Yulu Hypothesis

1. Exploratory Data Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy.stats import ttest_ind
from scipy.stats import chi2_contingency
from scipy.stats import f_oneway, levene, shapiro, kruskal
from statsmodels.graphics.gofplots import qqplot
```

```
df = pd.read_csv('bike_sharing.csv')
```

```
# Display the shape of the dataset
print("Dataset Shape:", df.shape)

# Display information about the dataset
print("\nDataset Information:")
df.info()

# Display statistical summary of numerical columns
print("\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

Statistical Summary:
             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
```

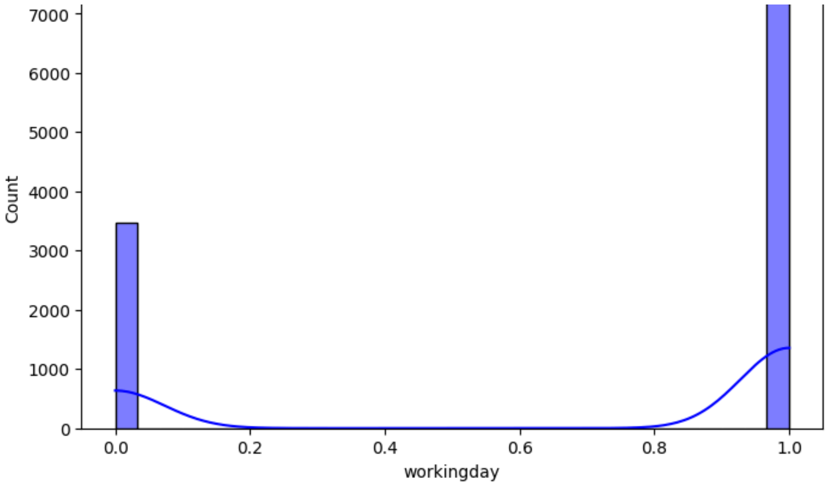
```
# Check for missing values
missing_values = df.isnull().sum()

# Display missing values
print("\nMissing Values:")
print(missing_values)
```

```
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
```

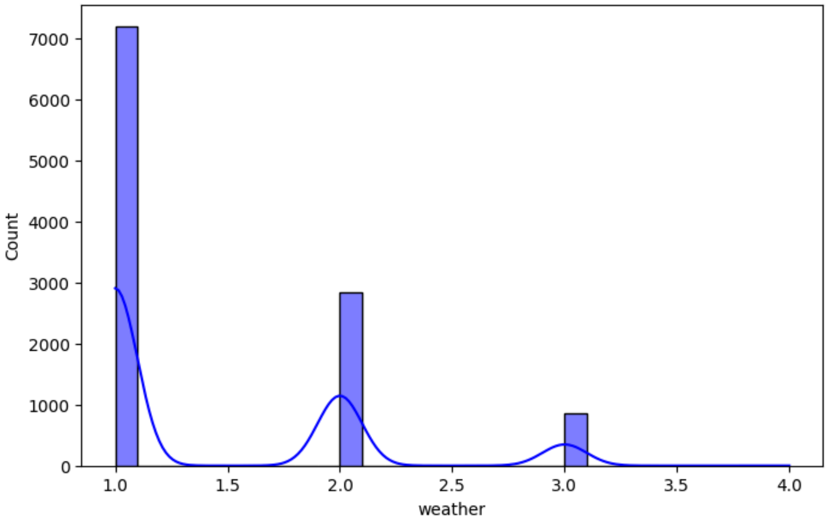
```
# Check for duplicate records
duplicate_rows = df[df.duplicated()]

# Display duplicate records
print("\nDuplicate Records:")
print(duplicate_rows)

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

```
Duplicate Records:
Empty DataFrame
Columns: [datetime, season, holiday, workingday, weather, temp, atemp, humidity, windspeed, casual, registered, count]
Index: []
```

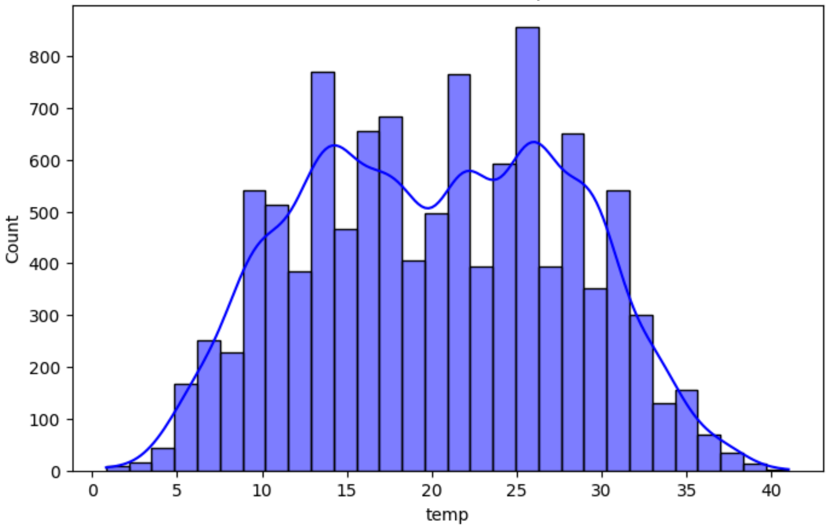
```
# Numerical features - Histogram and Distplot
numerical_features = df.select_dtypes(include=np.number).columns
for feature in numerical_features:
    plt.figure(figsize=(8, 5))
    sns.histplot(df[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
    plt.show()

# Categorical features - Countplot and Pie Chart
categorical_features = df.select_dtypes(include='object').columns
for feature in categorical_features:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=df, x=feature, palette='viridis')
    plt.title(f'Countplot of {feature}')
    plt.show()

    plt.figure(figsize=(8, 8))
    df[feature].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('viridis'), wedgeprops=dict(width=0.
    plt.title(f'Pie Chart of {feature}')
    plt.ylabel('')
    plt.show()
```



