





```
import pandas as pd
df = pd.read_csv("/bin/data/bike_sharing.csv")
df
```



	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	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88



10886 rows × 12 columns

Next steps:

Generate code with df

 View recommended plots

```
# Examine dataset structure
print("Dataset shape:", df.shape)
print("\nDataset information:")

print(df.info())
print("\n\nStatistical summary:")
print(df.describe())
```

```
Dataset shape: (10886, 12)

Dataset information:
<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
None
```

```
Statistical summary:
      season      holiday      workingday      weather      temp \
count 10886.000000 10886.000000 10886.000000 10886.000000 10886.000000
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

```
# Identify missing values
missing_values = df.isna().sum()
print("\nMissing values:")
print(missing_values)

# Perform imputation (if necessary)
# Example:
# If there are missing values in a column 'column_name', you can impute them with the mean:
# data['column_name'].fillna(data['column_name'].mean(), inplace=True)
```

```
Missing values:
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
```

Insights

Seems there are no null values

```
# Identify duplicate records
duplicates = df.duplicated().sum()
print("\nDuplicate records:", duplicates)
```

```
# Remove duplicates
df.drop_duplicates(inplace=True)
```

```
Duplicate records: 0
```

**Insights ** Seems there are no duplicate records as well

```

import matplotlib.pyplot as plt
import seaborn as sns

# Separate numerical and categorical features
numerical_features = []
categorical_features = []

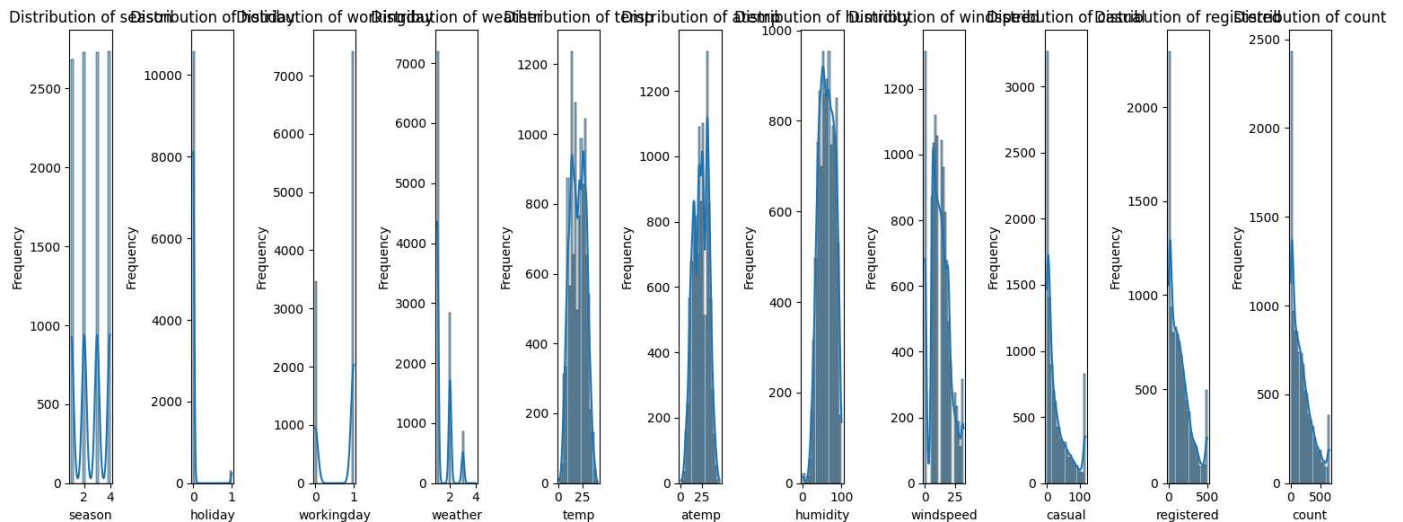
for column in data.columns:
    if data[column].dtype in ['int64', 'float64']:
        numerical_features.append(column)
    else:
        categorical_features.append(column)

