yulu_case

November 22, 2024

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind,f_oneway,chi2_contingency,shapiro,levene
from statsmodels.stats.anova import AnovaRM
```

[6]:
```
#importing the dataset
bike = pd.read_csv(r"C:\Users\samvj\Downloads\bike_sharing.csv")
bike.head()
```

[6]:
```
           datetime  season  holiday  workingday  weather  temp   atemp  \
0  01-01-2011 00:00       1        0           0        1  9.84  14.395
1  01-01-2011 01:00       1        0           0        1  9.02  13.635
2  01-01-2011 02:00       1        0           0        1  9.02  13.635
3  01-01-2011 03:00       1        0           0        1  9.84  14.395
4  01-01-2011 04:00       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
```

[7]:
```
#examining the data type
bike.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
```

[8]:

```
#converting the data into desired data_type
bike["datetime"] = pd.to_datetime(bike["datetime"], errors="coerce")
bike["holiday"] = bike["holiday"].astype("object")
bike["workingday"] = bike["workingday"].astype("object")
bike["weather"] = bike["weather"].astype("object")
bike["season"] = bike["season"].astype("object")
```

[9]:

```
#after converting the data type checking for null in the dataset
bike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    6878 non-null   datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

[10]:

```
#finding the null data
missing_rows=bike[bike["datetime"].isnull()]
print(missing_rows.head())
```

```
    datetime season holiday workingday weather  temp  atemp  humidity  \
277      NaT      1       0          1       1  5.74   6.06        59
278      NaT      1       0          1       1  5.74   6.06        50
279      NaT      1       0          1       1  5.74   6.06        50
```

```
280       NaT      1       0          1       1  5.74   6.06         50
281       NaT      1       0          1       1  5.74   6.06         50

     windspeed  casual  registered  count
277    19.0012       1           6      7
278    19.0012       0           2      2
279    23.9994       0           2      2
280    22.0028       0           3      3
281    16.9979       0           4      4
```

[11]:
```
#filling the missing data by using filling method
bike["datetime"] = bike["datetime"].ffill().bfill()
```

[12]:
```
bike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

[13]:
```
#checking the filled null values
print(bike.isnull().sum())
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
```

```
registered    0
count         0
dtype: int64
```

[14]:
```
#Finding whether there is duplicates in data
np.any(bike.duplicated())
```

[14]: True

[15]:
```
duplicates = bike.duplicated().sum()
print(duplicates)
```

```
4
```

[16]:
```
print(bike[bike.duplicated()])
```

```
                datetime season holiday workingday weather   temp   atemp  \
858  2011-12-02 23:00:00      1       0          0       1  16.40  20.455
1185 2011-12-03 23:00:00      1       0          1       1  10.66  14.395
5258 2011-12-12 23:00:00      4       0          1       1   8.20  12.880
9931 2012-12-10 23:00:00      4       0          1       1  18.04  21.970

      humidity  windspeed  casual  registered  count
858         15    22.0028       0           3      3
1185        65     6.0032       0           1      1
5258        80     0.0000       0           4      4
9931        88    15.0013       0           5      5
```

[17]:
```
#removing the duplicates since there is only 4 duplictes
bike=bike.drop_duplicates()
```

[311]:
```
bike.info()
```

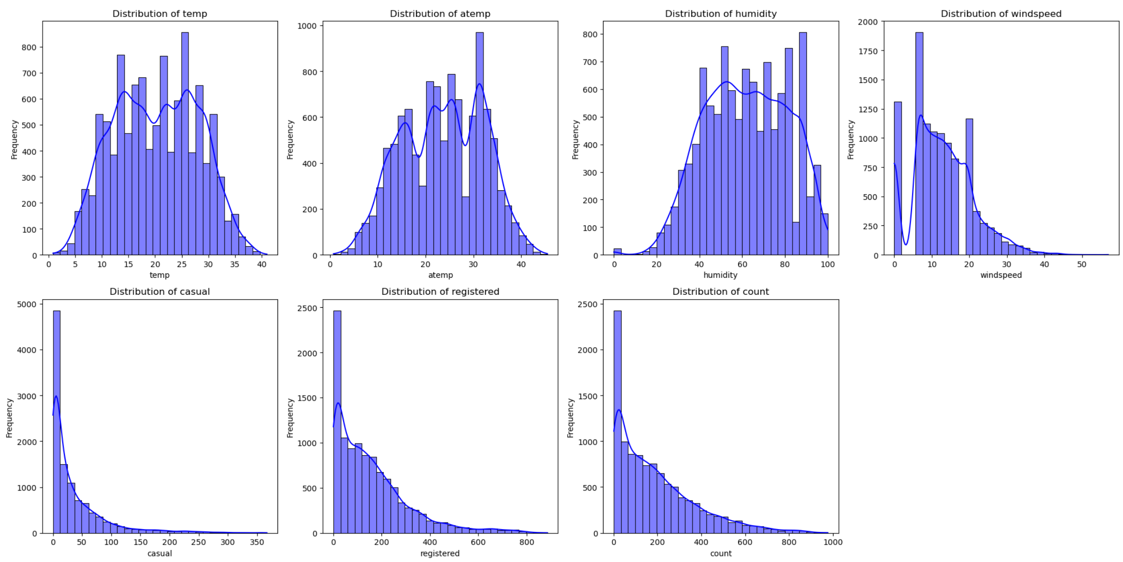
```
<class 'pandas.core.frame.DataFrame'>
Index: 10882 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10882 non-null  datetime64[ns]
 1   season      10882 non-null  object
 2   holiday     10882 non-null  object
 3   workingday  10882 non-null  object
 4   weather     10882 non-null  object
 5   temp        10882 non-null  float64
 6   atemp       10882 non-null  float64
 7   humidity    10882 non-null  int64
 8   windspeed   10882 non-null  float64
 9   casual      10882 non-null  int64
```

```
 10  registered  10882 non-null  int64
 11  count       10882 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1.1+ MB
```

[18]:
```
# Define numerical columns
numerical_cols = ["temp", "atemp", "humidity", "windspeed", "casual",␣
↪"registered", "count"]

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))  # Adjust the␣
↪layout
axes = axes.flatten()  # Flatten the axes array for easy iteration

# Plot histograms for each numerical column
for i, col in enumerate(numerical_cols):
    sns.histplot(bike[col], kde=True, bins=30, ax=axes[i], color='blue')
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

# Turn off the last unused subplot (if any)
if len(numerical_cols) < len(axes):
    for j in range(len(numerical_cols), len(axes)):
        axes[j].set_visible(False)

# Adjust layout
plt.tight_layout()
plt.show()
```

```
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
```

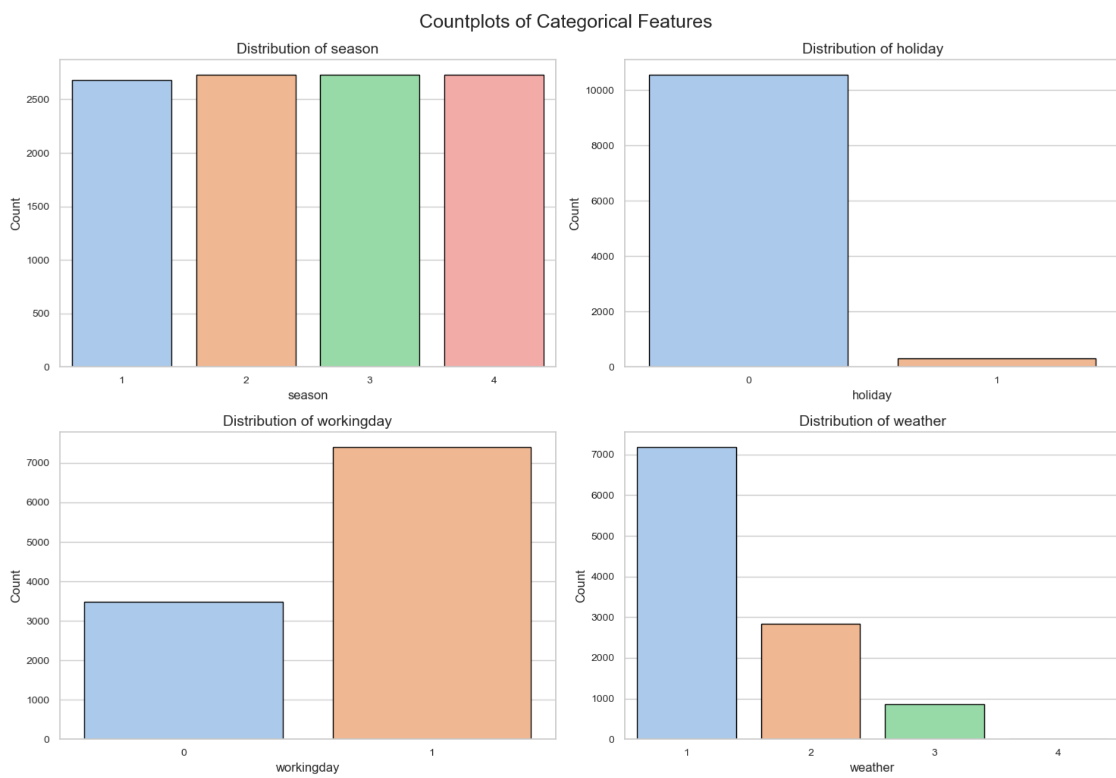
```
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\samvj\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[ ]:
```
"""
    distribution in temp is normal distribution
    distribution in atemp is normal distribution
    distribution in humidity is normal distribution
    distribution in windspeed is right-skewed
    distribution in casual is right-skewed
    distribution in registered is right-skewed
    distribution in count is right-skewed  """
```