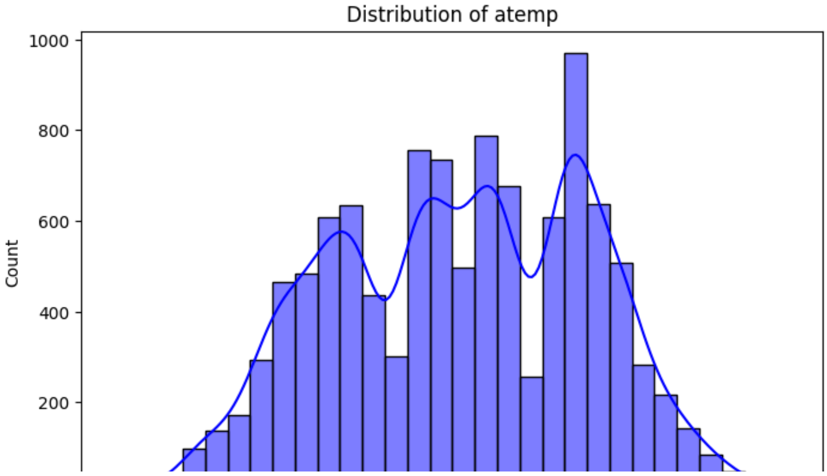
Distribution of weather

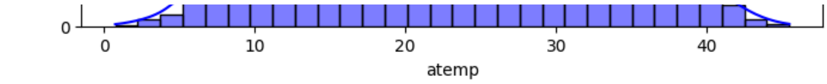


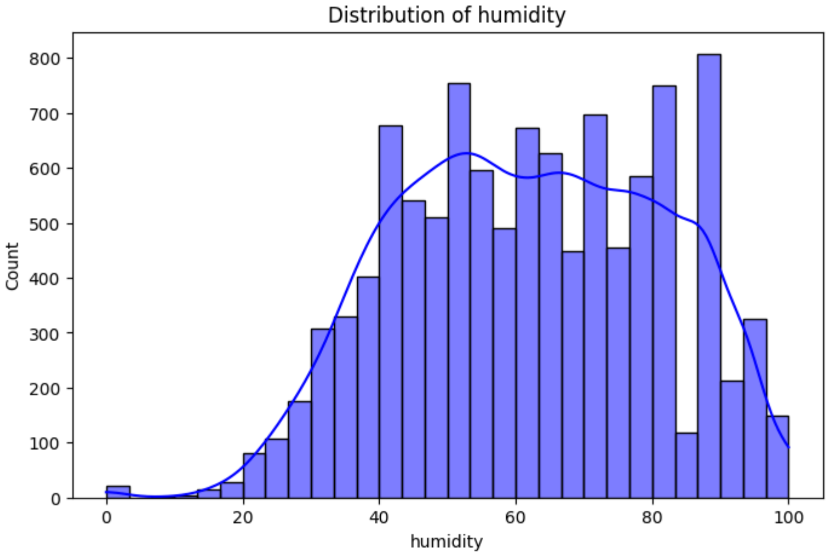
Distribution of temp

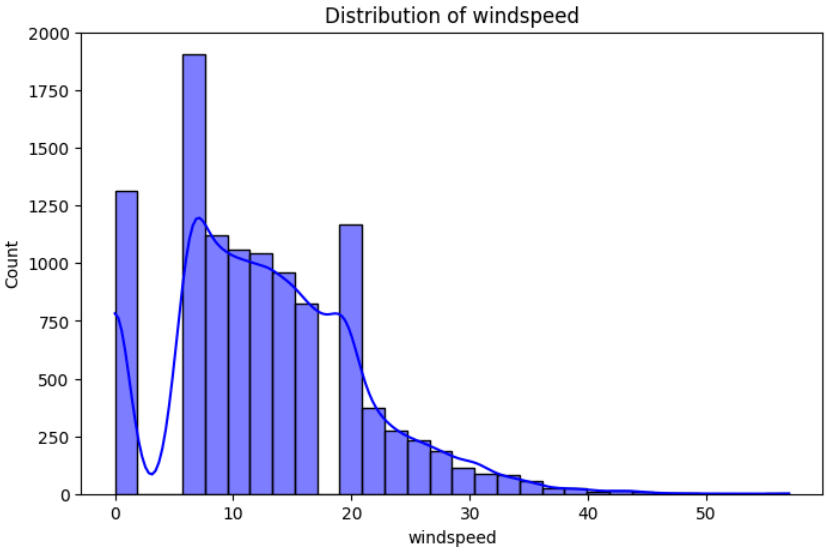


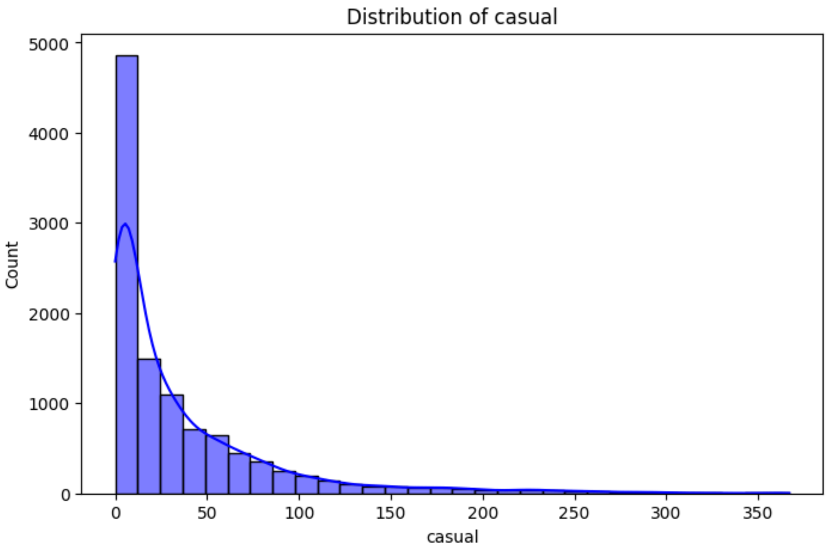
Distribution of atemp

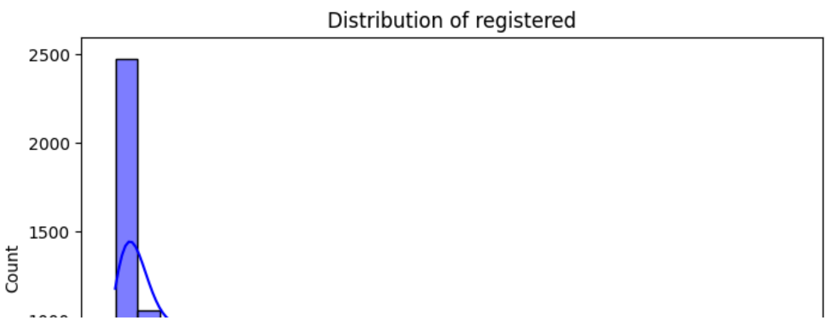


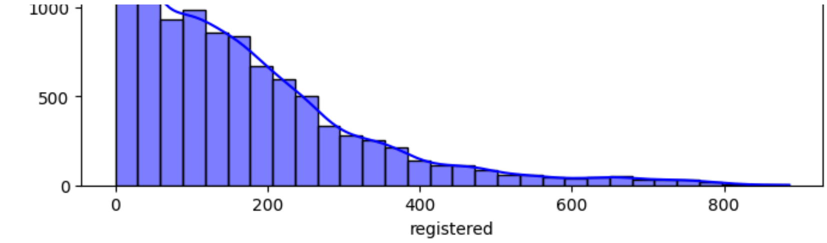

0
0
10
20
30
40
atemp

