untitled83

November 26, 2024

1 Yulu - Hypothesis Testing

Q1. Define the Problem Statement, Import the required Libraries and perform Exploratory Data Analysis.

a. Examine dataset structure, characteristics, and statistical summary.

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

[2]:
```
df = pd.read_csv('bike_sharing.csv')
df
```

[2]:
```
                  datetime  season  holiday  workingday  weather   temp  \
0      2011-01-01 00:00:00       1        0           0        1   9.84
1      2011-01-01 01:00:00       1        0           0        1   9.02
2      2011-01-01 02:00:00       1        0           0        1   9.02
3      2011-01-01 03:00:00       1        0           0        1   9.84
4      2011-01-01 04:00:00       1        0           0        1   9.84
…                        …      …        …           …        …     …
10881  2012-12-19 19:00:00       4        0           1        1  15.58
10882  2012-12-19 20:00:00       4        0           1        1  14.76
10883  2012-12-19 21:00:00       4        0           1        1  13.94
10884  2012-12-19 22:00:00       4        0           1        1  13.94
10885  2012-12-19 23:00:00       4        0           1        1  13.12

        atemp  humidity  windspeed  casual  registered  count
0      14.395        81     0.0000       3          13     16
1      13.635        80     0.0000       8          32     40
2      13.635        80     0.0000       5          27     32
3      14.395        75     0.0000       3          10     13
4      14.395        75     0.0000       0           1      1
…           …         …          …       …           …
10881  19.695        50    26.0027       7         329    336
10882  17.425        57    15.0013      10         231    241
10883  15.910        61    15.0013       4         164    168
```

```
10884  17.425         61     6.0032       12          117    129
10885  16.665         66     8.9981        4           84     88

[10886 rows x 12 columns]
```

[3]:
```
data_shape = df.shape
data_info = df.info()
data_head = df.head()
data_summary = df.describe()

data_shape, data_info, data_head, data_summary
```

```
<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
```

[3]:
```
((10886, 12),
 None,
               datetime  season  holiday  workingday  weather  temp   atemp  \
 0  2011-01-01 00:00:00       1        0           0        1  9.84  14.395
 1  2011-01-01 01:00:00       1        0           0        1  9.02  13.635
 2  2011-01-01 02:00:00       1        0           0        1  9.02  13.635
 3  2011-01-01 03:00:00       1        0           0        1  9.84  14.395
 4  2011-01-01 04:00:00       1        0           0        1  9.84  14.395

    humidity  windspeed  casual  registered  count
 0        81        0.0       3          13     16
 1        80        0.0       8          32     40
 2        80        0.0       5          27     32
 3        75        0.0       3          10     13
 4        75        0.0       0           1      1  ,
```

```
             season       holiday    workingday       weather         temp  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.00000
mean       2.506614      0.028569      0.680875      1.418427     20.23086
std        1.116174      0.166599      0.466159      0.633839      7.79159
min        1.000000      0.000000      0.000000      1.000000      0.82000
25%        2.000000      0.000000      0.000000      1.000000     13.94000
50%        3.000000      0.000000      1.000000      1.000000     20.50000
75%        4.000000      0.000000      1.000000      2.000000     26.24000
max        4.000000      1.000000      1.000000      4.000000     41.00000

              atemp      humidity     windspeed        casual    registered  \
count  10886.000000  10886.000000  10886.000000  10886.000000  10886.000000
mean      23.655084     61.886460     12.799395     36.021955    155.552177
std        8.474601     19.245033      8.164537     49.960477    151.039033
min        0.760000      0.000000      0.000000      0.000000      0.000000
25%       16.665000     47.000000      7.001500      4.000000     36.000000
50%       24.240000     62.000000     12.998000     17.000000    118.000000
75%       31.060000     77.000000     16.997900     49.000000    222.000000
max       45.455000    100.000000     56.996900    367.000000    886.000000

              count
count  10886.000000
mean     191.574132
std      181.144454
min        1.000000
25%       42.000000
50%      145.000000
75%      284.000000
max      977.000000  )
```

There are no missing values in the dataset.

---

b. Identify missing values and perform Imputation using an appropriate method.

[4]:
```
missing_values = df.isnull().sum()
missing_values
```

[4]:
```
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
```

There are no duplicate records in the dataset.

---

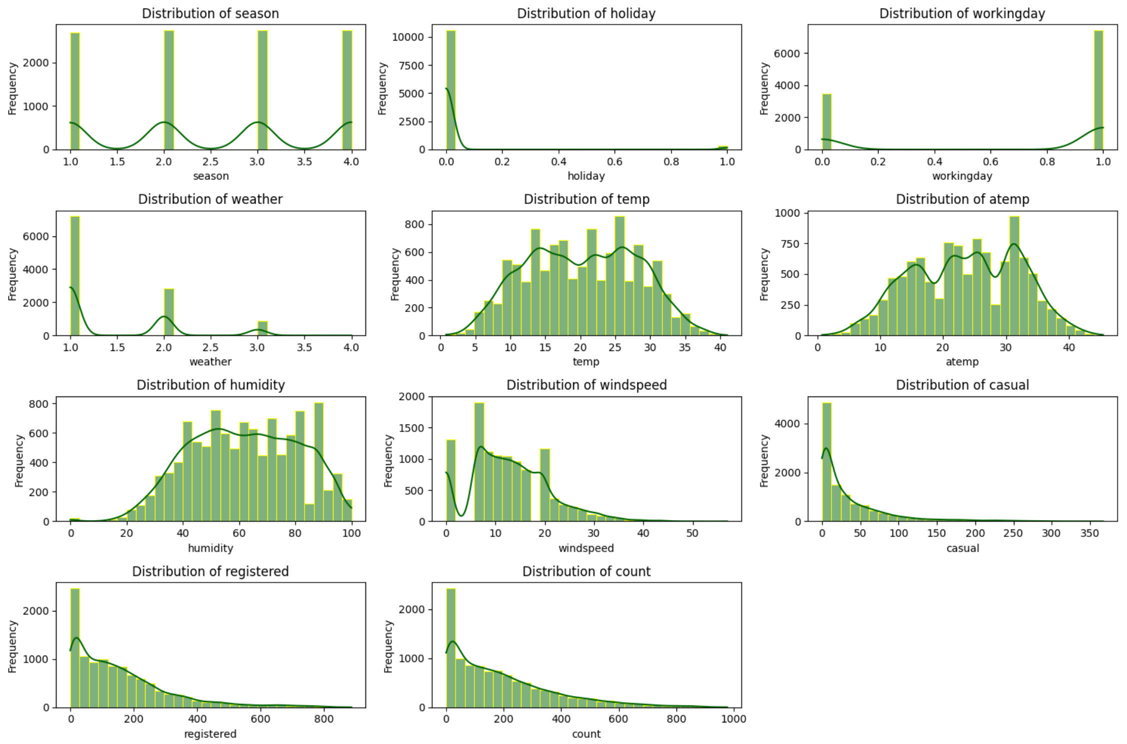
c. Identify and remove duplicate records.

[5]:
```
duplicate_records = df.duplicated().sum()
duplicate_records
```

[5]:
```
0
```

d. Analyze the distribution of Numerical & Categorical variables, separately

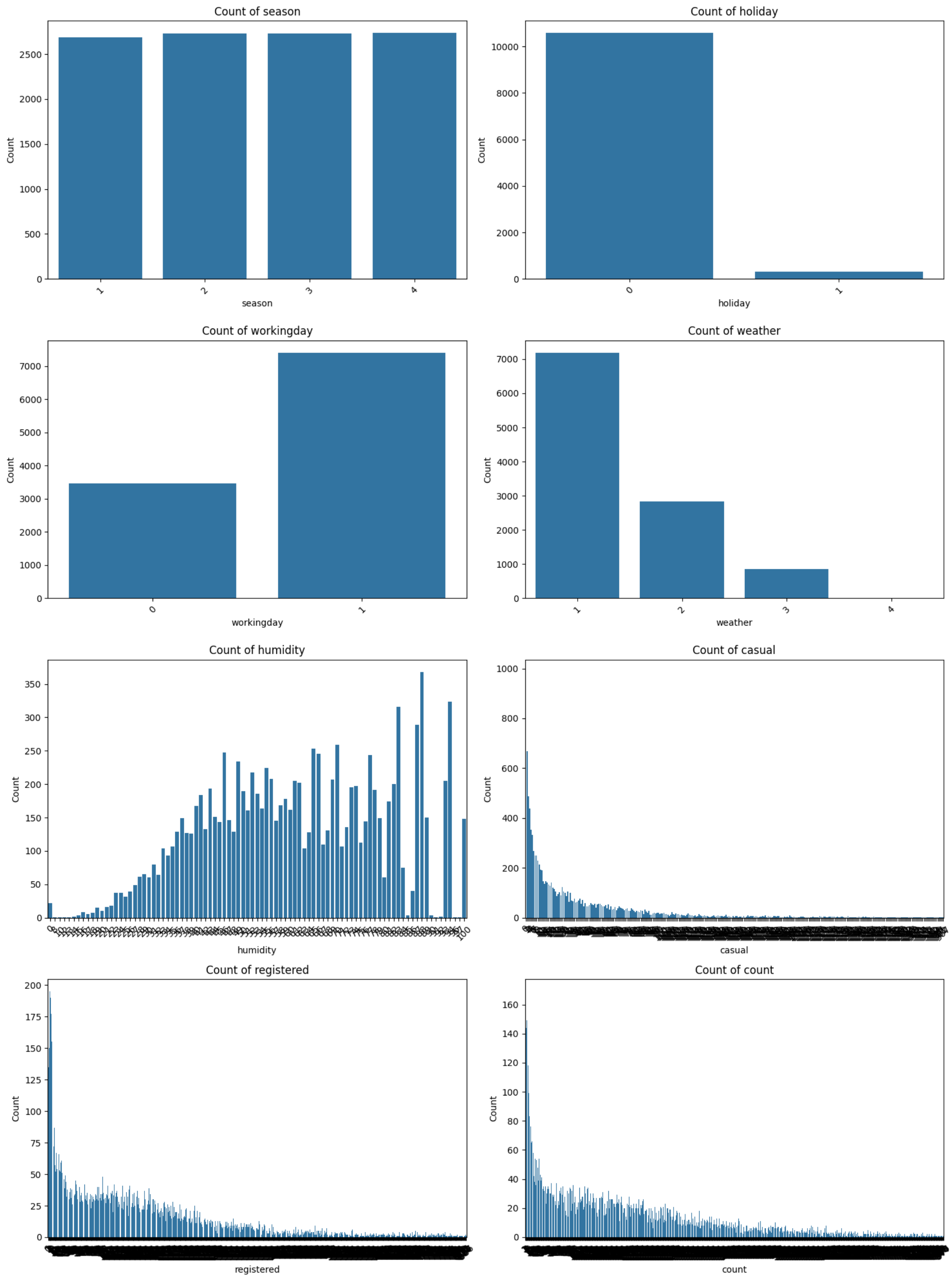
[6]:
```
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(4, 3, i)
    sns.histplot(df[column], kde=True, bins=30,color='darkgreen',␣
 ↪edgecolor='yellow')
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

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



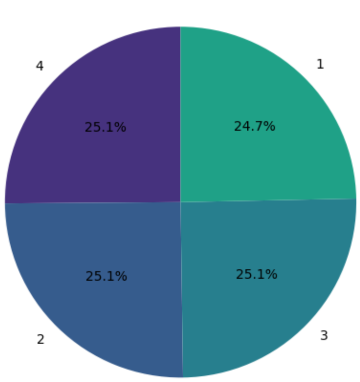
[7]:

