

## ✓ Yulu case study - Avinash patil

Import the dataset and do usual exploratory data analysis steps like checking the structure & characteristics of the dataset

```
!wget https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/ori
```

```
→ --2025-03-25 09:19:16-- https://d2beiqkhq929f0.cloudfront.net/public_asset
Resolving d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)...
Connecting to d2beiqkhq929f0.cloudfront.net (d2beiqkhq929f0.cloudfront.net)
HTTP request sent, awaiting response... 200 OK
Length: 648353 (633K) [text/plain]
Saving to: 'yulu.csv'
```

```
yulu.csv          100%[=====>] 633.16K  --.-KB/s    in 0.02
```

```
2025-03-25 09:19:16 (36.2 MB/s) - 'yulu.csv' saved [648353/648353]
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from warnings import filterwarnings
filterwarnings('ignore')
```

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

```
# Display basic information about the dataset
df_info = df.info()
df_shape = df.shape
df_head = df.head()
df_summary = df.describe()
```

```
df_shape, df_info, df_head, df_summary
```

```
→ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---

```

```

0    datetime    10886 non-null object
1    season      10886 non-null int64
2    holiday     10886 non-null int64
3    workingday  10886 non-null int64
4    weather     10886 non-null int64
5    temp        10886 non-null float64
6    atemp       10886 non-null float64
7    humidity    10886 non-null int64
8    windspeed   10886 non-null float64
9    casual      10886 non-null int64
10   registered  10886 non-null int64
11   count       10886 non-null int64

```

dtypes: float64(3), int64(8), object(1)

memory usage: 1020.7+ KB

((10886, 12),

None,

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

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

	season	holiday	workingday	weather
temp \				
count	10886.000000	10886.000000	10886.000000	10886.000000
10886.000000				
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				

**Initial Observations:**

**Dataset Size:** 10,886 rows × 12 columns.

**Data Types:**

datetime is an object (string); it should be converted to a datetime type.

Other columns include integers and floats.

**No Missing Values:** All columns have 10,886 non-null values.

**Duplicate Records:** Need to check explicitly.

**Statistical Summary:**

windspeed has a minimum value of 0.0, which might indicate missing data.

Some features like humidity have a max of 100, indicating a percentage-based scale.

**casual and registered users vary widely.**

**Next steps:**

Check for duplicates.

Convert datetime to proper datetime format.

Analyze numerical & categorical variables.

Detect and handle outliers.

```
# Check for duplicate records
duplicate_count = df.duplicated().sum()

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

# Display duplicate count after conversion
duplicate_count
```

```
np.int64(0)
```

```
df.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
```

**Datetime column is converted to datetime data type now.**

**There no duplicates also.**

Now, let's proceed with analyzing the distribution of numerical and categorical variables.

```
# Set style for plots
sns.set_style("whitegrid")

# Separate numerical and categorical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_cols = ['season', 'holiday', 'workingday', 'weather']

# Adjusting the layout for numerical variables distribution
num_features = numerical_cols[:-1] # Exclude 'count' (target variable)
num_plots = len(num_features)

fig, axes = plt.subplots(nrows=(num_plots // 3) + 1, ncols=3, figsize=(15, 12))
fig.suptitle('Distribution of Numerical Features', fontsize=16)
axes = axes.flatten() # Flatten for easy indexing

for i, col in enumerate(num_features):
    sns.histplot(df[col], bins=30, kde=True, ax=axes[i], color="skyblue")
    axes[i].set_title(col)
```