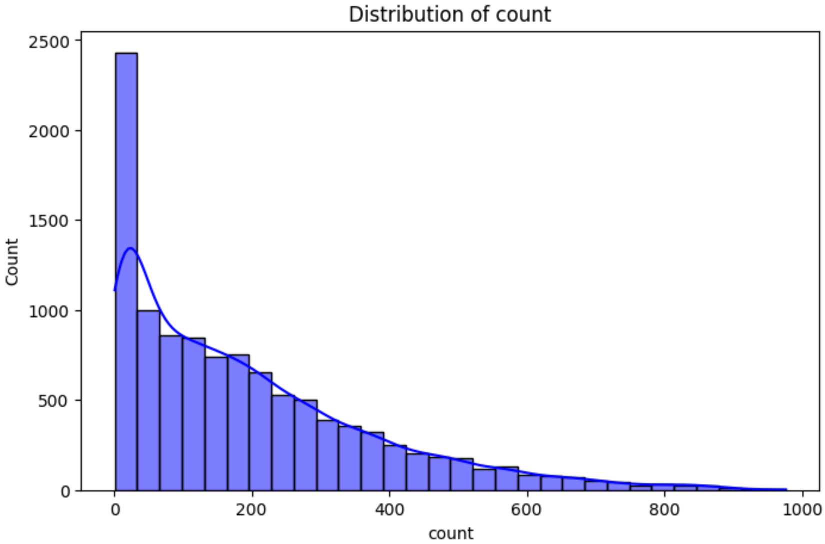

Distribution of humidity
Count
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100
humidity


Distribution of windspeed
Count
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50
windspeed

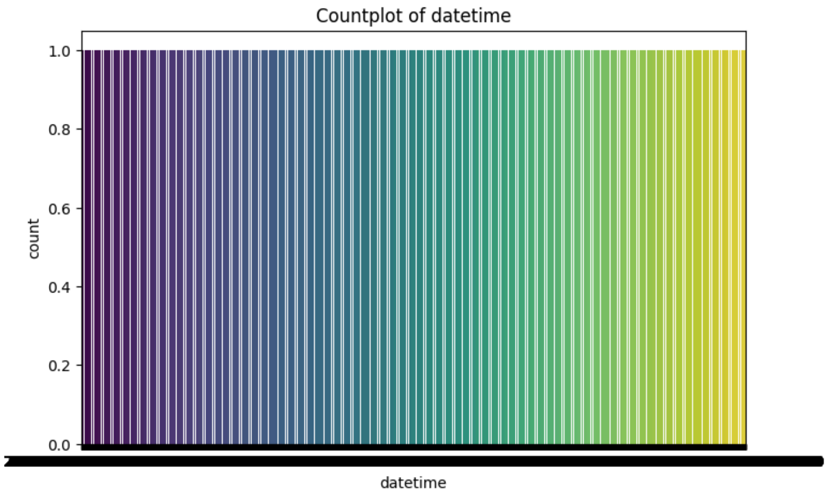

Distribution of casual
Count
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
300
350
casual