# Set up the figure and axes for numerical features
fig, axes = plt.subplots(nrows=1, ncols=len(numerical_features), figsize=(15, 6))

# Plot histograms for numerical features
for i, feature in enumerate(numerical_features):
    sns.histplot(data[feature], bins=20, kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature}')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Frequency')

plt.tight_layout()
plt.show()

```



Insights : We can see that, temp, humidity, are distributed normally, while windspeed, casual, registered are left-skewed, we can make them, normally dist, with log 10 distribution

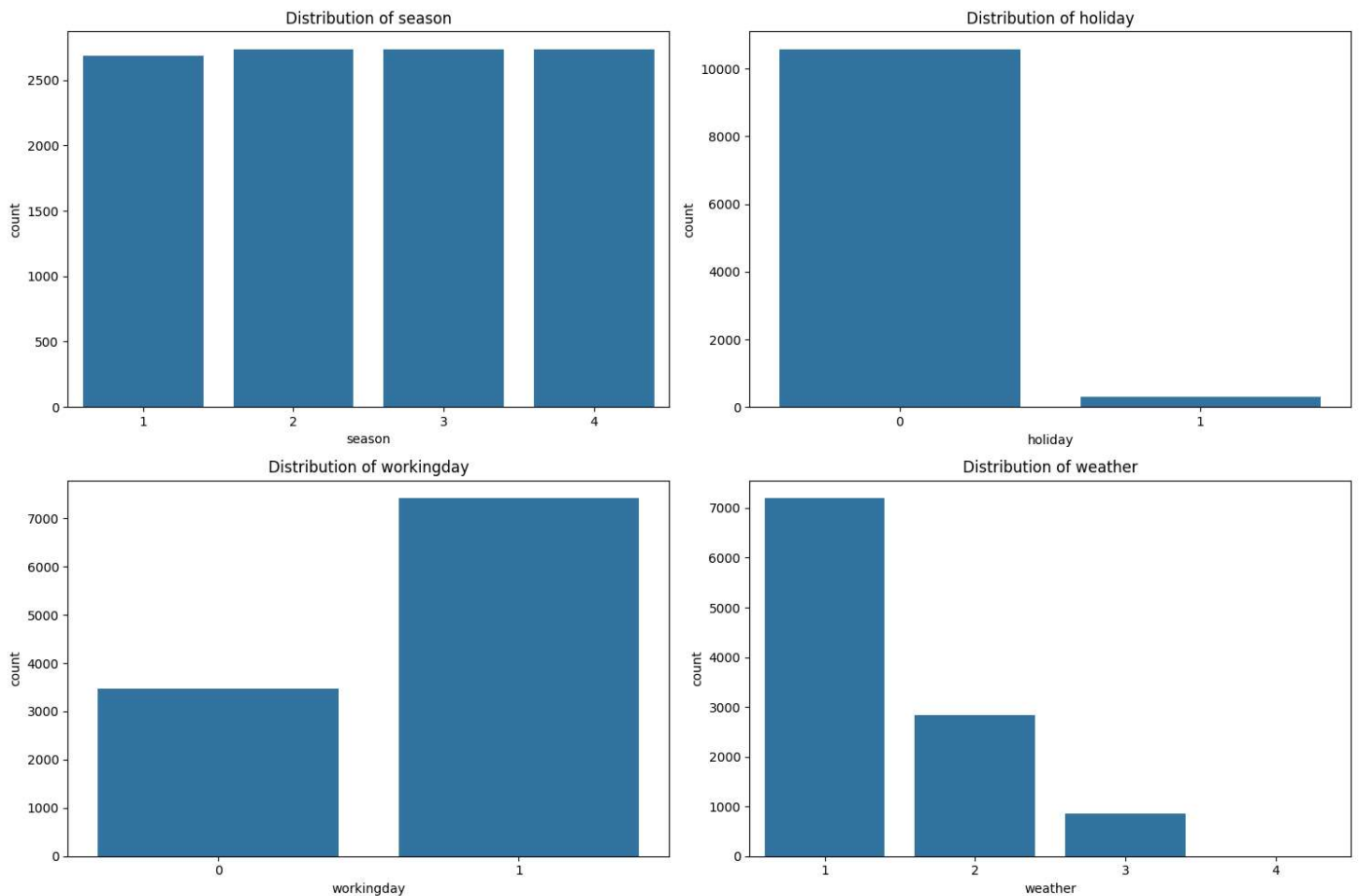
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming you have loaded your data into a DataFrame called 'data'

# Categorical Features
categorical_features = ['season', 'holiday', 'workingday', 'weather']
data = df
# Plot countplots for categorical features
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))
axes = axes.ravel()

for i, feature in enumerate(categorical_features):
    sns.countplot(data=data, x=feature, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature}')

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



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