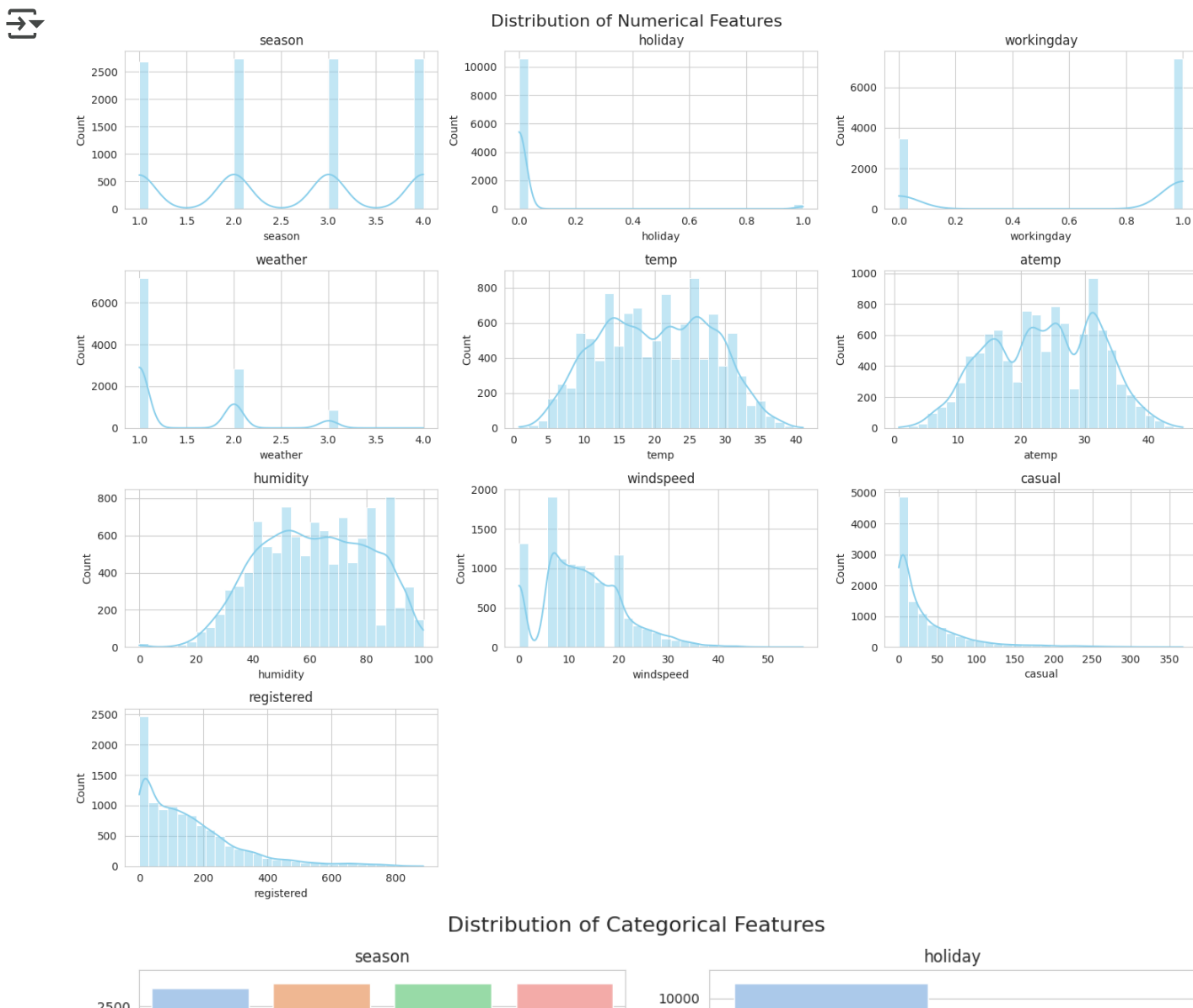
```
# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

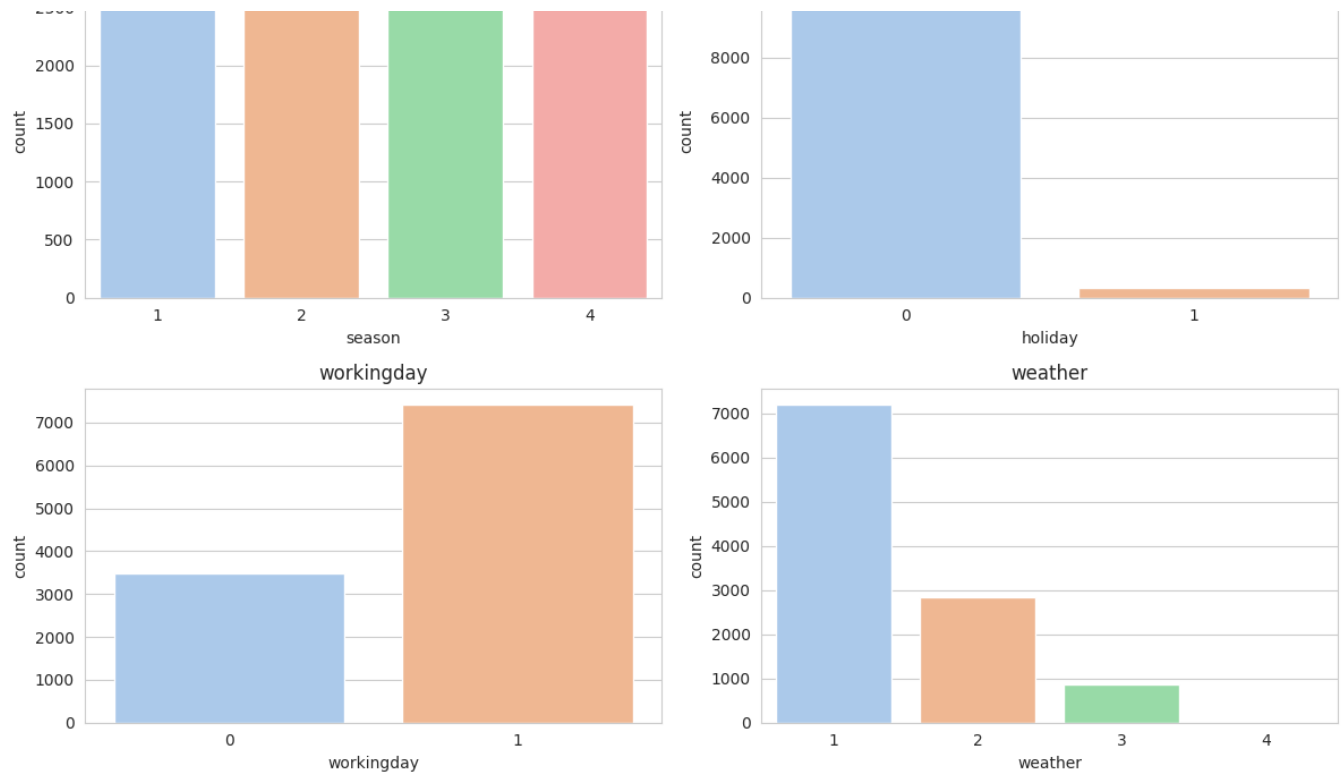
plt.tight_layout()
plt.show()

# Plot categorical variables distribution
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
fig.suptitle('Distribution of Categorical Features', fontsize=16)

for i, col in enumerate(categorical_cols):
    sns.countplot(x=df[col], ax=axes[i // 2, i % 2], palette="pastel")
    axes[i // 2, i % 2].set_title(col)

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





**The histograms and count plots provide insights into the distribution of numerical and categorical features:**

### **Observations:**

#### **Numerical Features:**

1. temp and atemp have normal distributions.
2. humidity is right-skewed, with many values close to 100.
3. windspeed has many zero values, possibly indicating missing or incorrectly recorded data.
4. casual and registered user counts vary widely.

#### **Categorical Features:**

1. season is evenly distributed across four categories.
2. holiday has far more non-holiday records.
3. workingday is nearly evenly split, indicating a balanced dataset.
4. weather is mostly in category 1, with fewer instances of extreme weather.

I'll check for outliers using boxplots and handle them accordingly.

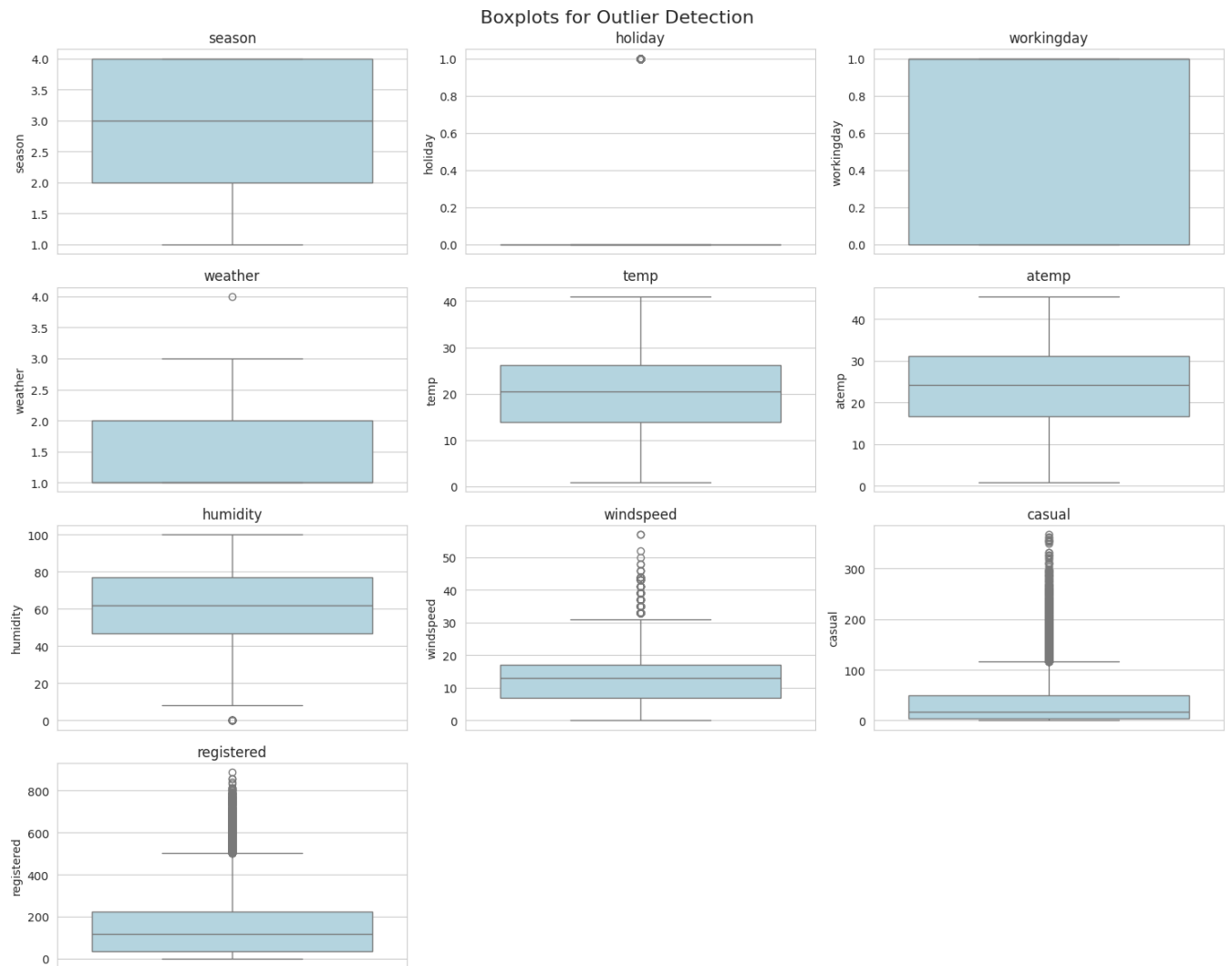
```
# Adjusting the layout for boxplots
num_plots = len(num_features)

