```
! pip install pandas
```

```
Requirement already satisfied: pandas in e:\rasa\lib\site-packages
(2.1.4)
Requirement already satisfied: numpy<2,>=1.23.2 in e:\rasa\lib\site-
packages (from pandas) (1.24.3)
Requirement already satisfied: python-dateutil>=2.8.2 in e:\rasa\lib\
site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in e:\rasa\lib\site-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in e:\rasa\lib\site-
packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in e:\rasa\lib\site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
```

BUSINESS CASE: YULU-HYPOTHESIS TESTING

IMPORTING DATASET AND OBSERVING DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#IMPORTING DATASET
data=pd.read_csv('G:/dsml-scaler/probability and
stats/casestudy/bike_sharing.txt')

data.head()
```

```
              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
```

```
data.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
```

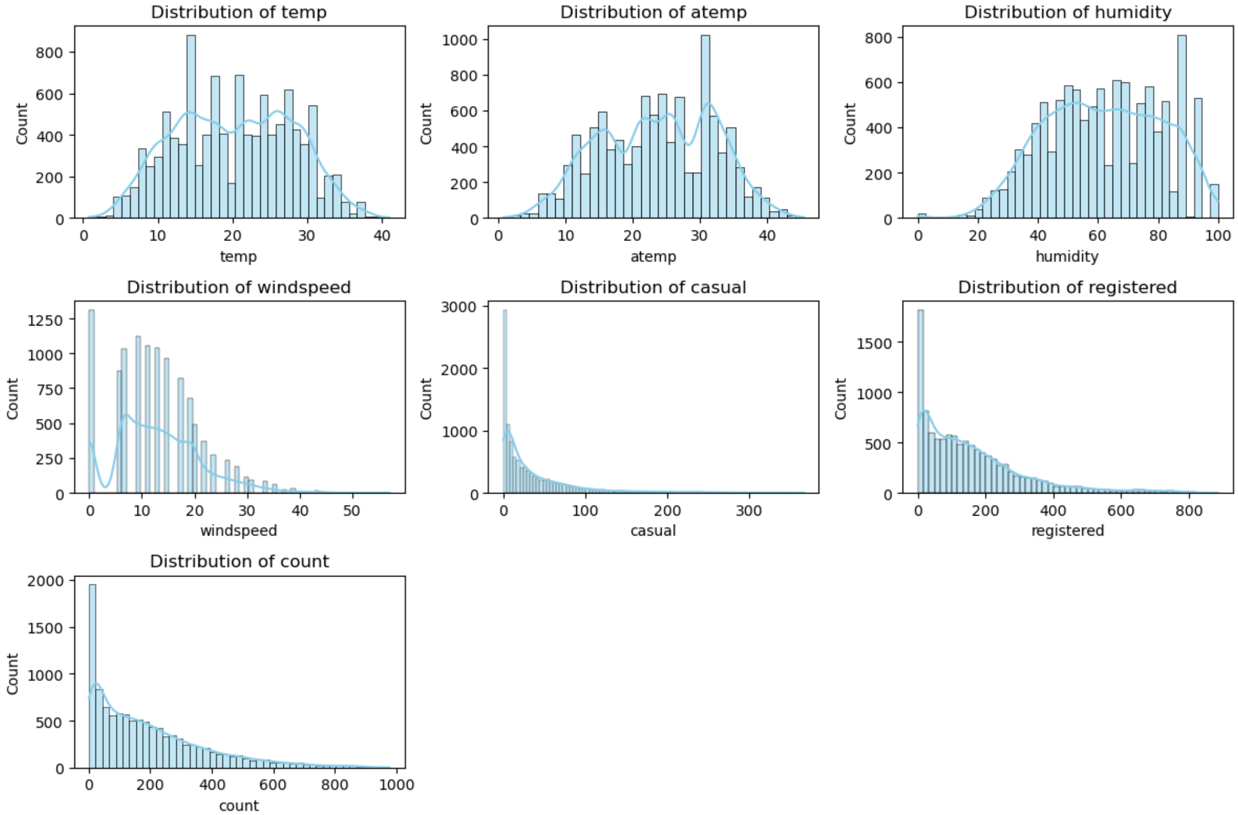
```
# 1. Descriptive Statistics
numerical_variables = ['temp', 'atemp', 'humidity', 'windspeed', 
'casual', 'registered', 'count']
print(data[numerical_variables].describe())
```

