Business Case:Yulu----> Hypothesis Testing

About Yulu:

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India. Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting. Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!.

Business Problem

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.

Importing Important Libraries for data exploration and visualization

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import math
from scipy.stats import norm,binom,geom,poisson
from scipy.stats import ttest_1samp,ttest_ind

from scipy.stats import chisquare # statistcal test (chistat,pvalue)
from scipy.stats import chi2_contingency # when expected value has to be computed
from scipy.stats import chi2 #distribution

from scipy.stats import f_oneway,kruskal # Numerical versus categorical for many categ
from scipy.stats import shapiro # test Gaussisn (50-200 samples)
from scipy.stats import levene  # test variance
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import spearmanr,pearsonr,ttest_rel

import warnings
warnings.filterwarnings("ignore")
```

UPload and read csv file in pandas Dataframe

In [2]:
```
df=pd.read_csv("Yulu_bike_sharing 2.csv",sep=",")
```

Inspecting Dataset and Analyzing different metrics:-

In [3]:
```
df.head()
```

Out[3]:

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

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

Out[4]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4

Column Profiling:-

datetime: datetime

season: season (1: spring, 2: summer, 3: fall, 4: winter)

holiday: whether day is a holiday or not.

working day: 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.

observation On

shape of data

size of data

statistical summary

In [5]:
```
df.shape
```

Out[5]:
```
(10886, 12)
```

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

Out[6]:
```
130632
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [8]:
```
df.dtypes
```

Out[8]:
```
datetime       object
season          int64
holiday         int64
workingday      int64
weather         int64
temp          float64
atemp         float64
humidity        int64
windspeed     float64
casual          int64
registered      int64
count           int64
dtype: object
```

In [9]:
```
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
```

In [10]:
```
df.describe()
```

Out[10]:

	season	holiday	workingday	weather	temp	atemp	humidi
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.0000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.8864
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.2450
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.0000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.0000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.0000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.0000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.0000

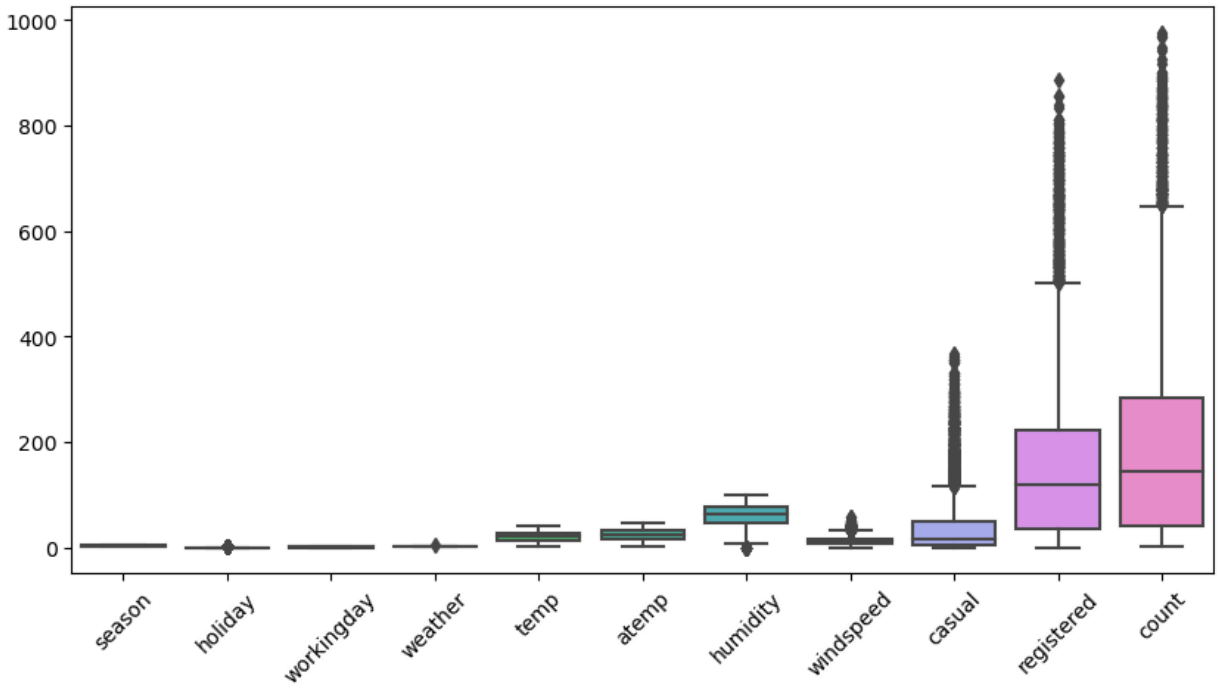
In [11]:
```
df.describe(include="object")
```

Out[11]:

	datetime
count	10886
unique	10886
top	2011-01-01 00:00:00
freq	1

Checking Outliers in the data.

In [12]:
```
plt.figure(figsize=(10,5))
sns.boxplot(df)
plt.xticks(rotation=45)
plt.show()
```



Comments

outliers are more for the columns casual,registered and count.