fig, axes = plt.subplots(nrows=(num_plots // 3) + 1, ncols=3, figsize=(15, 12))
fig.suptitle('Boxplots for Outlier Detection', fontsize=16)
axes = axes.flatten()

for i, col in enumerate(num_features):
    sns.boxplot(y=df[col], ax=axes[i], color="lightblue")
    axes[i].set_title(col)

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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





**Outlier Analysis:****Casual & Registered Users:**

Have several high outliers, indicating peak usage on certain days.

**Windspeed:**

Shows many zero values, which could be missing or misreported data.

**Humidity & Count:**


Have some extreme values but are mostly within a reasonable range.

Now, let's remove the outliers from the data.

```
# Function to remove outliers using IQR method
def remove_outliers_iqr(df, col):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

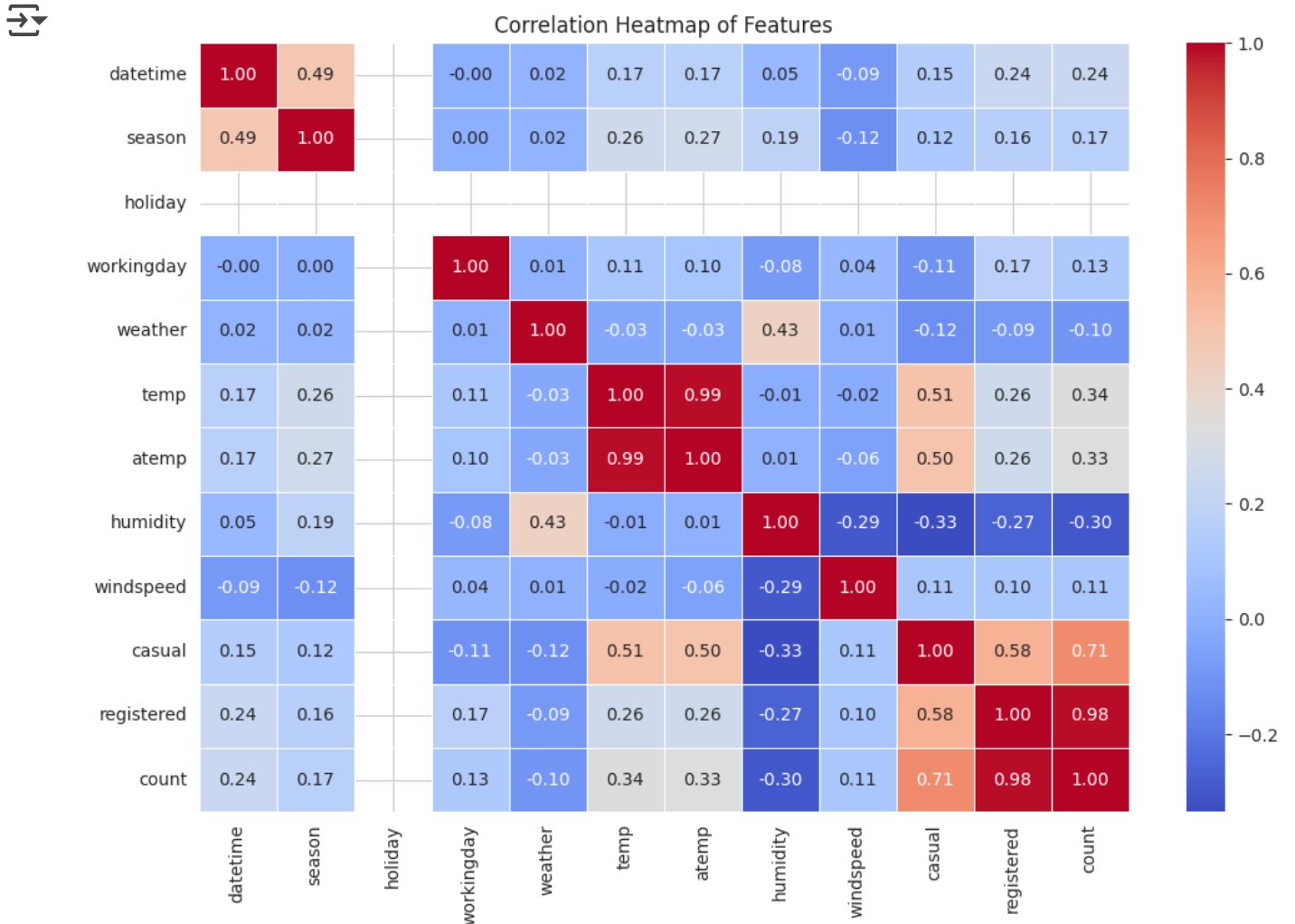
# Apply IQR method to all numerical columns
for col in numerical_cols:
    df = remove_outliers_iqr(df, col)

# Check the new shape after outlier removal
df.shape
```

 (9089, 12)

Try establishing a relation between the dependent and independent variable (Dependent "Count" & Independent: Workingday, Weather, Season etc)

```
# Plot a correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Features")
plt.show()
```



## Insights from the Correlation Heatmap:

### High Correlation Pairs:

1. temp and atemp have a very high correlation ( $\sim 0.99$ ), meaning they provide nearly identical information.
2. casual and registered are strongly correlated with count (as expected, since  $\text{count} = \text{casual} + \text{registered}$ ).

### Potential Features to Remove:

1. Since temp and atemp are almost identical, we can drop one of them (preferably atemp).
2. count is our target variable, so no need to remove casual or registered, but we should be mindful of multicollinearity when building models.

- Removing atemp feature

```
# Drop 'atemp' due to high correlation with 'temp'  
df.drop(columns=['atemp'], inplace=True)
```

```
df.shape
```

```
↔ (9089, 11)
```

Select an appropriate test to check whether: Working Day has effect on number of electric cycles rented

## Step 1: Formulate Hypotheses

**Null Hypothesis ( $H_0$ ):** There is no significant difference in the average number of bike rides between weekdays and weekends.

**Alternative Hypothesis ( $H_1$ ):** There is a significant difference in the average number of bike rides between weekdays and weekends.