```
              temp         atemp      humidity     windspeed
casual  \
count  10886.00000  10886.000000  10886.000000  10886.000000
10886.000000   
mean      20.23086     23.655084     61.886460     12.799395
36.021955   
std        7.79159      8.474601     19.245033      8.164537
49.960477   
min        0.82000      0.760000      0.000000      0.000000
0.000000   
25%       13.94000     16.665000     47.000000      7.001500
4.000000   
50%       20.50000     24.240000     62.000000     12.998000
17.000000   
75%       26.24000     31.060000     77.000000     16.997900
49.000000   
max       41.00000     45.455000    100.000000     56.996900
367.000000   

         registered         count  
count  10886.000000  10886.000000  
mean     155.552177    191.574132  
std      151.039033    181.144454  
```

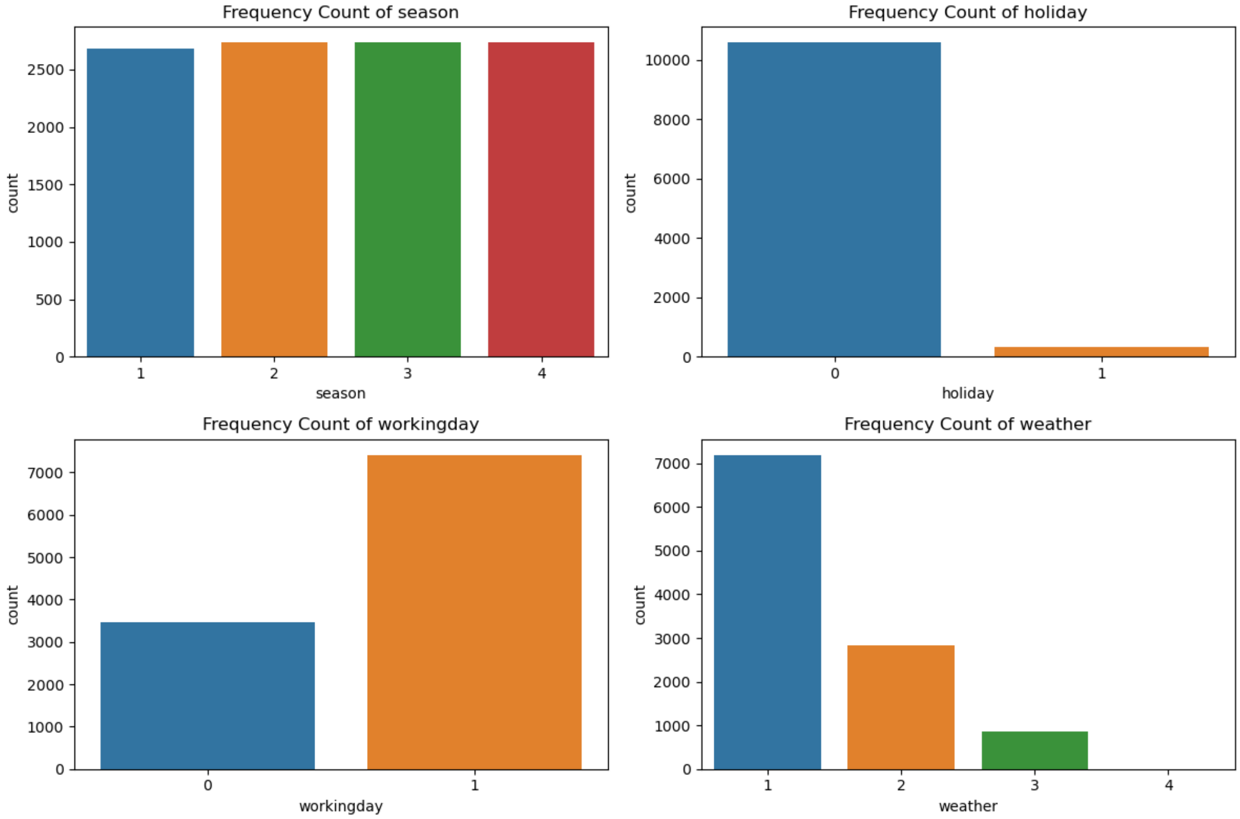
```
min        0.000000      1.000000
25%       36.000000     42.000000
50%      118.000000    145.000000
75%      222.000000    284.000000
max      886.000000    977.000000
```