Data cleaning:

Cleaning for the missing values and Nulls

In [13]:
```
df.isnull().sum().sort_values()
```

Out[13]:
```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

In [14]:
```
df[df.duplicated()]
```

Out[14]:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	regis

Comments

No missing values and duplicates are present in data.

Changing the datatypes of Attributes

Datetime to numeric

season to categorical

holiday to categorical

Workingday to categorical

weather to categorical

In [15]:
```
# changing Datetime to datetime:
df["datetime"]=pd.to_datetime(df["datetime"])
```

In [16]:
```
# changing season to categorical:
df["season"]=df["season"].astype("object")
```

In [17]:
```
# changing holiday to categorical:
df["holiday"]=df["holiday"].astype("object")
```

In [18]:
```
# changing worKingday to categorical:
df["workingday"]=df["workingday"].astype("object")
```

In [19]:
```
# changing weather to categorical:
df["weather"]=df["weather"].astype("object")
```

In [20]:
```
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  object
 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(4), object(4)
memory usage: 1020.7+ KB
```

Extract hour,month and Year:

In [21]:
```
df["hour"]=df["datetime"].dt.hour
```

In [22]:
```
df["month"]=df["datetime"].dt.month
```

In [23]:
```
df["year"]=df["datetime"].dt.year
```

In [24]:
```
df.head()
```

Out[24]:

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

Non Graphical Analysis:-

In [25]:
```
# season wise unique values and count
df["season"].value_counts()
```

Out[25]:
```
season
4    2734
2    2733
3    2733
1    2686
Name: count, dtype: int64
```

In [26]:
```
df["season"].value_counts(normalize=True).round(4)*100
```

Out[26]:
```
season
4    25.11
2    25.11
3    25.11
1    24.67
Name: proportion, dtype: float64
```

Comments

we have four types of seasons namely 1(spring), 2(summer), 3(fall), 4(winter)
which are having nearly same count in the given dataset.

In [27]:
```
# Holiday wise unique values and count:-
df["holiday"].value_counts()
```

Out[27]:
```
holiday
0    10575
1      311
Name: count, dtype: int64
```

In [28]:
```
df["holiday"].value_counts(normalize=True).round(2)*100
```

Out[28]:
```
holiday
0    97.0
1     3.0
Name: proportion, dtype: float64
```

Comment:-

We have 311 holidays(i.e 3%) in whole dataset.

In [29]:
```
# Workingday wise unique values and count:-
df["workingday"].value_counts()
```

Out[29]:
```
workingday
1    7412
0    3474
Name: count, dtype: int64
```

In [30]:
```
df["workingday"].value_counts(normalize=True).round(2)*100
```

Out[30]:
```
workingday
1    68.0
0    32.0
Name: proportion, dtype: float64
```

Comments:-

After Analyzing whole dataset,total weekday is 68% and 32% is weekend and holiday.

In [31]:
```
# weatherwise unique value and count
df["weather"].value_counts()
```

Out[31]:
```
weather
1    7192
2    2834
3     859
4       1
Name: count, dtype: int64
```

In [32]:
```
# percentage of time a particular weather condition prevail in India.
df["weather"].value_counts(normalize=True).round(2)*100
```

Out[32]:
```
weather
1    66.0
2    26.0
3     8.0
4     0.0
Name: proportion, dtype: float64
```

Comment

In the given dataset weather has been classified into 4 types:

1 represents Clear, Few clouds, partly cloudy, partly cloudy = 66% (max)

2 represents Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist = 26% (medium)

3 represents Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds = 8% (less)

4 represents Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog = 0.01% (very low)

In [33]:
```
# tempwise unique values and count
df["temp"].value_counts().head()
```

Out[33]:
```
temp
14.76    467
26.24    453
28.70    427
13.94    413
18.86    406
Name: count, dtype: int64
```

In [34]:
```
df["temp"].max()
```

Out[34]:
```
41.0
```

In [35]:
```
df["temp"].min()
```

Out[35]:
```
0.82
```

Comment

After analyzing dataset it was observed that temperature range is from .82 to 41.0

In [36]:
```
# atemp wise unique values and count
df["atemp"].value_counts().head()
```

Out[36]:
```
atemp
31.060    671
25.760    423
22.725    406
20.455    400
26.515    395
Name: count, dtype: int64
```

In [37]:
```
df["atemp"].max()
```

Out[37]:
```
45.455
```

In [38]:
```
df["atemp"].min()
```

Out[38]:
```
0.76
```

Comment :-

Range of atemp observed from the data is .76 to 45.455

In [39]:
```
# humidity unique value and count
df["humidity"].value_counts().head()
```

Out[39]:
```
humidity
88    368
94    324
83    316
87    289
70    259
Name: count, dtype: int64
```

In [40]:
```
# Range of humidity:-
print("maxm_humidity:",df["humidity"].max(),"\n","minm_humidity:",df["humidity"].min()
```

```
maxm_humidity: 100
 minm_humidity: 0
```

Comments :

we are having humidity data ranging from 0 to 100

In [41]:
```
# Range of windspeed:-
print("Min:",df["windspeed"].min(),"\n","Max:",df["windspeed"].max())
```