[313]:
```
#Analysing the distribution on categorical variables

categorical_cols = ['season', 'holiday', 'workingday', 'weather']

# Set up subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))  # Adjust rows and columns as␣
 ↪needed
axes = axes.flatten()  # Flatten axes for easier indexing
```

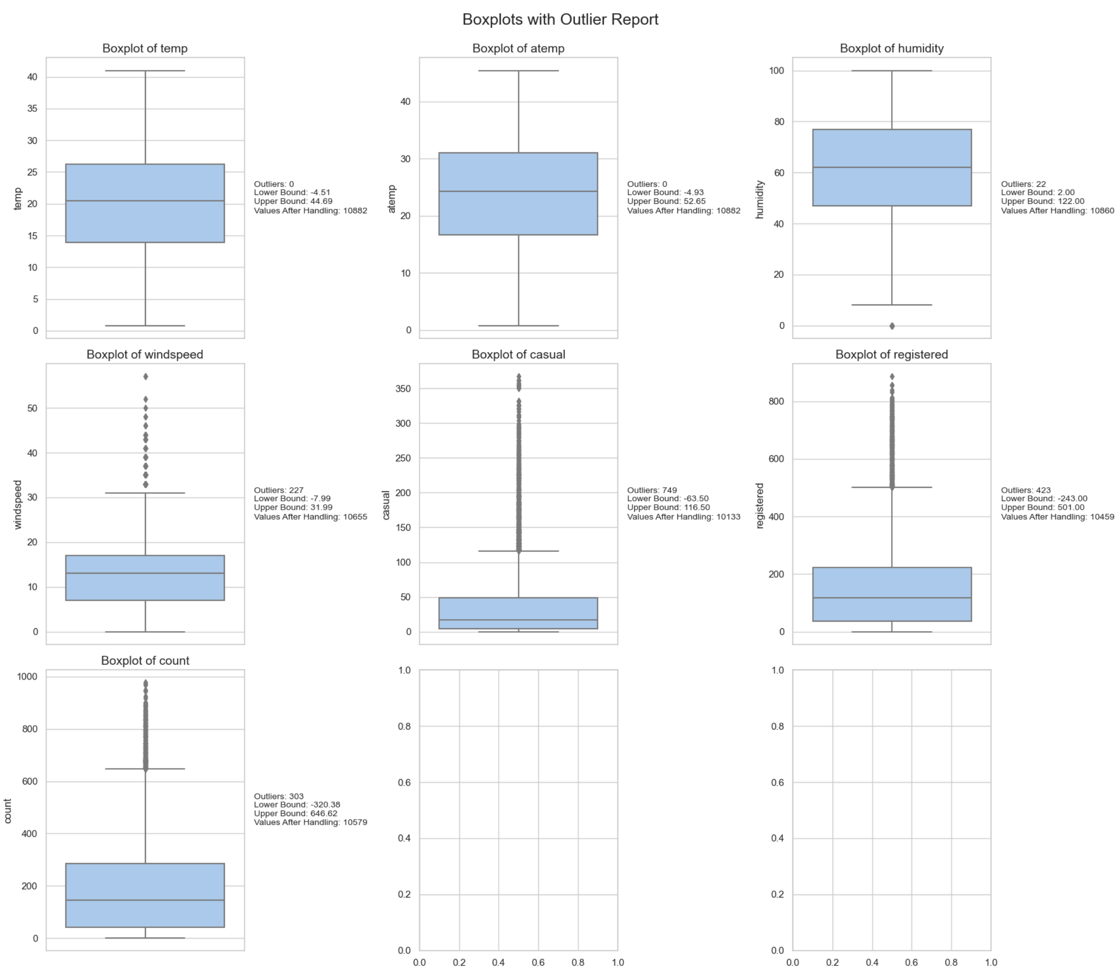
```
# Create countplots for each categorical variable
for i, col in enumerate(categorical_cols):
    sns.countplot(data=bike, x=col, palette='pastel', edgecolor='black',␣
 ↪ax=axes[i])
    axes[i].set_title(f'Distribution of {col}', fontsize=14)
    axes[i].set_xlabel(col, fontsize=12)
    axes[i].set_ylabel('Count', fontsize=12)
    axes[i].tick_params(axis='x', labelsize=10)
    axes[i].tick_params(axis='y', labelsize=10)

# Hide any unused subplot axes
for i in range(len(categorical_cols), len(axes)):
    axes[i].set_visible(False)

# Adjust layout
plt.tight_layout()
plt.suptitle('Countplots of Categorical Features', y=1.02, fontsize=18)
plt.show()
```



[314]:
```
#Checking for Outliers and solving them

# List of numerical columns
```

```
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual',␣
↪'registered', 'count']

# Initialize a dictionary to store the report data
outlier_report = {}

# Set up subplots
fig, axes = plt.subplots(3, 3, figsize=(18, 15))
axes = axes.flatten()  # Flatten the 2D array of axes for easy indexing

# Detect and handle outliers using IQR and plot side-by-side
for i, col in enumerate(numerical_cols):
    Q1 = bike[col].quantile(0.25)  # First quartile
    Q3 = bike[col].quantile(0.75)  # Third quartile
    IQR = Q3 - Q1  # Interquartile range
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = bike[(bike[col] < lower_bound) | (bike[col] > upper_bound)]

    # Store the count of outliers in the report
    outlier_report[col] = {
        'outlier_count': outliers.shape[0],  # Number of outliers
        'lower_bound': lower_bound,
        'upper_bound': upper_bound,
        'values_after_handling': bike[col].shape[0] - outliers.shape[0]  #␣
↪Count after removal
    }

    # Plot boxplot
    sns.boxplot(data=bike, y=col, palette='pastel', ax=axes[i])
    axes[i].set_title(f'Boxplot of {col}', fontsize=14)
    axes[i].set_ylabel(col, fontsize=12)

    # Add the outlier report as text beside the plot
    text = (f"Outliers: {outlier_report[col]['outlier_count']}\n"
            f"Lower Bound: {outlier_report[col]['lower_bound']:.2f}\n"
            f"Upper Bound: {outlier_report[col]['upper_bound']:.2f}\n"
            f"Values After Handling:␣
↪{outlier_report[col]['values_after_handling']}")
    axes[i].text(1.05, 0.5, text, fontsize=10, va='center', transform=axes[i].
↪transAxes)

    # Clip outliers
    bike[col] = np.where(bike[col] < lower_bound, lower_bound, bike[col])  #␣
↪Clip lower outliers
```