```python
# 2. Data Distribution
plt.figure(figsize=(12, 8))
for i, col in enumerate(numerical_variables):
    plt.subplot(3, 3, i+1)
    sns.histplot(data[col], kde=True, color='skyblue')
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()
```

```
E:\rasa\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):
E:\rasa\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):
E:\rasa\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):
E:\rasa\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):
E:\rasa\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):
E:\rasa\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):
E:\rasa\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):
```




```python
# 3. Frequency Counts
categorical_variables = ['season', 'holiday', 'workingday', 'weather']
plt.figure(figsize=(12, 8))
for i, col in enumerate(categorical_variables):
    plt.subplot(2, 2, i+1)
    sns.countplot(data=data, x=col)
    plt.title(f'Frequency Count of {col}')
plt.tight_layout()
plt.show()
```




```
print("The shape of dataset is:",data.shape)
```

```
The shape of dataset is: (10886, 12)
```

```
print("The datatype of attributes are:")
print(data.dtypes)
```

```
The datatype of attributes are:
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
humidity        int64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

```
# Convert 'datetime' column to datetime format
data['datetime'] = pd.to_datetime(data['datetime'])
```

```python
# Verify the data type conversion
print(data.info())
```

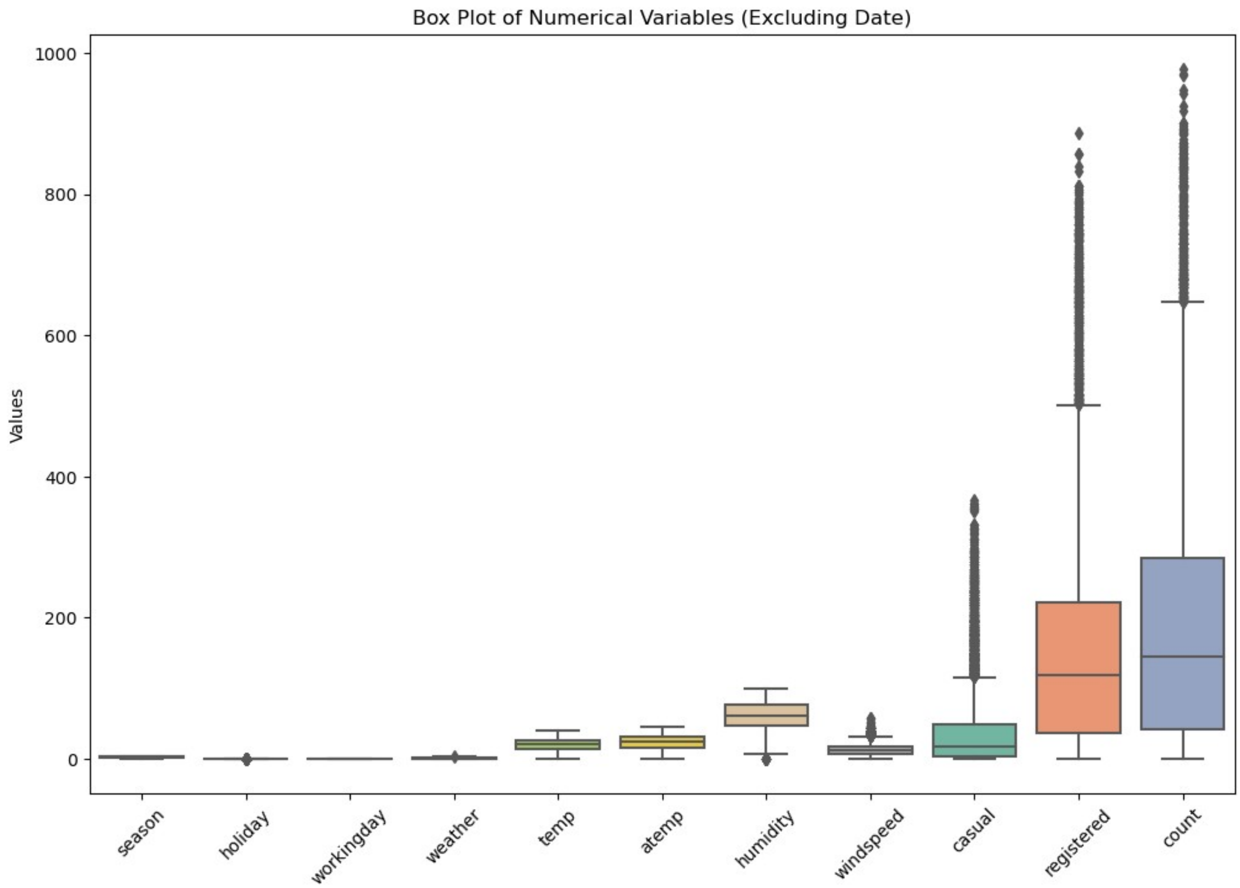
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  int64
 2   holiday     10886 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: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
None
```

```python
# checking the null values
data.isna().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
```

```python
# checking for outliers
# Select numerical variables excluding the 'datetime' column
numerical_variables = ['season', 'holiday', 'workingday', 'weather',
'temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered',
'count']

# Plot box plots for selected numerical variables
plt.figure(figsize=(12, 8))
sns.boxplot(data=data[numerical_variables], orient='v',
```

```
palette='Set2')
plt.title('Box Plot of Numerical Variables (Excluding Date)')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show()
```




```
q1= data.windspeed.quantile(0.25)
q3 = data.windspeed.quantile(0.75)
iqr = q3-q1
outliers = data[((data.windspeed < (q1-1.5*iqr)) | ((data.windspeed >
(q3+1.5*iqr))))]
print("number of outliers",len(outliers))
print("percentage of outliers", len(outliers)/data.shape[0])
```

```
number of outliers 227
percentage of outliers 0.02085247106375161
```

```
q1= data.casual.quantile(0.25)
q3 = data.casual.quantile(0.75)
iqr = q3-q1
outliers = data[((data.casual < (q1-1.5*iqr)) | ((data.casual >
(q3+1.5*iqr))))]
```

```
print("number of outliers",len(outliers))
print("percentage of outliers", len(outliers)/data.shape[0])
```

```
number of outliers 749
percentage of outliers 0.06880396839977954
```

```
q1= data.registered.quantile(0.25)
q3 = data.registered.quantile(0.75)
iqr = q3-q1
outliers = data[((data.registered < (q1-1.5*iqr)) | ((data.registered
> (q3+1.5*iqr))))]
print("number of outliers",len(outliers))
print("percentage of outliers", len(outliers)/data.shape[0])
```

```
number of outliers 423
percentage of outliers 0.03885724784126401
```

```
# Define the column name
column_name = 'count'

# Calculate quartiles and IQR
q1 = data[column_name].quantile(0.25)
q3 = data[column_name].quantile(0.75)
iqr = q3 - q1

# Determine outliers
outliers = data[(data[column_name] < (q1 - 1.5 * iqr)) |
(data[column_name] > (q3 + 1.5 * iqr))]

# Calculate the number of outliers
num_outliers = len(outliers)

# Calculate the percentage of outliers
percentage_outliers = (num_outliers / data.shape[0]) * 100

# Print the results
print("Number of outliers:", num_outliers)
print("Percentage of outliers:", percentage_outliers)
```

```
Number of outliers: 300
Percentage of outliers: 2.75583318023149
```

```
# Handling outliers using imputation method
# Define the columns with outliers
columns_with_outliers = ['windspeed', 'casual', 'registered', 'count']

# Define a function to impute outliers
def impute_outliers(df, column):
    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

    # Replace outliers with median value
    df[column] = np.where((df[column] < lower_bound) | (df[column] >
upper_bound),
                          df[column].median(), df[column])
    return df

# Impute outliers for each column
for column in columns_with_outliers:
    df = impute_outliers(data, column)

# Verify that outliers have been imputed
print("Number of outliers after imputation:")
for column in columns_with_outliers:
    num_outliers = ((df[column] < df[column].quantile(0.25) - 1.5 *
(df[column].quantile(0.75) - df[column].quantile(0.25))) |
                    (df[column] > df[column].quantile(0.75) + 1.5 *
(df[column].quantile(0.75) - df[column].quantile(0.25)))).sum()
    print(f"{column}: {num_outliers}")
```

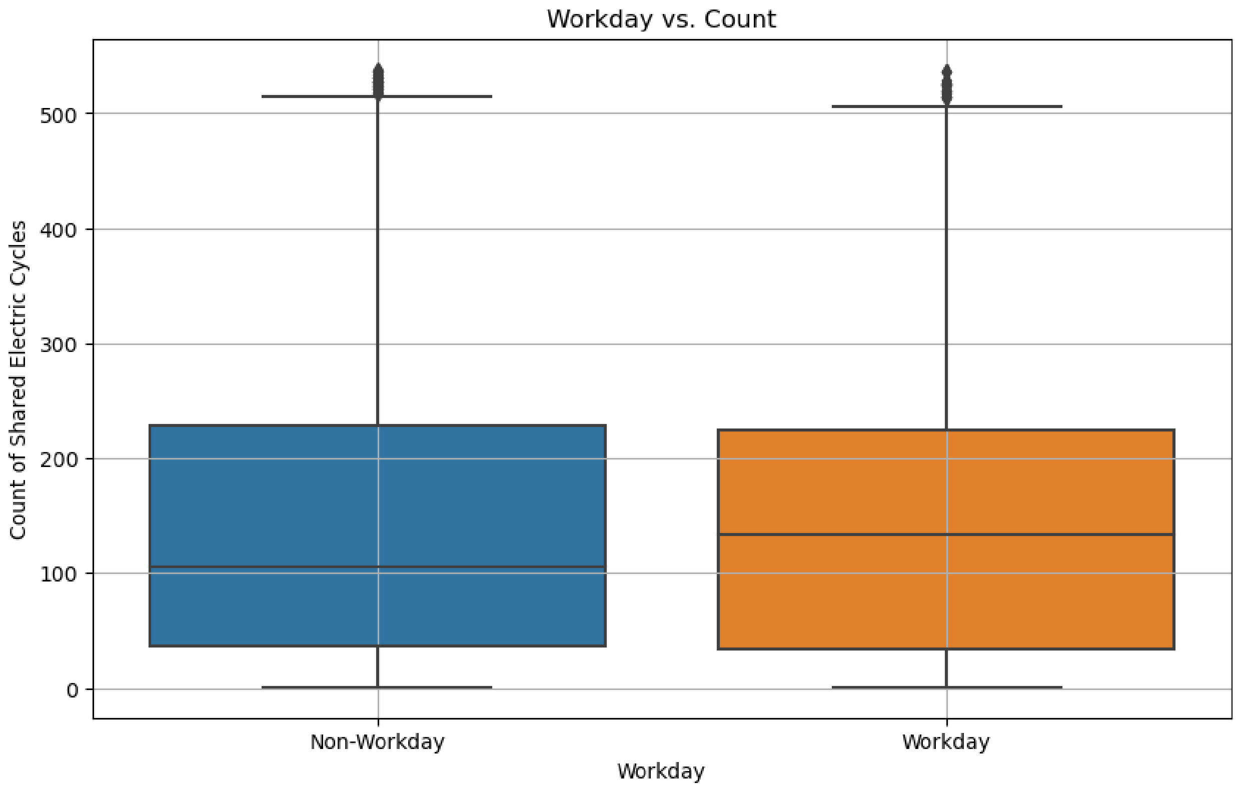
```
Number of outliers after imputation:
windspeed: 0
casual: 557
registered: 161
count: 114
```

```
# Define the columns with remaining outliers
columns_with_remaining_outliers = ['casual', 'registered', 'count']

# Remove remaining outliers for each column
for column in columns_with_remaining_outliers:
    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

    # Filter out rows with outliers
    df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Print the shape of the dataset after outlier removal
print("Shape of the dataset after outlier removal:", df.shape)
```

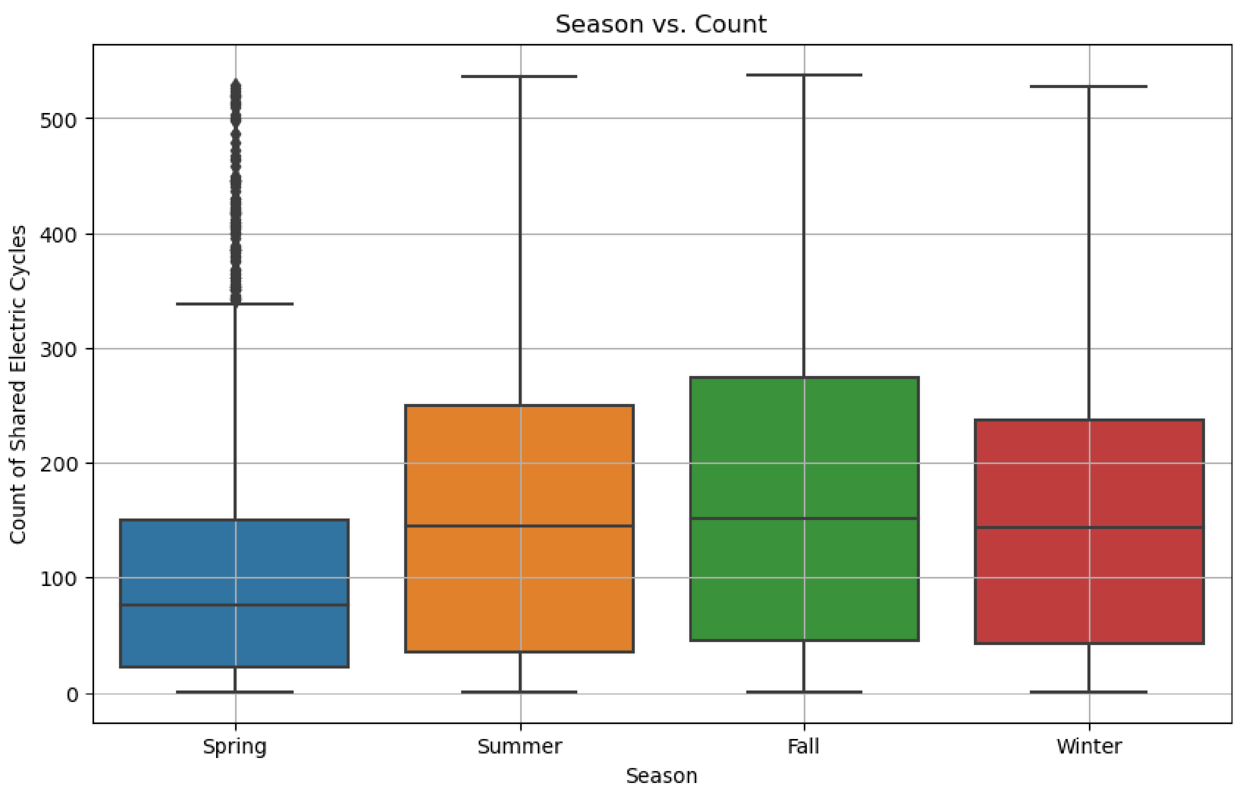
```
Shape of the dataset after outlier removal: (9866, 12)
```

Bivariate Analysis

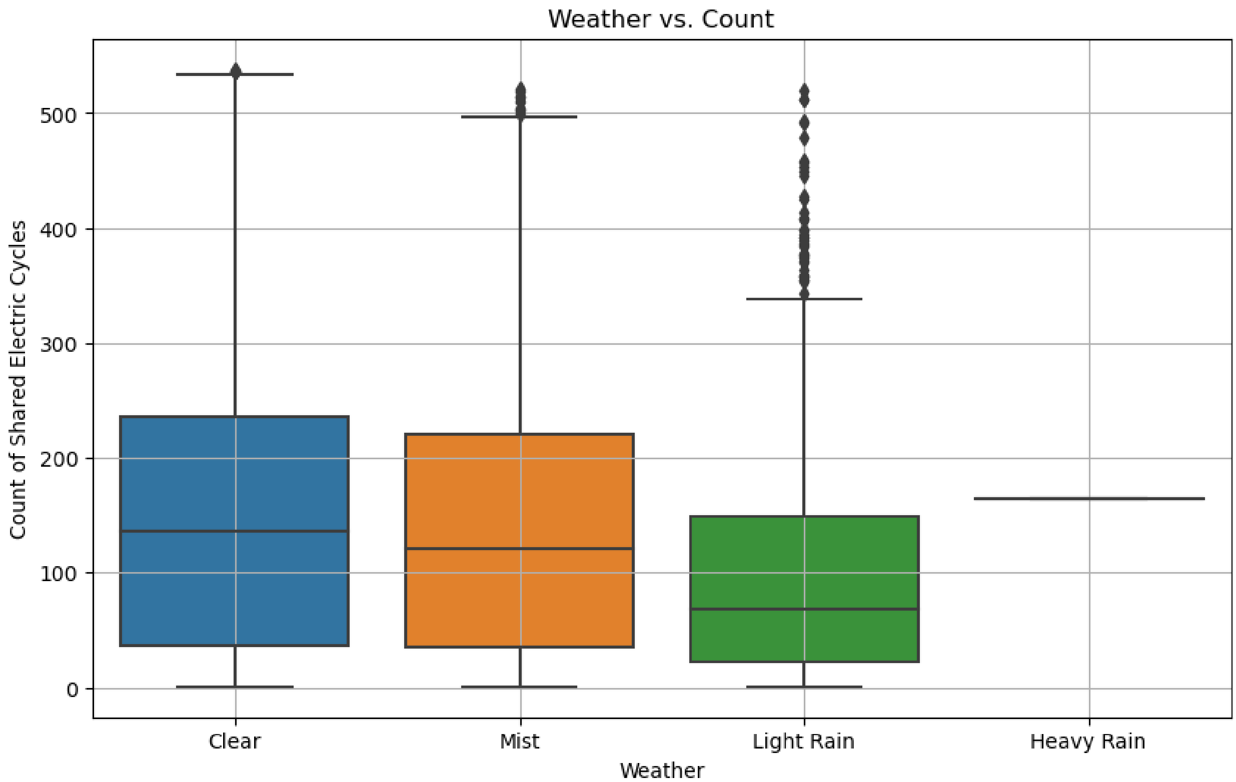
```python
# Plotting Workday vs. Count
plt.figure(figsize=(10, 6))
sns.boxplot(x='workingday', y='count', data=df)
plt.title('Workday vs. Count')
plt.xlabel('Workday')
plt.ylabel('Count of Shared Electric Cycles')
plt.xticks([0, 1], ['Non-Workday', 'Workday'])
plt.grid(True)
plt.show()

# Plotting Season vs. Count
plt.figure(figsize=(10, 6))
sns.boxplot(x='season', y='count', data=df)
plt.title('Season vs. Count')
plt.xlabel('Season')
plt.ylabel('Count of Shared Electric Cycles')
plt.xticks([0, 1, 2, 3], ['Spring', 'Summer', 'Fall', 'Winter'])
plt.grid(True)
plt.show()

# Plotting Weather vs. Count
plt.figure(figsize=(10, 6))
sns.boxplot(x='weather', y='count', data=df)
plt.title('Weather vs. Count')
plt.xlabel('Weather')
plt.ylabel('Count of Shared Electric Cycles')
plt.xticks([0, 1, 2, 3], ['Clear', 'Mist', 'Light Rain', 'Heavy
Rain'])
plt.grid(True)
plt.show()
```



Workday vs. Count
Count of Shared Electric Cycles
0
100
200
300
400
500
Non-Workday
Workday
Workday



Season vs. Count
Count of Shared Electric Cycles
0
100
200
300
400
500
Spring
Summer
Fall
Winter
Season




```python
# using the group by functionality to know the excat numbers the above
visuals represent
# Grouping by workingday and calculating the count
count_by_workday = df.groupby('workingday')['count'].sum()
print("Count of shared electric cycles by workingday:\n",
count_by_workday)

# Grouping by season and calculating the count
count_by_season = df.groupby('season')['count'].sum()
print("\nCount of shared electric cycles by season:\n",
count_by_season)

# Grouping by weather and calculating the count
count_by_weather = df.groupby('weather')['count'].sum()
print("Count of shared electric cycles by weather:\n",
count_by_weather)
```

```
Count of shared electric cycles by workingday:
 workingday
0    472387.0
1    995166.0
Name: count, dtype: float64

Count of shared electric cycles by season:
 season
1    269414.0
```

```
2    388265.0
3    410093.0
4    399781.0
Name: count, dtype: float64
Count of shared electric cycles by weather:
 weather
1    1002070.0
2     379216.0
3      86103.0
4        164.0
Name: count, dtype: float64
```

HYPOTHESIS TESTING

2-SAMPLE T-TEST

```
from scipy.stats import ttest_ind

# Subset data for working day and non-working day
working_day_data = df[df['workingday'] == 1]['count']
non_working_day_data = df[df['workingday'] == 0]['count']

# Perform 2-sample T-test
t_statistic, p_value = ttest_ind(working_day_data,
non_working_day_data)

# Print results
print("T-statistic:", t_statistic)
print("P-value:", p_value)

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant
difference in the number of electric cycles rented between working
days and non-working days.")
else:
    print("Fail to reject the null hypothesis. There is no significant
difference in the number of electric cycles rented between working
days and non-working days.")
```

```
T-statistic: -2.124770091239619
P-value: 0.033630595775153704
Reject the null hypothesis. There is a significant difference in the
number of electric cycles rented between working days and non-working
days.
```

```
from scipy.stats import f_oneway

# Perform ANOVA test
```

```
weather_groups = [df[df['weather'] == i]['count'] for i in range(1,
5)]
f_statistic, p_value = f_oneway(*weather_groups)

# Print results
print("F-statistic:", f_statistic)
print("P-value:", p_value)

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant
difference in the number of cycles rented across different weather
conditions.")
else:
    print("Fail to reject the null hypothesis. There is no significant
difference in the number of cycles rented across different weather
conditions.")

F-statistic: 43.97308431171039
P-value: 3.209921847226834e-28
Reject the null hypothesis. There is a significant difference in the
number of cycles rented across different weather conditions.

from scipy.stats import chi2_contingency

# Create contingency table
weather_season_table = pd.crosstab(df['weather'], df['season'])

# Perform Chi-square test
chi2_statistic, p_value, dof, expected =
chi2_contingency(weather_season_table)

# Print results
print("Chi-square statistic:", chi2_statistic)
print("P-value:", p_value)

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. Weather and season are
dependent on each other.")
else:
    print("Fail to reject the null hypothesis. Weather and season are
independent of each other.")

Chi-square statistic: 43.69751021011294
P-value: 1.6046004443637115e-06
Reject the null hypothesis. Weather and season are dependent on each
other.
```