In [1]:

```
from IPython import display
display.Image('yulu.jpg')
```

Out[1]:

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 that what variables, for instance, season, weather, holidays, working days are significant in predicting the demand for shared electric cycles in the Indian market and how well those variables describe the electric cycle demands? Are these variables associated with each other to impact electric cycles usages?

Dataset:

Importing Libraries and Dataset

In [2]:

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
from statsmodels.graphics.gofplots import qqplot
import scipy.stats as spy
warnings.filterwarnings("ignore")
```

- **Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.**

In [3]:

```
df=pd.read_csv("yulu.txt")
df.head(10)
```

Out[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casu
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	
5	2011-01-01 05:00:00	1	0	0	2	9.84	12.880	75	6.0032	
6	2011-01-01 06:00:00	1	0	0	1	9.02	13.635	80	0.0000	
7	2011-01-01 07:00:00	1	0	0	1	8.20	12.880	86	0.0000	
8	2011-01-01 08:00:00	1	0	0	1	9.84	14.395	75	0.0000	
9	2011-01-01 09:00:00	1	0	0	1	13.12	17.425	76	0.0000	

Basic Information about the Dataset

Shape of the dataset

In [4]:

```
df.shape
```

Out[4]:

```
(10886, 12)
```

Columns in the Dataset

In [5]:

```
df.keys()
```

Out[5]:

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

Column Profiling:

- **datetime**: datetime
- **season**: season (1: spring, 2: summer, 3: fall, 4: winter)
- **holiday**: whether day is a holiday or not.
- **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

Null value check:

In [6]:

```
df.isna().sum().sum()
```

Out[6]:

```
0
```

Duplicate values

In [7]:

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

Out[7]:

```
False
```

Colums datatype and null values check

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

Datatype conversion

In [9]:

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

In [10]:

```
df["season"]=df["season"].apply(lambda x:"spring"  if x==1  else "summer" if x==2 else "f
df["holiday"]=df["holiday"].apply(lambda x:"holiday"  if x==1  else "non_holiday")
df["workingday"]=df["workingday"].apply(lambda x:"working"  if x==1  else "holiday_weeker
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
weather	10886.0	1.418427	0.633839	1.00	1.0000	1.000	2.0000	4.0000
temp	10886.0	20.230860	7.791590	0.82	13.9400	20.500	26.2400	41.0000
atemp	10886.0	23.655084	8.474601	0.76	16.6650	24.240	31.0600	45.4550
humidity	10886.0	61.886460	19.245033	0.00	47.0000	62.000	77.0000	100.0000
windspeed	10886.0	12.799395	8.164537	0.00	7.0015	12.998	16.9979	56.9969
casual	10886.0	36.021955	49.960477	0.00	4.0000	17.000	49.0000	367.0000
registered	10886.0	155.552177	151.039033	0.00	36.0000	118.000	222.0000	886.0000
count	10886.0	191.574132	181.144454	1.00	42.0000	145.000	284.0000	977.0000

- **These statistics provide insights into the central tendency, spread, and range of the numerical features in the dataset.**
- **Maximum usages are during "Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog" followed by "Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist**
- **Booking by casual user is very low than registered users.**

Unique

In [11]:

```
df.nunique()
```

Out[11]:

```
datetime      10886
season            4
holiday           2
workingday        2
weather           4
temp             49
atemp            60
humidity         89
windspeed        28
casual          309
registered      731
count           822
dtype: int64
```

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object        
 2   holiday     10886 non-null  object        
 3   workingday  10886 non-null  object        
 4   weather     10886 non-null  int64         
 5   temp        10886 non-null  float64       
 6   atemp       10886 non-null  float64       
 7   humidity    10886 non-null  int64         
 8   windspeed   10886 non-null  float64       
 9   casual      10886 non-null  int64         
 10  registered  10886 non-null  int64         
 11  count       10886 non-null  int64         
dtypes: datetime64[ns](1), float64(3), int64(5), object(3)
memory usage: 1020.7+ KB
```

Insights

- **There are 11 numerical data and one object type data.**

- **There conversion, we have 1 datetime64[ns], 3 object data season, holiday, workingday,4 integer data and 3 float data type.**
- **No missing data or null values present neither any duplicate row is there.**

Optimizing Memory Usage of the Dataframe

Updating dtype of season column

In [13]:

```
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 sav
df['season'] = df['season'].astype('category')
print('Updated Memory usage of season column : ', df['season'].memory_usage())
```

```
Memory usage of season column :  87216
Updated Memory usage of season column :  11218
```

Updating dtype of count column

In [14]:

```
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 t
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 :  87216
Updated Memory usage of count column :  21900
```

Updating dtype of holiday column

In [15]:

```
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 t
df['holiday'] = df['holiday'].astype('category')
print('Updated Memory usage of holiday column : ', df['holiday'].memory_usage())
```

```
Max value entry in holiday column :  non_holiday
Memory usage of holiday column :  87216
Updated Memory usage of holiday column :  11138
```

Updating dtype of registered column

In [16]:

```
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 conv
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 :  87216
Updated Memory usage of registered column :  21900
```

Updating dtype of casual column

In [17]:

```
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
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 :  87216
Updated Memory usage of casual column :  21900
```

Updating dtype of count column

In [18]:

```
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 t
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 :  21900
Updated Memory usage of count column :  21900
```

Updating dtype of windspeed column

In [19]:

```
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
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 :  87216
Updated Memory usage of windspeed column :  43672
```

Updating dtype of weather column

In [20]:

```
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 t
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 :  87216
Updated Memory usage of weather column :  11218
```

Updating dtype of humidity column

In [21]:

```
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 conver
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 :  87216
Updated Memory usage of humidity column :  11014
```

Updating dtype of workingday column

In [22]:

```
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 conver
df['workingday'] = df['workingday'].astype('category')
print('Updated Memory usage of workingday column : ', df['workingday'].memory_usage())
```

```
Max value entry in workingday column :  working
Memory usage of workingday column :  87216
Updated Memory usage of workingday column :  11138
```

Updating dtype of temp column

In [23]:

```
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
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 :  87216
Updated Memory usage of temp column :  43672
```

Updating dtype of atemp column

In [24]:

```
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 conv
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 :  87216
Updated Memory usage of atemp column :  43672
```

In [25]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  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
dtypes: category(4), datetime64[ns](1), float32(3), int16(3), int8(1)
memory usage: 330.3 KB
```

Earlier the dataset was using 1.2+ MB of memory but now it has been reduced to 500.3+ KB. Around 41.6 % reduction in the memory usage.

In [26]:

```
df
```

Out[26]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	v
0	2011-01-01 00:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	81	
1	2011-01-01 01:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	
2	2011-01-01 02:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	
3	2011-01-01 03:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	
4	2011-01-01 04:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	
...	...	...	...	...	...	...	...	...	
10881	2012-12-19 19:00:00	winter	non_holiday	working	1	15.58	19.695000	50	
10882	2012-12-19 20:00:00	winter	non_holiday	working	1	14.76	17.424999	57	
10883	2012-12-19 21:00:00	winter	non_holiday	working	1	13.94	15.910000	61	
10884	2012-12-19 22:00:00	winter	non_holiday	working	1	13.94	17.424999	61	
10885	2012-12-19 23:00:00	winter	non_holiday	working	1	13.12	16.665001	66	

10886 rows × 12 columns

In [27]:

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

Out[27]:

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

In [28]:

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

Out[28]:

```
non_holiday    97.14
holiday         2.86
Name: holiday, dtype: float64
```

In [29]:

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

Out[29]:

```
working            68.09
holiday_weekend    31.91
Name: workingday, dtype: float64
```

In [30]:

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

Out[30]:

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

Univariate Analysis and Bivariate Analysis

In [31]:

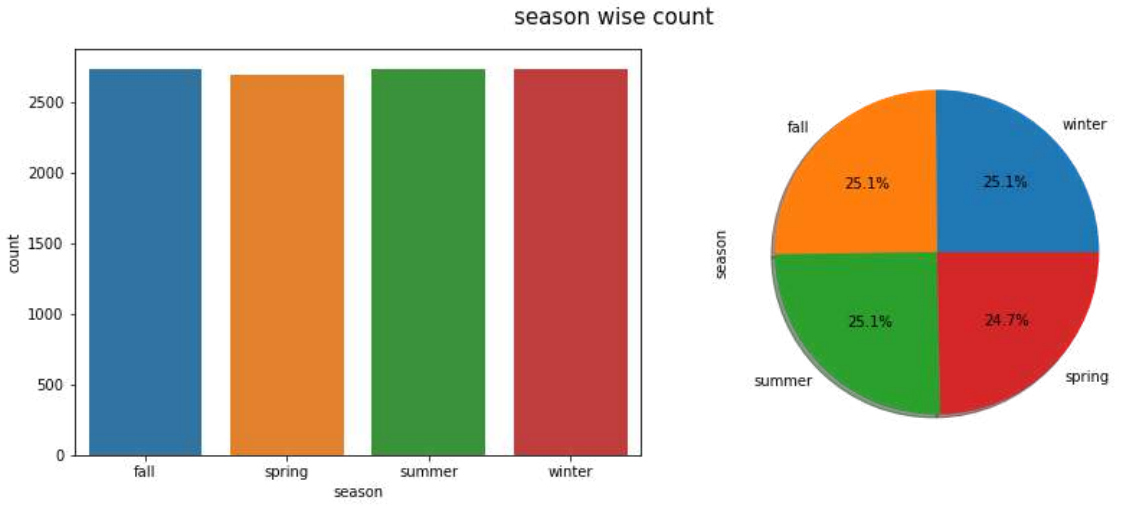
```
#EDA on Univariate Categorical variables
def cat_feat(col_data):
    fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(12,5))
    fig.suptitle(col_data.name+' wise count',fontsize=15)
    sns.countplot(col_data,ax=ax[0])
    col_data.value_counts().plot.pie(autopct='%1.1f%%',ax=ax[1], shadow = True)
    plt.tight_layout()
cat_cols = ['season', 'holiday', 'workingday', 'weather']
cat_cols
```

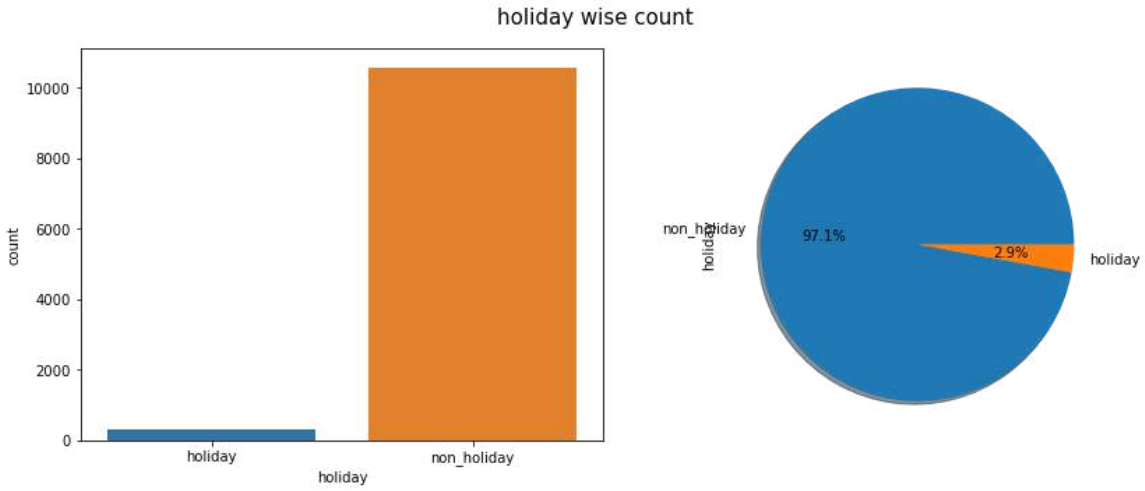
Out[31]:

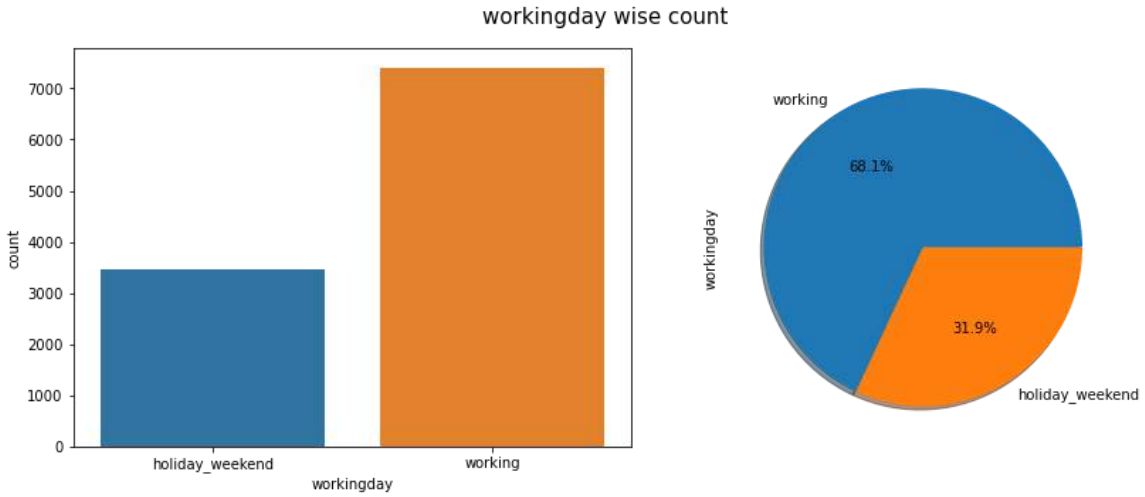
```
['season', 'holiday', 'workingday', 'weather']
```

