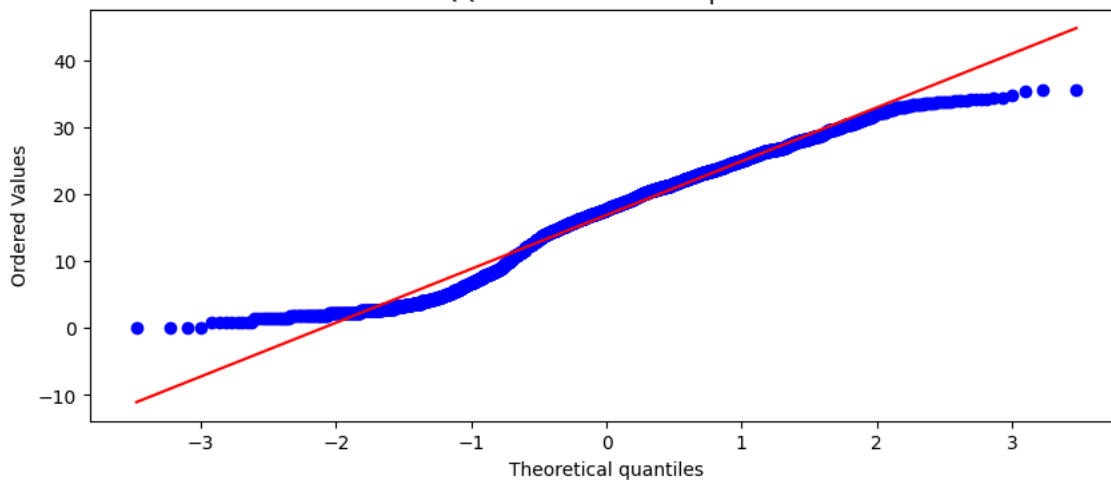
```
shapiro_stat for summer
stat: 0.9730870955575379,p_value: 2.785605703808764e-22
Data is not normally distributed
<Figure size 640x480 with 0 Axes>
```

QQ Plot for season Group: summer



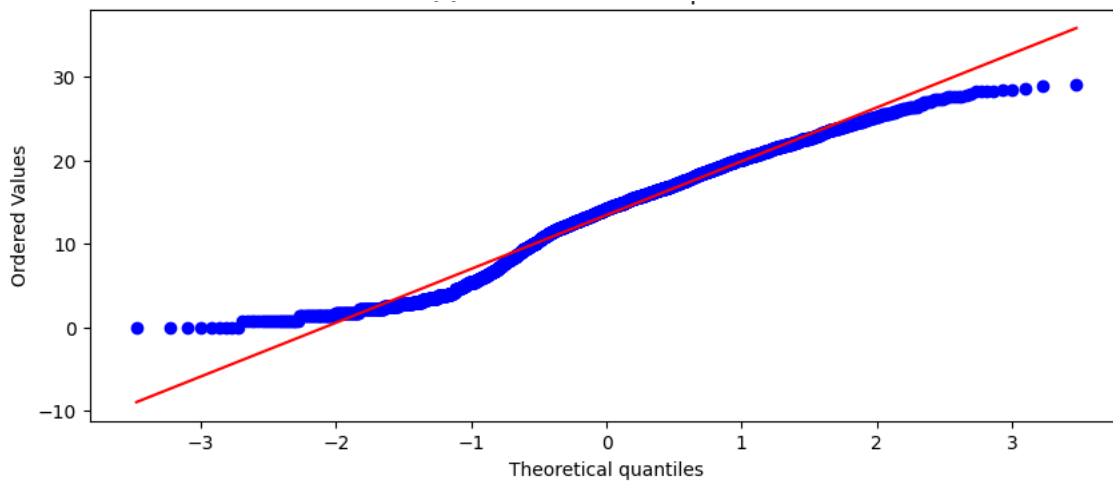
```
shapiro_stat for fall
stat: 0.9733730176318035,p_value: 3.6387973034430233e-22
Data is not normally distributed
<Figure size 640x480 with 0 Axes>
```

QQ Plot for season Group: fall



```
shapiro_stat for winter
stat: 0.9763012843901773,p_value: 6.333511688881777e-21
Data is not normally distributed
<Figure size 640x480 with 0 Axes>
```

QQ Plot for season Group: winter



<Figure size 640x480 with 0 Axes>

```
# after transformation, check for data normality
# checking normality for transformed data in weather

for weather_condition, transformed_data in transformed_weather_data.items():
    # Perform Shapiro-Wilk test
    stat,p_value=shapiro(transformed_weather_data[weather_condition])
    print(f'shapiro_stat for {weather_condition}')
    print(f'stat: {stat},p_value: {p_value}')

    if p_value > 0.05:
        print('Data is normally distributed')
    else:
        print('Data is not normally distributed')

# checking normality for transformed data in weather

for season_condition, transformed_data in transformed_season_data.items():
    # Perform Shapiro-Wilk test
    stat,p_value=shapiro(transformed_season_data[season_condition])
    print(f'shapiro_stat for {season_condition}')
    print(f'stat: {stat},p_value: {p_value}')

    if p_value > 0.05:
        print('Data is normally distributed')
        print()
    else:
        print('Data is not normally distributed')
        print()
```

```
shapiro_stat for clear
stat: 0.9771627462003276,p_value: 2.0663779794427112e-32
Data is not normally distributed
shapiro_stat for cloudy
stat: 0.9802164862657785,p_value: 1.9245396515414659e-19
Data is not normally distributed
shapiro_stat for light_rain
stat: 0.987789103522957,p_value: 1.4117548371138875e-06
Data is not normally distributed
shapiro_stat for spring
stat: 0.9828613151293278,p_value: 1.7152265938758798e-17
Data is not normally distributed

shapiro_stat for summer
stat: 0.9730870955575379,p_value: 2.785605703808764e-22
Data is not normally distributed

shapiro_stat for fall
stat: 0.9733730176318035,p_value: 3.6387973034430233e-22
Data is not normally distributed

shapiro_stat for winter
stat: 0.9763012843901773,p_value: 6.333511688881777e-21
Data is not normally distributed
```

oth original data and the Boxcoxed data doesn't follow the normal distribution, so We cannot performAnova. We will have to go with the Kruskel-Wallis H Test.

```
# performing kruskal wallis test on season and weather both
from scipy.stats import kruskal

kruskal_result_weather = kruskal(*transformed_weather_data.values())
kruskal_result_season = kruskal(*transformed_season_data.values())

print("Kruskal-Wallis H-test for Weather:")
print("Statistic:", kruskal_result_weather.statistic)
print("p-value:", kruskal_result_weather.pvalue)

if kruskal_result_weather.pvalue<0.05:
    print('Reject the null hypothesis:Reject the null hypothesis: Number of cycles rented differs across weather conditions')
else:
    print('Fail to reject the null hypothesis: ')

print("\nKruskal-Wallis H-test for Season:")
print("Statistic:", kruskal_result_season.statistic)
print("p-value:", kruskal_result_season.pvalue)

if kruskal_result_season.pvalue<0.05:
    print('Reject the null hypothesis:Reject the null hypothesis: Number of cycles rented differs across weather conditions')
else:
    print('Fail to reject the null hypothesis: ')
```

```

Kruskal-Wallis H-test for Weather:
Statistic: 853.0447712977627
p-value: 5.803386625002815e-186
Reject the null hypothesis:Reject the null hypothesis: Number of cycles rented differs across weather conditions

Kruskal-Wallis H-test for Season:
Statistic: 2141.495739207715
p-value: 0.0
Reject the null hypothesis:Reject the null hypothesis: Number of cycles rented differs across weather conditions

```

Insights:

From the Krukal-walis test , we can confirm that The mean number of E-bikes rented differs across various Seasons and weather.

Chi-square test to check if Weather is dependent on the season

STEP-1 : Set up Null Hypothesis

Null Hypothesis (  $H_0$  ) - weather is independent of season

Alternate Hypothesis (  $H_a$  ) -weather is dependent of seasons

STEP-2: Checking for basic assumptions for the hypothesis (Non-Parametric Test)

- The data in the cells should be frequencies, or counts of cases. The levels (or categories) of the variables are mutually exclusive. That is, a particular subject ts into one and only one level of each of the variables.
- There are 2 variables, and both are measured as categories.
- The value of the cell expecteds should be 5 or morein at least 80% of the cells, and no cell should have an expected of less than one (3).

STEP-3: Dene Test statistics; Distribution of T under  $H_0$ .

The test statistic for a Chi- square test . Under  $H_0$ , the test statistic should follow Chi-Square Distribution.

STEP-4: Decide the kind of test.

Here we will perform the chisquare Test of independence (i.e) chi2\_contingency

STEP-5: Compute the p-value and x value of alpha.

- we will be computing the chi square-test p-value using the chi 2 function using scipy.stats. We setour alpha to be 0.05

STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

- $p\text{-val} > \alpha$  : Accept H0
- $p\text{-val} < \alpha$  : Reject H0

# creating a contingency table

```
data_ws = pd.crosstab(df['weather'],df['season'])
data_ws
```



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

Next steps:

[Generate code with data\\_ws](#)
[View recommended plots](#)
[New interactive sheet](#)

```
from scipy.stats import chi2_contingency
```

```
# Perform Chi-Square Test of Independence
```

```
chi2_stat,p_value,dof,expected=chi2_contingency(data_ws)
```

```
print('chi_squared test of independence between season and weather')
print('chi2_stat : ',chi2_stat)
print('p_value : ',p_value)
print('dof : ',dof)
```

```
if p_value>0.05:
```

```
    print('Fail to reject the null hypothesis:')
```

```
    print('weather is independent of season')
```

```
else:
```

```
    print('Reject the null hypothesis:')
```

```
    print('weather is dependent of season')
```

```
chi_squared test of independence between season and weather
chi2_stat : 49.15865559689363
p_value : 1.5499250736864862e-07
dof : 9
Reject the null hypothesis:
weather is dependent of season
```

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

## Buisness Insights

1. Maximum bike rentals occur during summer, while the minimum is observed in winter.
2. Clear weather is associated with the highest bike rental counts, whereas rentals sharply decrease in rain, thunderstorm, snow, or fog.
3. Humidity, windspeed, temperature and weather are correlated with season and impacts the count of cycles rented.
4. Lower temperatures correspond to lower bike rentals, and demand rises with increasing temperatures.
5. Bike rentals peak during the day, decline through the night, indicating a pattern fluctuatio
6. Less rentals on holidays and weekends, with a demand increase on non-working days. However, the overall count on working and non-holiday days are similar.
7. Casual riders dominate on weekends, while registered users are more active on working days.
8. The hourly rental count shows impressive annual growth from 2011 to 2012.
9. Approximately 19% of users are casual, and 81% are registered.
10. Notable seasonal patterns, with peak demand in spring and summer, and a decline in fall and winter.
11. January to March sees the lowest rental counts, and a distinctive daily trend shows peak usage during the afternoon.
12. Clear and partly\_cloudy weather correlates with higher rental counts, while extreme weather conditions have limited data representation.
13. Temperature and feeling temperature exhibit a strong positive correlation.
14. Registered Users and Total\_riders exhibit a strong positive correlation as well.
15. Limited correlation observed between weather-related factors and bike rental counts.