Business Case: Yulu - Hypothesis Testing

About Yulu

Yulu is India's leading ***micro-mobility service provider***, which ***offers unique vehicles for the daily commute***. Starting off as a ***mission to eliminate traffic congestion in India***, Yulu provides the safest commute solution through a ***user-friendly mobile app to enable shared, solo and sustainable commuting.***

Yulu zones are located at all the appropriate locations (including ***metro stations, bus stands, office spaces, residential areas, corporate offices, etc***) to make those first and last miles smooth, affordable, and convenient!

Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want ***to understand the factors affecting the demand for these shared electric cycles in the Indian market***.

Problem Statement

The company wants to know:

- **Which variables are significant in predicting the demand for shared electric cycles in the Indian market?**
- **How well those variables describe the electric cycle demands**

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import scipy.stats as spy
```

Reading the dataset

```
df =
pd.read_csv(r"https://d2beiqkhq929f0.cloudfront.net/public_assets/
assets/000/001/428/original/bike_sharing.csv?1642089089")
```

Shape of the dataset

```
df.shape

(10886, 12)
```

Columns in the Dataset

```
df.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather',
'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered',
'count'],
      dtype='object')
```

Basic information about the values present in the dataset

```
df.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
```

```
df.tail()
```

```
                  datetime  season  holiday  workingday  weather
temp  \
10881  2012-12-19 19:00:00       4        0           1        1
15.58
10882  2012-12-19 20:00:00       4        0           1        1
14.76
10883  2012-12-19 21:00:00       4        0           1        1
13.94
10884  2012-12-19 22:00:00       4        0           1        1
13.94
10885  2012-12-19 23:00:00       4        0           1        1
13.12

        atemp  humidity  windspeed  casual  registered  count
10881  19.695        50    26.0027       7         329    336
```

```
10882  17.425        57     15.0013       10          231      241
10883  15.910        61     15.0013        4          164      168
10884  17.425        61      6.0032       12          117      129
10885  16.665        66      8.9981        4           84       88
```

Column Profiling:

- **datetime**: datetime
- **season**: season (1: spring, 2: summer, 3: fall, 4: winter)
- **holiday**: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- **workingday**: if day is neither weekend nor holiday is 1, otherwise is 0.
- **weather**:
- 1: Clear, Few clouds, partly cloudy, partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- **temp**: temperature in Celsius
- **atemp**: feeling temperature in Celsius
- **humidity**: humidity
- **windspeed**: wind speed
- **casual**: count of casual users
- **registered**: count of registered users
- **count**: count of total rental bikes including both casual and registered

Is there any null value in the dataset ?

```
np.any(df.isna())

False
```

Is there any duplicated values in the dataset ?

```
np.any(df.duplicated())

False
```

Datatype of the columns

```
df.dtypes

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
```

Converting the datatype of datetime column from object to datetime

```
df['datetime'] = pd.to_datetime(df['datetime'])
```

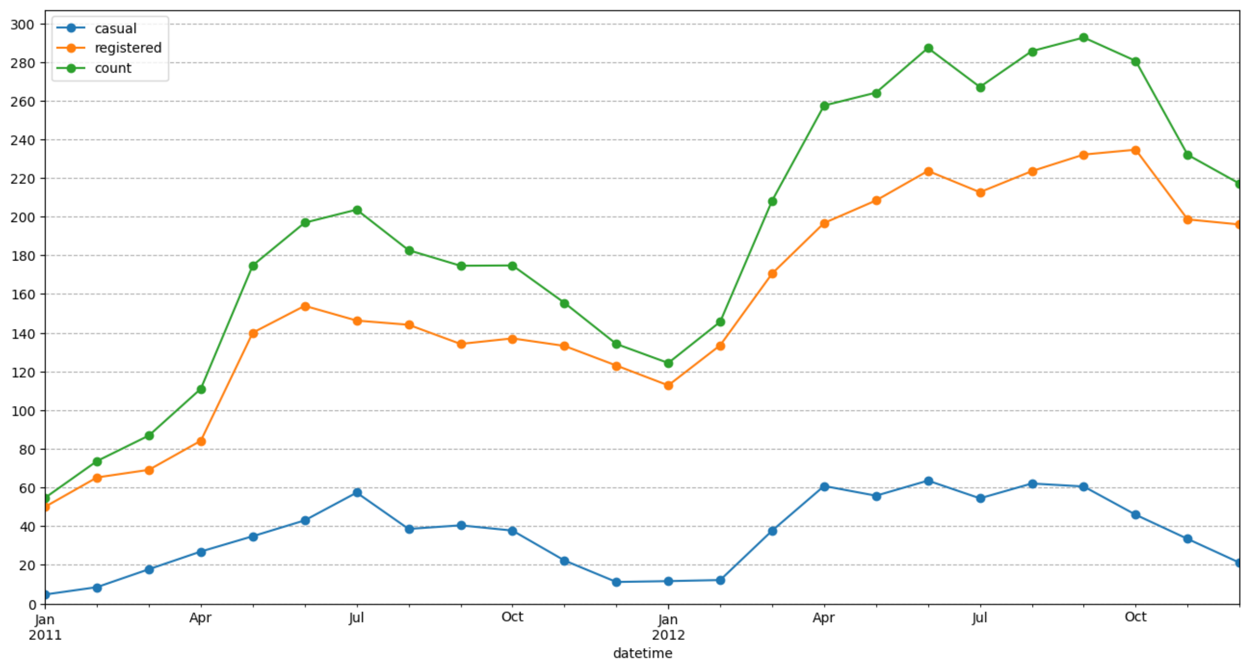
What is the time period for which the data is given ?

```
df['datetime'].min()

Timestamp('2011-01-01 00:00:00')

df['datetime'].max()

Timestamp('2012-12-19 23:00:00')

df['datetime'].max() - df['datetime'].min()

Timedelta('718 days 23:00:00')

df['day'] = df['datetime'].dt.day_name()

# setting the 'datetime' column as the index of the DataFrame 'df'
df.set_index('datetime', inplace = True)

# By setting the 'datetime' column as the index, it allows for easier
and more efficient access,
    # filtering, and manipulation of the data based on the datetime
values.
# It enables operations such as resampling, slicing by specific time
periods, and
    # applying time-based calculations.
```

Slicing Data by Time

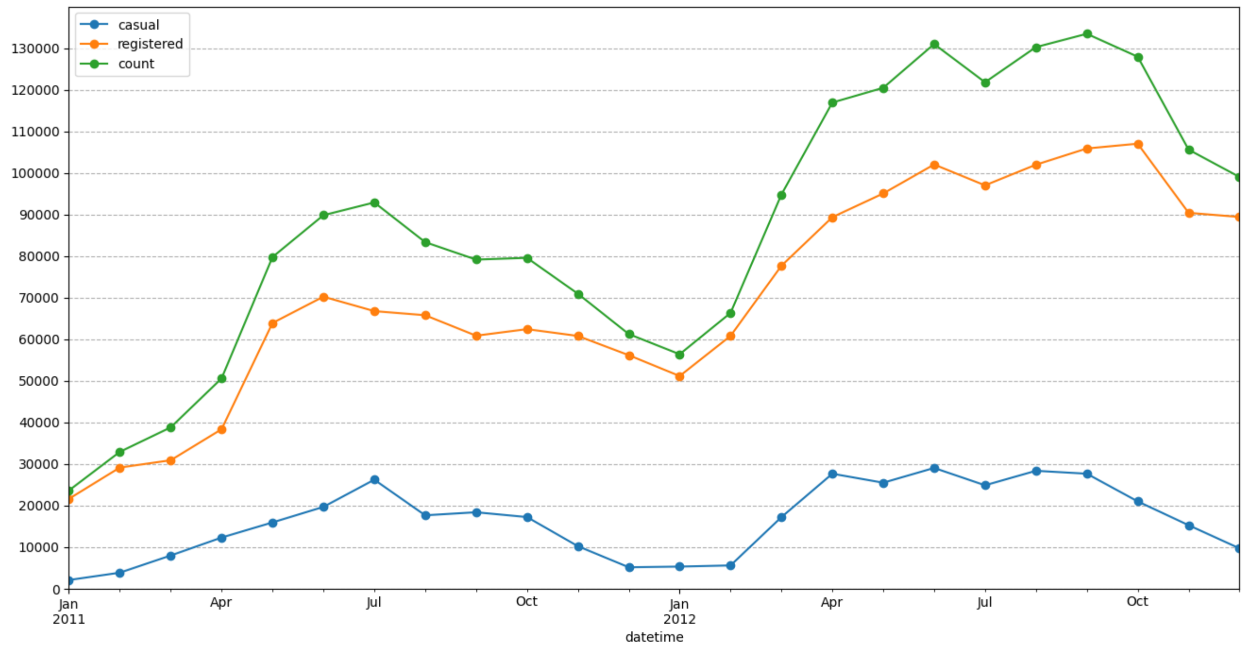
```
# The below code visualizes the trend of the monthly average values
for the 'casual', 'registered',
    # and 'count' variables,  allowing for easy comparison and
analysis of their patterns over time

plt.figure(figsize = (16, 8))

# plotting a lineplot by resampling the data on a monthly basis, and
calculating the mean value
    # of 'casual', 'registered' and 'count' users for each month
df.resample('M')['casual'].mean().plot(kind = 'line', legend =
```

```
'casual', marker = 'o')
df.resample('M')['registered'].mean().plot(kind = 'line', legend =
'registered', marker = 'o')
df.resample('M')['count'].mean().plot(kind = 'line', legend = 'count',
marker = 'o')

plt.grid(axis = 'y', linestyle = '--')    # adding gridlines only
along the y-axis
plt.yticks(np.arange(0, 301, 20))
plt.ylim(0,)    # setting the lower y-axis limit to 0
plt.show()      # displaying the plot
```




```
# The below code visualizes the trend of the monthly total values for
the 'casual', 'registered',
    # and 'count' variables,  allowing for easy comparison and
analysis of their patterns over time

plt.figure(figsize = (16, 8))

# plotting a lineplot by resampling the data on a monthly basis, and
calculating the sum
    # of 'casual', 'registered' and 'count' users for each month
df.resample('M')['casual'].sum().plot(kind = 'line', legend =
'casual', marker = 'o')
df.resample('M')['registered'].sum().plot(kind = 'line', legend =
'registered', marker = 'o')
df.resample('M')['count'].sum().plot(kind = 'line', legend = 'count',
marker = 'o')
```

```
plt.grid(axis = 'y', linestyle = '--')      # adding gridlines only
along the y-axis
plt.yticks(np.arange(0, 130001, 10000))
plt.ylim(0,)         # setting the lower y-axis limit to 0
plt.show()           # displaying the plot
```



I want to know if there is an increase in the average hourly count of rental bikes from the year 2011 to 2012

```
# resampling the DataFrame by the year
df1 = df.resample('Y')['count'].mean().to_frame().reset_index()

# Create a new column 'prev_count' by shifting the 'count' column one
position up
    # to compare the previous year's count with the current year's
count
df1['prev_count'] = df1['count'].shift(1)

# Calculating the growth percentage of 'count' with respect to the
'count' of previous year
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 /
df1['prev_count']
df1
```

	datetime	count	prev_count	growth_percent
0	2011-12-31	144.223349	NaN	NaN
1	2012-12-31	238.560944	144.223349	65.410764

- This data suggests that there was substantial growth in the count of the variable over the course of one year.

- The mean total hourly count of rental bikes is 144 for the year 2011 and 239 for the year 2012. An annual growth rate of 65.41 % can be seen in the demand of electric vehicles on an hourly basis.

It indicates positive growth and potentially a successful outcome or increasing demand for the variable being measured.

```
df.reset_index(inplace = True)
```

How does the average hourly count of rental bikes varies for different month ?

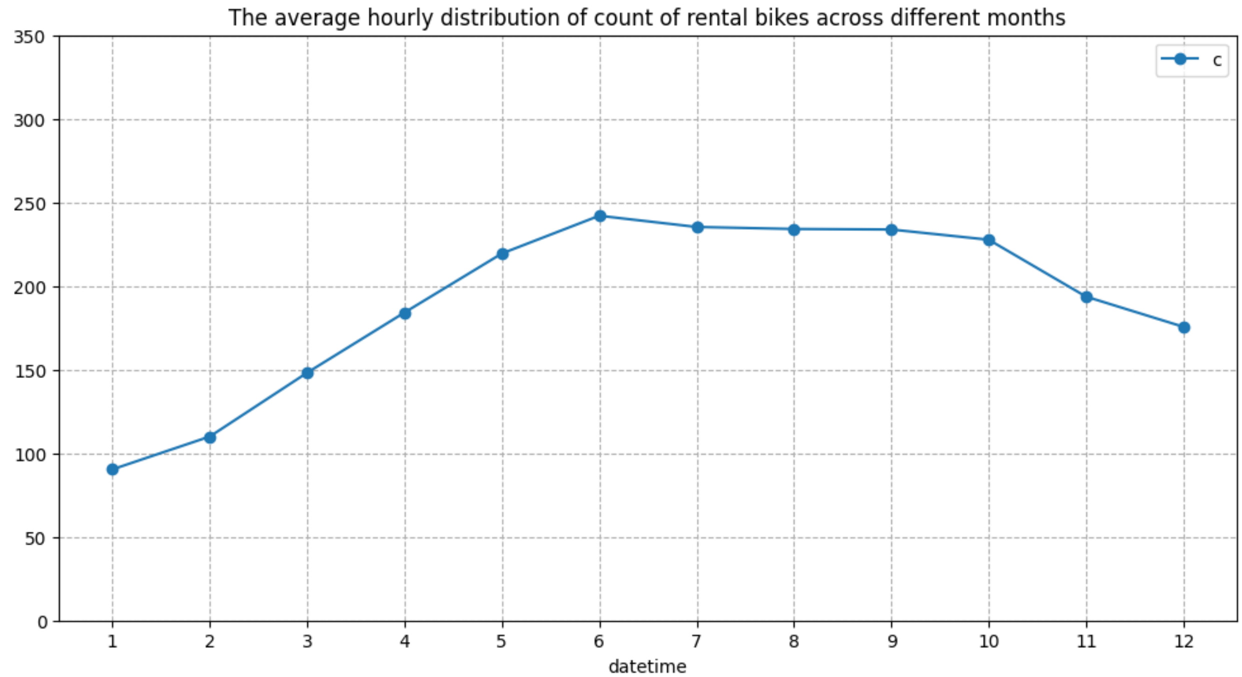
```
# Grouping the DataFrame by the month
df1 = df.groupby(by = df['datetime'].dt.month)
['count'].mean().reset_index()
df1.rename(columns = {'datetime' : 'month'}, inplace = True)

# Create a new column 'prev_count' by shifting the 'count' column one
position up
    # to compare the previous month's count with the current month's
count
df1['prev_count'] = df1['count'].shift(1)

# Calculating the growth percentage of 'count' with respect to the
'count' of previous month
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 /
df1['prev_count']
df1.set_index('month', inplace = True)
df1
```

month	count	prev_count	growth_percent
1	90.366516	NaN	NaN
2	110.003330	90.366516	21.730188
3	148.169811	110.003330	34.695751
4	184.160616	148.169811	24.290241
5	219.459430	184.160616	19.167406
6	242.031798	219.459430	10.285440
7	235.325658	242.031798	-2.770768
8	234.118421	235.325658	-0.513007
9	233.805281	234.118421	-0.133753
10	227.699232	233.805281	-2.611596
11	193.677278	227.699232	-14.941620
12	175.614035	193.677278	-9.326465

- The count of rental bikes shows an increasing trend from January to March, with a significant growth rate of 34.70% between February and March.
- The growth rate starts to stabilize from April to June, with a relatively smaller growth rate.
- From July to September, there is a slight decrease in the count of rental bikes, with negative growth rates.

- The count further declines from October to December, with the largest drop observed between October and November (-14.94%).

```
# The resulting plot visualizes the average hourly distribution of the count of rental bikes for each
    # month, allowing for comparison and identification of any patterns or trends throughout the year.

# Setting the figure size for the plot
plt.figure(figsize = (12, 6))

# Setting the title for the plot
plt.title("The average hourly distribution of count of rental bikes across different months")

# Grouping the DataFrame by the month and calculating the mean of the 'count' column for each month.
    # Ploting the line graph using markers ('o') to represent the average count per month.
df.groupby(by = df['datetime'].dt.month)['count'].mean().plot(kind = 'line', marker = 'o')

plt.ylim(0,)    # Setting the y-axis limits to start from zero
plt.xticks(np.arange(1, 13))   # Setting the x-ticks to represent the months from 1 to 12
plt.legend('count')    # Adding a legend to the plot for the 'count' line.
plt.yticks(np.arange(0, 400, 50))
# Adding gridlines to both the x and y axes with a dashed line style
plt.grid(axis = 'both', linestyle = '--')
plt.plot()     # Displaing the plot.

[]
```



- The average hourly count of rental bikes is the highest in the month of June followed by July and August.
- The average hourly count of rental bikes is the lowest in the month of January followed by February and March.

Overall, these trends suggest a seasonal pattern in the count of rental bikes, with higher demand during the spring and summer months, a slight decline in the fall, and a further decrease in the winter months. It could be useful for the rental bike company to consider these patterns for resource allocation, marketing strategies, and operational planning throughout the year.

What is the distribution of average count of rental bikes on an hourly basis in a single day ?

```
# Grouping the DataFrame by the hour
df1 = df.groupby(by = df['datetime'].dt.hour)
['count'].mean().reset_index()
df1.rename(columns = {'datetime' : 'hour'}, inplace = True)

# Create a new column 'prev_count' by shifting the 'count' column one
position up
    # to compare the previous hour's count with the current hour's
count
df1['prev_count'] = df1['count'].shift(1)

# Calculating the growth percentage of 'count' with respect to the
'count' of previous hour
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 /
df1['prev_count']
```

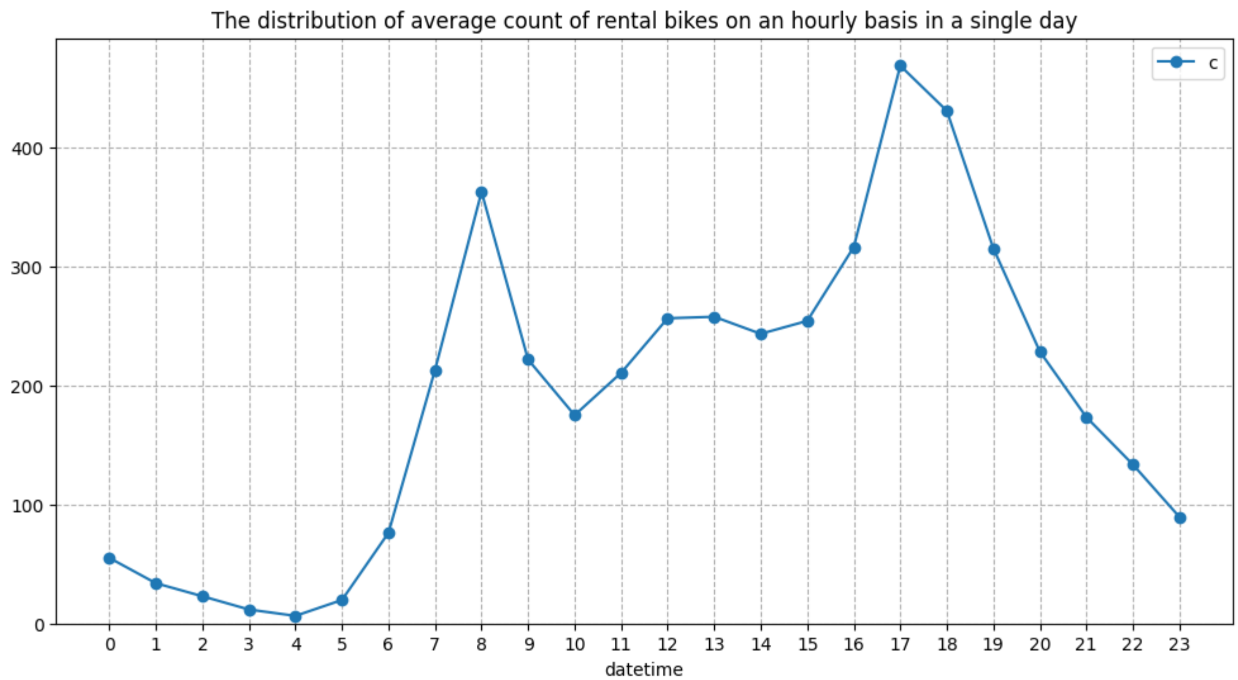
```python
df1.set_index('hour', inplace = True)
df1
```

hour	count	prev_count	growth_percent
0	55.138462	NaN	NaN
1	33.859031	55.138462	-38.592718
2	22.899554	33.859031	-32.367959
3	11.757506	22.899554	-48.656179
4	6.407240	11.757506	-45.505110
5	19.767699	6.407240	208.521293
6	76.259341	19.767699	285.777526
7	213.116484	76.259341	179.462793
8	362.769231	213.116484	70.221104
9	221.780220	362.769231	-38.864655
10	175.092308	221.780220	-21.051432
11	210.674725	175.092308	20.322091
12	256.508772	210.674725	21.755835
13	257.787281	256.508772	0.498427
14	243.442982	257.787281	-5.564393
15	254.298246	243.442982	4.459058
16	316.372807	254.298246	24.410141
17	468.765351	316.372807	48.168661
18	430.859649	468.765351	-8.086285
19	315.278509	430.859649	-26.825705
20	228.517544	315.278509	-27.518833
21	173.370614	228.517544	-24.132471
22	133.576754	173.370614	-22.953059
23	89.508772	133.576754	-32.990757

- During the early morning hours (hours 0 to 5), there is a significant decrease in the count, with negative growth percentages ranging from -38.59% to -48.66%.
- However, starting from hour 5, there is a sudden increase in count, with a sharp positive growth percentage of 208.52% observed from hour 4 to hour 5.
- The count continues to rise significantly until reaching its peak at hour 17, with a growth percentage of 48.17% compared to the previous hour.
- After hour 17, there is a gradual decrease in count, with negative growth percentages ranging from -8.08% to -32.99% during the late evening and nighttime hours.