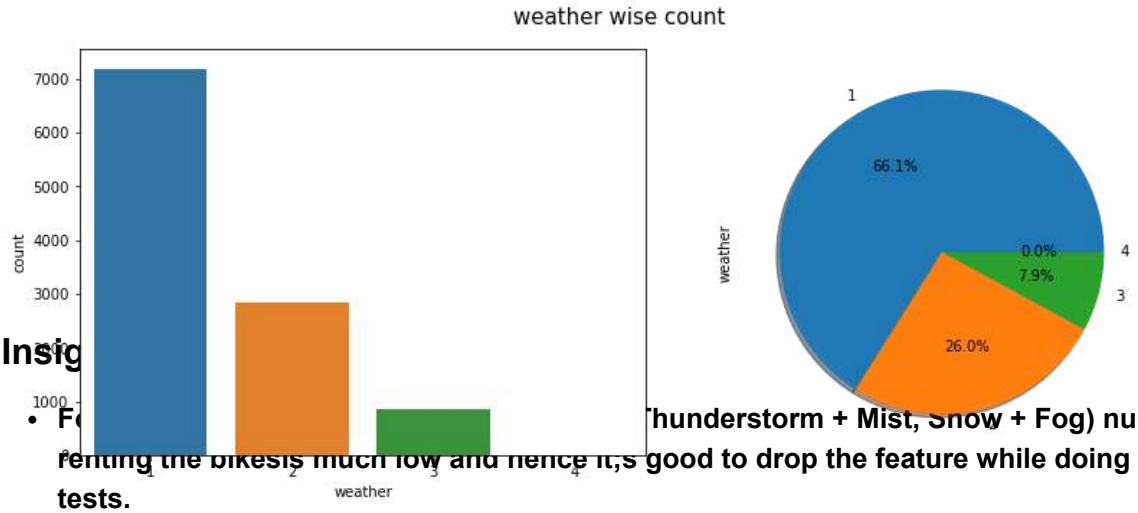
In [32]:

```
for i in cat_cols:
    cat_feat(df[i])
```









Insig

- **F Thunderstorm + Mist, Snow + Fog) number of users renting the bikesis much low and hence it,s good to drop the feature while doing the further tests.**
- **When weather is good (Clear, Few clouds, partly cloudy, partly cloudy) people tend to rent more bikes.**
- **Count of renting the bikes on working day is much higher than non-working day.**
- **During Holidays people dont prefer to ride bikes.**
- **During season (spring,summer,fall,winter) the count of renting the bikes is more or less.**

In [33]:

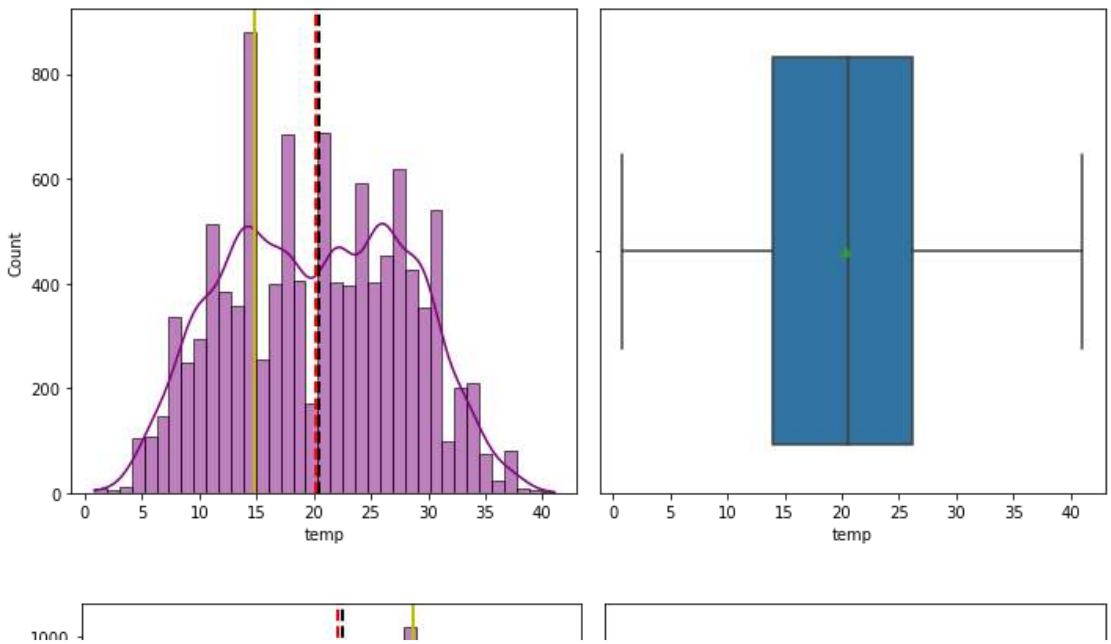
```
#Univariate analysis for numerical/continuos variables
def num_feat(col_data):
    fig,ax = plt.subplots(nrows=1,ncols=2,figsize=(10,5))
    sns.histplot(col_data, kde=True, ax=ax[0], color = 'purple')
    ax[0].axvline(col_data.mean(), color='r', linestyle='--',linewidth=2)
    ax[0].axvline(col_data.median(), color='k', linestyle='dashed', linewidth=2)
    ax[0].axvline(col_data.mode()[0],color='y',linestyle='solid',linewidth=2)
    sns.boxplot(x=col_data, showmeans=True, ax=ax[1])
    plt.tight_layout()

num_cols = ['temp','atemp','humidity','count','windspeed']
num_cols
```

Out[33]:

```
['temp', 'atemp', 'humidity', 'count', 'windspeed']
```

In [34]:

```
for i in num_cols:
    num_feat(df[i])
```



Insights

- **There are outliers in the windspeed and casual users which tells us that, the windspeed is not uniform.**
- **The exponential decay curve for the count tells us that, as the users renting bikes increases the frequency decreases.**
- **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 and count.**

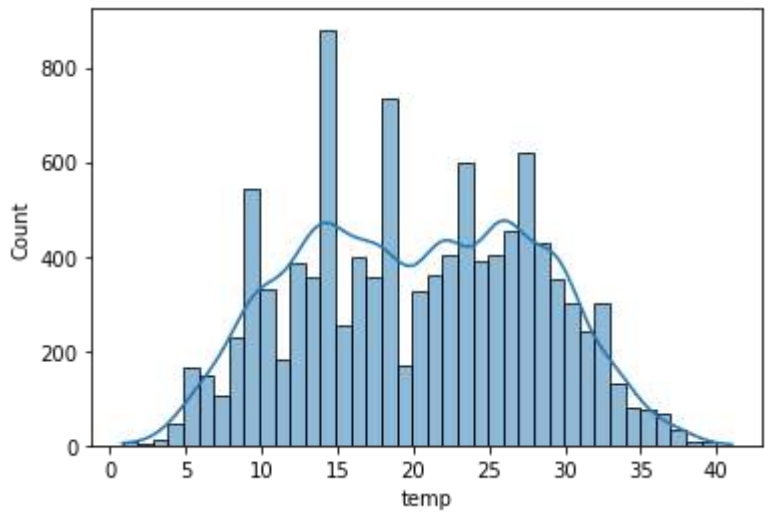
Check assumptions of the test (Normality, Equal Variance). You can check it using Histogram, Q-Q plot or statistical methods like levene's test, Shapiro-wilk test (optional):Histogram

In [35]:

```
# The below code generates a histogram plot for the 'temp' feature, showing the distribut
    # 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 a
    # data distribution.

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

Out[35]:

```
[]
```



In [36]:

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

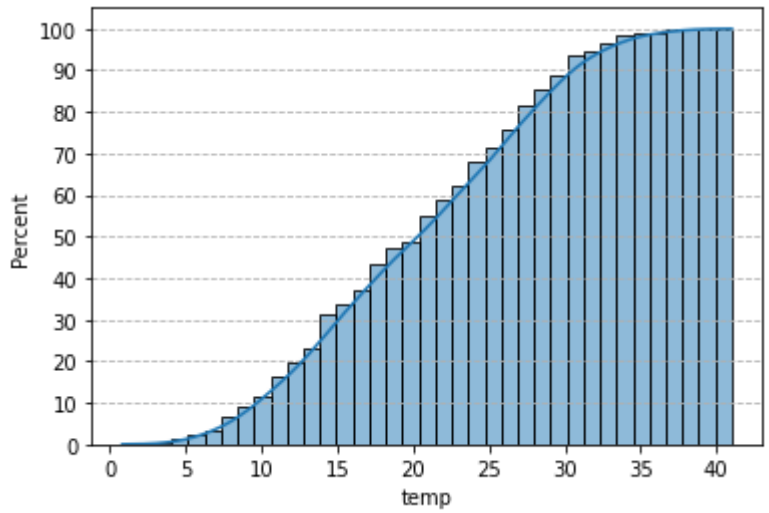
Out[36]:

```
(20.23, 7.79)
```

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

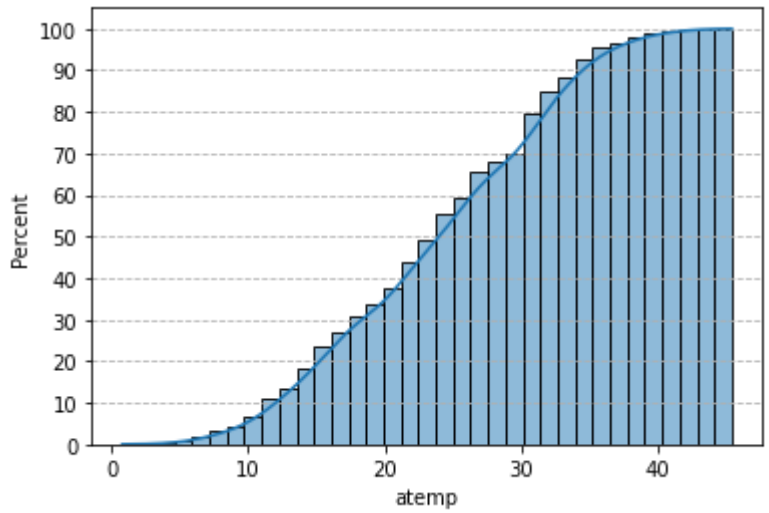
In [37]:

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



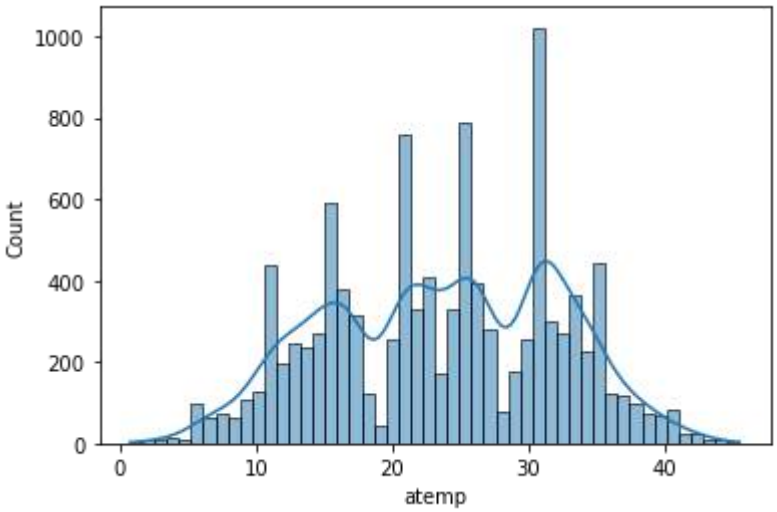
In [38]:

```
# The below code generates a histogram plot for the 'atemp' 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 a
    # data distribution.

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();
```



In [39]:

```
# The below code generates a histogram plot for the 'atemp' feature, showing the distribu
    # 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 a
    # data distribution.

