Busines Case - Tech-driven Mobility Company - Hypothesis Testing

Problem Statement

X company 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. How you can help here?

The company wants to know:

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

In [2]:
```
#importing required libraries for data analysis and data visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#importing required libraries for hypothesis testing
from scipy.stats import norm, t, shapiro, levene, mannwhitneyu, probplot, kruskal
from scipy.stats import ttest_1samp, ttest_ind, ttest_rel, f_oneway, chi2_contingency, chi2
```

In [3]:
```
#importing required data-set
df = pd.read_csv('D:\\Scaler\\Scaler\\Hypothesis Testing\\Business Case\\bike_sharing.csv')
```

In [4]:
```
df.head()
```

Out[4]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	c
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	

In [5]:
```
df['datetime'] = pd.to_datetime(df['datetime'])
```

In [6]:
```
df.dtypes
```

Out[6]:
```
datetime      datetime64[ns]
season                 int64
holiday                int64
workingday             int64
weather                int64
temp                 float64
atemp                float64
humidity               int64
windspeed            float64
casual                 int64
registered             int64
count                  int64
dtype: object
```

In [7]:
```
df['datetime'].min(), df['datetime'].max()
```

Out[7]:
```
(Timestamp('2011-01-01 00:00:00'), Timestamp('2012-12-19 23:00:00'))
```

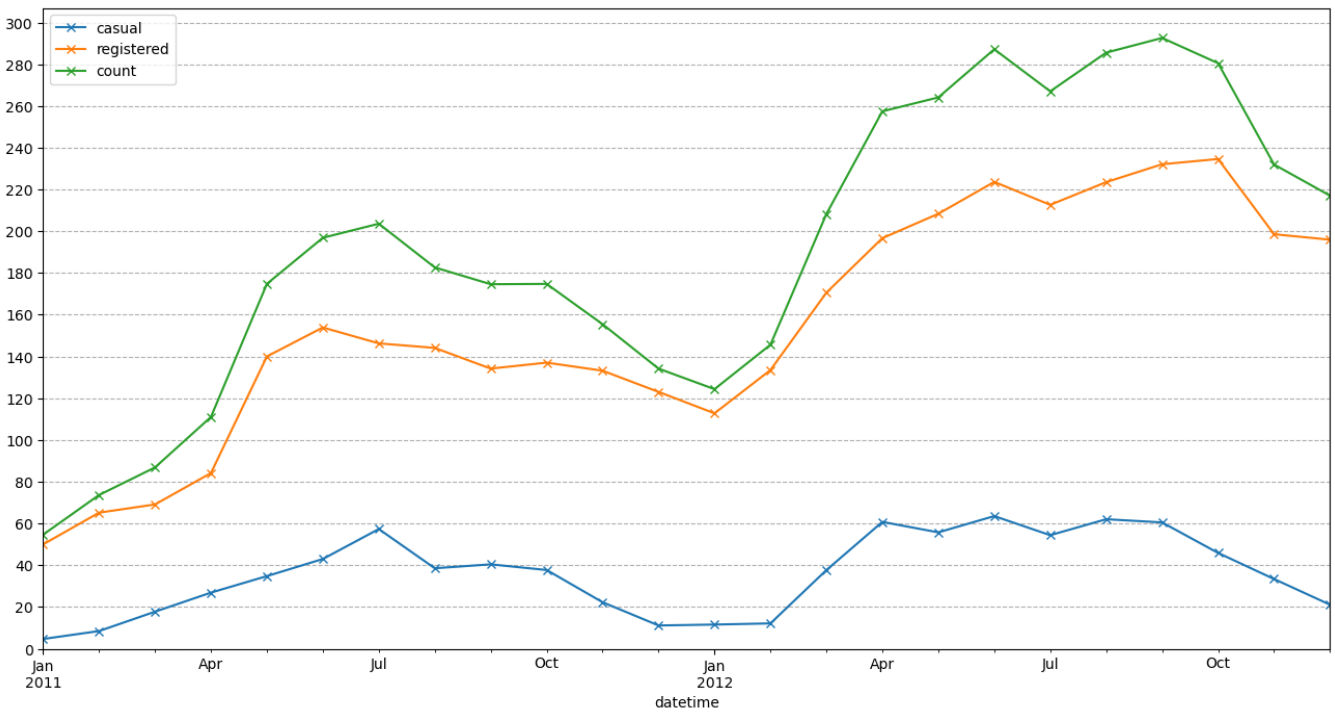
In [8]:
```
df['datetime'].max() - df['datetime'].min()
```

Out[8]:
```
Timedelta('718 days 23:00:00')
```

In [9]:
```
df['day'] = df['datetime'].dt.day_name()
```

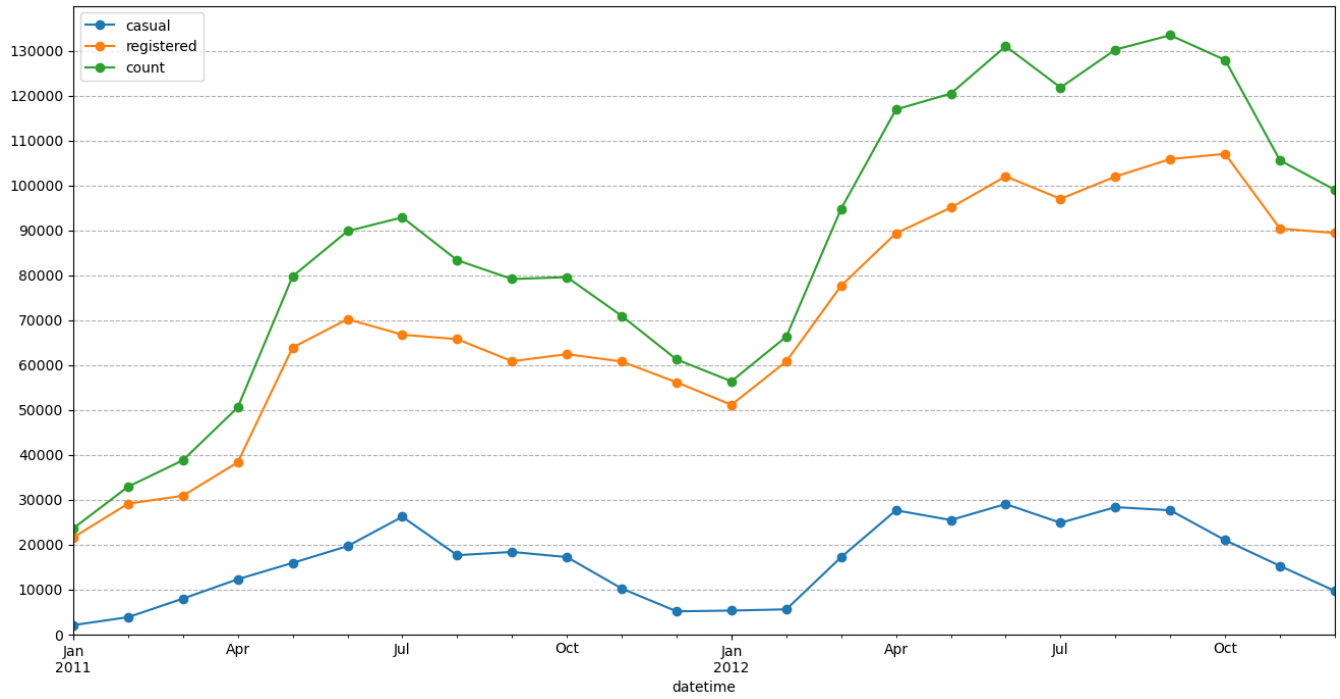
In [10]:
```
df.set_index('datetime', inplace=True)
```

In [11]:
```
plt.figure(figsize = (16,8))
# .resample() method to resample the DataFrame based on the frequency provided. In this case,
# ..'M' stands for monthly resampling. It means that the data will be aggregated and grouped
df.resample('M')['casual'].mean().plot(kind='line', legend = 'casual', marker ='x')
df.resample('M')['registered'].mean().plot(kind='line', legend ='registered', marker='x')
df.resample('M')['count'].mean().plot(kind='line', legend ='count', marker='x')
# A lineplot on a monthly basis, and calculating the MEAN VALUE of 'casual', 'registered' and
plt.grid(axis ='y', linestyle = '--')
plt.yticks(np.arange(0,301,20))
plt.ylim(0,)
plt.show()
```



In [12]:
```
plt.figure(figsize=(16,8))

# A lineplot on a monthly basis, and calculating the SUM of casual', 'registered' and 'count'
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= '--')
plt.yticks(np.arange(0, 130001,10000))
plt.ylim(0,) # setting the lower y-axis limit to 0
plt.show()
```



In [13]:
```
df1 = df.resample('Y')['count'].mean().to_frame().reset_index()
#creating a new column for previous year value
df1['prev_count'] = df1['count'].shift(1)

#calculating the increase/decrease percentage
df1['change'] = (df1['count'] - df1['prev_count']) * 100/ df1['prev_count']
df1
```

Out[13]:

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

In [14]:
```
df.reset_index(inplace = True)
df.head()
```

Out[14]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	c
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	

In [15]:
```
df1 = df.groupby(by = df['datetime'].dt.month)['count'].mean().reset_index()
df1.rename(columns={'datetime': 'month'}, inplace=True)
# Define the order of months
month_order = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
]

# Convert the 'month' column to a Categorical data type with the specified order
df1['month'] = pd.Categorical(df1['month'].apply(lambda x: month_order[x-1]),
                              categories=month_order, ordered = True)

# Sort the DataFrame based on the categorical order of months
df1.sort_values('month', inplace = True)

# Resetting the index after sorting
df1.reset_index(drop = True, 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 the previous mo
df1['change'] = (df1['count'] - df1['prev_count']) *100 / df1['prev_count']

# Set the 'month' column as the index
df1.set_index('month', inplace = True)

# Display the DataFrame
df1
```

Out[15]:

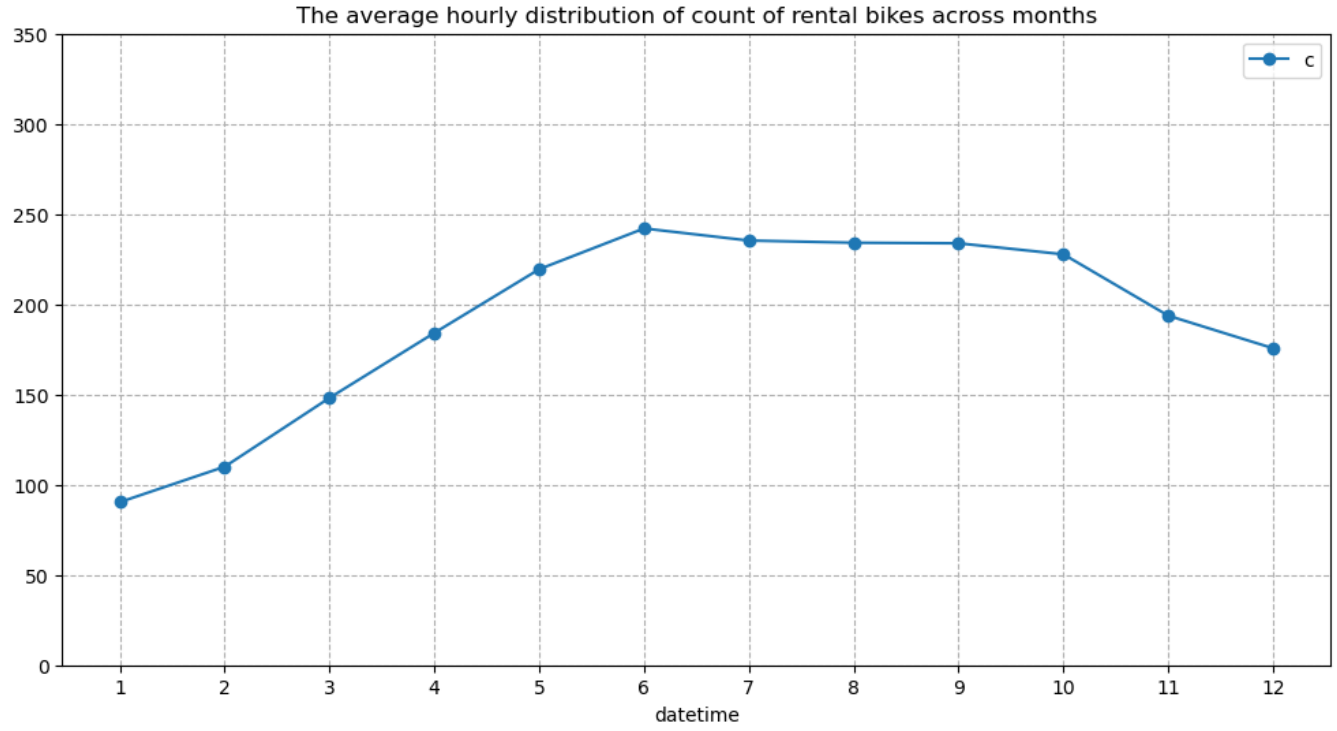
month	count	prev_count	change
January	90.366516	NaN	NaN
February	110.003330	90.366516	21.730188
March	148.169811	110.003330	34.695751
April	184.160616	148.169811	24.290241
May	219.459430	184.160616	19.167406
June	242.031798	219.459430	10.285440
July	235.325658	242.031798	-2.770768
August	234.118421	235.325658	-0.513007
September	233.805281	234.118421	-0.133753
October	227.699232	233.805281	-2.611596
November	193.677278	227.699232	-14.941620
December	175.614035	193.677278	-9.326465

In [16]:
```
# The resulting plot visualizes the average hourly distribution of the count of rental bikes
    # month, allowing for comparison and identification of any patterns or trends throughout

# 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 months")

# Grouping the DataFrame by the month and calculating the mean of the 'count' column for each
    # 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.
```

Out[16]:

```
[]
```



In [17]:

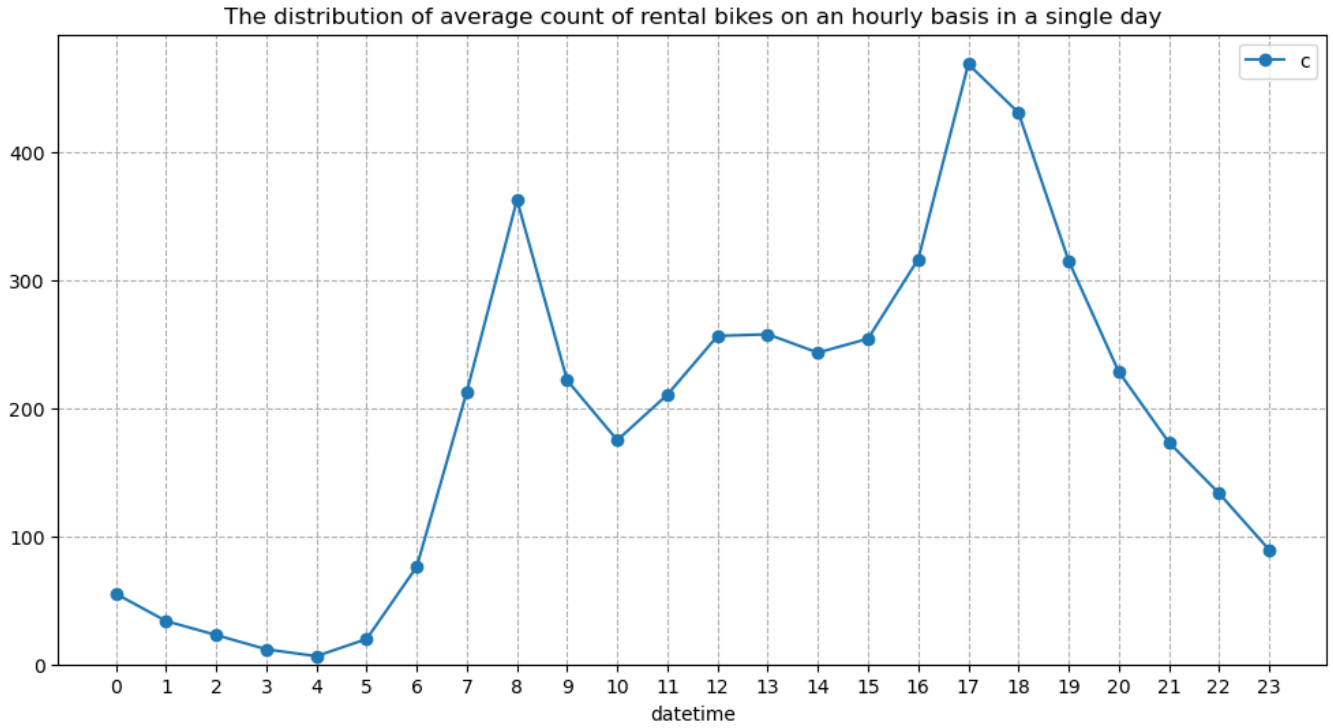
```
# Grouping the DataFrame by the hour
df1 = df.groupby(by = df['datetime'].dt.hour)['count'].mean().reset_index()
df1.rename(columns = {'datetime' : 'hour'}, inplace = True)

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

Out[17]:

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

In [18]:
```
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on an hourly basis in a single d
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()
```

Out[18]:
```
[]
```



In [ ]:

Univariate Analysis on discrete (Categorical) variables:

* season - to categorical

* holiday - to categorical

* workingday - to categorical

* weather - to categorical

Season

In [19]:
```
# 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'
    elif x == 4:
        return 'winter'