```python
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on an hourly basis in a single day")
df.groupby(by = df['datetime'].dt.hour)['count'].mean().plot(kind = 'line', marker = 'o')
plt.ylim(0,)
plt.xticks(np.arange(0, 24))
plt.legend('count')
plt.grid(axis = 'both', linestyle = '--')
plt.plot()
```

```
[ ]
```



- The average count of rental bikes is the highest at 5 PM followed by 6 PM and 8 AM of the day.
- The average count of rental bikes is the lowest at 4 AM followed by 3 AM and 5 AM of the day.

These patterns indicate that there is a distinct fluctuation in count throughout the day, with low counts during early morning hours, a sudden increase in the morning, a peak count in the afternoon, and a gradual decline in the evening and nighttime.

Basic Information about the Dataset

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 13 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
 12  day          10886 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(8), object(1)
memory usage: 1.1+ MB
```

- The dataframe requires a memory usage of about 1.1+ MB.
- Though the memory usage is small but can we still decrease the memory usage ?

```
# 1: spring, 2: summer, 3: fall, 4: winter
def season_category(x):
    if x == 1:
        return 'spring'
    elif x == 2:
        return 'summer'
    elif x == 3:
        return 'fall'
    else:
        return 'winter'
df['season'] = df['season'].apply(season_category)
```

Optimizing Memory Usage of the Dataframe

Updating dtype of season column

```
print('Memory usage of season column : ', df['season'].memory_usage())

# Since the dtype of season column is object, we can convert the dtype
to category to save memory
df['season'] = df['season'].astype('category')
print('Updated Memory usage of season column : ',
df['season'].memory_usage())

Memory usage of season column :  87220
Updated Memory usage of season column :  11222
```

Updating dtype of holiday column

```
print('Max value entry in holiday column : ', df['holiday'].max())

print('Memory usage of holiday column : ',
df['holiday'].memory_usage())
# Since the maximum entry in holiday column is 1 and the dtype is
int64, we can convert the dtype to category to save memory
df['holiday'] = df['holiday'].astype('category')
print('Updated Memory usage of holiday column : ',
df['holiday'].memory_usage())
```

```
Max value entry in holiday column :  1
Memory usage of holiday column :  87220
Updated Memory usage of holiday column :  11142
```

Updating dtype of workingday column

```python
print('Max value entry in workingday column : ',
df['workingday'].max())
print('Memory usage of workingday column : ',
df['workingday'].memory_usage())
# Since the maximum entry in workingday column is 1 and the dtype is
int64, we can convert the dtype to category to save memory
df['workingday'] = df['workingday'].astype('category')
print('Updated Memory usage of workingday column : ',
df['workingday'].memory_usage())
```

```
Max value entry in workingday column :  1
Memory usage of workingday column :  87220
Updated Memory usage of workingday column :  11142
```

Updating dtype of weather column

```python
print('Max value entry in weather column : ', df['weather'].max())

print('Memory usage of weather column : ',
df['weather'].memory_usage())
# Since the maximum entry in weather column is 4 and the dtype is
int64, we can convert the dtype to category to save memory
df['weather'] = df['weather'].astype('category')
print('Updated Memory usage of weather column : ',
df['weather'].memory_usage())
```

```
Max value entry in weather column :  4
Memory usage of weather column :  87220
Updated Memory usage of weather column :  11222
```

Updating dtype of temp column

```python
print('Max value entry in temp column : ', df['temp'].max())
print('Memory usage of temp column : ', df['temp'].memory_usage())
# Since the maximum entry in temp column is 41.0 and the dtype is
float64, we can convert the dtype to float32 to save memory
df['temp'] = df['temp'].astype('float32')
print('Updated Memory usage of temp column : ',
df['temp'].memory_usage())
```

```
Max value entry in temp column :  41.0
Memory usage of temp column :  87220
Updated Memory usage of temp column :  43676
```

Updating dtype of atemp column

```
print('Max value entry in atemp column : ', df['atemp'].max())

print('Memory usage of atemp column : ', df['atemp'].memory_usage())
# Since the maximum entry in atemp column is 45.455 and the dtype is
float64, we can convert the dtype to float32 to save memory
df['atemp'] = df['atemp'].astype('float32')
print('Updated Memory usage of atemp column : ',
df['atemp'].memory_usage())
```

```
Max value entry in atemp column :  45.455
Memory usage of atemp column :  87220
Updated Memory usage of atemp column :  43676
```

Updating dtype of humidity column

```
print('Max value entry in humidity column : ', df['humidity'].max())

print('Memory usage of humidity column : ', df['temp'].memory_usage())

# Since the maximum entry in humidity column is 100 and the dtype is
int64, we can convert the dtype to int8 to save memory
df['humidity'] = df['humidity'].astype('int8')
print('Updated Memory usage of humidity column : ',
df['humidity'].memory_usage())
```

```
Max value entry in humidity column :  100
Memory usage of humidity column :  43676
Updated Memory usage of humidity column :  11018
```

Updating dtype of windspeed column

```
print('Max value entry in windspeed column : ', df['windspeed'].max())

print('Memory usage of windspeed column : ',
df['windspeed'].memory_usage())
# Since the maximum entry in windspeed column is 56.9969 and the dtype
is float64, we can convert the dtype to float32 to save memory
df['windspeed'] = df['windspeed'].astype('float32')
print('Updated Memory usage of windspeed column : ',
df['windspeed'].memory_usage())
```

```
Max value entry in windspeed column :  56.9969
Memory usage of windspeed column :  87220
Updated Memory usage of windspeed column :  43676
```

Updating dtype of casual column

```
print('Max value entry in casual column : ', df['casual'].max())

print('Memory usage of casual column : ', df['casual'].memory_usage())
```

```
# Since the maximum entry in casual column is 367 and the dtype is int64, we can convert the dtype to int16 to save memory
df['casual'] = df['casual'].astype('int16')
print('Updated Memory usage of casual column : ', df['casual'].memory_usage())
```

```
Max value entry in casual column :  367
Memory usage of casual column :  87220
Updated Memory usage of casual column :  21904
```

Updating dtype of registered column

```
print('Max value entry in registered column : ', df['registered'].max())
print('Memory usage of registered column : ', df['registered'].memory_usage())
# Since the maximum entry in registered column is 886 and the dtype is int64, we can convert the dtype to int16 to save memory
df['registered'] = df['registered'].astype('int16')
print('Updated Memory usage of registered column : ', df['registered'].memory_usage())
```

```
Max value entry in registered column :  886
Memory usage of registered column :  87220
Updated Memory usage of registered column :  21904
```

Updating dtype of count column

```
print('Max value entry in count column : ', df['count'].max())

print('Memory usage of count column : ', df['count'].memory_usage())
# Since the maximum entry in count column is 977 and the dtype is int64, we can convert the dtype to int16 to save memory
df['count'] = df['count'].astype('int16')
print('Updated Memory usage of count column : ', df['count'].memory_usage())
```

```
Max value entry in count column :  977
Memory usage of count column :  87220
Updated Memory usage of count column :  21904
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  category
```

```
 2   holiday     10886 non-null  category
 3   workingday  10886 non-null  category
 4   weather     10886 non-null  category
 5   temp        10886 non-null  float32
 6   atemp       10886 non-null  float32
 7   humidity    10886 non-null  int8
 8   windspeed   10886 non-null  float32
 9   casual      10886 non-null  int16
 10  registered  10886 non-null  int16
 11  count       10886 non-null  int16
 12  day         10886 non-null  object
dtypes: category(4), datetime64[ns](1), float32(3), int16(3), int8(1), 
object(1)
memory usage: 415.4+ KB
```

Earlier the dataset was using 1.1+ MB of memory but now it has been reduced to 415.2+ KB. Around 63.17 % reduction in the memory usage.

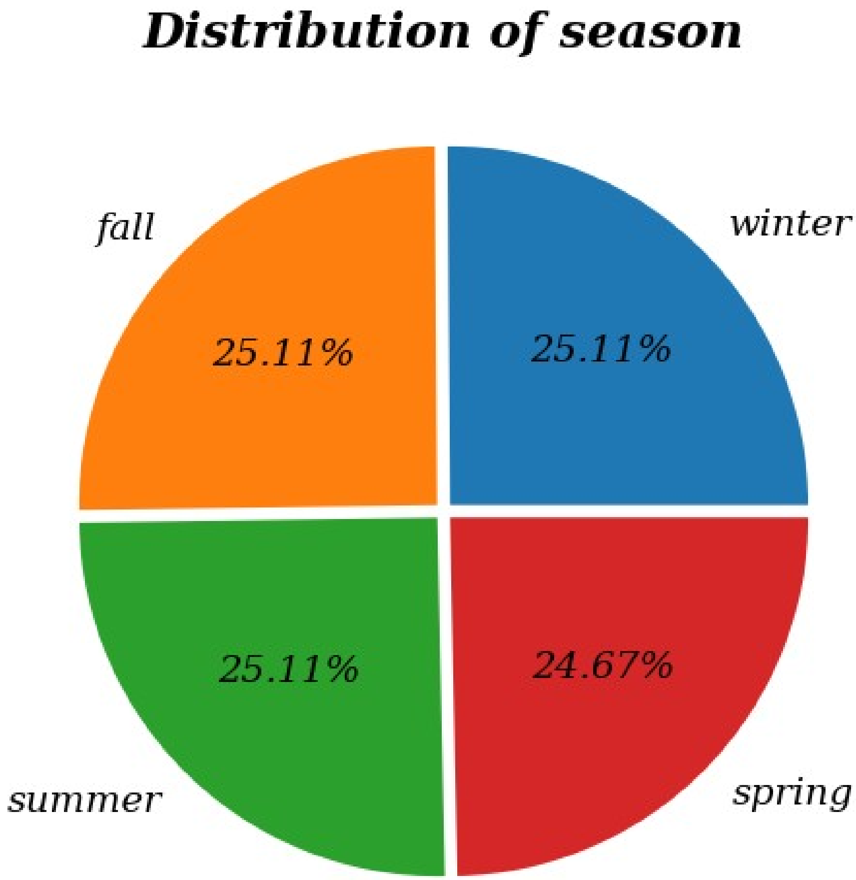
Basic Description of the dataset

```
df.describe()

                            datetime          temp         atemp  \
count                          10886  10886.000000  10886.000000   
mean   2011-12-27 05:56:22.399411968     20.230862     23.655085   
min              2011-01-01 00:00:00      0.820000      0.760000   
25%              2011-07-02 07:15:00     13.940000     16.665001   
50%              2012-01-01 20:30:00     20.500000     24.240000   
75%              2012-07-01 12:45:00     26.240000     31.059999   
max              2012-12-19 23:00:00     41.000000     45.455002   
std                              NaN      7.791590      8.474601   

           humidity     windspeed        casual    registered
count  
count  10886.000000  10886.000000  10886.000000  10886.000000  
10886.000000  
mean      61.886460     12.799396     36.021955    155.552177  
191.574132  
min        0.000000      0.000000      0.000000      0.000000  
1.000000  
25%       47.000000      7.001500      4.000000     36.000000  
42.000000  
50%       62.000000     12.998000     17.000000    118.000000  
145.000000  
75%       77.000000     16.997900     49.000000    222.000000  
284.000000  
max      100.000000     56.996899    367.000000    886.000000  
977.000000  
std       19.245033      8.164537     49.960477    151.039033  
181.144454
```

- These statistics provide insights into the central tendency, spread, and range of the numerical features in the dataset.

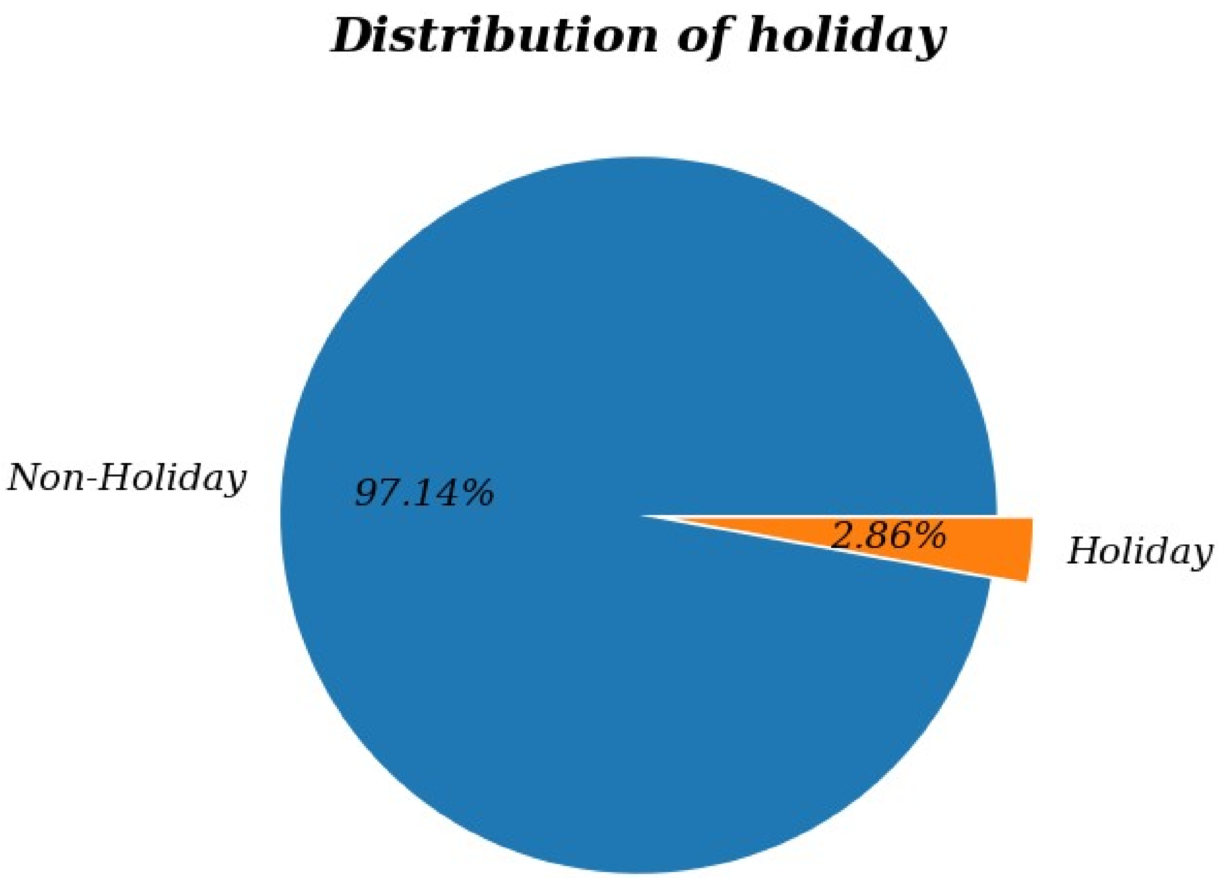
```
np.round(df['season'].value_counts(normalize = True) * 100, 2)

season
winter    25.11
fall      25.11
summer    25.11
spring    24.67
Name: proportion, dtype: float64

np.round(df['holiday'].value_counts(normalize = True) * 100, 2)

holiday
0    97.14
1     2.86
Name: proportion, dtype: float64

np.round(df['workingday'].value_counts(normalize = True) * 100, 2)

workingday
1    68.09
0    31.91
Name: proportion, dtype: float64

np.round(df['weather'].value_counts(normalize = True) * 100, 2)

weather
1    66.07
2    26.03
3     7.89
4     0.01
Name: proportion, dtype: float64

plt.figure(figsize=(6, 6))
plt.title('Distribution of season', fontdict={'fontsize': 18,
                                              'fontweight': 600,
                                              'fontstyle': 'oblique',
                                              'fontfamily': 'serif'})

# Counting the occurrences of each season
season_counts = df['season'].value_counts(normalize=True)

# Creating the pie-chart
plt.pie(x=season_counts,
        explode=[0.025, 0.025, 0.025, 0.025],
        labels=season_counts.index,
        autopct='%.2f%%',
        textprops={'fontsize': 14,
                   'fontstyle': 'oblique',
                   'fontfamily': 'serif',
```

```
                    'fontweight': 500})

plt.show()  # Displaying the plot
```



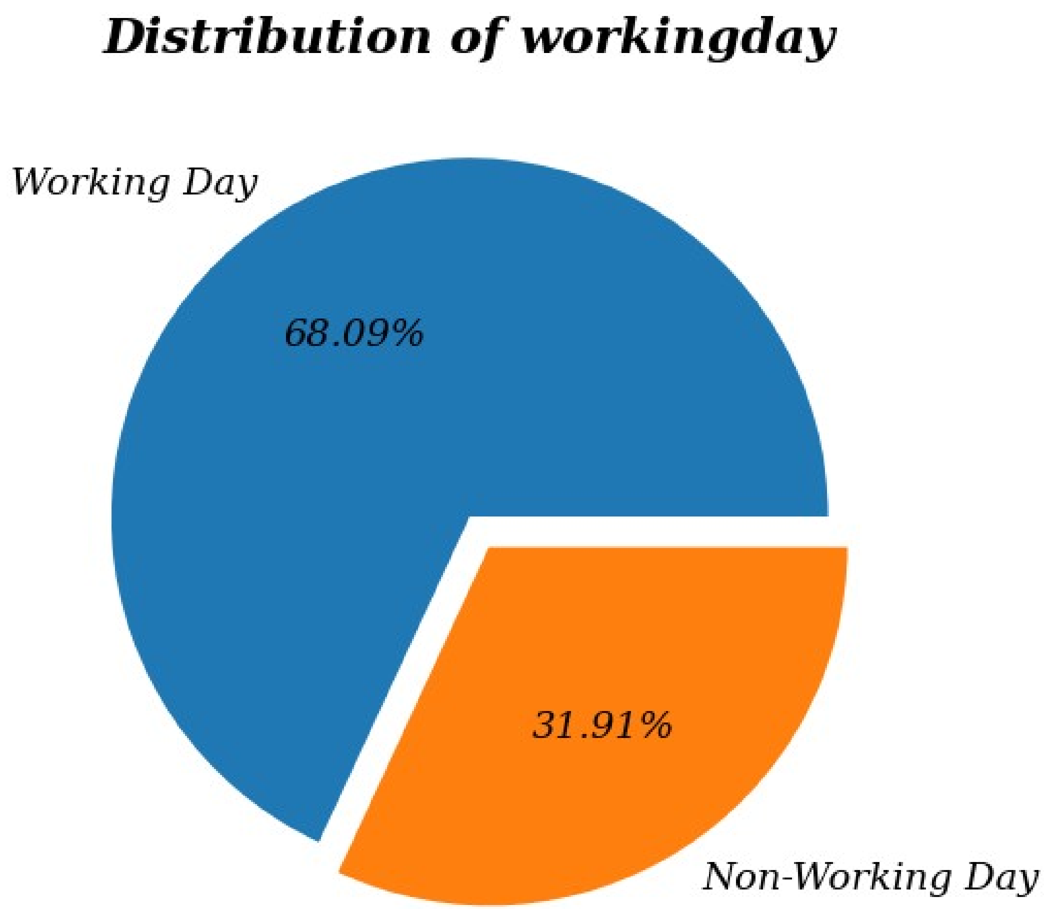

```
print(df['season'].unique())
print(df['season'].dtype)

['spring', 'summer', 'fall', 'winter']
Categories (4, object): ['fall', 'spring', 'summer', 'winter']
category

plt.figure(figsize=(6, 6))
plt.title('Distribution of holiday', fontdict={'fontsize': 18,
                                               'fontweight': 600,
                                               'fontstyle': 'oblique',
                                               'fontfamily': 'serif'})

# Counting the occurrences of each holiday value
holiday_counts = df['holiday'].value_counts(normalize=True)
```

```
# Creating the pie-chart
plt.pie(x=holiday_counts,
        explode=[0, 0.1],
        labels=['Non-Holiday', 'Holiday'],
        autopct='%.2f%%',
        textprops={'fontsize': 14,
                   'fontstyle': 'oblique',
                   'fontfamily': 'serif',
                   'fontweight': 500})

plt.show()  # Displaying the plot
```