# Assuming you have loaded your data into a DataFrame called 'data'

# Numerical Features
numerical_features = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']

# Boxplots to identify outliers
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
axes = axes.ravel()

for i, feature in enumerate(numerical_features):
    sns.boxplot(data=data, x=feature, ax=axes[i])
    axes[i].set_title(f'Boxplot for {feature}')

plt.tight_layout()
plt.show()

# Dealing with outliers
for feature in numerical_features:
    Q1 = data[feature].quantile(0.25)
    Q3 = data[feature].quantile(0.75)
    IQR = Q3 - Q1

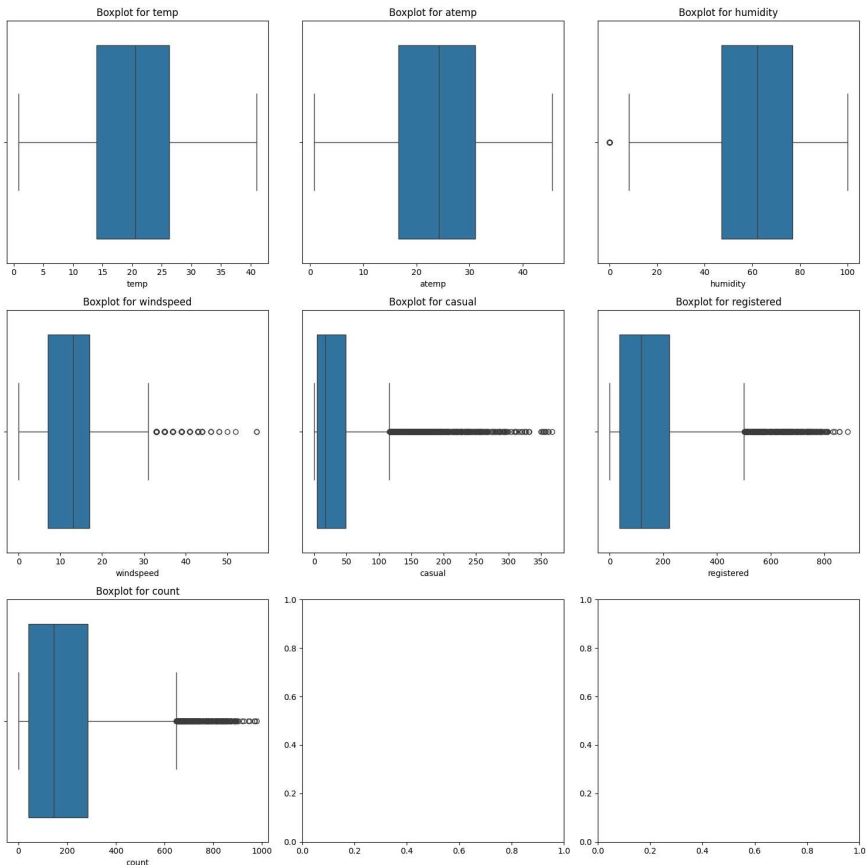
    # Identify outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[feature] < lower_bound) | (data[feature] > upper_bound)]

    print(f"Outliers in {feature}:")
    print(outliers[[feature]])

    # Option 1: Remove outliers
    # data = data[(data[feature] >= lower_bound) & (data[feature] <= upper_bound)]

    # Option 2: Clip outliers
    data.loc[data[feature] < lower_bound, feature] = lower_bound
    data.loc[data[feature] > upper_bound, feature] = upper_bound

print("Data after dealing with outliers:")
print(data[numerical_features].describe())
```