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



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

In [40]:

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

Out[40]:

```
(23.66, 7.79)
```

In [41]:

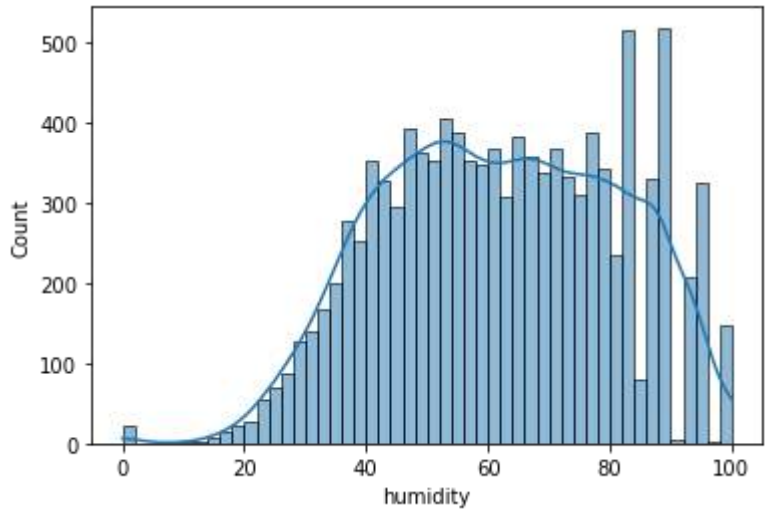
```
humidity_mean = np.round(df['humidity'].mean(), 2)
humidity_std = np.round(df['humidity'].std(), 2)
humidity_mean, humidity_std
```

Out[41]:

```
(61.89, 19.25)
```

In [42]:

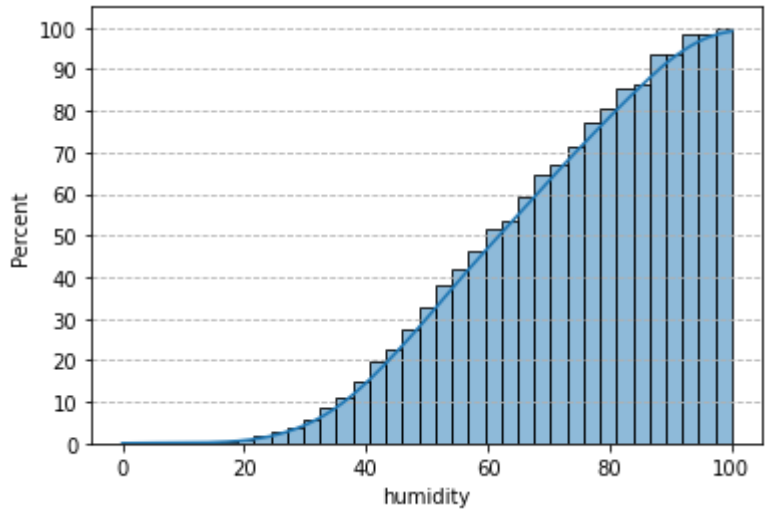
```
# The below code generates a histogram plot for the 'atemp' feature, showing the distribu
    # feeling temperature values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape, making it easier to a
    #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

sns.histplot(data = df, x = 'humidity', kde = True, bins = 50)
plt.plot();
```



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

In [43]:

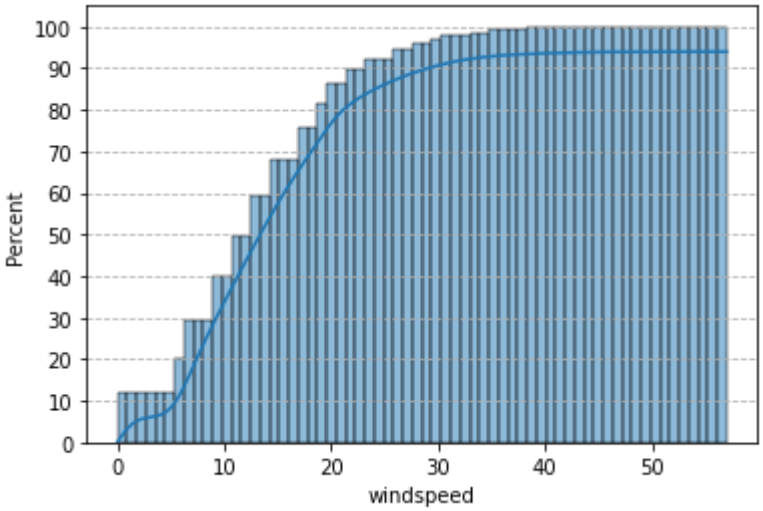
```
# The below code generates a histogram plot for the 'humidity' feature, showing the cumul
    # distribution of humidity values in the dataset.
# The addition of the kernel density estimation plot provides
    # a visual representation of the underlying distribution shape, making it easier to a
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

In [44]:

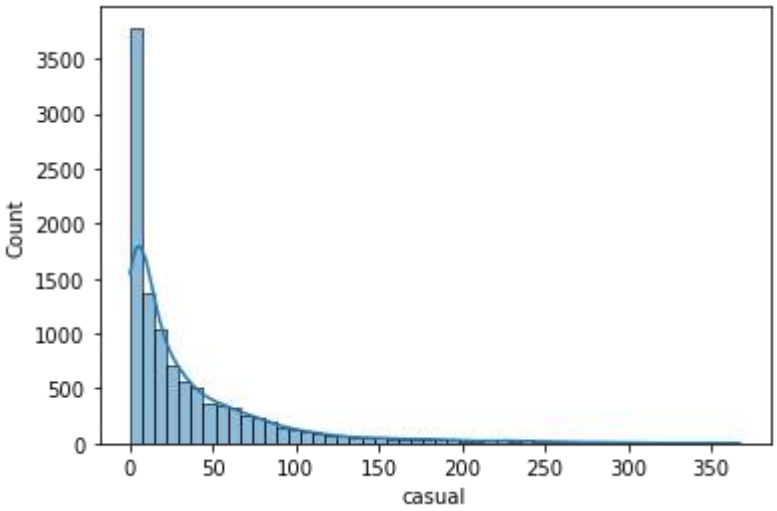
```
sns.histplot(data = df, x = 'windspeed', kde = True, cumulative = True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot();       # displaying the chart
```



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

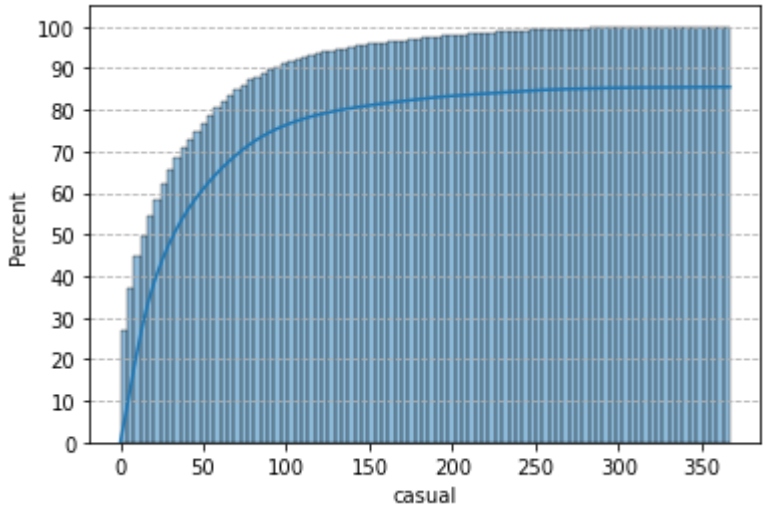
In [45]:

```
# The below code generates a histogram plot for the 'casual' feature, showing the distrib
    # 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 a
    # data distribution.

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



In [46]:

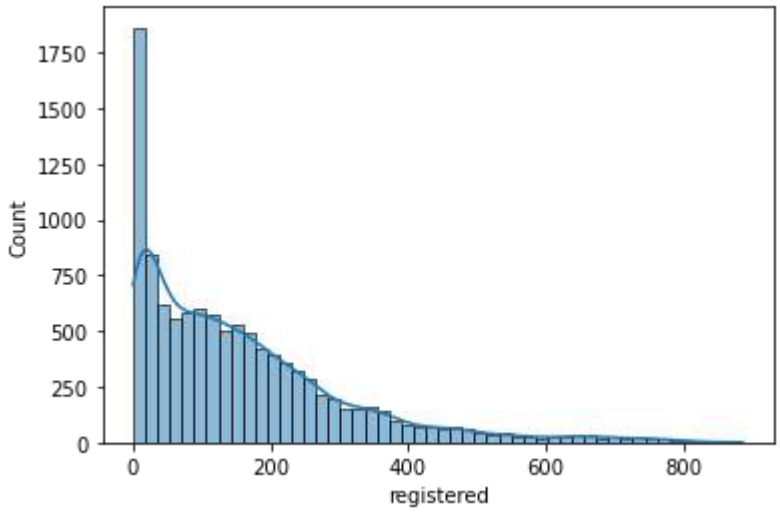
```
sns.histplot(data = df, x = 'casual', kde = True, cumulative = True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot();        # displaying the chart
```



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

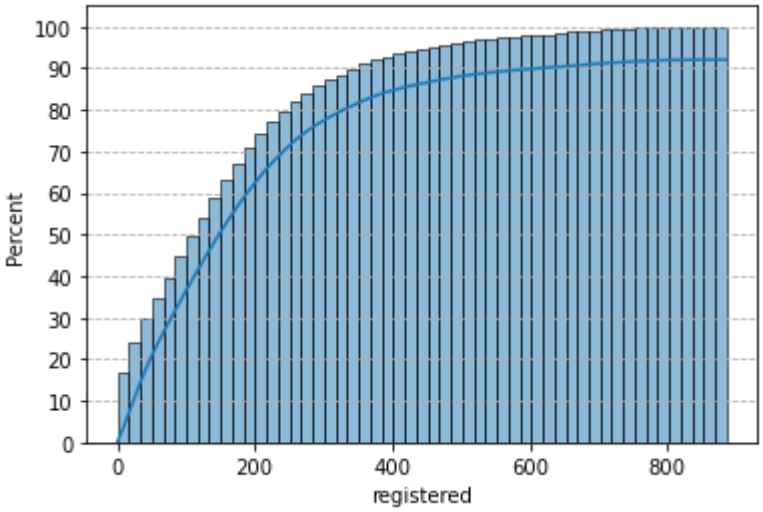
In [47]:

```
# The below code generates a histogram plot for the 'registered' feature, showing the dis
    # 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 a
    # data distribution.

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



In [48]:

```
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
```

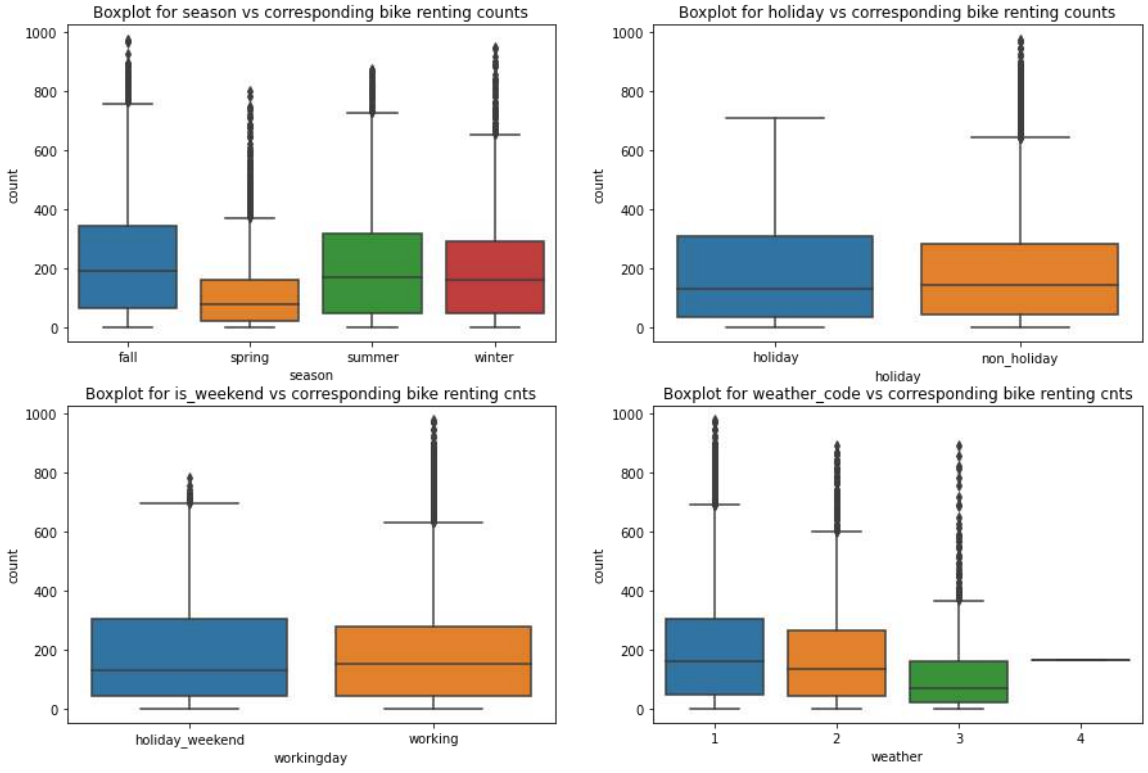


- **More than 85 % of the time, the count of registered users is less than 300.**
- **Histograms show that temp,atemp,windspeed,registered,casual ,none of them are normal distributions.**

Outlier detections and removal

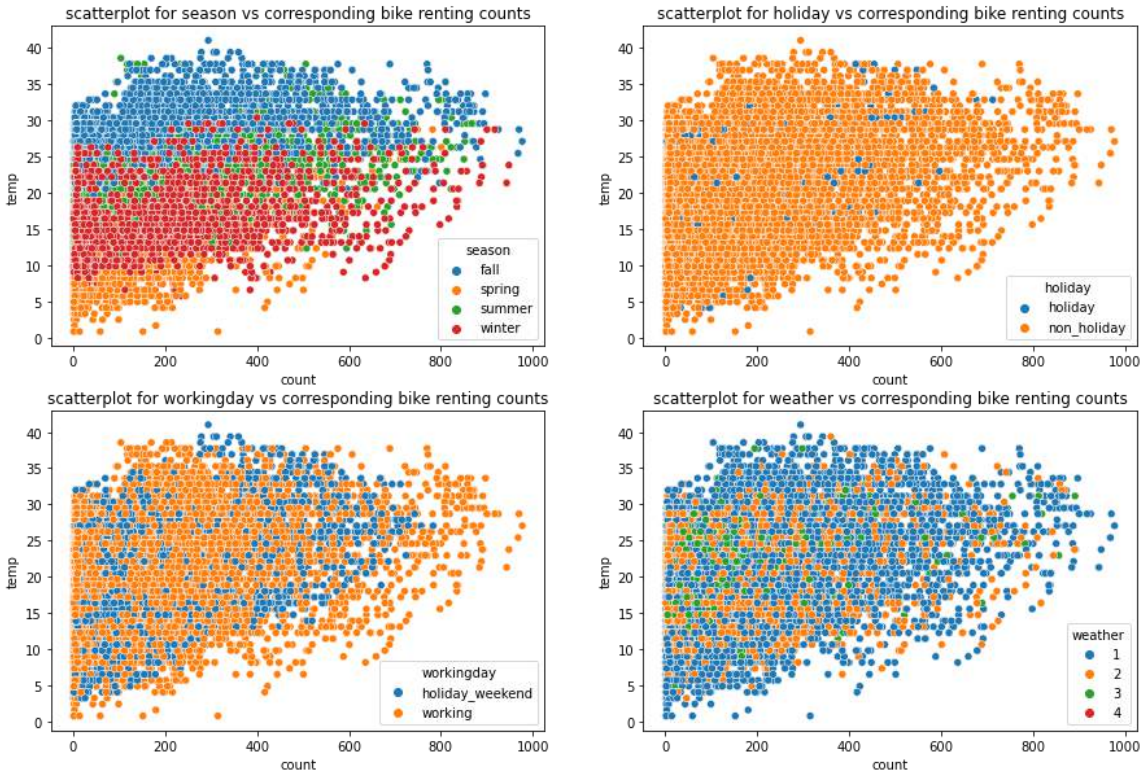
In [49]:

```
# Visualization before outlier removal
fig = plt.figure(figsize = (15,10))

ax1=fig.add_subplot(221)
sns.boxplot(x='season',y='count',data=df)
ax1.set_title('Boxplot for season vs corresponding bike renting counts')

ax1=fig.add_subplot(222)
sns.boxplot(x='holiday',y='count',data=df)
ax1.set_title('Boxplot for holiday vs corresponding bike renting counts')

ax1 = fig.add_subplot(223)
sns.boxplot(x = 'workingday', y = 'count', data = df)
ax1.set_title('Boxplot for is_weekend vs corresponding bike renting cnts')

ax1 = fig.add_subplot(224)
sns.boxplot(x = 'weather', y = 'count', data = df)
ax1.set_title('Boxplot for weather_code vs corresponding bike renting cnts')

plt.show()
```



In [50]:

```
fig = plt.figure(figsize = (15,10))

ax1 = fig.add_subplot(221)
sns.scatterplot(x = 'count', y = 'temp',data = df, hue ='season' )
ax1.set_title('scatterplot for season vs corresponding bike renting counts')

ax1 = fig.add_subplot(222)
sns.scatterplot(x ='count', y = 'temp', data = df, hue ='holiday')
ax1.set_title('scatterplot for holiday vs corresponding bike renting counts')

ax1 = fig.add_subplot(223)
sns.scatterplot(x = 'count', y = 'temp',data = df, hue ='workingday')
ax1.set_title('scatterplot for workingday vs corresponding bike renting counts')

ax1 = fig.add_subplot(224)
sns.scatterplot(x = 'count',y = 'temp',data = df, hue ='weather')
ax1.set_title('scatterplot for weather vs corresponding bike renting counts')

plt.show()
```



In [51]:

```
# Taking backup of original data before removing outliers
df_copy = df.copy()
```

In [52]:

```
q1=df['count'].quantile(0.25)
q3=df['count'].quantile(0.75)
iqr=q3-q1
df = df[(df['count'] >= q1 - 1.5*iqr) & (df['count'] <= q3 +1.5*iqr)]
df.shape
```

Out[52]:

```
(10586, 12)
```

In [53]:

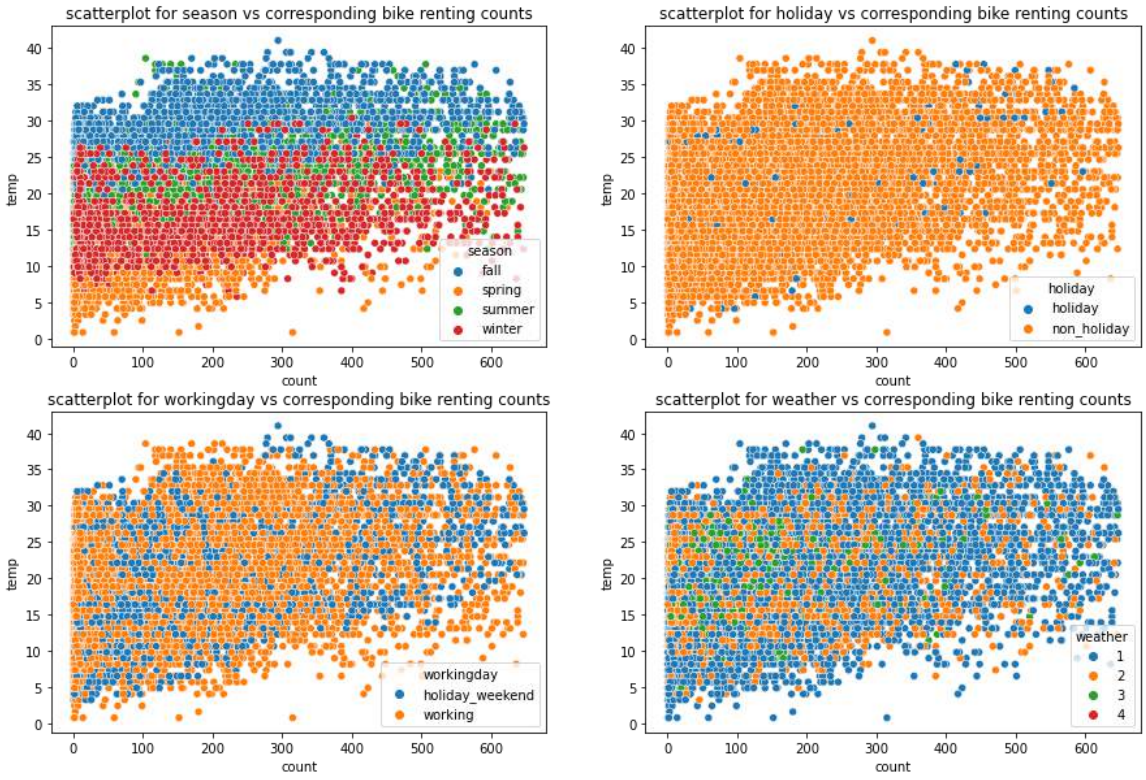
```
df_copy.shape[0] - df.shape[0]
```

Out[53]:

```
300
```

In [54]:

```
#Visualization after removing outliers
fig = plt.figure(figsize = (15,10))

ax1 = fig.add_subplot(221)
sns.scatterplot(x = 'count', y = 'temp',data = df, hue ='season' )
ax1.set_title('scatterplot for season vs corresponding bike renting counts')

ax1 = fig.add_subplot(222)
sns.scatterplot(x ='count', y = 'temp', data = df, hue ='holiday')
ax1.set_title('scatterplot for holiday vs corresponding bike renting counts')

ax1 = fig.add_subplot(223)
sns.scatterplot(x = 'count', y = 'temp',data = df, hue ='workingday')
ax1.set_title('scatterplot for workingday vs corresponding bike renting counts')

ax1 = fig.add_subplot(224)
sns.scatterplot(x = 'count',y = 'temp',data = df, hue ='weather')
ax1.set_title('scatterplot for weather vs corresponding bike renting counts')

plt.show()
```

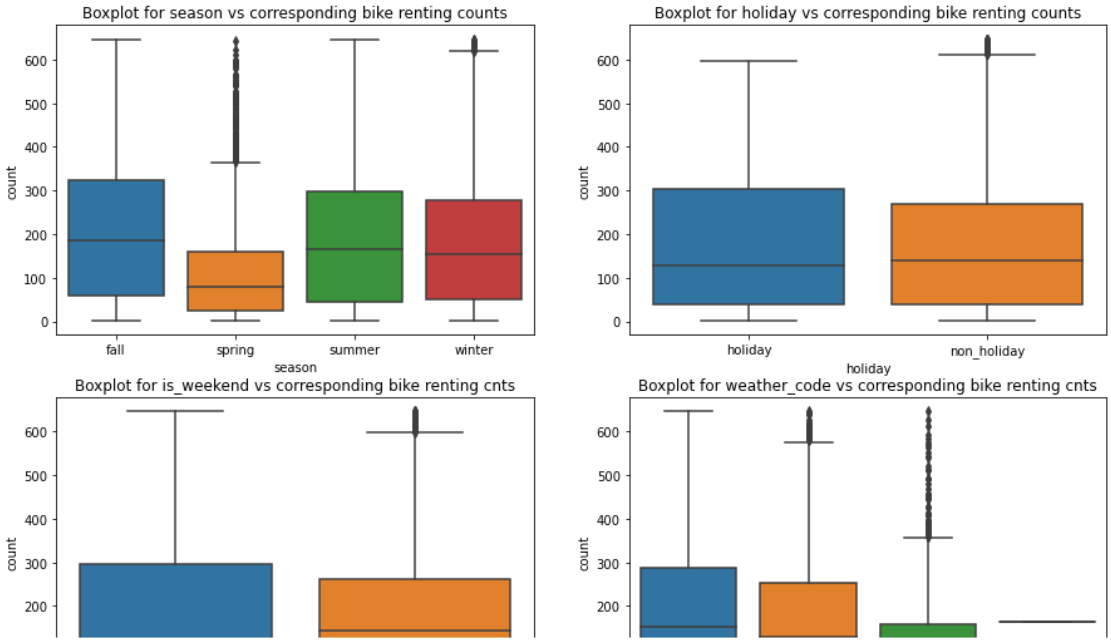


Insights

- After dealing with the outliers, total of 300 rows are removed out off 10886 from the dataset.

In [55]:

```
fig = plt.figure(figsize = (15,10))

ax1=fig.add_subplot(221)
sns.boxplot(x='season',y='count',data=df)
ax1.set_title('Boxplot for season vs corresponding bike renting counts')

ax1=fig.add_subplot(222)
sns.boxplot(x='holiday',y='count',data=df)
ax1.set_title('Boxplot for holiday vs corresponding bike renting counts')

ax1 = fig.add_subplot(223)
sns.boxplot(x = 'workingday', y = 'count', data = df)
ax1.set_title('Boxplot for is_weekend vs corresponding bike renting cnts')

ax1 = fig.add_subplot(224)
sns.boxplot(x = 'weather', y = 'count', data = df)
ax1.set_title('Boxplot for weather_code vs corresponding bike renting cnts')

plt.show()
```



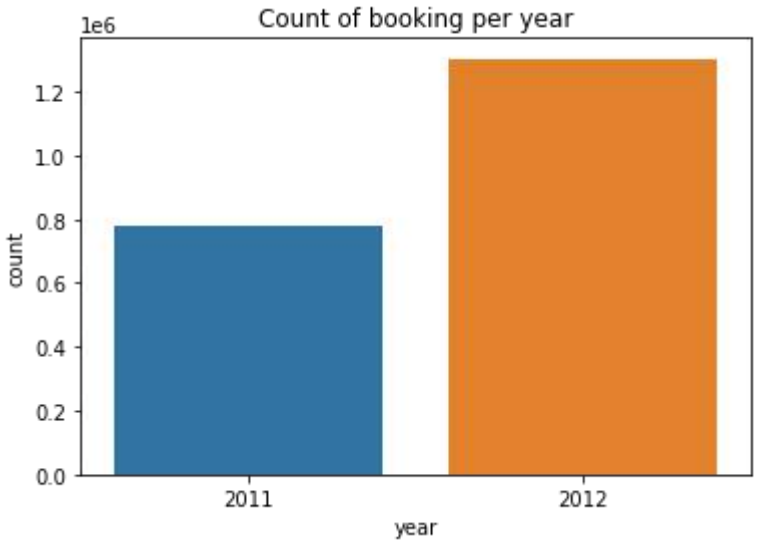
- **The hourly count of total rental bikes is higher in the fall season, followed by the summer and winter seasons. It is generally low in the spring season. Although, the median and min demand of bikes in summer and winter are almost same.**
- **The median and minimum hourly count of total rental bikes on holidays is almost similar to the median and minimum hourly count of bikes on non holidays. The count of on holidays is higher than non holidays.**
- **The count of bikes on weekend or holidays is higher than working days.**
- **The hourly count of total rental bikes is higher in the clear and cloudy weather, followed by the misty weather and rainy weather. There is just one record for extreme weather conditions.**

In [58]:

```
df["Date"]=df['datetime'].dt.date
df["time"]=df['datetime'].dt.time
df["year"]=df['datetime'].dt.year
df["month"]=df['datetime'].dt.month
```

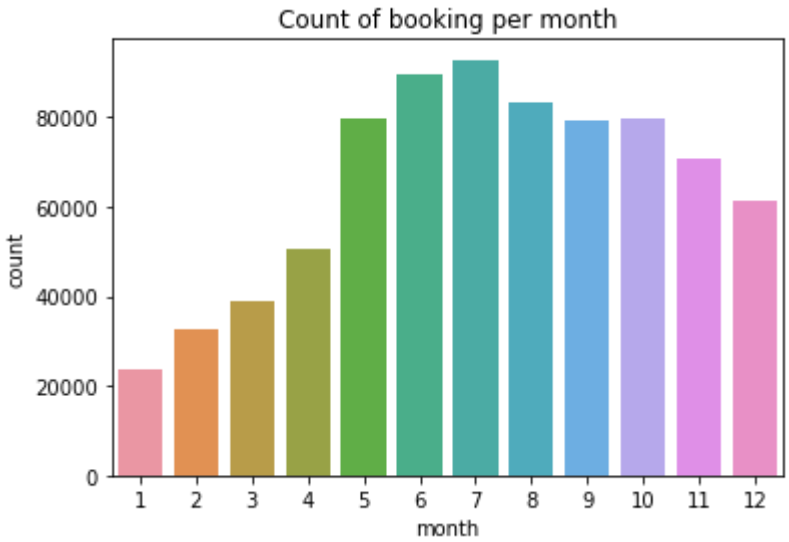
In [59]:

```
df=df_copy
year_data = df.groupby(['year'])['count'].sum()
year_data = year_data.reset_index()
sns.barplot(x='year',y='count',data=year_data)
plt.title('Count of booking per year')
plt.show()
```



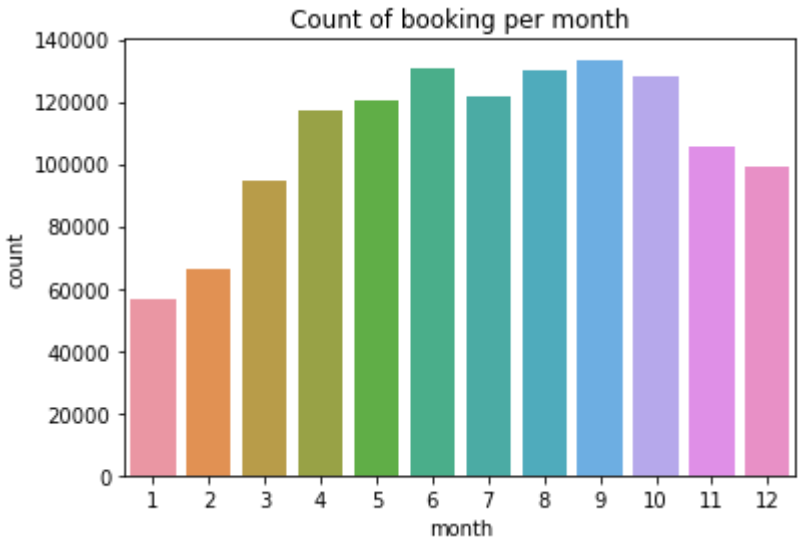
In [60]:

```
df_2011=df[df["year"]==2011]
month_data =df_2011.groupby(['month'])['count'].sum()
month_data = month_data.reset_index()
sns.barplot(x='month',y='count',data=month_data)
plt.title('Count of booking per month')
plt.show()
```



In [61]:

```
df_2012=df[df["year"]==2012]
month_data =df_2012.groupby(['month'])['count'].sum()
month_data = month_data.reset_index()
sns.barplot(x='month',y='count',data=month_data)
plt.title('Count of booking per month')
plt.show()
```



Insights

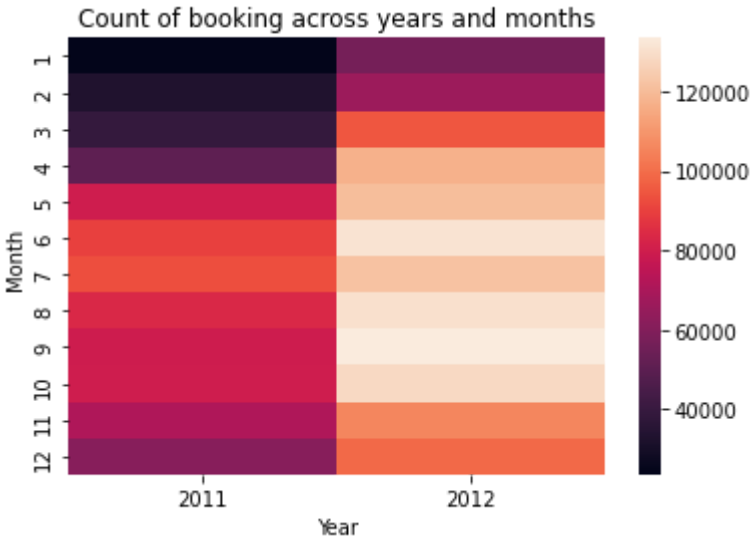
- Highest booking is in the month of July and September in 2011 and 2012 respectively.
- A growing trend from Jan to June in both years and a decline from September to December.
- January is the minimum uasage month in both years.
- In 2011, there is a gradually decline from July to December.

In [62]:

```
mon_year_data = df.groupby(['year','month'])['count'].sum()
mon_year_data = pd.DataFrame(mon_year_data)
mon_year_data.reset_index(inplace = True)
myy = mon_year_data.pivot('month','year','count')
```

In [63]:

```
sns.heatmap(myy)
plt.title('Count of booking across years and months')
plt.xlabel('Year')
plt.ylabel('Month')
plt.show()
```



Insights

- **More booking in 2012 compare to 2011 in each month.**

In [64]:

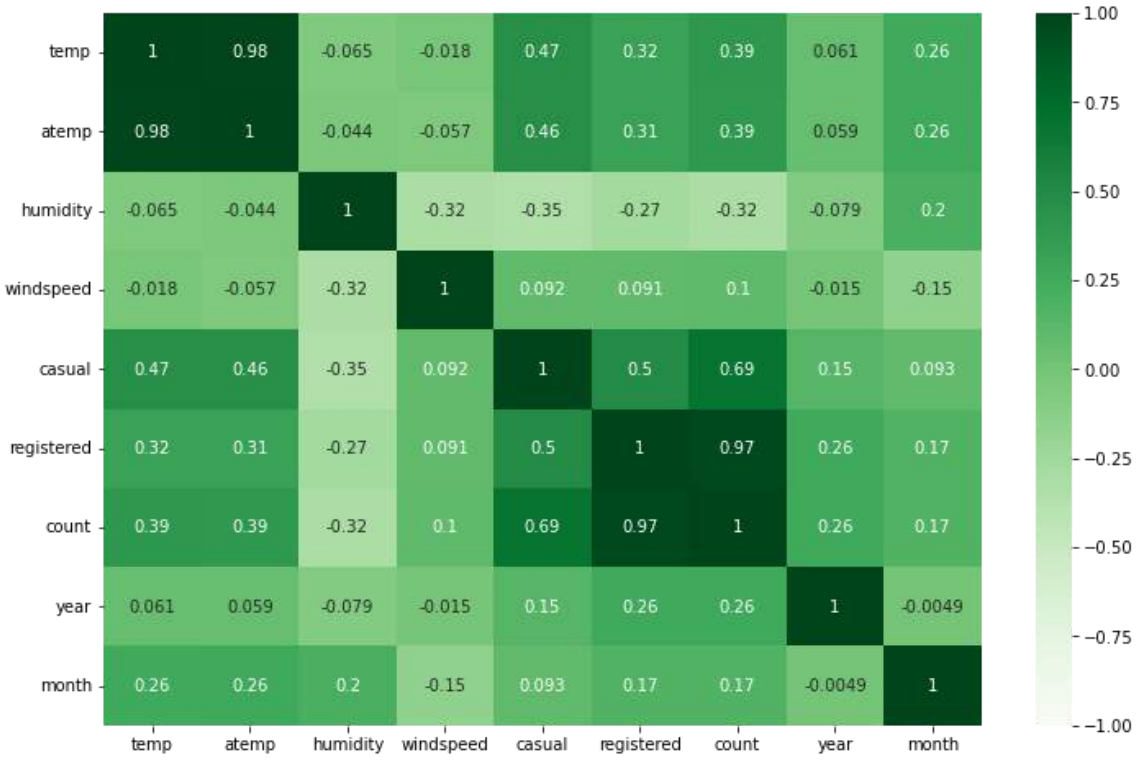
```
corr_data = df.corr()
corr_data
```

Out[64]:

	temp	atemp	humidity	windspeed	casual	registered	count	
temp	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.394454	0.06
atemp	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.389784	0.05
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.317371	-0.07
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.101369	-0.01
casual	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.690414	0.14
registered	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.970948	0.26
count	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.000000	0.26
year	0.061226	0.058540	-0.078606	-0.015221	0.145241	0.264265	0.260403	1.00
month	0.257589	0.264173	0.204537	-0.150192	0.092722	0.169451	0.166862	-0.00

In [65]:

```
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.

Dataset's is the time period for which the data is given ?

In [66]:

```
df.describe(include = ['object','datetime64[ns]']).T
```

Out[66]:

	count	unique	top	freq	first	last
datetime	10886	10886	2011-01-01 00:00:00	1	2011-01-01	2012-12-19 23:00:00
Date	10886	456	2011-01-01	24	NaT	NaT
time	10886	24	12:00:00	456	NaT	NaT

- The data is given for the time period 2011-01-01 to 2012-12-19.
- Most usages are done in winter season.

In [67]:

```
df['Day'] = df['datetime'].dt.day_name()
df
```

Out[67]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	ca
0	2011-01-01 00:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	81	0.000000	
1	2011-01-01 01:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	0.000000	
2	2011-01-01 02:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	0.000000	
3	2011-01-01 03:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	0.000000	
	2011-01-									

In [68]:

```
df.groupby('Day')["count"].sum().sort_values()
```

Out[68]:

```
Day
Sunday       285546.0
Tuesday      291985.0
Wednesday    292226.0
Monday       295296.0
Friday       302504.0
Thursday     306401.0
Saturday     311518.0
Name: count, dtype: float64
```

- **Maximum electric bikes are rented on Saturday followed by Thursday.**

In [69]:

```
df.set_index("datetime", inplace = True)
df
```

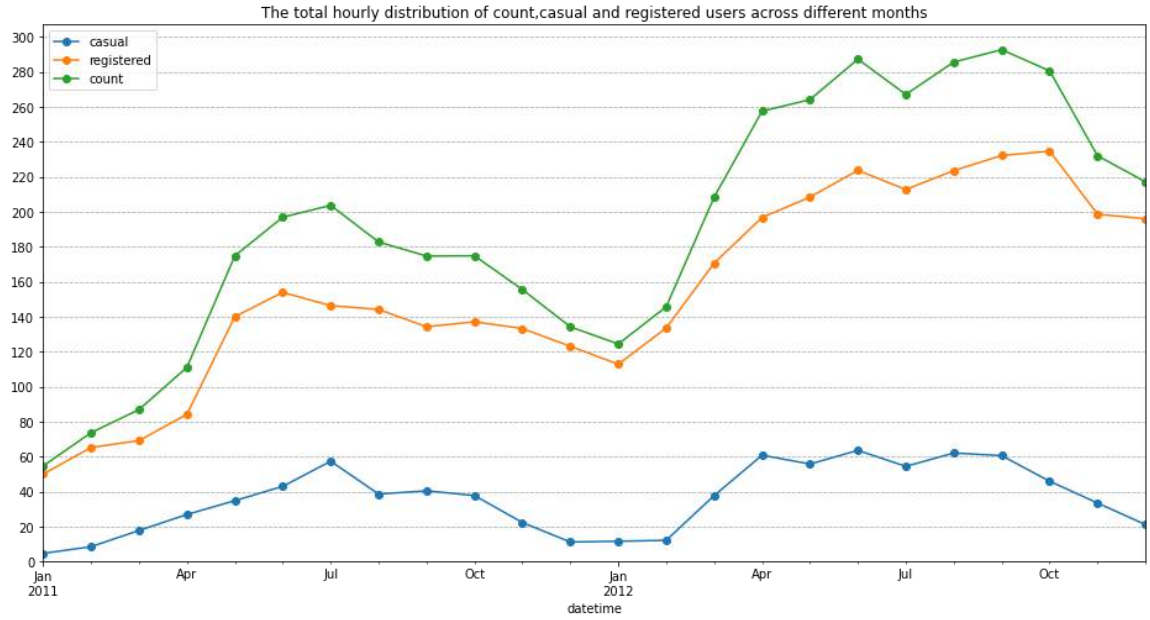
Out[69]:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	re
2011-01-01 00:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	81	0.000000	3	
2011-01-01 01:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	0.000000	8	
2011-01-01 02:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	0.000000	5	
2011-01-01 03:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	0.000000	3	

Slicing Data by Time

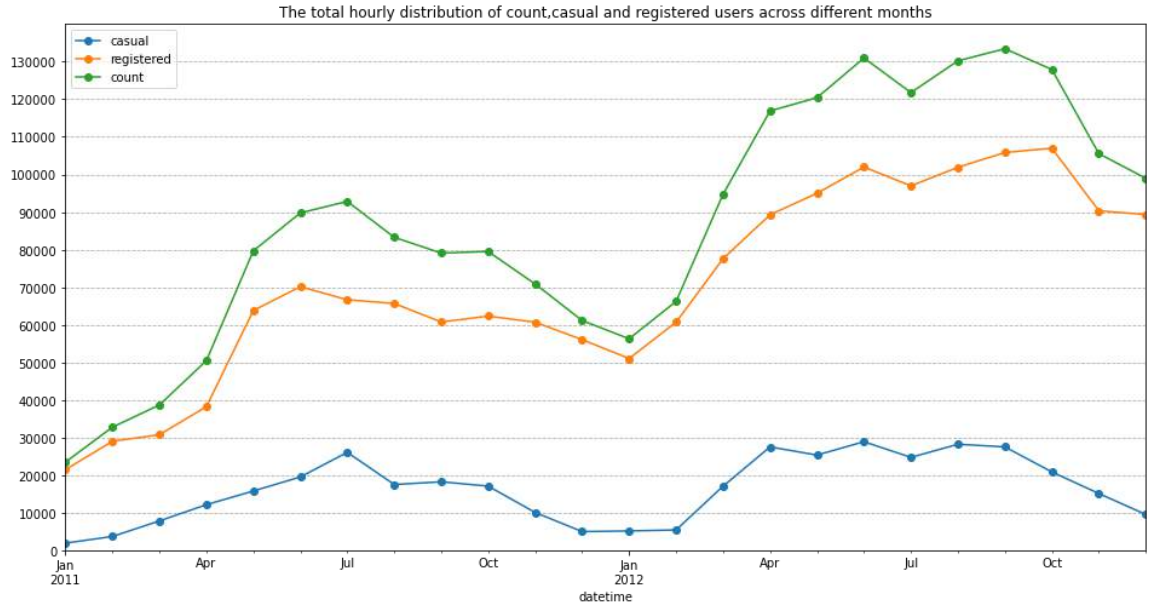
In [70]:

```
# The below code visualizes the trend of the monthly average values for the 'casual', 're
    # and 'count' variables,  allowing for easy comparison and analysis of their patterns
plt.figure(figsize=(16,8))
plt.title("The total hourly distribution of count,casual and registered users across diff
df.resample('M')['casual'].mean().plot(kind = 'line', legend = 'casual', marker="o")
df.resample('M')['registered'].mean().plot(kind = 'line', legend = 'registered', marker =
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()
```



In [71]:

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

plt.figure(figsize = (16, 8))
# Setting the title for the plot
plt.title("The total hourly distribution of count,casual and registered users across diff
# 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 =
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
```



- **The highest average hourly users count is in summer and minimum in winter.**

Overall, these trends suggest a seasonal pattern in the usages 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.

Is there an increase in the demand of rental electric cycles from the year 2011 to 2012

In [72]:

```
# 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 ye
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1
```

Out[72]:

	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 reflects a substantial growth in the averegae hourly count of the electric cycles over the course of one year.**
- **The average hourly count of electric cycles 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.**
  - *It indicates positive growth and potentially a successful outcome or increasing demand for the variable being measured.*

In [73]:

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

Out[73]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	v
0	2011-01-01 00:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	81	
1	2011-01-01 01:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	
2	2011-01-01 02:00:00	spring	non_holiday	holiday_weekend	1	9.02	13.635000	80	
3	2011-01-01 03:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	
4	2011-01-01 04:00:00	spring	non_holiday	holiday_weekend	1	9.84	14.395000	75	
...	...	...	...	...	...	...	...	...	
10881	2012-12-19 19:00:00	winter	non_holiday	working	1	15.58	19.695000	50	
10882	2012-12-19 20:00:00	winter	non_holiday	working	1	14.76	17.424999	57	
10883	2012-12-19 21:00:00	winter	non_holiday	working	1	13.94	15.910000	61	
10884	2012-12-19 22:00:00	winter	non_holiday	working	1	13.94	17.424999	61	
10885	2012-12-19 23:00:00	winter	non_holiday	working	1	13.12	16.665001	66	

10886 rows × 17 columns

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

In [74]:

```
# Grouping the DataFrame by the month
df1 = df.groupby(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 mo
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1.set_index('month', inplace = True)
df1
```

Out[74]:

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

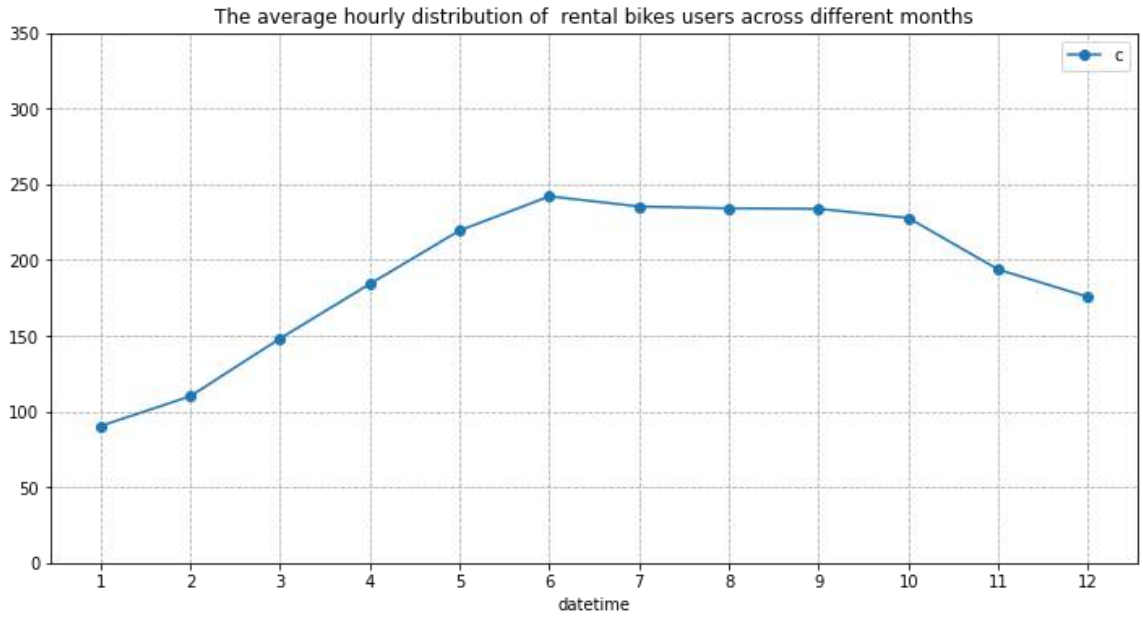
- **The rental bikes users shows an increasing trend from January to March, with a significant growth rate of 34.70% between February and March.**
- **The growth rate starts to stabilize from April to June, with a relatively smaller growth rate.**
- **From July to September, there is a slight decrease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

In [75]:

```
# The resulting plot visualizes the average hourly distribution of the count of rental bi
    # month, allowing for comparison and identification of any patterns or trends through

# 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  rental bikes users across different months

# Grouping the DataFrame by the month and calculating the mean of the 'count' column for
    # 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[75]:

```
[]
```



- **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 ?

In [76]:

```
# 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 ho
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1.set_index('hour', inplace = True)
df1
```

Out[76]:

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

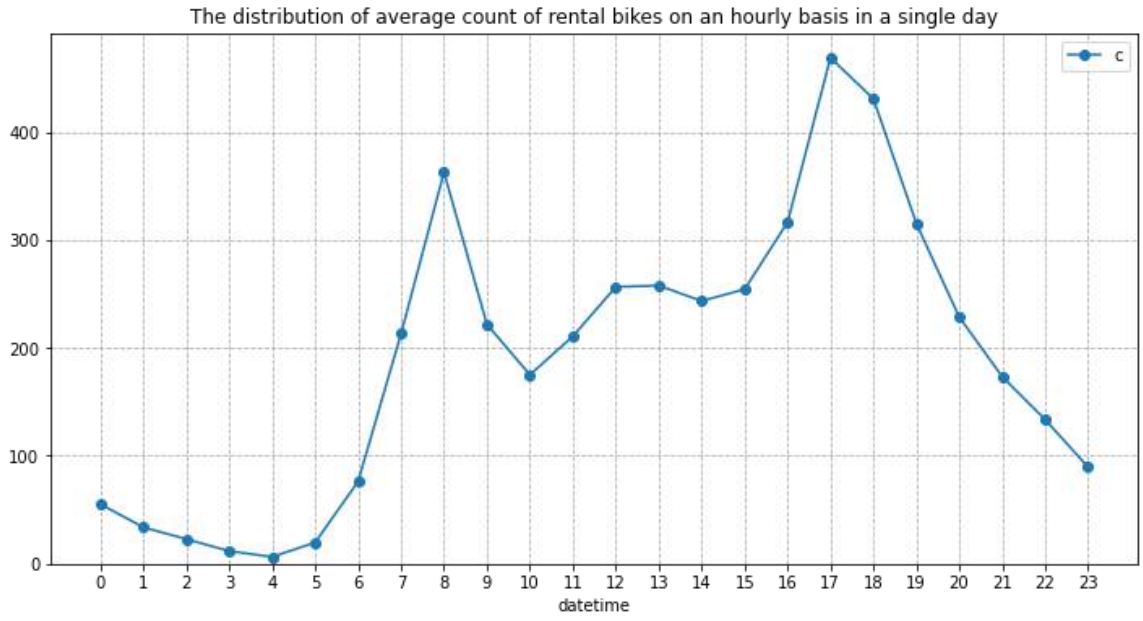
- **During the early morning hours (hours 0 to 5), there is a significant decrease in the count, with negative growth percentages ranging from -38.59% to -48.66%.**
- **However, starting from hour 5, there is a sudden increase in count, with a sharp positive growth percentage of 208.52% observed from 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.**

In [77]:

```
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on an hourly basis in a sing
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[77]:

```
[]
```



- **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.

What is the distribution of average count of rental bikes on weekly basis?

In [78]:

```
# Grouping the DataFrame by the hour
df1 = df.groupby( df['datetime'].dt.hour)['count'].mean().reset_index()
df1.rename(columns = {'datetime' : 'quarter'}, 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 ho
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1.set_index('quarter', inplace = True)
df1
```

Out[78]:

quarter	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

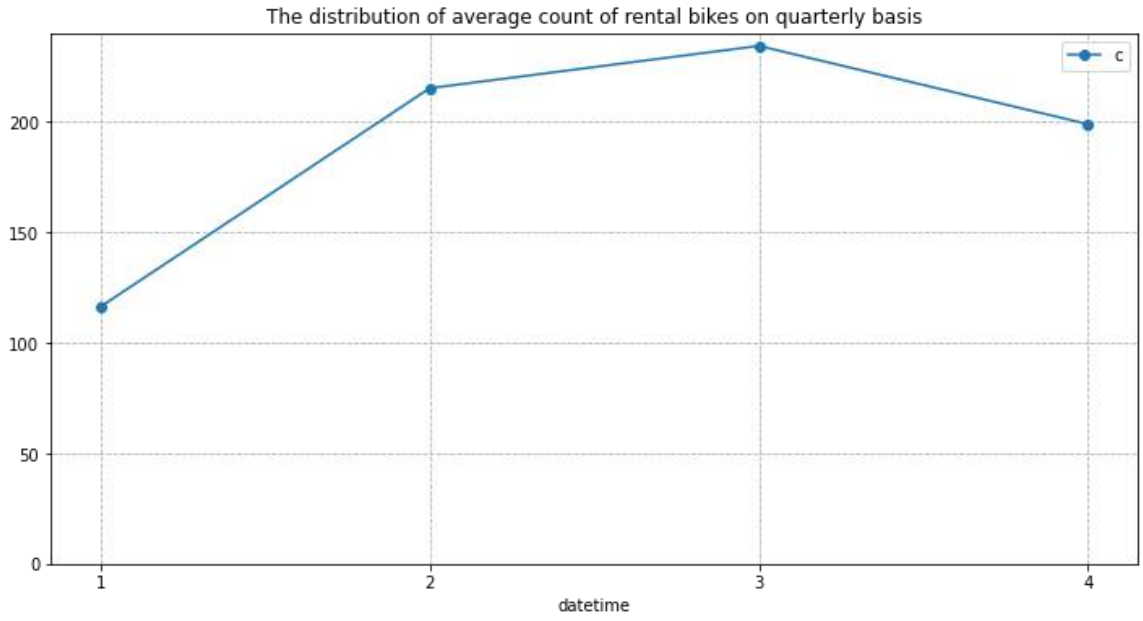
- **There is significant growth of 85% in second quarter.**

In [79]:

```
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on quarterly basis")
df.groupby(df['datetime'].dt.quarter)['count'].mean().plot(kind = 'line', marker = 'o')
plt.ylim(0,)
plt.xticks(np.arange(-1, 5))
plt.legend('count')
plt.grid(axis = 'both', linestyle = '--')
plt.plot()
```

Out[79]:

```
[]
```



- **There is an increasing trend in the weekly average count of rental bikes in 2nd and 3rd quarters, while with a decline in 4 quarter.**

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

Visual Tests to know if the samples follow normal distributionplt.figure(figsize = (15, 5))

In [80]:

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

Out[80]:

workingday	count	mean	std	min	25%	50%	75%	max
holiday_weekend	3474.0	188.506621	173.724015	1.0	44.0	128.0	304.0	783.0
working	7412.0	193.011873	184.513659	1.0	41.0	151.0	277.0	977.0

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

- Normal 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**

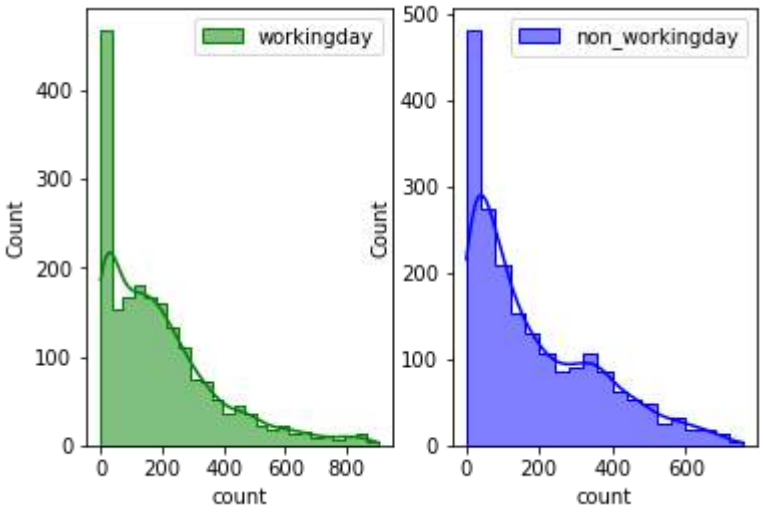
STEP-5: Compare p-value and alpha.

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

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

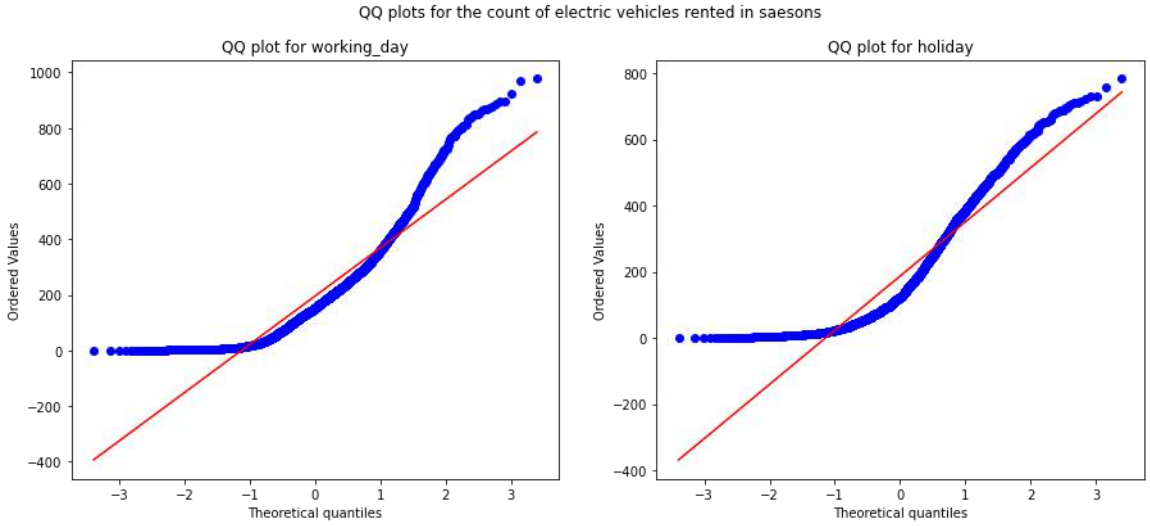
In [81]:

```
plt.subplot(1, 2, 1)
sns.histplot(df.loc[df['workingday'] == "working", 'count'].sample(2000),
             element = 'step', color = 'green', kde = True, label = 'workingday')
plt.legend()
plt.subplot(1, 2, 2)
sns.histplot(df.loc[df['workingday'] == "holiday_weekend", 'count'].sample(2000),
             element = 'step', color = 'blue', kde = True, label = 'non_workingday')
plt.legend()
plt.plot();
```



In [82]:

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in saesons')
spy.probplot(df.loc[df['workingday'] =="working" , 'count'].sample(2000), plot = plt, dis
plt.title('QQ plot for working_day')
plt.subplot(1, 2, 2)
spy.probplot(df.loc[df['workingday'] == "holiday_weekend", 'count'].sample(2000), plot =
plt.title('QQ plot for holiday')
plt.plot();
```



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

- **Applying Shapiro-Wilk test to check normality of weather column**

H0: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

In [83]:

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] =="working" , 'count'].sample(20
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 1.657288676722569e-37
The sample does not follow normal distribution
```

In [84]:

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] =="holiday_weekend" , 'count'].s
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 1.7496077217442417e-36
The sample does not follow normal distribution
```

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

In [85]:

```
transformed_workingday = spy.boxcox(df.loc[df['workingday'] == "working", '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 1.6156165171724373e-33
The sample does not follow normal distribution
```

In [86]:

```
transformed_non_workingday = spy.boxcox(df.loc[df['workingday'] == "holiday_weekend", 'co
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 8.139842317861285e-24
The sample does not follow normal distribution
```

After applying the boxcox transformation on each of the "working" and "holiday_weekend" data, the samples do not follow normal distribution.

Homogeneity of Variances using Lavene's test

In [87]:

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['workingday'] == "working", 'count'].sample(200
                                df.loc[df['workingday'] == "holiday_weekend", 'count'].sa
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 0.48898995674890544
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.

In [88]:

```
# 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'] == "working", 'count'],
                                      df.loc[df['workingday'] == "holiday_weekend", 'cour
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 ?

In [89]:

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

Out[89]:

holiday	count	mean	std	min	25%	50%	75%	max
holiday	311.0	185.877814	168.300531	1.0	38.5	133.0	308.0	712.0
non_holiday	10575.0	191.741655	181.513131	1.0	43.0	145.0	283.0	977.0

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.

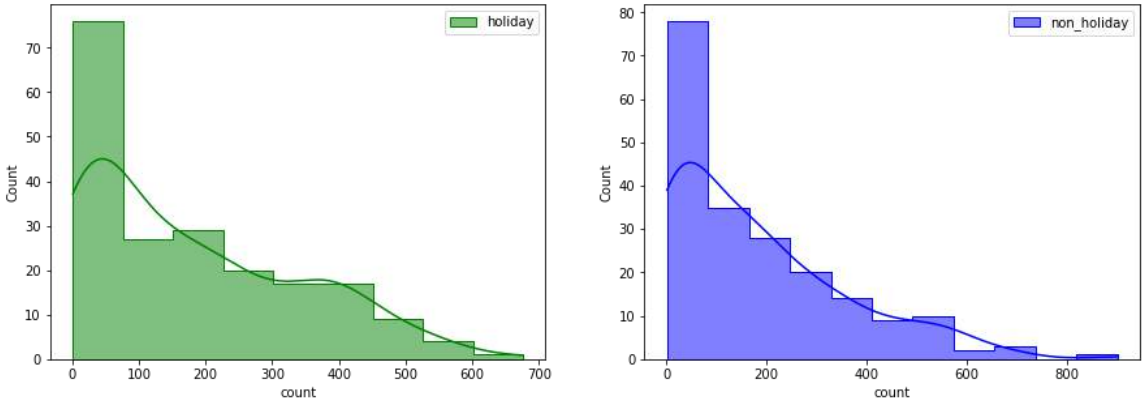
- Based on p-value, we will accept or reject H0.

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

Visual Tests to know if the samples follow normal distribution

In [90]:

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

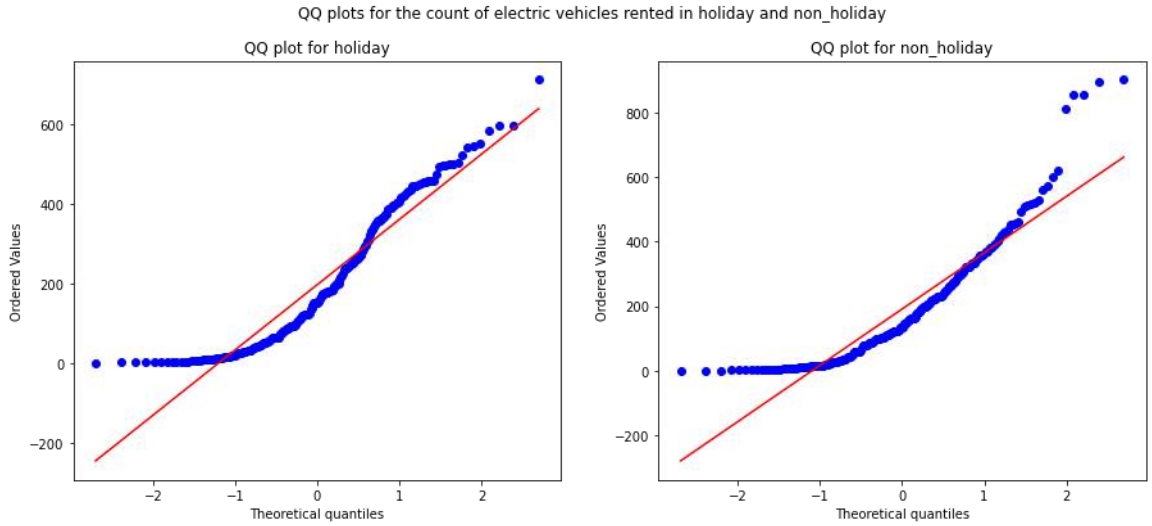


It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

In [91]:

```
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_holid
spy.probplot(df.loc[df['holiday'] == "holiday", 'count'].sample(200), plot = plt, dist =
plt.title('QQ plot for holiday')
plt.subplot(1, 2, 2)
spy.probplot(df.loc[df['holiday'] == "non_holiday", 'count'].sample(200), plot = plt, dis
plt.title('QQ plot for non_holiday')
plt.plot();
```



Distribution check using Boxcox transformation

In [92]:

```
transformed_weather1 = spy.boxcox(df.loc[df['holiday'] == "holiday", 'count'].sample(300)
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 3.5944790965913853e-07
The sample does not follow normal distribution
```

In [93]:

```
transformed_weather1 = spy.boxcox(df.loc[df['holiday'] == "non_holiday", 'count'].sample(
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 1.0880002884877634e-25
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test

In [94]:

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['holiday'] == "holiday", 'count'].sample(200),
                                df.loc[df['holiday'] == "non_holiday", 'count'].sample(20
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 0.4436624448344251
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.

In [95]:

```
# 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'] == "holiday", 'count'].sample(
                                      df.loc[df['holiday'] == "non_holiday", 'count'].sam
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 : 0.727729705836152
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 ?

In [96]:

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

Out[96]:

	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

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

- The data in the cells should be **frequencies**, or **counts** of cases.
- 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.
- There are 2 variables, and both are measured as **categories**.
- 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

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 [97]:

```
# First, finding the contingency table such that each value is the total number of total
  # 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
```

Out[97]:

weather	1	2	3	4
season				
fall	470116.0	139386.0	31160.0	0.0
spring	223009.0	76406.0	12919.0	164.0
summer	426350.0	134177.0	27755.0	0.0
winter	356588.0	157191.0	30255.0	0.0

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 [98]:

```
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
```

Out[98]:

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

In [99]:

```
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

In [100]:

```
if p_value < 0.05:
    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 ?

In [101]:

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

Out[101]:

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

In [102]:

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

Out[102]:

```
(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.