```
Min: 0.0
 Max: 56.9969
```

Comment :

speed Range of wind blowing is 0.0 to 56.96

In [42]:
```
# Range of values present in casual attributes:
print("min:",df["casual"].min(),"\n","max:",df["casual"].max())
```

```
min: 0
 max: 367
```

Comments:-

Range of casual user is 0 to 367.

In [43]:
```
# Range of values present in registered attributes:
print("min:",df["registered"].min(),"\n","max:",df["registered"].max())
```

```
min: 0
 max: 886
```

Comment

Range of registered user is from 0 to 886

In [44]:
```
# Range of values present in count attributes:
print("min:",df["count"].min(),"\n","max:",df["count"].max())
```

```
min: 1
 max: 977
```

Comment:-

Range of count of total rental bike both for casual and registered user is from 1 to 977

In [45]:
```
# hourwise unique values and count
df["hour"].value_counts()
```

Out[45]:
```
hour
12    456
13    456
22    456
21    456
20    456
19    456
18    456
17    456
16    456
15    456
14    456
23    456
11    455
10    455
9     455
8     455
7     455
6     455
0     455
1     454
5     452
2     448
4     442
3     433
Name: count, dtype: int64
```

Comments

we have almost equal no of counts for everyhour ranging from 0 to 23hrs

In [46]:
```
# monthwise values and count
df["month"].value_counts()
```

Out[46]:
```
month
5     912
6     912
7     912
8     912
12    912
10    911
11    911
4     909
9     909
2     901
3     901
1     884
Name: count, dtype: int64
```

Comment :

Almost every month have the same count.

In [47]:
```
# Yearwise unique values and count
df["year"].value_counts()
```

Out[47]:
```
year
2012    5464
2011    5422
Name: count, dtype: int64
```

Comment:-

We have data of year 2012 and 2011

In [48]:
```
# calculating total no of registered user:
df1=df["registered"].sum()
df2=df["casual"].sum()
percentage_of_registered_user=df1/(df1+df2).round(3)*100
print("percentage_of_registered_user:",percentage_of_registered_user)
```

```
percentage_of_registered_user: 81.1968586548107
```

In [49]:
```
# Mapping all values of season with their respective representatives.
season_mapping={1:"spring",2:"summer",3:"fall",4:"winter"}
# replace values in the season
df["season"]=df["season"].replace(season_mapping)
df.head()
```

Out[49]:

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

Encoding the real interpretations of the value in season attributes.

In [57]:
```
# setting the 'datetime' column as the index of the DataFrame 'df'
df.set_index('datetime', inplace = True)
# Setting 'datetime' column as the index allows us easier analysis and resampling.
```

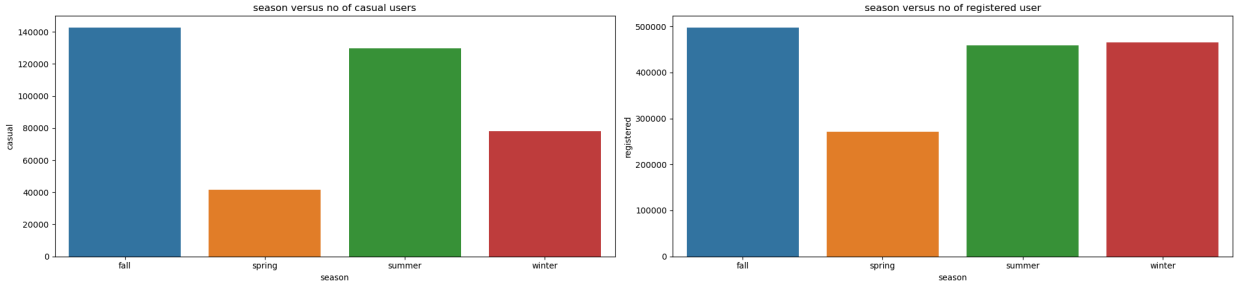
In [58]:
```
df.head()
```

Out[58]:

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registe
2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	0.0	3	
2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	0.0	8	
2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	0.0	5	
2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	0.0	3	
2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	0.0	0	

Visual Analysis:

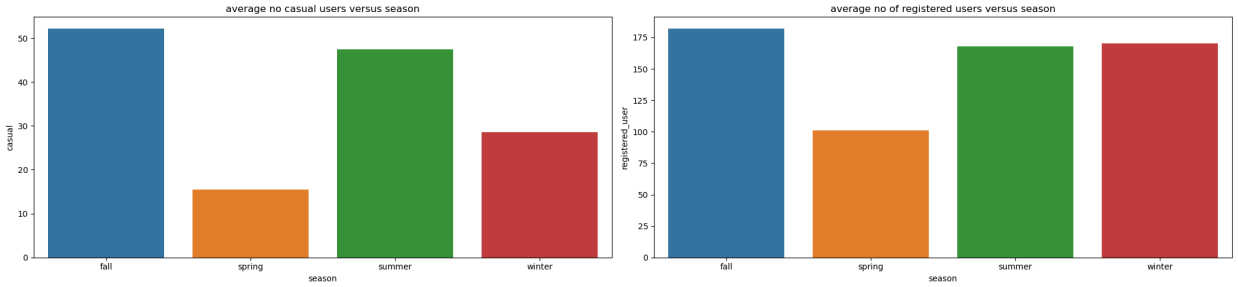
In [50]:
```
# subplot for no of  causal and registered user per season:-
df1=df.groupby("season")["casual"].sum().reset_index()
df2=df.groupby("season")["registered"].sum().reset_index()
# chart 1:- No of causal user in each season
plt.figure(figsize=(21,5))
plt.subplot(1,2,1)
sns.barplot(data=df1,x="season",y="casual")
plt.xlabel("season")
plt.ylabel("casual")
plt.title("season versus no of casual users")