df['season'] = df['season'].apply(season_category)
```

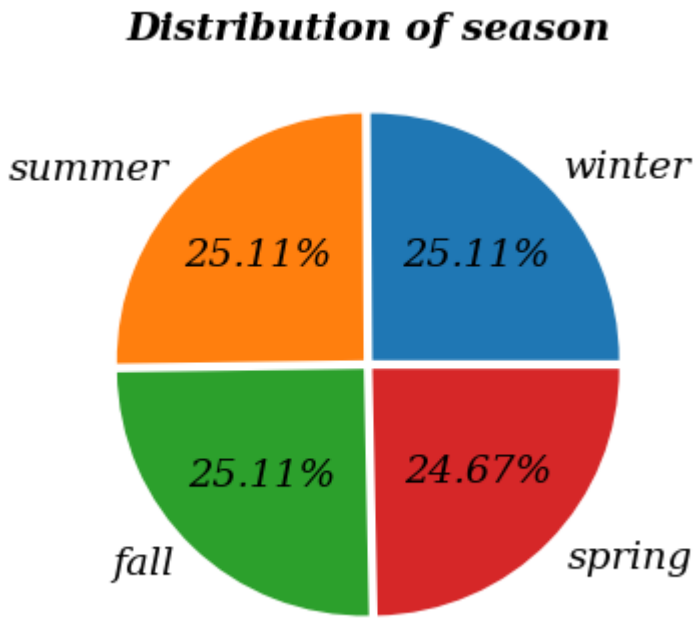
In [20]:
```
np.round(df['season'].value_counts(normalize = True)*100,2)
```

Out[20]:
```
winter    25.11
summer    25.11
fall      25.11
spring    24.67
Name: season, dtype: float64
```

In [21]:
```
# The below code generates a visually appealing pie chart to showcase the
    # distribution of seasons in the dataset

plt.figure(figsize = (4, 4))      # setting the figure size to 6*6
```

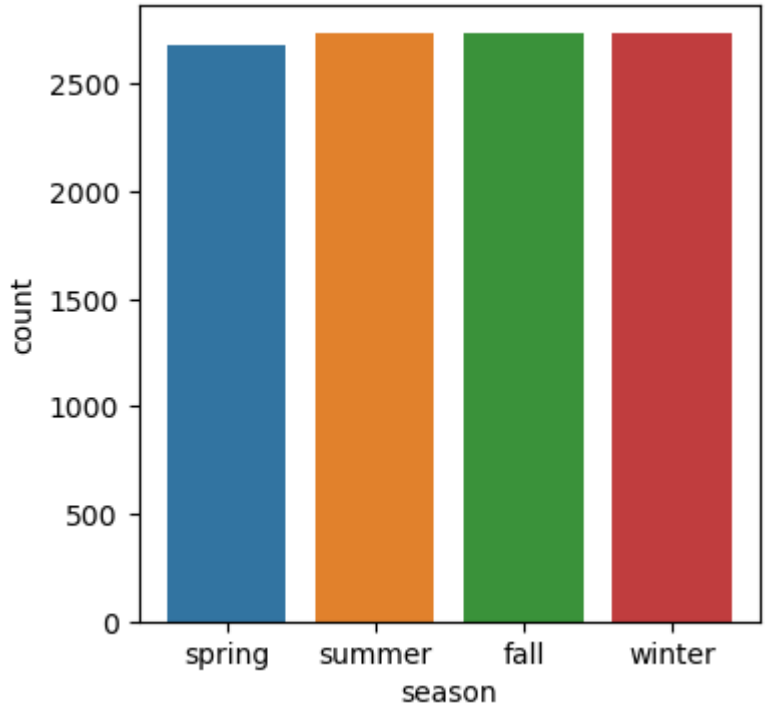
```
# setting the title of the plot
plt.title('Distribution of season', fontdict = {'fontsize' : 14,
                                                'fontweight' : 600,
                                                'fontstyle' : 'oblique',
                                                'fontfamily' : 'serif'})

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

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

plt.plot()     # displaying the plot
```

Out[21]:
```
[]
```



In [22]:
```
# The below code generates a visually appealing count plot to showcase the
    # distribution of season in the dataset

plt.figure(figsize = (4, 4))
sns.countplot(data = df, x = 'season')
plt.plot()   # displaying the plot
```

Out[22]:
```
[]
```



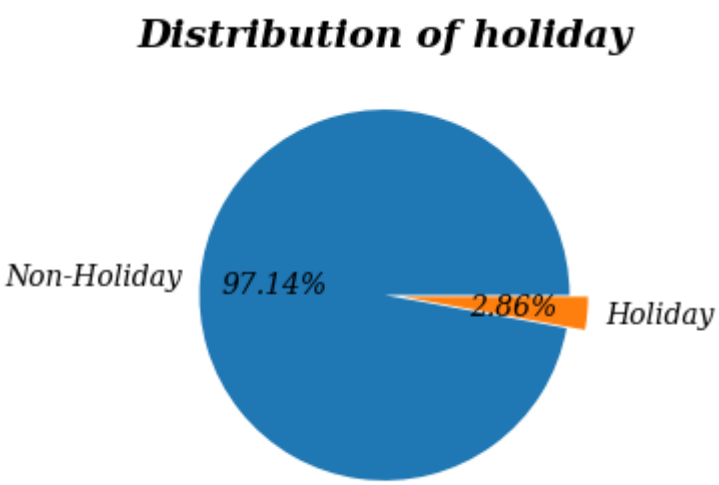
Holiday

In [23]:
```
np.round(df['holiday'].value_counts(normalize = True) * 100, 2)
```

Out[23]:
```
0    97.14
1     2.86
Name: holiday, dtype: float64
```

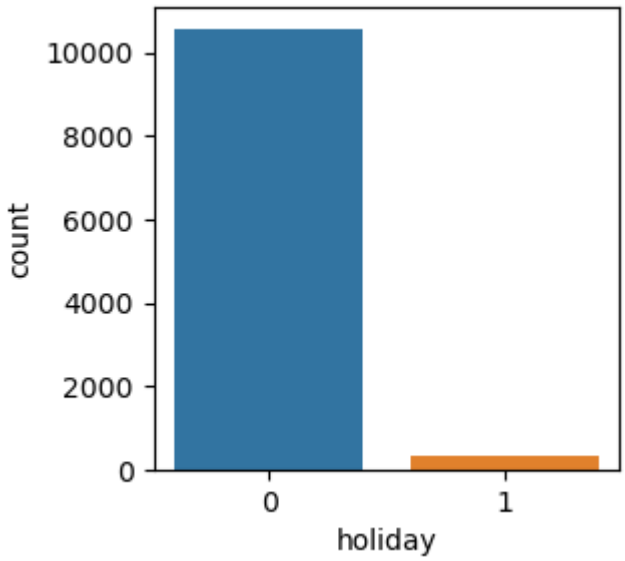
In [24]:
```
# The below code generates a visually appealing pie chart to showcase the
# distribution of holiday in the dataset

plt.figure(figsize=(3, 3))  # setting the figure size to 6*6

# setting the title of the plot
plt.title('Distribution of holiday', fontdict={'fontsize': 14,
                                               'fontweight': 600,
                                               'fontstyle': 'oblique',
                                               'fontfamily': 'serif'})

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

# Creating the pie-chart
plt.pie(x=df_holiday['holiday'],
        explode=[0, 0.1],
        labels=['Non-Holiday', 'Holiday'],
        autopct='%.2f%%',
        textprops={'fontsize': 10,
                   'fontstyle': 'oblique',
                   'fontfamily': 'serif',
                   'fontweight': 500})

plt.plot()  # displaying the plot
```

Out[24]:
```
[]
```



In [25]:
```
# The below code generates a visually appealing count plot to showcase the
    # distribution of holiday in the dataset

plt.figure(figsize=(3, 3))
sns.countplot(data = df, x = 'holiday')
plt.plot()       # displaying the chart
```

Out[25]:
```
[]
```



Workingday

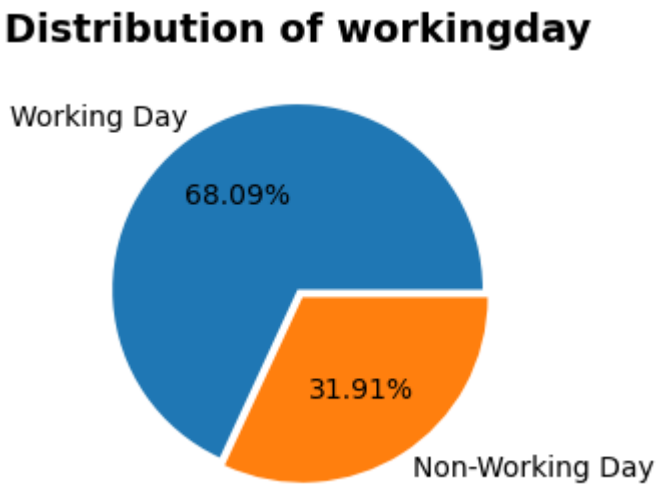
In [26]:
```
np.round(df['workingday'].value_counts(normalize= True)*100,2)
```

Out[26]:
```
1    68.09
0    31.91
Name: workingday, dtype: float64
```

In [27]:
```
plt.figure(figsize = (3,3))

plt.title('Distribution of workingday', fontdict = {'fontsize' : 14,
                                                    'fontweight' : 600})
df_workingday = np.round(df['workingday'].value_counts(normalize= True)*100,2).to_frame()

plt.pie(x=df_workingday['workingday'],
        explode = [0, 0.05],
        labels = ['Working Day', 'Non-Working Day'],
         autopct = '%.2f%%',
          textprops = {'fontsize' : 10,
                       'fontweight' : 500})
```

```
plt.plot()
```

Out[27]:
```
[]
```

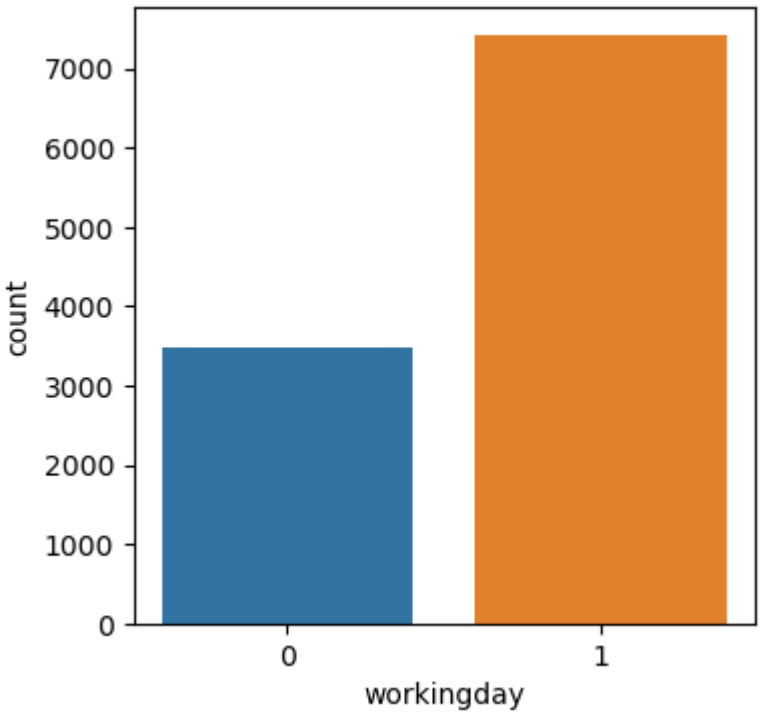


In [28]:
```
#plt.title?
plt.pie
```

Out[28]:
```
<function matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctd
istance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedg
eprops=None, textprops=None, center=(0, 0), frame=False, rotatelabels=False, *, normalize=Tru
e, hatch=None, data=None)>
```

In [29]:
```
# The below code generates a visually appealing count plot to showcase the
    # distribution of workingday in the dataset

plt.figure(figsize = (4,4))
sns.countplot(data = df, x = 'workingday')
plt.plot()      # displaying the chart
```

Out[29]:
```
[]
```



Weather

In [30]:
```
# 1: Clear, Few clouds, partly cloudy, partly cloudy,
# 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist,
```

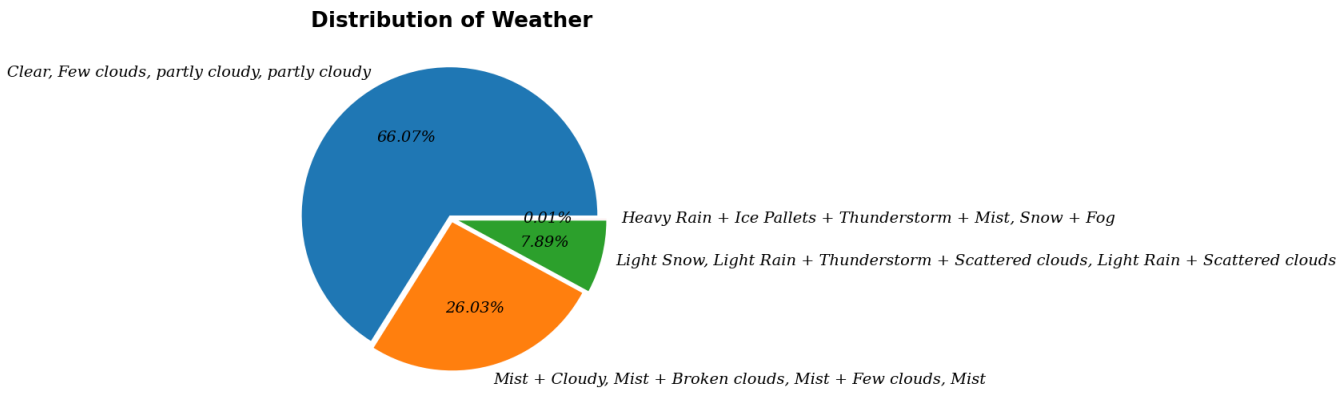
```
# 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
# 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

def weather_category(z):
    if z == 1:
        return "Clear, Few clouds, partly cloudy, partly cloudy"
    elif z == 2:
        return "Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist"
    elif z == 3:
        return "Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scatte
    elif z == 4:
        return "Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog"

df['weather'] = df['weather'].apply(weather_category)
```

In [31]:
```
np.round(df['weather'].value_counts(normalize = True)*100, 2)
```

Out[31]:
```
Clear, Few clouds, partly cloudy, partly cloudy                                 6
6.07
Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist                    2
6.03
Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
7.89
Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
0.01
Name: weather, dtype: float64
```

In [32]:
```
# The below code generates a visually appealing pie chart to showcase the
    # distribution of weather in the dataset

plt.figure(figsize = (6,6))
plt.title("Distribution of Weather", fontdict = {'fontsize' : 19, 'fontweight' : 600})