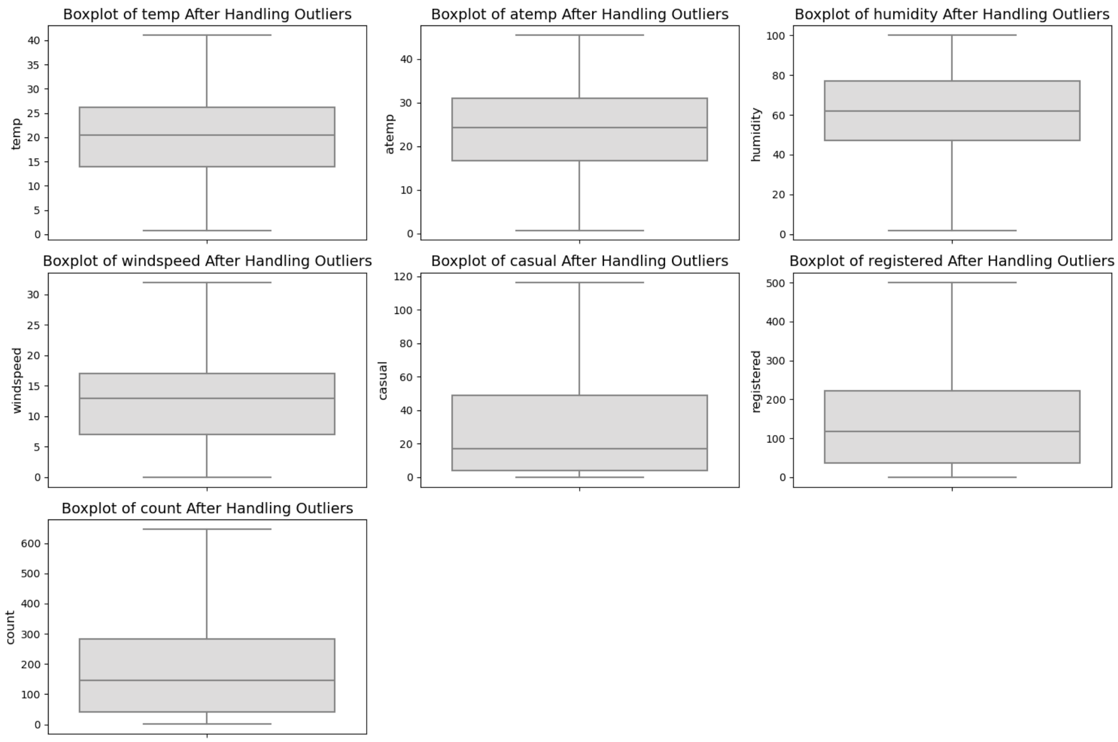
```
    bike[col] = np.where(bike[col] > upper_bound, upper_bound, bike[col])  # Clip upper outliers

# Adjust layout
plt.tight_layout()
plt.suptitle("Boxplots with Outlier Report", fontsize=18, y=1.02)
plt.show()

# Generate the consolidated report for outliers
print("Outlier Report:")
print("-" * 50)
for col, stats in outlier_report.items():
    print(f"Column: {col}")
    print(f"  - Number of outliers: {stats['outlier_count']}")
    print(f"  - Lower bound: {stats['lower_bound']}")
    print(f"  - Upper bound: {stats['upper_bound']}")
    print(f"  - Count of values after outlier handling: {stats['values_after_handling']}")
    print("-" * 50)
```



```
Outlier Report:
--------------------------------------------------
Column: temp
  - Number of outliers: 0
  - Lower bound: -4.51
  - Upper bound: 44.69
  - Count of values after outlier handling: 10882
--------------------------------------------------
Column: atemp
  - Number of outliers: 0
  - Lower bound: -4.9275000000000002
  - Upper bound: 52.6525
  - Count of values after outlier handling: 10882
--------------------------------------------------
Column: humidity
  - Number of outliers: 22
  - Lower bound: 2.0
  - Upper bound: 122.0
  - Count of values after outlier handling: 10860
--------------------------------------------------
Column: windspeed
  - Number of outliers: 227
  - Lower bound: -7.993100000000002
  - Upper bound: 31.992500000000003
  - Count of values after outlier handling: 10655
--------------------------------------------------
Column: casual
  - Number of outliers: 749
  - Lower bound: -63.5
  - Upper bound: 116.5
  - Count of values after outlier handling: 10133
--------------------------------------------------
Column: registered
  - Number of outliers: 423
  - Lower bound: -243.0
  - Upper bound: 501.0
  - Count of values after outlier handling: 10459
--------------------------------------------------
Column: count
  - Number of outliers: 303
  - Lower bound: -320.375
  - Upper bound: 646.625
  - Count of values after outlier handling: 10579
--------------------------------------------------
```

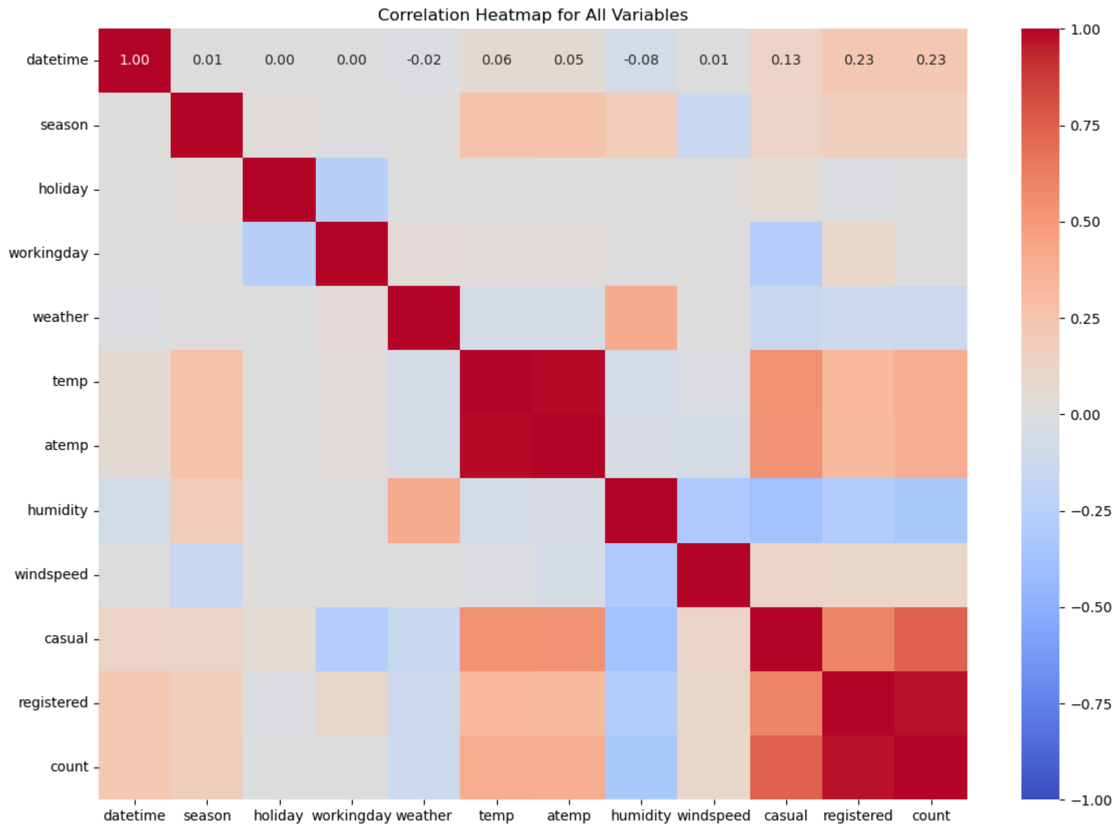
[19]:

```python
#solving the outliers by cliping them

#List of numerical columns
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual',
 ↪'registered', 'count']

# Detect and handle outliers using IQR
for col in numerical_cols:
    Q1 = bike[col].quantile(0.25)  # First quartile
    Q3 = bike[col].quantile(0.75)  # Third quartile
    IQR = Q3 - Q1  # Interquartile range
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove or clip outliers
    bike[col] = np.where(bike[col] < lower_bound, lower_bound, bike[col])  #
 ↪Clip lower outliers
    bike[col] = np.where(bike[col] > upper_bound, upper_bound, bike[col])  #
 ↪Clip upper outliers

# Verify changes with updated boxplots
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(data=bike, y=col, palette='coolwarm')
    plt.title(f'Boxplot of {col} After Handling Outliers', fontsize=14)
    plt.ylabel(col, fontsize=12)
plt.tight_layout()
plt.show()
```