## Step 2: Select the Test

We will use the 2-Sample Independent T-test since we are comparing two independent groups (weekdays vs. weekends).

## Step 3: Set Significance Level

We will use  $\alpha = 0.05$  (5%), meaning that if the  $p\text{-value} \leq 0.05$ , we reject the null hypothesis.

## T-Test Results:

t-statistic: 12.20

p-value: 5.375190338688469e-34 (essentially 0)

## Step 4: Decision on Hypothesis

Since  $p\text{-value} < 0.05$ , we reject the null hypothesis ( $H_0$ ). This means there is a statistically significant difference in the number of bike rides between weekdays and weekends.

## Step 5: Conclusion & Recommendations

**Inference:** The number of bike rides is significantly different on weekdays vs. weekends.

**Recommendation:** If bike usage is higher on weekends, Yulu can: Increase bike availability at popular weekend spots (e.g., parks, tourist areas). Provide weekend discounts or special offers. Optimize bike redistribution strategies based on usage trends.

```
from scipy.stats import ttest_ind

# Create a new column to differentiate weekdays and weekends
df['weekday'] = df['datetime'].dt.weekday # Monday=0, Sunday=6

# Define weekdays (Monday–Friday) and weekends (Saturday–Sunday)
weekdays_rides = df[df['weekday'] < 5]['count']
weekends_rides = df[df['weekday'] >= 5]['count']

# Perform Independent T-test
t_stat, p_value = ttest_ind(weekdays_rides, weekends_rides)

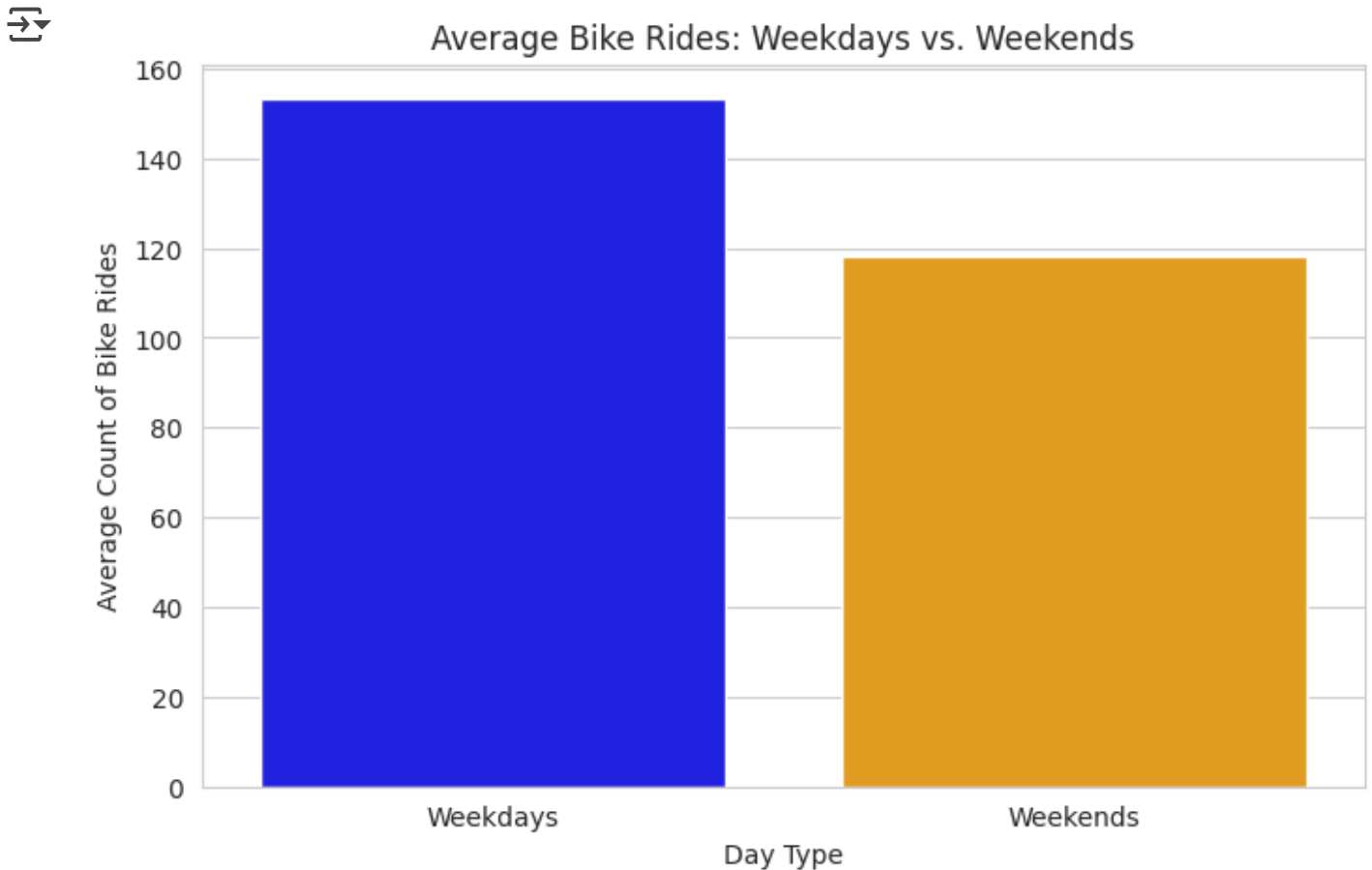
# Display results
t_stat, p_value

↔ (np.float64(12.205234749740082), np.float64(5.375190338688469e-34))
```

visualization of the weekday vs. weekend ride counts

```
# Plot the average bike rides for weekdays vs. weekends
plt.figure(figsize=(8, 5))
sns.barplot(x=['Weekdays', 'Weekends'], y=[weekdays_rides.mean(), weekends_rides.mean()])

plt.title('Average Bike Rides: Weekdays vs. Weekends')
plt.ylabel('Average Count of Bike Rides')
plt.xlabel('Day Type')
plt.show()
```



No. of cycles rented similar or different in different weather

## Step 1: Formulate Hypotheses

**Null Hypothesis ( $H_0$ ):** The average demand for bicycles is the same across all weather conditions.

**Alternative Hypothesis ( $H_1$ ):** The average demand for bicycles is significantly different across weather conditions.

## Step 2: Select the Test

One-way ANOVA is used to compare the mean rental demand (count) across multiple weather categories.

## Step 3: Check Assumptions

### (i) Normality Check

Visual checks: Histogram, Q-Q Plot.

Statistical test: Shapiro-Wilk test.