# chart 2:-
plt.subplot(1,2,2)
sns.barplot(data=df2,x="season",y="registered")
plt.xlabel("season")
plt.ylabel("registered")
plt.title("season versus no of registered user")

# To prevent overlap
plt.tight_layout()
plt.show()
```



1.From the graph it is observed that no of registered users are more compaired to casual user in all the four seasons.

2. And maximum users are renting bikes in fall season followed by summer and winter.

In [51]:
```
# subplot for average no of registered and casual user per season
df1=df.groupby("season")["casual"].mean().reset_index()
df2=df.groupby("season")["registered"].mean().reset_index()
plt.figure(figsize=(21,5))

# chart1: plot for average no of user per season
plt.subplot(1,2,1)
sns.barplot(data=df1,x="season",y="casual")
plt.xlabel("season")
plt.ylabel("casual")
plt.title("average no casual users versus season")

# chart2: plot for average no of registered user per season
plt.subplot(1,2,2)
sns.barplot(data=df2,x="season",y="registered")
plt.xlabel("season")
plt.ylabel("registered_user")
plt.title("average no of registered users versus season")

# To prevent overlap
plt.tight_layout()
plt.show()
```



Trend of average number of user(both registered and casual) is same as total no users.

In [65]:
```
# To see the year on year growth of the average hourly count of bike rents.
# 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 down
 # 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
```

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

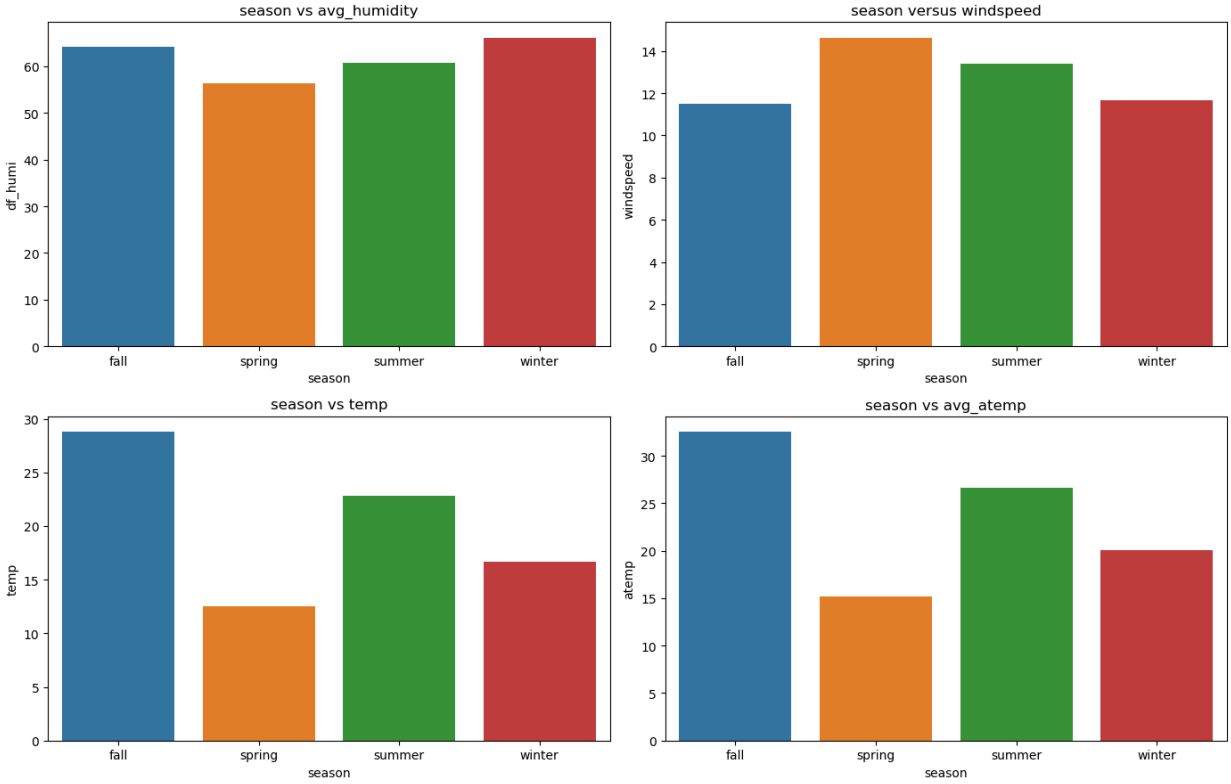
Out[65]:

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

From the graph it is clear that demand of the yulu rental bike has increased in 2012 compaired to 2011.

In [53]:

```
# To see the variation of mean humidity, temp, atemp and windspeed over different seas
df_humi=df.groupby("season")["humidity"].mean().reset_index()
df_temp=df.groupby("season")["temp"].mean().reset_index()
df_atemp=df.groupby("season")["atemp"].mean().reset_index()
df_windspeed=df.groupby("season")["windspeed"].mean().reset_index()

plt.figure(figsize=(14,9))
plt.subplot(2,2,1)
sns.barplot(data=df_humi,x="season",y="humidity")
plt.xlabel("season")
plt.ylabel("df_humi")
plt.title("season vs avg_humidity")

plt.subplot(2,2,3)
sns.barplot(data=df_temp,x="season",y="temp")
plt.xlabel("season")
plt.ylabel("temp")
plt.title("season vs temp")

plt.subplot(2,2,4)
sns.barplot(data=df_atemp,x="season",y="atemp")
plt.xlabel("season")
plt.ylabel("atemp")
plt.title("season vs avg_atemp")

plt.subplot(2,2,2)
sns.barplot(data=df_windspeed,x="season",y="windspeed")
plt.xlabel("season")
plt.ylabel("windspeed")
plt.title("season versus windspeed")

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



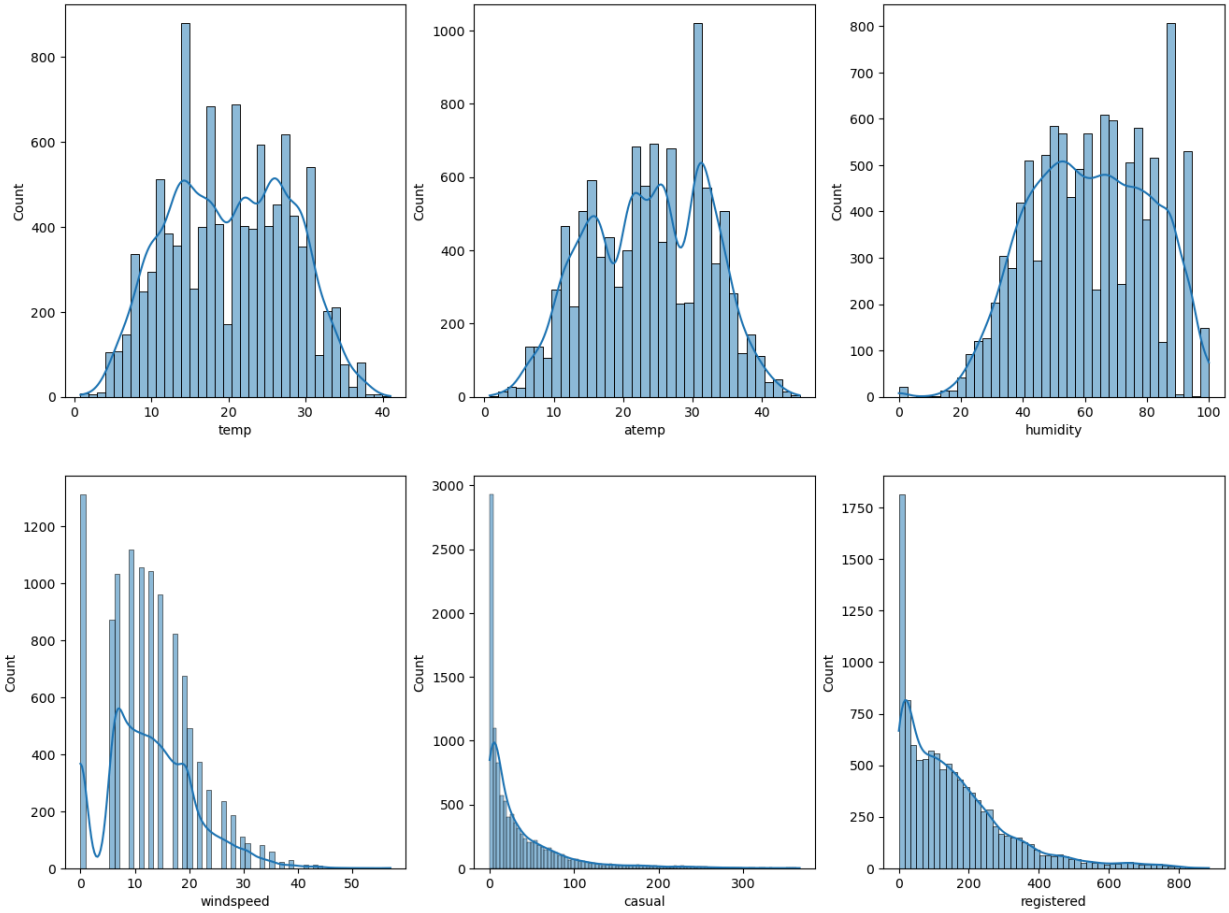
Spring season is having least temperature, least humidity and fastest windspeed.

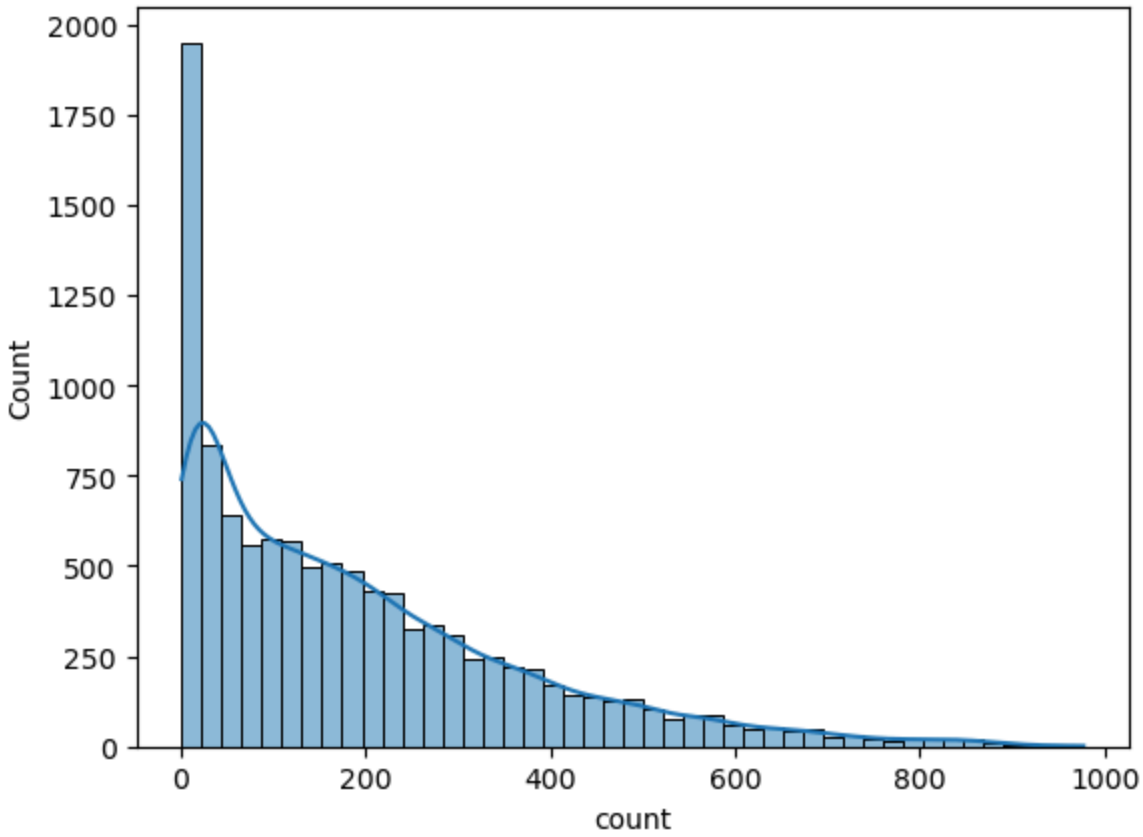
Fall is the season having having most average temperature and humidity.

Interesting to see summer season is having lesser temperature than fall season.

Winter is the most humid season as per the dataset.

In [54]:
```
# Analyzing the distribution for numerical variables:-
num_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(16,12))
index = 0
for row in range(2):
 for col in range(3):
     sns.histplot(df[num_cols[index]], ax=axis[row, col], kde=True)
     index += 1

plt.show()
# Extra 1 graph of Total is plotted outside of the subplots for clear visualization.
sns.histplot(df[num_cols[-1]], kde=True)
plt.show()
```





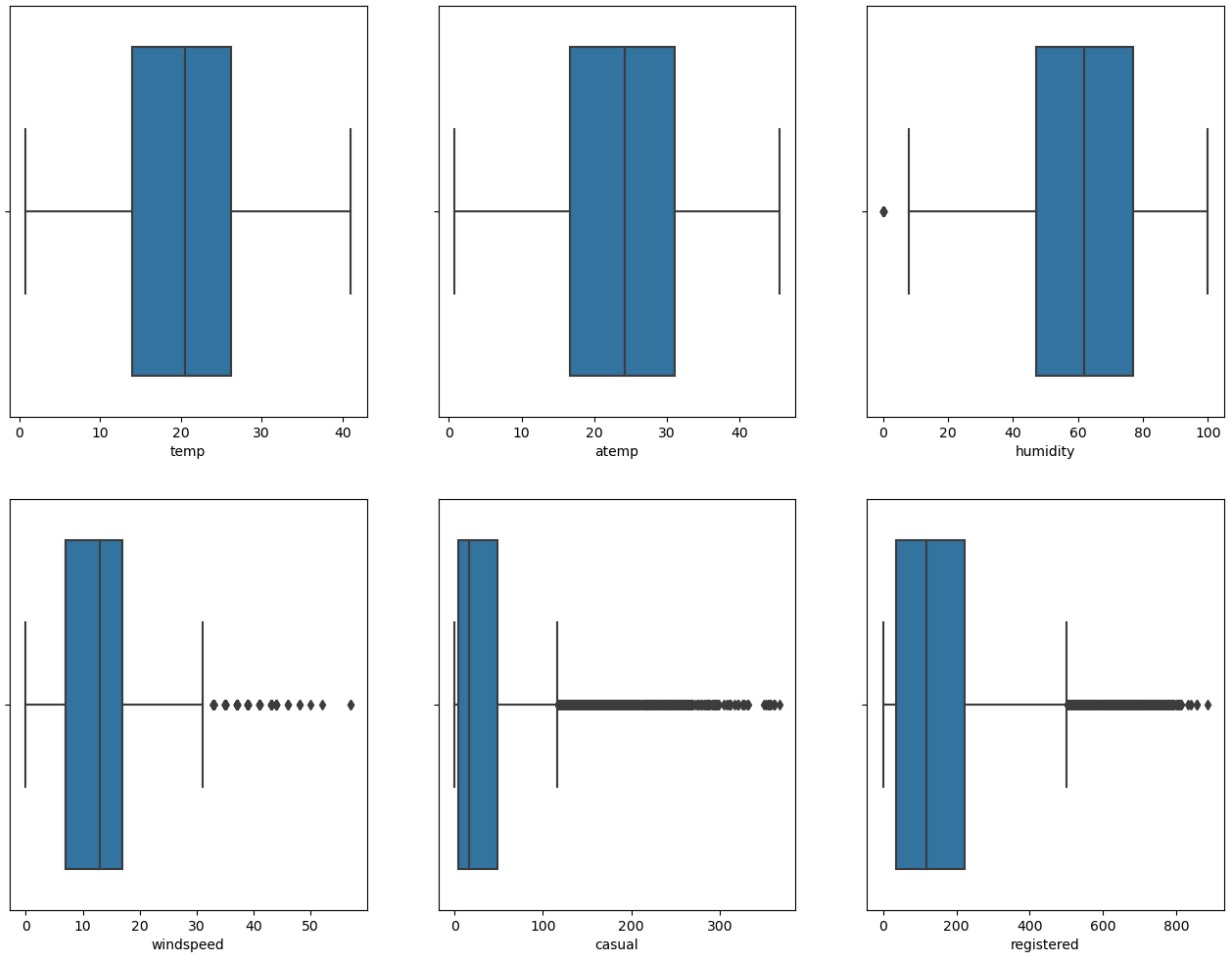
Casual, registered and count somewhat looks like Log Normal Distribution.

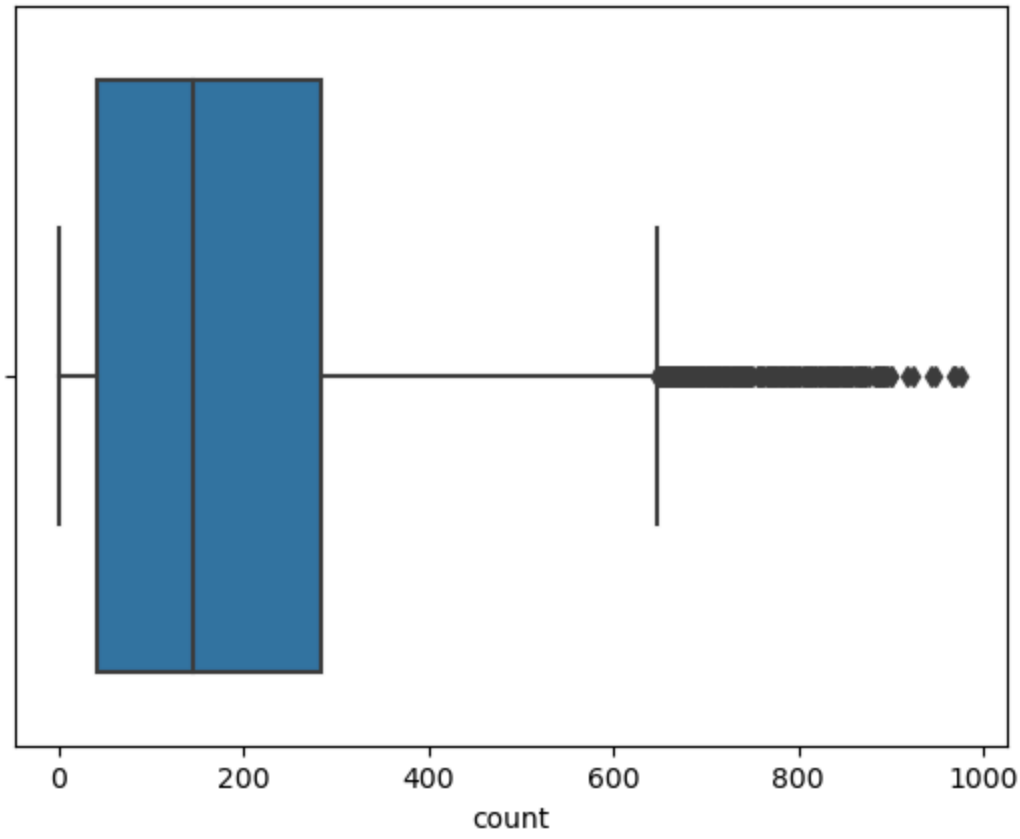
Temp, atemp and humidity looks like they follows the Normal Distribution.