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

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

plt.plot()
```

Out[32]:
```
[]
```



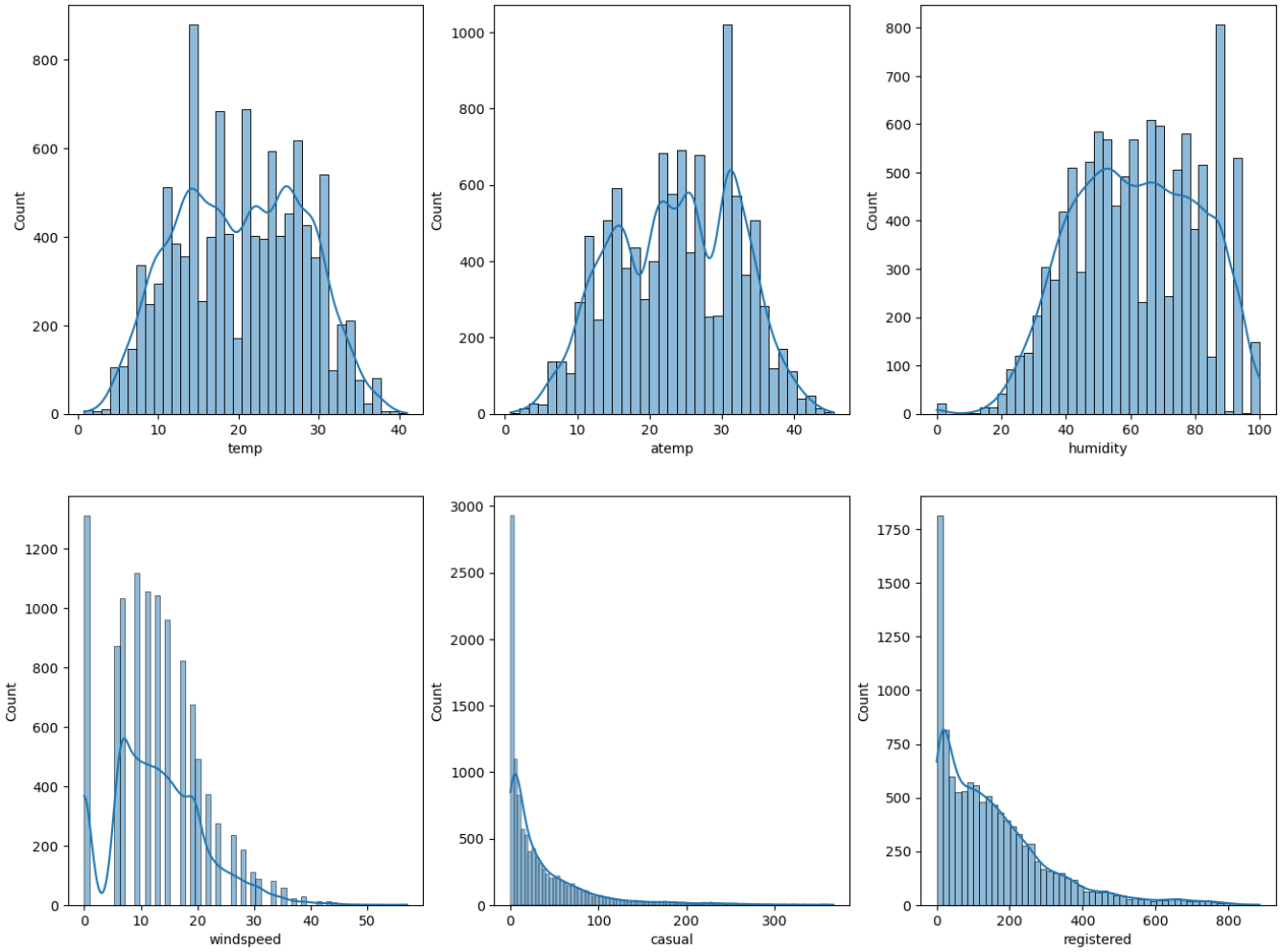
Insights:

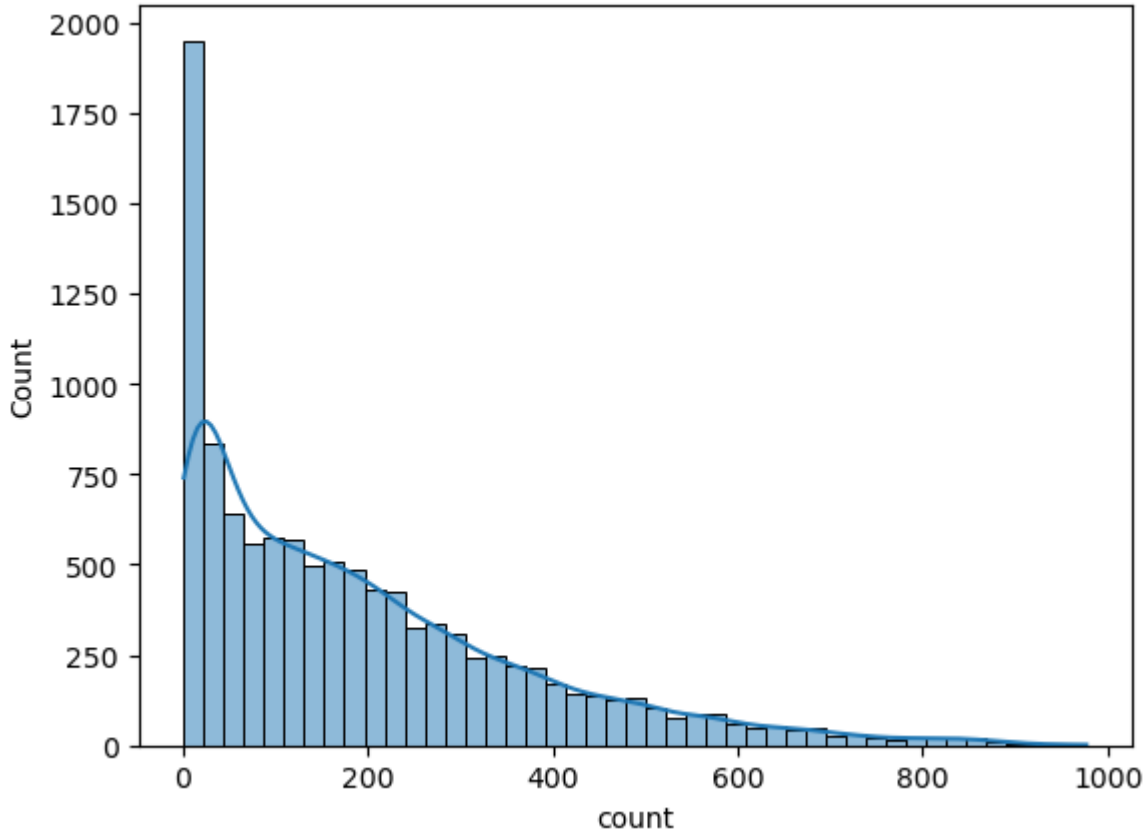
- Data looks common as it should be like equal number of days in each season, more working days and weather is mostly Clear, Few clouds, partly cloudy, partly cloudy.

Distribution Plots of the ContinuousVariable(s)

In [33]:
```
cont_val_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']
```

In [34]:
```
fig, axis = plt.subplots(nrows=2, ncols=3, figsize = (16,12))

index = 0

for row in range(2):
    for col in range(3):
        sns.histplot(df[cont_val_cols[index]], ax = axis[row,col], kde =  True)
        index += 1