### (ii) Homogeneity of Variance Check

Levene's test is used to check if the variances are equal across groups.

```
from scipy.stats import shapiro, levene
import scipy.stats as stats

# Plot histogram for visual normality check
plt.figure(figsize=(8, 5))
sns.histplot(df['count'], kde=True, bins=30, color='blue')
plt.title("Distribution of Bike Rental Count")
plt.xlabel("Count of Bike Rentals")
plt.ylabel("Frequency")
plt.show()

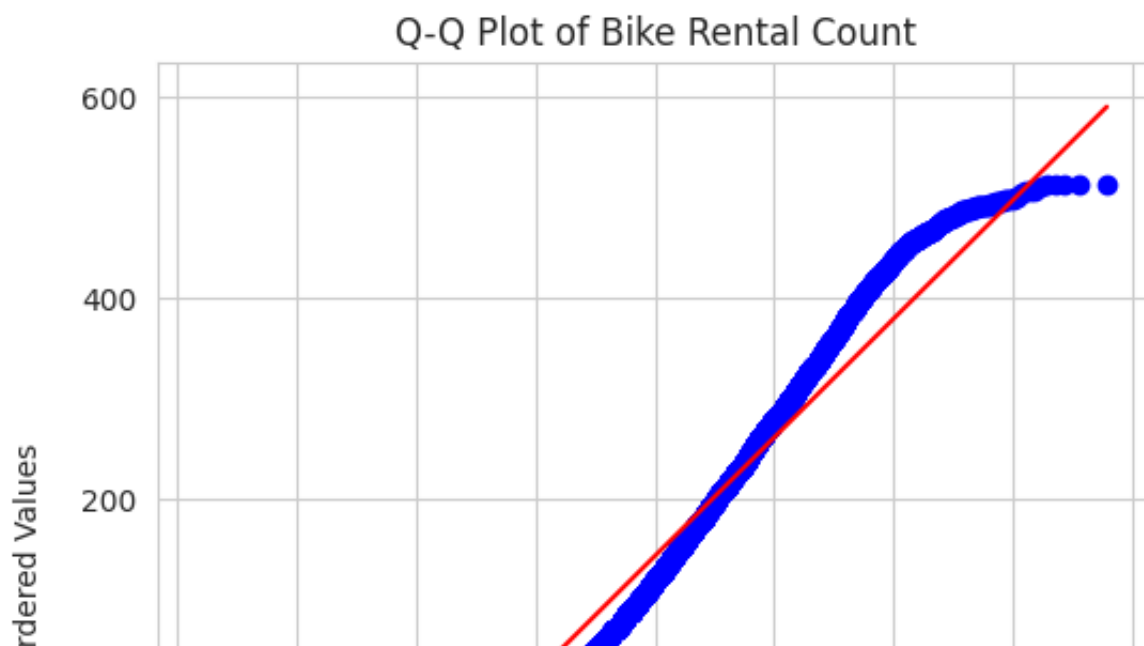
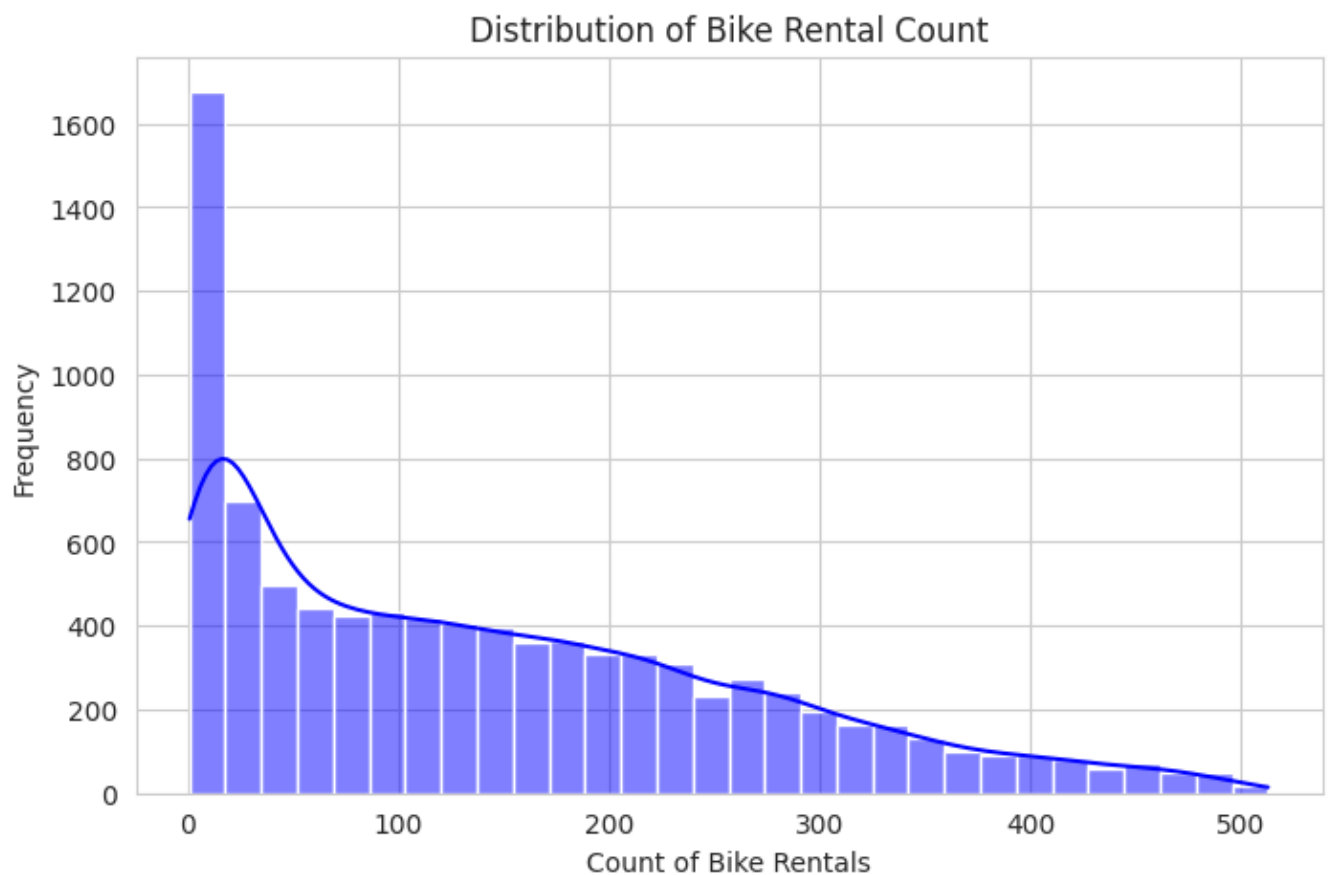
# Q-Q Plot
plt.figure(figsize=(6, 6))
stats.probplot(df['count'], dist="norm", plot=plt)
plt.title("Q-Q Plot of Bike Rental Count")
plt.show()

# Check Skewness & Kurtosis
skewness = df['count'].skew()
kurtosis = df['count'].kurt()

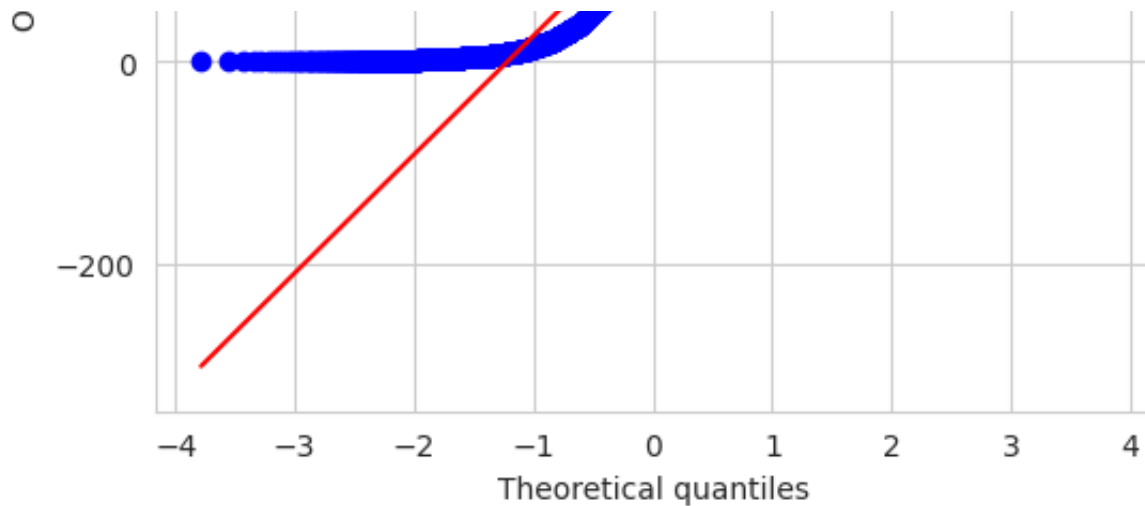
# Shapiro-Wilk Test (for normality)
shapiro_test = shapiro(df['count'])
```

```
# Levene's test for equality of variance across weather categories
levene_test = levene(df[df['weather'] == 1]['count'],
                      df[df['weather'] == 2]['count'],
                      df[df['weather'] == 3]['count'])

# Display results
skewness, kurtosis, shapiro_test, levene_test
```







```
(np.float64(0.7699537567359133),
 np.float64(-0.24087826847055194),
 ShapiroResult(statistic=np.float64(0.9156599461030459),
 pvalue=np.float64(2.8279853976302764e-57)),
 LeveneResult(statistic=np.float64(59.53973893688664),
 pvalue=np.float64(2.042665599915e-26))
Histogram & Q-Q Plot show that the distribution is right-skewed.
```