[21]:
```
# Encode categorical variables to numeric codes
bike['season'] = bike['season'].astype('category').cat.codes
bike['holiday'] = bike['holiday'].astype('category').cat.codes
bike['workingday'] = bike['workingday'].astype('category').cat.codes
bike['weather'] = bike['weather'].astype('category').cat.codes

# Calculate correlation matrix
correlation_matrix = bike.corr()

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
 ↪vmin=-1, vmax=1)
plt.title("Correlation Heatmap for All Variables")
plt.show()

# Identify and remove highly correlated variables
# Set a threshold for high correlation, e.g., 0.9
threshold = 0.9
# Find the columns that have high correlations
high_correlation = correlation_matrix[(correlation_matrix > threshold) &
 ↪(correlation_matrix < 1.0)].stack().index.tolist()
```

```python
# List of highly correlated variables to remove
print("Highly correlated variables:", high_correlation)
```




```
Highly correlated variables: [('temp', 'atemp'), ('atemp', 'temp'),
('registered', 'count'), ('count', 'registered')]
```

[22]:
```python
print(correlation_matrix)
```

```
            datetime    season   holiday  workingday   weather      temp  \
datetime    1.000000  0.011926  0.003560    0.004691 -0.022222  0.057482
season      0.011926  1.000000  0.029377   -0.008395  0.008881  0.258836
holiday     0.003560  0.029377  1.000000   -0.250524 -0.007116  0.000239
workingday  0.004691 -0.008395 -0.250524    1.000000  0.033816  0.030102
weather    -0.022222  0.008881 -0.007116    0.033816  1.000000 -0.055266
temp        0.057482  0.258836  0.000239    0.030102 -0.055266  1.000000
atemp       0.053456  0.264867 -0.005262    0.024778 -0.055566  0.984948
humidity   -0.077853  0.190027  0.001895   -0.011231  0.407205 -0.065069
windspeed   0.006863 -0.143569  0.009380    0.015538  0.003856 -0.015792
casual      0.127722  0.123026  0.040824   -0.270983 -0.150544  0.542115
registered  0.231661  0.169282 -0.018118    0.108174 -0.115862  0.330379
count       0.229511  0.165837 -0.003174    0.003073 -0.131189  0.399396
```

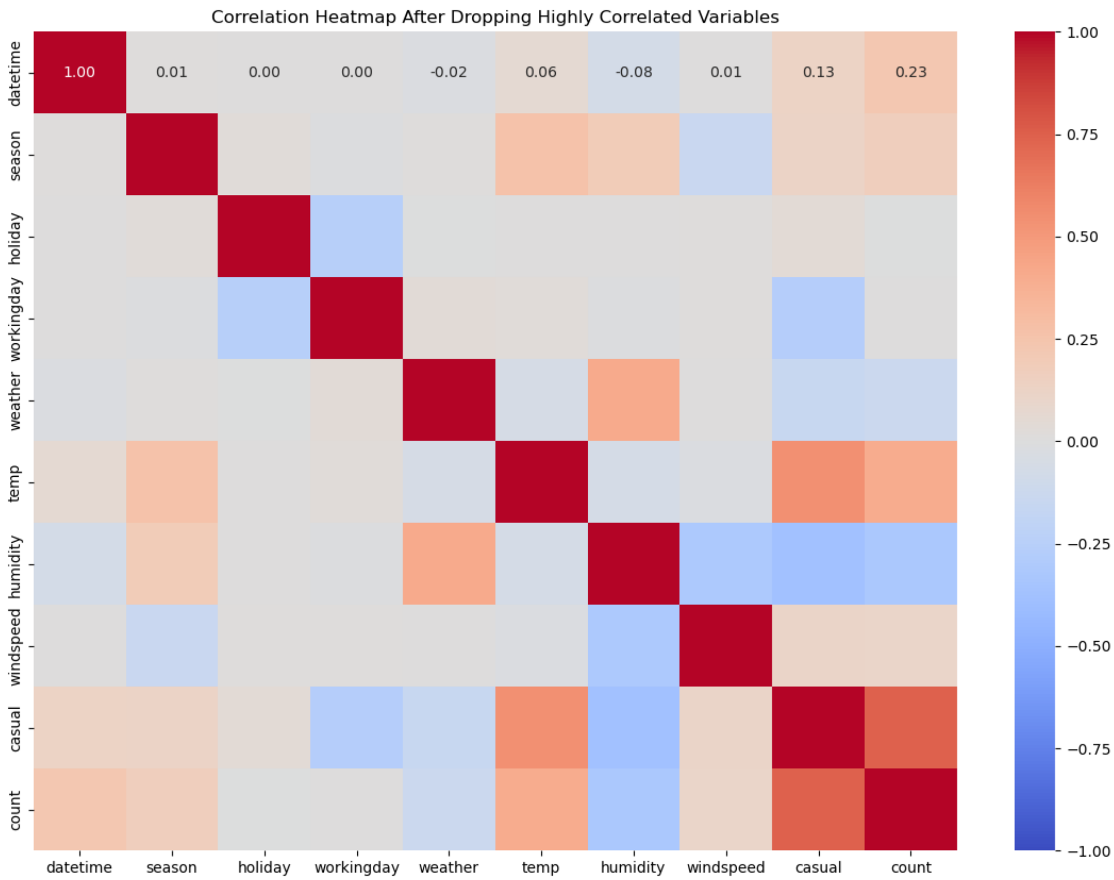
```
               atemp  humidity  windspeed    casual  registered     count
datetime    0.053456 -0.077853   0.006863  0.127722    0.231661  0.229511
season      0.264867  0.190027  -0.143569  0.123026    0.169282  0.165837
holiday    -0.005262  0.001895   0.009380  0.040824   -0.018118 -0.003174
workingday  0.024778 -0.011231   0.015538 -0.270983    0.108174  0.003073
weather    -0.055566  0.407205   0.003856 -0.150544   -0.115862 -0.131189
temp        0.984948 -0.065069  -0.015792  0.542115    0.330379  0.399396
atemp       1.000000 -0.043630  -0.055552  0.535367    0.326579  0.394927
humidity   -0.043630  1.000000  -0.319982 -0.378502   -0.283447 -0.324086
windspeed  -0.055552 -0.319982   1.000000  0.110580    0.103090  0.109014
casual      0.535367 -0.378502   0.110580  1.000000    0.599522  0.744376
registered  0.326579 -0.283447   0.103090  0.599522    1.000000  0.971970
count       0.394927 -0.324086   0.109014  0.744376    0.971970  1.000000
```

[23]:

```python
# Drop 'atemp' and 'registered' columns due to high correlation
bike_cleaned = bike.drop(columns=['atemp', 'registered'])

# Recalculate the correlation matrix after dropping highly correlated columns
correlation_matrix_cleaned = bike_cleaned.corr()

# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix_cleaned)

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix_cleaned, annot=True, cmap='coolwarm', fmt=".2f",
    vmin=-1, vmax=1)
plt.title("Correlation Heatmap After Dropping Highly Correlated Variables")
plt.show()
```

```
Correlation Matrix:
            datetime    season   holiday  workingday   weather      temp  \
datetime    1.000000  0.011926  0.003560    0.004691 -0.022222  0.057482
season      0.011926  1.000000  0.029377   -0.008395  0.008881  0.258836
holiday     0.003560  0.029377  1.000000   -0.250524 -0.007116  0.000239
workingday  0.004691 -0.008395 -0.250524    1.000000  0.033816  0.030102
weather    -0.022222  0.008881 -0.007116    0.033816  1.000000 -0.055266
temp        0.057482  0.258836  0.000239    0.030102 -0.055266  1.000000
humidity   -0.077853  0.190027  0.001895   -0.011231  0.407205 -0.065069
windspeed   0.006863 -0.143569  0.009380    0.015538  0.003856 -0.015792
casual      0.127722  0.123026  0.040824   -0.270983 -0.150544  0.542115
count       0.229511  0.165837 -0.003174    0.003073 -0.131189  0.399396

            humidity  windspeed    casual     count
datetime   -0.077853   0.006863  0.127722  0.229511
season      0.190027  -0.143569  0.123026  0.165837
```

```
holiday      0.001895   0.009380  0.040824 -0.003174
workingday -0.011231   0.015538 -0.270983  0.003073
weather      0.407205   0.003856 -0.150544 -0.131189
temp        -0.065069  -0.015792  0.542115  0.399396
humidity     1.000000  -0.319982 -0.378502 -0.324086
windspeed   -0.319982   1.000000  0.110580  0.109014
casual      -0.378502   0.110580  1.000000  0.744376
count       -0.324086   0.109014  0.744376  1.000000
```