plt.show()
sns.histplot(df[cont_val_cols[-1]], kde = True)
plt.show()
```





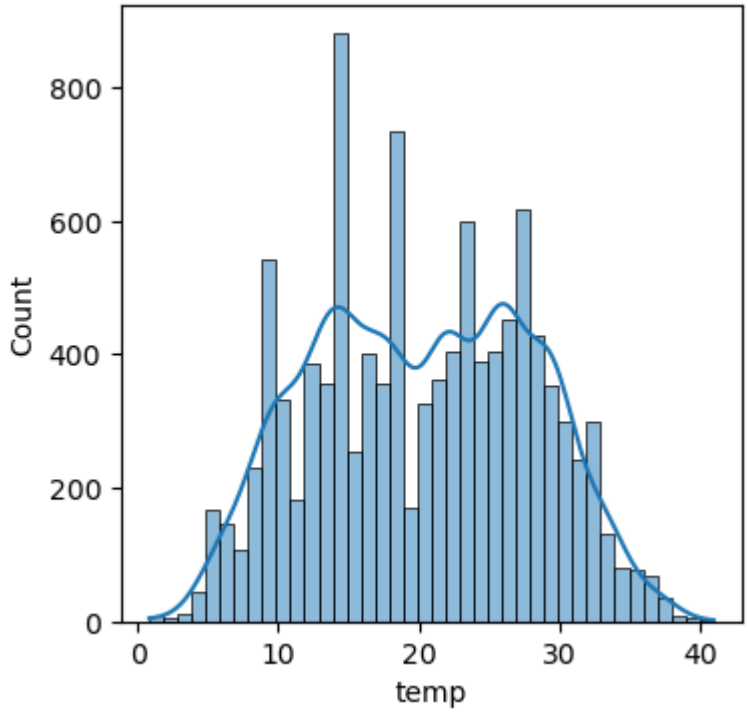
Insights:

1. temp, atemp and humidity seems to follow the Normal Distribution
2. windspeed follows the binomial distribution
3. casual, registered and count somewhat looks like Log Normal Distrinution

temp

In [35]:
```
# The below code generates a histogram plot for the 'temp' feature, showing the distribution
    # temperature values in the dataset.
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'temp', kde = True, bins = 40)
plt.plot()
```

Out[35]:
```
[]
```



In [36]:
```
temp_mean = np.round(df['temp'].mean(),2)
temp_std = np.round(df['temp'].std(),2)
temp_mean, temp_std
```

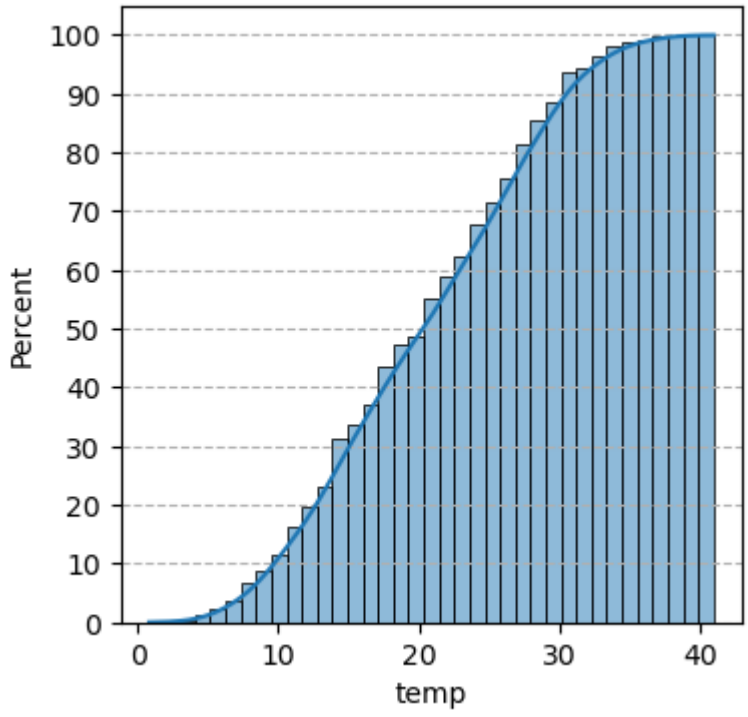
Out[36]:
```
(20.23, 7.79)
```

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

In [37]:
```
## The below code generates a histogram plot for the 'temp' feature, showing the
    # cumulative distribution of temperature values in the dataset.
```

In [38]:
```
plt.figure(figsize = (4,4))
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()
```

Out[38]:
```
[]
```

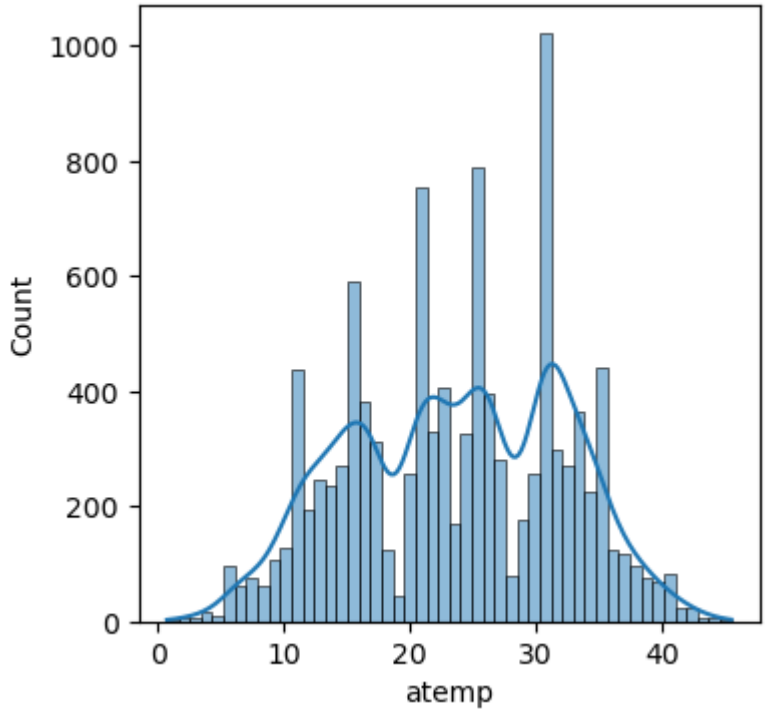


Insight: More than 80 % of the time, the temperature is less than 28 degrees celcius.

atemp

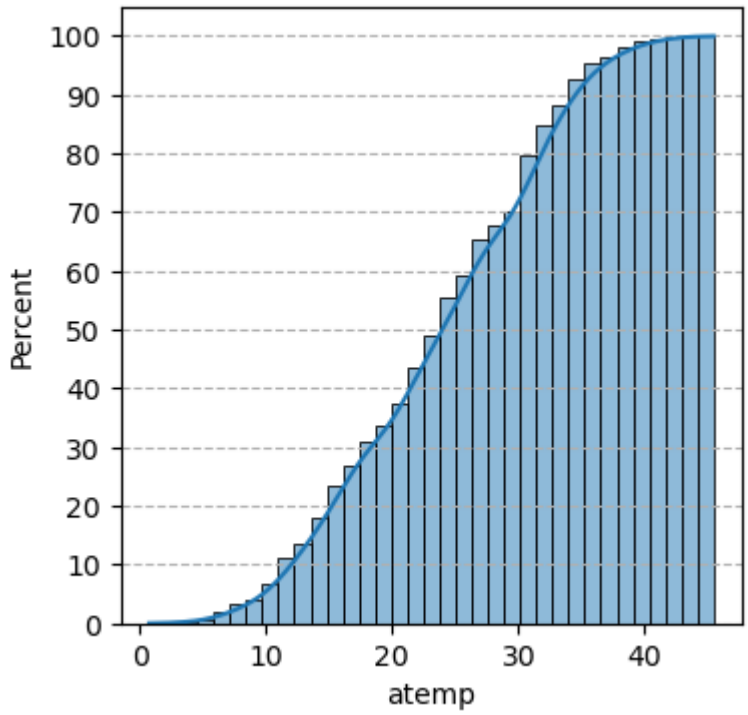
In [39]:
```
# The below code generates a histogram plot for the 'atemp' feature, showing the distribution
    # feeling temperature values in the dataset.
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'atemp', kde = True, bins = 50)
plt.plot()
```

Out[39]:
```
[]
```



In [40]:
```
## The below code generates a histogram plot for the 'temp' feature, showing the
## cumulative distribution of temperature values in the dataset.
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'atemp', kde = True, cumulative = True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()
```

Out[40]:
```
[]
```



In [41]:
```
temp_mean = np.round(df['atemp'].mean(), 2)
temp_std = np.round(df['atemp'].std(), 2)
temp_mean, temp_std
```

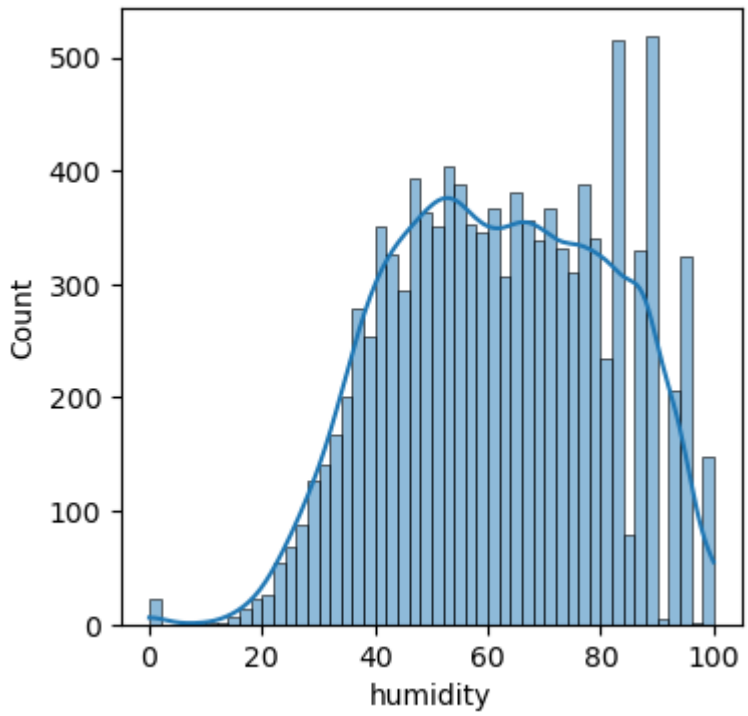
Out[41]:
```
(23.66, 8.47)
```

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

humidity

In [42]:
```
# The below code generates a histogram plot for the 'humidity' feature, showing the distribut
    # humidity values in the dataset.
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'humidity', kde = True, bins = 50)
plt.plot()
```

Out[42]:
```
[]
```



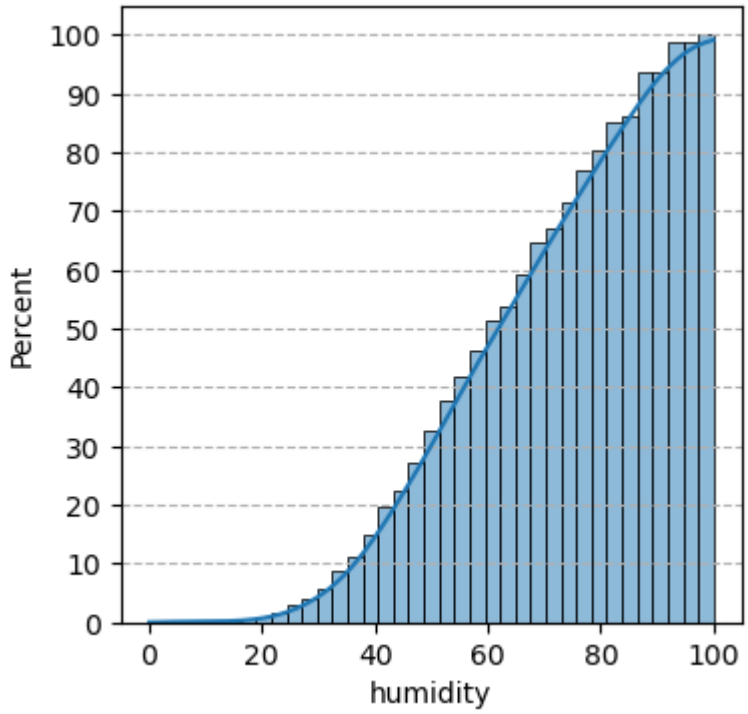
In [43]:
```
humidity_mean = np.round(df['humidity'].mean(), 2)
humidity_std = np.round(df['humidity'].std(), 2)
humidity_mean, humidity_std
```

Out[43]:
```
(61.89, 19.25)
```

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

In [44]:
```
# The below code generates a histogram plot for the 'humidity' feature, showing the cumulativ
    # distribution of humidity values in the dataset.
plt.figure(figsize = (4,4))
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()
```

Out[44]:
```
[]
```

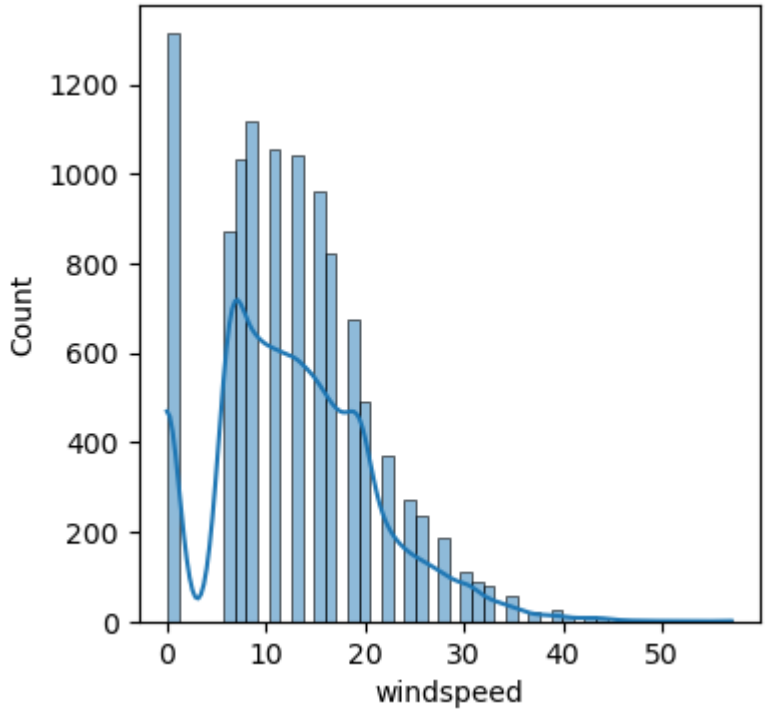


Insights: More than 80 % of the time, the humidity value is greater than 40. Thus for most of the time, humidity level varies from optimum to too moist.

windspeed

In [45]:
```
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'windspeed', kde = True, bins = 50)
plt.plot()
```

Out[45]:
```
[]
```



In [46]:
```
len(df[df['windspeed'] < 20])/len(df)
```

Out[46]:
```
0.8626676465184641
```

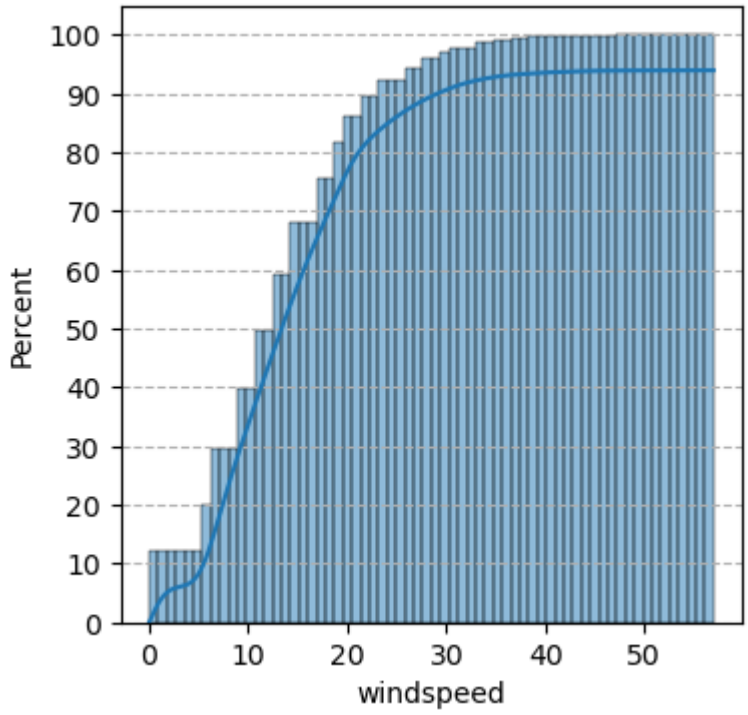
Insight: More than 85 % of the total windspeed data has a value of less than 20.

In [47]:
```
windspeed_mean = np.round(df['windspeed'].mean(),2)
windspeed_std = np.round(df['windspeed'].std(), 2)
windspeed_mean, windspeed_std
```

Out[47]:
```
(12.8, 8.16)
```

In [48]:
```
plt.figure(figsize = (4,4))
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()
```

Out[48]:
```
[]
```

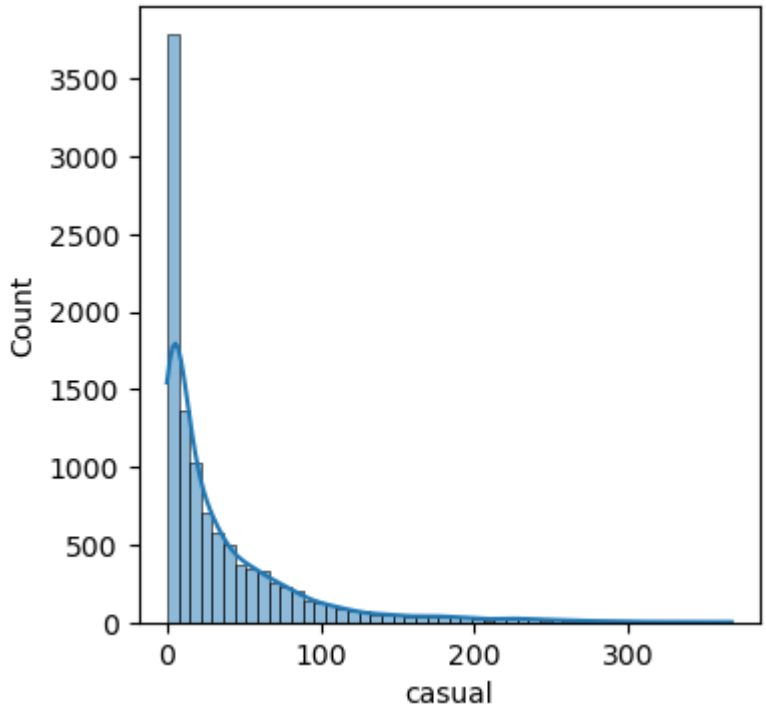


casual

In [49]:
```
# The below code generates a histogram plot for the 'casual' feature, showing the distributio
    # casual users' values in the dataset.
```

In [50]:
```
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'casual', kde = True, bins = 50)
plt.plot()
```

Out[50]:
```
[]
```

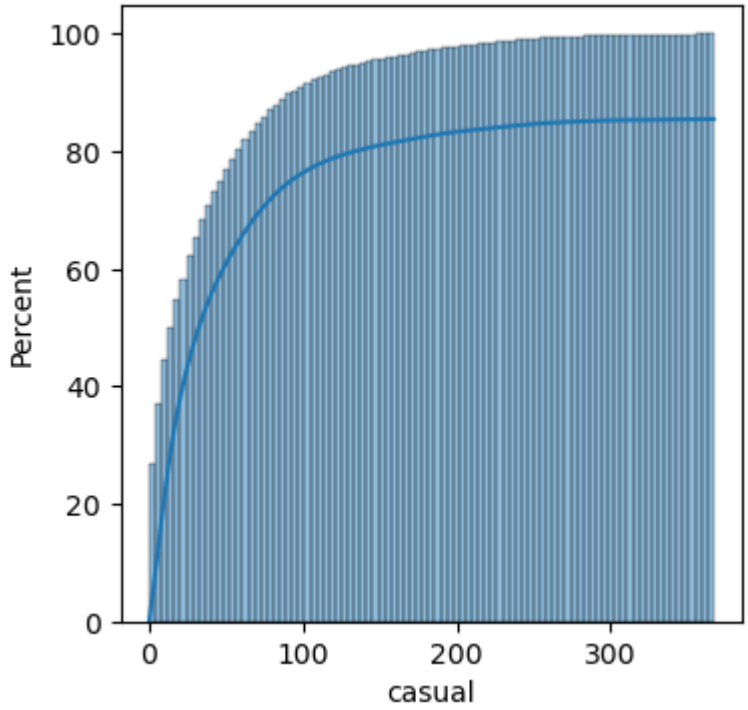


In [51]:
```
len(df[df['casual'] < 60])/len(df)
```

Out[51]:
```
0.800845122175271
```

In [52]:
```
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'casual', kde = True, cumulative = True, stat = 'percent')
plt.plot()
```

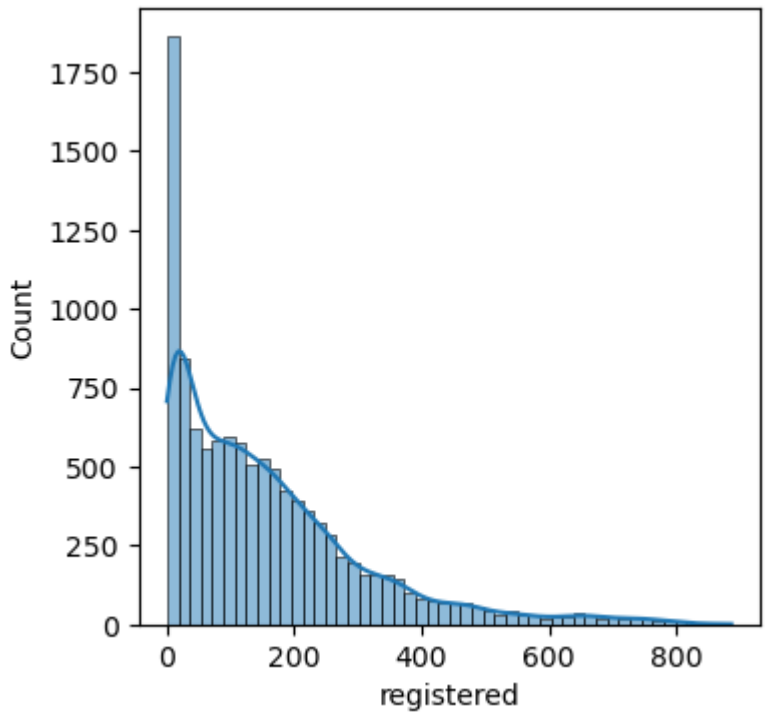
Out[52]:
```
[]
```



registered (user)

In [53]:
```
# The below code generates a histogram plot for the 'registered' feature, showing the distrib
    # registered users' values in the dataset.
plt.figure(figsize = (4,4))
sns.histplot(data = df, x = 'registered', kde = True, bins = 50)
plt.plot()
```

Out[53]:
```
[]
```



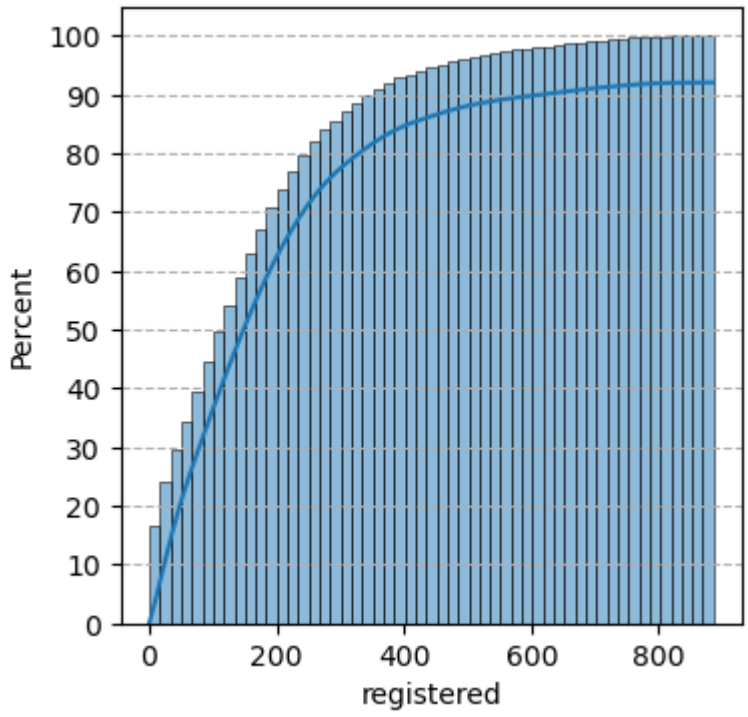
In [54]:
```
len(df[df['registered'] < 300])/len(df)
```

Out[54]:
```
0.8551350358258314
```

In [55]:
```
plt.figure(figsize = (4,4))
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()
```

Out[55]: []

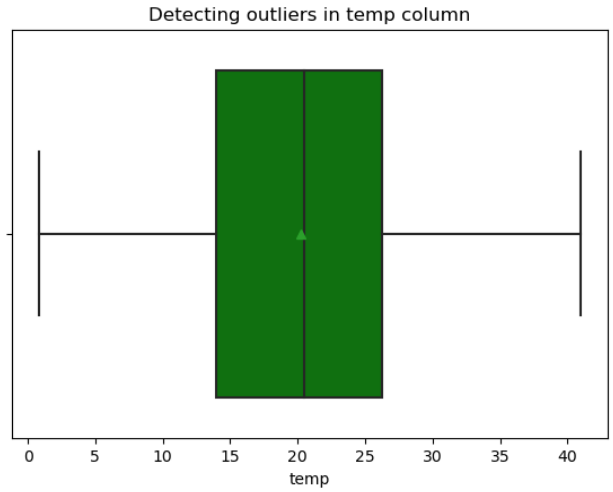


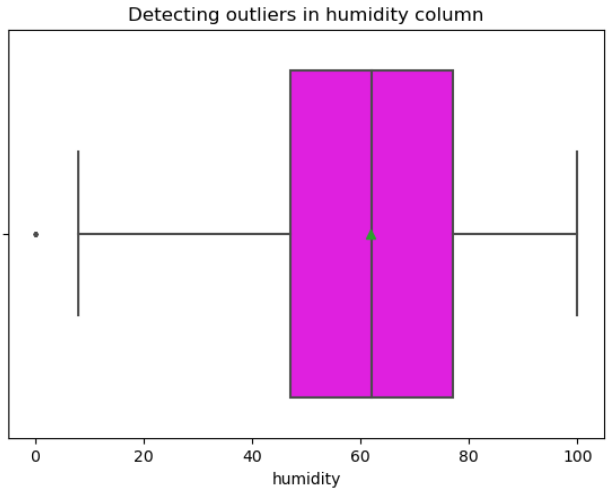
Insight: More than 85 % of the time, the count of registered users is less than 300.

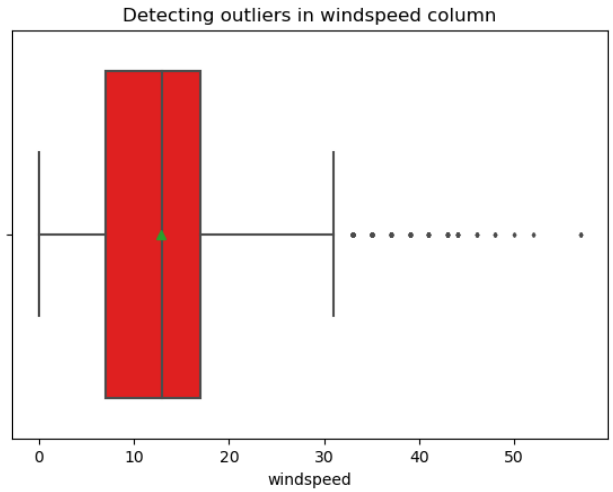
Outliers Detection

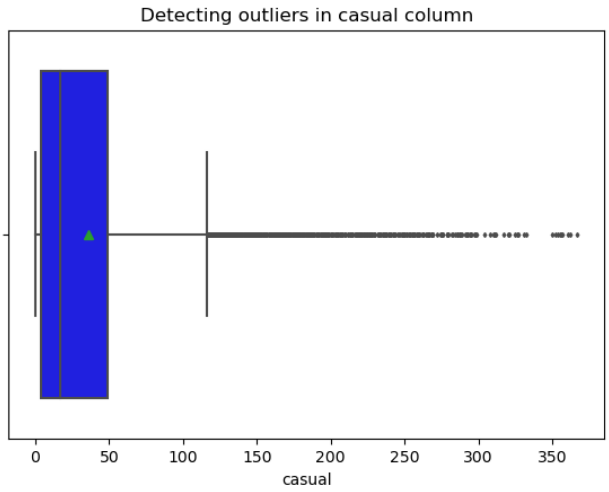
Univariate Analysis

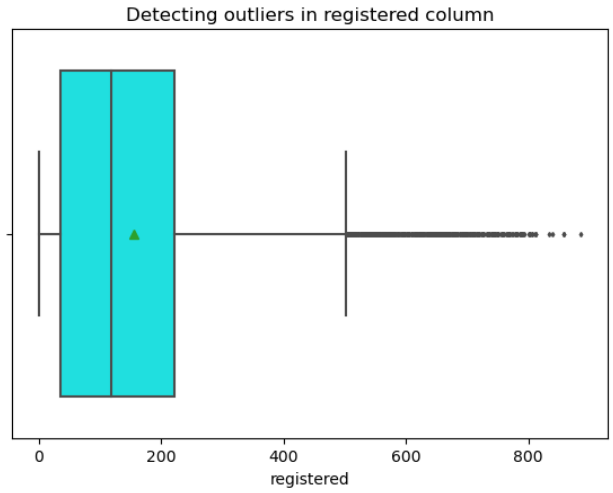
In [56]:
```
columns = ['temp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
colors = np.random.permutation(['red', 'blue', 'green', 'magenta', 'cyan', 'gray'])
count =1
plt.figure(figsize = (15,16))
for i in columns:
    plt.subplot(3,2, count)
    plt.title(f"Detecting outliers in {i} column")
    sns.boxplot(data = df, x = df[i], color = colors[count - 1], showmeans = True, fliersize
    plt.plot()
    count += 1
```

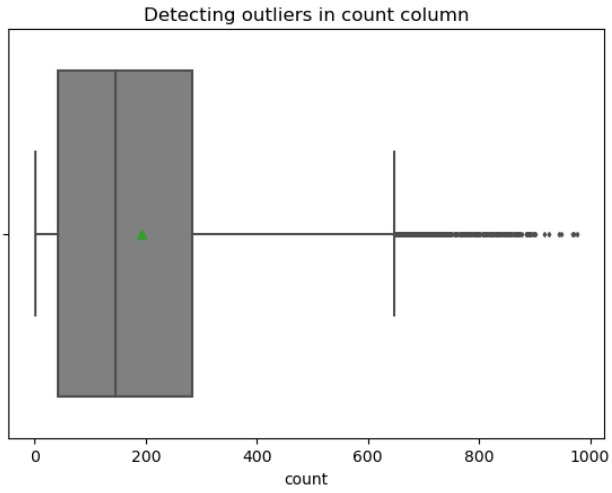












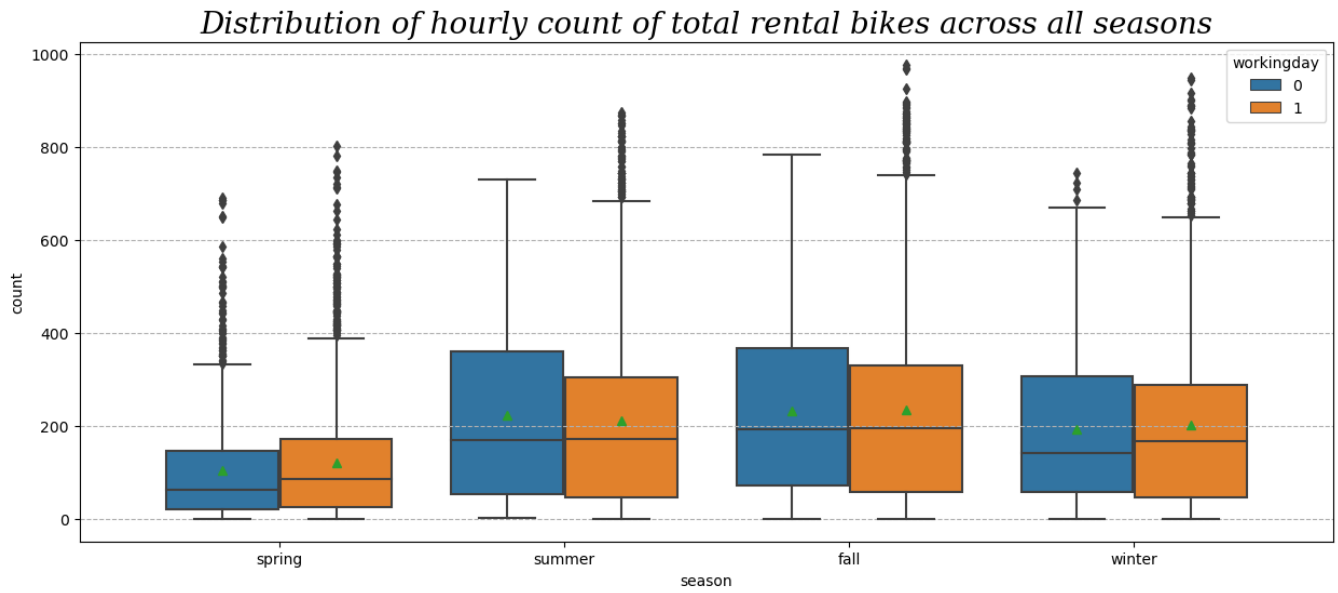
Insights:

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

In [57]:
```
# count vs season
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()
```

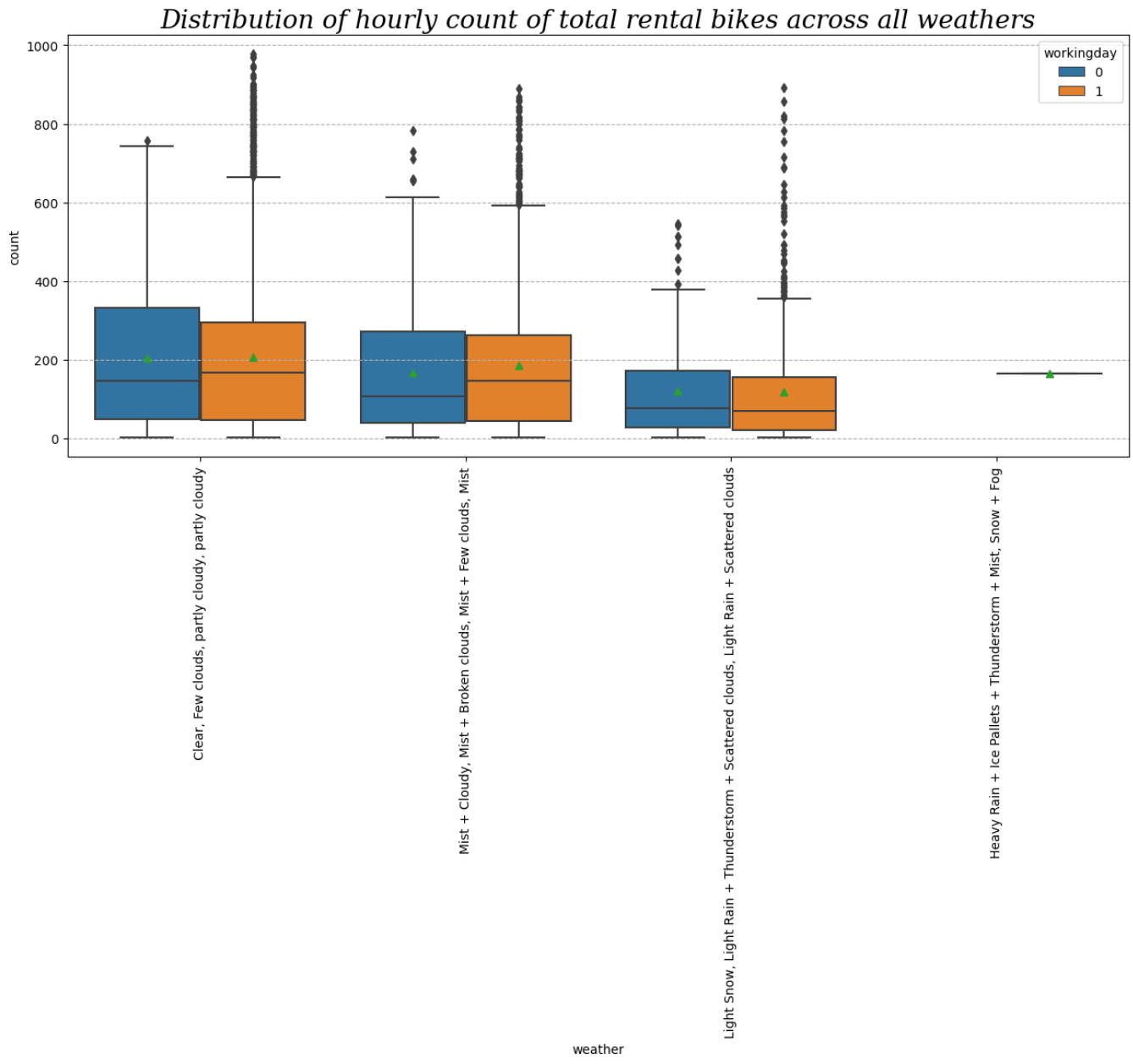
Out[57]: []



Insights:

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.

In [58]:
```
#count vs weather
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.xticks(rotation='vertical')
plt.show()
```



Insight:

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.

Q1. Is there any effect of Working Day on the number of electric cycles rented ?

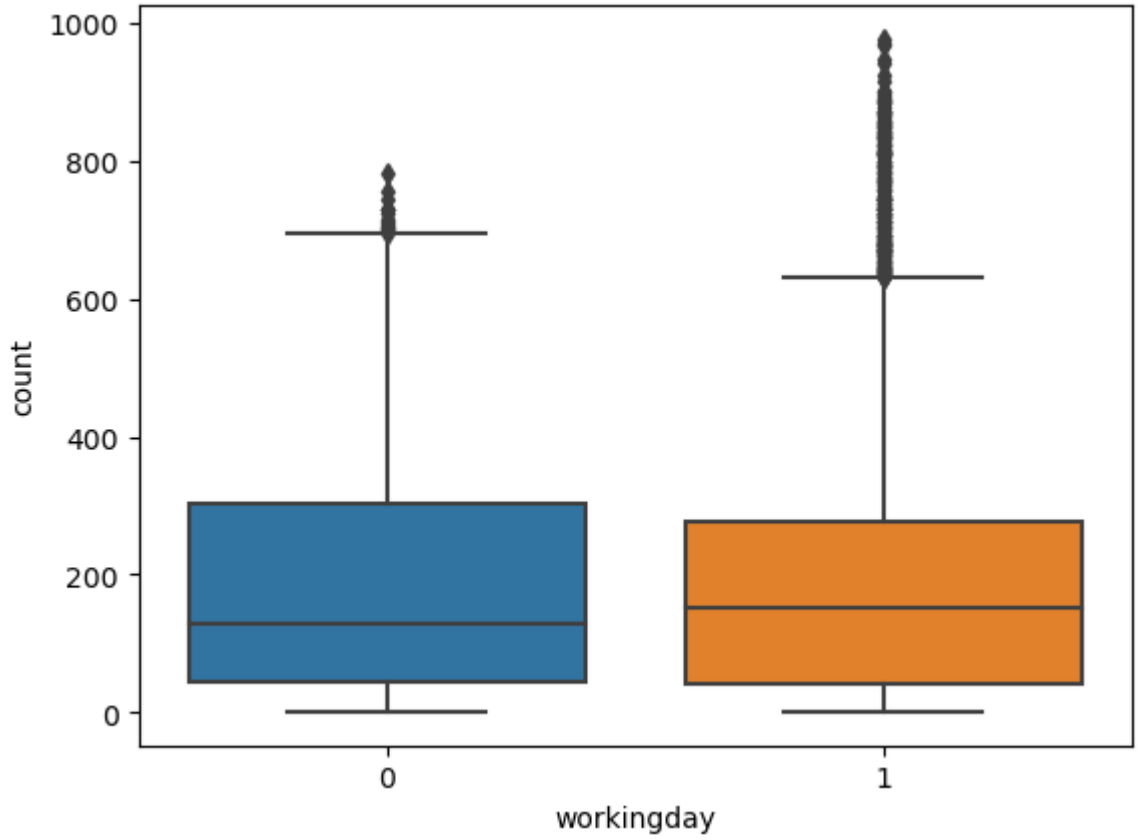
In [59]:
```
df.groupby(by = 'workingday')['count'].describe()
```

Out[59]:

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

In [60]:
```
sns.boxplot(data = df, x = 'workingday', y = 'count')
plt.plot()
```

Out[60]:
```
[]
```



Hypothesis Testing to answer the above-mentioned question

Step 1: Set Up Hypothesis

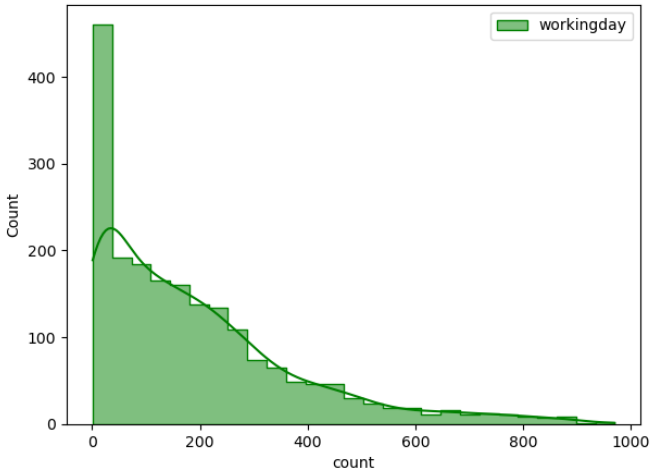
Null Hypothesis (H0) - Working Day does not have any effect on the number of electric cycles rented.

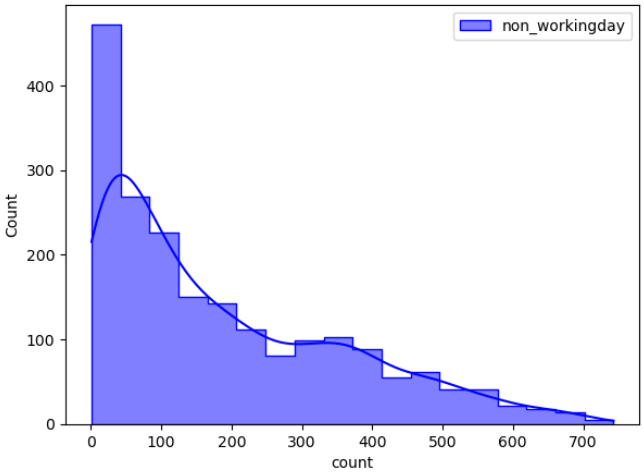
Alternate Hypothesis (HA) - Working Day has some effect on the number of electric cycles rented

Checking the distribution of the given data-set: Visual test (histplot) to check the distribution

In [61]:
```
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()
```

Out[61]:
```
[]
```





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

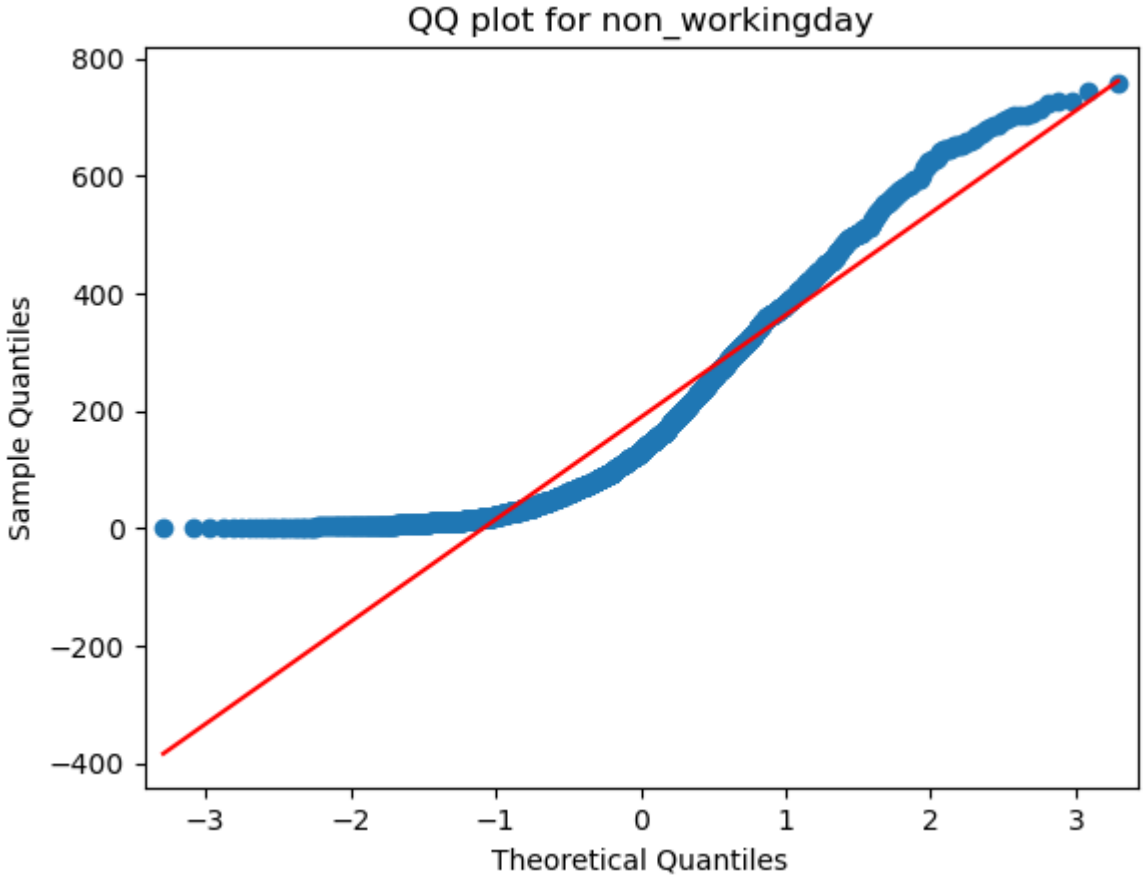
Distribution check using QQ Plot

In [62]:
```
import statsmodels.api as sm
plt.figure(figsize = (15, 6))
#plt.suptitle('QQ plots for the count of electric vehicles rented in workingday and non_worki
sample_data1 = df.loc[df['workingday'] == 1, 'count'].sample(2000)
sm.qqplot(sample_data1, line='s')
plt.title('QQ plot for workingday')
```

Out[62]:
```
Text(0.5, 1.0, 'QQ plot for workingday')