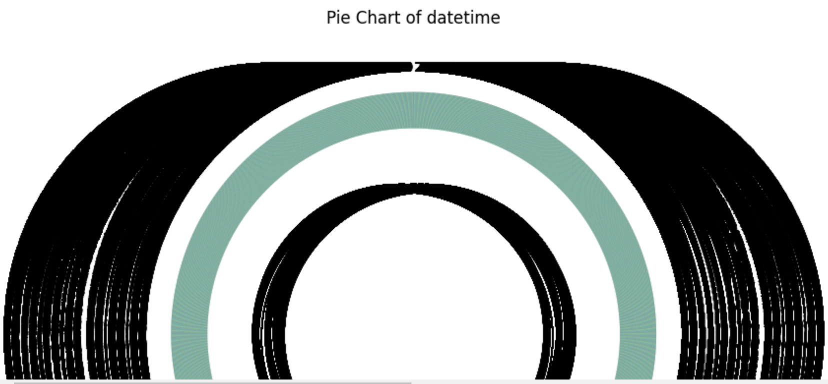

Distribution of registered
Count
1500
2000
2500



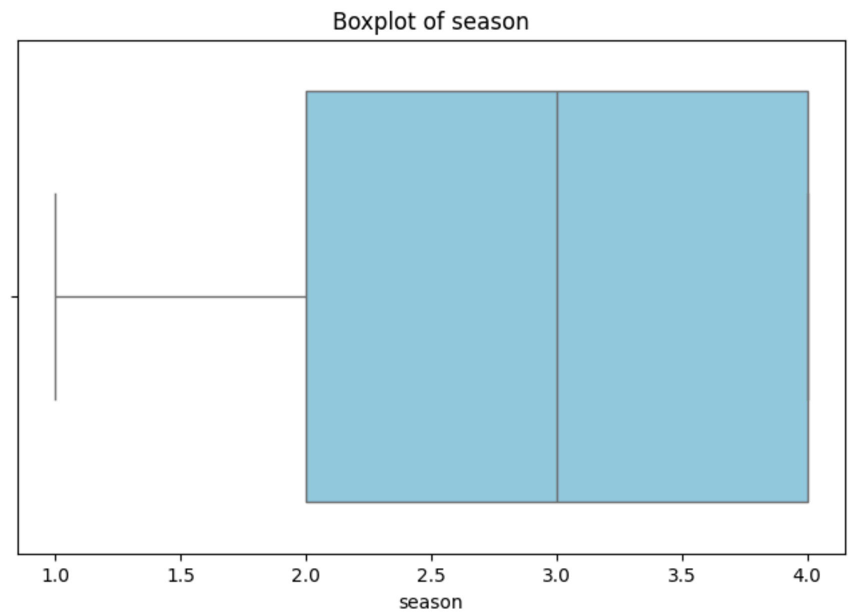


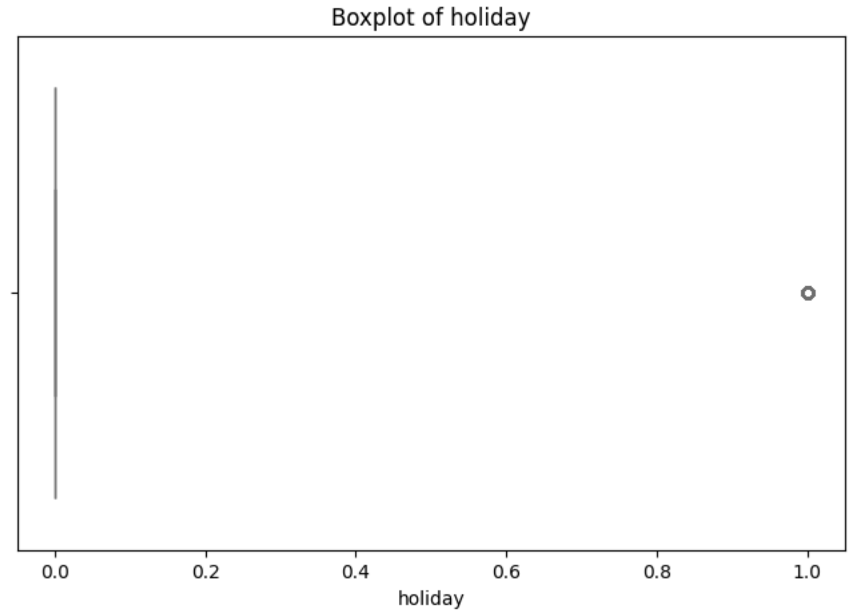

```
<ipython-input-12-c6ce68ee7938>:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.

  sns.countplot(data=df, x=feature, palette='viridis')
```

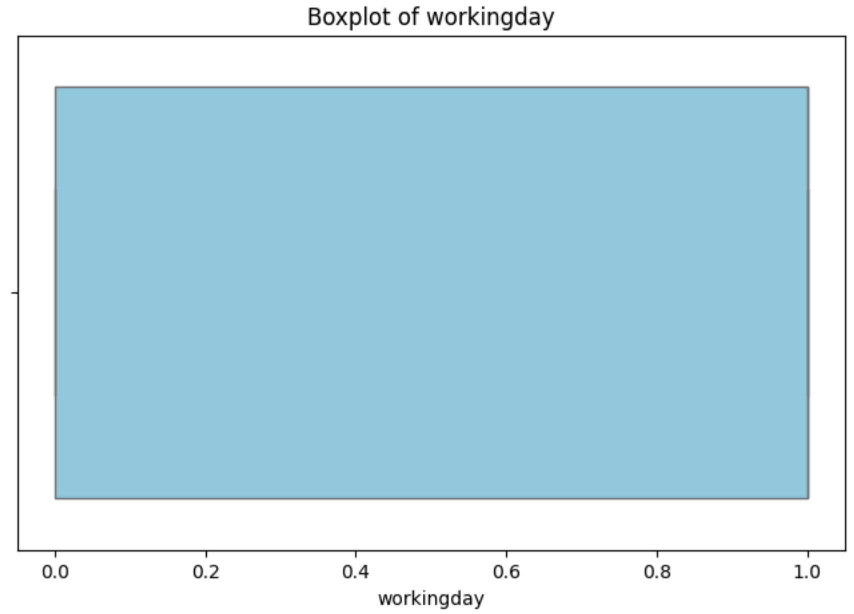


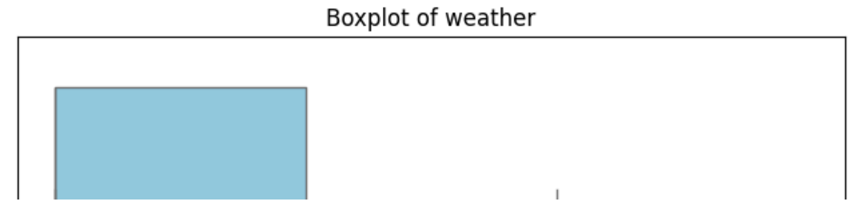


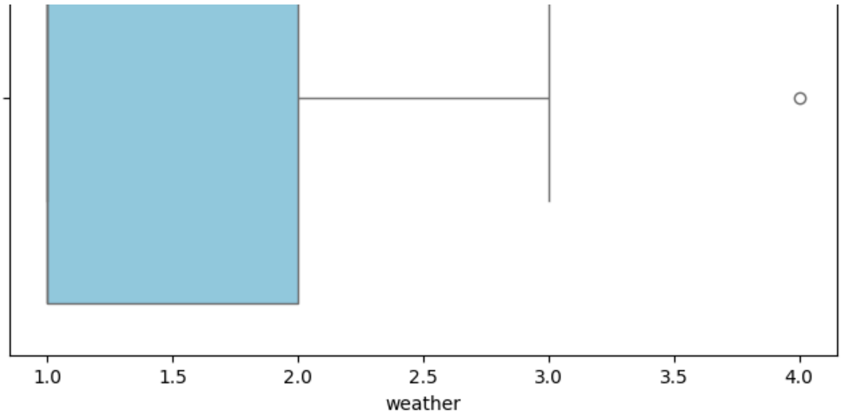
```
# Numerical features - Boxplot and IQR
for feature in numerical_features:
    plt.figure(figsize=(8, 5))
    sns.boxplot(x=df[feature], color='skyblue')
    plt.title(f'Boxplot of {feature}')
    plt.show()

    # Calculate IQR
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1

    # Remove or clip outliers
    df[feature] = np.where((df[feature] < (Q1 - 1.5 * IQR)) | (df[feature] > (Q3 + 1.5 * IQR)), df[feature].median(), df[feature])
```