[ ]:
```
""" Check if there any signi cant difference between the no. of bike rides on␣
↪Weekdays and Weekends?
Formulating H0 and H1:
Null Hypothesis (H0): There is no signi cant difference in the number of bike␣
↪rides between weekdays and weekends.
Alternate Hypothesis (H1): There is a signi cant difference in the number of␣
↪bike rides between weekdays and weekends """
```

[31]:
```
# Extract weekday and weekend data
weekday_data = bike[bike['workingday'] == 1]['count']
```

```
weekend_data = bike[bike['workingday'] == 0]['count']

# Compute and print variances
weekday_variance = np.var(weekday_data, ddof=0)  # Population variance
weekend_variance = np.var(weekend_data, ddof=0)  # Population variance

print(f"Weekday Variance: {weekday_variance}")
print(f"Weekend Variance: {weekend_variance}")
```

```
Weekday Variance: 29760.907355018793
Weekend Variance: 29611.16524206959
```

[ ]:

```
"""Before conducting the two-sample T-Test we need to nd if the given data␣
↪groups have the same variance. If the ratio of the larger data groups
to the small data group is less than 4:1 then we can consider that the given␣
↪data groups have equal variance.
Here, the ratio is 29760.90 / 29611.16 which is less than 4:1"""
```

[24]:

```
# Create a new column to indicate whether the day is a weekday or weekend
bike_cleaned['day_type'] = bike_cleaned['workingday'].apply(lambda x: 'Weekday'␣
↪if x == 1 else 'Weekend')

# Calculate the number of bike rides for weekdays and weekends
weekday_rides = bike_cleaned[bike_cleaned['day_type'] == 'Weekday']['count']
weekend_rides = bike_cleaned[bike_cleaned['day_type'] == 'Weekend']['count']

# Perform 2-sample independent t-test
t_stat, p_value = stats.ttest_ind(weekday_rides, weekend_rides)

# Output the test statistics and p-value
print(f"T-Statistic: {t_stat}")
print(f"P-Value: {p_value}")

# Conclusion based on p-value
alpha = 0.05
if p_value <= alpha:
    print("Reject the Null Hypothesis: There is a significant difference in the␣
↪number of bike rides on weekdays and weekends.")
else:
    print("Fail to Reject the Null Hypothesis: There is no significant␣
↪difference in the number of bike rides on weekdays and weekends.")
```

```
T-Statistic: 0.3205273662023574
P-Value: 0.7485747441741293
Fail to Reject the Null Hypothesis: There is no significant difference in the
number of bike rides on weekdays and weekends.
```

[ ]:

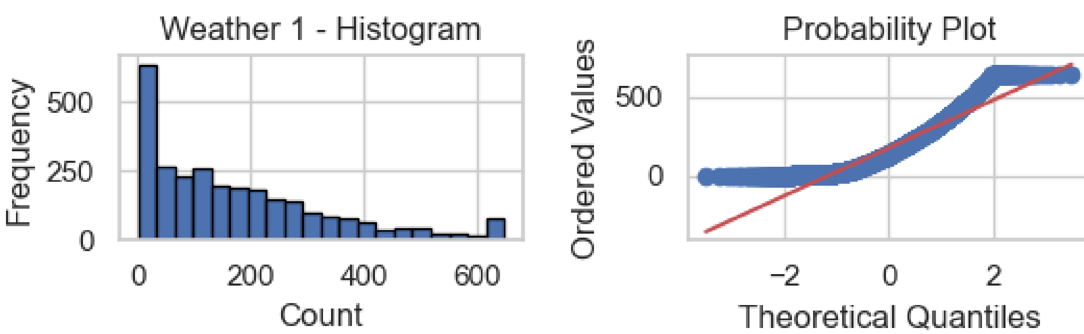
```
""" Check if the demand of bicycles on rent is the same for different Weather␣
↪conditions?
Formulatin H0 and H1:
Null Hypothesis: Number of cycles rented is similar in different weather and␣
↪season.
Alternate Hypothesis: Number of cycles rented is not similar in different␣
↪weather and season. """
```

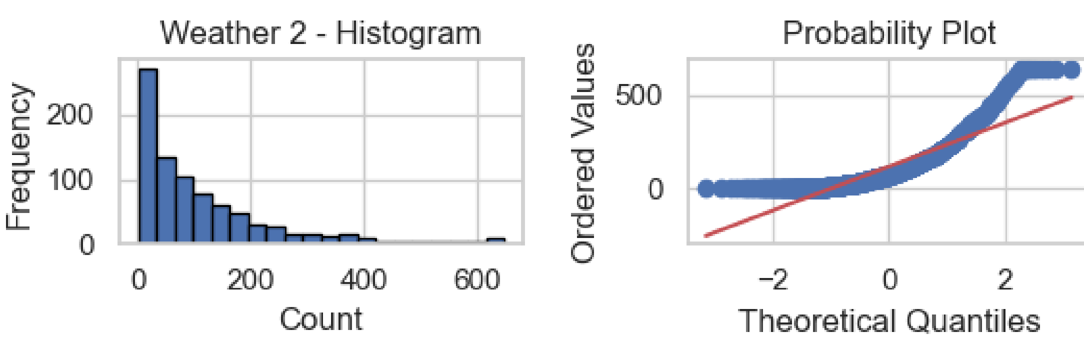
[319]:

```
#using ANOVA TEST
# Defining the data groups for the ANOVA
gp1 = bike[bike['weather'] == 1]['count'].values
gp2 = bike[bike['weather'] == 2]['count'].values
gp3 = bike[bike['weather'] == 3]['count'].values
gp4 = bike[bike['weather'] == 4]['count'].values
gp5 = bike[bike['season'] == 1]['count'].values
gp6 = bike[bike['season'] == 2]['count'].values
gp7 = bike[bike['season'] == 3]['count'].values
gp8 = bike[bike['season'] == 4]['count'].values

groups = [gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8]
group_names = ['Weather 1', 'Weather 2', 'Weather 3', 'Weather 4', 'Season 1',␣
↪'Season 2', 'Season 3', 'Season 4']

# Visual Inspection for Normality Assumption
print("Normality Assumption:")
for i, (group, name) in enumerate(zip(groups, group_names), 1):
    # Plot histogram
    plt.figure(figsize=(6, 2))
    plt.subplot(1, 2, 1)
    plt.hist(group, bins=20, edgecolor='black')
    plt.title(f'{name} - Histogram')
    plt.xlabel('Count')
    plt.ylabel('Frequency')

    # Plot Q-Q plot
    plt.subplot(1, 2, 2)
    plt.title(f'{name} - Q-Q Plot')
    stats.probplot(group, dist="norm", plot=plt)
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Ordered Values')
    plt.tight_layout()
    plt.show()

# Levene's Test for Equality of Variance
print("\nEquality of Variance:")
levene_stat, levene_p = levene(*groups)
```

```
print(f"Levene's Test - p-value = {levene_p:.4f} (Statistic: {levene_stat:.4f})")

# One-Way ANOVA Test
print("\nOne-Way ANOVA Test:")
anova_stat, anova_p = stats.f_oneway(gp1, gp2, gp3, gp4, gp5, gp6, gp7, gp8)
print(f"F-Statistic = {anova_stat}, p-value = {anova_p}")

# Decide to reject or fail to reject the null hypothesis
alpha = 0.05
if anova_p <= alpha:
    print("Reject the Null Hypothesis: There is a significant difference in bike rentals across weather and season conditions.")
else:
    print("Fail to Reject the Null Hypothesis: There is no significant difference in bike rentals across weather and season conditions.")
```

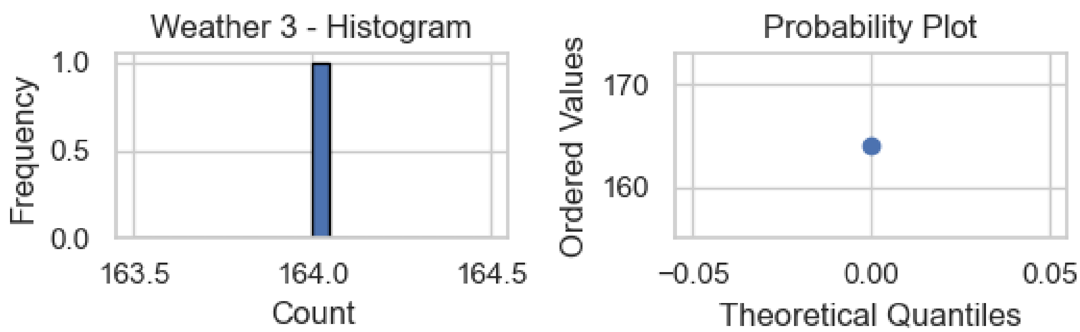
```
Normality Assumption:
```

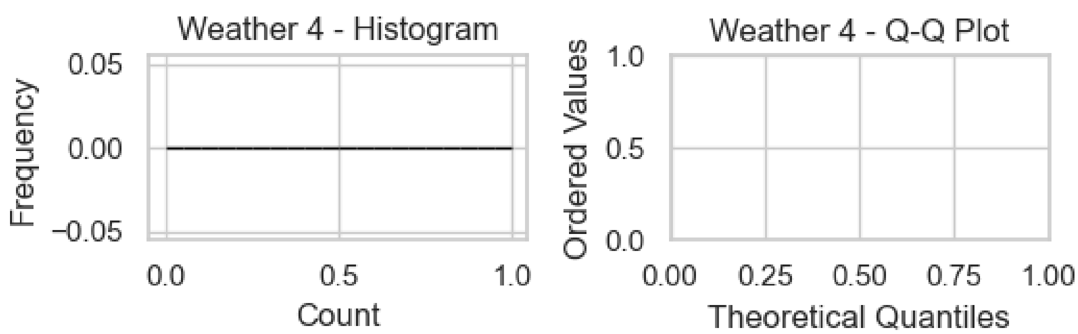


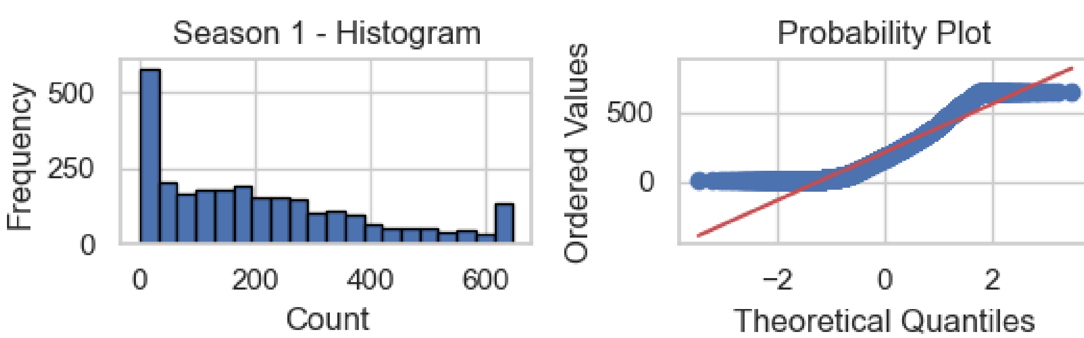


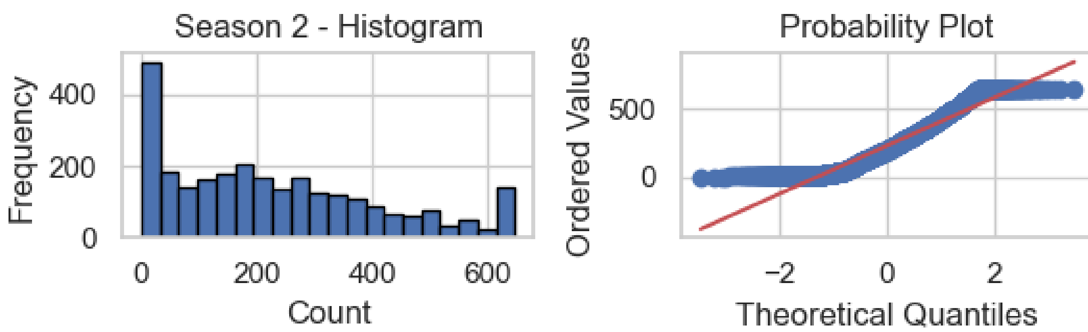

```
C:\Users\samvj\anaconda3\Lib\site-packages\scipy\stats\_stats_mstats_common.py:182: RuntimeWarning: invalid value encountered in scalar divide
```

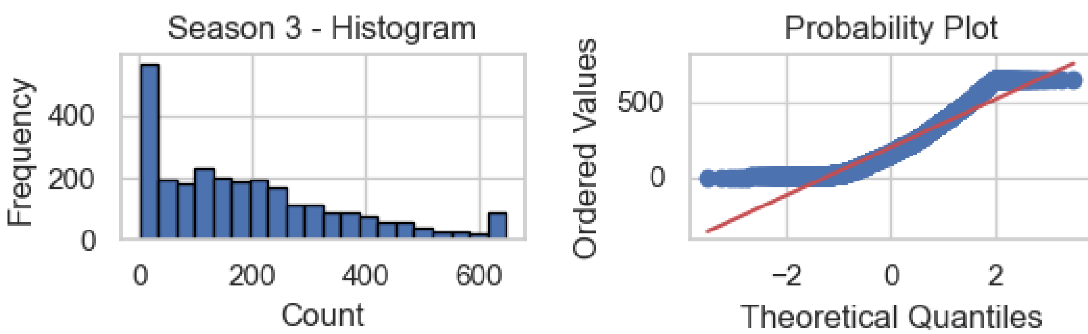
```
  slope = ssxym / ssxm
C:\Users\samvj\anaconda3\Lib\site-
packages\scipy\stats\_stats_mstats_common.py:196: RuntimeWarning: invalid value
encountered in sqrt
  t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY)))
C:\Users\samvj\anaconda3\Lib\site-
packages\scipy\stats\_stats_mstats_common.py:199: RuntimeWarning: invalid value
encountered in scalar divide
  slope_stderr = np.sqrt((1 - r**2) * ssym / ssxm / df)
```

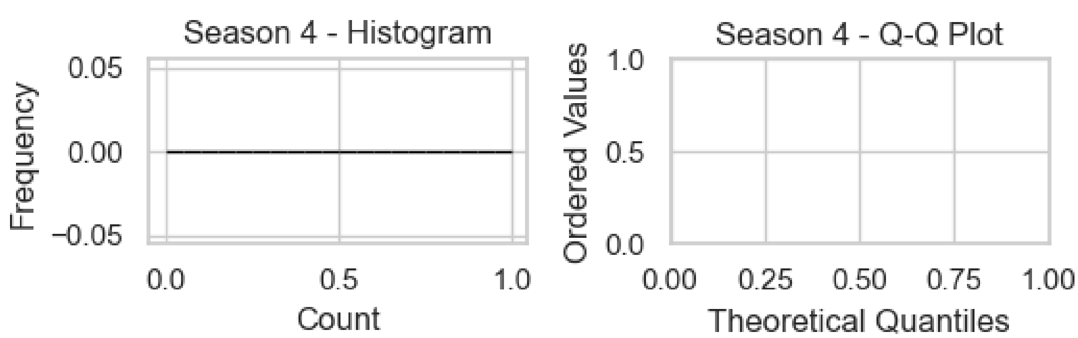













```
Equality of Variance:
Levene's Test - p-value = nan (Statistic: nan)

One-Way ANOVA Test:
F-Statistic = nan, p-value = nan
Fail to Reject the Null Hypothesis: There is no significant difference in bike
rentals across weather and season conditions.
```

```
C:\Users\samvj\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:3504:
RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
C:\Users\samvj\anaconda3\Lib\site-packages\numpy\core\_methods.py:129:
RuntimeWarning: invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
C:\Users\samvj\anaconda3\Lib\site-packages\scipy\stats\_stats_py.py:4133:
DegenerateDataWarning: at least one input has length 0
  warnings.warn(stats.DegenerateDataWarning('at least one input '
```