Outliers in temp:

Empty DataFrame

Columns: [temp]

Index: []

Outliers in atemp:

Empty DataFrame

Columns: [atemp]

Index: []

Outliers in humidity:

humidity	
1091	0
1092	0
1093	0
1094	0
1095	0
1096	0
1097	0
1098	0
1099	0
1100	0

Insights : In humidity we can see that, there are no outliers

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

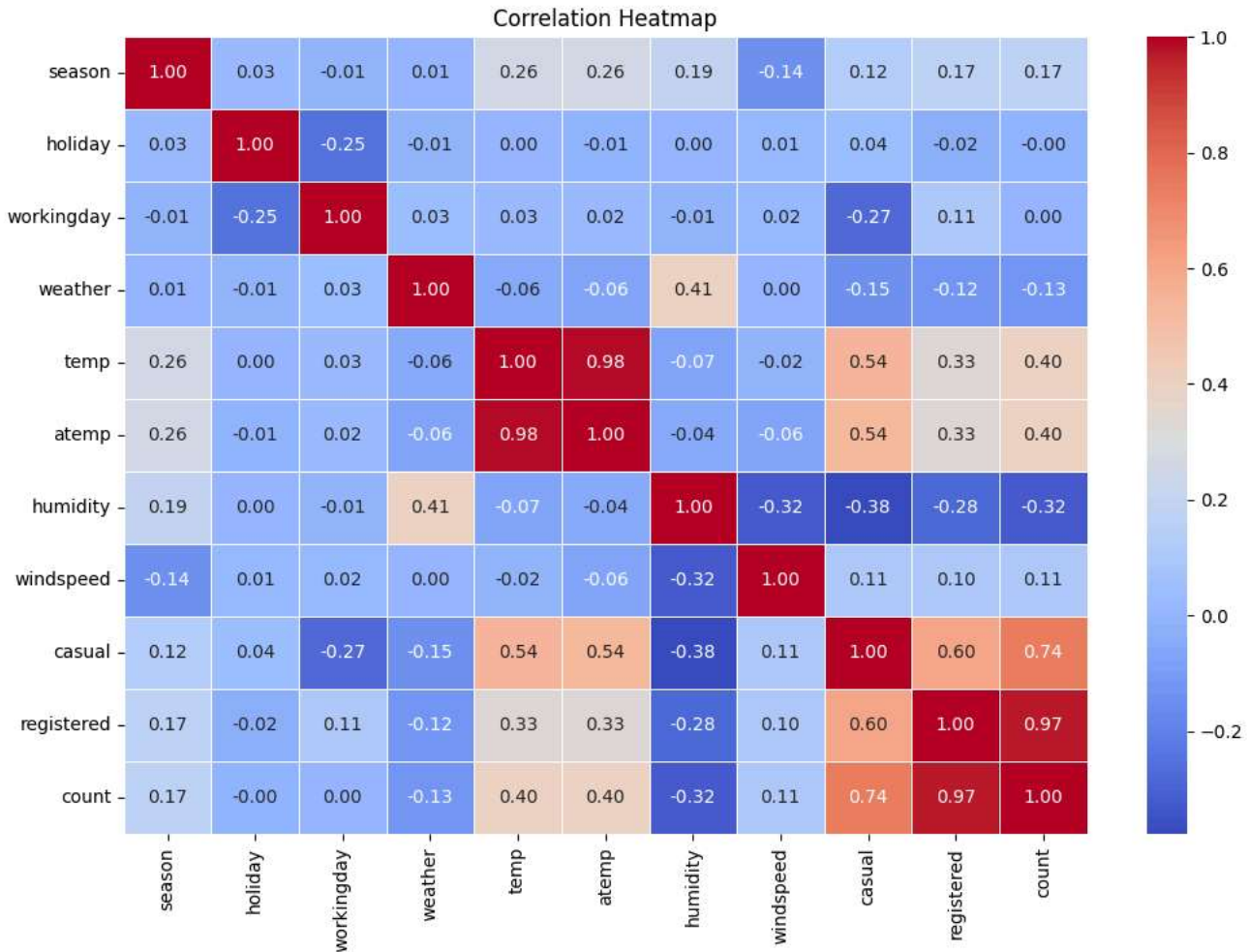
Hint: i. Plot a Correlation Heatmap and draw insights. ii. Remove the highly correlated variables, if any.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Exclude non-numeric columns from correlation computation
numerical_columns = data.select_dtypes(include=['int64', 'float64']).columns