```
categorical_columns = df.select_dtypes(include=['object', 'int64']).columns
categorical_columns = categorical_columns[categorical_columns != 'datetime']

num_cats = len(categorical_columns)
rows = (num_cats + 1) // 2  # Two plots per row

# Plotting count plots for categorical columns
plt.figure(figsize=(15, 5 * rows))
for i, column in enumerate(categorical_columns, 1):
    plt.subplot(rows, 2, i)
    sns.countplot(data=df, x=column)
    plt.title(f'Count of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# Plotting pie charts for categorical columns
plt.figure(figsize=(15, 5 * rows))
for i, column in enumerate(categorical_columns, 1):
    plt.subplot(rows, 2, i)
```

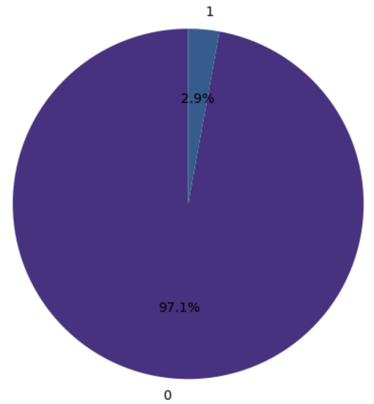
```
    df[column].value_counts().plot.pie(
        autopct='%1.1f%%', startangle=90, colors=sns.color_palette("viridis"))
    plt.title(f'Percentage Distribution of {column}')
    plt.ylabel('')  # Hide y-axis label for clarity

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


Count of season
Count
season
Count of holiday
Count
holiday
Count of workingday
Count
workingday
Count of weather
Count
weather
Count of humidity
Count
humidity
Count of casual
Count
casual
Count of registered
Count
registered
Count of count
Count
count

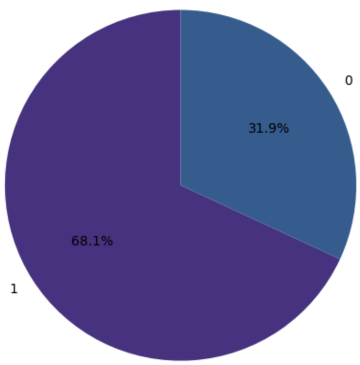
Percentage Distribution of season



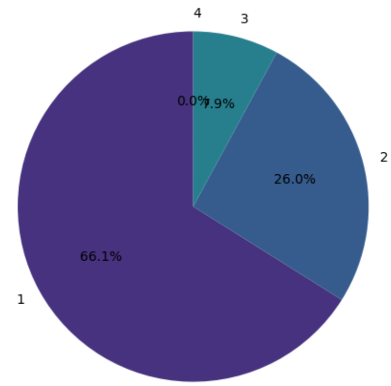
Percentage Distribution of holiday



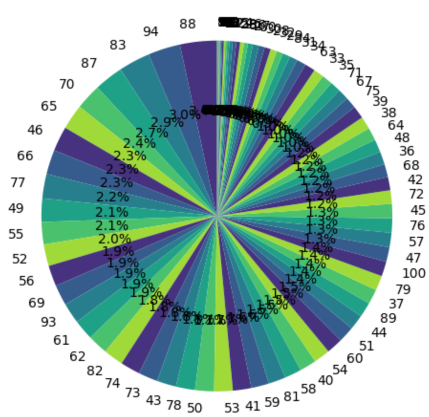
Percentage Distribution of workingday



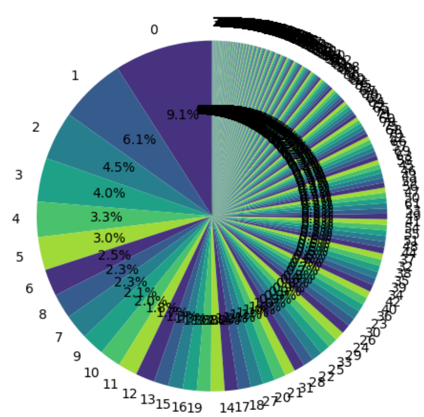
Percentage Distribution of weather



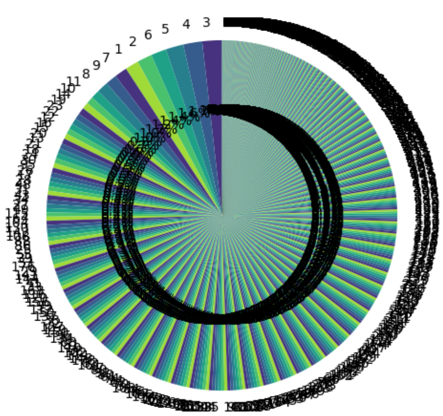
Percentage Distribution of humidity



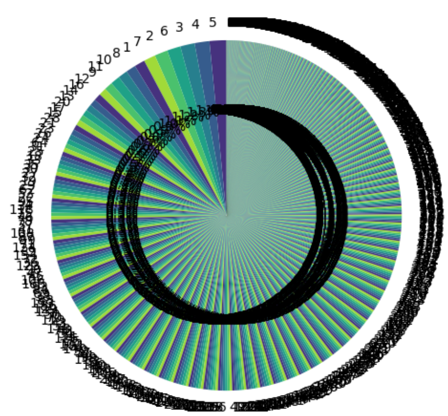
Percentage Distribution of casual



Percentage Distribution of registered

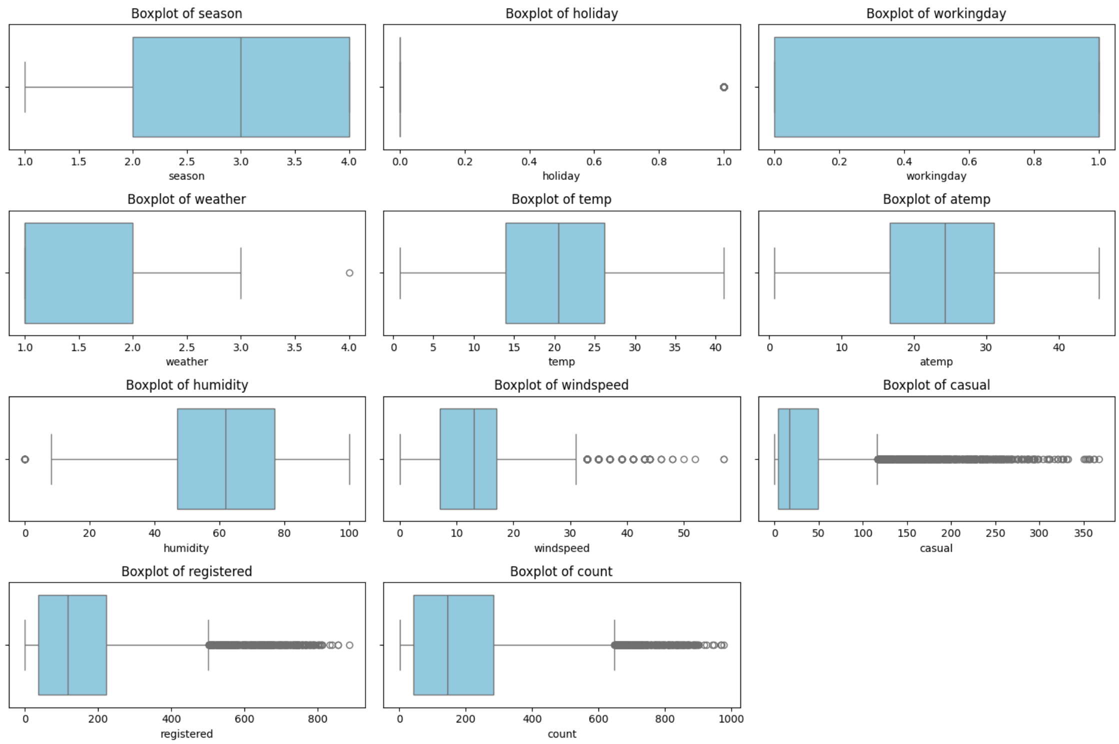


Percentage Distribution of count



e. Check for Outliers and deal with them accordingly.