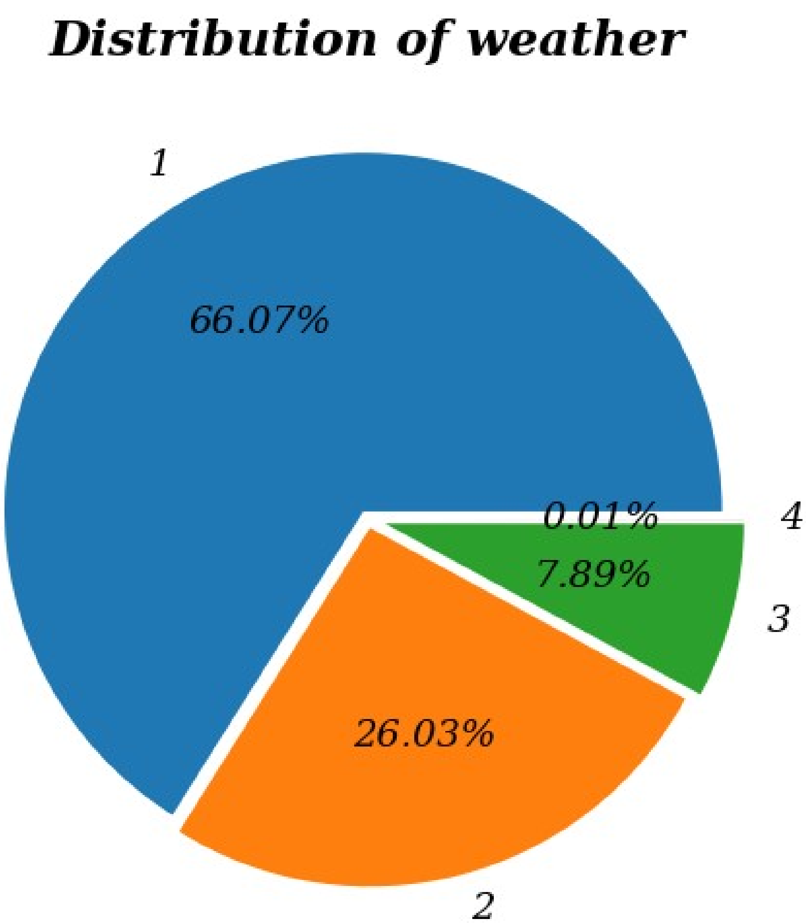

```
plt.figure(figsize=(6, 6))
plt.title('Distribution of workingday', fontdict={'fontsize': 18,
                                                  'fontweight': 600,
                                                  'fontstyle':
'oblique',
                                                  'fontfamily':
'serif'})
```

```
# Counting the occurrences of each workingday value
workingday_counts = df['workingday'].value_counts(normalize=True)

# Creating the pie-chart
plt.pie(x=workingday_counts,
        explode=[0, 0.1],
        labels=['Working Day', 'Non-Working Day'],
        autopct='%.2f%%',
        textprops={'fontsize': 14,
                   'fontstyle': 'oblique',
                   'fontfamily': 'serif',
                   'fontweight': 500})

plt.show()  # Displaying the plot
```



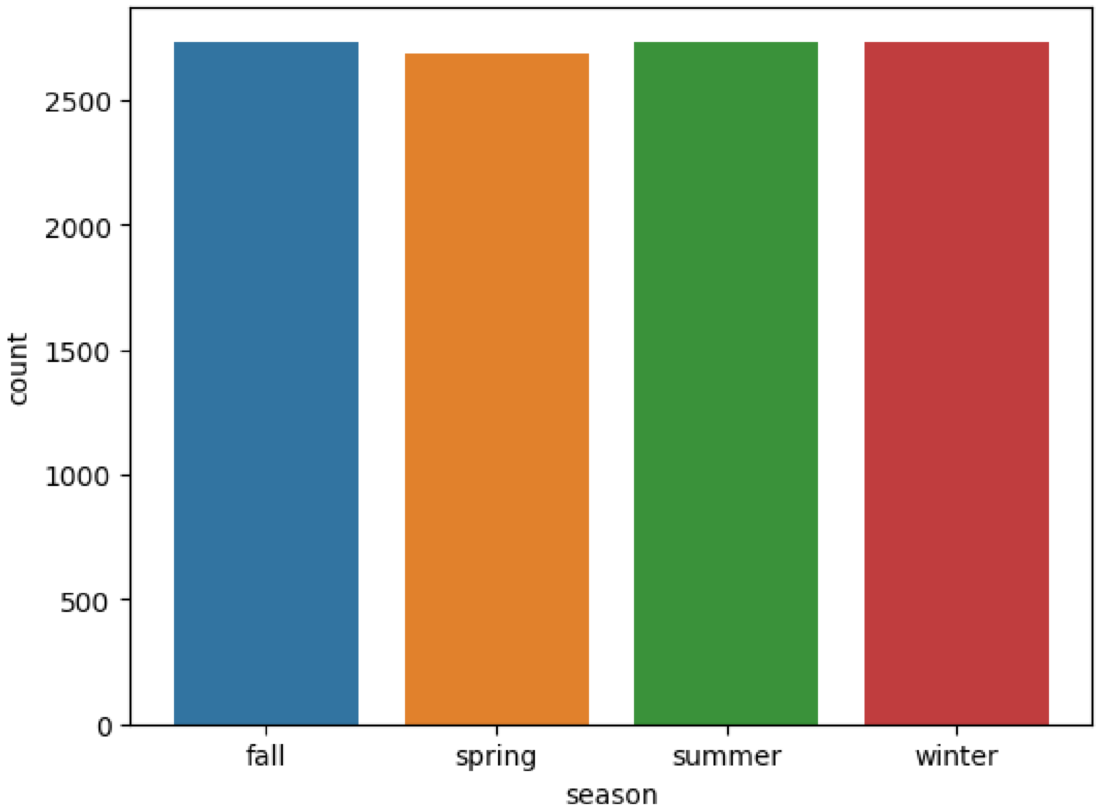

```
plt.figure(figsize=(6, 6))
plt.title('Distribution of weather', fontdict={'fontsize': 18,
                                                'fontweight': 600,
```

```
                                                              'fontstyle':
'oblique',
                                                              'fontfamily':
'serif'})

# Counting the occurrences of each weather value
weather_counts = df['weather'].value_counts(normalize=True)

# Creating the pie-chart
plt.pie(x=weather_counts,
        explode=[0.025, 0.025, 0.05, 0.05],
        labels=weather_counts.index,
        autopct='%.2f%%',
        textprops={'fontsize': 14,
                   'fontstyle': 'oblique',
                   'fontfamily': 'serif',
                   'fontweight': 500})

plt.show()  # Displaying the plot
```

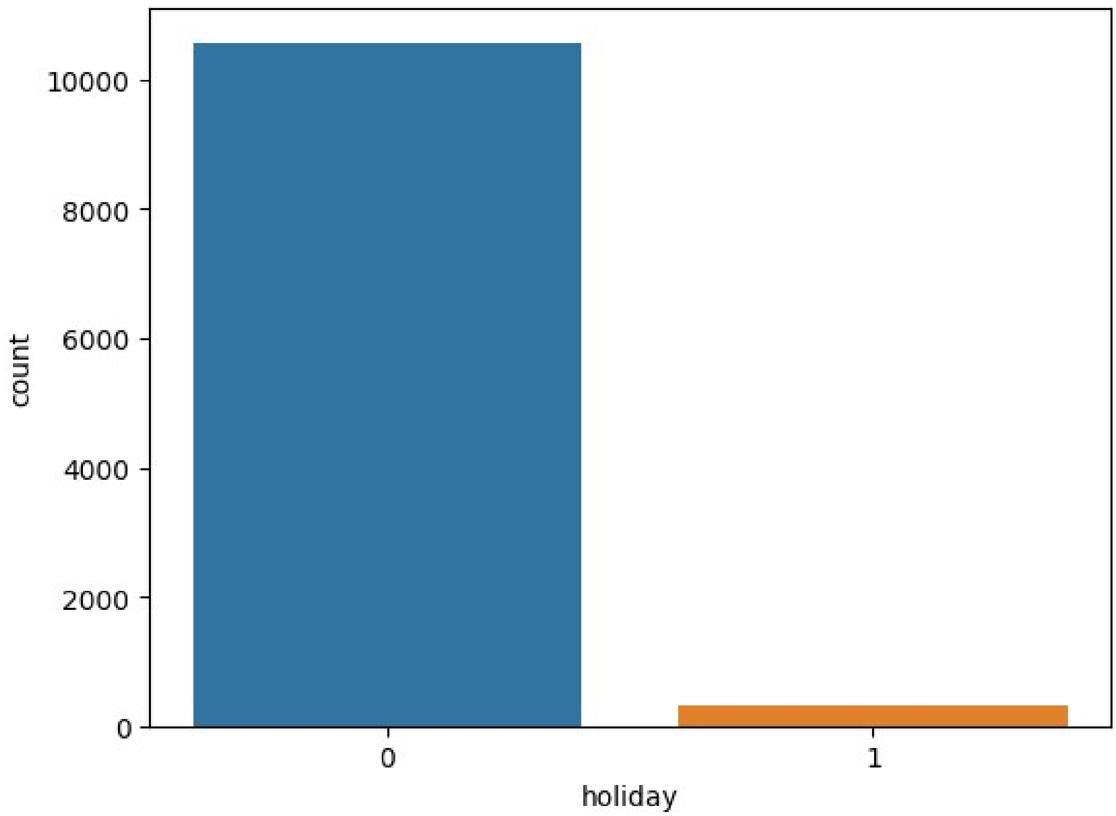


Univariate Analysis

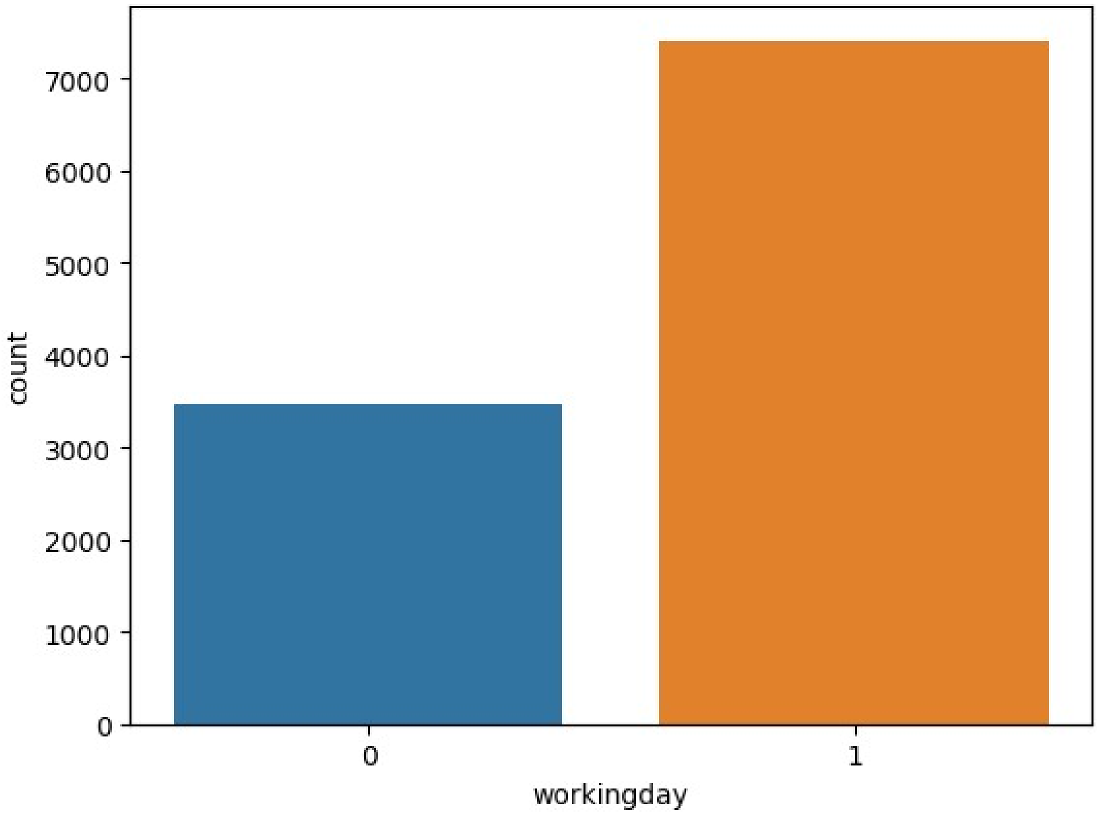
```
# The below code generates a visually appealing count plot to showcase the
    # distribution of season in the dataset
sns.countplot(data = df, x = 'season')
plt.plot()   # displaying the plot

[]
```



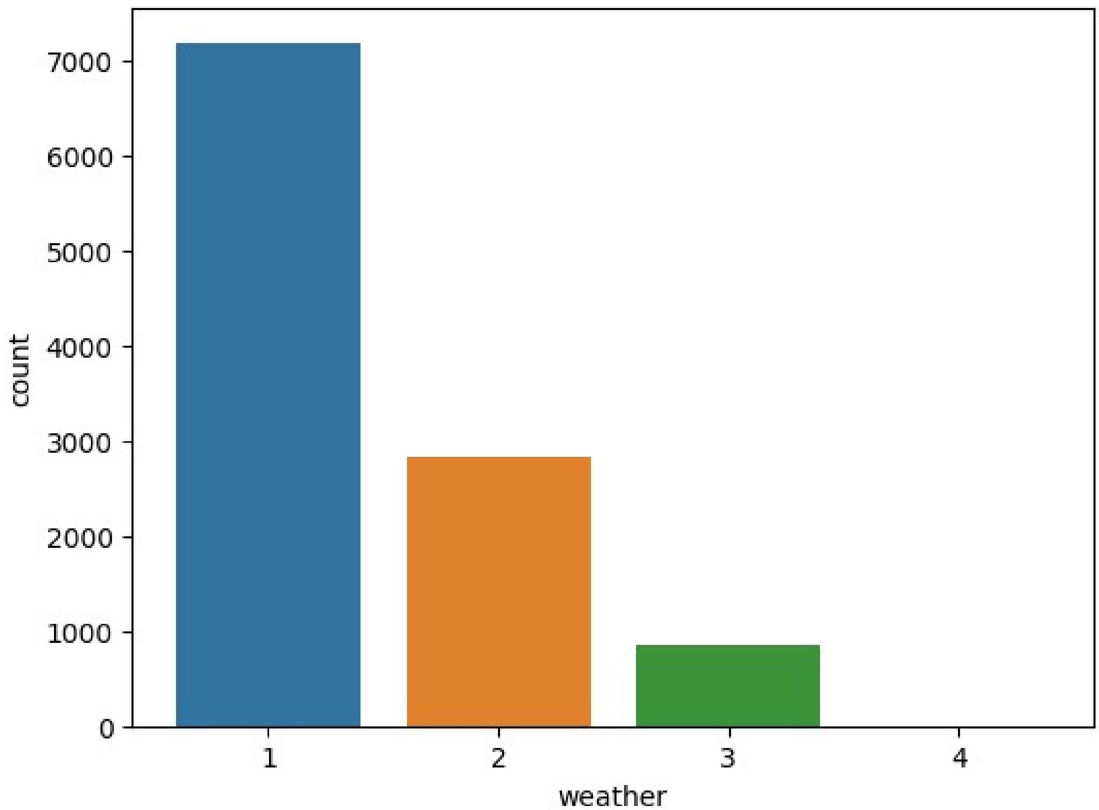

```
# The below code generates a visually appealing count plot to showcase the
    # distribution of holiday in the dataset

sns.countplot(data = df, x = 'holiday')
plt.plot()       # displaying the chart

[]
```



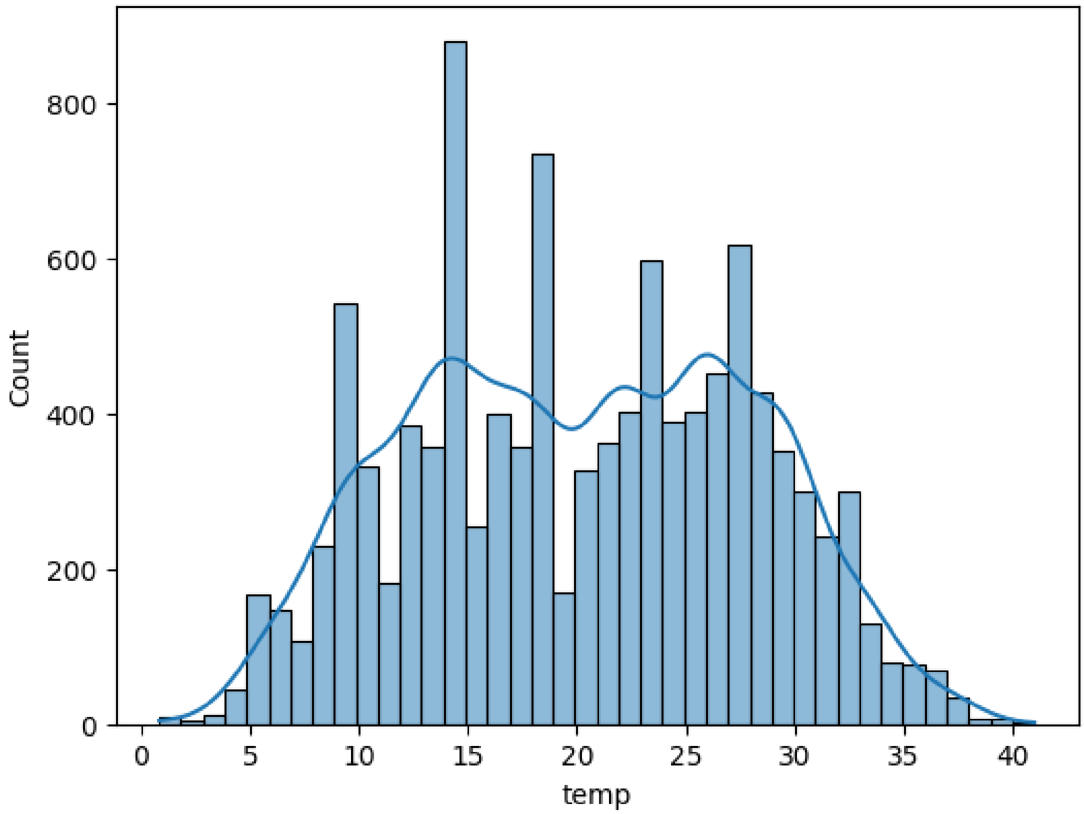

```
# The below code generates a visually appealing count plot to showcase the
    # distribution of workingday in the dataset

sns.countplot(data = df, x = 'workingday')
plt.plot()      # displaying the chart

[]
```




```
# The below code generates a visually appealing count plot to showcase
the
    # distribution of weather in the dataset

sns.countplot(data = df, x = 'weather')
plt.plot()        # displaying the chart

[]
```



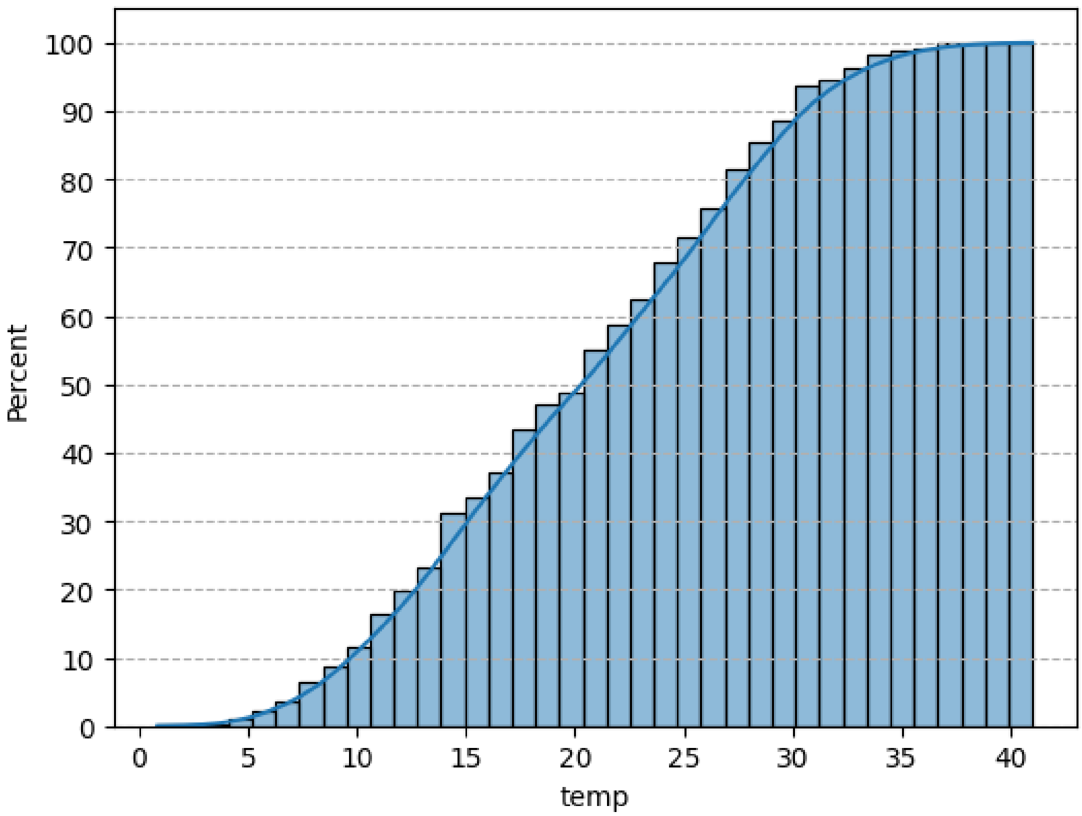

```
# The below code generates a histogram plot for the 'temp' feature,
showing the distribution of
    # temperature values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'temp', kde = True, bins = 40)
plt.plot()        # displaying the chart

[]
```




```
temp_mean = np.round(df['temp'].mean(), 2)
temp_std = np.round(df['temp'].std(), 2)
temp_mean, temp_std

(20.23, 7.79)
```

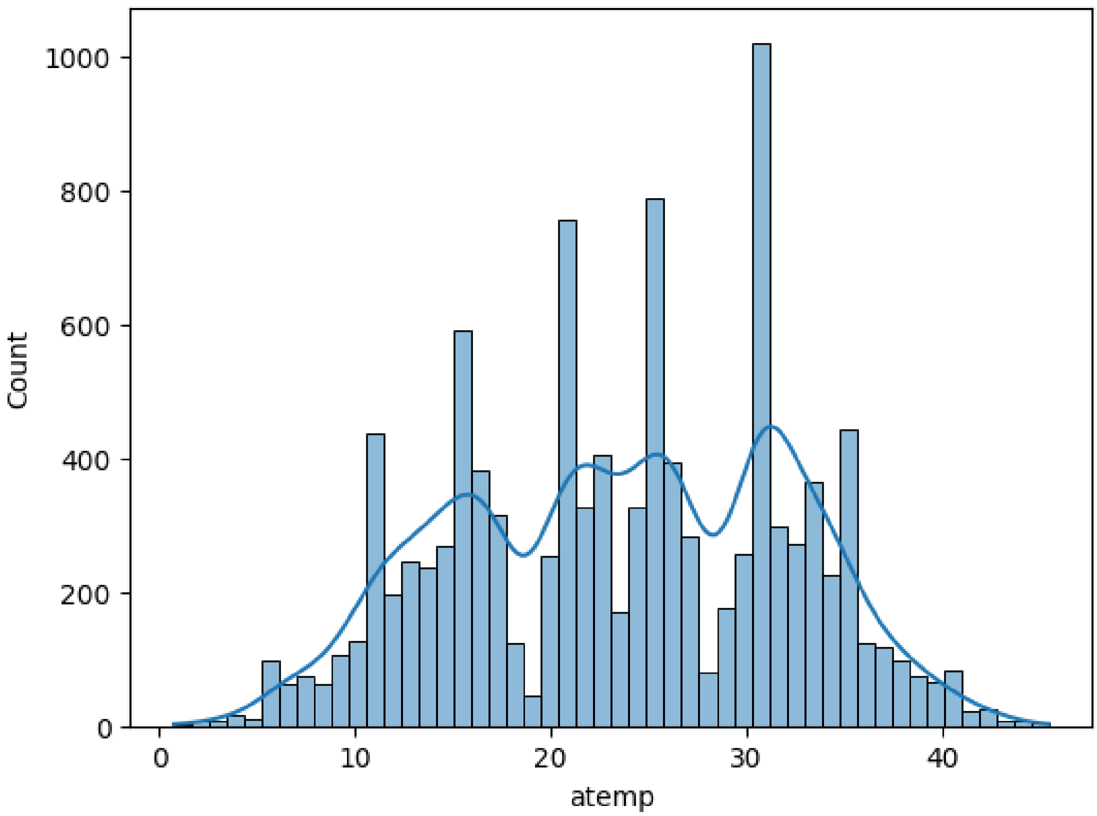
- The mean and the standard deviation of the temp column is 20.23 and 7.79 degree celcius respectively.

```
# The below code generates a histogram plot for the 'temp' feature,
showing the cumulative
    # distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'temp', kde = True, cumulative = True,
stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()          # displaying the chart

[]
```



- More than 80 % of the time, the temperature is less than 28 degrees celcius.