<Figure size 1500x600 with 0 Axes>
```

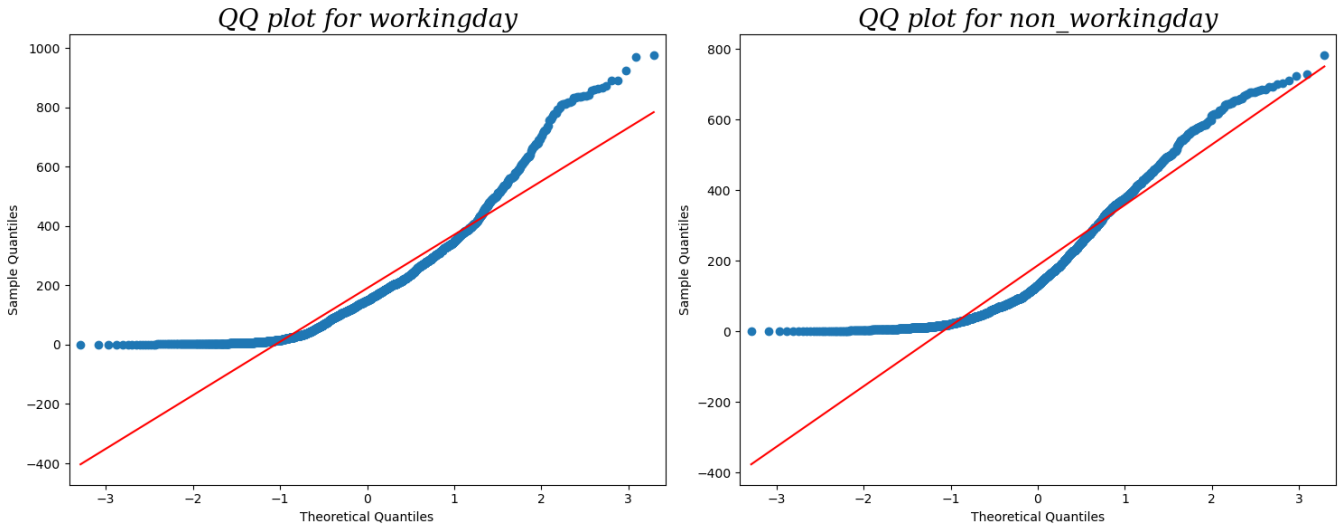
In [63]:
```
plt.figure(figsize = (15, 6))
sample_data2 = df.loc[df['workingday'] == 0, 'count'].sample(2000)
sm.qqplot(sample_data2, line='s')
plt.title('QQ plot for non_workingday')
plt.show()
```