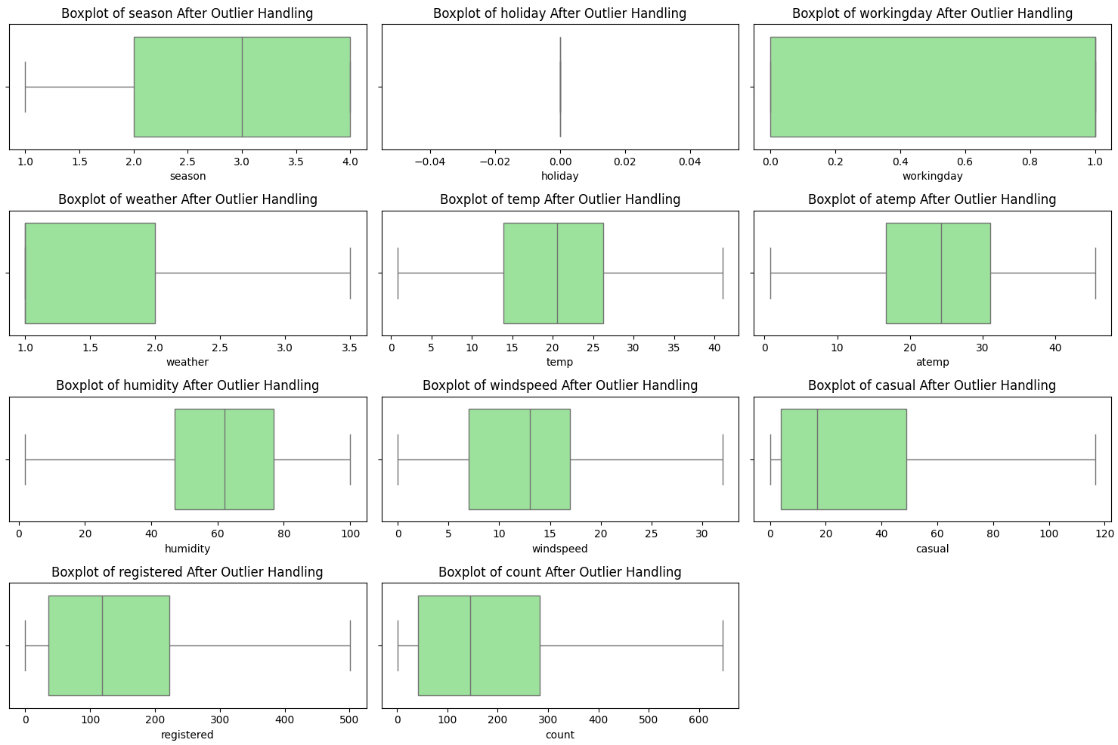
[8]:
```
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(data=df, x=column, color='skyblue')
    plt.title(f'Boxplot of {column}')

plt.tight_layout()
plt.show()

outlier_summary = {}
for column in numerical_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[column] < lower_bound) |
                                  (df[column] > upper_bound)]
    outlier_summary[column] = outliers.shape[0]

print("Outlier Counts for Numerical Columns:")
print(outlier_summary)
```




```
Outlier Counts for Numerical Columns:
{'season': 0, 'holiday': 311, 'workingday': 0, 'weather': 1, 'temp': 0, 'atemp':
0, 'humidity': 22, 'windspeed': 227, 'casual': 749, 'registered': 423, 'count':
300}
```

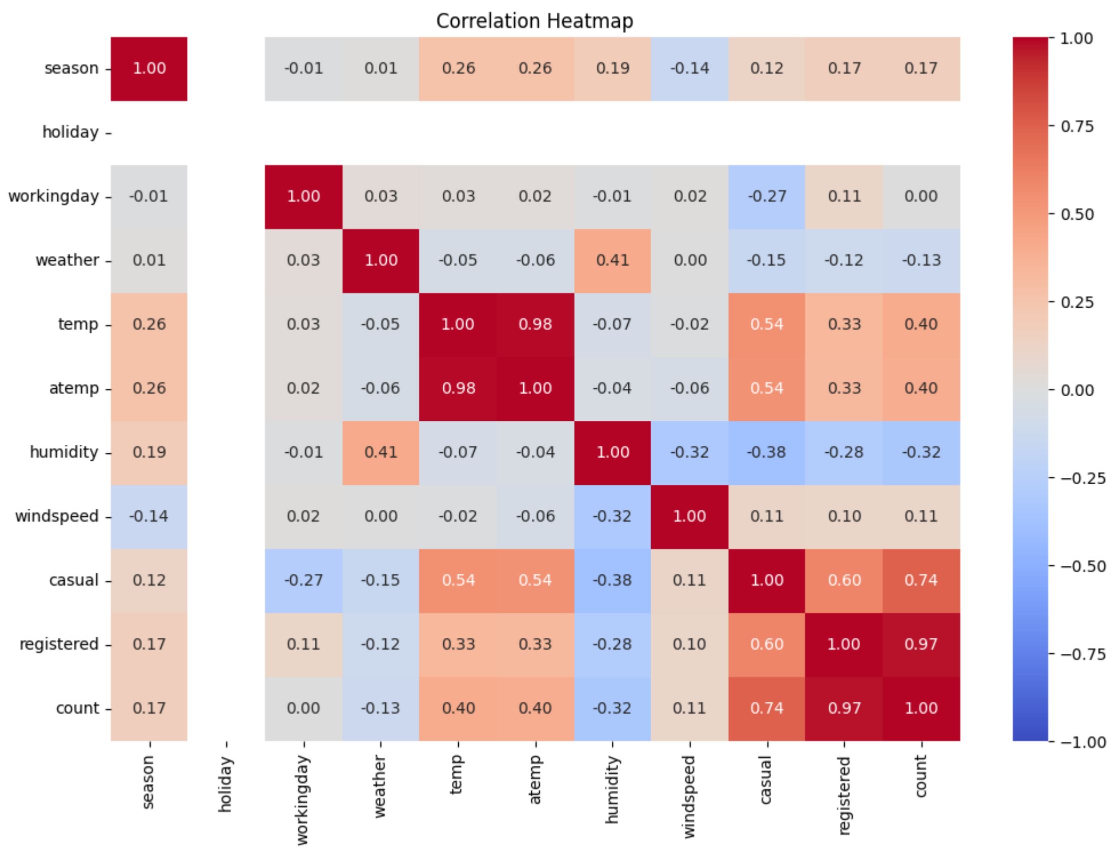
[9]:

```
for column in numerical_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)

plt.figure(figsize=(15, 10))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(data=df, x=column, color='lightgreen')
    plt.title(f'Boxplot of {column} After Outlier Handling')

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



2. Try establishing a Relationship between the Dependent and Independent Variables.

[10]:
```
numeric_data = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_data.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()

high_corr_pairs = []
threshold = 0.8

for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            high_corr_pairs.append((correlation_matrix.columns[i],␣
↪correlation_matrix.columns[j], correlation_matrix.iloc[i, j]))
```



[11]:
```python
print("Highly Correlated Variable Pairs (Threshold > 0.8):")
for pair in high_corr_pairs:
    print(f"{pair[0]} and {pair[1]}: Correlation = {pair[2]:.2f}")
```

```
Highly Correlated Variable Pairs (Threshold > 0.8):
atemp and temp: Correlation = 0.98
count and registered: Correlation = 0.97
```

3. Check if there any significant difference between the no. of bike rides on Weekdays and Weekends?

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1)

Null Hypothesis (H): There is no significant difference in the number of bike rides between weekdays and weekends.

Alternate Hypothesis (H): There is a significant difference in the number of bike rides between weekdays and weekends.

b. Select an appropriate test -

c. Set a significance level

d. Calculate test Statistics / p-value

e. Decide whether to accept or reject the Null Hypothesis.

[12]:
```
from scipy.stats import ttest_ind

df['day_of_week'] = pd.to_datetime(df['datetime']).dt.dayofweek
df['day_type'] = df['day_of_week'].apply(lambda x: 'Weekend' if x >= 5 else␣
↪'Weekday')

weekday_rides = df[df['day_type'] == 'Weekday']['count']
weekend_rides = df[df['day_type'] == 'Weekend']['count']

t_stat, p_value = ttest_ind(weekday_rides, weekend_rides, equal_var=False)

print("T-Statistic:", t_stat)
print("P-Value:", p_value)

alpha = 0.05
if p_value < alpha:
    print("Reject the Null Hypothesis (H ): There is a significant difference␣
↪in bike rides between weekdays and weekends.")
else:
    print("Fail to Reject the Null Hypothesis (H ): There is no significant␣
↪difference in bike rides between weekdays and weekends.")
```