Boxplot of season
1.0
1.5
2.0
2.5
3.0
3.5
4.0
season

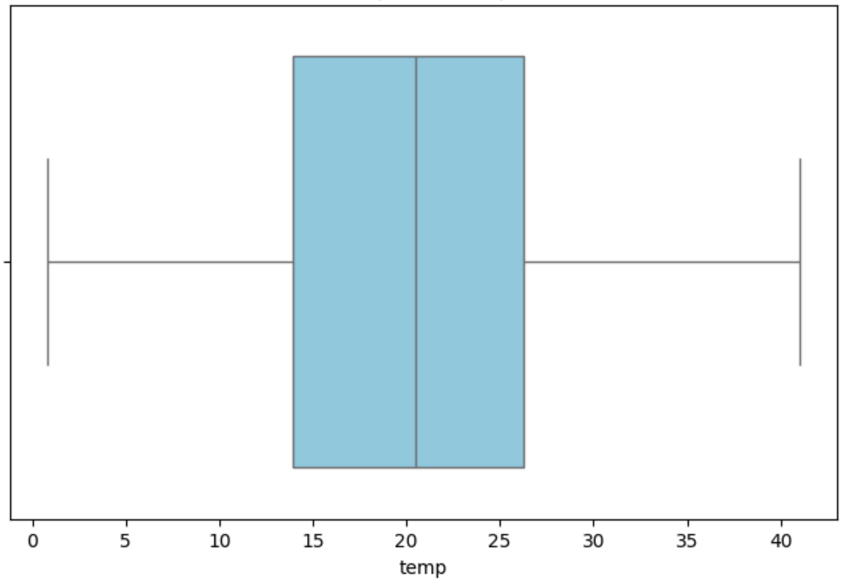

Boxplot of holiday
0.0
0.2
0.4
0.6
0.8
1.0
holiday


Boxplot of workingday
0.0
0.2
0.4
0.6
0.8
1.0
workingday

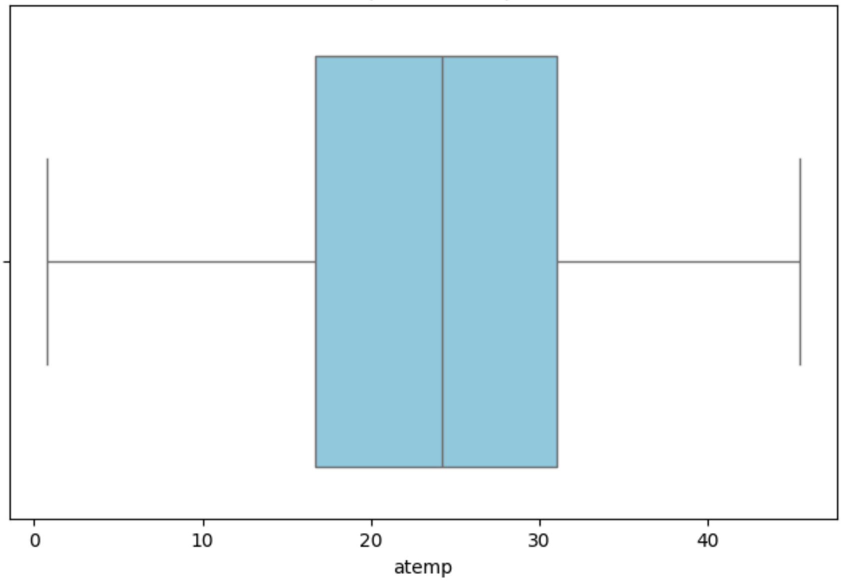

Boxplot of weather



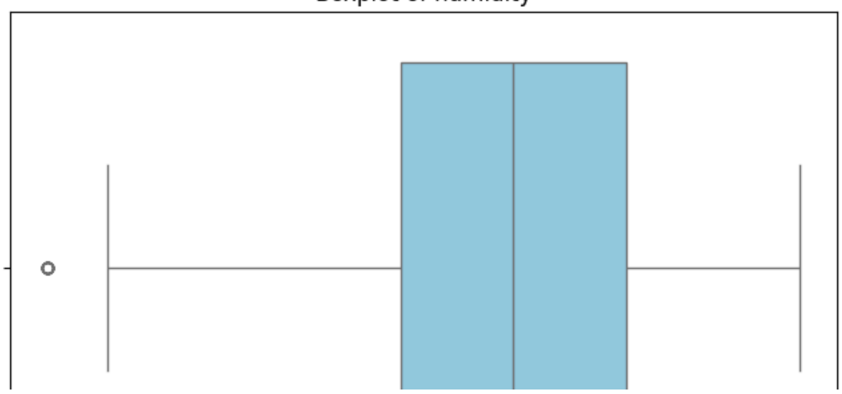
Boxplot of temp

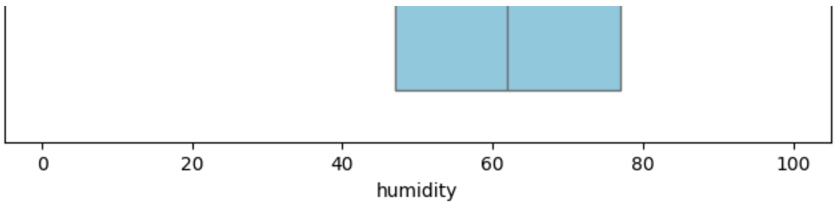


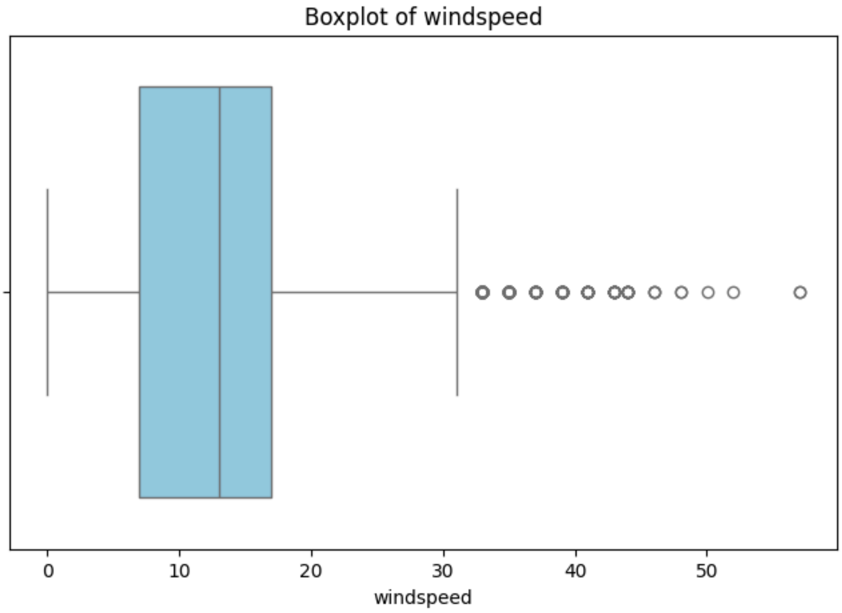
Boxplot of atemp

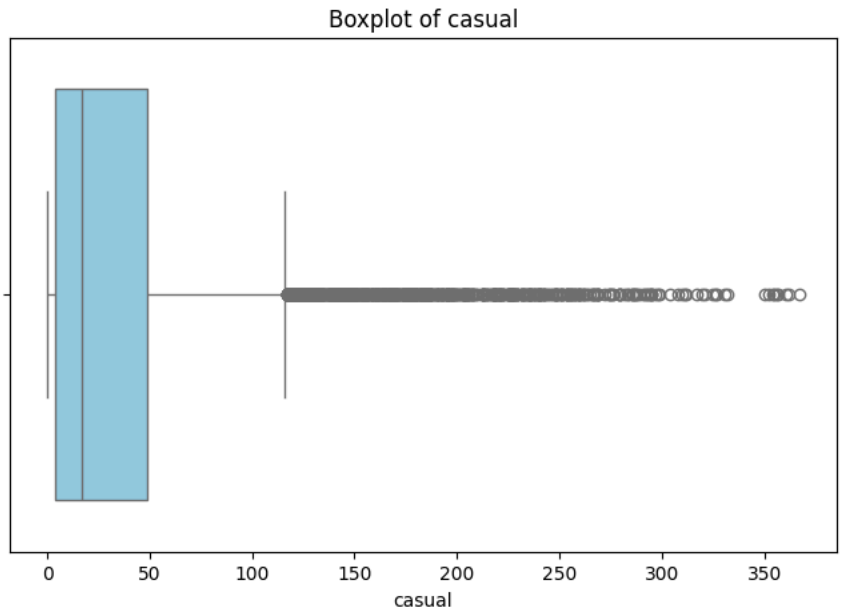


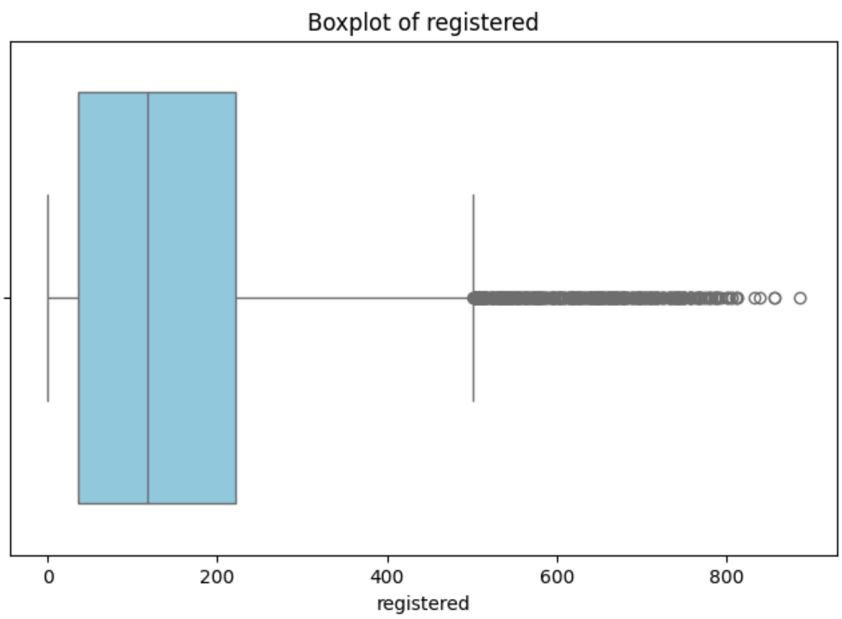
Boxplot of humidity

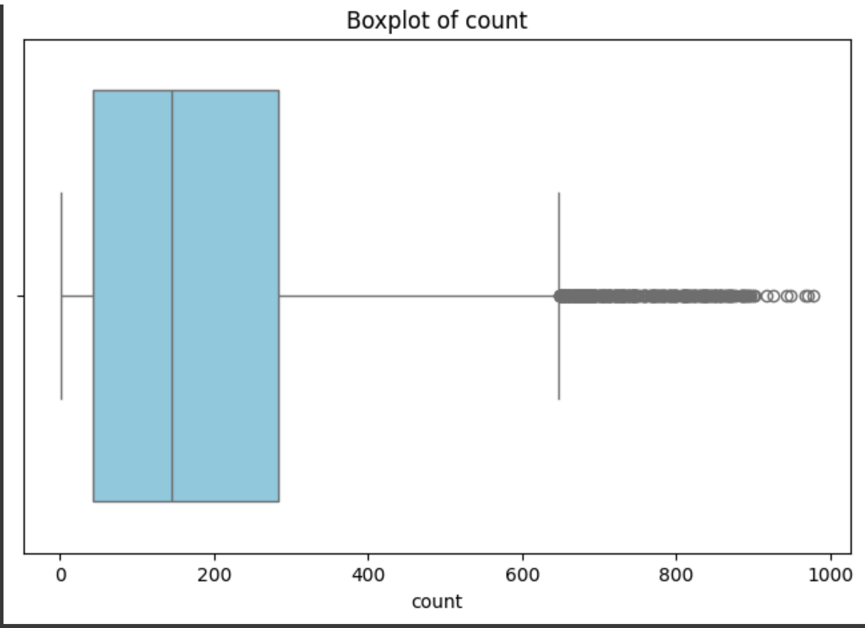



0
20
40
60
80
100
humidity


Boxplot of windspeed
0
10
20
30
40
50
windspeed


Boxplot of casual
0
50
100
150
200
250
300
350
casual


Boxplot of registered
0
200
400
600
800
registered

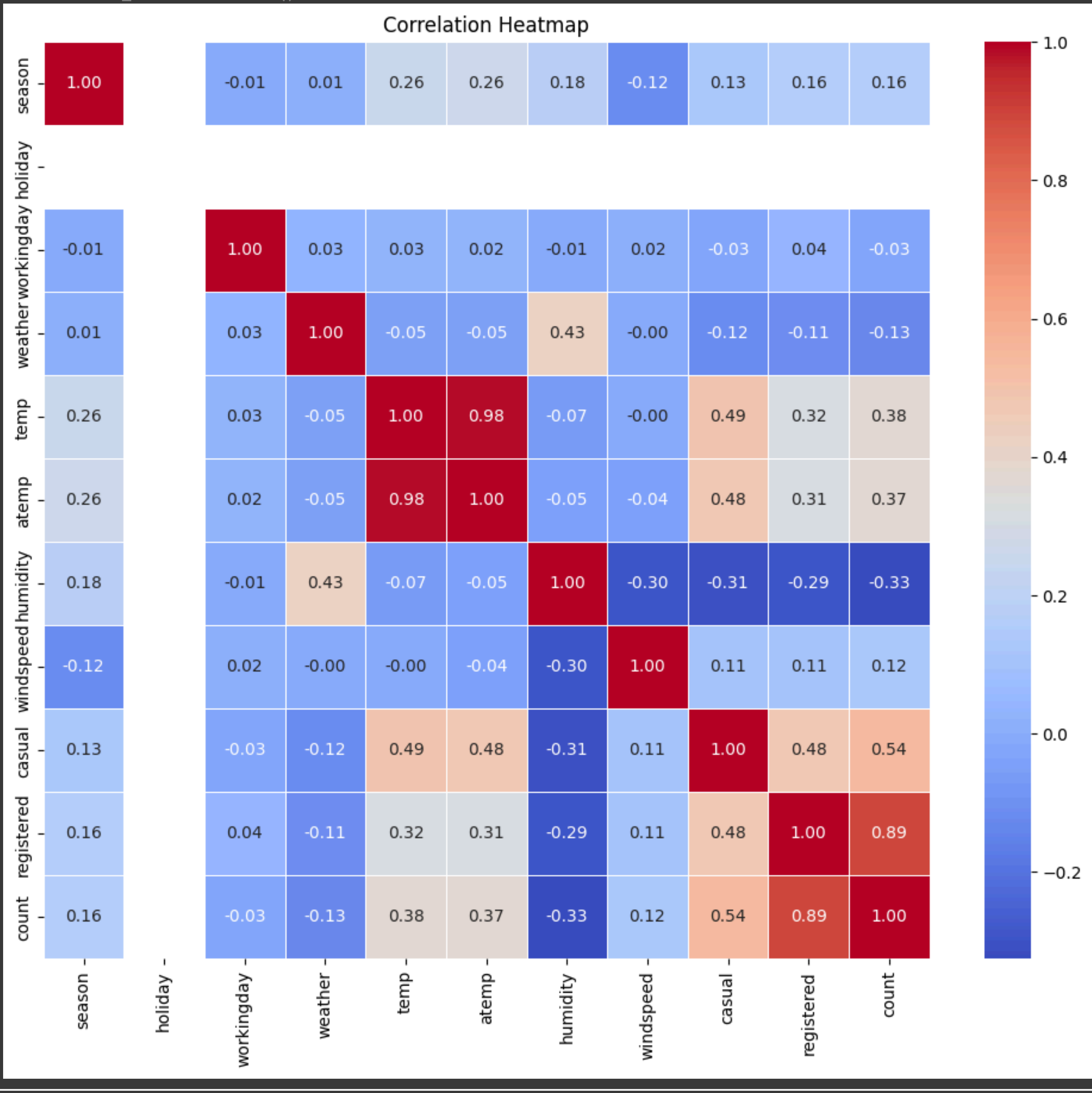


2. Try establishing a Relationship between the Dependent and Independent Variables.

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

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Plot a correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

```
<ipython-input-14-062546f54784>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  correlation_matrix = df.corr()
```



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)

•HO:- There is no significant diffence between the number of bike rides on weekdays and weekends.

H1:- There is a significant difference between the number of bike rides on Weekdays and Weekends.

```
#significance level

alpha= 0.05
```

Sample Independent T-test

```
data1 = df[df['workingday'] == 1]
weekdays = data1['count'].astype('int')
weekdays
```

```
47        5
48        2
49        1
50        3
51       30
        ...
10881   336
```

```
10882    241
10883    168
10884    129
10885     88
Name: count, Length: 7412, dtype: int64
```

```
sample_mean1 = weekdays.mean()
sample_mean1
```

```
172.06300593631948
```

```
data2 = df[df['workingday'] ==0 ]
weekends = data2['count'].astype('int')
weekends
```

```
0         16
1         40
2         32
3         13
4          1
        ...
10809    109
10810    122
10811    106
10812     89
10813     33
Name: count, Length: 3474, dtype: int64
```

```
sample_mean2 = weekends.mean()
sample_mean2
```

```
180.86067933218192
```

```
test_statistics, p_value = ttest_ind (weekdays, weekends, alternative ='two-sided')
```

```
test_statistics
```

```
-2.7743551537766913
```

```
p_value
```

```
0.005540553589361102
```

```
if alpha > p_value:
  print("Reject Null, we can conclude that there is a difference between the number of bike ride")
else:
  print("Fail to reject null hypothesis, we can conclude that there is no difference")
```

```
Reject Null, we can conclude that there is a difference between the number of bike ride
```

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

Formulate Hypothesis

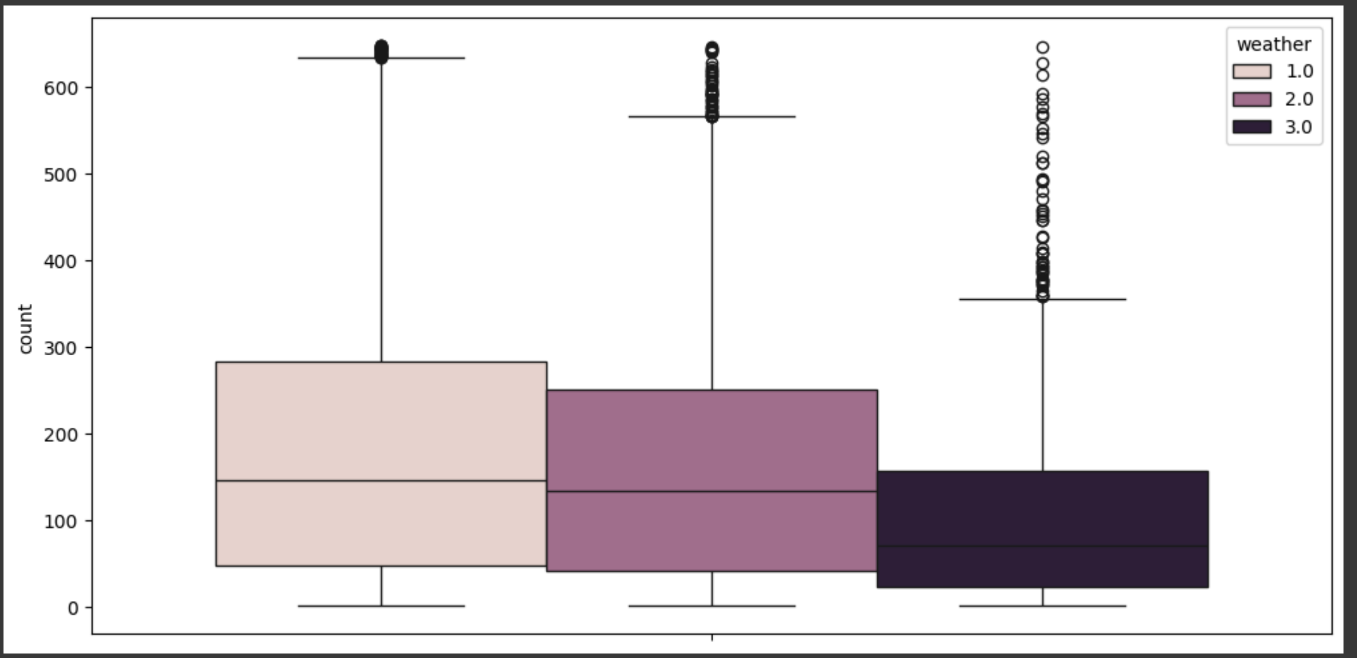
- HO:- The demand of bicycles on rent is not same for different weather conditions.
- H1:- The demand of bicycles on rent is same for different weather conditions.

```
#significance level

alpha = 0.05
```

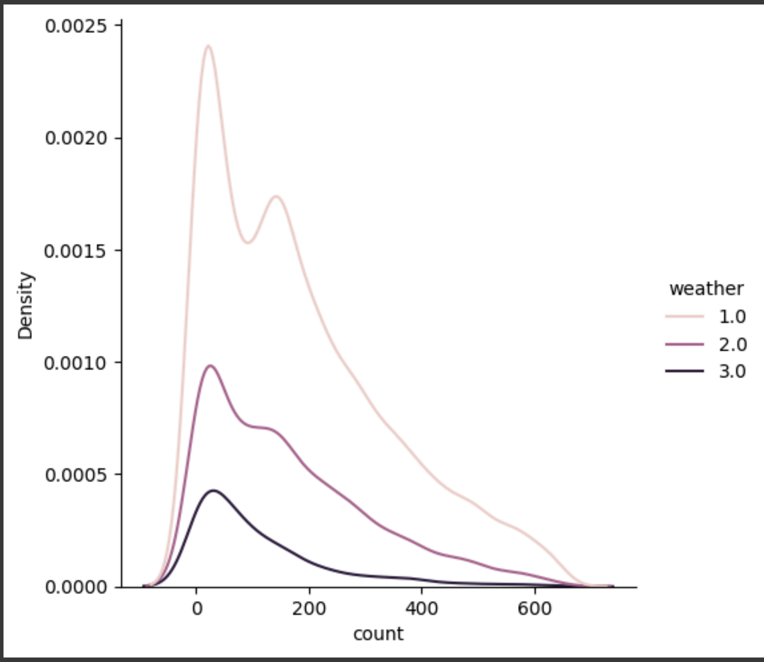
```
#Select One-way ANOVA test for the test.

w1 = df[df['weather'] == 1]['count']
w2 = df[df['weather'] == 2]['count']
w3 = df[df['weather'] == 3]['count']
w4 = df[df['weather'] == 4]['count']
```

Test for variance

```
plt.figure(figsize=(12,6))
sns.boxplot(data = df,y='count' ,hue = 'weather')
plt.show()
```




```
sns.displot(data = df, x = 'count', hue = 'weather', kind = 'kde')
plt.show()
```



Observation

- Variance within each group is not same.

Statistical test for variance using levene

- HO:- Variance are same.
- H1:- Variance are different

```
alpha = 0.05
```

```
test_statistics, p_value = levene(w1, w2, w3, w4)
```

```
test_statistics
```

nan

```
p_value
```

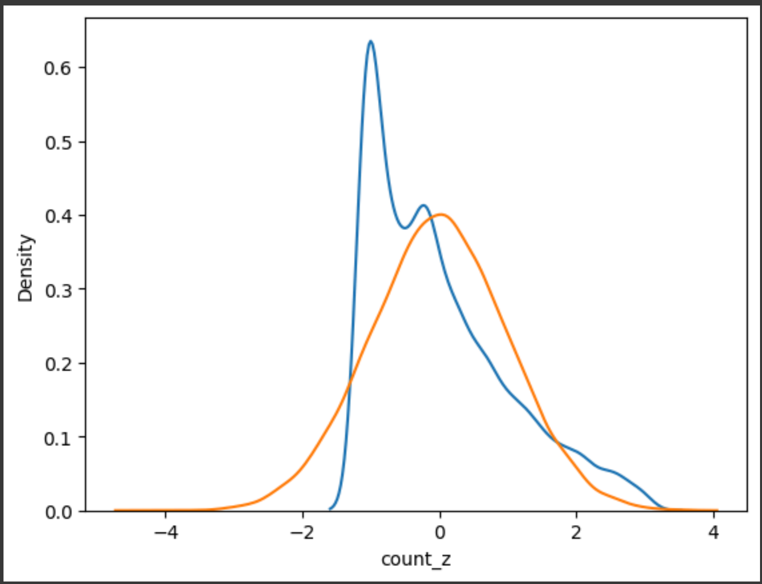
nan

Test for Normality

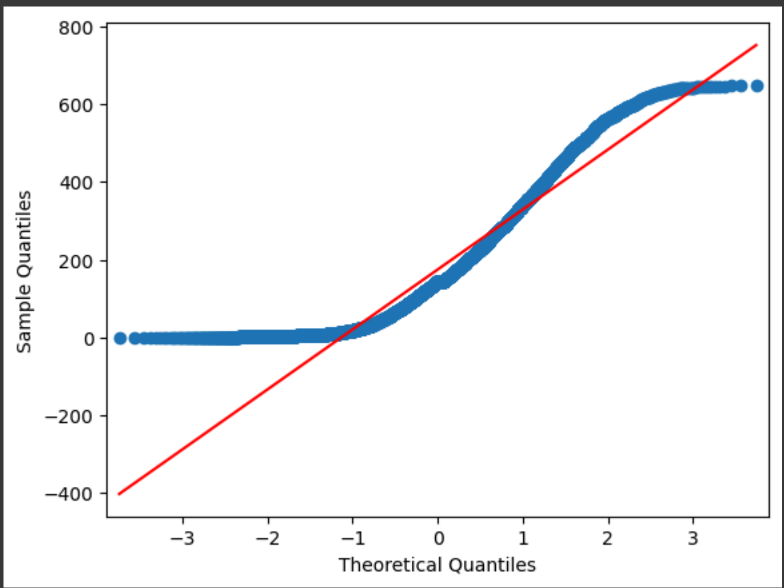
```
data1 = np.random.normal(0,1,10886)
```

```
df['count_z'] = (df['count'] - df['count'].mean()) / df['count'].std()
```

```
sns.kdeplot(df['count_z'])
sns.kdeplot(data1)
plt.show()
```




```
qqplot(df['count'], line= 's')
plt.show()
```



Statistical test for normality

- HO: Data is normally distributed.
- H1: Data is not normally distributed.

```
alpha = 0.05
```

```
test_statistics, p_value = shapiro(df['count'].sample(100))
```

```
test_statistics
```

```
0.8741297125816345
```

```
p_value
```

```
1.0326224497703151e-07
```

The assumptions of ANONA is not met, so we use Kruskal-Wallis test for the same

- HO:- The demand of bicycles on rent is not same for different weather conditions.
- H1:- The demand of bicycles on rent is same for different weather conditions.

```
alpha = 0.05
```

```
test_statistics, p_value = kruskal(w1, w2, w3, w4)

test_statistics
```

```
nan
```

```
p_value
```

```
nan
```

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

Formulate Hypothesis

- HO:- The demand of bicycles on rent is not same for different seasons.
- H1:- The demand of bicycles on rent is same for different seasons

```
alpha = 0.05
```

One way anova test

```
s1 = df[df['season'] == 1]['count']
s2 = df[df['season'] == 2]['count']
s3 = df[df['season'] == 3]['count']
s4 = df[df['season'] == 4]['count']
```

```
sns.boxplot(data=df,y= 'count' ,hue= 'season')
plt.show()
```