# Compute the correlation matrix
correlation_matrix = data[numerical_columns].corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

```
# Step 1: Compute the correlation matrix
correlation_matrix = data[numerical_columns].corr()

# Step 2: Set a threshold for correlation coefficient
threshold = 0.8 # You can adjust this threshold based on your needs

# Step 3: Identify highly correlated pairs of variables
highly_correlated = set()
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname_i = correlation_matrix.columns[i]
            colname_j = correlation_matrix.columns[j]
            highly_correlated.add((colname_i, colname_j))

# Step 4: Remove one variable from each highly correlated pair
columns_to_drop = set()
for col1, col2 in highly_correlated:
    # Remove one of the variables based on some criteria (e.g., keep the variable with higher correlation with the target variable)
    # Here, we simply remove the second variable in the pair
    columns_to_drop.add(col2)

# Drop the identified columns from the dataset
data_filtered = data.drop(columns=columns_to_drop)

# Print the columns that have been dropped
print("Columns dropped due to high correlation:", columns_to_drop)

Columns dropped due to high correlation: {'registered', 'temp'}
```

Insights : the columns which are having correlation close to 1 are having high correlation, we need to remove them to avoid the multicollinearity issues. Here From the above data, we can see registered and temp have high correlation

3.3. 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) b. Select an appropriate test - i. Hint: 2- Sample Independent T-test c. Set a significance level i. Hint: $\alpha=5\%$ is recommended d. Calculate test Statistics / p-value e. Decide whether to accept or reject the Null Hypothesis.

Hint: i. If the p-value is less than or equal to the predetermined level of significance (α), we have evidence to reject the null hypothesis. ii. If the p-value is greater than the predetermined level of significance (α), we do not have sufficient evidence to reject the null hypothesis. f. Draw inferences & conclusions from the analysis and provide recommendations.

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1) H0: There is no significant difference in the number of bike rides between weekdays and weekends. H1: There is a significant difference in the number of bike rides between weekdays and weekends.

b. Select an appropriate test Since we're comparing the means of two independent groups (weekdays and weekends), we'll use the two-sample independent t-test.