```
T-Statistic: 0.20549142219541028
P-Value: 0.8371953236691908
Fail to Reject the Null Hypothesis (H ): There is no significant difference in
bike rides between weekdays and weekends.
```

4. Check if the demand of bicycles on rent is the same for different Weather conditions?

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1)

b. Select an appropriate test -

c. Check assumptions of the test

i. Normality

ii. Equality Variance

iii. Please continue doing the analysis even if some assumptions fail (Levene's test or Shapiro-wilk test) but double check using visual analysis and report wherever necessary.

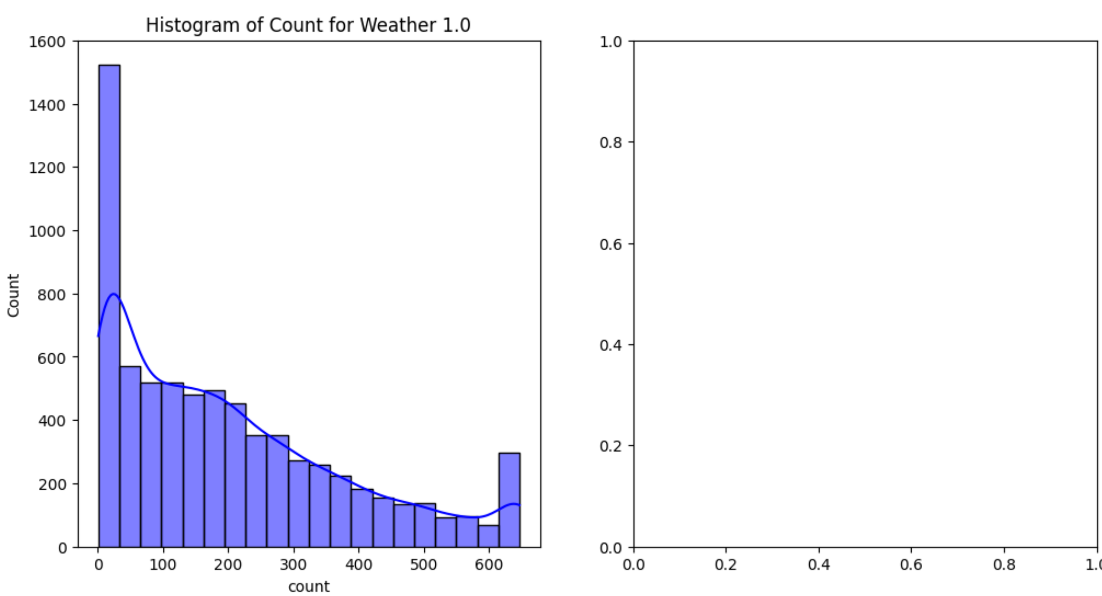
d. Set a significance level and Calculate the test Statistics / p-value.

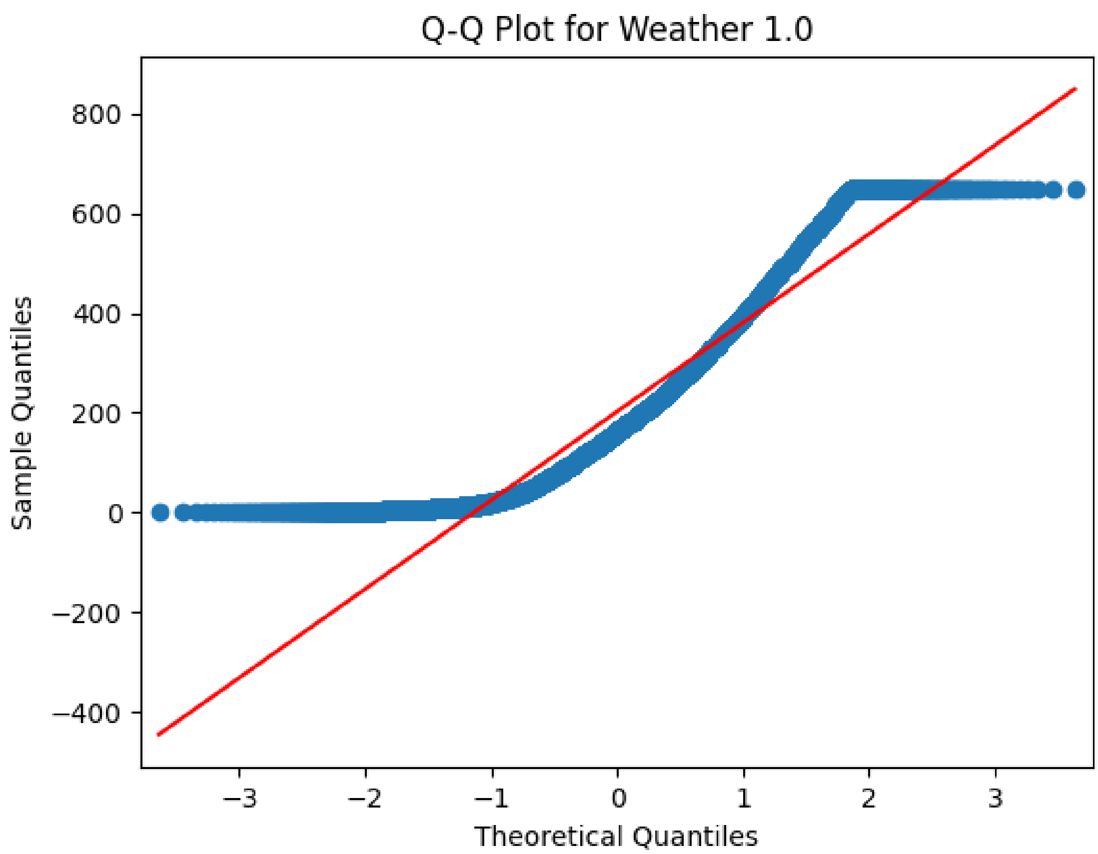
e. Decide whether to accept or reject the Null Hypothesis.

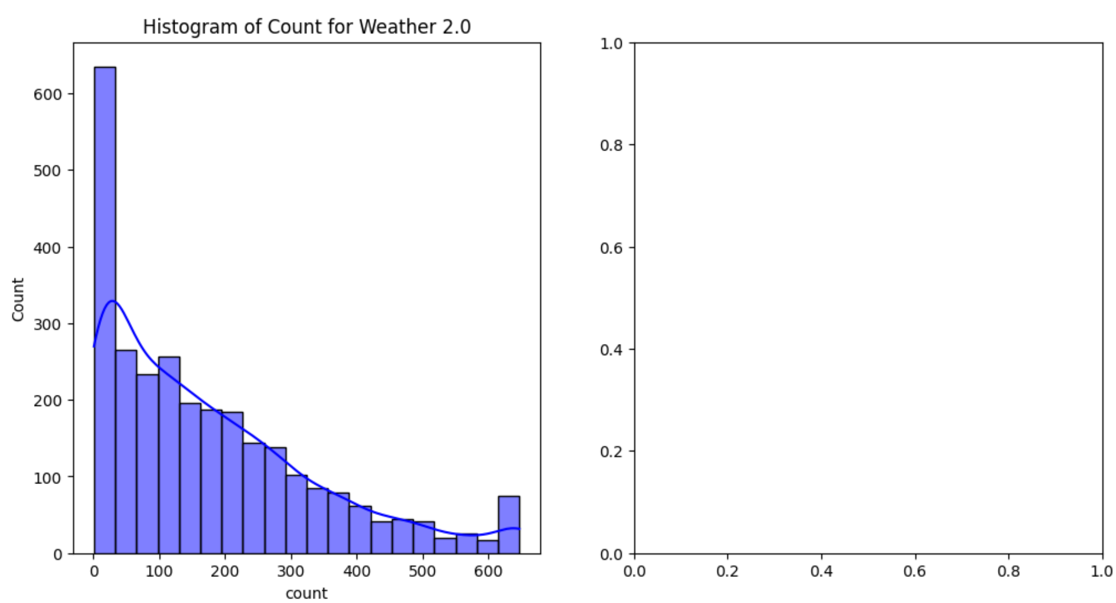
Null Hypothesis (H): The mean demand for bicycles is the same across all weather conditions.

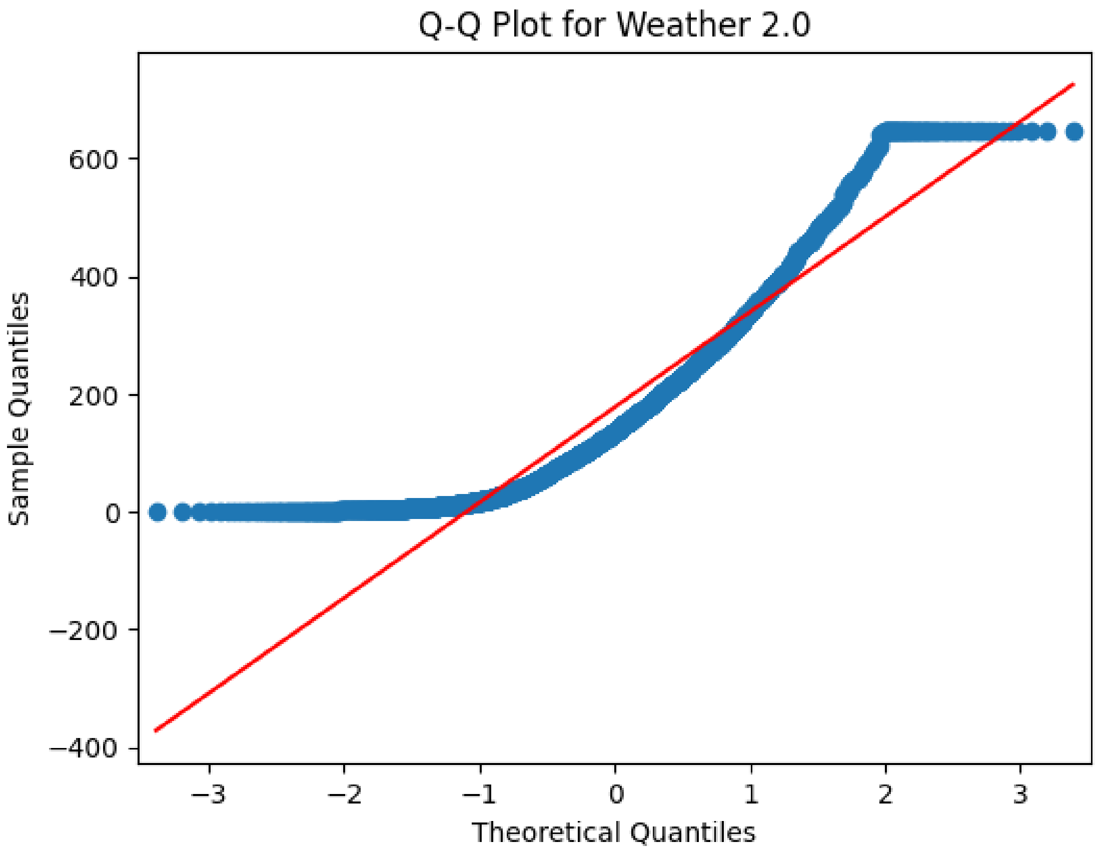
Alternate Hypothesis (H): The mean demand for bicycles differs across at least one weather condition.

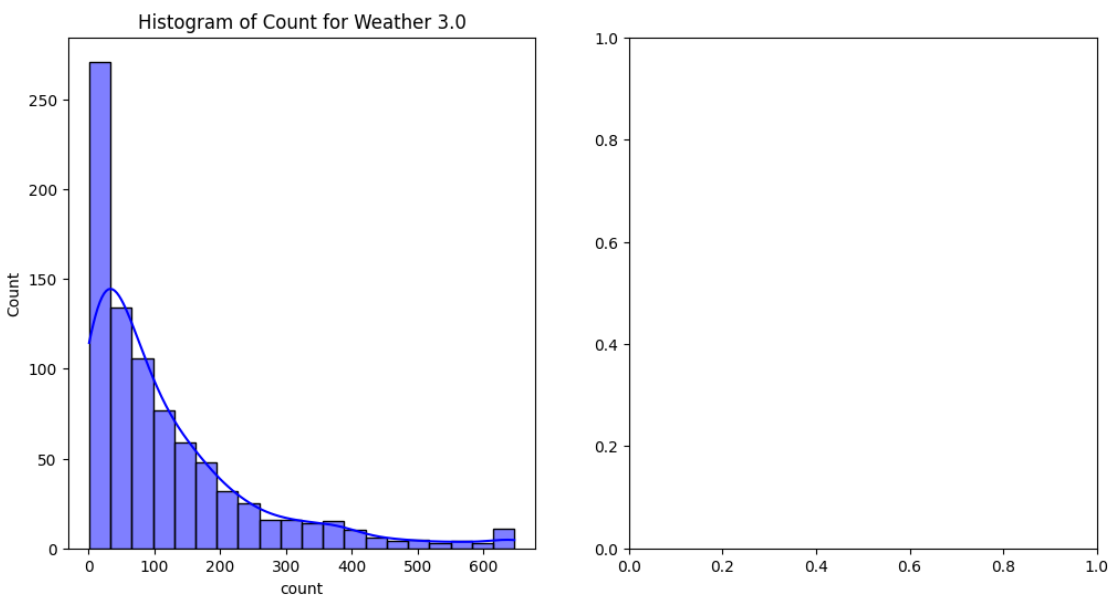
[13]:
```python
from scipy.stats import shapiro, levene, f_oneway
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Check normality for each weather condition group
weather_groups = df.groupby('weather')['count']