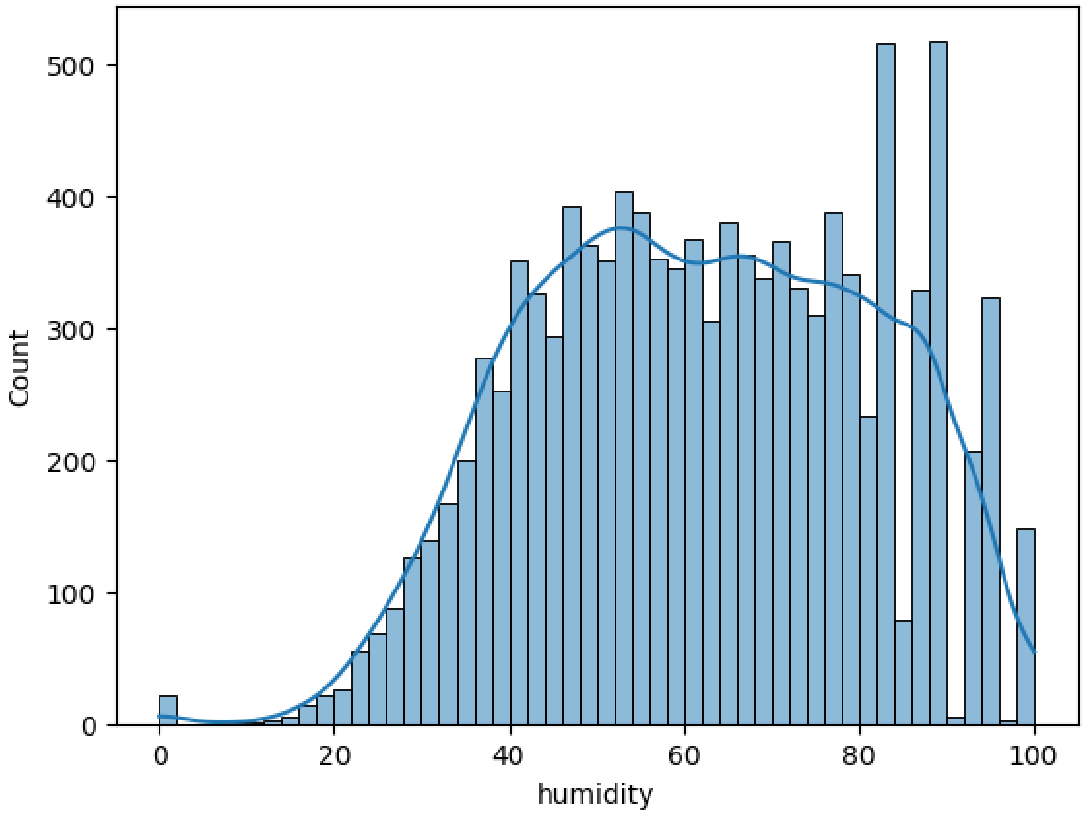
```
# The below code generates a histogram plot for the 'atemp' feature,
showing the distribution of
    # feeling temperature values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'atemp', kde = True, bins = 50)
plt.plot()          # displaying the chart

[]
```




```
temp_mean = np.round(df['atemp'].mean(), 2)
temp_std = np.round(df['atemp'].std(), 2)
temp_mean, temp_std

(23.66, 8.47)
```

- The mean and the standard deviation of the atemp column is 23.66 and 8.47 degree celcius respectively.

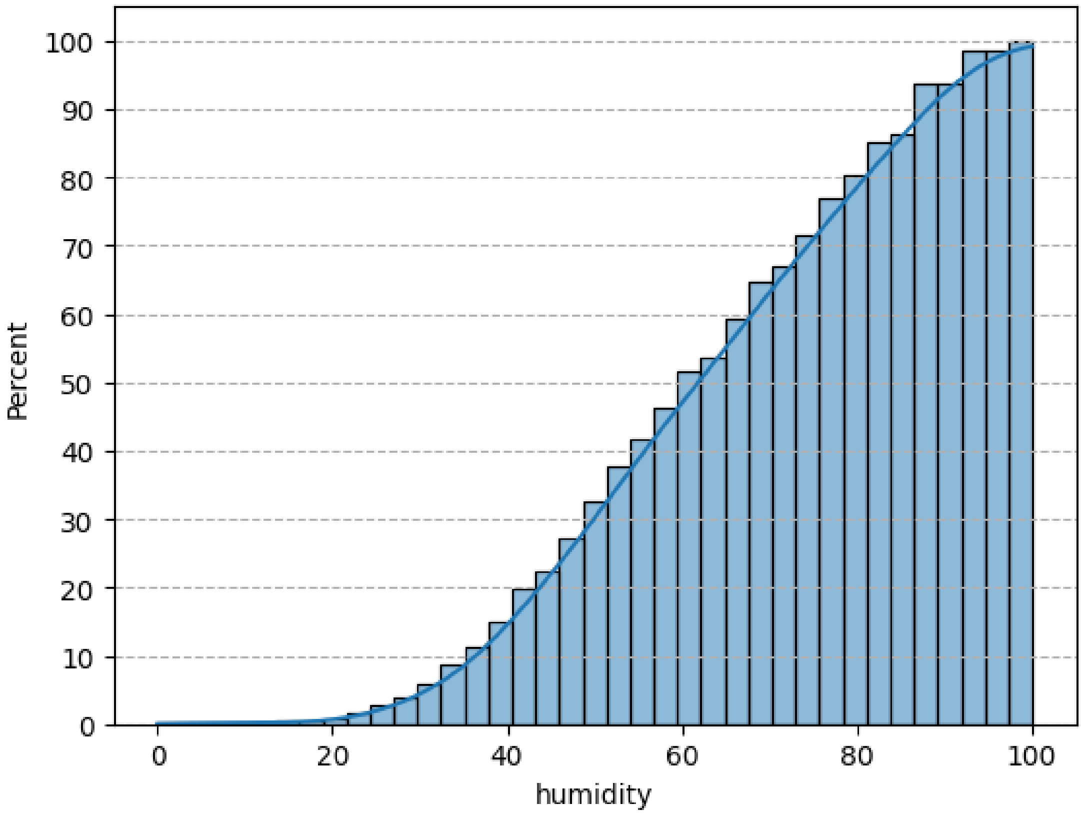
```
# The below code generates a histogram plot for the 'humidity'
feature, showing the distribution of
    # humidity values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'humidity', kde = True, bins = 50)
plt.plot()       # displaying the chart

[]
```




```
humidity_mean = np.round(df['humidity'].mean(), 2)
humidity_std = np.round(df['humidity'].std(), 2)
humidity_mean, humidity_std

(61.89, 19.25)
```

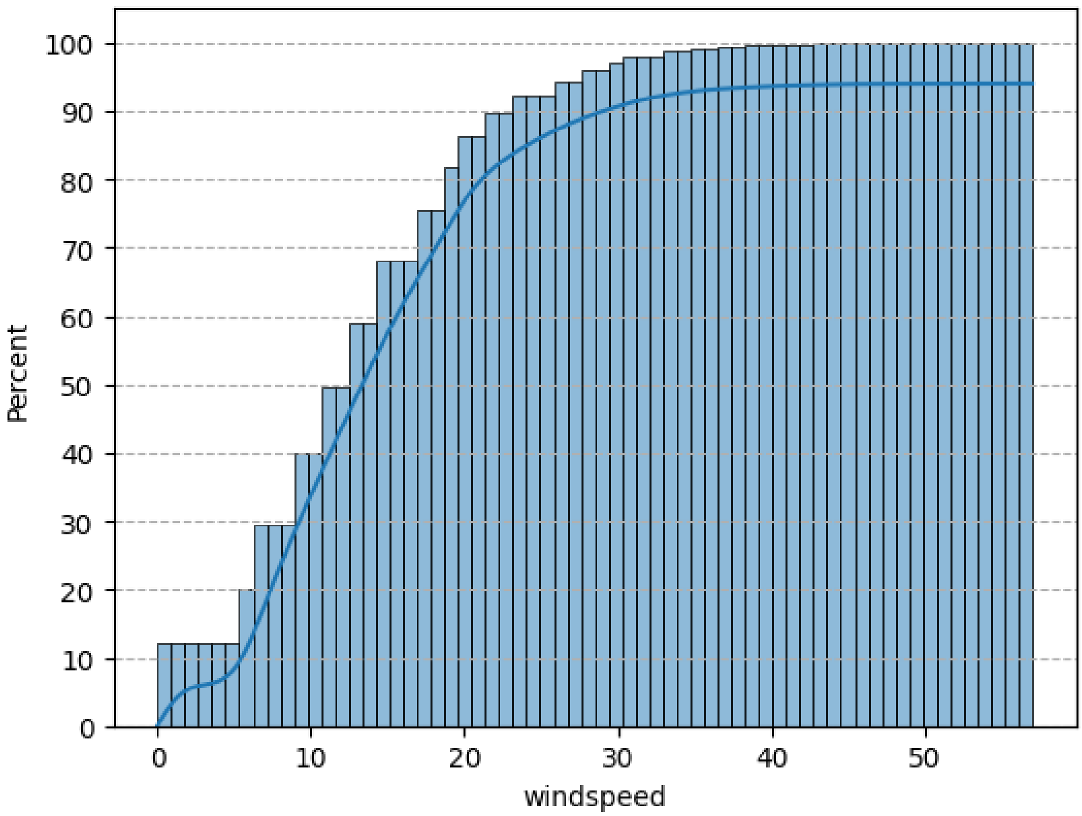
- The mean and the standard deviation of the humidity column is 61.89 and 19.25 respectively.

```
# The below code generates a histogram plot for the 'humidity'
feature, showing the cumulative
    # distribution of humidity values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'humidity', kde = True, cumulative = True,
stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')    # setting the gridlines
along y axis
plt.yticks(np.arange(0, 101, 10))
plt.plot()       # displaying the chart

[]
```



- More than 80 % of the time, the humidity value is greater than 40. Thus for most of the time, humidity level varies from optimum to too moist.

```
sns.histplot(data = df, x = 'windspeed', kde = True, cumulative =
True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()       # displaying the chart

[]
```



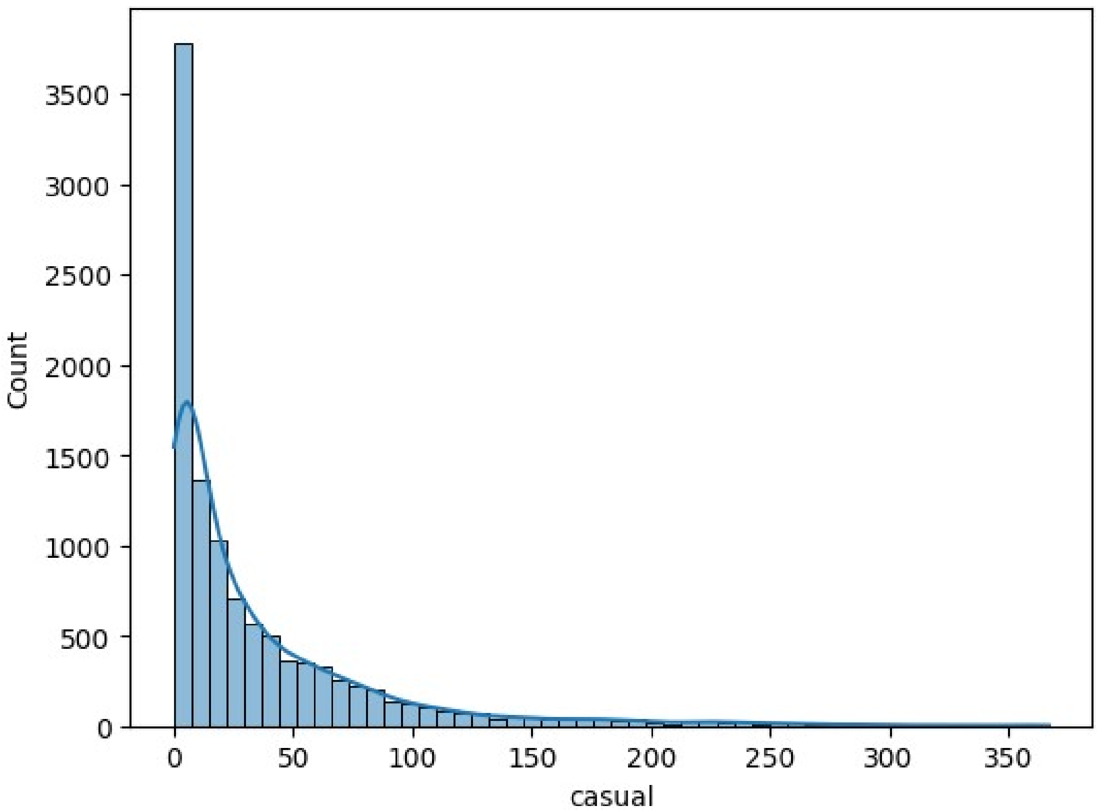
- More than 85 % of the total windspeed data has a value of less than 20.

```
len(df[df['windspeed'] < 20]) / len(df)
```

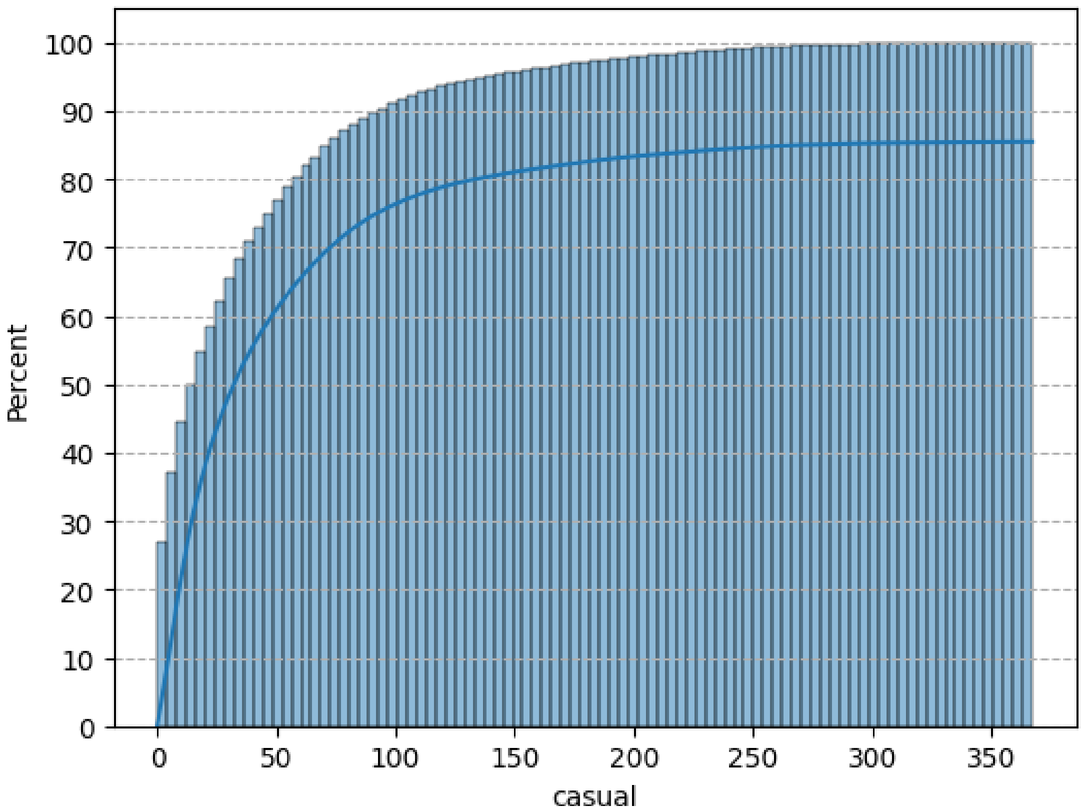
```
0.8626676465184641
```

```
# The below code generates a histogram plot for the 'casual' feature, showing the distribution of
    # casual users' values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape, making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'casual', kde = True, bins = 50)
plt.plot()      # displaying the chart
```

```
[]
```

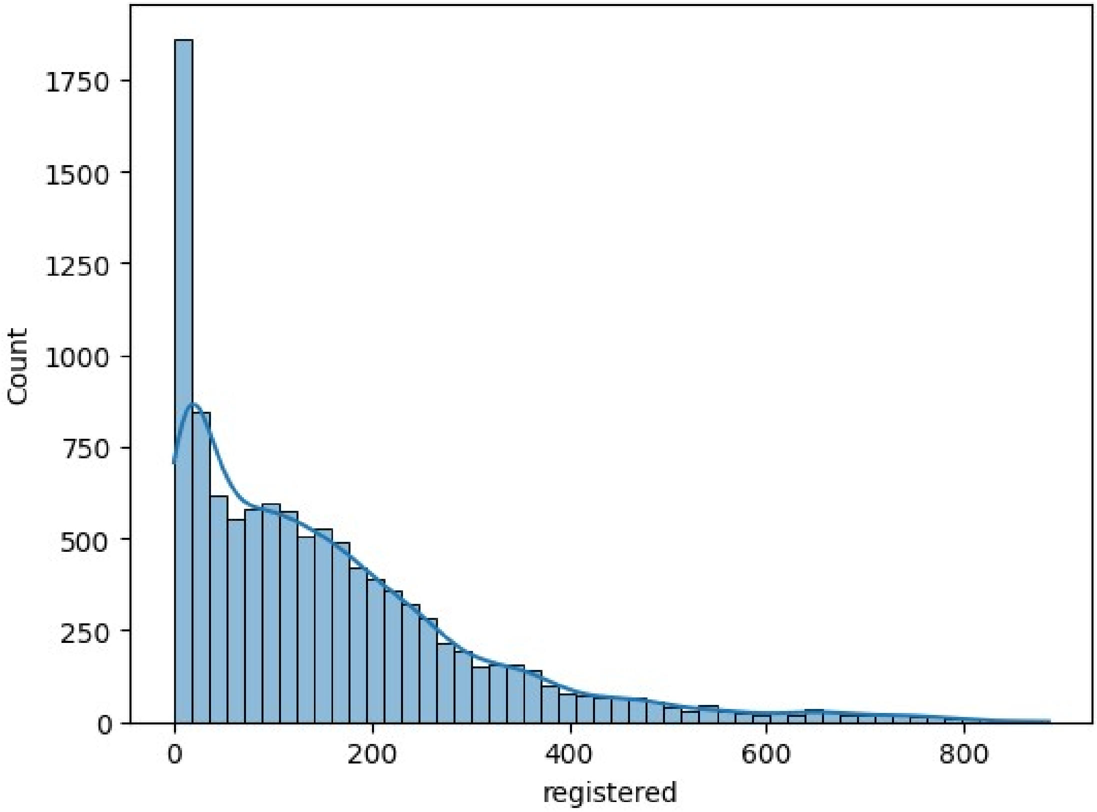



```
sns.histplot(data = df, x = 'casual', kde = True, cumulative = True,
stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()        # displaying the chart

[]
```