```
<Figure size 1500x600 with 0 Axes>
```



In [64]:

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Sample 2000 data points from 'count' column
sample_data1 = df.loc[df['workingday'] == 1, 'count'].sample(2000)
sample_data2 = df.loc[df['workingday'] == 0, 'count'].sample(2000)

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Subplot 1: QQ plot for workingday
sm.qqplot(sample_data1, line='s', ax=axes[0])
axes[0].set_title('QQ plot for workingday',
         fontdict = {'size' : 20,
                     'style' : 'oblique',
                     'family' : 'serif'})

# Subplot 2: QQ plot for non_workingday
sm.qqplot(sample_data2, line='s', ax=axes[1])
axes[1].set_title('QQ plot for non_workingday',
                          fontdict = {'size' : 20,
                     'style' : 'oblique',
                     'family' : 'serif'})

# Adjust layout to prevent overlapping
plt.tight_layout()

plt.show()
```



Insight: 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.

To cross-check normality, we perform the Shapiro-Wilk test

Null Hypothesis (H0) - The sample follows normal distribution

Alternate Hypothesis (HA) - The sample does not follow normal distribution

In [65]:
```
alpha = 0.05
test_stat, p_value = 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 8.353528585774545e-38
The sample does not follow normal distribution
```

In [66]:
```
test_stat, p_value = 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 4.2111700411517905e-12
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test

Null Hypothesis (H0) - Homogenous Variance

Alternate Hypothesis (HA) -Non Homogenous Variance

In [67]:
```
test_stat, p_value = 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 0.618085890325626
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.

Null Hypothesis (Ho) : Mean no.of electric cycles rented is same for working and non-working days

Alternate Hypothesis (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

In [68]:
```
test_stat, p_value = 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 different 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
```

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

Q2. Is there any effect of holidays on the number of electric cycles rented ?

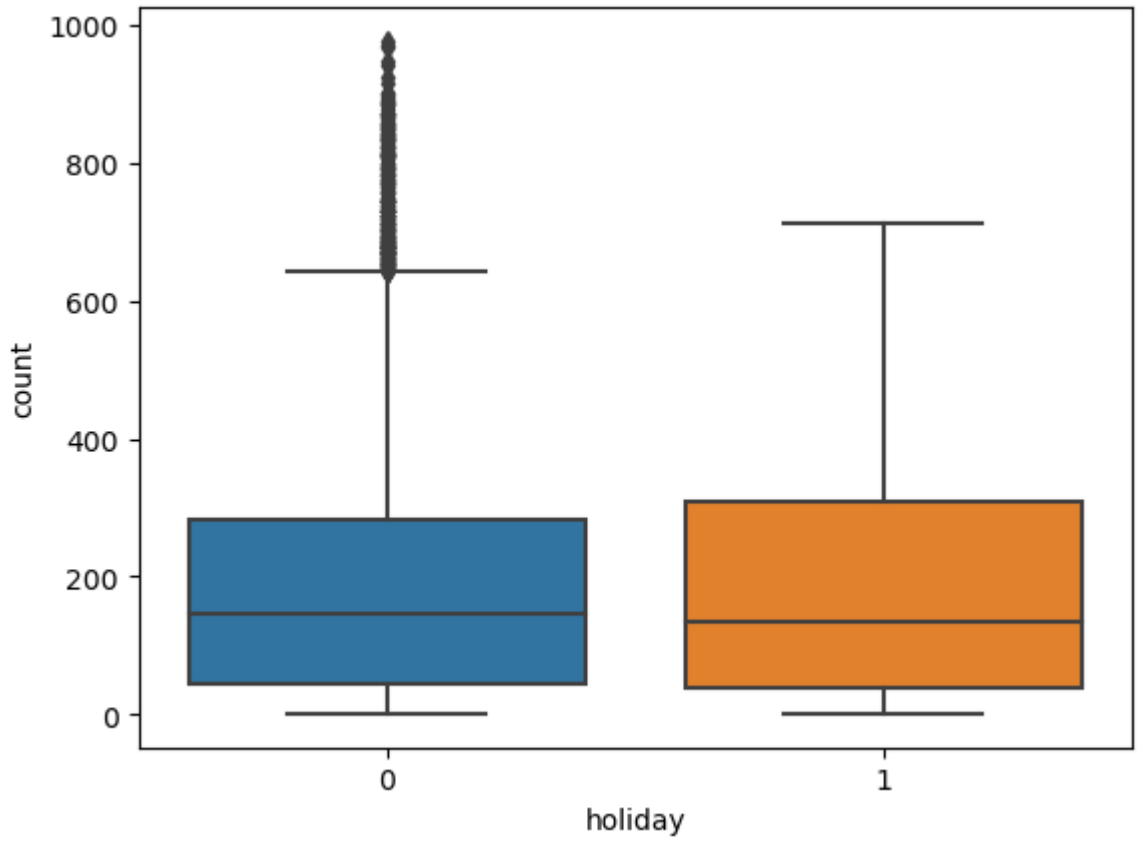
In [69]:
```
df.groupby(by = 'holiday')['count'].describe()
```

Out[69]:

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

In [70]:
```
sns.boxplot(data = df, x = 'holiday', y = 'count')
plt.plot()
```

Out[70]:
```
[]
```

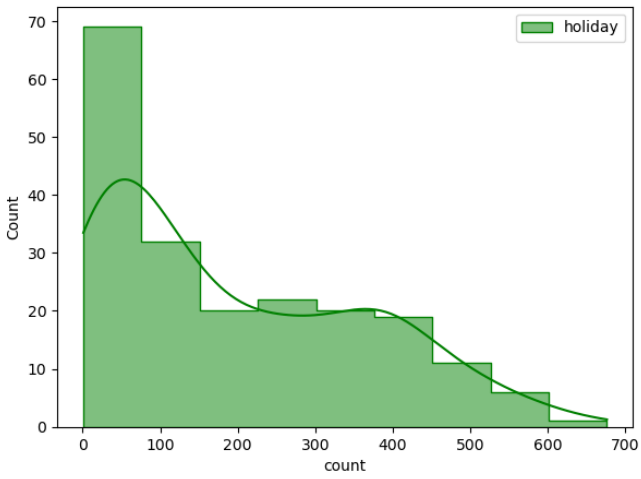


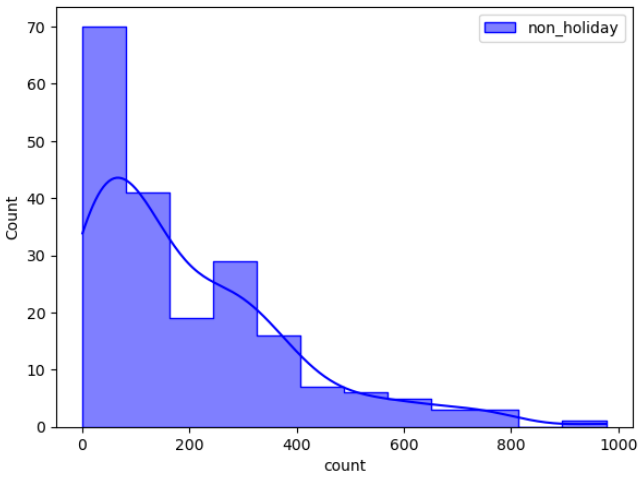
Hypothesis Test

Visual Tests (histplot) to check if the samples follow normal distribution

In [71]:
```
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()
```

Out[71]:
```
[]
```

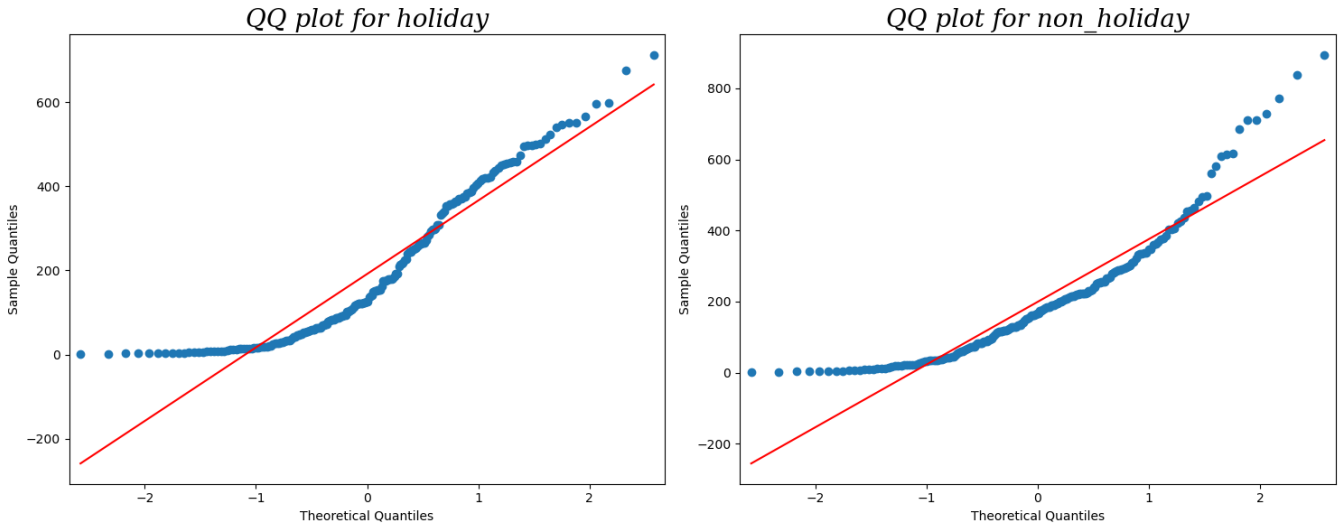




Insights: From the graph, it can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

In [72]:
```
# Sample 2000 data points from 'count' column
sample_data1 = df.loc[df['holiday'] == 1, 'count'].sample(200)
sample_data2 = df.loc[df['holiday'] == 0, 'count'].sample(200)

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Subplot 1: QQ plot for workingday
sm.qqplot(sample_data1, line='s', ax=axes[0])
axes[0].set_title('QQ plot for holiday',
         fontdict = {'size' : 20,
                     'style' : 'oblique',
                     'family' : 'serif'})

# Subplot 2: QQ plot for non_workingday
sm.qqplot(sample_data2, line='s', ax=axes[1])
axes[1].set_title('QQ plot for non_holiday',
                          fontdict = {'size' : 20,
                     'style' : 'oblique',
                     'family' : 'serif'})

# Adjust layout to prevent overlapping
plt.tight_layout()

plt.show()
```



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

To cross-check normality, we perform the Shapiro-Wilk test

Null Hypothesis (H0) : The sample follows normal distribution

Alternate Hypothesis (Ha) : The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [73]:
```
test_stat, p_value = 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 4.505549267008746e-10
The sample does not follow normal distribution
```

In [74]:
```
test_stat, p_value = 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 7.917214106534232e-11
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [75]:
```
test_stat, p_value = 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 0.9096748850102185
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.

Null Hypothesis (Ho) : No.of electric cycles rented is similar for holidays and non-holidays

Alternate HypothesisHa : No.of electric cycles rented is different for holidays and non-holidays days

In [76]:
```
# Assuming significance Level to be 0.05

test_stat, p_value = 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 different for holidays and non-holidays days')
else:
    print('No.of electric cycles rented is similar for holidays and non-holidays')
```

```
P-value : 0.5300273065542247
No.of electric cycles rented is similar for holidays and non-holidays
```

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

In [ ]:
```

```

Q3.Is weather dependent on the season ?

In [77]:
```
df[['weather', 'season']].describe()
```

Out[77]:

	weather	season
count	10886	10886
unique	4	4
top	Clear, Few clouds, partly cloudy, partly cloudy	winter
freq	7192	2734

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

Null Hypothesis (H0) - weather is independent of season

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

Since we have two categorical features, the Chi- square test is applicable here.

Note: 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.