# Plot Histogram and Q-Q Plot
for weather, group_data in weather_groups:
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    sns.histplot(group_data, kde=True, bins=20, color='blue')
    plt.title(f'Histogram of Count for Weather {weather}')

    plt.subplot(1, 2, 2)
    sm.qqplot(group_data, line='s')
    plt.title(f'Q-Q Plot for Weather {weather}')
    plt.show()

# Perform Shapiro-Wilk test
for weather, group_data in weather_groups:
    stat, p = shapiro(group_data)
    print(f"Weather {weather}: Shapiro-Wilk Test Statistic = {stat:.4f}, p-value = {p:.4f}")
```

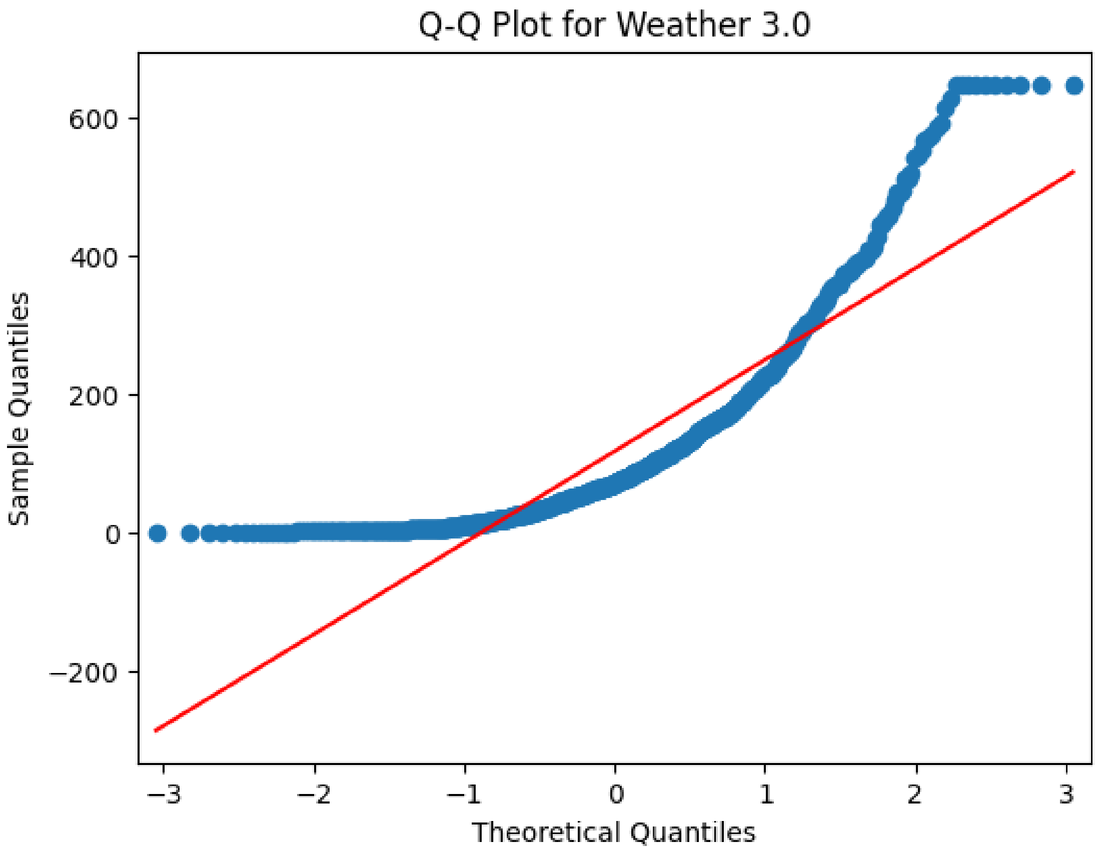


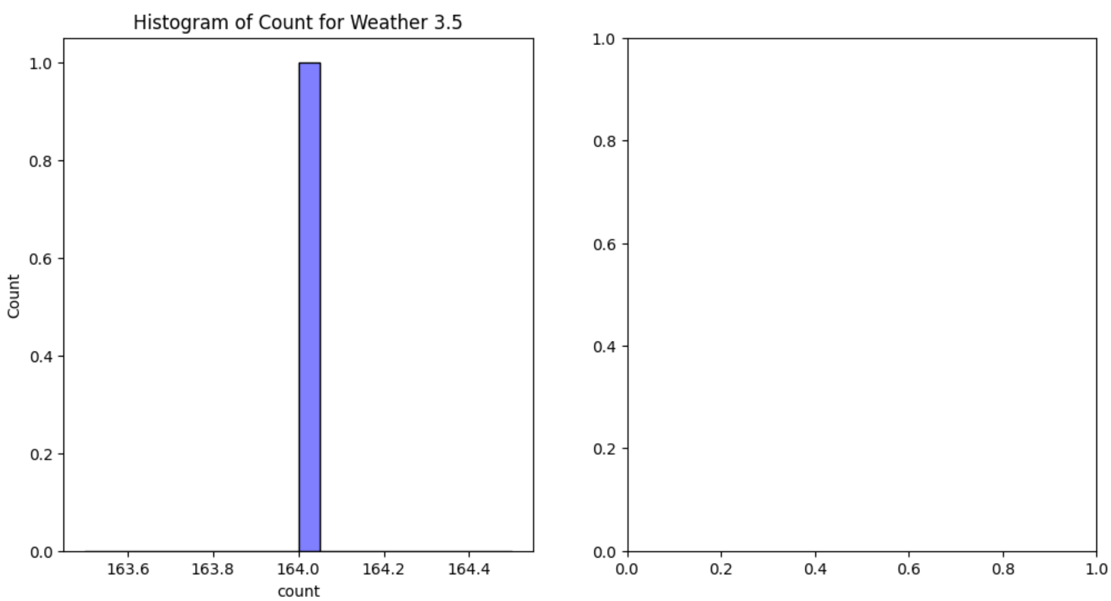

Q-Q Plot for Weather 1.0
Sample Quantiles
Theoretical Quantiles
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3

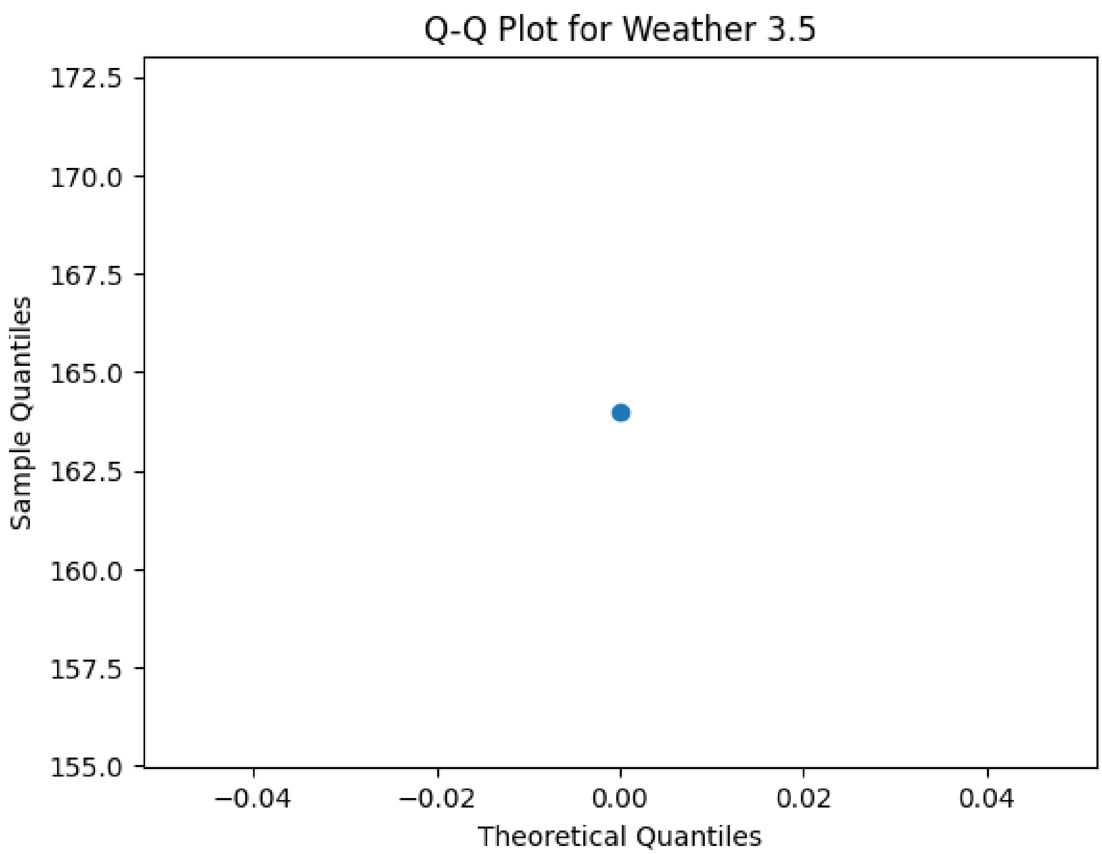

Histogram of Count for Weather 2.0
Count
count
0
100
200
300
400
500
600
0.0
0.2
0.4
0.6
0.8
1.0


Q-Q Plot for Weather 2.0
Sample Quantiles
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3
Theoretical Quantiles


Histogram of Count for Weather 3.0
Count
250
200
150
100
50
0
0
100
200
300
400
500
600
count
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Q-Q Plot for Weather 3.0
Sample Quantiles
Theoretical Quantiles
600
400
200
0
−200
−3
−2
−1
0
1
2
3


Histogram of Count for Weather 3.5
Count
count
1.0
0.8
0.6
0.4
0.2
0.0
163.6
163.8
164.0
164.2
164.4