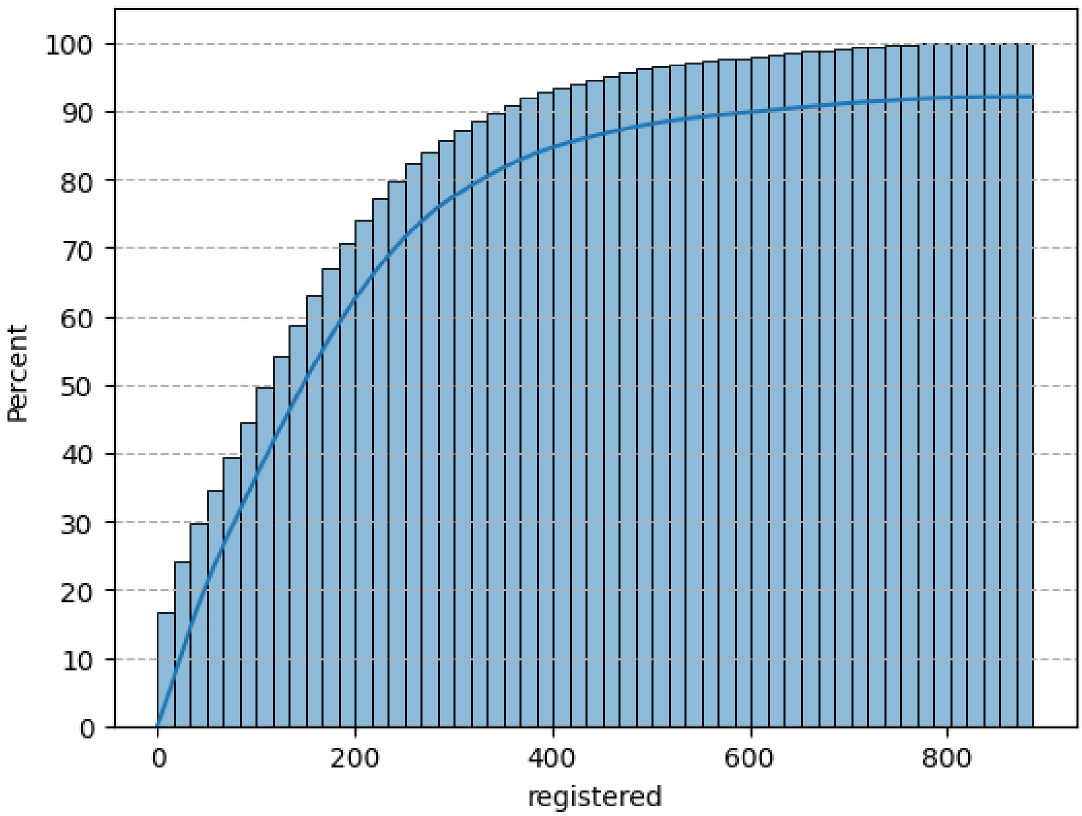


- More than 80 % of the time, the count of casual users is less than 60.

```
# The below code generates a histogram plot for the 'registered'
feature, showing the distribution of
    # registered users' values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape,
making it easier to analyze the
    # data distribution.

sns.histplot(data = df, x = 'registered', kde = True, bins = 50)
plt.plot()          # displaying the chart

[]
```



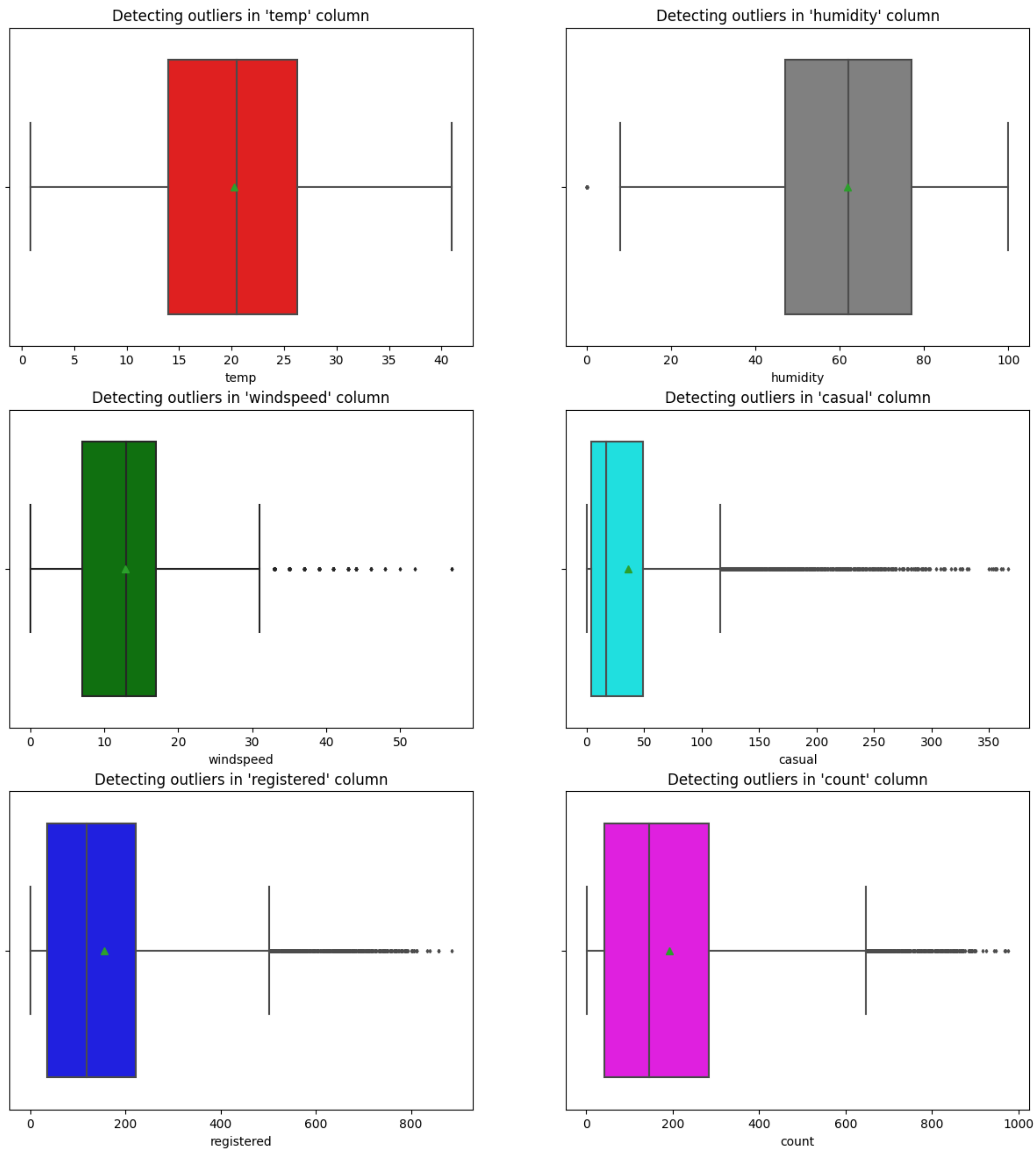

```
sns.histplot(data = df, x = 'registered', kde = True, cumulative =
True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()        # displaying the chart

[]
```



- More than 85 % of the time, the count of registered users is less than 300.

Outliers Detection

```
columns = ['temp', 'humidity', 'windspeed', 'casual', 'registered',
'count']
colors = np.random.permutation(['red', 'blue', 'green', 'magenta',
'cyan', 'gray'])
count = 1
plt.figure(figsize = (15, 16))
for i in columns:
    plt.subplot(3, 2, count)
    plt.title(f"Detecting outliers in '{i}' column")
    sns.boxplot(data = df, x = df[i], color = colors[count - 1],
showmeans = True, fliersize = 2)
    plt.plot()
    count += 1
```

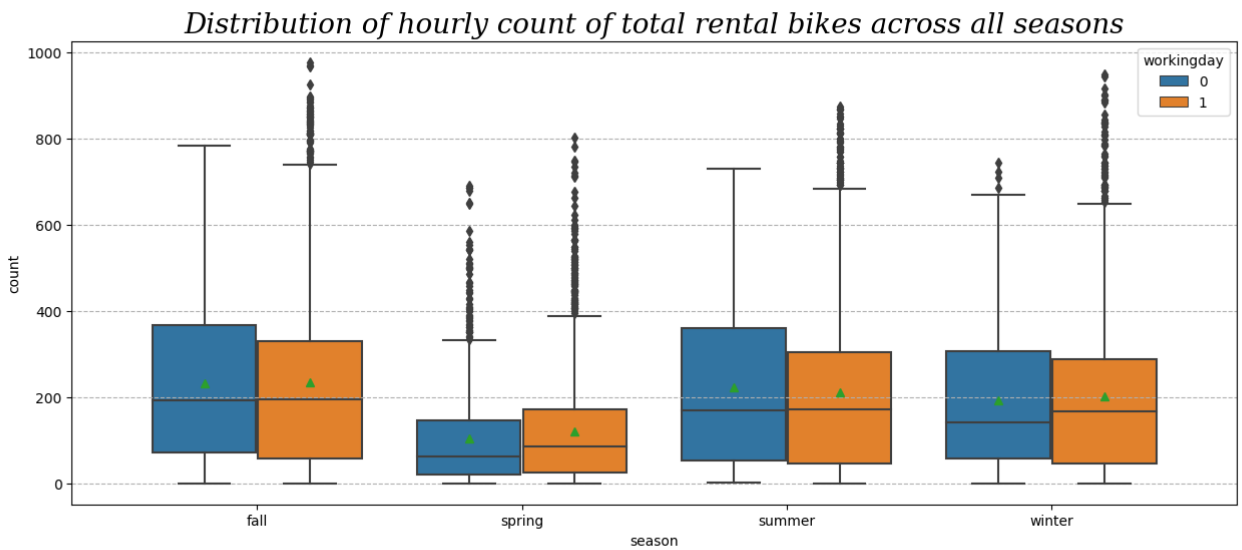


- There is no outlier in the temp column.
- There are few outliers present in humidity column.
- There are many outliers present in each of the columns : windspeed, casual, registered, count.

Bivariate Analysis

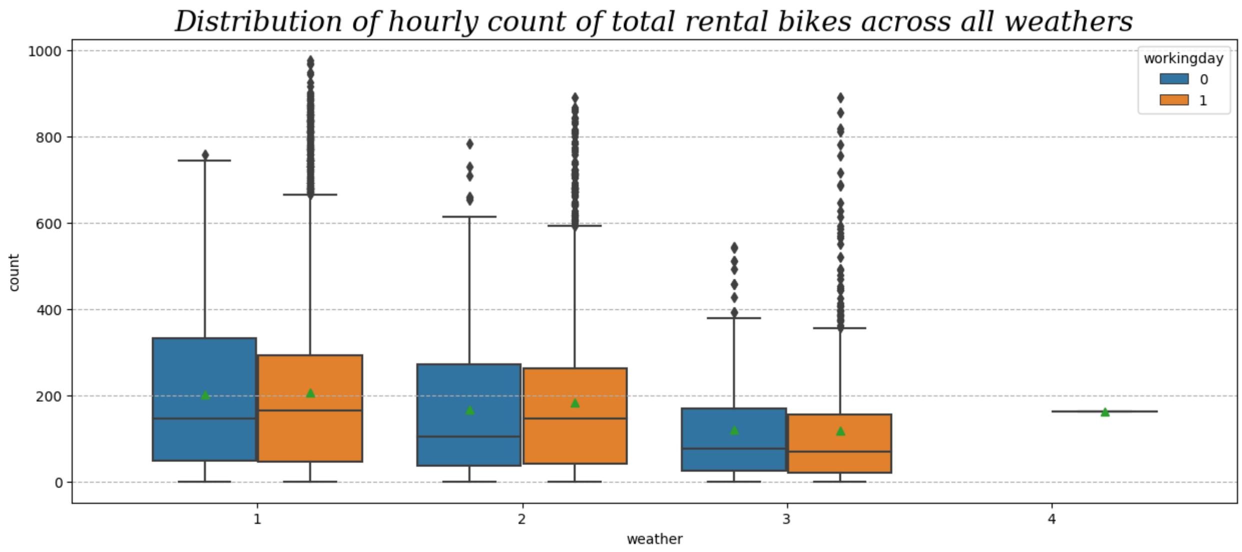
```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across
```

```
all seasons',
         fontdict = {'size' : 20,
                    'style' : 'oblique',
                    'family' : 'serif'})
sns.boxplot(data = df, x = 'season', y = 'count', hue = 'workingday',
showmeans = True)
plt.grid(axis = 'y', linestyle = '--')
plt.plot()

[]
```



- The hourly count of total rental bikes is higher in the fall season, followed by the summer and winter seasons. It is generally low in the spring season.

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across
all weathers',
         fontdict = {'size' : 20,
                    'style' : 'oblique',
                    'family' : 'serif'})
sns.boxplot(data = df, x = 'weather', y = 'count', hue = 'workingday',
showmeans = True)
plt.grid(axis = 'y', linestyle = '--')
plt.plot()

[]
```



- The hourly count of total rental bikes is higher in the clear and cloudy weather, followed by the misty weather and rainy weather. There are very few records for extreme weather conditions.

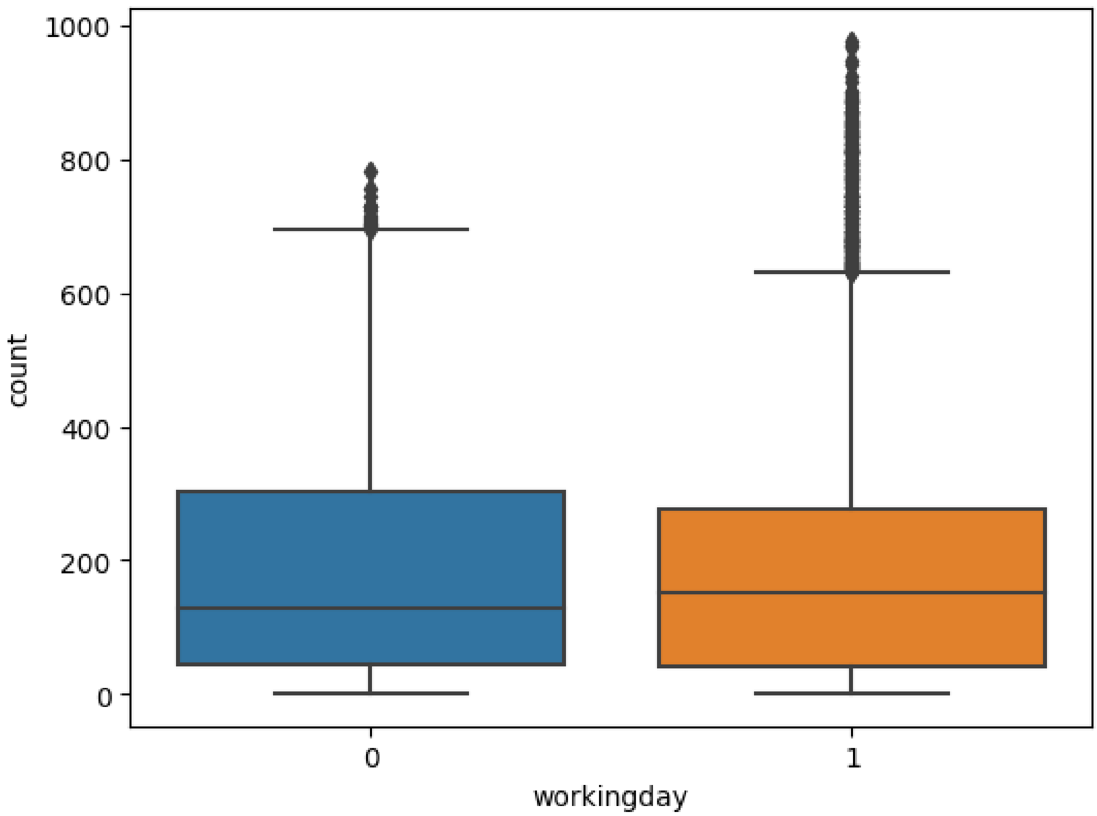
Is there any effect of Working Day on the number of electric cycles rented ?

```
df.groupby(by = 'workingday')['count'].describe()
```

```
                count        mean         std  min   25%    50%    75%
max
workingday

0              3474.0  188.506621  173.724015  1.0  44.0  128.0  304.0
783.0
1              7412.0  193.011873  184.513659  1.0  41.0  151.0  277.0
977.0
```

```
sns.boxplot(data = df, x = 'workingday', y = 'count')
plt.plot()
```

```
[]
```



STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Working Day does not have any effect on the number of electric cycles rented.
- **Alternate Hypothesis (HA)** - Working Day has some effect on the number of electric cycles rented

STEP-2 : Checking for basic assumpitons for the hypothesis

- Distribution check using **QQ Plot**
- Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

- If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha.

- We set our ***alpha to be 0.05***

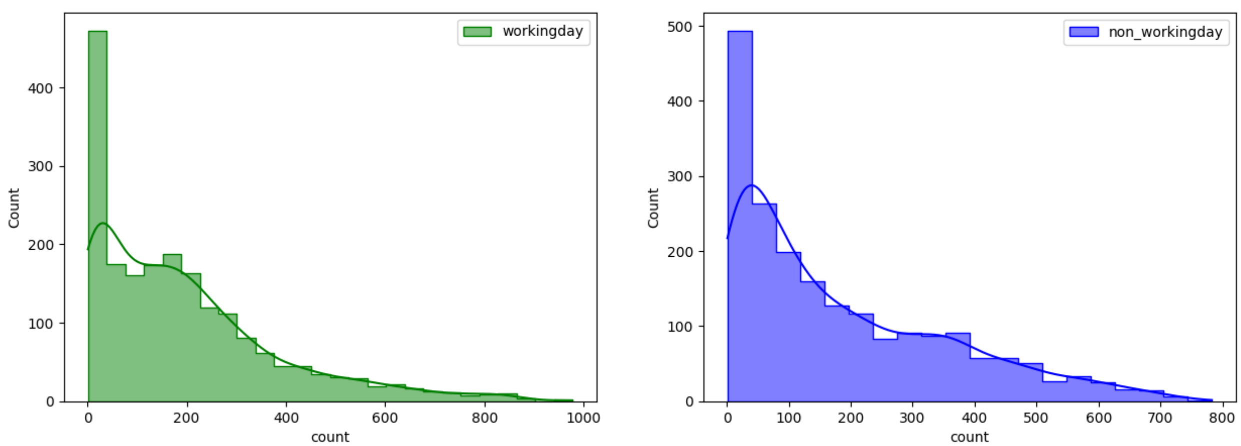
STEP-5: Compare p-value and alpha.

- Based on p-value, we will accept or reject H0.

  a. **p-val > alpha** : Accept H0
  b. **p-val < alpha** : Reject H0

Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (15, 5))
plt.subplot(1, 2, 1)
sns.histplot(df.loc[df['workingday'] == 1, 'count'].sample(2000),
             element = 'step', color = 'green', kde = True, label =
'workingday')
plt.legend()
plt.subplot(1, 2, 2)
sns.histplot(df.loc[df['workingday'] == 0, 'count'].sample(2000),
             element = 'step', color = 'blue', kde = True, label =
'non_workingday')
plt.legend()
plt.plot()

[]
```

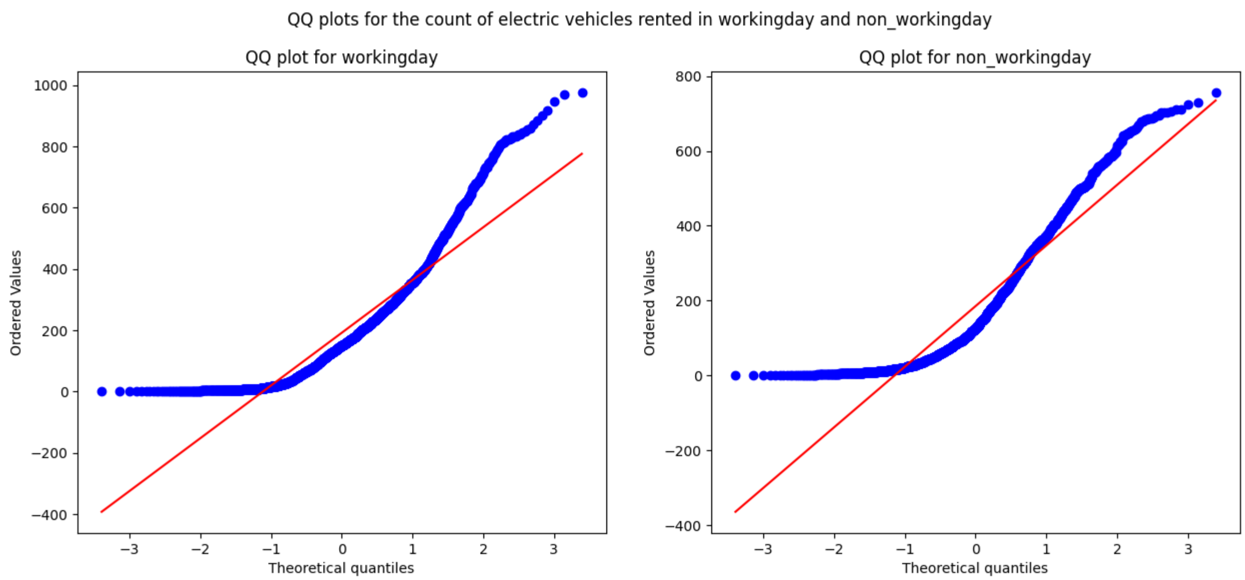


- It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
```

```
plt.suptitle('QQ plots for the count of electric vehicles rented in
workingday and non_workingday')
spy.probplot(df.loc[df['workingday'] == 1, 'count'].sample(2000), plot
= plt, dist = 'norm')
plt.title('QQ plot for workingday')
plt.subplot(1, 2, 2)
spy.probplot(df.loc[df['workingday'] == 0, 'count'].sample(2000), plot
= plt, dist = 'norm')
plt.title('QQ plot for non_workingday')
plt.plot()

[]
```



- It can be inferred from the above plot that the distributions do not follow normal distribution.

It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] == 1,
'count'].sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
```

```
else:
    print('The sample follows normal distribution')
```

```
p-value 1.5371759864893973e-37
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] == 0,
'count'].sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.3961317311518112e-36
The sample does not follow normal distribution
```

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
transformed_workingday = spy.boxcox(df.loc[df['workingday'] == 1,
'count'])[0]
test_stat, p_value = spy.shapiro(transformed_workingday)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.6136431560309944e-33
The sample does not follow normal distribution
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/scipy/stats/_morestats.py:1882: UserWarning: p-value may
not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

```
transformed_non_workingday = spy.boxcox(df.loc[df['workingday'] == 1,
'count'])[0]
test_stat, p_value = spy.shapiro(transformed_non_workingday)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.6136431560309944e-33
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "workingday" and "non_workingday" data, the samples do not follow normal distribution.

- Homogeneity of Variances using **Lavene's test**

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['workingday'] == 1,
'count'].sample(2000),
                                df.loc[df['workingday'] == 0,
'count'].sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')

p-value 0.9437671082861091
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

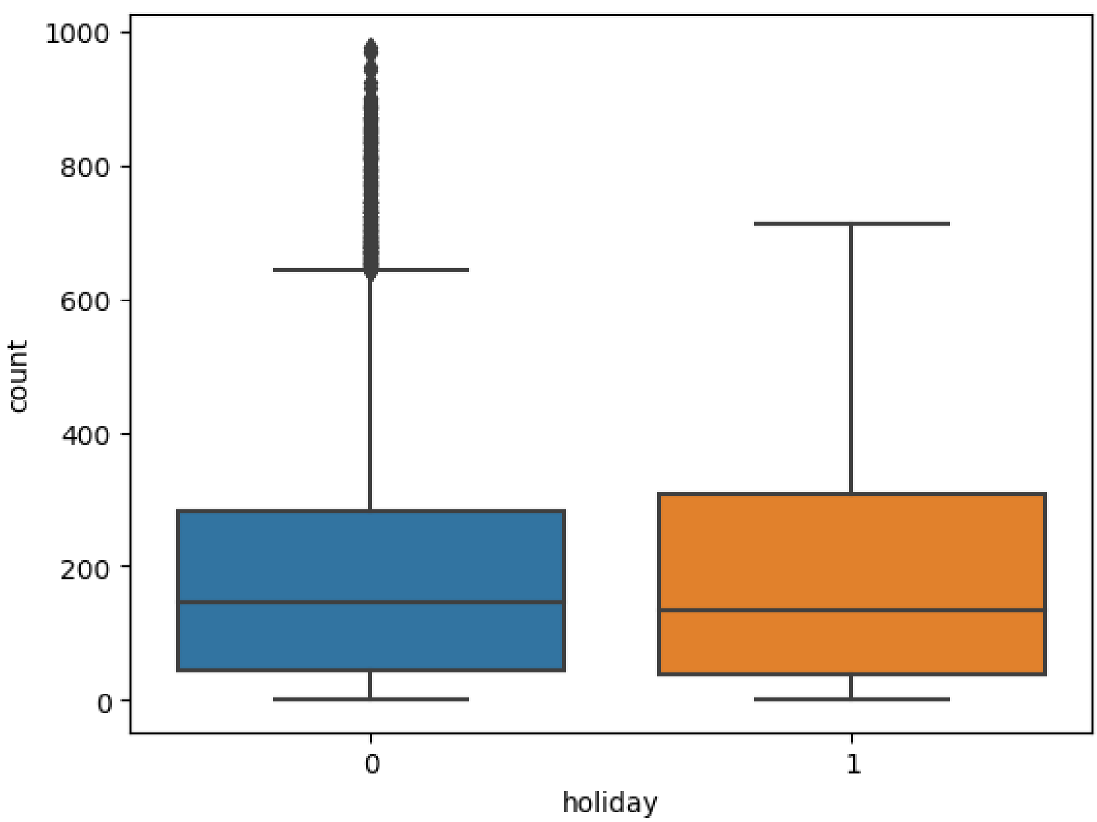
```
# Ho : Mean no.of electric cycles rented is same for working and non-
working days
# Ha : Mean no.of electric cycles rented is not same for working and
non-working days
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent
samples

test_stat, p_value = spy.mannwhitneyu(df.loc[df['workingday'] == 1,
'count'],
                                      df.loc[df['workingday'] == 0,
'count'])
print('P-value :',p_value)
if p_value < 0.05:
    print('Mean no.of electric cycles rented is not same for working
and non-working days')
else:
    print('Mean no.of electric cycles rented is same for working and
non-working days')

P-value : 0.9679139953914079
Mean no.of electric cycles rented is same for working and non-working
days
```

Therefore, the mean hourly count of the total rental bikes is statistically same for both working and non- working days .

Is there any effect of holidays on the number of electric cycles rented ?

```
df.groupby(by = 'holiday')['count'].describe()
```

```
             count        mean         std  min   25%    50%    75%
max
holiday

0          10575.0  191.741655  181.513131  1.0  43.0  145.0  283.0
977.0
1            311.0  185.877814  168.300531  1.0  38.5  133.0  308.0
712.0

sns.boxplot(data = df, x = 'holiday', y = 'count')
plt.plot()

[]
```



STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Holidays have no effect on the number of electric vehicles rented
- **Alternate Hypothesis (HA)** - Holidays has some effect on the number of electric vehicles rented

STEP-2 : Checking for basic assumpitons for the hypothesis

* Distribution check using **QQ Plot**
* Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

---

* If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha.

---

* We set our ***alpha to be 0.05***

STEP-5: Compare p-value and alpha.

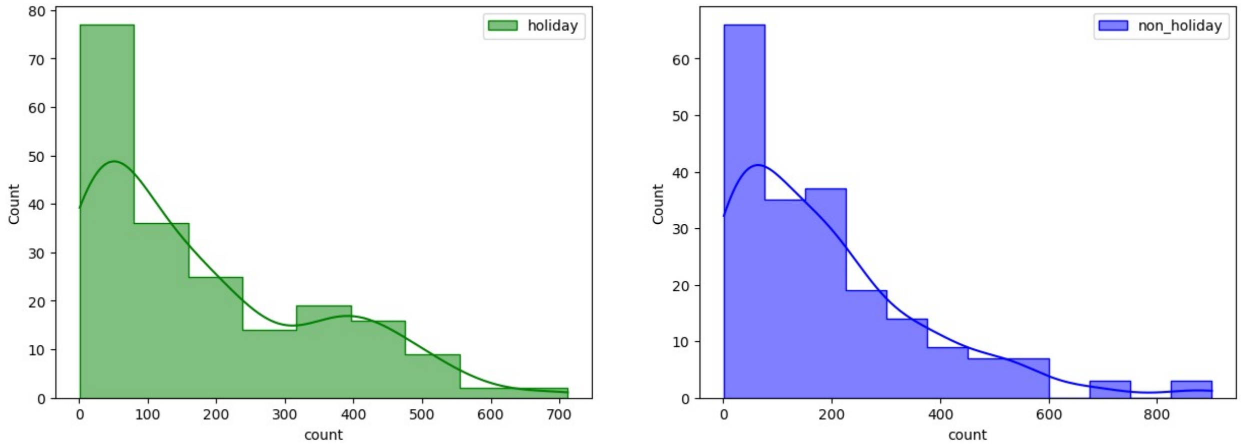
---

* Based on p-value, we will accept or reject H0.

    a. **p-val > alpha** : Accept H0
    b. **p-val < alpha** : Reject H0

Visual Tests to know if the samples follow normal distribution

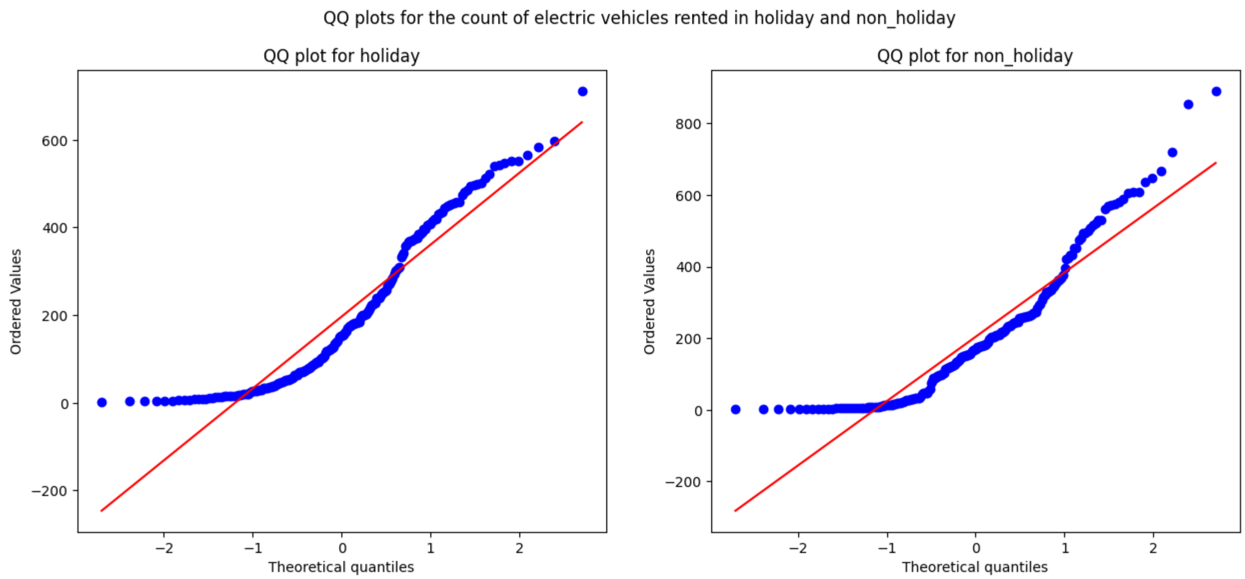
```
plt.figure(figsize = (15, 5))
plt.subplot(1, 2, 1)
sns.histplot(df.loc[df['holiday'] == 1, 'count'].sample(200),
             element = 'step', color = 'green', kde = True, label =
'holiday')
plt.legend()
plt.subplot(1, 2, 2)
sns.histplot(df.loc[df['holiday'] == 0, 'count'].sample(200),
             element = 'step', color = 'blue', kde = True, label =
'non_holiday')
plt.legend()
plt.plot()

[]
```



- It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in holiday and non_holiday')
spy.probplot(df.loc[df['holiday'] == 1, 'count'].sample(200), plot = plt, dist = 'norm')
plt.title('QQ plot for holiday')
plt.subplot(1, 2, 2)
spy.probplot(df.loc[df['holiday'] == 0, 'count'].sample(200), plot = plt, dist = 'norm')
plt.title('QQ plot for non_holiday')
plt.plot()

[]
```



- It can be inferred from the above plot that the distributions do not follow normal distribution.

It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df.loc[df['holiday'] == 1,
'count'].sample(200))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

p-value 1.8166641801986572e-10
The sample does not follow normal distribution

test_stat, p_value = spy.shapiro(df.loc[df['holiday'] == 0,
'count'].sample(200))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 4.3818013860342717e-14
The sample does not follow normal distribution
```

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
transformed_holiday = spy.boxcox(df.loc[df['holiday'] == 1, 'count'])
[0]
test_stat, p_value = spy.shapiro(transformed_holiday)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 2.1349286782879062e-07
The sample does not follow normal distribution
```

```
transformed_non_holiday = spy.boxcox(df.loc[df['holiday'] == 0,
'count'].sample(5000))[0]
test_stat, p_value = spy.shapiro(transformed_non_holiday)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.0057126667202041e-25
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the "holiday" and "non_holiday" data, the samples do not follow normal distribution.

Homogeneity of Variances using Levene's test

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['holiday'] == 0,
'count'].sample(200),
                                df.loc[df['holiday'] == 1,
'count'].sample(200))
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 0.9562314837936287
The samples have Homogenous Variance
```

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
# Ho : No.of electric cycles rented is similar for holidays and non-holidays
# Ha : No.of electric cycles rented is not similar for holidays and non-holidays days
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

test_stat, p_value = spy.mannwhitneyu(df.loc[df['holiday'] == 0, 'count'].sample(200),
                                      df.loc[df['holiday'] == 1, 'count'].sample(200))
print('P-value :',p_value)
if p_value < 0.05:
    print('No.of electric cycles rented is not similar for holidays and non-holidays days')
else:
    print('No.of electric cycles rented is similar for holidays and non-holidays')

P-value : 0.7591256657474473
No.of electric cycles rented is similar for holidays and non-holidays
```

Therefore, the number of electric cycles rented is statistically similar for both holidays and non - holidays.

Is weather dependent on the season ?

```
df[['weather', 'season']].describe()
```

	weather	season
count	10886	10886
unique	4	4
top	1	winter
freq	7192	2734

- It is clear from the above statistical description that both 'weather' and 'season' features are categorical in nature.

STEP-1 : Set up Null Hypothesis

---

1. **Null Hypothesis (H0)** - weather is independent of season

2. **Alternate Hypothesis (HA)** - weather is dependent of seasons.

STEP-2: Define Test statistics

Since we have two categorical features, the Chi- square test is applicable here. Under H0, the test statistic should follow **Chi-Square Distribution**.

STEP-3: Checking for basic assumptons for the hypothesis (Non-Parametric Test)

---

1. The data in the cells should be **frequencies**, or **counts** of cases.
2. The levels (or categories) of the variables are **mutually exclusive**. That is, a particular subject fits into one and only one level of each of the variables.
3. There are 2 variables, and both are measured as **categories**.
4. The **value of the cell expecteds should be 5 or more** in at least 80% of the cells, and no cell should have an expected of less than one (3).

STEP-4: Compute the p-value and fix value of alpha.

---

we will be computing the chi square-test p-value using the chi2_contingency function using scipy.stats. We set our **alpha to be 0.05**

STEP-5: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

1. **p-val > alpha** : Accept H0
2. **p-val < alpha** : Reject H0

The **Chi-square statistic is a non-parametric** (distribution free) tool designed to analyze group differences when the dependent variable is measured at a nominal level. Like all non-parametric statistics, the Chi-square is robust with respect to the distribution of the data. Specifically, it does not require equality of variances among the study groups or homoscedasticity in the data.

```
# First, finding the contingency table such that each value is the
total number of total bikes rented
  # for a particular season and weather
cross_table = pd.crosstab(index = df['season'],
                          columns = df['weather'],
                          values = df['count'],
                          aggfunc = np.sum).replace(np.nan, 0)
cross_table

weather       1       2      3    4
season
```

```
fall     470116  139386  31160    0
spring   223009   76406  12919  164
summer   426350  134177  27755    0
winter   356588  157191  30255    0
```

Since the above contingency table has one column in which the count of the rented electric vehicle is less than 5 in most of the cells, we can remove the weather 4 and then proceed further.

```
cross_table = pd.crosstab(index = df['season'],
                          columns = df.loc[df['weather'] != 4,
'weather'],
                          values = df['count'],
                          aggfunc = np.sum).to_numpy()[:, :3]
cross_table

array([[470116, 139386,  31160],
       [223009,  76406,  12919],
       [426350, 134177,  27755],
       [356588, 157191,  30255]])

chi_test_stat, p_value, dof, expected = spy.chi2_contingency(observed
= cross_table)
print('Test Statistic =', chi_test_stat)
print('p value =', p_value)
print('-' * 65)
print("Expected : '\n'", expected)

Test Statistic = 10838.372332480214
p value = 0.0
-----------------------------------------------------------------
Expected : '
' [[453484.88557396 155812.72247031  31364.39195574]
 [221081.86259035  75961.44434981  15290.69305984]
 [416408.3330293  143073.60199337  28800.06497733]
 [385087.91880639 132312.23118651  26633.8500071 ]]
```

Comparing p value with significance level

```
alpha = 0.05
if p_value < alpha:
    print('Reject Null Hypothesis')
else:
    print('Failed to reject Null Hypothesis')

Reject Null Hypothesis
```

Therefore, there is statistically significant dependency of weather and season based on the number of number of bikes rented.

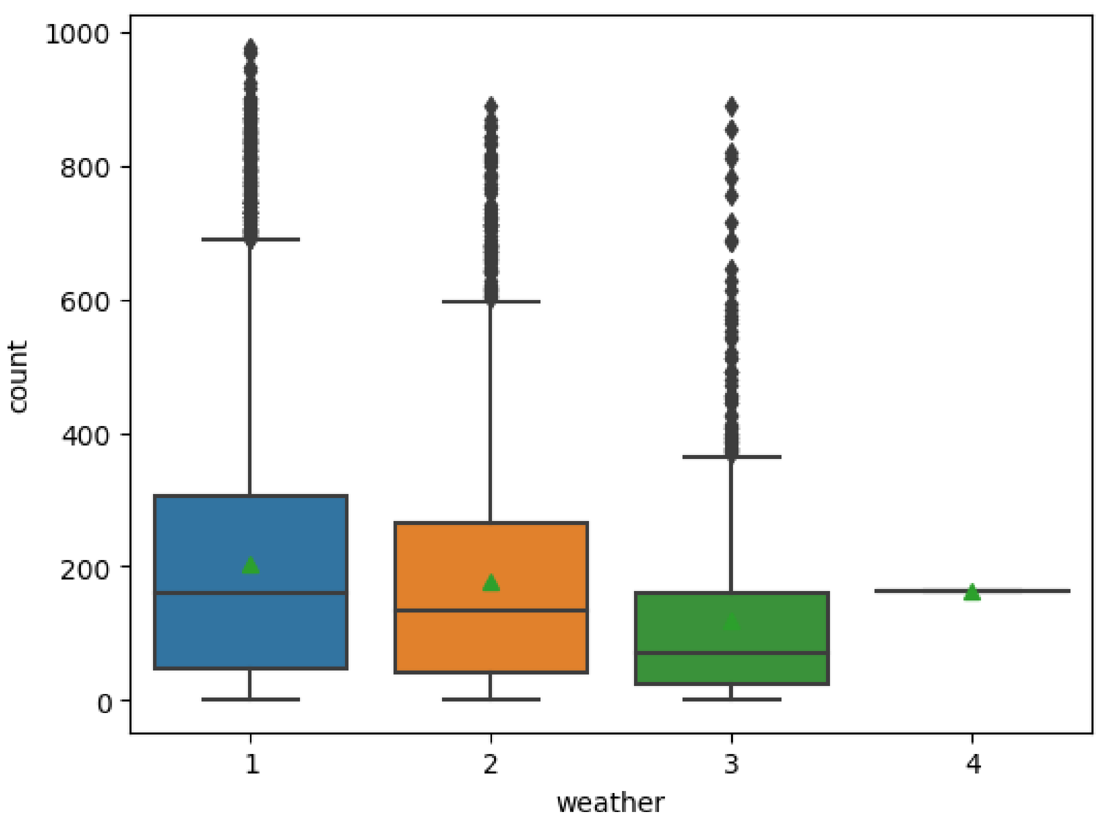
Is the number of cycles rented is similar or different in different weather ?

```
df.groupby(by = 'weather')['count'].describe()
```

	count	mean	std	min	25%	50%	75%	max
weather								
1	7192.0	205.236791	187.959566	1.0	48.0	161.0	305.0	977.0
2	2834.0	178.955540	168.366413	1.0	41.0	134.0	264.0	890.0
3	859.0	118.846333	138.581297	1.0	23.0	71.0	161.0	891.0
4	1.0	164.000000	NaN	164.0	164.0	164.0	164.0	164.0

```
sns.boxplot(data = df, x = 'weather', y = 'count', showmeans = True)
plt.plot()
```

```
[]
```



```
df_weather1 = df.loc[df['weather'] == 1]
df_weather2 = df.loc[df['weather'] == 2]
df_weather3 = df.loc[df['weather'] == 3]
df_weather4 = df.loc[df['weather'] == 4]
len(df_weather1), len(df_weather2), len(df_weather3), len(df_weather4)
```

```
(7192, 2834, 859, 1)
```

STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Mean of cycle rented per hour is same for weather 1, 2 and 3. (We wont be considering weather 4 as there in only 1 data point for weather 4 and we cannot perform a ANOVA test with a single data point for a group)
- **Alternate Hypothesis (HA)** -Mean of cycle rented per hour is not same for season 1,2,3 and 4 are different.

STEP-2 : Checking for basic assumpitons for the hypothesis

Normality check using **QQ Plot**. If the distribution is not normal, use **BOX-COX transform** to transform it to normal distribution.

Homogeneity of Variances using **Levene's test**

Each observations are **independent**.

STEP-3: Define **Test statistics**

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

F=MSB / MSW

Under H0, the test statistic should follow **F-Distribution**.

STEP-4: Decide the kind of test.

We will be performing **right tailed f-test**

STEP-5: Compute the **p-value** and fix value of alpha.

---

we will be computing the anova-test p-value using the f_oneway function using scipy.stats. We set our **alpha to be 0.05**

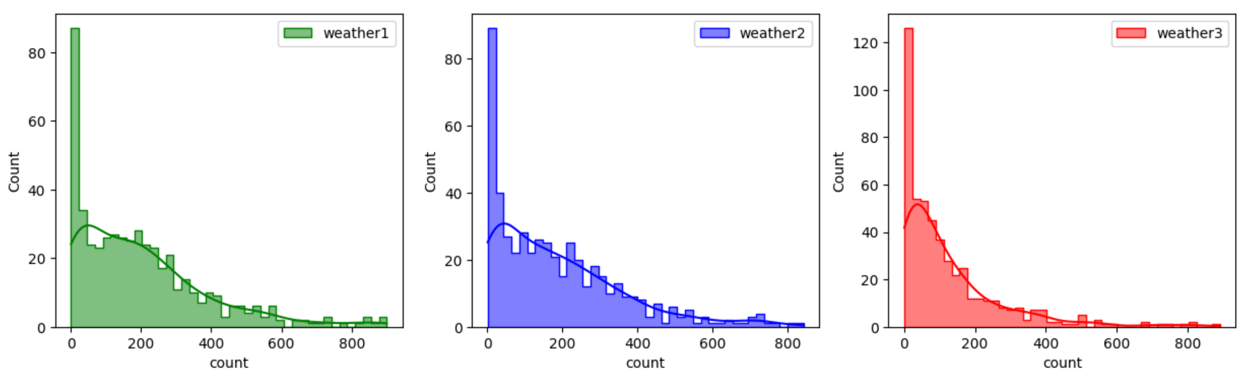
STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

- **p-val > alpha** : Accept H0
- **p-val < alpha** : Reject H0

Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (15, 4))
plt.subplot(1, 3, 1)
sns.histplot(df_weather1.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'green', kde = True, label =
'weather1')
plt.legend()
plt.subplot(1, 3, 2)
sns.histplot(df_weather2.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'blue', kde = True, label =
'weather2')
plt.legend()
plt.subplot(1, 3, 3)
sns.histplot(df_weather3.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'red', kde = True, label =
'weather3')
plt.legend()
plt.plot()

[]
```

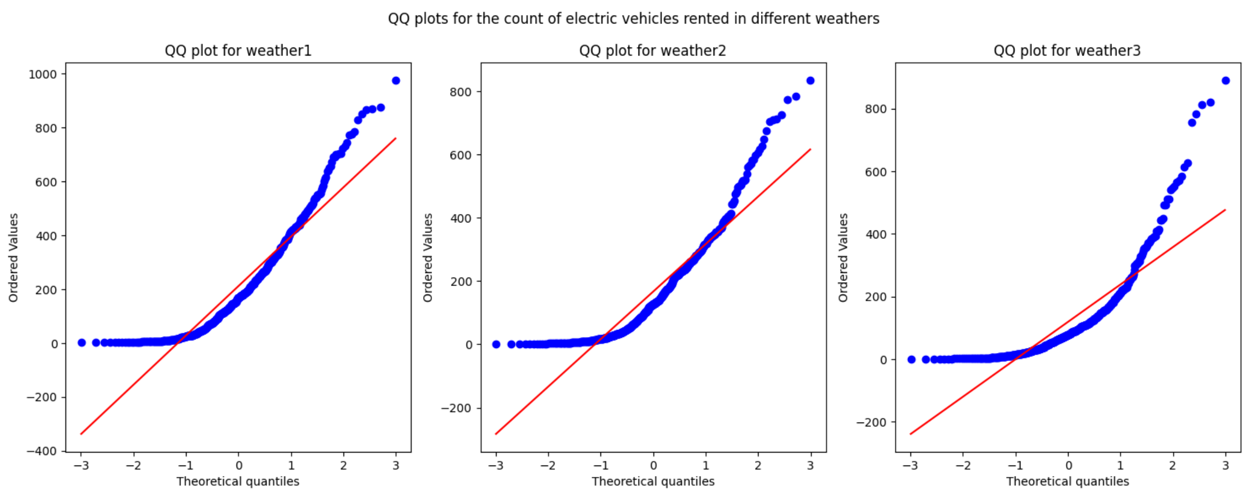


- It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
plt.figure(figsize = (18, 6))
plt.subplot(1, 3, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in
```

```
different weathers')
spy.probplot(df_weather1.loc[:, 'count'].sample(500), plot = plt, dist
= 'norm')
plt.title('QQ plot for weather1')
plt.subplot(1, 3, 2)
spy.probplot(df_weather2.loc[:, 'count'].sample(500), plot = plt, dist
= 'norm')
plt.title('QQ plot for weather2')
plt.subplot(1, 3, 3)
spy.probplot(df_weather3.loc[:, 'count'].sample(500), plot = plt, dist
= 'norm')
plt.title('QQ plot for weather3')
plt.plot()

[]
```



- It can be inferred from the above plot that the distributions do not follow normal distribution.

It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df_weather1.loc[:,
'count'].sample(500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
```

```
else:
    print('The sample follows normal distribution')
```

```
p-value 1.2403542679549986e-17
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df_weather2.loc[:,
'count'].sample(500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.460265233075048e-20
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df_weather3.loc[:,
'count'].sample(500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 7.11154161752216e-26
The sample does not follow normal distribution
```

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
transformed_weather1 = spy.boxcox(df_weather1.loc[:,
'count'].sample(5000))[0]
test_stat, p_value = spy.shapiro(transformed_weather1)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.062597854152139e-27
The sample does not follow normal distribution
```

```
transformed_weather2 = spy.boxcox(df_weather2.loc[:, 'count'])[0]
test_stat, p_value = spy.shapiro(transformed_weather2)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.9212781916101391e-19
The sample does not follow normal distribution
```

```python
transformed_weather3 = spy.boxcox(df_weather3.loc[:, 'count'])[0]
test_stat, p_value = spy.shapiro(transformed_weather3)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.4131240959613933e-06
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the weather data, the samples do not follow normal distribution.

Homogeneity of Variances using Levene's test

```python
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df_weather1.loc[:, 
'count'].sample(500),
                                df_weather2.loc[:, 
'count'].sample(500),
                                df_weather3.loc[:, 
'count'].sample(500))
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 4.9590591283621447e-14
The samples do not have  Homogenous Variance
```

Since the samples are not normally distributed and do not have the same variance, f_oneway test cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal-Wallis H-test for independent samples.

```python
# Ho : Mean no. of cycles rented is same for different weather
# Ha : Mean no. of cycles rented is different for different weather
# Assuming significance Level to be 0.05
alpha = 0.05
test_stat, p_value = spy.kruskal(df_weather1, df_weather2, 
df_weather3)
print('Test Statistic =', test_stat)
print('p value =', p_value)
```

```
Test Statistic = [1.36471292e+01 3.87838808e+01 5.37649760e+00
1.56915686e+01
 1.08840000e+04 3.70017441e+01 4.14298489e+01 1.83168690e+03
 2.80380482e+01 2.84639685e+02 1.73745440e+02 2.04955668e+02
 7.08445555e+01]
p value = [1.08783632e-03 3.78605818e-09 6.79999165e-02 3.91398508e-04
 0.00000000e+00 9.22939752e-09 1.00837627e-09 0.00000000e+00
 8.15859150e-07 1.55338046e-62 1.86920588e-38 3.12206618e-45
 4.13333147e-16]
```

Comparing p value with significance level

```
if p_value.all() < alpha:
    print('Reject Null Hypothesis')
else:
    print('Failed to reject Null Hypothesis')
```

```
Reject Null Hypothesis
```

Therefore, the average number of rental bikes is statistically different for different weathers.

Is the number of cycles rented is similar or different in different season ?

```
df.groupby(by = 'season')['count'].describe()
```

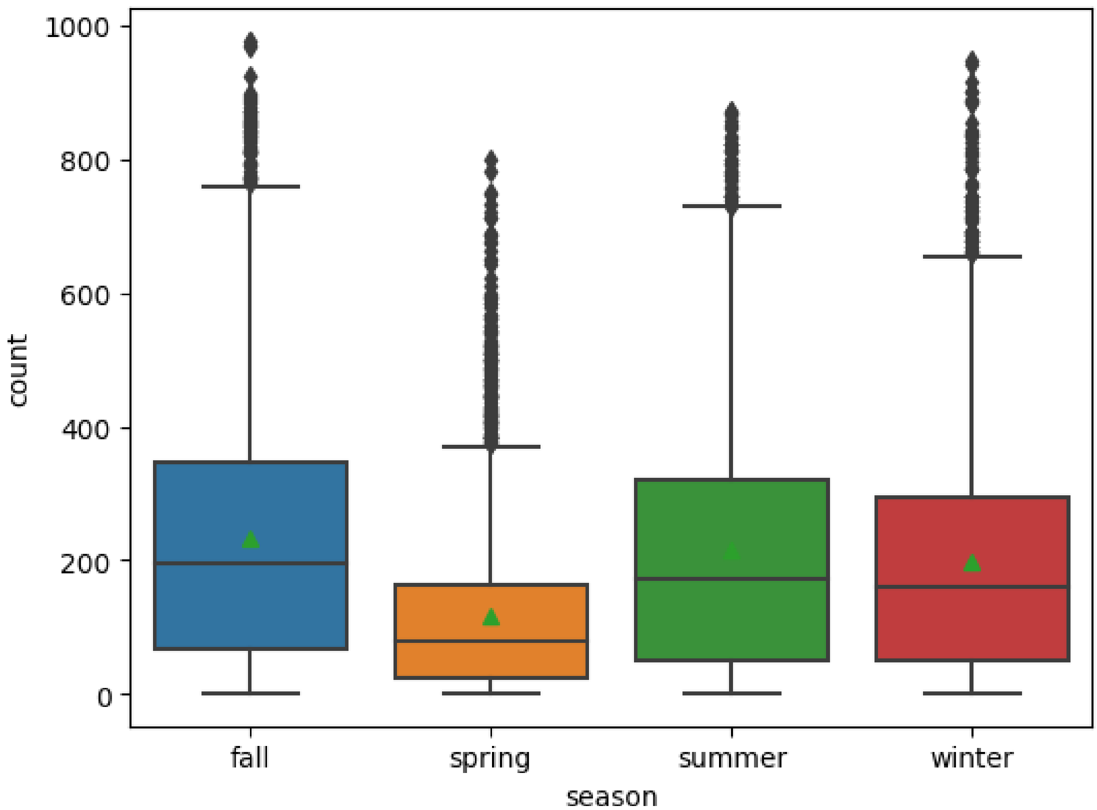
season	count	mean	std	min	25%	50%	75%	max
fall	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0
spring	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0
summer	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0
winter	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0

```
df_season_spring = df.loc[df['season'] == 'spring', 'count']
df_season_summer = df.loc[df['season'] == 'summer', 'count']
df_season_fall = df.loc[df['season'] == 'fall', 'count']
df_season_winter = df.loc[df['season'] == 'winter', 'count']
len(df_season_spring), len(df_season_summer), len(df_season_fall),
len(df_season_winter)
```

```
(2686, 2733, 2733, 2734)
```

```
sns.boxplot(data = df, x = 'season', y = 'count', showmeans = True)
plt.plot()
```

```
[]
```



STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Mean of cycle rented per hour is same for season 1,2,3 and 4.
- **Alternate Hypothesis (HA)** -Mean of cycle rented per hour is different for season 1,2,3 and 4.

STEP-2 : Checking for basic assumpitons for the hypothesis

---

1. **Normality check** using QQ Plot. If the distribution is not normal, use **BOX-COX transform** to transform it to normal distribution.
2. Homogeneity of Variances using **Levene's test**
3. Each observations are **independent**.

STEP-3: Define Test statistics

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

F=MSB/MSW

Under H0, the test statistic should follow **F-Distribution**.

STEP-4: Decide the kind of test.

We will be performing **right tailed f-test**

STEP-5: Compute the p-value and fix value of alpha.

---

we will be computing the anova-test p-value using the **f_oneway** function using scipy.stats. We set our alpha to be **0.05**

STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0. p-val > alpha : Accept H0 p-val < alpha : Reject H0

The one-way ANOVA compares the means between the groups you are interested in and determines whether any of those means are statistically significantly different from each other.

Specifically, it tests the null hypothesis (H0):

μ1 = μ2 = μ3 = ... = μk

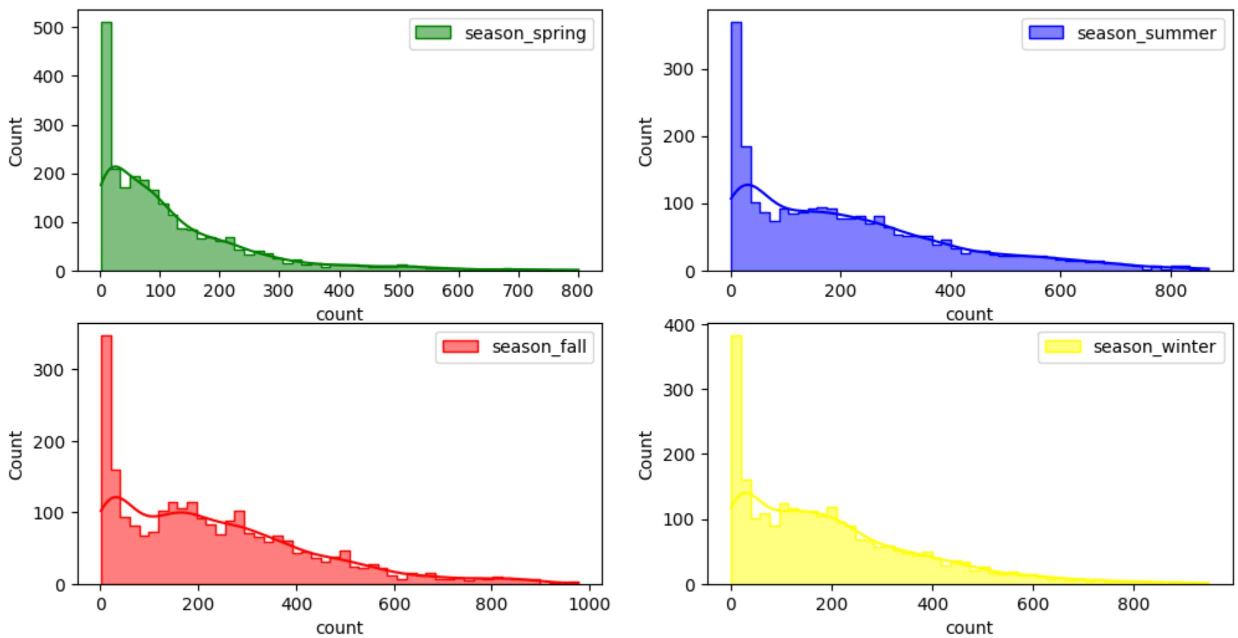
where, μ = group mean and k = number of groups.

If, however, the one-way ANOVA returns a statistically significant result, we accept the alternative hypothesis (HA), which is that there are at least two group means that are statistically significantly different from each other.

Visual Tests to know if the samples follow normal distribution

```
plt.figure(figsize = (12, 6))
plt.subplot(2, 2, 1)
sns.histplot(df_season_spring.sample(2500), bins = 50,
             element = 'step', color = 'green', kde = True, label =
'season_spring')
plt.legend()
plt.subplot(2, 2, 2)
sns.histplot(df_season_summer.sample(2500), bins = 50,
             element = 'step', color = 'blue', kde = True, label =
'season_summer')
plt.legend()
plt.subplot(2, 2, 3)
```

```
sns.histplot(df_season_fall.sample(2500), bins = 50,
             element = 'step', color = 'red', kde = True, label =
'season_fall')
plt.legend()
plt.subplot(2, 2, 4)
sns.histplot(df_season_winter.sample(2500), bins = 50,
             element = 'step', color = 'yellow', kde = True, label =
'season_winter')
plt.legend()
plt.plot()

[]
```

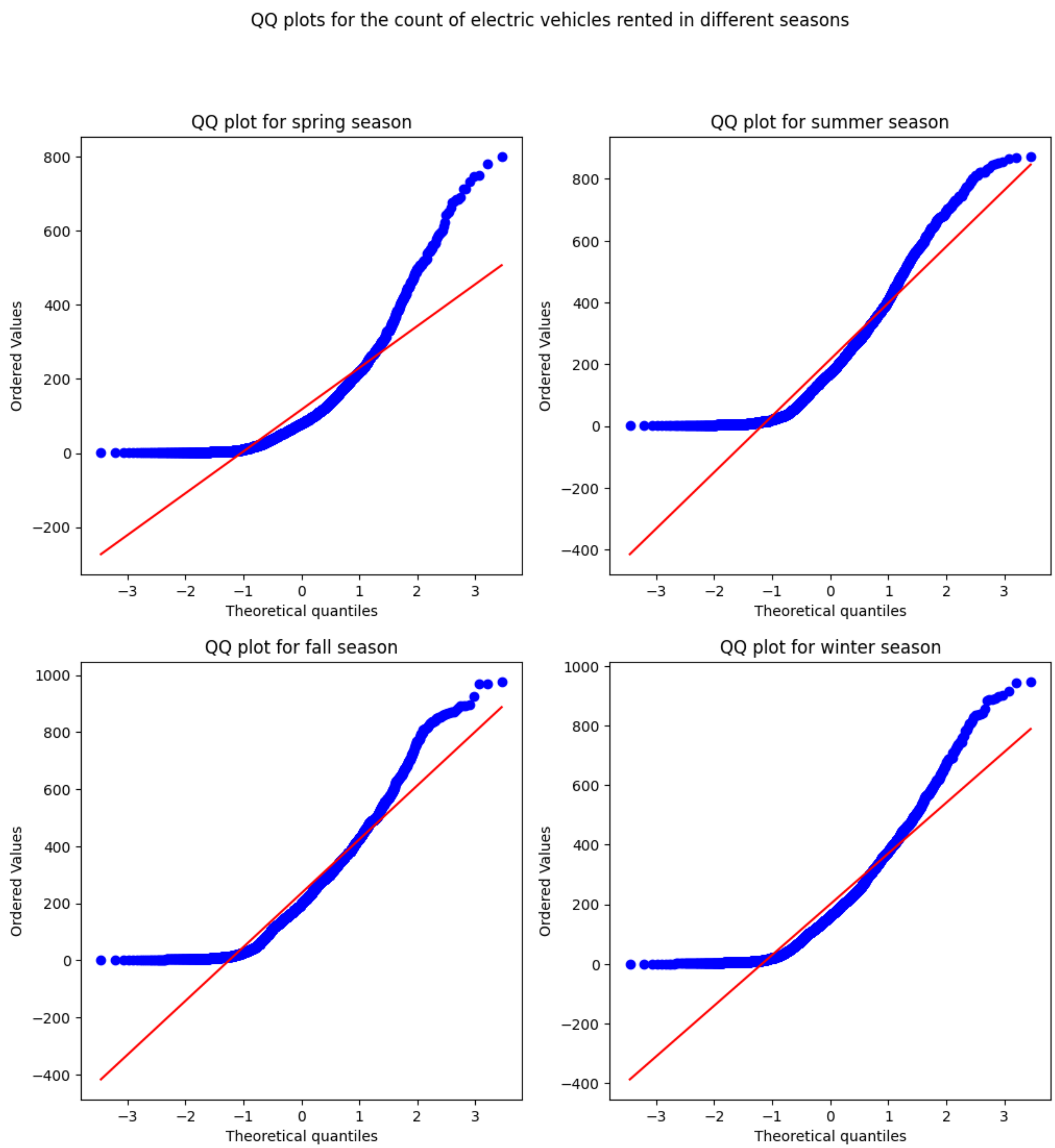


- It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
plt.figure(figsize = (12, 12))
plt.subplot(2, 2, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in
different seasons')
spy.probplot(df_season_spring.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for spring season')

plt.subplot(2, 2, 2)
spy.probplot(df_season_summer.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for summer season')

plt.subplot(2, 2, 3)
```

```
spy.probplot(df_season_fall.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for fall season')

plt.subplot(2, 2, 4)
spy.probplot(df_season_winter.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for winter season')
plt.plot()

[]
```



- It can be inferred from the above plots that the distributions do not follow normal distribution.

It can be seen from the above plots that the samples do not come from normal distribution.

- Applying Shapiro-Wilk test for normality

H_0 : The sample **follows normal distribution** H_1 : The sample **does not follow normal distribution**

alpha = 0.05

Test Statistics : **Shapiro-Wilk test for normality**

```
test_stat, p_value = spy.shapiro(df_season_spring.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 0.0
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df_season_summer.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.7158084700470534e-37
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df_season_fall.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 4.144913041116242e-35
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df_season_winter.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.1838389532686226e-38
The sample does not follow normal distribution
```

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
transformed_df_season_spring =
spy.boxcox(df_season_spring.sample(2500))[0]
test_stat, p_value = spy.shapiro(transformed_df_season_spring)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 6.265174485572999e-17
The sample does not follow normal distribution
```

```
transformed_df_season_summer =
spy.boxcox(df_season_summer.sample(2500))[0]
test_stat, p_value = spy.shapiro(transformed_df_season_summer)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 2.9526242257339083e-21
The sample does not follow normal distribution
```

```
transformed_df_season_fall = spy.boxcox(df_season_fall.sample(2500))
[0]
test_stat, p_value = spy.shapiro(transformed_df_season_fall)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 3.662290867187029e-21
The sample does not follow normal distribution
```

```
transformed_df_season_winter =
spy.boxcox(df_season_winter.sample(2500))[0]
test_stat, p_value = spy.shapiro(transformed_df_season_winter)
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 1.9496411808388168e-19
The sample does not follow normal distribution
```

- Even after applying the boxcox transformation on each of the season data, the samples do not follow normal distribution.

Homogeneity of Variances using Levene's test

```
# Null Hypothesis(H0) - Homogenous Variance

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df_season_spring.sample(2500),
                                df_season_summer.sample(2500),
                                df_season_fall.sample(2500),
                                df_season_winter.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')

p-value 2.7133500329584075e-111
The samples do not have  Homogenous Variance
```

Since the samples are not normally distributed and do not have the same variance, f_oneway test cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal-Wallis H-test for independent samples.

```
# Ho : Mean no. of cycles rented is same for different weather
# Ha : Mean no. of cycles rented is different for different weather
# Assuming significance Level to be 0.05
alpha = 0.05
test_stat, p_value = spy.kruskal(df_season_spring, df_season_summer,
df_season_fall,df_season_winter)
print('Test Statistic =', test_stat)
print('p value =', p_value)

Test Statistic = 699.6668548181988
p value = 2.479008372608633e-151
```

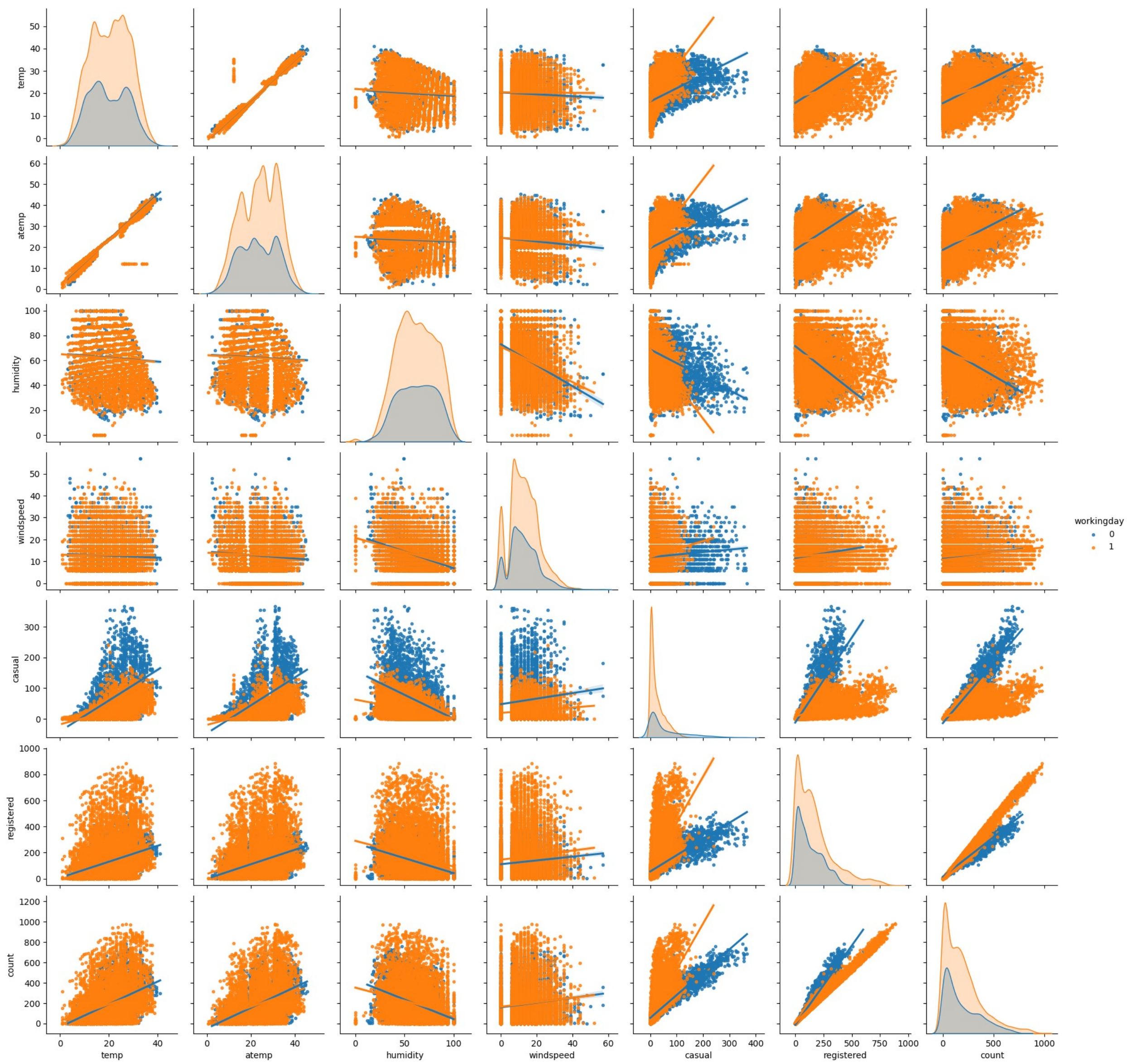
Comparing p value with significance level

```
if p_value < alpha:
    print('Reject Null Hypothesis')
else:
    print('Failed to reject Null Hypothesis')

Reject Null Hypothesis
```

Therefore, the average number of rental bikes is statistically different for different seasons.

```
sns.pairplot(data = df,
             kind = 'reg',
```

```
             hue = 'workingday',
             markers = '.')
plt.plot()

[]
```




```
# Assuming 'day' column contains categorical data like 'Saturday'
# If 'day' column is categorical, we should drop it before encoding
df_numeric = df.drop(columns=['day'])

# One-hot encoding categorical columns
df_encoded = pd.get_dummies(df_numeric)

# Compute correlation matrix
```

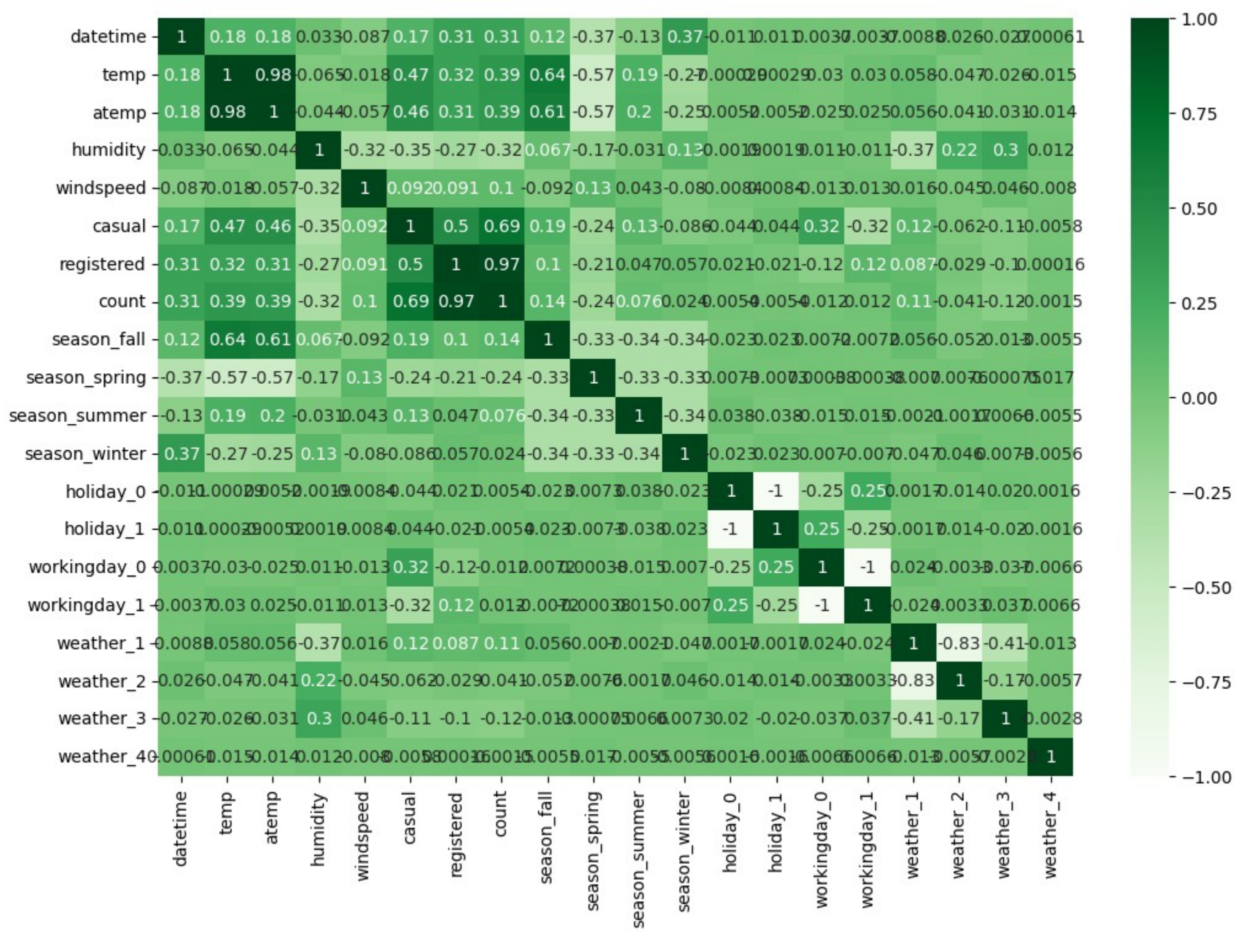
```python
corr_data = df_encoded.corr()
print(corr_data)
```

```
                datetime      temp     atemp  humidity  windspeed
casual  \
datetime        1.000000  0.180986  0.181823  0.032856  -0.086888
0.172728
temp            0.180986  1.000000  0.984948 -0.064949  -0.017852
0.467097
atemp           0.181823  0.984948  1.000000 -0.043536  -0.057473
0.462067
humidity        0.032856 -0.064949 -0.043536  1.000000  -0.318607 -
0.348187
windspeed      -0.086888 -0.017852 -0.057473 -0.318607   1.000000
0.092276
casual          0.172728  0.467097  0.462067 -0.348187   0.092276
1.000000
registered      0.314879  0.318571  0.314635 -0.265458   0.091052
0.497250
count           0.310187  0.394454  0.389784 -0.317371   0.101369
0.690414
season_fall     0.122125  0.635975  0.607090  0.067308  -0.091521
0.187726
season_spring -0.367648 -0.565655 -0.569082 -0.166208   0.128819 -
0.235222
season_summer -0.130557  0.192661  0.204421 -0.031095   0.042991
0.132405
season_winter  0.373909 -0.266220 -0.245690  0.129018  -0.079535 -
0.086258
holiday_0      -0.010988 -0.000295  0.005215 -0.001929  -0.008409 -
0.043799
holiday_1       0.010988  0.000295 -0.005215  0.001929   0.008409
0.043799
workingday_0    0.003658 -0.029966 -0.024660  0.010880  -0.013373
0.319111
workingday_1   -0.003658  0.029966  0.024660 -0.010880   0.013373 -
0.319111
weather_1      -0.008822  0.058430  0.055825 -0.374837   0.015920
0.119728
weather_2       0.026342 -0.046925 -0.040792  0.222398  -0.045016 -
0.062184
weather_3      -0.027404 -0.025715 -0.031154  0.295894   0.045597 -
0.108853
weather_4       0.000615 -0.014800 -0.013901  0.012010  -0.007979 -
0.005760

               registered     count  season_fall  season_spring  \
datetime         0.314879  0.310187     0.122125      -0.367648
temp             0.318571  0.394454     0.635975      -0.565655
atemp            0.314635  0.389784     0.607090      -0.569082
```

```
humidity        -0.265458 -0.317371     0.067308      -0.166208
windspeed        0.091052  0.101369    -0.091521       0.128819
casual           0.497250  0.690414     0.187726      -0.235222
registered       1.000000  0.970948     0.102142      -0.207278
count            0.970948  1.000000     0.136942      -0.237704
season_fall      0.102142  0.136942     1.000000      -0.331365
season_spring   -0.207278 -0.237704    -0.331365       1.000000
season_summer    0.046969  0.075681    -0.335214      -0.331365
season_winter    0.056961  0.023704    -0.335296      -0.331446
holiday_0        0.020956  0.005393    -0.022790       0.007336
holiday_1       -0.020956 -0.005393     0.022790      -0.007336
workingday_0    -0.119460 -0.011594     0.007194       0.000379
workingday_1     0.119460  0.011594    -0.007194      -0.000379
weather_1        0.086621  0.105246     0.055660      -0.006996
weather_2       -0.028997 -0.041329    -0.051895       0.007644
weather_3       -0.104936 -0.117519    -0.013089      -0.000750
weather_4        0.000155 -0.001459    -0.005549       0.016747

               season_summer  season_winter  holiday_0  holiday_1  \
datetime           -0.130557       0.373909  -0.010988   0.010988
temp                0.192661      -0.266220  -0.000295   0.000295
atemp               0.204421      -0.245690   0.005215  -0.005215
humidity           -0.031095       0.129018  -0.001929   0.001929
windspeed           0.042991      -0.079535  -0.008409   0.008409
casual              0.132405      -0.086258  -0.043799   0.043799
registered          0.046969       0.056961   0.020956  -0.020956
count               0.075681       0.023704   0.005393  -0.005393
season_fall        -0.335214      -0.335296  -0.022790   0.022790
season_spring      -0.331365      -0.331446   0.007336  -0.007336
season_summer       1.000000      -0.335296   0.038250  -0.038250
season_winter      -0.335296       1.000000  -0.022751   0.022751
holiday_0           0.038250      -0.022751   1.000000  -1.000000
holiday_1          -0.038250       0.022751  -1.000000   1.000000
workingday_0       -0.014620       0.007048  -0.250491   0.250491
workingday_1        0.014620      -0.007048   0.250491  -0.250491
weather_1          -0.002057      -0.046642   0.001708  -0.001708
weather_2          -0.001687       0.045976  -0.013868   0.013868
weather_3           0.006556       0.007278   0.019514  -0.019514
weather_4          -0.005549      -0.005551   0.001644  -0.001644

               workingday_0  workingday_1  weather_1  weather_2
weather_3  \
datetime           0.003658     -0.003658  -0.008822   0.026342  -
0.027404
temp              -0.029966      0.029966   0.058430  -0.046925  -
0.025715
atemp             -0.024660      0.024660   0.055825  -0.040792  -
0.031154
humidity           0.010880     -0.010880  -0.374837   0.222398
```

```
0.295894
windspeed          -0.013373      0.013373   0.015920  -0.045016
0.045597
casual              0.319111     -0.319111   0.119728  -0.062184  -
0.108853
registered         -0.119460      0.119460   0.086621  -0.028997  -
0.104936
count              -0.011594      0.011594   0.105246  -0.041329  -
0.117519
season_fall         0.007194     -0.007194   0.055660  -0.051895  -
0.013089
season_spring       0.000379     -0.000379  -0.006996   0.007644  -
0.000750
season_summer      -0.014620      0.014620  -0.002057  -0.001687
0.006556
season_winter       0.007048     -0.007048  -0.046642   0.045976
0.007278
holiday_0          -0.250491      0.250491   0.001708  -0.013868
0.019514
holiday_1           0.250491     -0.250491  -0.001708   0.013868  -
0.019514
workingday_0        1.000000     -1.000000   0.024078  -0.003324  -
0.036643
workingday_1       -1.000000      1.000000  -0.024078   0.003324
0.036643
weather_1           0.024078     -0.024078   1.000000  -0.827798  -
0.408402
weather_2          -0.003324      0.003324  -0.827798   1.000000  -
0.173644
weather_3          -0.036643      0.036643  -0.408402  -0.173644
1.000000
weather_4          -0.006562      0.006562  -0.013374  -0.005686  -
0.002805

               weather_4
datetime        0.000615
temp           -0.014800
atemp          -0.013901
humidity        0.012010
windspeed      -0.007979
casual         -0.005760
registered      0.000155
count          -0.001459
season_fall    -0.005549
season_spring   0.016747
season_summer  -0.005549
season_winter  -0.005551
holiday_0       0.001644
holiday_1      -0.001644
```

```
workingday_0   -0.006562
workingday_1    0.006562
weather_1      -0.013374
weather_2      -0.005686
weather_3      -0.002805
weather_4       1.000000

plt.figure(figsize = (12, 8))
sns.heatmap(data = corr_data, cmap = 'Greens', annot = True, vmin = -
1, vmax = 1)
plt.plot()

[]
```



- Very High Correlation (> 0.9) exists between columns [atemp, temp] and [count, registered]
- High positively / negatively correlation (0.7 - 0.9) does not exist between any columns.
- Moderate positive correlation (0.5 - 0.7) exists between columns [casual, count], [casual, registered].
- Low Positive correlation (0.3 - 0.5) exists between columns [count, temp], [count, atemp], [casual, atemp]

- Negligible correlation exists between all other combinations of columns.

Insights

- More than 80 % of the time, the humidity value is greater than 40. Thus for most of the time, humidity level varies from optimum to too moist.
- The data is given from Timestamp('2011-01-01 00:00:00') to Timestamp('2012-12-19 23:00:00'). The total time period for which the data is given is '718 days 23:00:00'.
- Out of every 100 users, around 19 are casual users and 81 are registered users.
- The mean total hourly count of rental bikes is 144 for the year 2011 and 239 for the year 2012. An annual growth rate of 65.41 % can be seen in the demand of electric vehicles on an hourly basis.
- There is a seasonal pattern in the count of rental bikes, with higher demand during the spring and summer months, a slight decline in the fall, and a further decrease in the winter months.
- The average hourly count of rental bikes is the lowest in the month of January followed by February and March.
- There is a distinct fluctuation in count throughout the day, with low counts during early morning hours, a sudden increase in the morning, a peak count in the afternoon, and a gradual decline in the evening and nighttime.
- More than 80 % of the time, the temperature is less than 28 degrees celcius.
- More than 85 % of the total, windspeed data has a value of less than 20.
- The hourly count of total rental bikes is the highest in the clear and cloudy weather, followed by the misty weather and rainy weather. There are very few records for extreme weather conditions.
- The mean hourly count of the total rental bikes is statistically similar for both working and non- working days.
- The hourly total number of rental bikes is statistically different for different weathers.
- There is no statistically significant dependency of weather 1, 2, 3 on season based on the average hourly total number of bikes rented.
- The hourly total number of rental bikes is statistically different for different seasons.
- There is statistically significant dependency of weather and season based on the hourly total number of bikes rented.

Recommendations

- **User Segmentation**: Given that around 81% of users are registered, and the remaining 19% are casual, Yulu can tailor its marketing and communication strategies accordingly. Provide loyalty programs, exclusive offers, or personalized recommendations for registered users to encourage repeat business. For casual users, focus on providing a seamless rental experience and promoting the benefits of bike rentals for occasional use.

- **Seasonal Marketing**: Since there is a clear seasonal pattern in the count of rental bikes, Yulu can adjust its marketing strategies accordingly. Focus on promoting bike rentals during the spring and summer months when there is higher demand. Offer seasonal discounts or special packages to attract more customers during these periods.

- **Time-based Pricing**: Take advantage of the hourly fluctuation in bike rental counts throughout the day. Consider implementing time-based pricing where rental rates are lower during off-peak hours and higher during peak hours. This can encourage customers to rent bikes during less busy times, balancing out the demand and optimizing the resources.
- **Optimize Inventory**: Analyze the demand patterns during different months and adjust the inventory accordingly. During months with lower rental counts such as January, February, and March, Yulu can optimize its inventory levels to avoid excess bikes. On the other hand, during peak months, ensure having sufficient bikes available to meet the higher demand.
- **Improve Weather Data Collection**: Given the lack of records for extreme weather conditions, consider improving the data collection process for such scenarios. Having more data on extreme weather conditions can help to understand customer behavior and adjust the operations accordingly, such as offering specialized bike models for different weather conditions or implementing safety measures during extreme weather.
- **Collaborations with Weather Services**: Consider collaborating with weather services to provide real-time weather updates and forecasts to potential customers. Incorporate weather information into your marketing campaigns or rental app to showcase the ideal biking conditions and attract users who prefer certain weather conditions.
- **Seasonal Bike Maintenance**: Allocate resources for seasonal bike maintenance. Before the peak seasons, conduct thorough maintenance checks on the bike fleet to ensure they are in top condition. Regularly inspect and service bikes throughout the year to prevent breakdowns and maximize customer satisfaction.
- **Customer Feedback and Reviews**: Encourage customers to provide feedback and reviews on their biking experience. Collecting feedback can help identify areas for improvement, understand customer preferences, and tailor the services to better meet customer expectations.
- **Social Media Marketing**: Leverage social media platforms to promote the electric bike rental services. Share captivating visuals of biking experiences in different weather conditions, highlight customer testimonials, and engage with potential customers through interactive posts and contests. Utilize targeted advertising campaigns to reach specific customer segments and drive more bookings.
- **Special Occasion Discounts**: Since Yulu focusses on providing a sustainable solution for vehicular pollution, it should give special discounts on the occassions like Zero Emissions Day (21st September), Earth day (22nd April), World Environment Day (5th June) etc in order to attract new users.
- **Weather-based Promotions**: Recognize the impact of weather on bike rentals. Create weather-based promotions that target customers during clear and cloudy

weather, as these conditions show the highest rental counts. Yulu can offer weather-specific discounts to attract more customers during these favorable weather conditions.

- **Customer Comfort**: Since humidity levels are generally high and temperature is often below 28 degrees Celsius, consider providing amenities like umbrellas, rain jackets, or water bottles to enhance the comfort and convenience of the customers. These small touches can contribute to a positive customer experience and encourage repeat business.