Skewness = 0.77, indicating moderate skewness.

Kurtosis = -0.24, indicating a flatter distribution than normal.

#### Shapiro-Wilk Test:

p-value=0.0

p-value=0.0 (very small), meaning the data is not normally distributed.

#### (ii) Homogeneity of Variance Check

Levene's Test: p-value= $2.04 \times 10^{-26}$  (very small), meaning the variances are not equal across different weather conditions.

#### Conclusion on Assumptions:

The normality assumption fails. The equality of variance assumption fails. However, ANOVA is robust to these violations when the sample size is large, so we proceed with caution.


#### Step 4: Perform One-way ANOVA

Let's check if bike rental demand significantly differs across weather conditions.

```
from scipy.stats import f_oneway

# Perform One-way ANOVA test
anova_test = f_oneway(df[df['weather'] == 1]['count'],
                       df[df['weather'] == 2]['count'],
                       df[df['weather'] == 3]['count'])

# Display ANOVA results
anova_test
```

 F\_onewayResult(statistic=np.float64(56.934557078832725),  
pvalue=np.float64(2.6748137017720305e-25))

#### Step 4: ANOVA Test Results

**F-statistic:** 56.93

**p-value:** 2.6748137017720305e-25 (essentially 0)

#### Step 5: Decision on Hypothesis

Since p-value < 0.05, we reject the null hypothesis ( $H_0$ ).

*This means there is a significant difference in bike rental demand across different weather conditions.*

#### Step 6: Conclusion & Recommendations

##### Key Insights:

The number of bike rentals is not the same across weather conditions.

Certain weather conditions (e.g., clear days) likely have higher rentals than others (e.g., rainy or foggy days).

##### Recommendations:

##### Increase bike availability on clear days:

More bikes should be deployed when the weather forecast is favorable.

##### Offer discounts on rainy/foggy days:

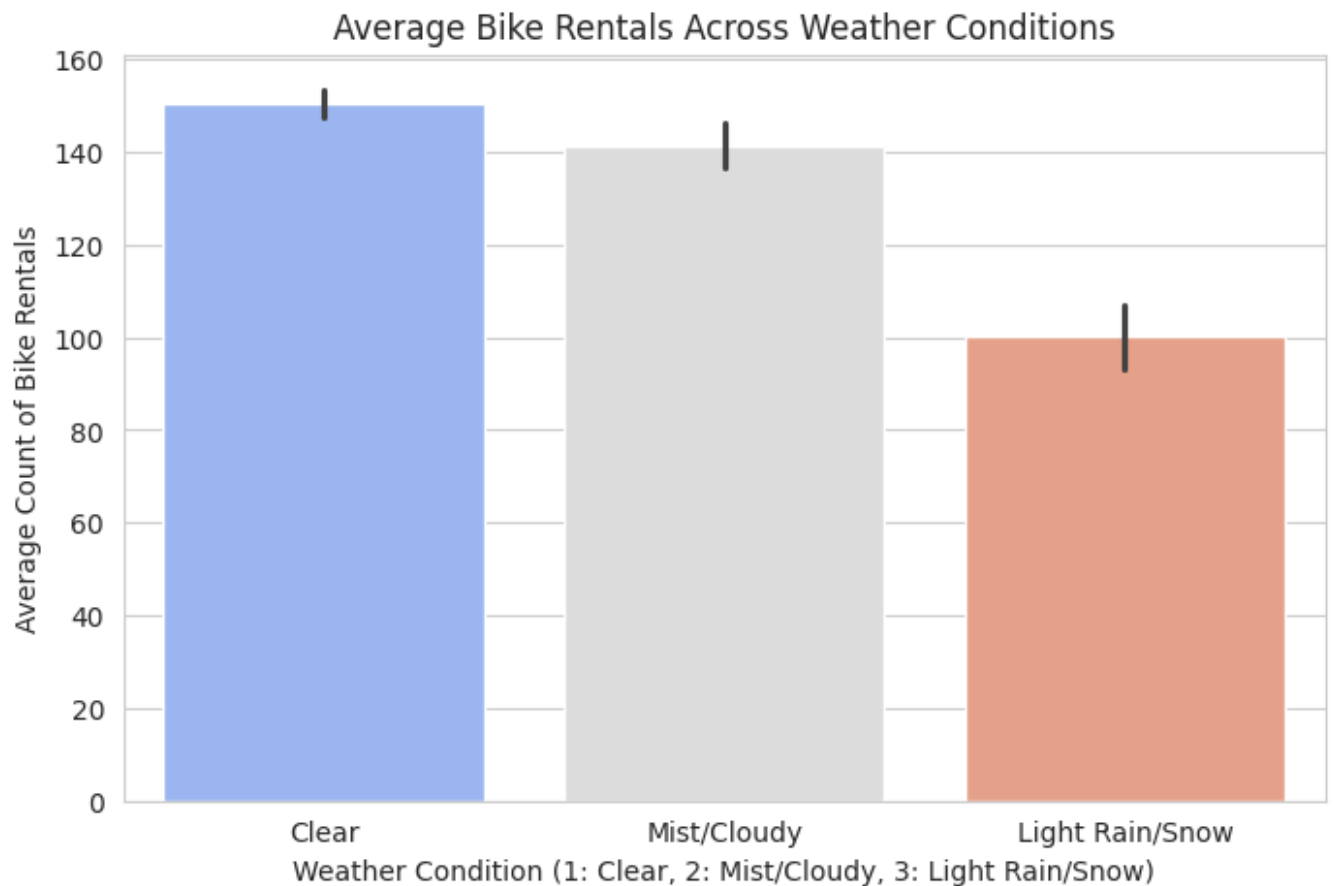
Encourage ridership through promotional offers during bad weather.