```
/usr/local/lib/python3.10/dist-packages/scipy/stats/_axis_nan_policy.py:531:
UserWarning: scipy.stats.shapiro: For N > 5000, computed p-value may not be
accurate. Current N is 7192.
  res = hypotest_fun_out(*samples, **kwds)
```

```
Weather 1.0: Shapiro-Wilk Test Statistic = 0.8988, p-value = 0.0000
Weather 2.0: Shapiro-Wilk Test Statistic = 0.8865, p-value = 0.0000
Weather 3.0: Shapiro-Wilk Test Statistic = 0.7887, p-value = 0.0000
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-08e3cded7f5f> in <cell line: 22>()
     21 # Perform Shapiro-Wilk test
     22 for weather, group_data in weather_groups:
---> 23     stat, p = shapiro(group_data)
     24     print(f"Weather {weather}: Shapiro-Wilk Test Statistic = {stat:.4f} ␣
 ↪p-value = {p:.4f}")

/usr/local/lib/python3.10/dist-packages/scipy/stats/_axis_nan_policy.py in␣
 ↪axis_nan_policy_wrapper(***failed resolving arguments***)
    529                 if sentinel:
```

```
    530                         samples = _remove_sentinel(samples, paired, sentinel)
--> 531                     res = hypotest_fun_out(*samples, **kwds)
    532                     res = result_to_tuple(res)
    533                     res = _add_reduced_axes(res, reduced_axes, keepdims)

/usr/local/lib/python3.10/dist-packages/scipy/stats/_morestats.py in shapiro(x)
   1992     N = len(x)
   1993     if N < 3:
-> 1994         raise ValueError("Data must be at least length 3.")
   1995
   1996     a = zeros(N//2, dtype=np.float64)

ValueError: Data must be at least length 3.
```

[14]:
```python
# Levene's test
stat, p = levene(
    *[group_data for weather, group_data in weather_groups]
)
print(f"Levene’s Test Statistic = {stat:.4f}, p-value = {p:.4f}")
```

```
Levene’s Test Statistic = 59.7862, p-value = 0.0000
```

[15]:
```python
stat, p = f_oneway(
    *[group_data for weather, group_data in weather_groups]
)
print(f"One-Way ANOVA Test Statistic = {stat:.4f}, p-value = {p:.4f}")

alpha = 0.05
if p <= alpha:
    print("Reject the Null Hypothesis (H ): Bicycle demand differs across weather conditions.")
else:
    print("Fail to Reject the Null Hypothesis (H ): Bicycle demand is the same across weather conditions.")
```