In [78]:
```
cross_table = pd.crosstab(index = df['season'], columns =  df['weather'], values = df['count'
                          aggfunc = np.sum).replace(np.nan, 0)
cross_table
```

Out[78]:

weather	Clear, Few clouds, partly cloudy, partly cloudy	Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds	Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
season				
fall	470116.0	0.0	31160.0	139386.0
spring	223009.0	164.0	12919.0	76406.0
summer	426350.0	0.0	27755.0	134177.0
winter	356588.0	0.0	30255.0	157191.0

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.

In [79]:
```
cross_table = pd.crosstab(index=df['season'],
                          columns=df['weather'],
                          values=df['count'],
                          aggfunc=np.sum).drop(columns='Heavy Rain + Ice Pallets + Thundersto
                                               errors='ignore').replace(np.nan, 0)

cross_table
```

Out[79]:

weather	Clear, Few clouds, partly cloudy, partly cloudy	Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds	Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
season			
fall	470116.0	31160.0	139386.0
spring	223009.0	12919.0	76406.0
summer	426350.0	27755.0	134177.0
winter	356588.0	30255.0	157191.0

In [80]:
```
chi_test_stat, p_value, dof, expected = chi2_contingency(observed = cross_table)
print('Test Statistic =', chi_test_stat)
print('p value =', p_value)
print('-' * 65)
print("Expected : '\n'", expected)
```

```
Test Statistic = 10838.372332480216
p value = 0.0
-----------------------------------------------------------------
Expected : '
' [[453484.88557396  31364.39195574 155812.72247031]
 [221081.86259035  15290.69305984  75961.44434981]
 [416408.3330293   28800.06497733 143073.60199337]
 [385087.91880639  26633.8500071  132312.23118651]]
```

In [81]:
```
if p_value < alpha:
    print('Reject Null Hypothesis: There is statistically significant dependency of weather a
else:
    print('Failed to reject Null Hypothesis: There is no dependency of weather and season')
```

```
Reject Null Hypothesis: There is statistically significant dependency of weather and season
```

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

Q4. Is the number of cycles rented is similar or different in different weather ?

In [82]:
```
df.groupby(by = 'weather')['count'].describe()
```

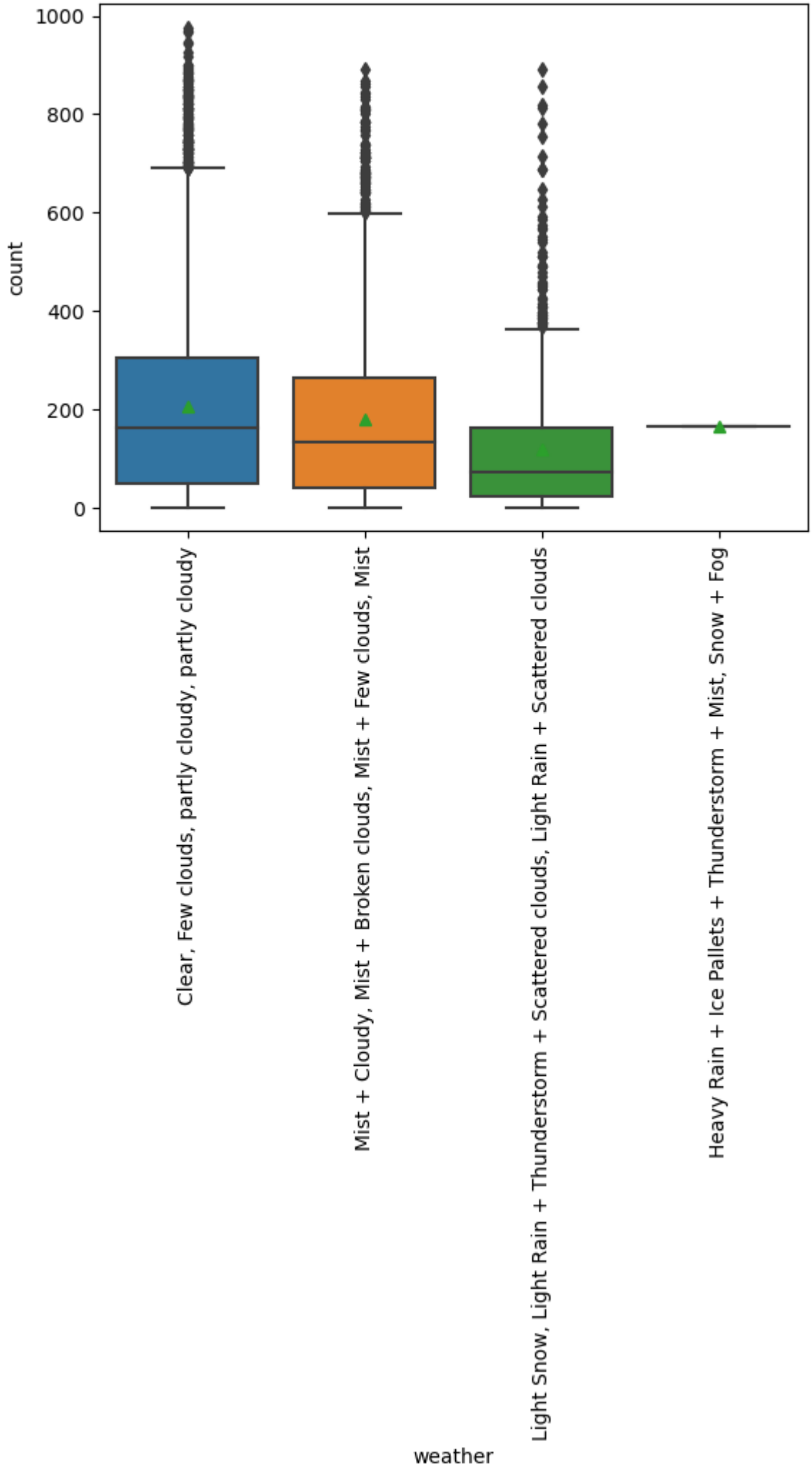
Out[82]:

weather	count	mean	std	min	25%	50%	75%	max
Clear, Few clouds, partly cloudy, partly cloudy	7192.0	205.236791	187.959566	1.0	48.0	161.0	305.0	977.0
Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog	1.0	164.000000	NaN	164.0	164.0	164.0	164.0	164.0
Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds	859.0	118.846333	138.581297	1.0	23.0	71.0	161.0	891.0
Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist	2834.0	178.955540	168.366413	1.0	41.0	134.0	264.0	890.0

In [83]:
```
sns.boxplot(data = df, x = 'weather', y = 'count', showmeans = True)
plt.xticks(rotation='vertical')
plt.plot()
```

Out[83]:

```
[]
```



In [84]:
```
df_weather1 = df.loc[df['weather'] == "Clear, Few clouds, partly cloudy, partly cloudy"]
df_weather2 = df.loc[df['weather'] =="Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds,
df_weather3 = df.loc[df['weather'] == "Light Snow, Light Rain + Thunderstorm + Scattered clou
```

```
df_weather4 = df.loc[df['weather'] == "Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow +
len(df_weather1), len(df_weather2), len(df_weather3), len(df_weather4)

# 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
```

Out[84]: (7192, 2834, 859, 1)

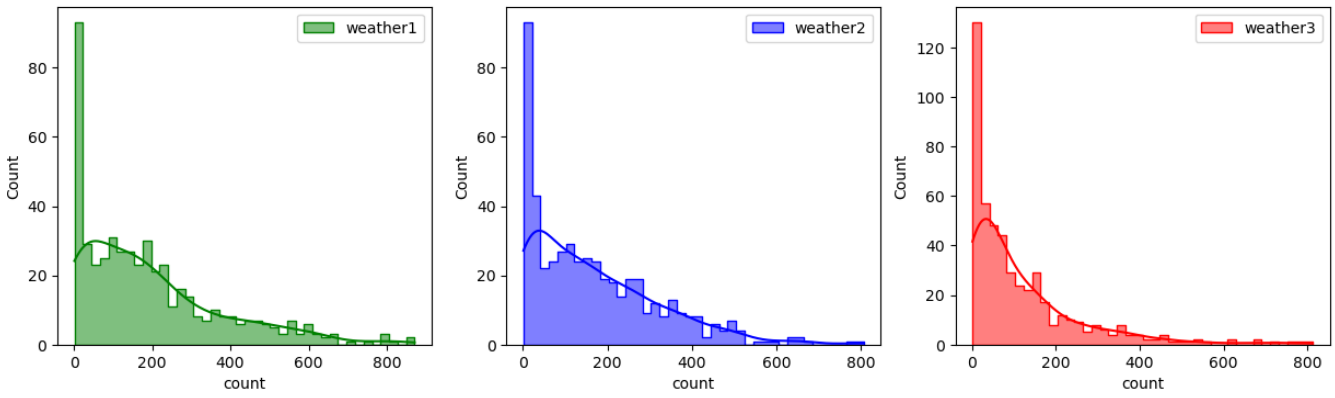
Hypotheis Test

Null Hypothesis (H0) - Mean of cycle rented per hour is same for weather 1, 2 and 3

Alternate Hypothesis (HA) -Mean of cycle rented per hour is not same for season 1,2,3 and 4 are different.

In [85]:
```
#### Visual Tests (histplot) 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()
```

Out[85]: []



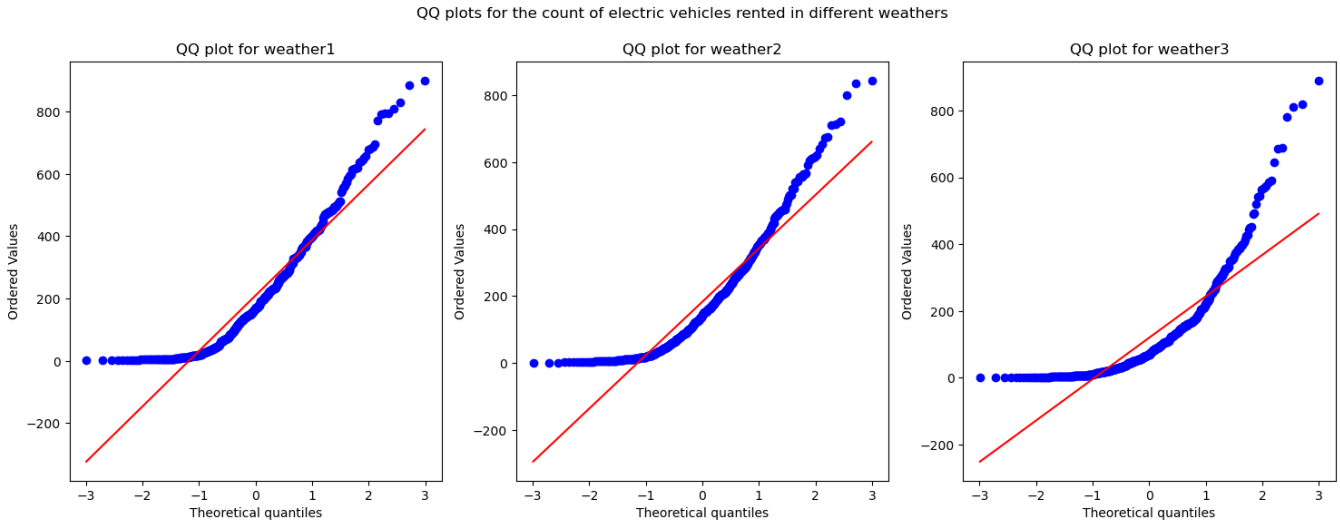
Insight: It can be inferred from the above plot that the distributions do not follow normal distribution

Distribution check using QQ Plot

In [86]:
```
plt.figure(figsize = (18, 6))
plt.subplot(1, 3, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in different weathers')
probplot(df_weather1.loc[:, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('QQ plot for weather1')
plt.subplot(1, 3, 2)
probplot(df_weather2.loc[:, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('QQ plot for weather2')
plt.subplot(1, 3, 3)
probplot(df_weather3.loc[:, 'count'].sample(500), plot = plt, dist = 'norm')
```

```
plt.title('QQ plot for weather3')
plt.plot()
```

Out[86]: []



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

To cross-check normality, we perform the Shapiro-Wilk test

Null Hypothesis (H0) : The sample follows normal distribution

Alternate Hypotheis (Ha) : The sample does not follow normal distribution

alpha = 0.05

In [87]:
```
test_stat, p_value = 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 1.2281251367211957e-19
The sample does not follow normal distribution
```

In [88]:
```
test_stat, p_value = 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 1.672285679680454e-20
The sample does not follow normal distribution
```

In [89]:
```
test_stat, p_value = 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 5.43441778634405e-26
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test*

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [90]:
```
test_stat, p_value = 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 1.0447596815088832e-12
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

In [91]:
```
alpha = 0.05

test_stat1, p_value1 = kruskal(df_weather1['count'], df_weather2['count'], df_weather3['count
print('Test Statistic for Weather =', test_stat1)
print('p value for Weather =', p_value1)

if p_value1 < alpha:
    print('Reject Null Hypothesis: The average number of rental bikes is statistically differ
else:
    print('Failed to reject Null Hypothesis: The average number of rental bikes is similar fo
```

```
Test Statistic for Weather = 204.95566833068537
p value for Weather = 3.122066178659941e-45
Reject Null Hypothesis: The average number of rental bikes is statistically different for dif
ferent weathers
```

Conclusion: Hence, the average number of rental bikes is statistically different for different weathers

Q5. Is the number of cycles rented is similar or different in different season ?

In [92]:
```
df.groupby(by = 'season')['count'].describe()
```

Out[92]:

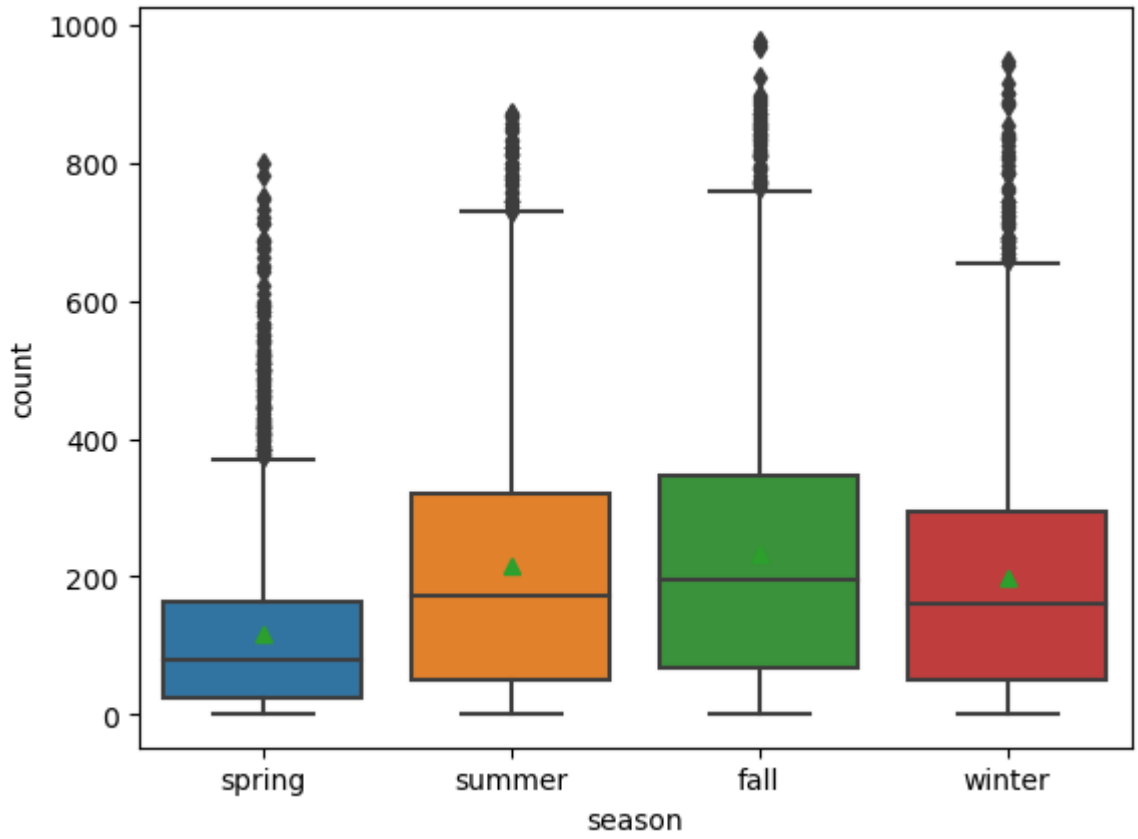
season	count	mean	std	min	25%	50%	75%	max
fall	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0
spring	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0
summer	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0
winter	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0

In [93]:
```
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)
```

Out[93]:
```
(2686, 2733, 2733, 2734)
```

In [94]:
```
sns.boxplot(data = df, x = 'season', y = 'count', showmeans = True)
plt.plot()
```