##### Enhance weather forecasting integration:

Implement dynamic pricing based on weather conditions.

```
# Plot average bike rentals across different weather conditions
plt.figure(figsize=(8, 5))
sns.barplot(x=df['weather'], y=df['count'], palette='coolwarm')

plt.title('Average Bike Rentals Across Weather Conditions')
plt.xlabel('Weather Condition (1: Clear, 2: Mist/Cloudy, 3: Light Rain/Snow)')
plt.ylabel('Average Count of Bike Rentals')
plt.xticks(ticks=[0, 1, 2], labels=['Clear', 'Mist/Cloudy', 'Light Rain/Snow'])
plt.show()
```



**Observations:**

Clear weather (1) has the highest number of bike rentals.

Mist/Cloudy weather (2) sees a slight drop in demand.

Light Rain/Snow (3) significantly reduces bike rentals.

**Recommendations:****Optimize fleet distribution:**

More bikes should be available on clear days.

**Weather-based promotions:**

Offer discounts or incentives for riding during cloudy or rainy days.

**Real-time updates:**

Notify users of available bikes and promote rides based on upcoming weather conditions.

No. of cycles rented similar or different in different seasons

**Step 1: Define Hypotheses****Null Hypothesis ( $H_0$ ):**

There is no significant difference in bike rental demand across different seasons.

**Alternative Hypothesis ( $H_1$ ):**


There is a significant difference in bike rental demand across different seasons.

**Step 2: Check Assumptions**

Normality using Histogram, Q-Q Plot, Skewness & Kurtosis, and the Shapiro-Wilk test.

Homogeneity of Variance using Levene's test.

```
df.head()
```



	datetime	season	holiday	workingday	weather	temp	humidity	windspeed
0	2011-01-01 00:00:00	1	0	0	1	9.84	81	0.0
1	2011-01-01 01:00:00	1	0	0	1	9.02	80	0.0
2	2011-01-01 01:00:00	1	0	0	1	9.02	80	0.0

```
from scipy.stats import shapiro
```

```
# Group bike rental counts by season
```

```
season_counts = [df[df['season'] == s]['count'] for s in df['season'].unique()]
```


```
# Check normality using Shapiro-Wilk test
```

```
shapiro_results = [shapiro(data) for data in season_counts]
```

```
# Extract p-values for normality test
```

```
shapiro_p_values = [res.pvalue for res in shapiro_results]
```

```
shapiro_p_values
```



```
[np.float64(2.1011452956854775e-40),
 np.float64(3.0480754165659514e-32),
 np.float64(4.7011625622732606e-29),
 np.float64(1.0565909222195703e-31)]
```

## Step 2: Normality Check Results

### Shapiro-Wilk Test p-values:

All p-values are extremely small ( $\approx 0$ ), meaning the data is not normally distributed in any season.

Since normality fails, ANOVA can still be performed because it is robust to normality violations for large sample sizes.

## Step 3: Check Homogeneity of Variance (Levene's Test)

Let's check if the variance of bike rentals is equal across seasons.

```
from scipy.stats import levene

# Perform Levene's test for homogeneity of variance
levene_stat, levene_p = levene(*season_counts)

# Display Levene's test results
levene_stat, levene_p

↵ (np.float64(124.41817087186855), np.float64(5.622706465240217e-79))
```

### Step 3: Levene's Test for Equal Variance

p-value  $\approx 0 \rightarrow$  This means we reject the null hypothesis of equal variance.

#### Conclusion:

The variance of bike rentals is not equal across different seasons.

Even though the assumptions of normality and equal variance fail,

ANOVA is still useful due to the large dataset size. Let's proceed with the ANOVA test.

```
from scipy.stats import f_oneway

# Perform one-way ANOVA test
anova_stat, anova_p = f_oneway(*season_counts)

# Display ANOVA results
anova_stat, anova_p

↵ (np.float64(140.0664990010157), np.float64(1.016251092903105e-88))
```

#### Step 4: One-Way ANOVA Results

p-value: 1.016251092903105e-88 ( $\approx 0$ )

#### Step 5: Decision

Since p-value < 0.05, we reject the null hypothesis.

This means bike rental demand is significantly different across seasons.

#### Step 6: Conclusions & Recommendations

Conclusion: Seasonality has a strong effect on bike rentals.

##### Recommendations:

Increase bike availability in high-demand seasons (most likely summer/spring).

Offer discounts in low-demand seasons to encourage usage.

Use season-specific marketing strategies to boost rentals.

Weather is dependent on season (check between 2 predictor variable)

#### Formulate Hypotheses:

**Null Hypothesis ( $H_0$ ):** Weather conditions are independent of seasons.

**Alternate Hypothesis ( $H_1$ ):** Weather conditions vary significantly across seasons.

Create a Contingency Table to observe the distribution of weather conditions across seasons.

Perform a Chi-Square Test to determine if there is a statistically significant relationship between season and weather.

Interpret the results based on the p-value.

```

from scipy.stats import chi2_contingency

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

# Perform Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

# Display results
chi2_stat, p_value, dof, contingency_table

```

```

↳ (np.float64(57.75760284613354),
  np.float64(1.283153007074087e-10),
  6,
  weather      1      2      3
  season
  1          1541   657   178
  2          1392   604   202
  3          1520   472   169
  4          1422   726   206)

```



**Results:**

Chi-Square Statistic: 57.75

p-value: 1.283153007074087e-10 (very small)

Degrees of Freedom: 6

**Interpretation:** The p-value is much smaller than the significance level  $\alpha = 0.05$ . This means we reject the null hypothesis.

**Conclusion:**

Weather conditions do significantly differ across different seasons.

**Recommendations:****Seasonal Planning:**

Since weather conditions change significantly across seasons, businesses relying on weather (e.g., outdoor activities, transport) should adjust strategies accordingly.

**Infrastructure Readiness:**

Authorities may need to prepare for varying weather conditions each season to ensure smooth city operations.