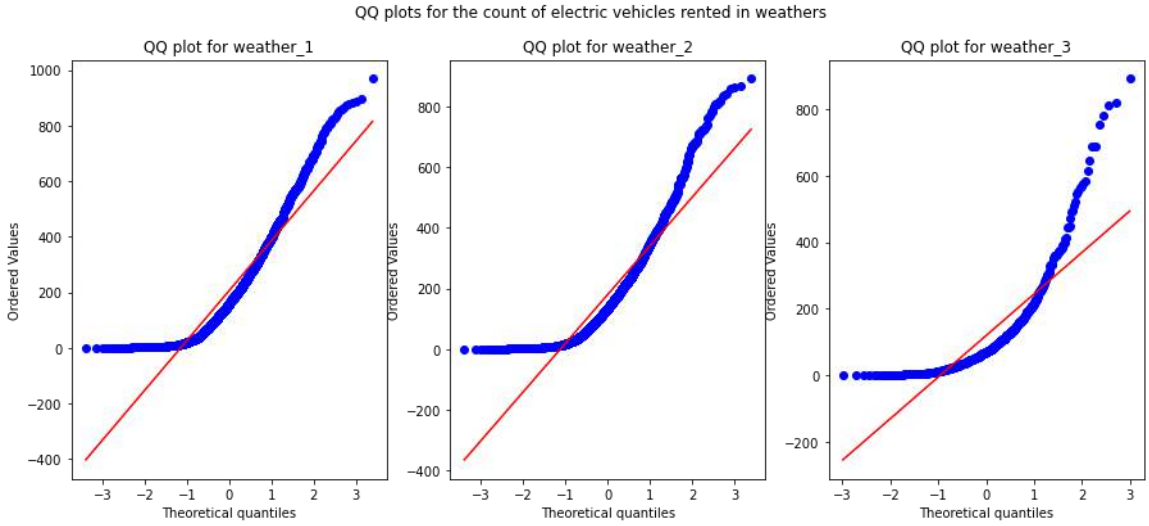
Based on p-value, we will accept or reject H0.

- p-val > alpha : Accept H0

Visual Tests to know if the samples follow normal distribution

In [103]:

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 3, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in weathers')
spy.probplot( df.loc[df['weather'] == 1, 'count'].sample(2000), plot = plt, dist = 'norm'
plt.title('QQ plot for weather_1')
plt.subplot(1, 3, 2)
spy.probplot( df.loc[df['weather'] == 2, 'count'].sample(2000), plot = plt, dist = 'norm'
plt.title('QQ plot for weather_2')
plt.subplot(1, 3, 3)
spy.probplot( df.loc[df['weather'] == 3, 'count'].sample(500), plot = plt, dist = 'norm')
plt.title('QQ plot for weather_3')
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** H0: The sample follows normal distribution H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [104]:

```
test_stat, p_value = spy.shapiro( df.loc[df['weather'] == 1, 'count'])
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 [105]:

```
test_stat, p_value = spy.shapiro( df.loc[df['weather'] == 2, 'count'])
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 9.781063280987223e-43
The sample does not follow normal distribution
```

In [106]:

```
test_stat, p_value = spy.shapiro( df.loc[df['weather'] == 3, 'count'])
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 3.876134581802921e-33
The sample does not follow normal distribution
```

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

In [107]:

```
transformed_weather1 = spy.boxcox(df.loc[df['weather'] == 1, '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 2.4357707859501747e-27
The sample does not follow normal distribution
```

In [108]:

```
transformed_weather1 = spy.boxcox(df.loc[df['weather'] == 2, 'count'].sample(2000))[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 1.1491517059688985e-15
The sample does not follow normal distribution
```

In [109]:

```
transformed_weather1 = spy.boxcox(df.loc[df['weather'] == 3, 'count'].sample(500))[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 0.00010053843288915232
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

In [110]:

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['weather'] == 1, 'count'].sample(500),
                                df.loc[df['weather'] == 2, 'count'].sample(500),
                                df.loc[df['weather'] == 3, '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 6.527206108600063e-13
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.

In [111]:

```
# Ho : all weathers have same variance
# Ha : one of the weather has different variance
# Assuming significance Level to be 0.05
alpha = 0.05
test_stat, p_value = spy.kruskal(df.loc[df['weather'] == 1, 'count'],
                                 df.loc[df['weather'] == 2, 'count'],
                                 df.loc[df['weather'] == 3, 'count'])
print('Test Statistic =', test_stat)
print('p value =', p_value)
```

```
Test Statistic = 204.95566833068537
p value = 3.122066178659941e-45
```

Comparing p value with significance level

In [112]:

```
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 weathers.

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

In [113]:

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

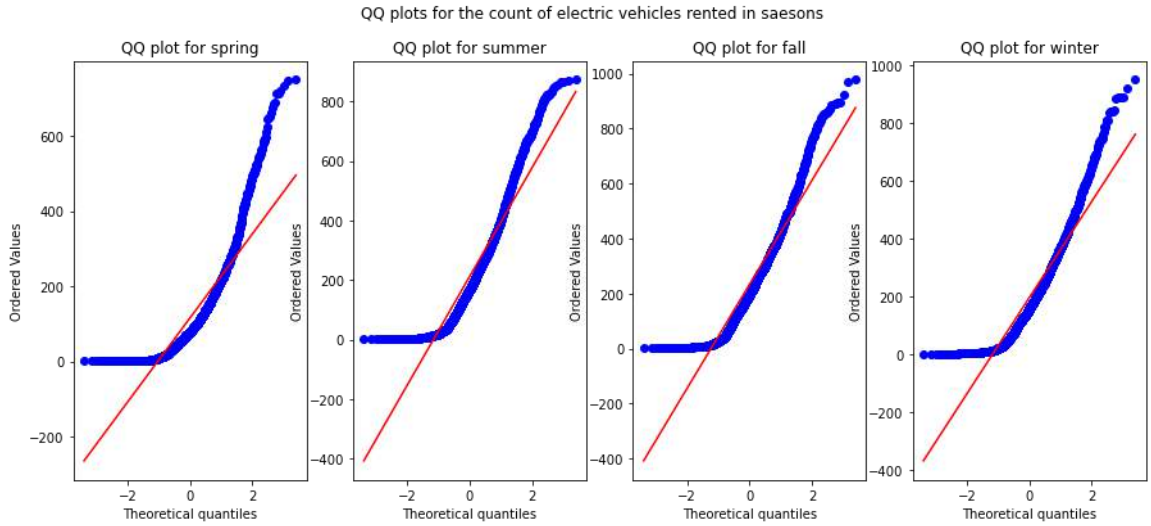
Out[113]:

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

Visual Tests to know if the samples follow normal distribution

In [114]:

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 4, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in saesons')
spy.probplot(df.loc[df['season'] =="spring" , 'count'].sample(2000), plot = plt, dist = '
plt.title('QQ plot for spring')
plt.subplot(1, 4, 2)
spy.probplot(df.loc[df['season'] == "summer", 'count'].sample(2000), plot = plt, dist = '
plt.title('QQ plot for summer')
plt.subplot(1, 4, 3)
plt.suptitle('QQ plots for the count of electric vehicles rented in saesons')
spy.probplot(df.loc[df['season'] =="fall" , 'count'].sample(2000), plot = plt, dist = 'no
plt.title('QQ plot for fall')
plt.subplot(1, 4, 4)
plt.suptitle('QQ plots for the count of electric vehicles rented in saesons')
spy.probplot(df.loc[df['season'] =="winter" , 'count'].sample(2000), plot = plt, dist = '
plt.title('QQ plot for winter')
plt.plot();
```



STEP-1 : Set up Null Hypothesis

- **Null Hypothesis (H0)** - Mean of cycle rented per hour is same for season summer, spring,fall and winter
- **Alternate Hypothesis (HA)** -Mean of cycle rented per hour is different for season summer, spring,fall and winter.

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

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.

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

- **Applying Shapiro-Wilk test for normality**

H0: The sample follows normal distribution

H1: The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

In [115]:

```
test_stat, p_value = spy.shapiro(df.loc[df['season'] =="spring" , 'count'].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 [116]:

```
test_stat, p_value = spy.shapiro(df.loc[df['season'] =="summer" , 'count'].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 4.684292456149985e-38
The sample does not follow normal distribution
```

In [117]:

```
test_stat, p_value = spy.shapiro(df.loc[df['season'] =="fall" , 'count'].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 2.2822888628561857e-35
The sample does not follow normal distribution
```

In [118]:

```
test_stat, p_value = spy.shapiro(df.loc[df['season'] =="winter" , 'count'].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 5.571230866679121e-38
The sample does not follow normal distribution
```

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

In [122]:

```
transformed_df_season_spring = spy.boxcox(df.loc[df['season'] =="spring" , 'count'].sampl
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 4.9370109424104946e-17
The sample does not follow normal distribution
```

In [124]:

```
transformed_df_season_summer = spy.boxcox(df.loc[df['season'] =="summer" , 'count'].sampl
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 8.212628784402924e-21
The sample does not follow normal distribution
```

In [125]:

```
transformed_df_season_fall = spy.boxcox(df.loc[df['season'] =="fall" , 'count'].sample(25
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 4.035976979049962e-21
The sample does not follow normal distribution
```

In [126]:

```
transformed_df_season_winter = spy.boxcox(df.loc[df['season'] =="winter" , 'count'].sampl
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 1.3393305227910005e-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

In [127]:

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['season'] =="winter" , 'count'].sample(2500),
                                df.loc[df['season'] =="summer" , 'count'].sample(2500),
                                df.loc[df['season'] =="fall" , 'count'].sample(2500),
                                df.loc[df['season'] =="spring" , 'count'].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 9.039513296289134e-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.

In [128]:

```
# Ho : Mean no. of cycles rented is same for different saeson
# Ha : Mean no. of cycles rented is different for different season
# Assuming significance Level to be 0.05
alpha = 0.05
test_stat, p_value = spy.kruskal(df.loc[df['season'] =="summer" , 'count'],
                                 df.loc[df['season'] =="spring" , 'count'],
                                 df.loc[df['season'] =="fall" , 'count'],
                                 df.loc[df['season'] =="winter" , 'count'])
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

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

In [129]:

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

```
Reject Null Hypothesis
```

Recommendations and Insights

- **There are 4 categorical features namely season, holiday, workingday, weather 7 numerical/continuous features and 1 datetime object. In total 12 independant features with 10886 rows.**
- **No missing data or null values present neither any duplicate row is there.**
- **After dealing with the outliers, total of 300 rows are removed out off 10886 from the dataset, As we can see from the above scatterplot, the data now looks more clean.**
- **Highest booking is in the month of june**
- **Almost same booking for the month of may,july,august,september,octomber and gradually decreasing for the rest of the month.**
- **The count is less during the cold months (November, December, January and February), where due to cold people prefer not to ride cycle**
- **As we can see from the monthwise bar plot, the demand for the bikes at the starting of the month is quite low as compared to the months from march 2012 onwards. There's a drop in the middle owing to cold and winter season**
- **Almost every months has the same number of bookings**
- **There are outliers in the windspeed and casual users which tells us that, the windspeed is not uniform**
- **The exponential decay curve for the count tells us that, as the users renting bikes increases the frequency decreases.**
- **For the weather (Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog) number of users renting the bikesis much low and hence it;s good to drop the feature while doing the further tests.**
- **When weather is good (Clear, Few clouds, partly cloudy, partly cloudy) people tend to rent more bikes**
- **Count of renting the bikes on working day is much higher than non-working day**
- **During Holidays people dont prefer to ride bikes**
- **During season (spring,summer,fall,winter) the count of renting the bikes is more or less**
- **The Registered users has higher correlation as comapred to the casual user count**
- **The windspeed and season has very nearly zero positive correlation with the count which means, that means the windspeed and season didnt have any affect on the bike renting**
- **temp and atemp has moderate correlation with the count. People tend to go out on bright sunny day when the temp is normal whereas during the harsh condition such as during too hot or too cold there is a drop in the renting the bike**
- **When its holiday, user count is considerably low but when its working day user count is moderately high.**
    - **Since the samples of weather and seasons 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.**
- **Since the samples of working day and holidays 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.**
- **As we are getting nan values for both the test and P value where P-value<alpha(0.05) and we cant predict the value of p, we reject Ho, thus we can say that count of bikes differ with a change in weather**

In order to conclude, we can say that the major factors affecting the count of bikes rented are season and weather. The working and non working days can't be considered as a significant factor in predicting the future of rental business. At the same time, the business team must focus on the

months other than winter months for increasing the bike parking zones as during the winter months of (Nov, Dec, Jan, Feb), theres's a considerable dip in the count. So the team can utilize these months for serving some other purpose such as renting electric cars, etc which can be a comfortable means for commute in cold.