Out[94]:
```
[]
```



Hypothesis test

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

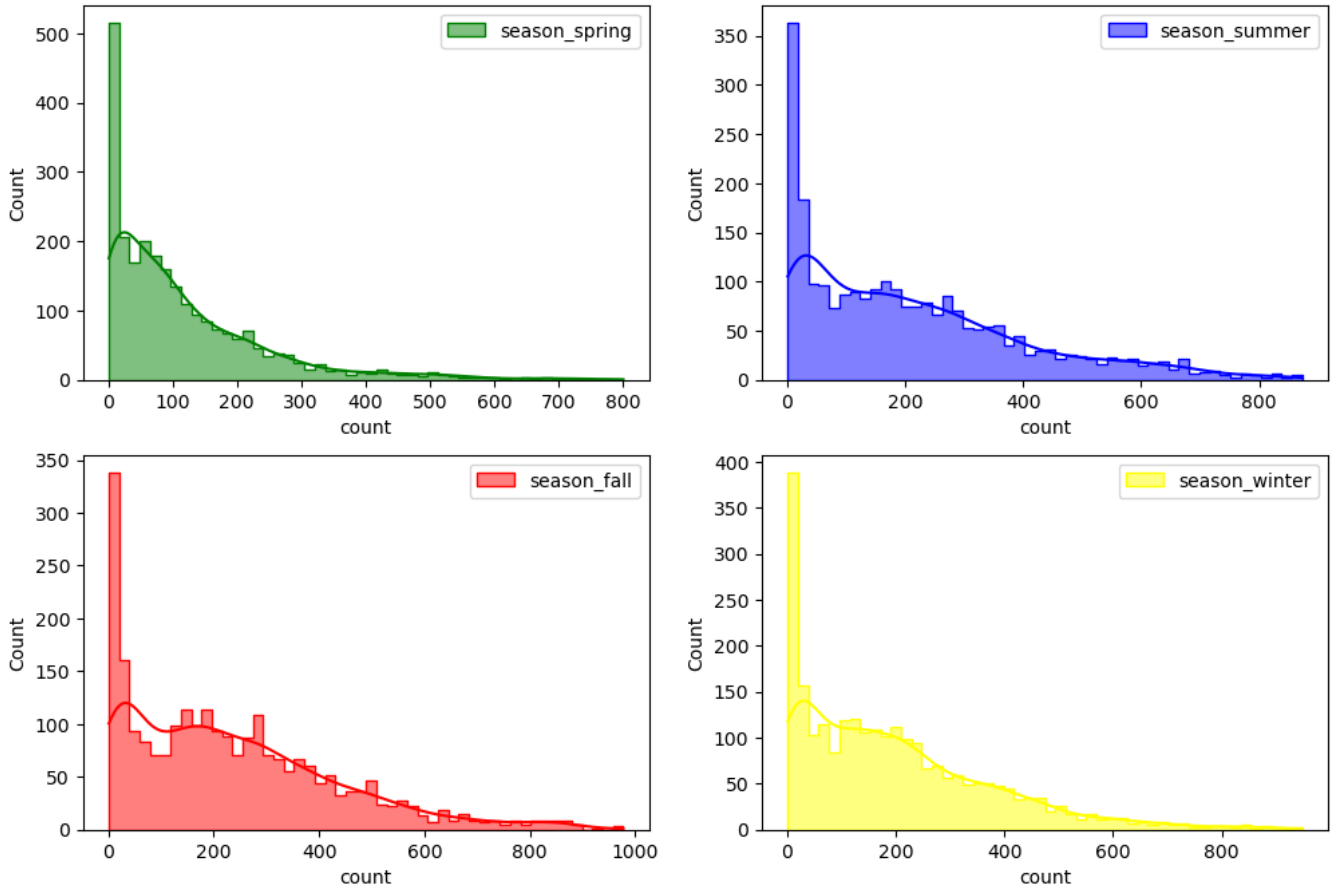
Alternate Hypothesis (HA) -Mean of cycle rented per hour is different for season 1,2,3 and 4.

Visual Tests (histplot) to know if the samples follow normal distribution

In [95]:
```
plt.figure(figsize = (12, 8))
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()
```

Out[95]:
```
[]
```

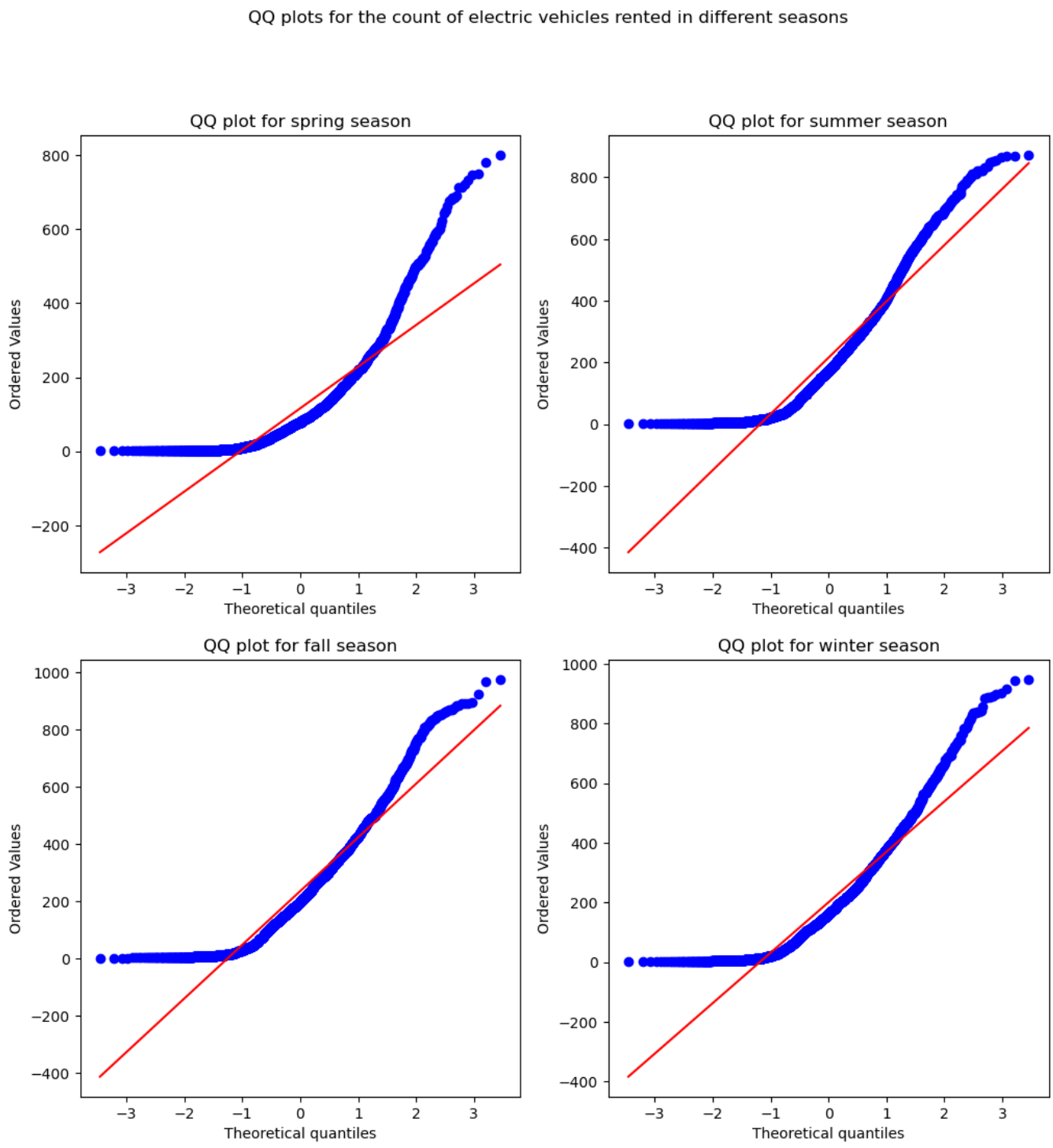


Insight: It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

In [96]:
```
plt.figure(figsize = (12, 12))
plt.subplot(2, 2, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in different seasons')
probplot(df_season_spring.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for spring season')

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

plt.subplot(2, 2, 3)
probplot(df_season_fall.sample(2500), plot = plt, dist = 'norm')
plt.title('QQ plot for fall season')

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

Out[96]:
```
[]
```



Insight: It can be inferred from the above plots that the distributions do not follow normal distribution

To cross-check normality, we perform the Shapiro-Wilk test

Null Hypothesis (H0) - The sample follows normal distribution

Alternate Hypothesis (HA) - The sample does not follow normal distribution

In [97]:
```
test_stat, p_value = 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
```

In [98]:
```
test_stat, p_value = 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 1.5097764536683927e-37
The sample does not follow normal distribution
```

In [99]:
```
test_stat, p_value = 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 1.248898893380258e-35
The sample does not follow normal distribution
```

In [100…
```
test_stat, p_value = 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 2.8262934462007525e-38
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test

Null Hypothesis(H0) - Homogenous Variance

Alternate Hypothesis(HA) - Non Homogenous Variance

In [101…
```
test_stat, p_value = 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 1.1290491785853757e-110
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

In [102…
```
alpha = 0.05
test_stat, p_value = kruskal(df_season_spring, df_season_summer, df_season_fall,df_season_win
print('Test Statistic =', test_stat)
print('p value =', p_value)
```

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

In [103…
```
if p_value < alpha:
    print('Reject Null Hypothesis: The average number of rental bikes is statistically differ
else:
    print('Failed to reject Null Hypothesis: The average number of rental bikes is same for d
```

```
Reject Null Hypothesis: The average number of rental bikes is statistically different for dif
ferent seasons
```

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

In [ ]:
```
```

Correlations

In [104…
```
sns.pairplot(data = df,
             kind = 'reg',
             hue = 'workingday',
             markers = '.')
plt.plot()
```

Out[104]:
```
[]
```

In [105…

```
corr_data = df.corr()
corr_data
```

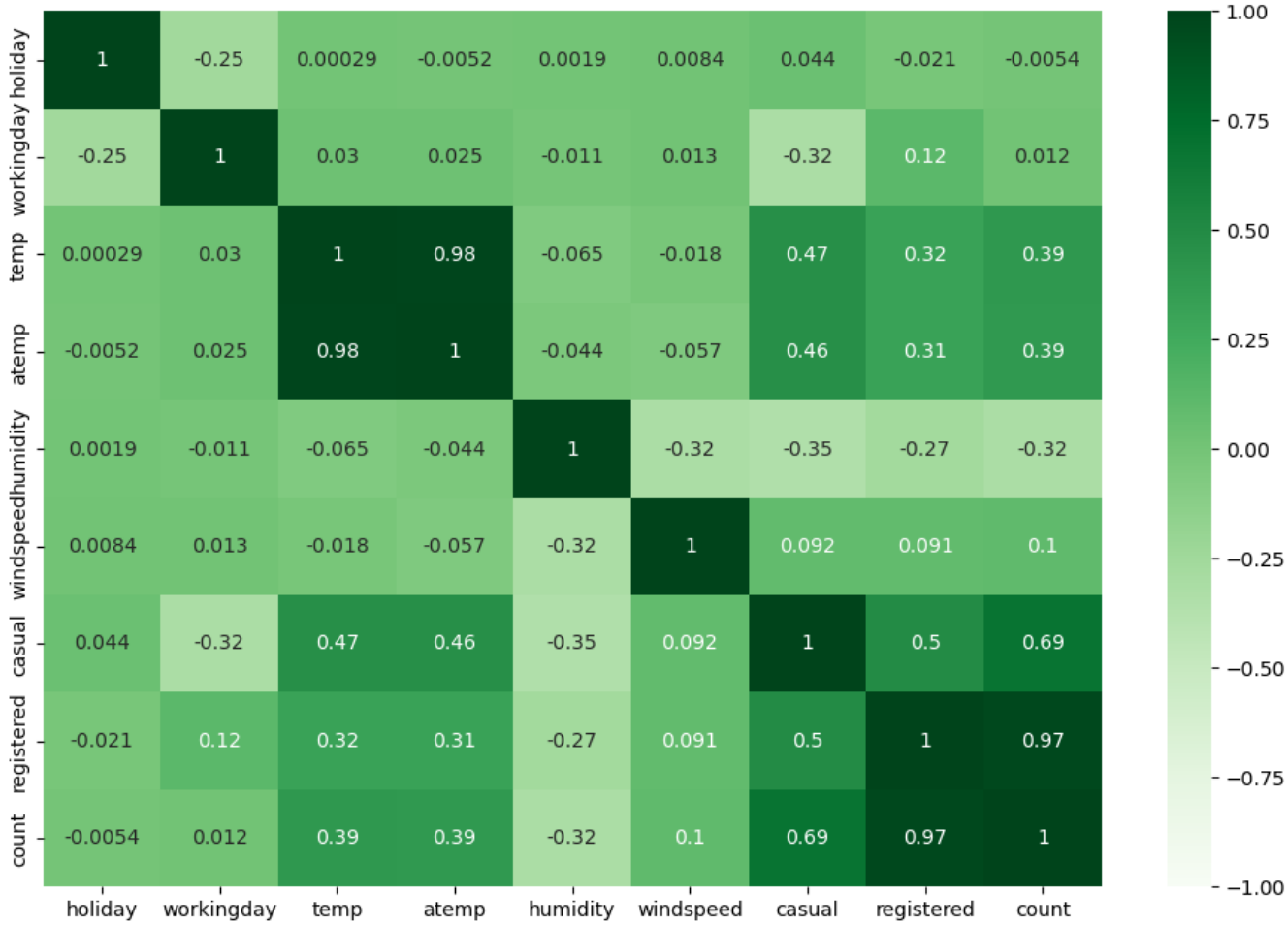
```
C:\Users\hp\AppData\Local\Temp\ipykernel_29604\919268980.py:1: FutureWarning: The default val
ue of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to F
alse. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr_data = df.corr()
```

Out[105]:

	holiday	workingday	temp	atemp	humidity	windspeed	casual	registered	cou
holiday	1.000000	-0.250491	0.000295	-0.005215	0.001929	0.008409	0.043799	-0.020956	-0.00539
workingday	-0.250491	1.000000	0.029966	0.024660	-0.010880	0.013373	-0.319111	0.119460	0.01159
temp	0.000295	0.029966	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.39445
atemp	-0.005215	0.024660	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.38978
humidity	0.001929	-0.010880	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.31737
windspeed	0.008409	0.013373	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.10136
casual	0.043799	-0.319111	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.69041
registered	-0.020956	0.119460	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.97094
count	-0.005393	0.011594	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.00000

In [106…]
```
plt.figure(figsize = (12, 8))
sns.heatmap(data = corr_data, cmap = 'Greens', annot = True, vmin = -1, vmax = 1)
plt.plot()
```

Out[106]:
```
[]
```



Insights:

```
* Strong Temperature Correlations: Very High Correlation (> 0.9) observed
between columns [atemp, temp].
* Count and Registered Users: Very High Correlation (> 0.9) found between
columns [count, registered].
* Correlation Insights:
    - No high positive/negative correlation (0.7 - 0.9) exists between any
columns.
    - Moderate positive correlation (0.5 - 0.7) observed between columns
[casual, count], [casual, registered].
* Temperature and Count Relationships: Low Positive correlation (0.3 - 0.5)
identified between columns [count, temp],      [count, atemp], [casual,
atemp].
* General Correlation Overview: Negligible correlation observed between all
other combinations of columns.
```

Overall Insights:

1. In summer and fall seasons more bikes are rented as compared to other seasons.
2. Whenever its a holiday more bikes are rented.
3. It is also clear from the workingday also that whenever day is holiday or weekend, slightly more bikes were rented.
4. Whenever there is rain, thunderstorm, snow or fog, there were less bikes were rented.

5. Whenever the humidity is less than 20, number of bikes rented is very very low.
6. Whenever the temperature is less than 10, number of bikes rented is less.
7. Whenever the windspeed is greater than 35, number of bikes rented is less.

Additional Insights:

1. Out of every 100 users, around 19 are casual users and 81 are registered users.
2. 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.
3. 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.
4. The average hourly count of rental bikes is the lowest in the month of January followed by February and March.
5. 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.
6. More than 80 % of the time, the temperature is less than 28 degrees celcius.
7. 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.
8. More than 85 % of the total, windspeed data has a value of less than 20.
9. 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.
10. The mean hourly count of the total rental bikes is statistically similar for both working and non-working days.
11. There is statistically significant dependency of weather and season based on the hourly total number of bikes rented.
12. The hourly total number of rental bikes is statistically different for different weathers.
13. There is no statistically significant dependency of weather 1, 2, 3 on season based on the average hourly total number of bikes rented.
14. The hourly total number of rental bikes is statistically different for different seasons.

Recommendations:

Seasonal Marketing:

Since there is a clear seasonal pattern in the count of rental bikes, X company 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.

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. The company can offer weather-specific discounts to attract more customers during these favorable weather conditions.

User Segmentation:

Given that around 81% of users are registered, and the remaining 19% are casual, it 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.

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, company 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.

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.

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 the company 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.

In [ ]: