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

import warnings
warnings.simplefilter('ignore')
```

In [2]:
```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089')
df.head()
```

Out[2]:

	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 [3]:
```
df_orig = df.copy(deep=True)
```

Problem Statement

Yulu, India's leading micro-mobility provider, aims to analyze the factors impacting demand for shared electric cycles. The company has experienced a revenue decline and seeks insights into how external conditions (weather, holidays, working days, etc.) affect ride demand. This analysis will help optimize operations, pricing, and service availability.

EDA

In [4]:
```
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
```

In [5]:
```
df['datetime'] = pd.to_datetime(df['datetime']) # converting to date time data type
```

In [6]:
```
df.describe().T
```

Out[6]:

	count	mean	min	25%	50%	75%	max	std
datetime	10886	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	10886.0	2.506614	1.0	2.0	3.0	4.0	4.0	1.116174
holiday	10886.0	0.028569	0.0	0.0	0.0	0.0	1.0	0.166599
workingday	10886.0	0.680875	0.0	0.0	1.0	1.0	1.0	0.466159
weather	10886.0	1.418427	1.0	1.0	1.0	2.0	4.0	0.633839
temp	10886.0	20.23086	0.82	13.94	20.5	26.24	41.0	7.79159
atemp	10886.0	23.655084	0.76	16.665	24.24	31.06	45.455	8.474601
humidity	10886.0	61.88646	0.0	47.0	62.0	77.0	100.0	19.245033
windspeed	10886.0	12.799395	0.0	7.0015	12.998	16.9979	56.9969	8.164537
casual	10886.0	36.021955	0.0	4.0	17.0	49.0	367.0	49.960477
registered	10886.0	155.552177	0.0	36.0	118.0	222.0	886.0	151.039033
count	10886.0	191.574132	1.0	42.0	145.0	284.0	977.0	181.144454

from date time description, we have 2 years of data (2011 and 2012)

In [7]:
```
df.shape # based on above table, there are no null values
```

Out[7]:
```
(10886, 12)
```

In [8]:
```
df.nunique()
```

Out[8]:

	0
datetime	10886
season	4
holiday	2
workingday	2
weather	4
temp	49
atemp	60
humidity	89
windspeed	28
casual	309
registered	731
count	822

dtype: int64

In [9]:
```
if df.shape[0] == df['datetime'].nunique():
    print("No duplicate rows found")
else:
    raise AssertionError("Duplicate rows found")
```

```
No duplicate rows found
```

In [10]:
```
df['year'] = df['datetime'].dt.year
df['month'] = df['datetime'].dt.month  # January = 1, December = 12
df['day_of_week'] = df['datetime'].dt.dayofweek  # Monday = 0, Sunday = 6
df['hour'] = df['datetime'].dt.hour # 0 - 23
```

We do cyclic transformation because days, months or hours because without transformation, sunday (6) and monday (0) seems far apart even though they are next to each other.

In [11]:
```
df.columns
```

Out[11]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count',
       'year', 'month', 'day_of_week', 'hour'],
      dtype='object')
```

In [12]:
```
categorical_cols = ['season', 'holiday', 'workingday', 'weather', 'day_of_week', 'month', 'hour']
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count', 'datetime']
```

In [13]:
```
df.isna().sum() # check Nan values
```

Out[13]:

	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
year	0
month	0
day_of_week	0
hour	0

dtype: int64

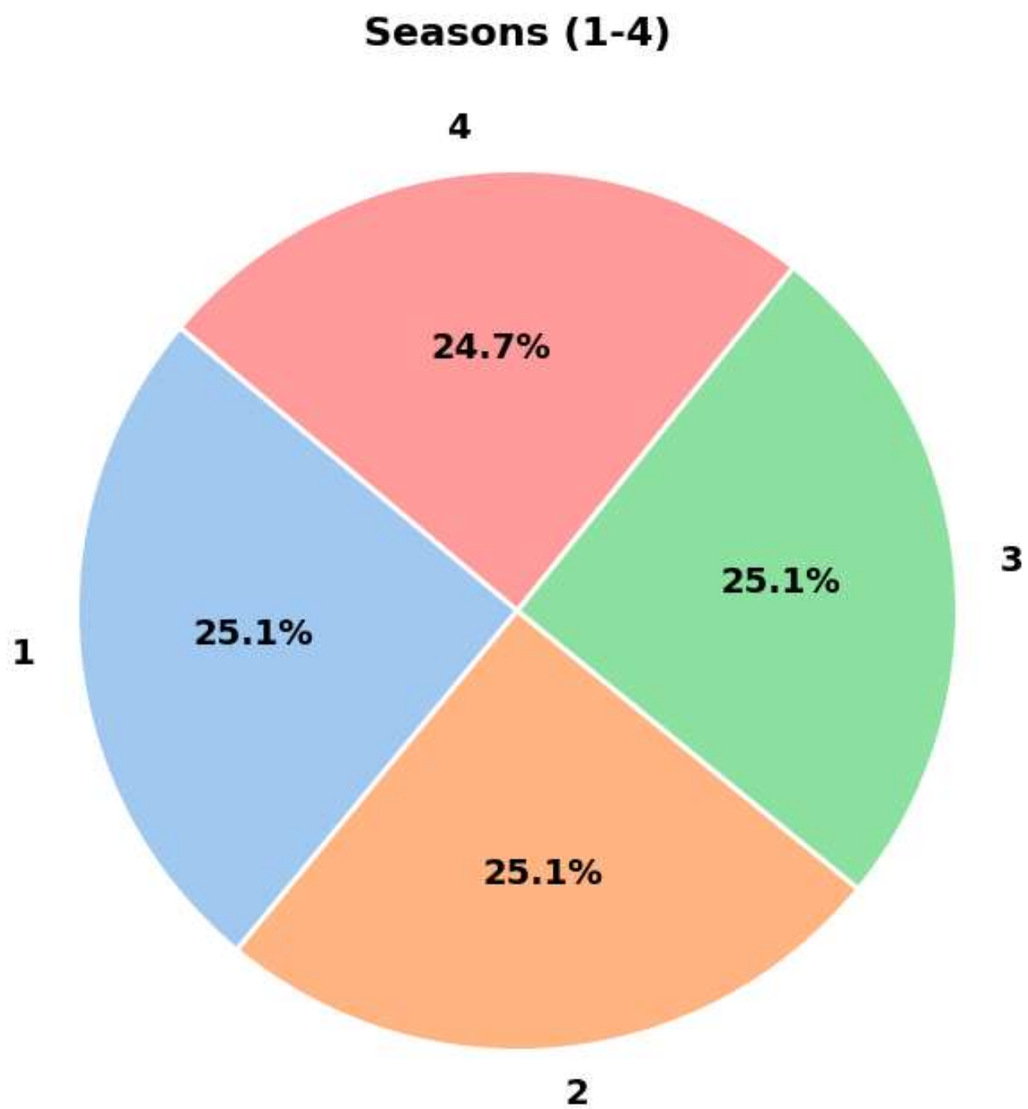
No NAN values, so we need not handle them

univariate analysis

In [14]:
```
def plot_pie_chart(df, column_name, title):
  labels = df[column_name].unique()
  sizes = df[column_name].value_counts()

  # Define a beautiful color palette
  colors = sns.color_palette("pastel")[:len(labels)]

  # Create the pie chart
  plt.figure(figsize=(8, 8))
  wedges, texts, autotexts = plt.pie(
      sizes, labels=labels, autopct='%1.1f%%', startangle=140,
      colors=colors, wedgeprops={'edgecolor': 'white', 'linewidth': 2}
  )

  # Improve text appearance
  for text in texts + autotexts:
      text.set_fontsize(14)
      text.set_fontweight('bold')
```

```
  # Add a title
  plt.title(title, fontsize=16, fontweight='bold') #0- weekends / holiday , 1 - working day

  # Show the plot
  plt.show()
```

In [15]:
```
plot_pie_chart(df, 'season', 'Seasons (1-4)')
```



From the 2 years data, the seasons are fairly distributed

In [16]:
```
x = df.groupby(['year', 'month', 'day_of_week', 'hour'])['datetime'].count()
```

In [17]:
```
full_date_range = pd.date_range(start = df['datetime'].min(), end=df['datetime'].max(), freq='h')
missing_dates = full_date_range[full_date_range.isin(df['datetime']) == False]
missing_dates
```

Out[17]:
```
DatetimeIndex(['2011-01-02 05:00:00', '2011-01-03 02:00:00',
               '2011-01-03 03:00:00', '2011-01-04 03:00:00',
               '2011-01-05 03:00:00', '2011-01-06 03:00:00',
               '2011-01-07 03:00:00', '2011-01-11 03:00:00',
               '2011-01-11 04:00:00', '2011-01-12 03:00:00',
               ...
               '2012-11-30 14:00:00', '2012-11-30 15:00:00',
               '2012-11-30 16:00:00', '2012-11-30 17:00:00',
               '2012-11-30 18:00:00', '2012-11-30 19:00:00',
               '2012-11-30 20:00:00', '2012-11-30 21:00:00',
               '2012-11-30 22:00:00', '2012-11-30 23:00:00'],
              dtype='datetime64[ns]', length=6370, freq=None)
```

The date time is not continuous (every hour for 2 years). From the shape of above output, we have 6370 missing entries

In [18]:
```
df['hourly_diff'] = df['datetime'].diff().dt.total_seconds() / 3600
df[df['hourly_diff'] >= 2]
```

Out[18]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	day_of_week	hour	hourly_diff
29	2011-01-02 06:00:00	1	0	0	3	17.22	21.210	77	19.9995	0	2	2	2011	1	6	6	2.0
49	2011-01-03 04:00:00	1	0	1	1	6.56	6.820	47	26.0027	0	1	1	2011	1	0	4	3.0
72	2011-01-04 04:00:00	1	0	1	1	5.74	9.090	63	6.0032	0	2	2	2011	1	1	4	2.0
95	2011-01-05 04:00:00	1	0	1	1	9.84	11.365	48	15.0013	0	2	2	2011	1	2	4	2.0
118	2011-01-06 04:00:00	1	0	1	2	6.56	9.850	64	6.0032	0	1	1	2011	1	3	4	2.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9063	2012-09-01 00:00:00	3	0	0	1	30.34	34.090	62	7.0015	22	146	168	2012	9	5	0	289.0
9519	2012-10-01 00:00:00	4	0	1	1	18.86	22.725	72	7.0015	6	39	45	2012	10	0	0	265.0
9975	2012-11-01 00:00:00	4	0	1	1	14.76	18.180	57	6.0032	8	52	60	2012	11	3	0	289.0
10146	2012-11-08 04:00:00	4	0	1	2	12.30	14.395	45	19.0012	1	9	10	2012	11	3	4	2.0
10430	2012-12-01 00:00:00	4	0	0	1	10.66	15.150	81	0.0000	9	99	108	2012	12	5	0	265.0

65 rows × 17 columns

In [19]:
```
df[df['hourly_diff'] >=2]['hourly_diff'].value_counts().sort_values(ascending=True)
```

Out[19]:

hourly_diff	count
217.0	1
241.0	1
13.0	1
3.0	5
265.0	8
289.0	13
2.0	36

dtype: int64

From above table, we can understand that,the most common missing interval is 2 hours. This suggests that short gaps (like 2 hours) are frequent, but some data points are missing for very long periods (9–12 days).

Let's understand if the large gaps (missing dates more than half-aday) is uniformly distributed or repeating on certain days

In [20]:
```
df['daily_diff'] = df['hourly_diff'] / 24
date_gaps = df[df['hourly_diff'] >= 2][['datetime','daily_diff','holiday','workingday','season','humidity','windspeed','count','hourly_diff']]
date_gaps = date_gaps[date_gaps['datetime'] > '2011-01-31']  # Exclude Jan 2011
date_gaps = date_gaps[date_gaps['daily_diff']>=0.5]
date_gaps
```

Out[20]:

	datetime	daily_diff	holiday	workingday	season	humidity	windspeed	count	hourly_diff
431	2011-02-01	12.041667	0	1	1	64	7.0015	8	289.0
877	2011-03-01	9.041667	0	1	1	70	31.0009	7	217.0
1323	2011-04-01	12.041667	0	1	2	100	11.0014	6	289.0
1778	2011-05-01	11.041667	0	0	2	67	6.0032	96	265.0
2234	2011-06-01	12.041667	0	1	2	79	11.0014	34	289.0
2690	2011-07-01	11.041667	0	1	3	50	0.0000	68	265.0
3146	2011-08-01	12.041667	0	1	3	70	15.0013	29	289.0
3602	2011-09-01	12.041667	0	1	3	78	7.0015	51	289.0
4055	2011-10-01	11.041667	0	0	4	63	26.0027	130	265.0
4510	2011-11-01	12.041667	0	1	4	87	11.0014	21	289.0
4966	2011-12-01	11.041667	0	1	4	52	22.0028	20	265.0
5422	2012-01-01	12.041667	0	0	1	66	0.0000	48	289.0
5875	2012-02-01	12.041667	0	1	1	38	19.0012	31	289.0
6330	2012-03-01	10.041667	0	1	1	94	0.0000	11	241.0
6785	2012-04-01	12.041667	0	0	2	76	16.9979	67	289.0
7239	2012-05-01	11.041667	0	1	2	59	12.9980	35	265.0
7695	2012-06-01	12.041667	0	1	2	50	16.9979	86	289.0
8151	2012-07-01	11.041667	0	0	3	66	0.0000	149	265.0
8607	2012-08-01	12.041667	0	1	3	79	11.0014	47	289.0
9063	2012-09-01	12.041667	0	0	3	62	7.0015	168	289.0
9519	2012-10-01	11.041667	0	1	4	72	7.0015	45	265.0
9975	2012-11-01	12.041667	0	1	4	57	6.0032	60	289.0
10430	2012-12-01	11.041667	0	0	4	81	0.0000	108	265.0

In [21]:
```
plt.figure(figsize=(12, 5))
plt.plot(date_gaps['datetime'], date_gaps['daily_diff'], marker='o', linestyle='-')
plt.xlabel("Date")
plt.ylabel("Daily Difference")
plt.title("Trend of Daily Difference Over Time")
plt.xticks(df[df['daily_diff'] >=1]['datetime'], rotation=90)
plt.yticks(range(0,14))
plt.grid()
plt.show()
```

The first month has less sum of missing days because 1st daily_diff value is NAN (as we did date diff)

Does missing dates areuniform across all days of month?

Null Hypothesis (H_0): The gaps are uniformly distributed across all days of the month (1–31), i.e., no monthly repeating pattern. Alternative Hypothesis (H_1): The gaps are not uniformly distributed and cluster on specific days (e.g., the 1st), indicating a repeating monthly pattern.

alpha = 0.05

In [22]:
```
date_gaps[['datetime','daily_diff']]
```

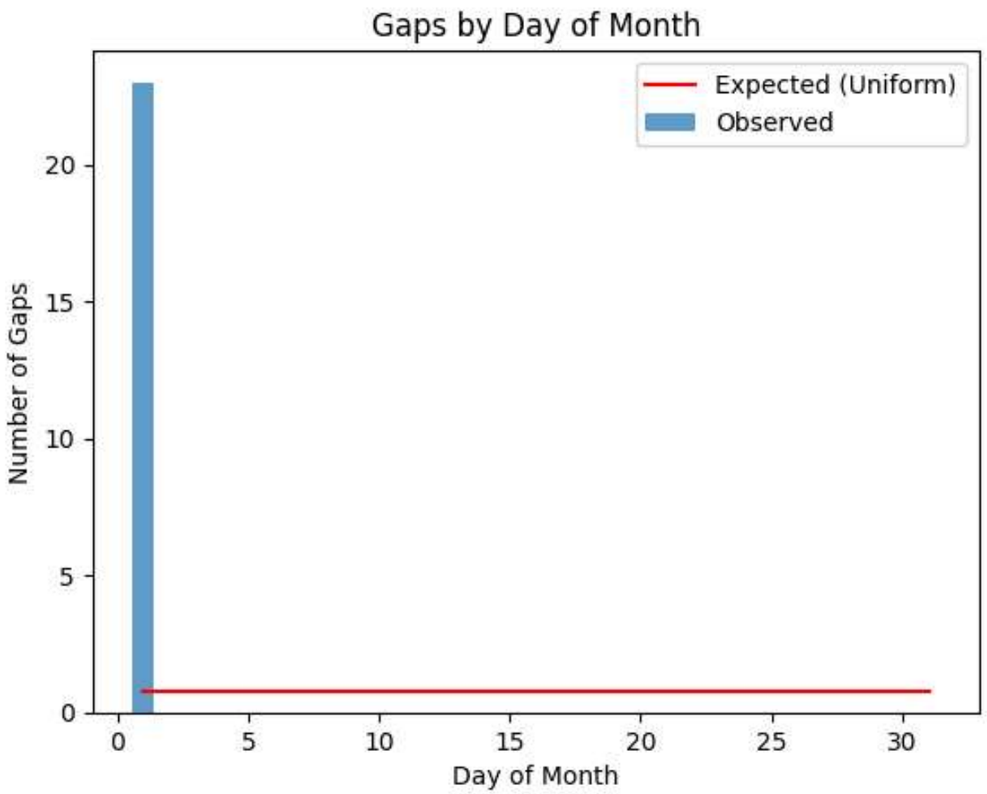
Out[22]:

	datetime	daily_diff
431	2011-02-01	12.041667
877	2011-03-01	9.041667
1323	2011-04-01	12.041667
1778	2011-05-01	11.041667
2234	2011-06-01	12.041667
2690	2011-07-01	11.041667
3146	2011-08-01	12.041667
3602	2011-09-01	12.041667
4055	2011-10-01	11.041667
4510	2011-11-01	12.041667
4966	2011-12-01	11.041667
5422	2012-01-01	12.041667
5875	2012-02-01	12.041667
6330	2012-03-01	10.041667
6785	2012-04-01	12.041667
7239	2012-05-01	11.041667
7695	2012-06-01	12.041667
8151	2012-07-01	11.041667
8607	2012-08-01	12.041667
9063	2012-09-01	12.041667
9519	2012-10-01	11.041667
9975	2012-11-01	12.041667
10430	2012-12-01	11.041667

In [23]:
```
# Extract day of month
date_gaps['day_of_month'] = date_gaps['datetime'].dt.day # we get day 1 for all months because that's where all the missing values are present

# Observed frequencies (all gaps on day 1)
observed = date_gaps['day_of_month'].value_counts().reindex(range(1,32)).fillna(0).astype(int)
observed_counts = observed.values

# Expected frequencies (uniform across 31 days)
total_gaps = len(date_gaps)  # 23
expected_freq = total_gaps / 31  # 23 / 31 ≈ 0.7419
expected = [expected_freq] * 31

# Chi-Square test
chi_stat, p_value = stats.chisquare(observed_counts, f_exp=expected)
```

```
# Results
print(f"Chi-Square Statistic: {chi_stat:.4f}")
print(f"P-Value: {p_value:.4e}")
if p_value > 0.05:
    print("Fail to reject H₀: Gaps are uniformly distributed across days.")
else:
    print("Reject H₀: Gaps cluster on specific days, suggesting a repeating pattern.")

# Plot observed vs expected
plt.bar(range(1, 32), observed_counts, label='Observed', alpha=0.7)
plt.plot(range(1, 32), expected, 'r-', label='Expected (Uniform)')
plt.xlabel('Day of Month')
plt.ylabel('Number of Gaps')
plt.title('Gaps by Day of Month')
plt.legend()
plt.show()
```

```
Chi-Square Statistic: 690.0000
P-Value: 5.9621e-126
Reject H₀: Gaps cluster on specific days, suggesting a repeating pattern.
```

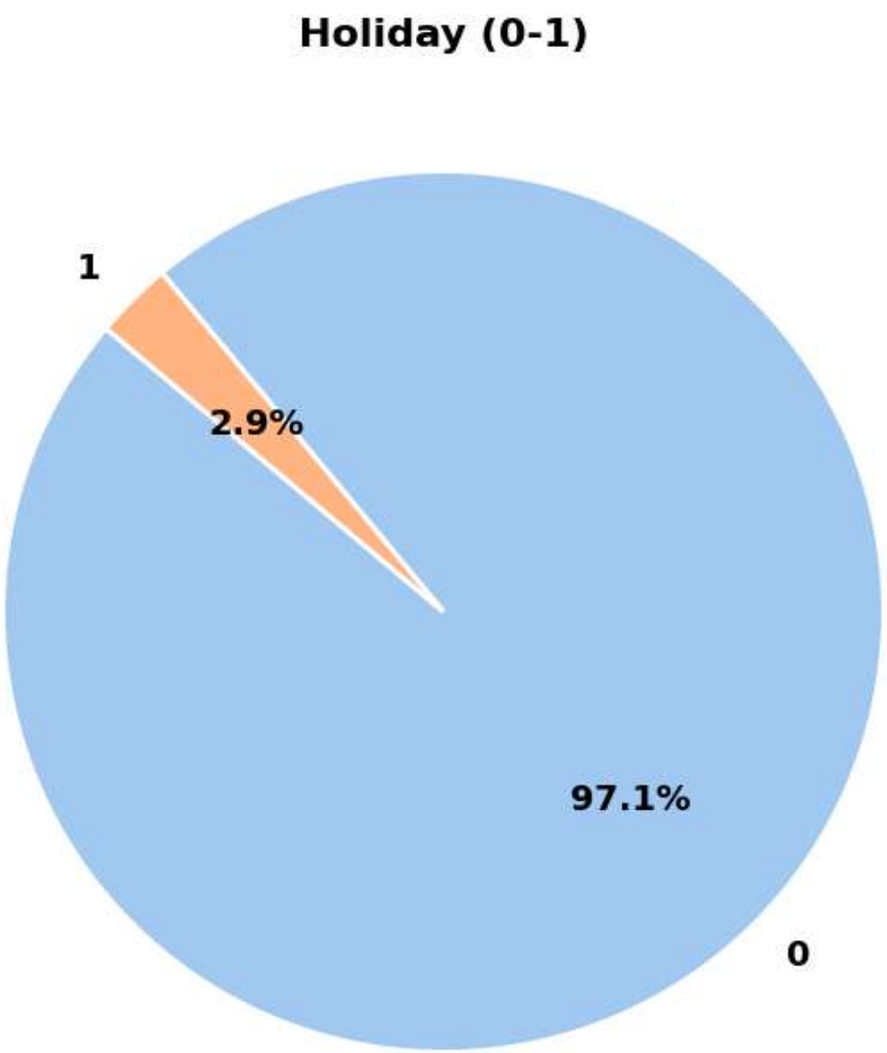


Insights: As we can see p value is very small suggesting a repeating pattern in missing days

In [24]:
```
df.drop(['hourly_diff', 'daily_diff'], axis=1, inplace=True)
df.columns
```

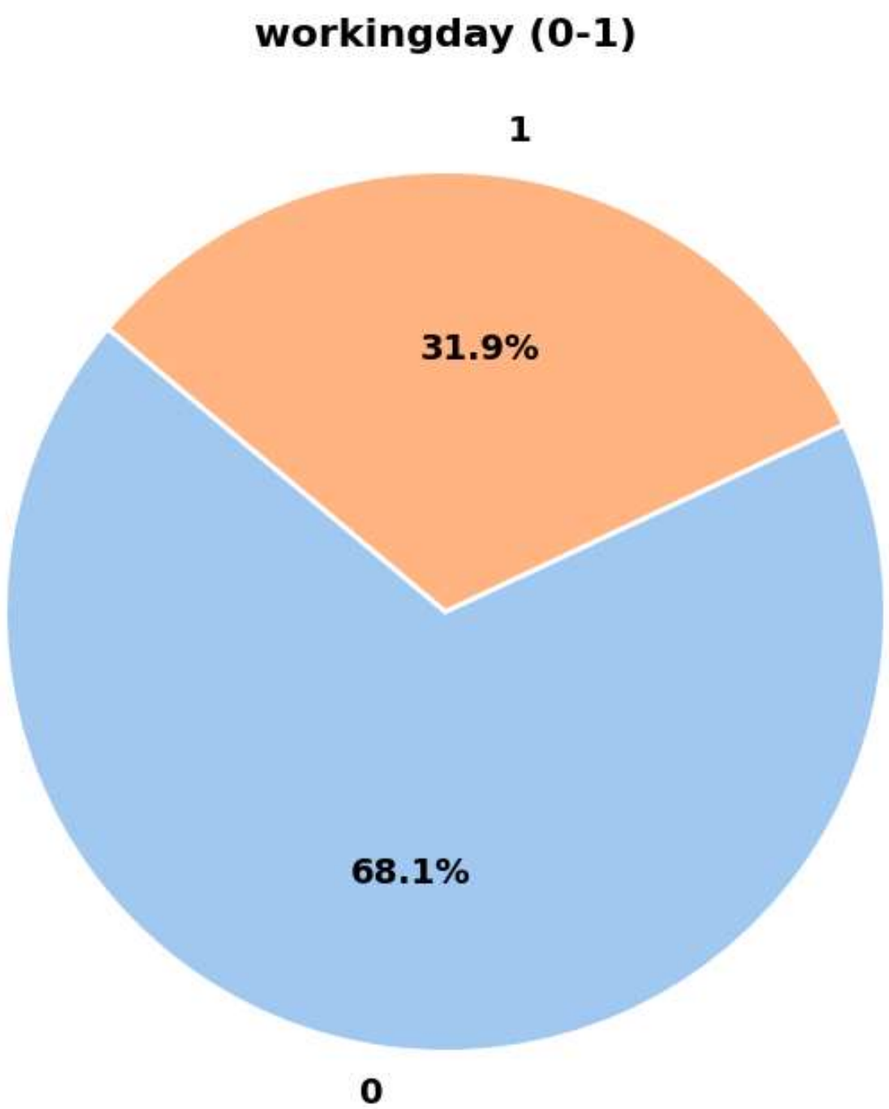
Out[24]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count',
       'year', 'month', 'day_of_week', 'hour'],
      dtype='object')
```

In [25]:
```
plot_pie_chart(df, 'holiday', 'Holiday (0-1)') # 0 - not a holiday, 1 - holiday
```



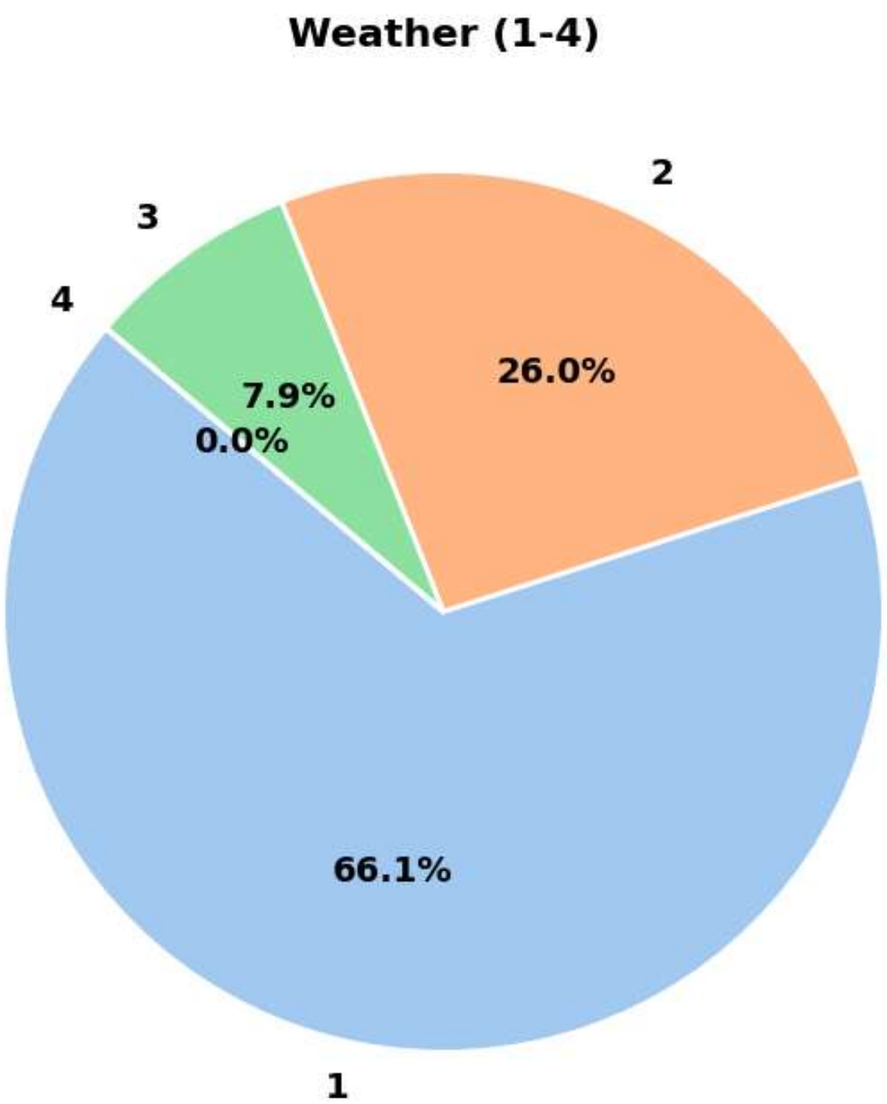
Most of the days are working days - data is heavily right skewed

In [26]:
```
plot_pie_chart(df, 'workingday', 'workingday (0-1)') # 0 - holiday or weekend, 1 - working day
```



From this pie chart, we can tell that working day is midly skewed to the right

In [27]:
```
plot_pie_chart(df, 'weather', 'Weather (1-4)')

# 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
```



In [28]:
```
df['weather'].value_counts()
```

Out[28]:

weather	count
1	7192
2	2834
3	859
4	1

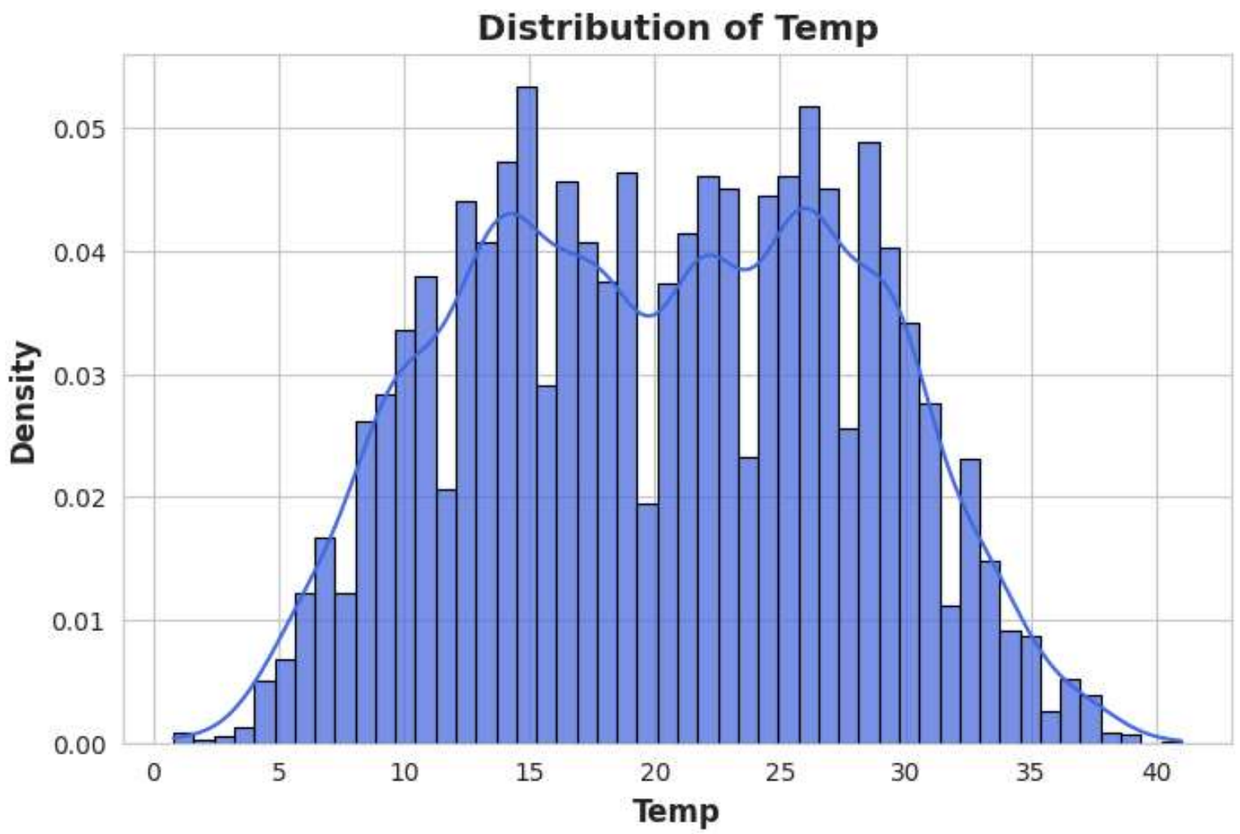
dtype: int64

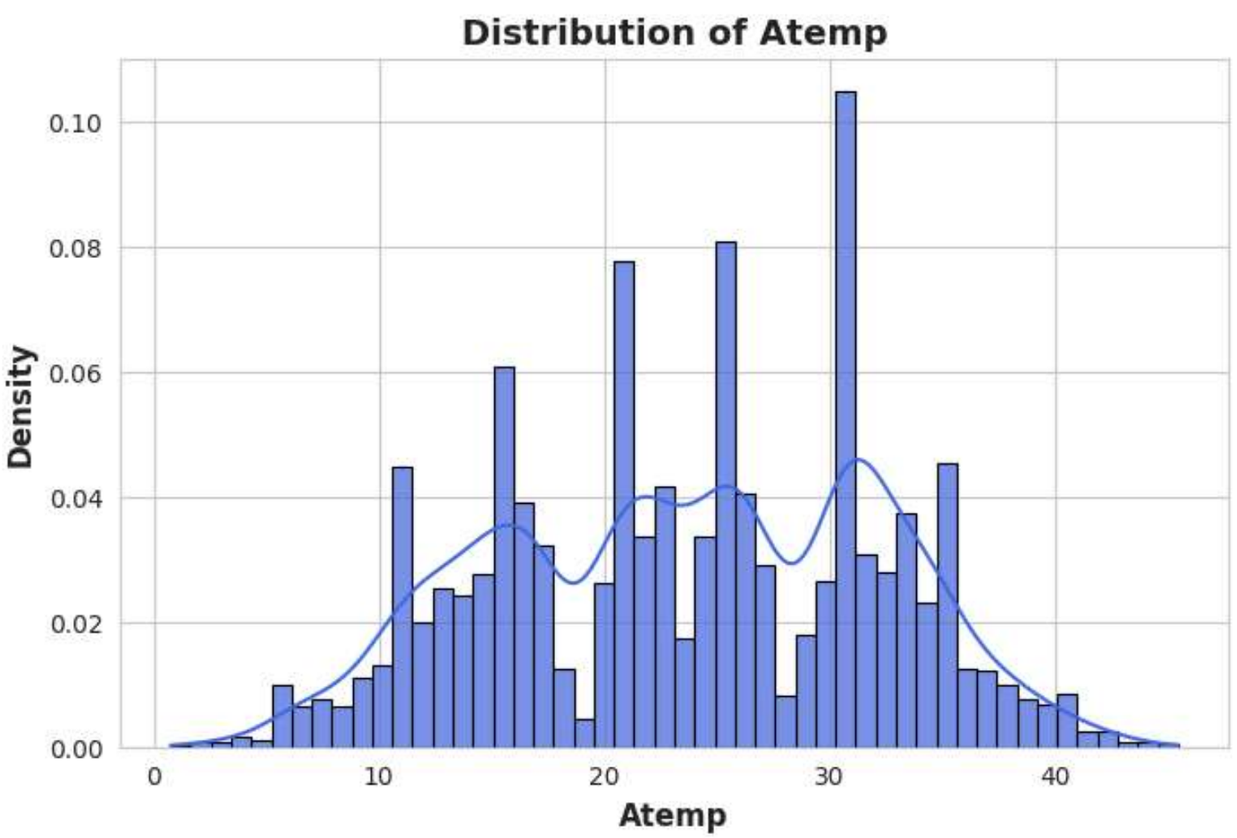
Insights: 4 - Heavyrain is 0% in above graph because it occured only once in 2 years

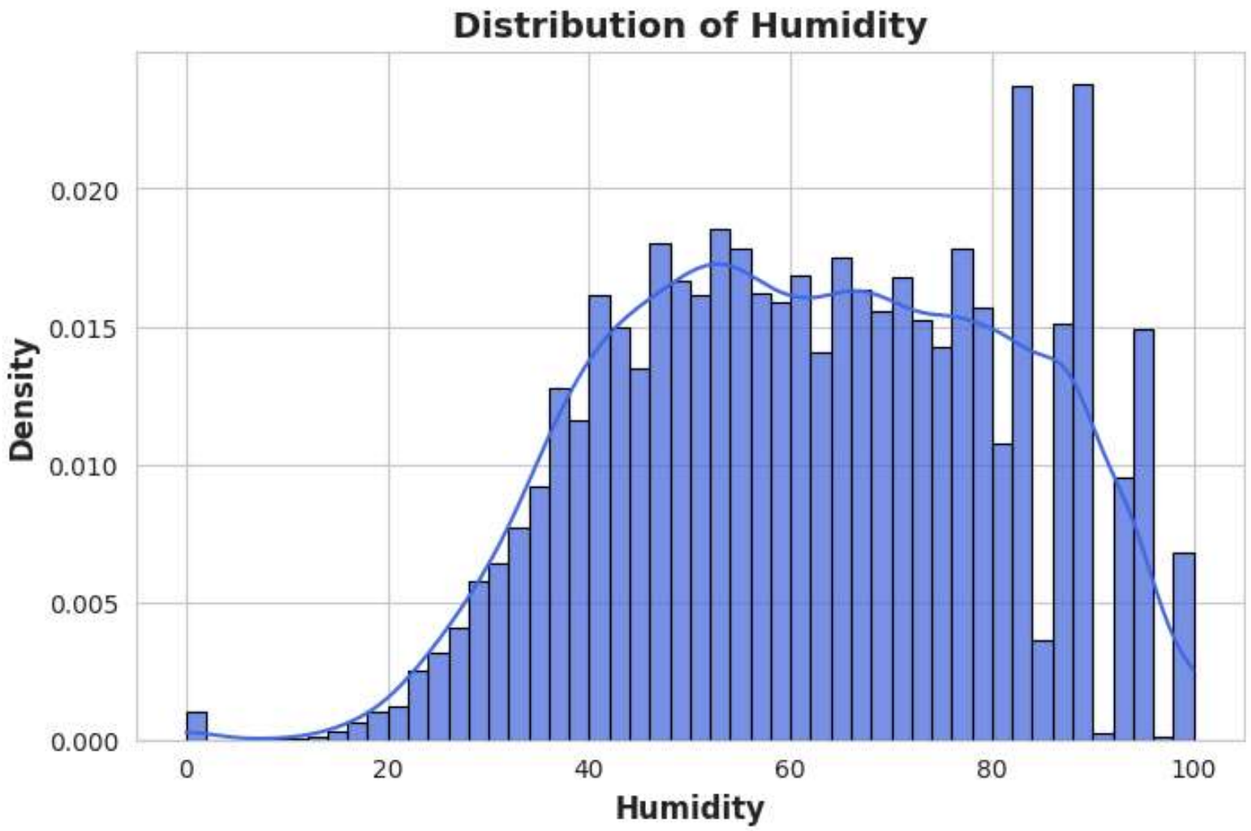
In [29]:
```
numerical_cols = [col for col in numerical_cols if col != 'datetime'] # ommiting datetime col
numerical_cols
```

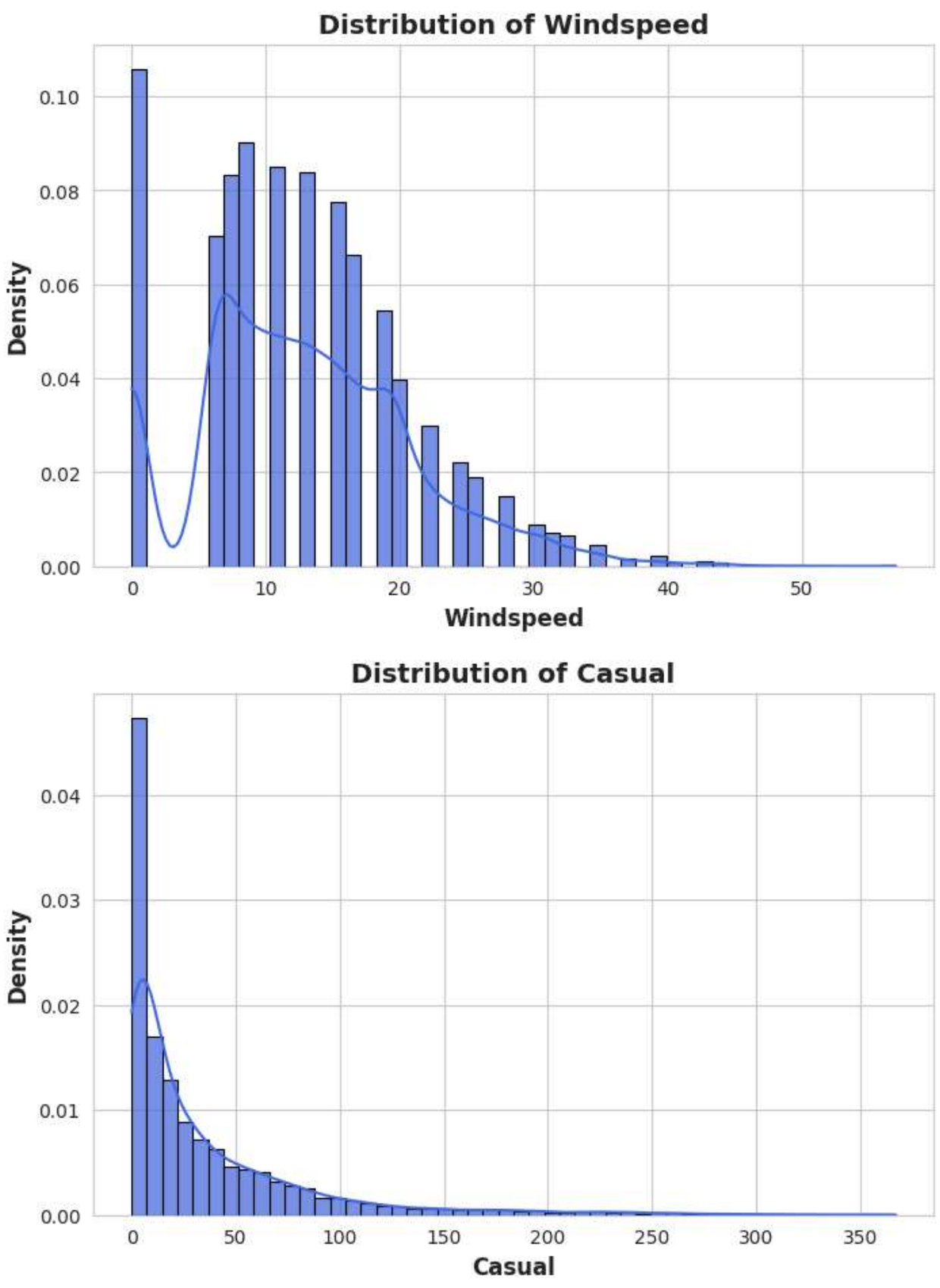
Out[29]:
```
['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
```

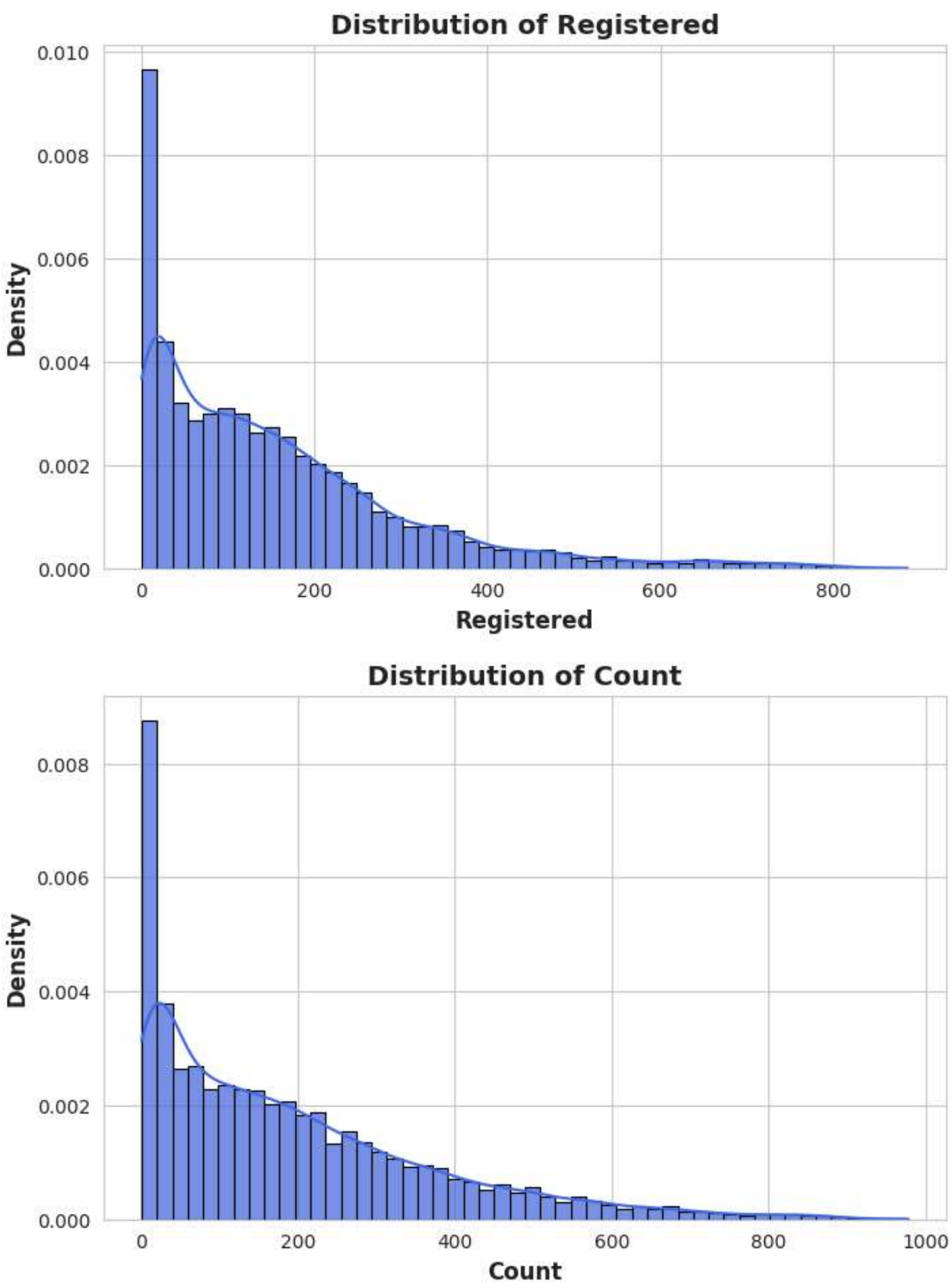
In [30]:
```
def plot_histogram(data, column, bins=50, kde=True, color='royalblue', alpha=0.7):
    # Set style
    sns.set_style("whitegrid")

    # Create figure
    plt.figure(figsize=(8, 5))

    # Plot histogram
    sns.histplot(data[column], bins=bins, kde=kde, color=color, alpha=alpha, edgecolor='black', stat="density")

    # Labels and title
    plt.xlabel(column.replace('_', ' ').title(), fontsize=12, fontweight='bold')
    plt.ylabel("Density", fontsize=12, fontweight='bold')
    plt.title(f"Distribution of {column.replace('_', ' ').title()}", fontsize=14, fontweight='bold')

    # Show plot
    plt.show()

for col in numerical_cols:
    plot_histogram(df, col)
```




Distribution of Atemp
Density
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
Atemp


Distribution of Humidity
Density
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
Humidity


Distribution of Windspeed
Density
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
Windspeed
Distribution of Casual
Density
0.04
0.03
0.02
0.01
0.00
0
50
100
150
200
250
300
350
Casual


Distribution of Registered
Density
0.010
0.008
0.006
0.004
0.002
0.000
0
200
400
600
800
Registered
Distribution of Count
Density
0.008
0.006
0.004
0.002
0.000
0
200
400
600
800
1000
Count

Insights:

1. Windspeed: Right-skewed, peaking at 5–10 units, with rare extreme values (~50). Most days have low windspeed (0–20), suggesting calm conditions. High windspeed could deter ridership, making extreme values worth analyzing.
2. Humidity: Right-skewed, mostly between 40–90%, with a noticeable spike at 100% (possibly rainy or foggy conditions). Low humidity (<20%) is rare. High humidity may negatively impact bike usage.
3. Temperature (Temp): Bimodal, with peaks at 10–15°C and 25–30°C, likely reflecting seasonal variations. Extreme values (close to 0°C or 40°C) are rare, indicating a temperate climate. Warmer temperatures may encourage casual ridership.
4. Apparent Temperature (Atemp): Similar to temp but with a slightly shifted second peak (30–35°C). Suggests humidity and windspeed don't drastically alter perceived temperature, though a spike at ~40°C indicates some extreme heat days.
5. Casual Users: Strongly right-skewed, mostly between 0–50, with rare peaks (~350) on weekends, holidays, or special events. Casual ridership is highly variable and influenced by external factors like weather and day type.
6. Registered Users: Right-skewed, typically 0–200, but with a higher maximum (~800) compared to casual users. Usage is more consistent, likely due to commuting patterns, though extreme weather might reduce counts.
7. Total Users (Count): Right-skewed, peaking near 0 but extending to ~1000. Reflects the combined behavior of casual and registered users. Registered users dominate overall usage, but total ridership spikes occur when both groups are high, particularly on warm, non-working days.

Bi-variate and hypothesis testsing

1. Working Day has effect on number of electric cycles rented - compare if means of count is same when it was a working day and when not

H0: Working day has no effect on number of cycles rented Ha: Working day has significant effect on number of cycles rented

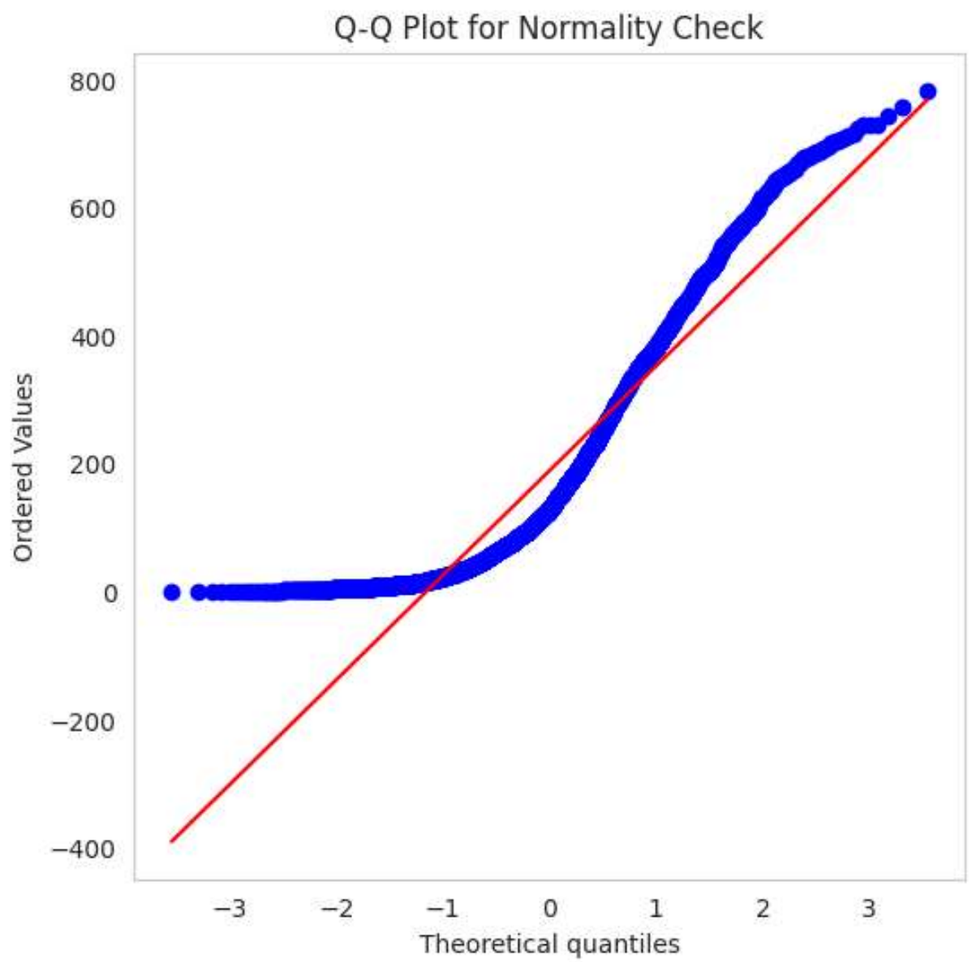
confidence level = 95% significance level = 5%

In [31]:
```
def normality_test(feature, significance_level=0.01):
    print("#"*100)
    print("\n")
    # Choosing the appropriate normality test
    if len(feature) <= 30:
        test_stat, p_value = stats.shapiro(feature)
        test = "Shapiro-Wilk Test"
    else:
        test_stat, p_value = stats.kstest(feature, 'norm', args=(np.mean(feature), np.std(feature, ddof=1)))
        test = "Kolmogorov-Smirnov Test"

     # Q-Q Plot
    plt.figure(figsize=(6, 6))
    stats.probplot(feature, dist="norm", plot=plt)
    plt.title("Q-Q Plot for Normality Check")
    plt.grid()
    plt.show()

    # Print test results
    print("\n")
    print("*"*50)
    print(f"{'Normality Test:':<20} {test:<10}")
    print(f"{'Test Statistic:':<20} {test_stat:<10.4e}")
```

```
    print(f"{'P-value:':<20} {p_value:<10.4e}")

    # Interpretation
    if p_value < significance_level:
        print("\nReject H₀: Data is **not normally distributed**.")
        is_normal = False
    else:
        print("\nFail to reject H₀: Data appears **normally distributed**.")
        is_normal = True
    print("*"*50)
    print("\n")
    return is_normal
```

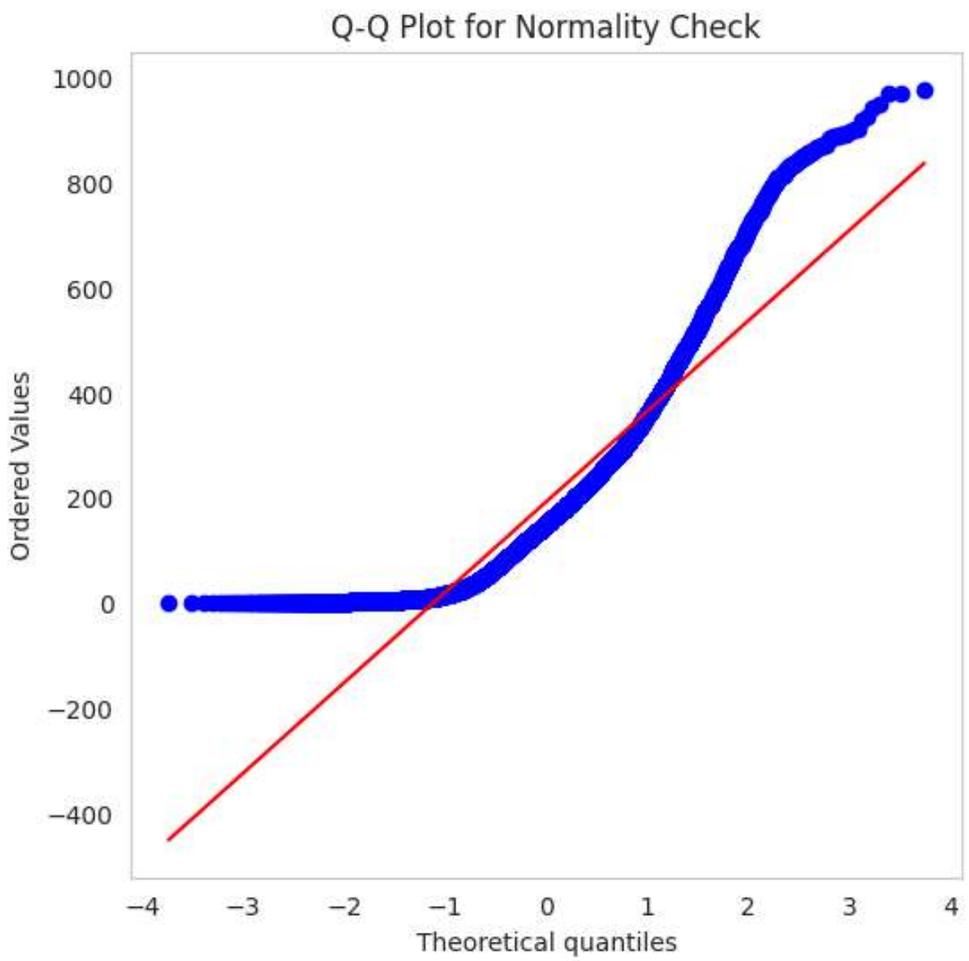
In [32]:
```
is_normal1 = normality_test(df[df['workingday'] == 0]['count'])
is_normal2 = normality_test(df[df['workingday'] == 1]['count'])

is_normal = is_normal1 and is_normal2
```

##



```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.4022e-01
P-value:             4.7245e-60

Reject H₀: Data is **not normally distributed**.
************************************************
```

```
#########################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.4902e-01
P-value:             3.7697e-144

Reject H₀: Data is **not normally distributed**.
************************************************
```

In [33]:
```python
def variance_test(feature, target_feature, is_normal=True, significance_level = 0.01):
  groups = [group for _, group in df.groupby(feature)[target_feature]]
```

```
  if not is_normal:
    test_stat, p_value = stats.levene(*groups)
    test = "Levene's Test"
  else:
    test_stat, p_value = stats.bartlett(*groups)
    test = "Bartlett's Test"

  print("\n")
  print("*"*50)
  print(f"{'Variance Test:':<20} {test:<10}")
  print(f"{'Test Statistic:':<20} {test_stat:<10.4e}")
  print(f"{'P-value:':<20} {p_value:<10.4e}")

  if p_value < significance_level:
    print("\nReject H₀: Variances are not equal.")
  else:
    print("\nFail to reject H₀: Variances are equal.")

  print("*"*50)
  print("\n")
```

In [34]:
```
variance_test('workingday', 'count', is_normal)
```

```
**************************************************
Variance Test:       Levene's Test
Test Statistic:      4.9728e-03
P-value:             9.4378e-01

Fail to reject H₀: Variances are equal.
**************************************************
```

Normality failed but Variance assumption is passed. According to the question, we can do 2 sample ttest to check if means of both independent groups are same

In [35]:
```
test_stats, p_value = stats.ttest_ind(df[df['workingday'] == 0]['count'], df[df['workingday'] == 1]['count'])
print("\n")
print("*"*100)
print(f"{'Independent Samples T-Test:':^100}")
print(f"{'Test Statistic':<20} {test_stats:<10.4e}")
print(f"{'P-value':<20} {p_value:<10.4e}")

if p_value < 0.05:
    print("\nReject H₀: There is a significant difference in means between the groups.")
else:
    print("\nFail to reject H₀: There is no significant difference in means between the groups.")
print("*"*100)
```

```
****************************************************************************************************
                                 Independent Samples T-Test:
Test Statistic       -1.2096e+00
P-value              2.2645e-01

Fail to reject H₀: There is no significant difference in means between the groups.
****************************************************************************************************
```

Insights:

Balanced usage: People might rent cycles for both commuting (on working days) and leisure (on non-working days), leading to no strong difference.

Other stronger influences: Factors like weather, holidays, or time of year might play a bigger role in rental patterns than simply whether a day is classified as a "working day."

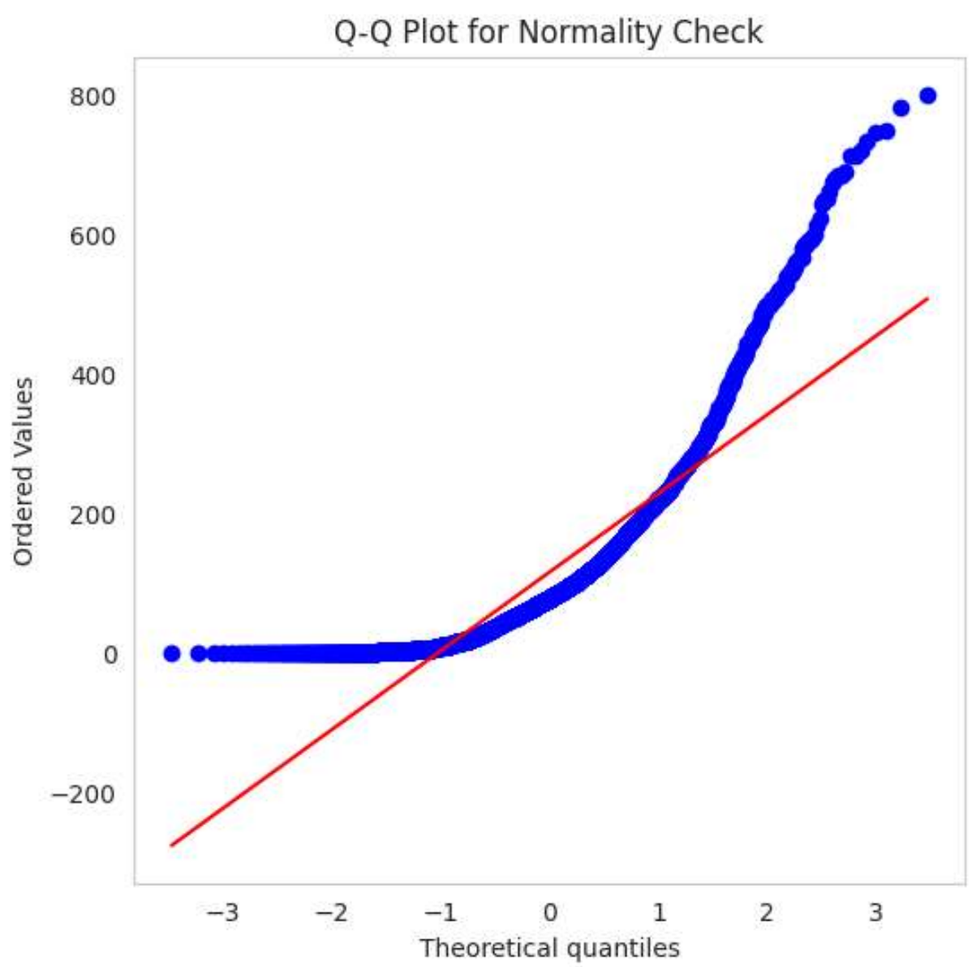
2. No. of cycles rented similar or different in different seasons

Null Hypothesis (H_0): The number of cycles rented does not significantly differ across seasons.

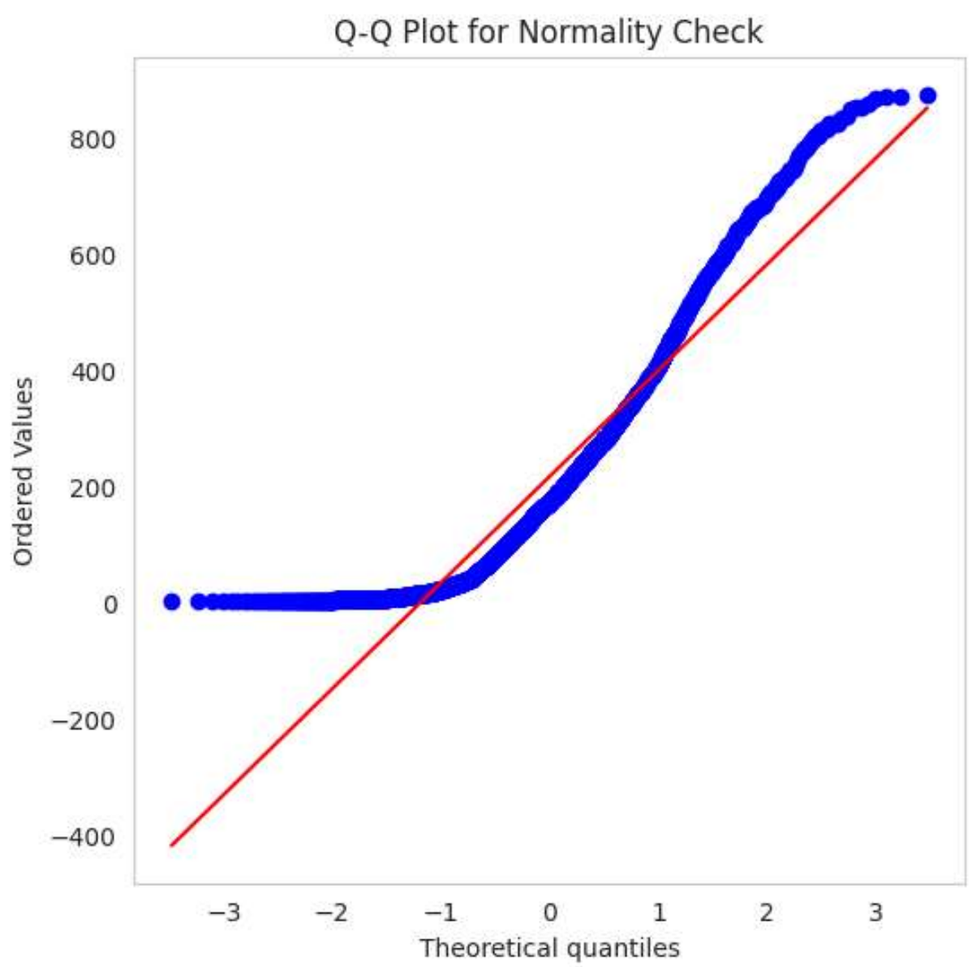
Alternative Hypothesis (H_1): The number of cycles rented significantly differs across seasons.

In [36]:
```
is_normal = True

for season in df['season'].unique():
    print(f"Season {season}")
    normal = normality_test(df[df['season'] == season]['count'])
    if not normal:
        is_normal = False

variance_test('season','count', is_normal)
```

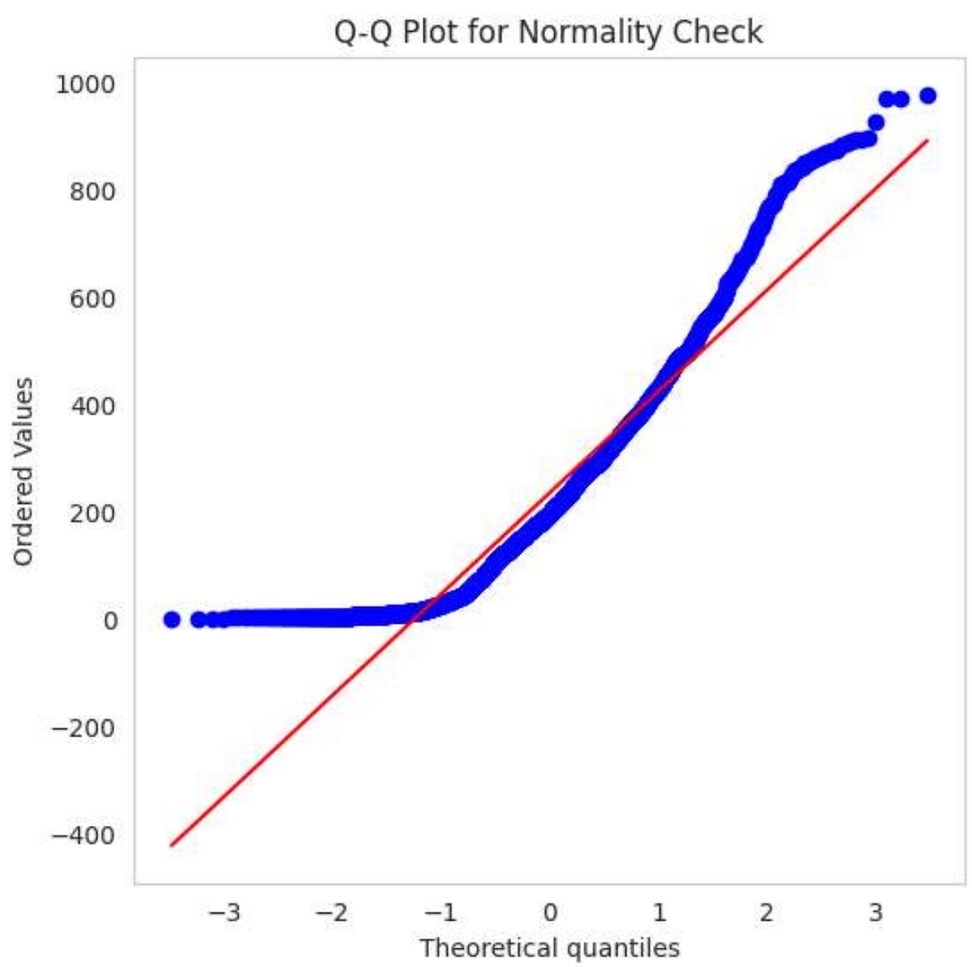
```
Season 1
#############################################################################################################
```



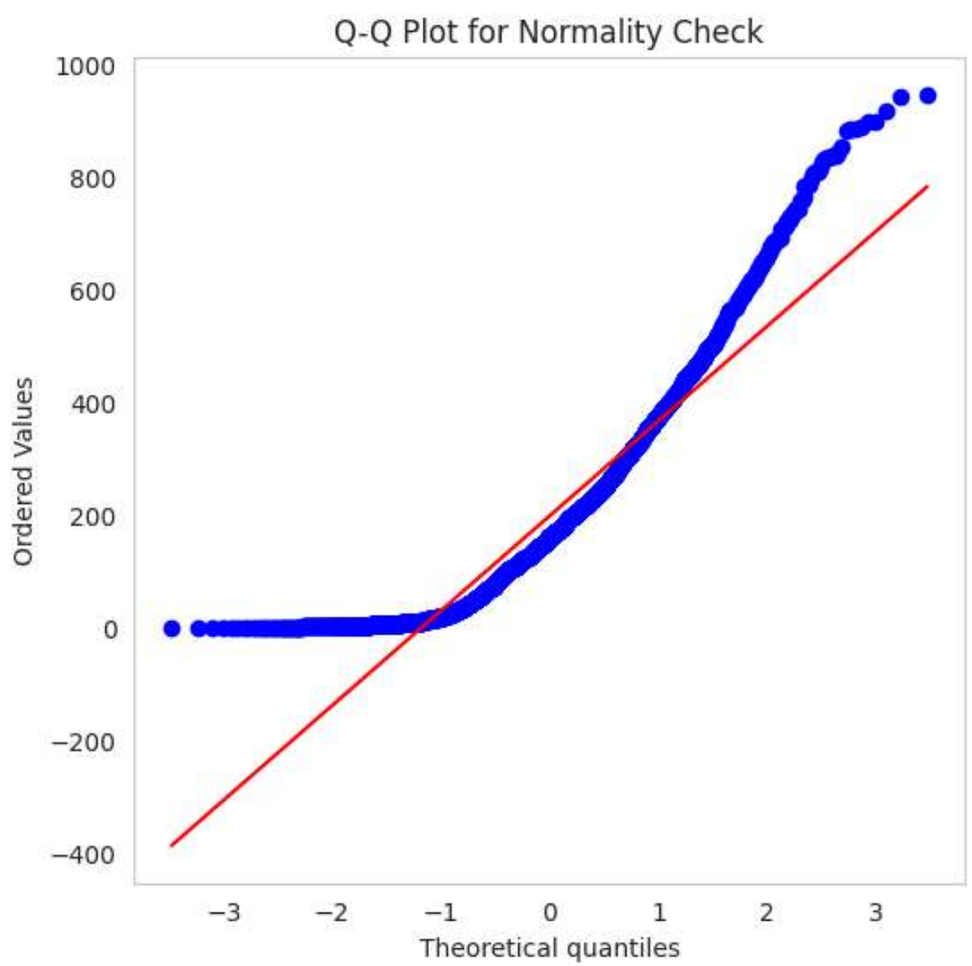

```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.7860e-01
P-value:             2.0076e-75

Reject H₀: Data is **not normally distributed**.
************************************************


Season 2
#########################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.3224e-01
P-value:             3.8735e-42

Reject H₀: Data is **not normally distributed**.
************************************************


Season 3
#########################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.1822e-01
P-value:             9.7900e-34

Reject H₀: Data is **not normally distributed**.
************************************************


Season 4
#############################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.3250e-01
P-value:             2.5790e-42

Reject H₀: Data is **not normally distributed**.
************************************************
```

```
************************************************
Variance Test:       Levene's Test
Test Statistic:      1.8777e+02
P-value:             1.0147e-118

Reject H₀: Variances are not equal.
************************************************
```

Both normality and variance assumptions failed. Even so, lets use ANOVA as mentioned in question

In [37]:
```
test_stat, p_value = stats.f_oneway(*[group['count'] for _, group in df.groupby('season')])
print("\n")
print("*"*100)
```

```
print(f"{'ANOVA Test:':^100}")
print(f"{'Test Statistic':<20} {test_stat:<10.4e}")
print(f"{'P-value':<20} {p_value:<10.4e}")

if p_value < 0.05:
    print("\nReject H₀: The number of cycles rented significantly differs across seasons.")
else:
    print("\nFail to reject H₀: The number of cycles rented does not significantly differ across seasons.")
print("*"*100)
print("\n")
```

```
****************************************************************************************************
                                        ANOVA Test:
Test Statistic       2.3695e+02
P-value              6.1648e-149

Reject H₀: The number of cycles rented significantly differs across seasons.
****************************************************************************************************
```

Insights:

Seasonal Variation: The significant result from the ANOVA test indicates that the number of cycles rented varies across different seasons, suggesting that external factors like weather, holidays, or events could influence bike rentals.

Data Implications: Businesses or city planners can use this insight to optimize cycle availability and marketing efforts, tailoring strategies to each season's demand patterns.

Actionable Strategy: Prepare for higher rentals during peak seasons and plan for lower demand periods by adjusting inventory and pricing strategies accordingly.

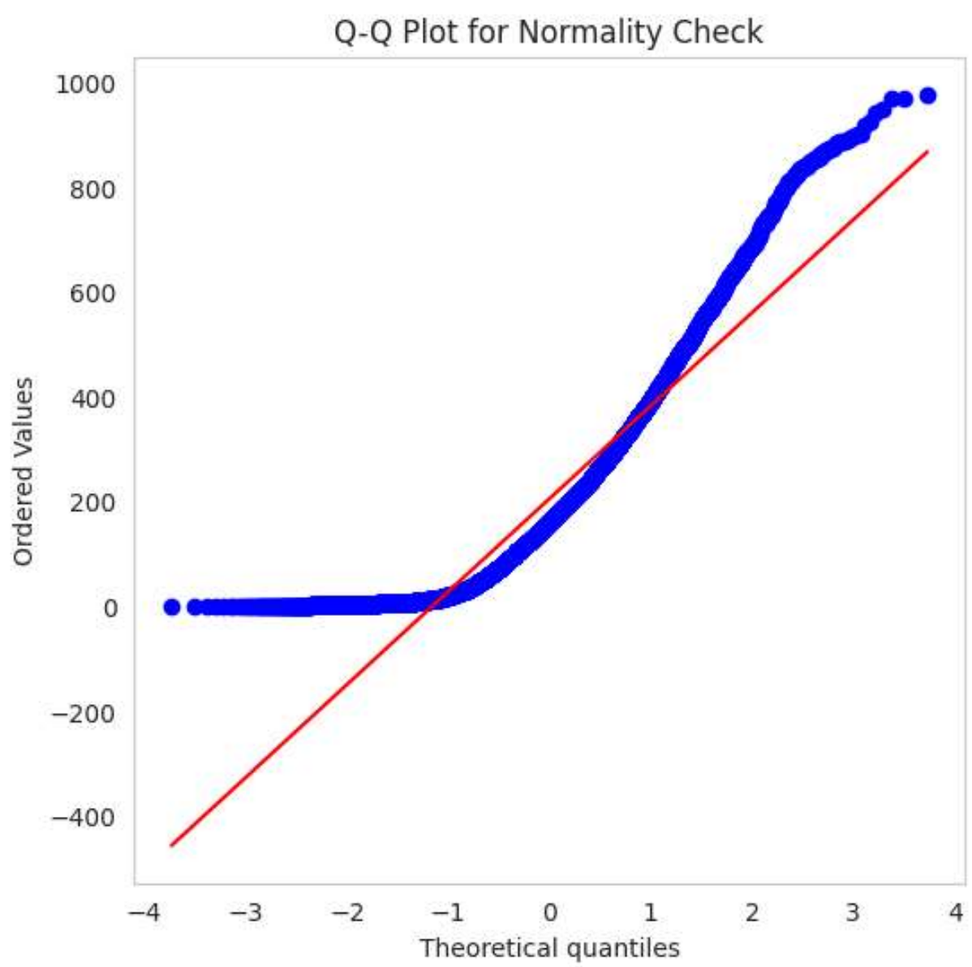
3. No. of cycles rented similar or different in different weather

Null Hypothesis (H_0): The number of cycles rented does not significantly differ across different weathers.

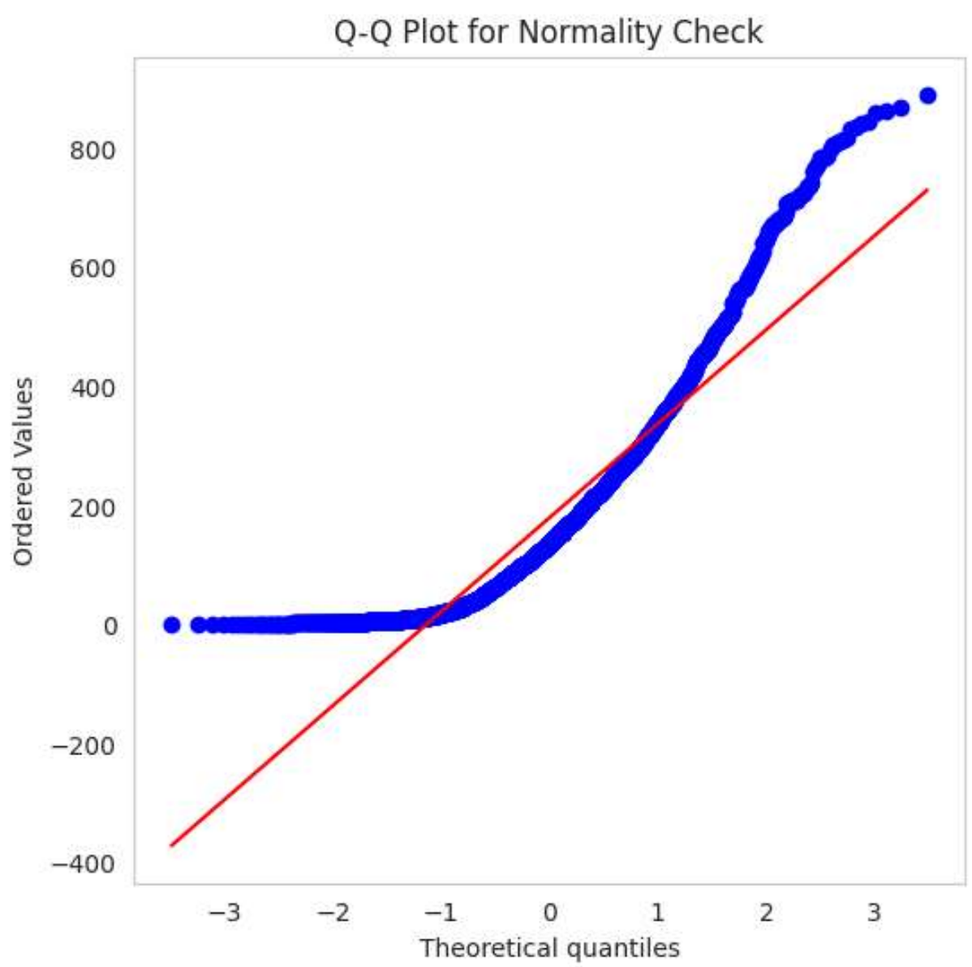
Alternative Hypothesis (H_1): The number of cycles rented significantly differs across different weathers.

In [38]:
```
is_normal = True

for season in df['weather'].unique():
    print(f"Weather {season}")
    normal = normality_test(df[df['weather'] == season]['count'])
    if not normal:
        is_normal = False

variance_test('weather','count', is_normal)
```

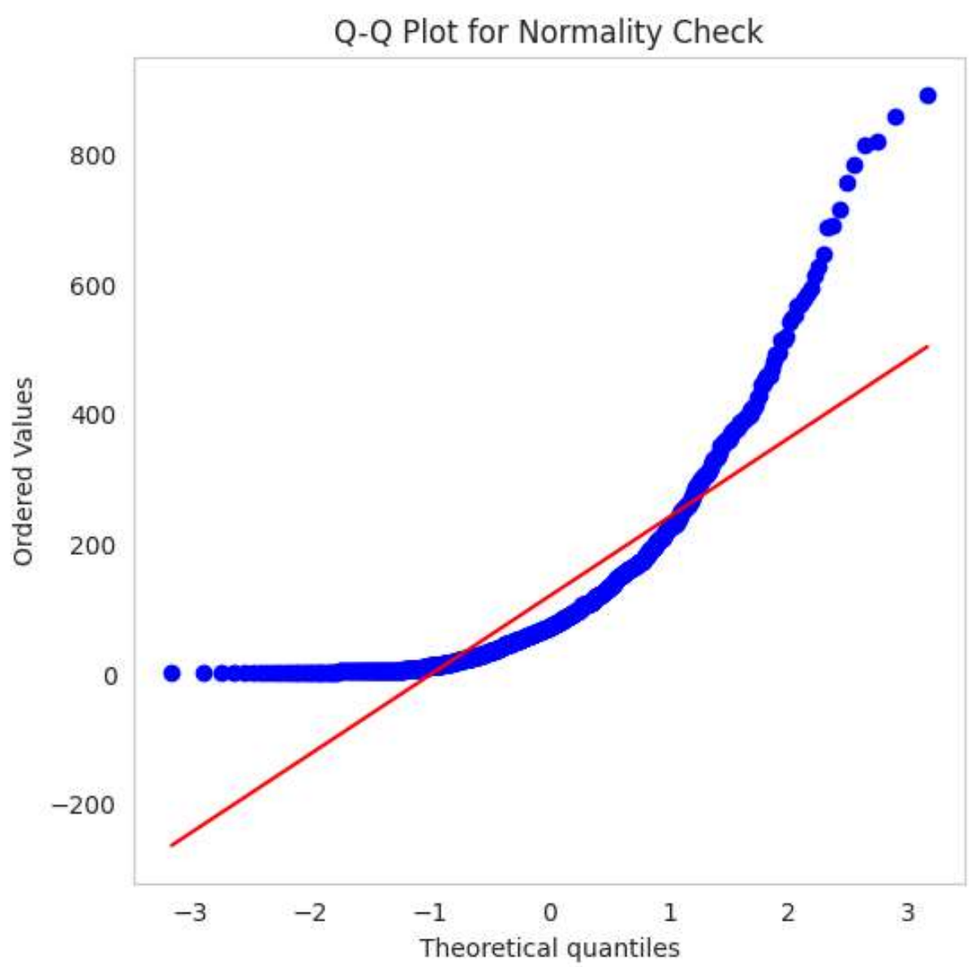
```
Weather 1
############################################################################################################
```



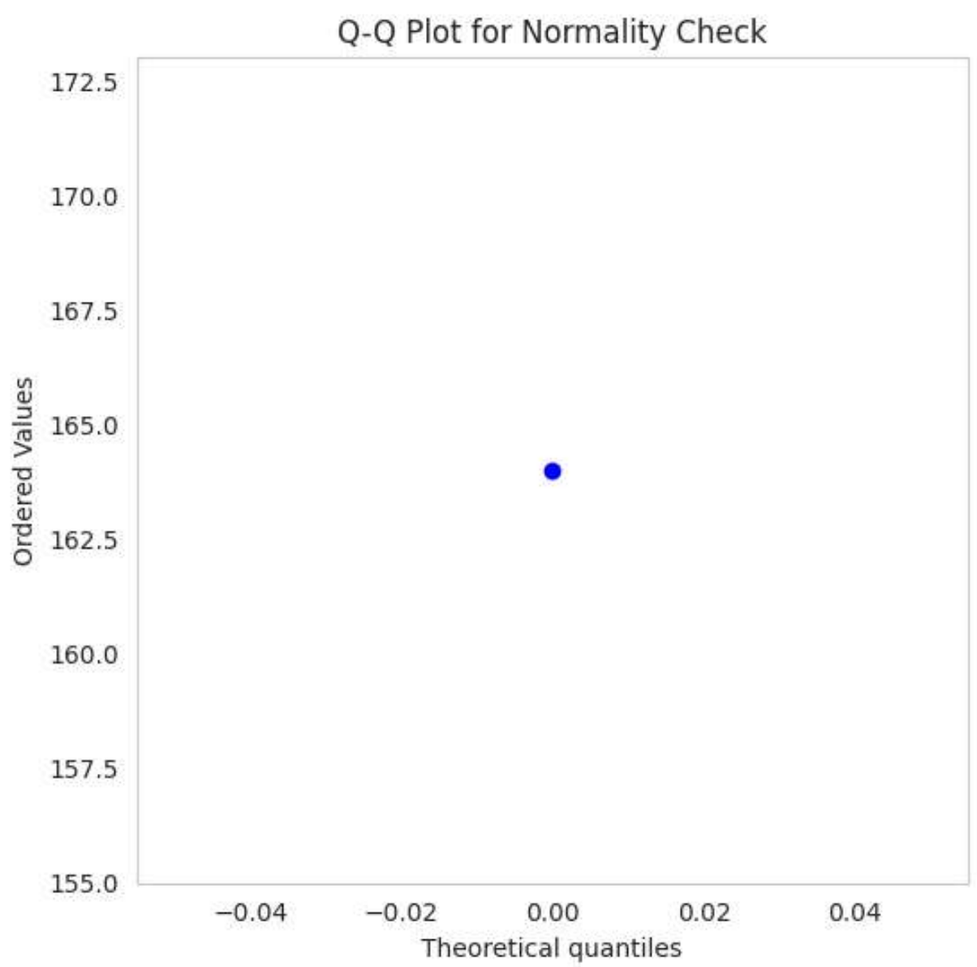

```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.3861e-01
P-value:             5.3978e-121

Reject H₀: Data is **not normally distributed**.
************************************************


Weather 2
#########################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.4527e-01
P-value:             1.1787e-52

Reject H₀: Data is **not normally distributed**.
************************************************


Weather 3
#########################################################################################################
```




```
************************************************
Normality Test:      Kolmogorov-Smirnov Test
Test Statistic:      1.9756e-01
P-value:             7.4572e-30

Reject H₀: Data is **not normally distributed**.
************************************************


Weather 4
#########################################################################################################
```




```
**************************************************
Normality Test:      Shapiro-Wilk Test
Test Statistic:      nan
P-value:             nan

Fail to reject H₀: Data appears **normally distributed**.
**************************************************



**************************************************
Variance Test:       Levene's Test
Test Statistic:      5.4851e+01
P-value:             3.5049e-35

Reject H₀: Variances are not equal.
**************************************************
```

Both normality and variance assumptions failed. Even so, lets use ANOVA as mentioned in question

In [39]:
```
test_stat, p_value = stats.f_oneway(*[group['count'] for _, group in df.groupby('weather')])
print("\n")
print("*"*100)
```

```
print(f"{'ANOVA Test:':^100}")
print(f"{'Test Statistic':<20} {test_stat:<10.4e}")
print(f"{'P-value':<20} {p_value:<10.4e}")

if p_value < 0.05:
    print("\nReject H₀: The number of cycles rented significantly differs across weathers.")
else:
    print("\nFail to reject H₀: The number of cycles rented does not significantly differ across weathers.")
print("*"*100)
print("\n")
```

```
****************************************************************************************************
                                        ANOVA Test:
Test Statistic       6.5530e+01
P-value              5.4821e-42

Reject H₀: The number of cycles rented significantly differs across weathers.
****************************************************************************************************
```

ANOVA Test Insights

- **Test Statistic**: The test statistic value is **65.53**, which is relatively large, indicating that there is a significant difference in the number of cycles rented across different weather conditions.
- **P-value**: The p-value is **5.48×10^{-42}**, which is extremely small. This suggests that the probability of observing such a difference if the null hypothesis were true is virtually zero. Hence, it provides very strong evidence against the null hypothesis.
- **Conclusion**: Since the p-value is far below the typical significance threshold (e.g., **$\alpha = 0.05$**), we **reject the null hypothesis**. This means that the number of cycles rented **significantly differs** across different weather conditions.

Key Insights

- Weather conditions have a **notable impact** on cycle rentals.

4. Weather is dependent on season (check between 2 predictor variable)

Null Hypothesis (H_0): Weather is independent of the season (no significant association).

Alternative Hypothesis (H_1): Weather is dependent on the season (there is a significant association).

In [40]:
```
# Since both variables are categorical, we use chi-squared independence test

contingency_table = pd.crosstab(df['weather'], df['season'])
display(contingency_table)

print("\n\n")
print("*"*50)
print(f"{'Chi-Square Test for Independence':^100}")
chi2, p_value, dof, expected = stats.chi2_contingency(contingency_table)
print(f"{'Test Statistic':<20} {chi2:<10.4e}")
print(f"{'P-value':<20} {p_value:<10.4e}")
print(f"{'Degrees of Freedom':<20} {dof:<10}")

if p_value < 0.05:
```

```
    print("\nReject H₀: Weather is dependent on the season.")
else:
    print("\nFail to reject H₀: Weather is independent of the season.")
print("*"*50)
```

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

```
**************************************************
                                Chi-Square Test for Independence
Test Statistic       4.9159e+01
P-value              1.5499e-07
Degrees of Freedom   9

Reject H₀: Weather is dependent on the season.
**************************************************
```

In [41]:
```
# Compute standardized residuals
observed = contingency_table.values  # Actual counts
standardized_residuals = (observed - expected) / np.sqrt(expected)

# Convert to DataFrame for better readability
residuals_df = pd.DataFrame(standardized_residuals, index=contingency_table.index, columns=contingency_table.columns)

# Display standardized residuals
display(residuals_df)
```

season	1	2	3	4
weather				
1	-0.369051	-0.108200	2.927643	-2.453131
2	0.595302	-0.130984	-4.029935	3.570104
3	-0.065168	0.568089	-1.134296	0.630696
4	1.516445	-0.501055	-0.501055	-0.501147

Fall has significantly more clear weather than expected.

Likely due to stable atmospheric conditions, making it a preferred season for outdoor activities.

Winter has significantly fewer clear weather days, probably due to increased cloud cover.

Mist and Cloudy Conditions are strongly seasonal.

Winter has a much higher occurrence of mist than expected.

Fall has significantly fewer misty/cloudy days.

Cold air in Winter retains moisture, leading to more fog and mist.

Heavy Rain and Ice Pellets are slightly overrepresented in Spring.

This suggests spring storms are a notable weather pattern.

Insights

Since the p-value is very small, we reject the null hypothesis (H_0) and conclude that weather is significantly dependent on the season.

Key Insights: Seasonal Variation in Weather: Certain weather conditions occur more frequently in specific seasons (e.g., more rain in monsoon, more clear skies in summer).

Practical Implications:

Businesses relying on weather forecasts (e.g., tourism, outdoor rentals) can plan accordingly.

City planning and transportation services can optimize resource allocation based on expected seasonal weather patterns.

Collated Insights and Recommendations

Bussiness Insights

Bivariate Analysis & Hypothesis Testing Insights

1. **Working Day vs. Cycle Rentals**
   - There is **no significant difference** in **average cycle rentals** between working and non-working days.
   - Other factors likely have a **greater influence** on cycle rentals than working status.
2. **Season vs. Cycle Rentals**
   - There is a **significant difference** in **cycle rentals across different seasons.**
3. **Weather vs. Cycle Rentals**
   - There is a **significant difference** in **cycle rentals across different weather conditions.**
4. **Weather vs. Season**
   - Weather patterns are **dependent on the season.**
   - **Fall has more clear days**, **winter has more misty/cloudy days**, and **spring shows slightly more heavy rain** than expected.

Recommendations

1. **Working Day vs. Cycle Rentals**
   - Focus on other factors such as **weather**, **seasonality**, and **holidays**, which could better predict **cycle rental demand**. These factors appear to have a stronger influence on rentals than working days.
2. **Season vs. Cycle Rentals**

- **Adjust inventory, maintenance**, and **marketing strategies** based on the demand trends across different seasons to improve operational efficiency and customer satisfaction.

3. **Weather vs. Cycle Rentals**

   - **Monitor weather forecasts** closely to anticipate **demand fluctuations** based on weather conditions.
   - Consider implementing **dynamic pricing** or targeted **promotions** to optimize rentals during weather changes.

4. **Weather vs. Season**

- It is crucial to account for **seasonal weather patterns** when forecasting demand, as these patterns have a direct impact on the availability of favorable conditions for cycling.
- During **winter**, when **mist and cloudy weather** dominate, it might be helpful to **limit rental availability** or plan for a **reduction in fleet size**, especially if these conditions deter customers from renting bikes.
- In **spring and fall**, focus on preparing for potential **rainy weather** by offering **weather-resistant options** and ensuring that bikes are equipped with **rain gear**.
- Additionally, **season-specific marketing strategies** should be developed, where campaigns are tailored to weather patterns, like promoting cycling in clear weather and offering incentives during less favorable weather to boost rentals.

In [42]:
```
## END
```