c. Set a significance level We'll set the significance level (α) to 5%, which is a commonly used value.

```
import pandas as pd
from scipy.stats import ttest_ind

# Assuming you have loaded your data into a DataFrame called 'data'

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

# Create a new column 'day_type' to categorize weekdays and weekends
data['day_type'] = data['datetime'].dt.day_name().apply(lambda x: 'weekday' if x in ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'])

# Separate weekday and weekend data
weekday_data = data[data['day_type'] == 'weekday']['count']
weekend_data = data[data['day_type'] == 'weekend']['count']

# Perform the two-sample independent t-test
t_statistic, p_value = ttest_ind(weekday_data, weekend_data)

print(f"T-statistic: {t_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Example inference and recommendation
if p_value <= 0.05:
    print("Since the p-value is less than or equal to the significance level (0.05), we reject the null hypothesis.")
    print("There is enough evidence to suggest a significant difference in the number of bike rides between weekdays and weekends.")
    print("Recommendation: Adjust bike availability and staffing levels accordingly to meet the different demand patterns on weekdays and weekends.")
else:
    print("Since the p-value is greater than the significance level (0.05), we fail to reject the null hypothesis.")
    print("There is no significant evidence of a difference in the number of bike rides between weekdays and weekends.")
    print("Recommendation: Maintain consistent bike availability and staffing levels across weekdays and weekends.")

T-statistic: 0.2056
P-value: 0.8371
Since the p-value is greater than the significance level (0.05), we fail to reject the null hypothesis.
There is no significant evidence of a difference in the number of bike rides between weekdays and weekends.
Recommendation: Maintain consistent bike availability and staffing levels across weekdays and weekends.
```

Insights : There is no significant evidence of a difference in the number of bike rides between weekdays and weekends. Recommendation: Maintain consistent bike availability and staffing levels across 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 -

i. Hint: One-way ANOVA test c. Check assumptions of the test i. Normality Hint:

1. Use Histogram, Q-Q Plot, Skewness & Kurtosis
2. Shapiro-Wilk's test ii. Equality Variance Hint:
3. Levene's test 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. i. Hint: $\alpha=5\%$ is recommended e. Decide whether to accept or reject the Null Hypothesis. Hint: i. If the p-value is less than or equal to the predetermined level of significance (α), we have evidence to reject the null hypothesis. ii. If the p-value is greater than the predetermined level of significance (α), we do not have sufficient evidence to reject the null hypothesis. f. Draw inferences & conclusions from the analysis and provide recommendations.

a. Formulate Null Hypothesis (H0) and Alternate Hypothesis (H1) H0: The demand for bicycles on rent is the same across different weather conditions. H1: The demand for bicycles on rent is not the same across different weather conditions.

b. Select an appropriate test Since we're comparing the means of more than two groups (different weather conditions), we'll use the One-way ANOVA test.

c. Check assumptions of the test i. Normality We'll check the normality assumption using a combination of visual methods (histogram, Q-Q plot) and the Shapiro-Wilk test.

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import shapiro, probplot

# Assuming you have loaded your data into a DataFrame called 'data'

# Separate data by weather conditions
weather_groups = data.groupby('weather')['count']

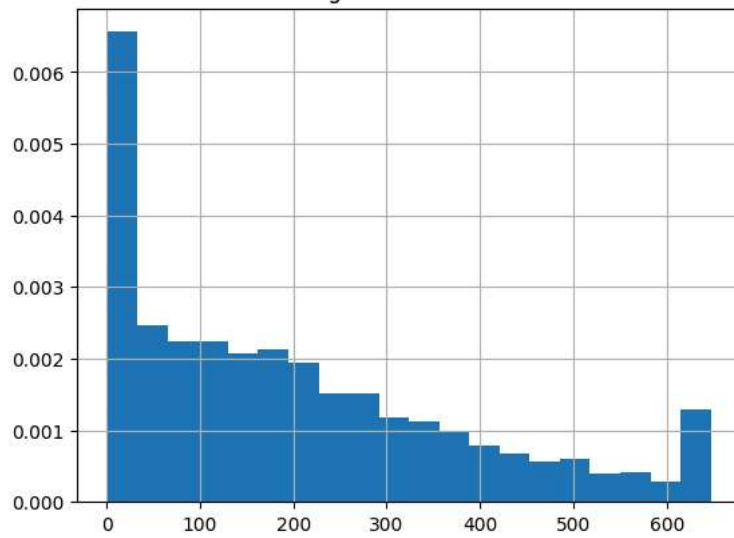
# Visual analysis of normality
for weather, group in weather_groups:
    plt.figure()
    group.hist(bins=20, density=True)
    plt.title(f'Histogram for Weather {weather}')
    plt.show()

    plt.figure()
    probplot(group, plot=plt)
    plt.title(f'Q-Q Plot for Weather {weather}')
    plt.show()

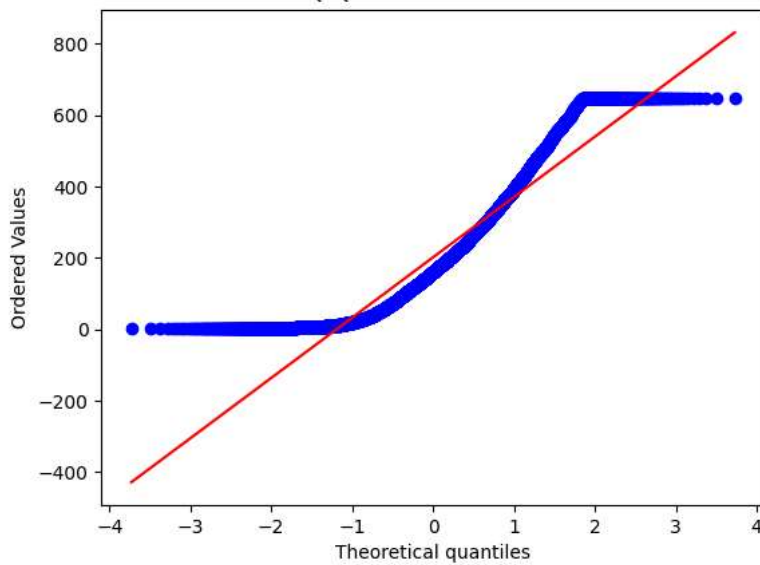
# Shapiro-Wilk test
for weather, group in weather_groups:
    if len(group) >= 3:
        stat, p_value = shapiro(group)
        print(f'Weather {weather}: Shapiro-Wilk Statistic={stat:.4f}, p-value={p_value:.4f}')
    else:
        print(f'Weather {weather}: Normality assumption cannot be tested (fewer than 3 data points)')

# Rest of the code for Levene's test, One-way ANOVA, and conclusions
```

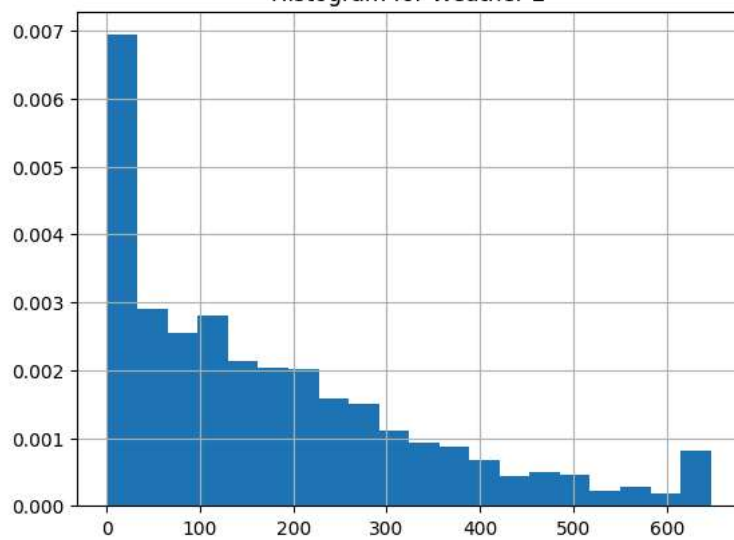
Histogram for Weather 1



Q-Q Plot for Weather 1



Histogram for Weather 2



Q-Q Plot for Weather 2