**Further Analysis:**

Investigate which specific weather categories show the most significant variation across seasons.

```
# Set plot style
plt.figure(figsize=(10, 6))
sns.heatmap(contingency_table, annot=True, fmt="d", cmap="Blues", linewidths=0.5)

# Title and labels
plt.title("Heatmap of Weather Conditions Across Seasons")
plt.xlabel("Weather Condition")
plt.ylabel("Season")

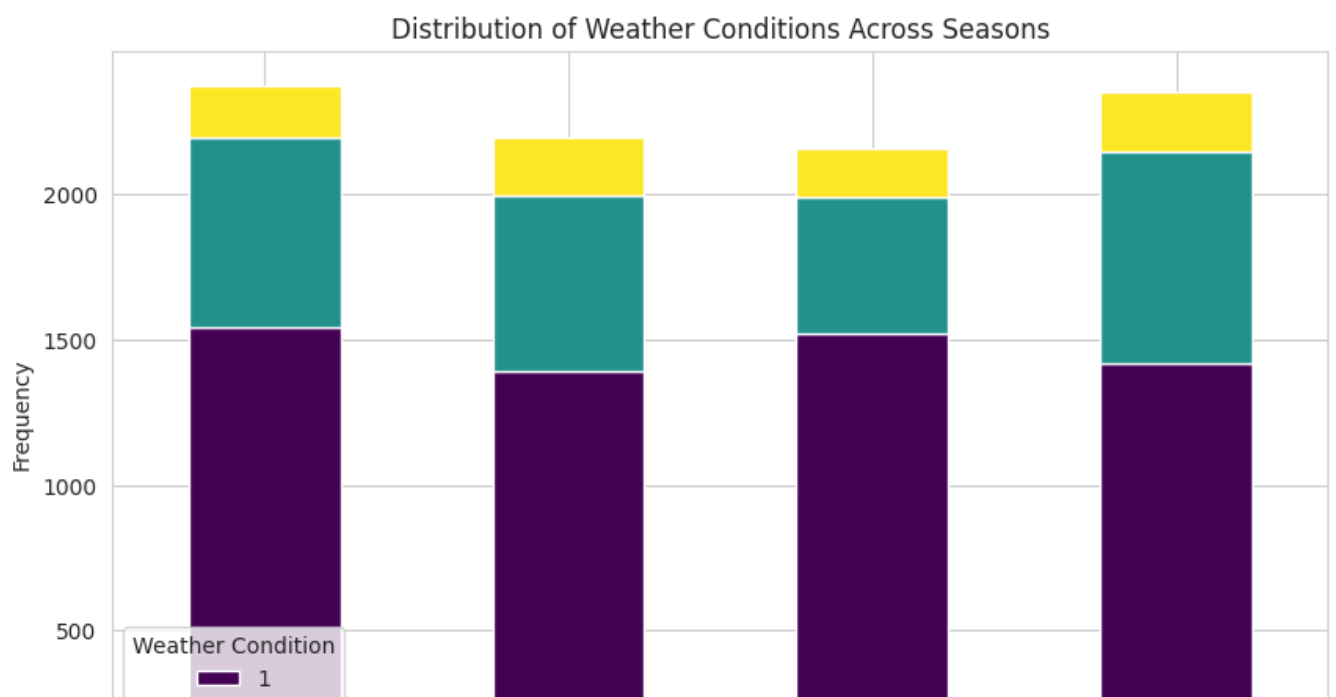
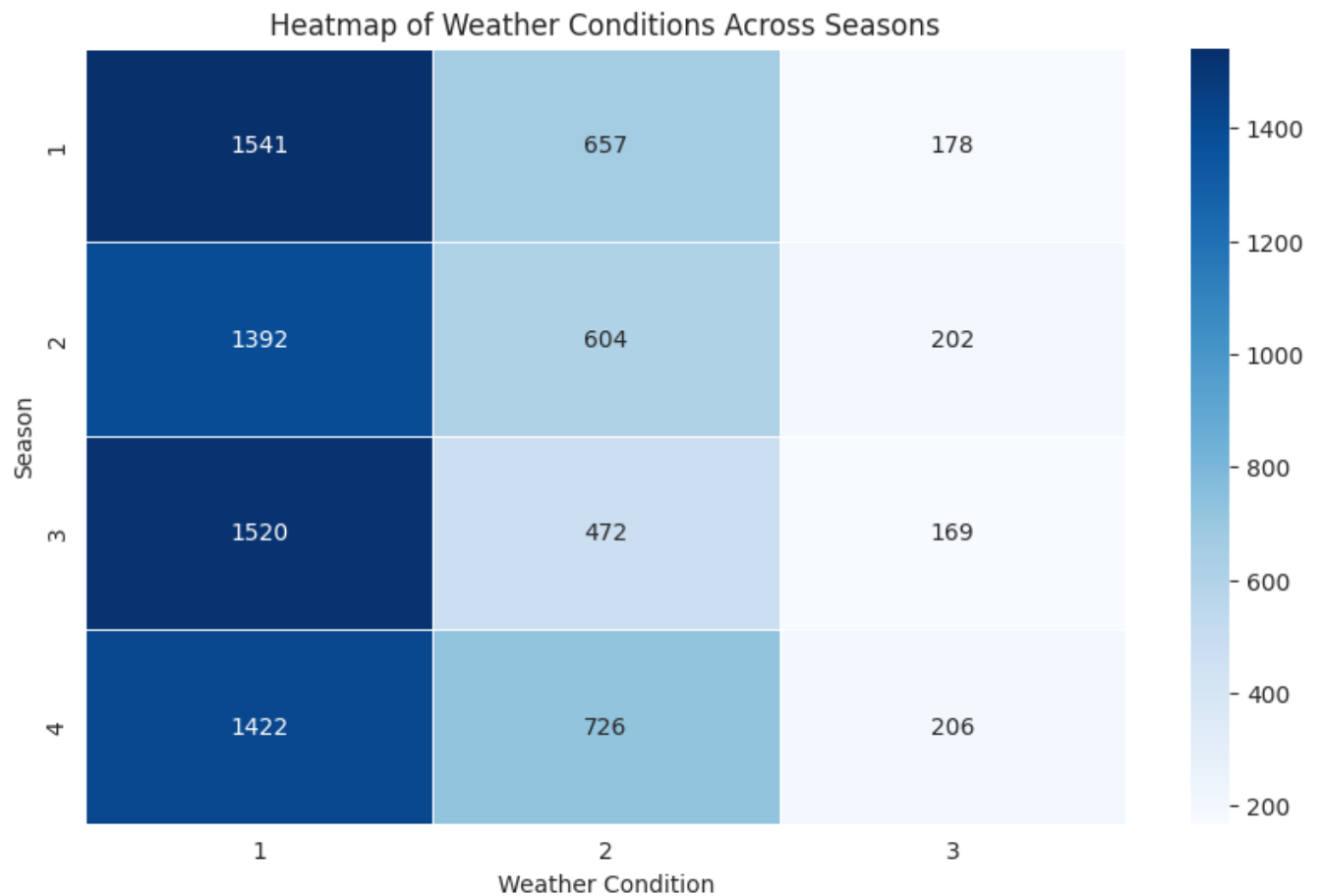
# Show the heatmap
plt.show()

# Bar plot for better comparison
contingency_table.plot(kind="bar", stacked=True, figsize=(10, 6), colormap="vir")

# Title and labels
plt.title("Distribution of Weather Conditions Across Seasons")
```

```
plt.xlabel("Season")
plt.ylabel("Frequency")
plt.legend(title="Weather Condition")
```

```
# Show the bar plot
plt.show()
```





**Here are the visualizations:**

### Heatmap:

Shows the frequency of different weather conditions across seasons.

Darker shades indicate higher counts, making it easier to spot trends.

### Stacked Bar Chart:

Illustrates how weather conditions are distributed within each season.

Highlights seasonal dominance of specific weather types.

From these visualizations, we can clearly see that certain weather conditions are more prevalent in specific seasons.

End of Case study