[ ]:

```
""" SINCE ANOVA returned F-Statistic = nan, p-value = nan checking for the␣
↪vaild data points
since ANOVA returns nan valu only if the data points in null or it is less than␣
↪3 """
```

[320]:

```
# Check the number of valid data points for each group
print("Number of entries in each group:")
for i, group in enumerate(groups, 1):
    print(f"Group {i}: {len(group)}")

# Filter out empty or too small groups
non_empty_groups = [(i, group) for i, group in enumerate(groups, 1) if␣
↪len(group) > 2]  # Keep groups with more than 2 data points
filtered_groups = [group for i, group in non_empty_groups]

# Proceed with the analysis if there are enough non-empty groups
if len(filtered_groups) > 1:
    # Levene's Test for Equality of Variance
    levene_stat, levene_p = levene(*filtered_groups)
    print(f"\nLevene's Test - p-value = {levene_p:.4f} (Statistic: {levene_stat:
↪.4f})")

    # One-Way ANOVA Test
    anova_stat, anova_p = stats.f_oneway(*filtered_groups)
    print(f"\nOne-Way ANOVA Test:")
    print(f"F-Statistic = {anova_stat}, p-value = {anova_p}")

    # Hypothesis testing
    alpha = 0.05
    if anova_p <= alpha:
        print("Reject the Null Hypothesis: There is a significant difference in␣
↪bike rentals across weather and season conditions.")
    else:
        print("Fail to Reject the Null Hypothesis: There is no significant␣
↪difference in bike rentals across weather and season conditions.")
else:
    print("Not enough data in the groups to perform the analysis.")
```

```
Number of entries in each group:
Group 1: 2834
Group 2: 859
Group 3: 1
Group 4: 0
Group 5: 2733
Group 6: 2733
Group 7: 2732
Group 8: 0

Levene's Test - p-value = 0.0000 (Statistic: 59.9627)

One-Way ANOVA Test:
F-Statistic = 84.78103892596496, p-value = 4.081530842574612e-71
Reject the Null Hypothesis: There is a significant difference in bike rentals
across weather and season conditions.
```

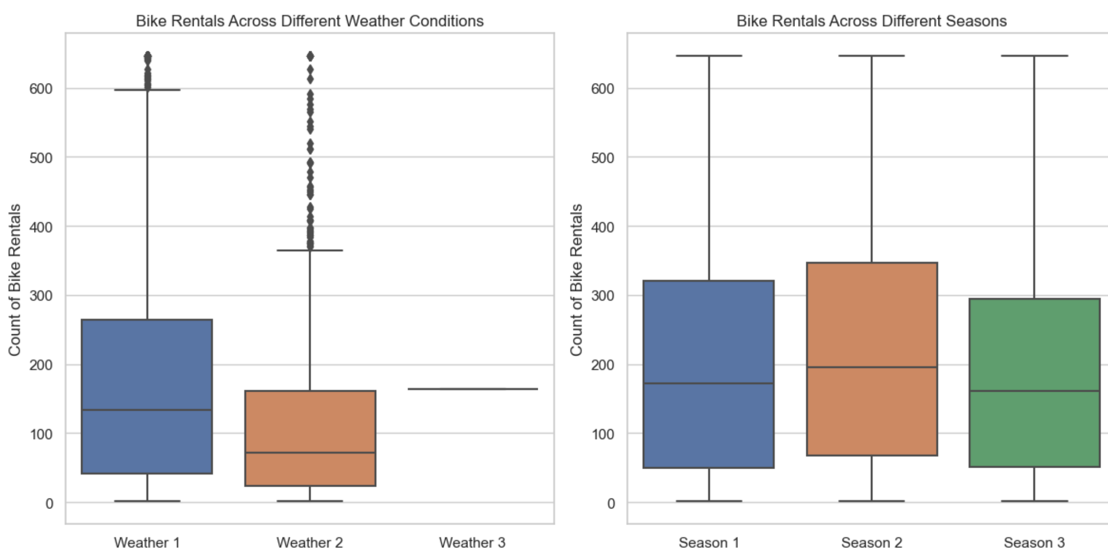
[ ]:

```
'''since With the presence of very small groups (like weather condition 4 and␣
↪season 4 having no data, and weather 3 with only 1 data point ),
the analysis might not be fully reliable then we are exploring the DATA using␣
↪non-parametric methods  the Kruskal-Wallis Test BY
REMOVING THE (weather condition 4 and season 4 having) '''
```

[321]:

```
# Grouping the data based on weather and season
gp1 = bike[bike['weather'] == 1]['count'].dropna().values
gp2 = bike[bike['weather'] == 2]['count'].dropna().values
gp3 = bike[bike['weather'] == 3]['count'].dropna().values
gp5 = bike[bike['season'] == 1]['count'].dropna().values
gp6 = bike[bike['season'] == 2]['count'].dropna().values
gp7 = bike[bike['season'] == 3]['count'].dropna().values

# Grouping for boxplot visualization
weather_groups = [gp1, gp2, gp3]
season_groups = [gp5, gp6, gp7]
group_labels_weather = ['Weather 1', 'Weather 2', 'Weather 3']
group_labels_season = ['Season 1', 'Season 2', 'Season 3']

# Plotting boxplots for visualization
plt.figure(figsize=(12, 6))

# Boxplot for Weather
plt.subplot(1, 2, 1)
sns.boxplot(data=weather_groups)
plt.title('Bike Rentals Across Different Weather Conditions')
plt.xticks([0, 1, 2], group_labels_weather)
plt.ylabel('Count of Bike Rentals')
```

```
# Boxplot for Season
plt.subplot(1, 2, 2)
sns.boxplot(data=season_groups)
plt.title('Bike Rentals Across Different Seasons')
plt.xticks([0, 1, 2], group_labels_season)
plt.ylabel('Count of Bike Rentals')

plt.tight_layout()
plt.show()

# Performing the Kruskal-Wallis H-test
stat, p_value = kruskal(gp1, gp2, gp3, gp5, gp6, gp7)

# Output the result of the Kruskal-Wallis test
print(f"Kruskal-Wallis Test: H-statistic = {stat}, p-value = {p_value}")

# Decision on Null Hypothesis
if p_value < 0.05:
    print("Reject the Null Hypothesis: There is a significant difference in␣
↪bike rentals across weather and season conditions.")
else:
    print("Fail to Reject the Null Hypothesis: There is no significant␣
↪difference in bike rentals across weather and season conditions.")
```



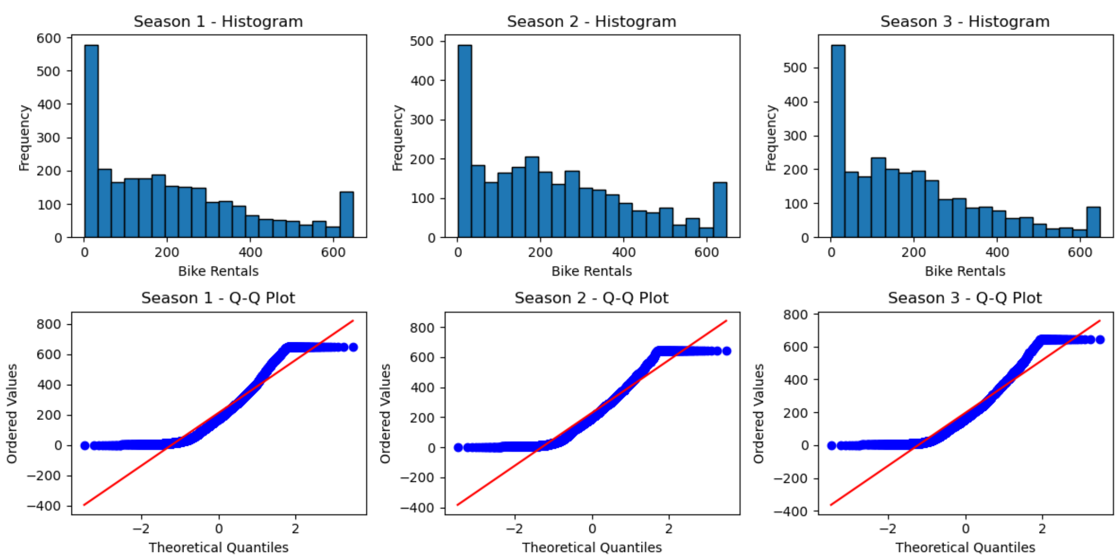

```
Kruskal-Wallis Test: H-statistic = 340.96799224838907, p-value =
1.5397348915133392e-71
Reject the Null Hypothesis: There is a significant difference in bike rentals
across weather and season conditions.
```

[ ]:
```
""" Check if the demand of bicycles on rent is the same for different Seasons?
 Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1):
 Null Hypothesis (H0): The demand for bicycles on rent is the same for all␣
 ↪seasons.
 Alternate Hypothesis (H1): There is no significant difference in bike rentals␣
 ↪across seasons """
```

[34]:
```
#dropinf group 4 sinnce it has zero values
from scipy.stats import shapiro, levene, f_oneway, skew, kurtosis, probplot

# Group the data based on seasons using updated groups (gp5, gp6, gp7)
gp5 = bike[bike['season'] == 1]['count'].dropna().values
gp6 = bike[bike['season'] == 2]['count'].dropna().values
gp7 = bike[bike['season'] == 3]['count'].dropna().values

# Updated group labels
group_labels = ['Season 1', 'Season 2', 'Season 3']

# Normality Check
print("Normality Check (Shapiro-Wilk's Test):")
for i, group in enumerate([gp5, gp6, gp7], start=1):
    stat, p_value = shapiro(group)
    print(f"Shapiro-Wilk Test for {group_labels[i-1]}: Statistic = {stat},␣
 ↪p-value = {p_value}")

# Skewness and Kurtosis for Normality Check
print("\nSkewness and Kurtosis for each group:")
for i, group in enumerate([gp5, gp6, gp7], start=1):
    print(f"{group_labels[i-1]} - Skewness: {skew(group)}, Kurtosis:␣
 ↪{kurtosis(group)}")

# Plotting Histograms and Q-Q Plots
plt.figure(figsize=(12, 6))
for i, group in enumerate([gp5, gp6, gp7], start=1):
    plt.subplot(2, 3, i)
    plt.hist(group, bins=20, edgecolor='black')
    plt.title(f'{group_labels[i-1]} - Histogram')
    plt.xlabel('Bike Rentals')
    plt.ylabel('Frequency')

    plt.subplot(2, 3, i+3)
    probplot(group, dist="norm", plot=plt)
    plt.title(f'{group_labels[i-1]} - Q-Q Plot')
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Ordered Values')

plt.tight_layout()
```

```
plt.show()

# Equality of Variance Check (Levene's Test)
levene_stat, levene_p = levene(gp5, gp6, gp7)
print(f"\nLevene's Test - p-value = {levene_p} (Statistic: {levene_stat})")

# One-Way ANOVA Test
anova_stat, anova_p = f_oneway(gp5, gp6, gp7)
print(f"\nOne-Way ANOVA Test: F-Statistic = {anova_stat}, p-value = {anova_p}")

# Conclusion
alpha = 0.05
if anova_p <= alpha:
    print("\nReject the Null Hypothesis: The demand for bicycles on rent is the␣
↪same for all seasons.")
else:
    print("\nFail to Reject the Null Hypothesis: There is no significant␣
↪difference in bike rentals across seasons.")
```

```
Normality Check (Shapiro-Wilk's Test):
Shapiro-Wilk Test for Season 1: Statistic = 0.9028080701828003, p-value =
1.3357483846307843e-38
Shapiro-Wilk Test for Season 2: Statistic = 0.9245984554290771, p-value =
5.309610981421456e-35
Shapiro-Wilk Test for Season 3: Statistic = 0.9059131145477295, p-value =
3.9998195332000494e-38

Skewness and Kurtosis for each group:
Season 1 - Skewness: 0.8191111657304861, Kurtosis: -0.25377318876410904
Season 2 - Skewness: 0.6634317796015348, Kurtosis: -0.46040025451785516
Season 3 - Skewness: 0.9126897208704497, Kurtosis: 0.11266395728126044
```



```
Levene's Test - p-value = 1.0526201242651206e-07 (Statistic: 16.098354449282194)

One-Way ANOVA Test: F-Statistic = 22.958770997126802, p-value =
1.1401513452125809e-10

Reject the Null Hypothesis: There is a significant difference in bike rentals
across seasons.
```

[ ]:
```
""" Check if the Weather conditions are signi cantly different during different␣
 ↪Seasons?
 Formulating H0 and H1:
 Null Hypothesis (H0): There is a significant relationship between weather␣
 ↪conditions and seasons
 Alternate Hypothesis (H1): There is no significant relationship between␣
 ↪weather conditions and seasons"""
```

[325]:
```
import pandas as pd
from scipy.stats import chi2_contingency

# Create a contingency table (cross-tabulation)
contingency_table = pd.crosstab(bike['weather'], bike['season'])

# Display the contingency table
print("Contingency Table:")
print(contingency_table)

# Perform the Chi-square test for independence
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Print the test statistic, p-value, degrees of freedom, and expected␣
 ↪frequencies
print(f"\nChi-square Statistic: {chi2_stat}")
print(f"p-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies Table:")
print(expected)

# Set the significance level
alpha = 0.05

# Decision on hypothesis test
if p_value <= alpha:
    print("\nReject the Null Hypothesis: There is a significant relationship␣
 ↪between weather conditions and seasons.")
```

```
else:
    print("\nFail to Reject the Null Hypothesis: There is no significant␣
↪relationship between weather conditions and seasons.")
```

```
Contingency Table:
season      0     1     2     3
weather
0        1757  1801  1930  1700
1         715   708   604   807
2         211   224   199   225
3           1     0     0     0

Chi-square Statistic: 49.431915478249664
p-value: 1.3773305739577996e-07
Degrees of Freedom: 9
Expected Frequencies Table:
[[1.77289028e+03 1.80525675e+03 1.80525675e+03 1.80459621e+03]
 [6.98994303e+02 7.11755376e+02 7.11755376e+02 7.11494946e+02]
 [2.11868774e+02 2.15736721e+02 2.15736721e+02 2.15657783e+02]
 [2.46645837e-01 2.51148686e-01 2.51148686e-01 2.51056791e-01]]

Reject the Null Hypothesis: There is a significant relationship between weather
conditions and seasons.
```

[ ]:
```
"""
Insights:
Seasonal Trends: Bike rentals are higher during summer and fall seasons␣
↪compared to other seasons.
Impact of Holidays: Bike rentals increase significantly on holidays.
Working Day vs. Holidays: Holidays and weekends see slightly higher bike␣
↪rentals compared to working days.
Weather Conditions: Adverse weather conditions such as rain, thunderstorms,␣
↪snow, or fog result in fewer bike rentals.
Effect of Low Humidity: When humidity drops below 20%, bike rentals are very␣
↪low.
Cold Temperature Impact: Temperatures below 10°C correspond to fewer bike␣
↪rentals.
High Wind Speed: Wind speeds above 35 km/h lead to reduced bike rentals.

Recommendations:
Seasonal Stock Management: Increase bike availability during summer and fall to␣
↪meet higher demand.
Holiday Planning: Ensure a higher stock of bikes on holidays to cater to␣
↪increased rentals.
Low-Humidity Days: Reduce bike stock on days with very low humidity (below 20%).
Cold Days Preparation: Maintain a lower stock of bikes during very cold days␣
↪(temperatures below 10°C).
```

```
High Wind and Storms: On days with wind speeds above 35 km/h or during storms, reduce the number of bikes available for rent.
"""
```