```
One-Way ANOVA Test Statistic = 68.4117, p-value = 0.0000
Reject the Null Hypothesis (H ): Bicycle demand differs across weather
conditions.
```

5. Check if the demand of bicycles on rent is the same for different Seasons?

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1)

b. Select an appropriate test -

c. Check assumptions of the test

i. Normality

ii. Equality Variance

iii. Please continue doing the analysis even if some assumptions fail (Levene's test or Shapiro-wilk test) but double check using visual analysis and report wherever necessary.

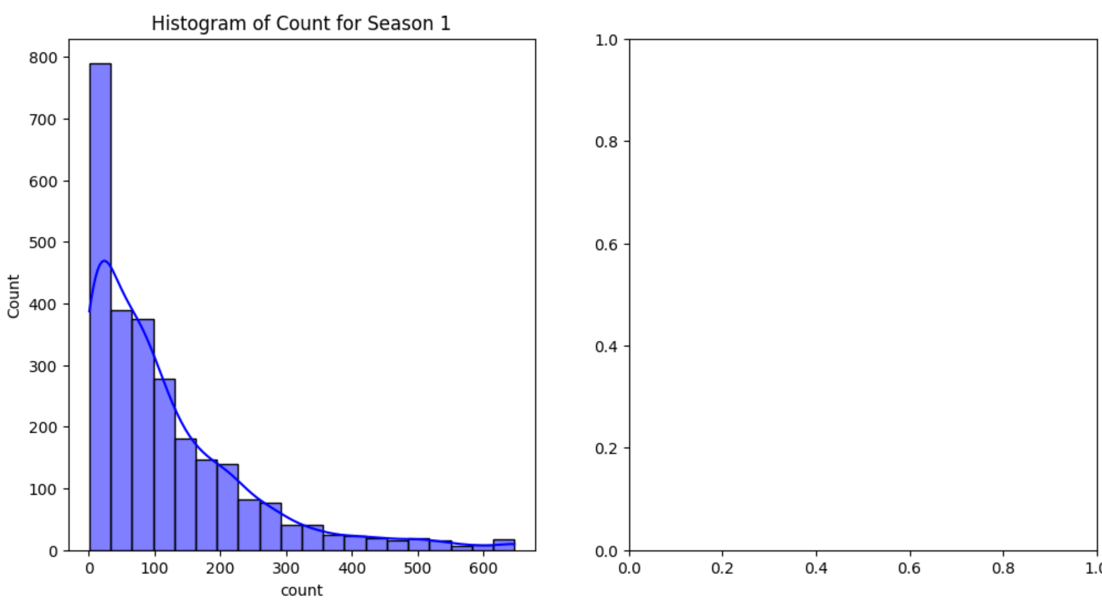
d. Set a significance level and Calculate the test Statistics / p-value.

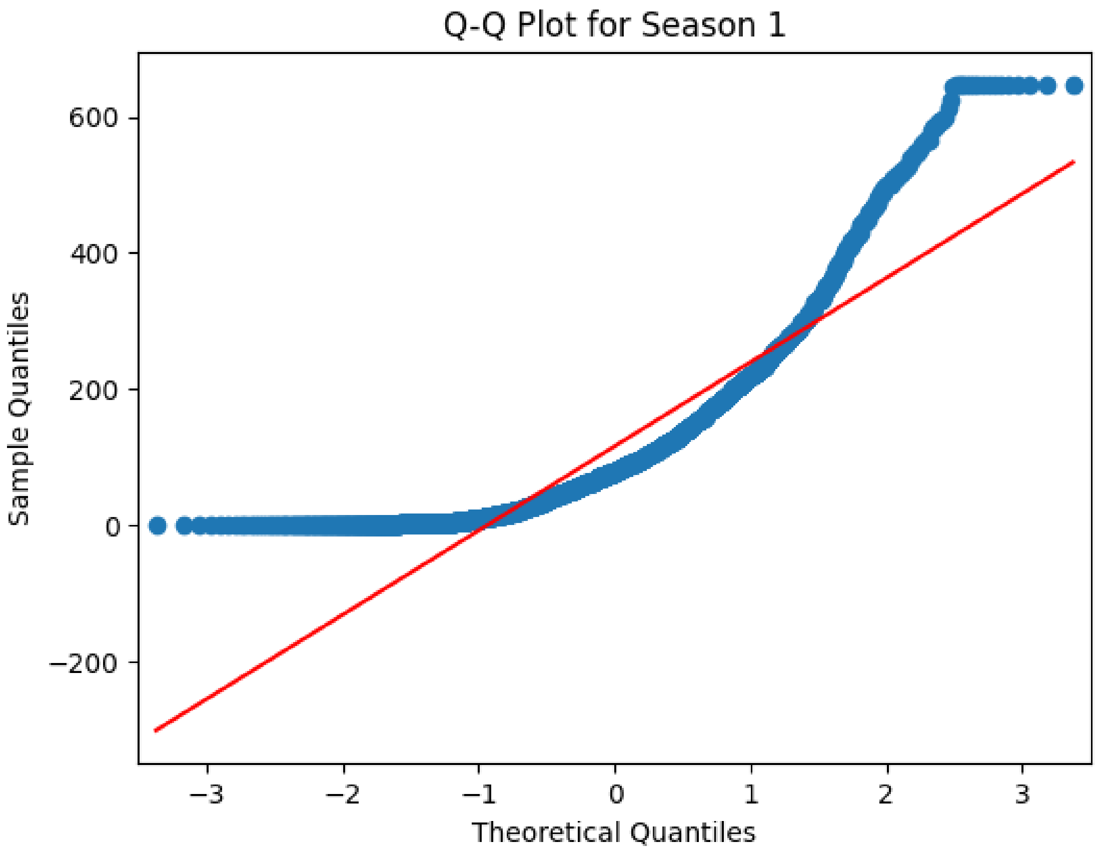
e. Decide whether to accept or reject the Null Hypothesis.

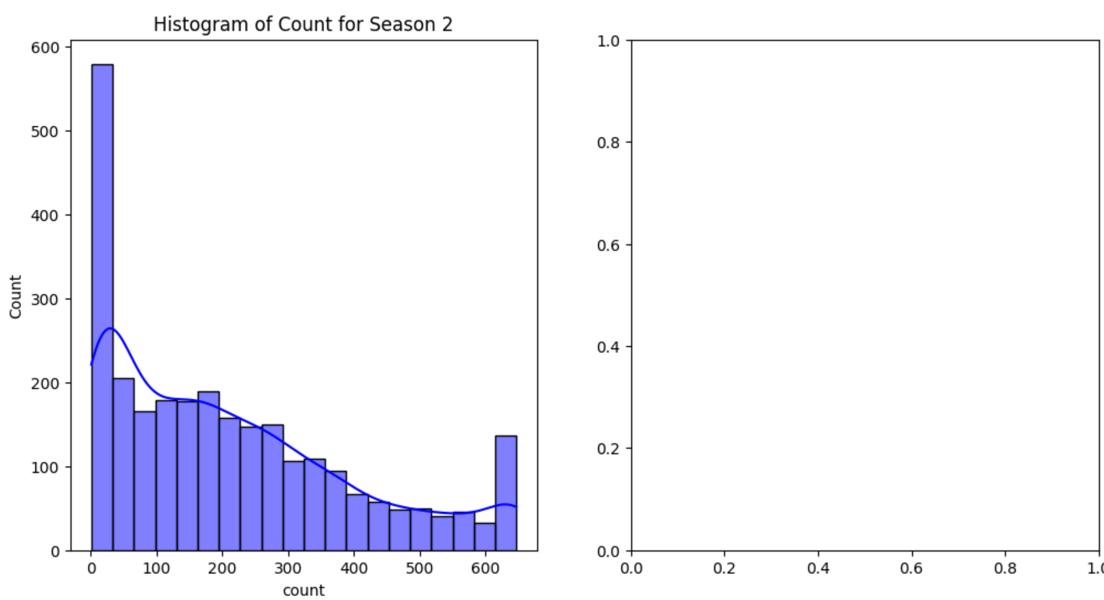
Null Hypothesis (H): The mean demand for bicycles is the same across all seasons.

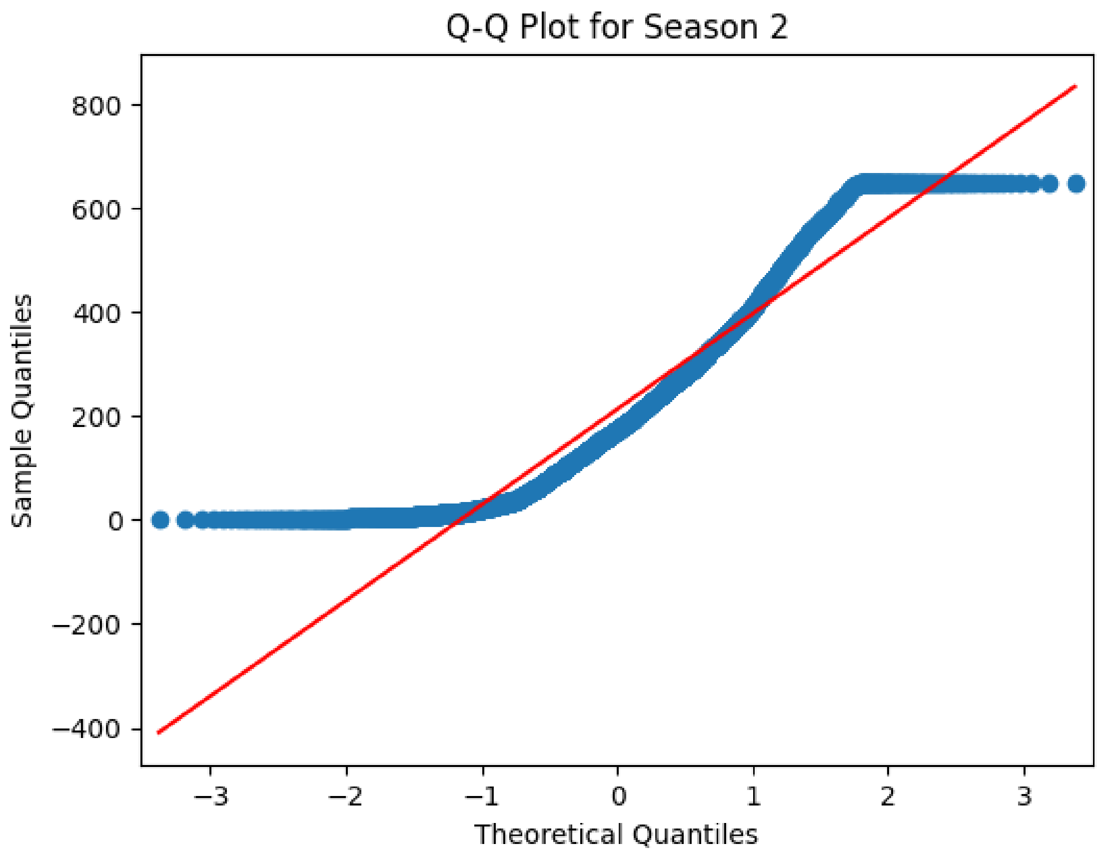
Alternate Hypothesis (H): The mean demand for bicycles differs across at least one season.

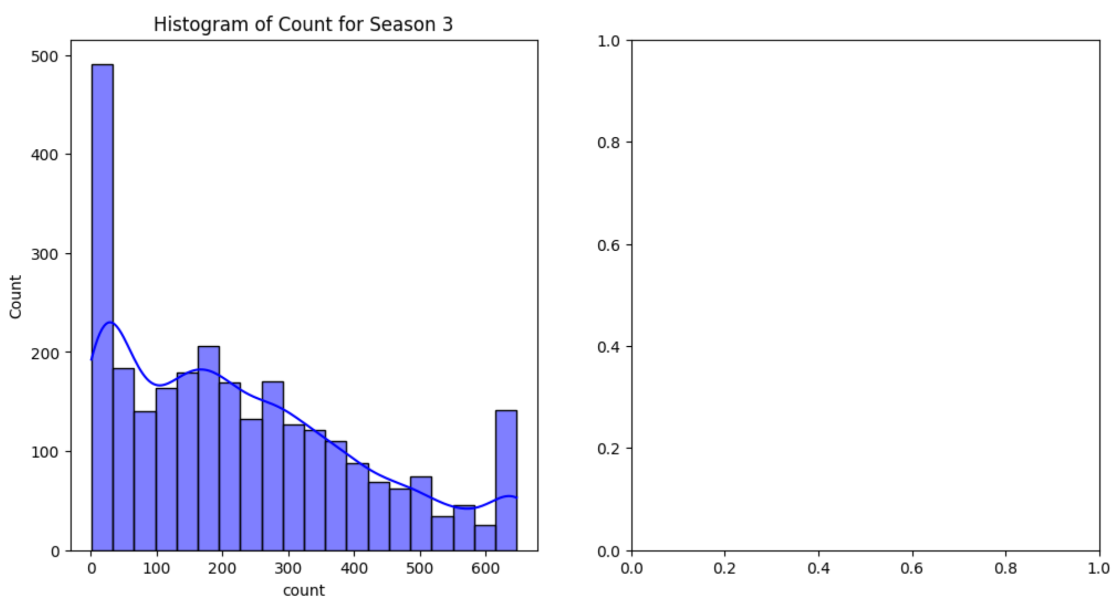
[16]:
```
season_groups = df.groupby('season')['count']

for season, group_data in season_groups:
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    sns.histplot(group_data, kde=True, bins=20, color='blue')
    plt.title(f'Histogram of Count for Season {season}')

    plt.subplot(1, 2, 2)
    sm.qqplot(group_data, line='s')
    plt.title(f'Q-Q Plot for Season {season}')
    plt.show()

for season, group_data in season_groups:
    stat, p = shapiro(group_data)
    print(f"Season {season}: Shapiro-Wilk Test Statistic = {stat:.4f}, p-value = {p:.4f}")
```

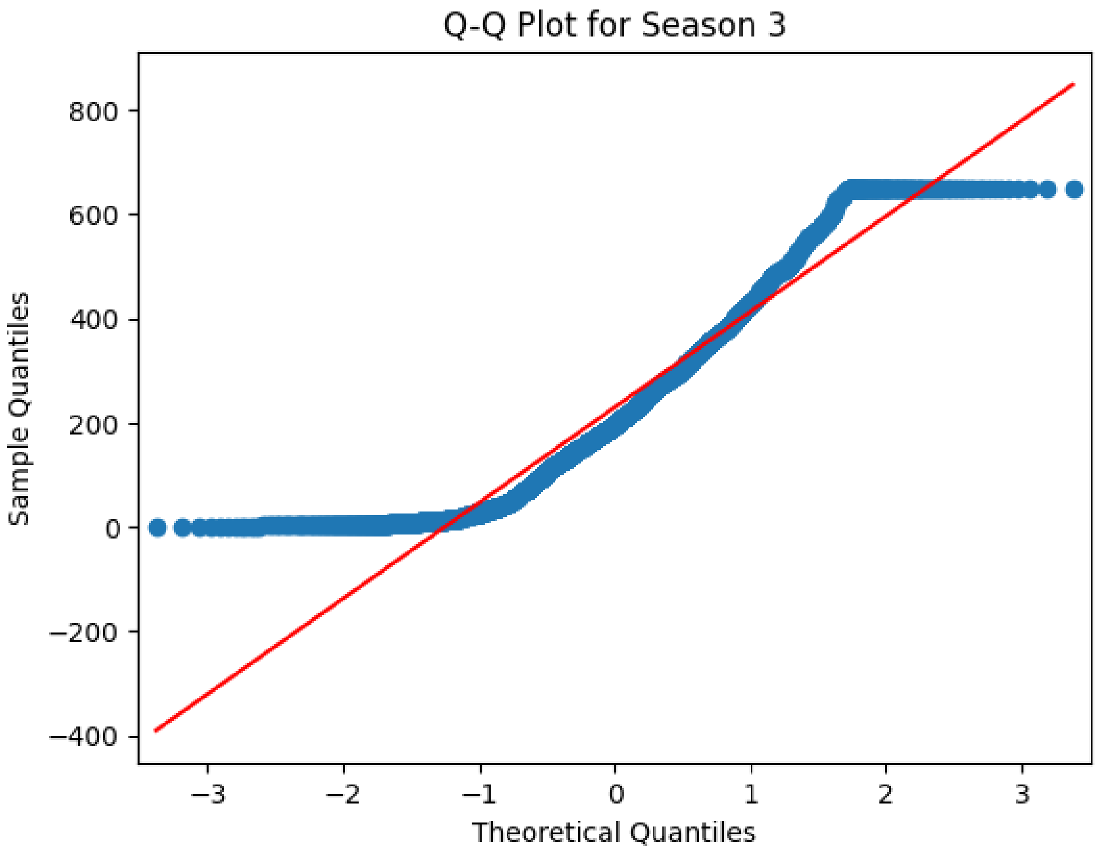



Q-Q Plot for Season 1
600
400
200
0
−200
Sample Quantiles
−3
−2
−1
0
1
2
3
Theoretical Quantiles

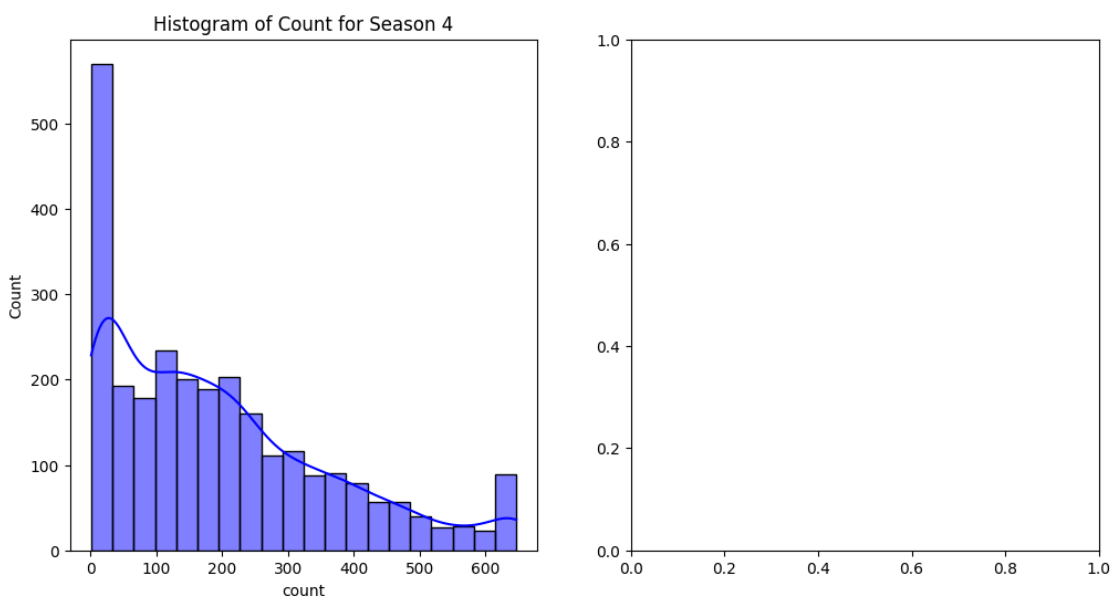

Histogram of Count for Season 2
600
500
400
300
200
100
0
Count
0
100
200
300
400
500
600
count
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Q-Q Plot for Season 2
Sample Quantiles
Theoretical Quantiles
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3

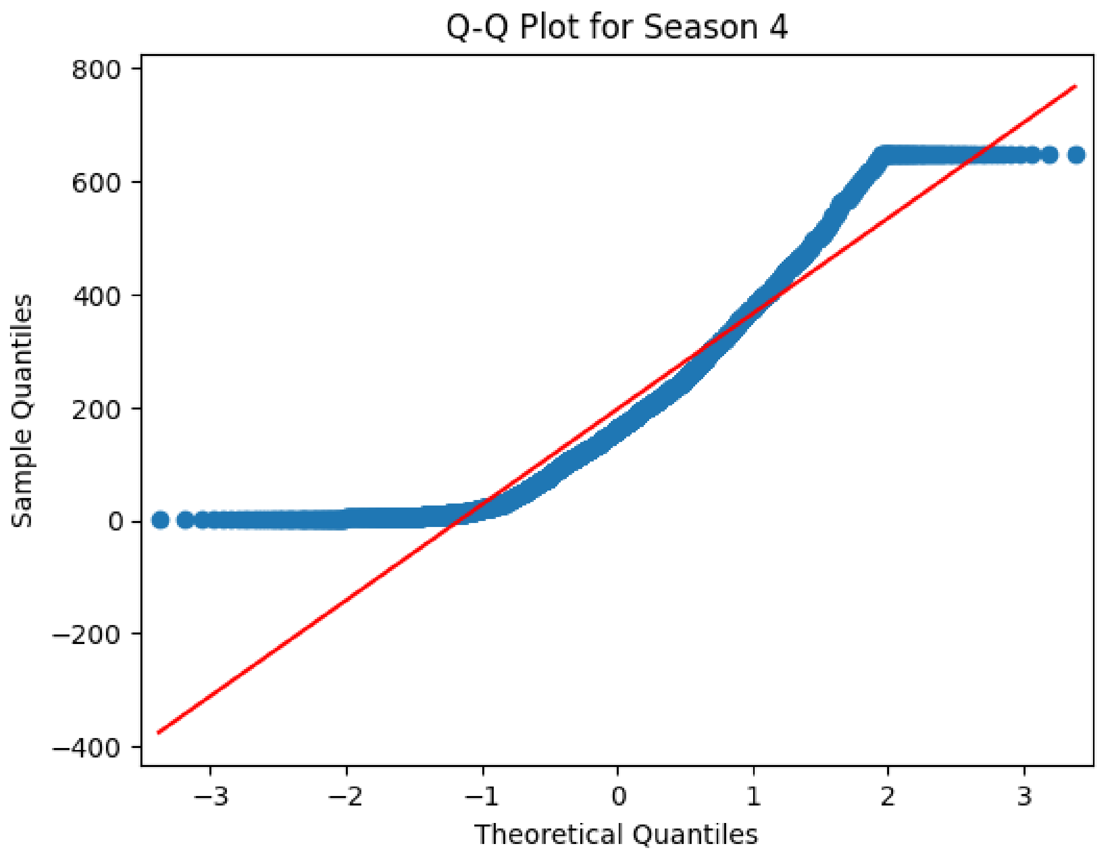

Histogram of Count for Season 3
Count
count
500
400
300
200
100
0
0
100
200
300
400
500
600
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Q-Q Plot for Season 3
Sample Quantiles
Theoretical Quantiles
800
600
400
200
0
−200
−400
−3
−2
−1
0
1
2
3



Histogram of Count for Season 4
Count
count
500
400
300
200
100
0
0
100
200
300
400
500
600
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0




```
Season 1: Shapiro-Wilk Test Statistic = 0.8147, p-value = 0.0000
Season 2: Shapiro-Wilk Test Statistic = 0.9028, p-value = 0.0000
Season 3: Shapiro-Wilk Test Statistic = 0.9246, p-value = 0.0000
Season 4: Shapiro-Wilk Test Statistic = 0.9057, p-value = 0.0000
```

[17]:

```
stat, p = levene(
    *[group_data for season, group_data in season_groups]
)
print(f"Levene's Test Statistic = {stat:.4f}, p-value = {p:.4f}")
```

```
Levene's Test Statistic = 199.5120, p-value = 0.0000
```

[18]:

```
stat, p = f_oneway(
    *[group_data for season, group_data in season_groups]
)
print(f"One-Way ANOVA Test Statistic = {stat:.4f}, p-value = {p:.4f}")
alpha = 0.05
if p <= alpha:
```

```
    print("Reject the Null Hypothesis (H ): Bicycle demand differs across␣
↪seasons.")
else:
    print("Fail to Reject the Null Hypothesis (H ): Bicycle demand is the same␣
↪across seasons.")
```

```
One-Way ANOVA Test Statistic = 243.3377, p-value = 0.0000
Reject the Null Hypothesis (H ): Bicycle demand differs across seasons.
```

6. Check if the Weather conditions are significantly different during different Seasons?

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1)

b. Select an appropriate test -

c. Create a Contingency Table against 'Weather' & 'Season' columns

d. Set a significance level and Calculate the test Statistics / p-value.

e. Decide whether to accept or reject the Null Hypothesis.

Null Hypothesis (H): There is no association between weather and season.

Alternate Hypothesis (H): There is an association between weather and season.

[19]:
```
from scipy.stats import chi2_contingency
contingency_table = pd.crosstab(df['season'], df['weather'])
print("Contingency Table:\n", contingency_table)
```

```
Contingency Table:
 weather   1.0  2.0  3.0  3.5
season
1         1759  715  211    1
2         1801  708  224    0
3         1930  604  199    0
4         1702  807  225    0
```

[20]:
```
chi2, p, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Square Statistic = {chi2:.4f}")
print(f"Degrees of Freedom = {dof}")
print(f"P-value = {p:.4f}")

alpha = 0.05
if p <= alpha:
    print("Reject the Null Hypothesis (H ): Weather and Season are␣
↪significantly associated.")
else:
    print("Fail to Reject the Null Hypothesis (H ): No significant association␣
↪between Weather and Season.")
```

```
Chi-Square Statistic = 49.1587
Degrees of Freedom = 9
P-value = 0.0000
Reject the Null Hypothesis (H ): Weather and Season are significantly
associated.
```

[20]:

```

```