Windspeed follows the binomial distribution.

Detecting outliers present in the different attributes of the dataset.

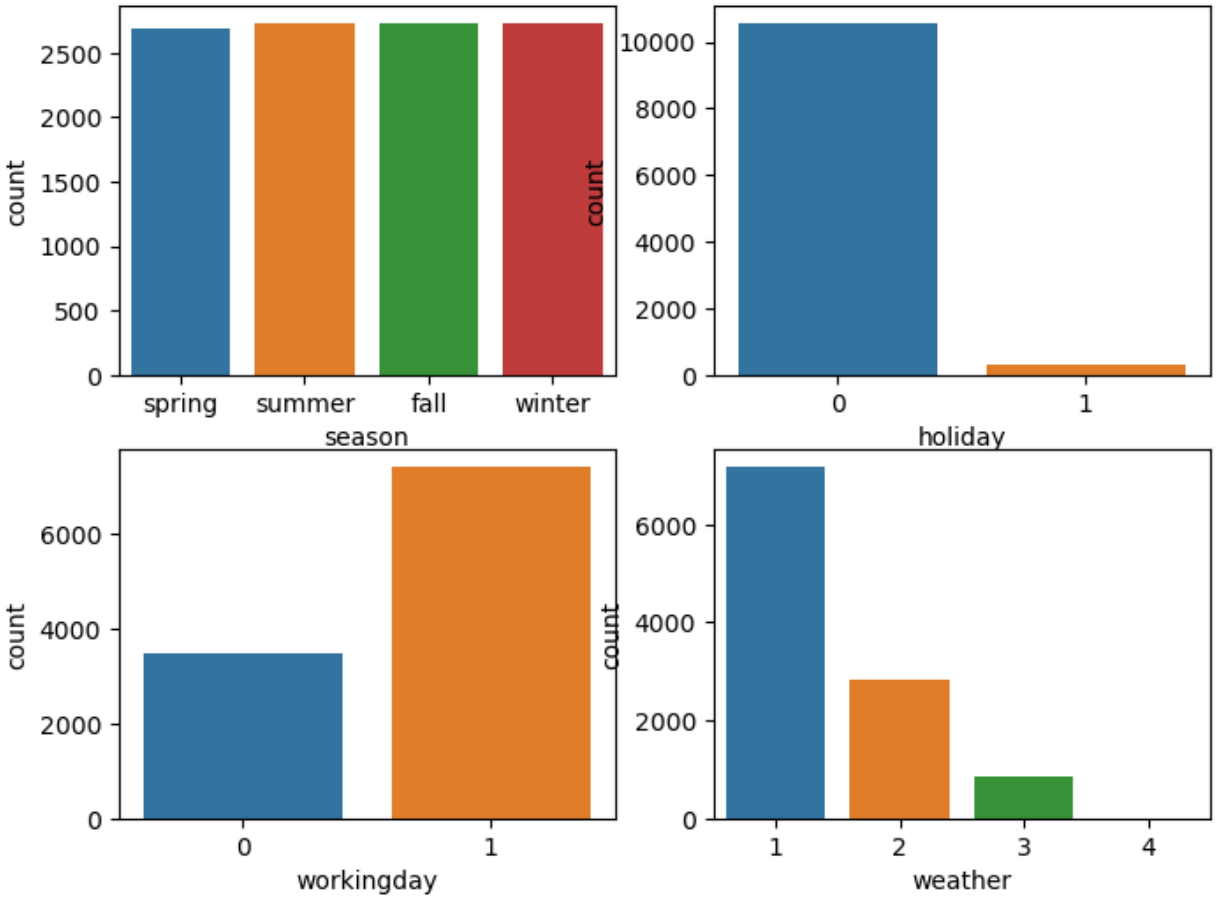
In [62]:
```
# outliers detection for numeric columns
num_col=["temp","atemp","humidity","windspeed","casual","registered","count"]
fig,axis=plt.subplots(nrows=2,ncols=3,figsize=(16,12))
index=0
for row in range(2):
    for col in range(3):
        sns.boxplot(x=df[num_col[index]],ax=axis[row,col])
        index+=1
plt.show()
# one extra graph is plotted outside for better clearity
sns.boxplot(x=df[num_col[-1]])
plt.show()
```





Except temp,atemp and humidity all other columns like (windspeed,casual,registered,count) is observed with too much of outliers

In [60]:
```
# countplot of each categorical column
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(8, 6))
cat_col=["season","holiday","workingday","weather"]
index = 0
for row in range(2):
    for col in range(2):
        sns.countplot(data=df, x=cat_col[index], ax=axis[row, col])
        index += 1
plt.show()
```



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

Bivariate Analysis:-

In [ ]:
```
# Boxplot for each categorical columns
cat_col=["season","holiday","workingday","weather"]
fig,axis=plt.subplots(nrows=2,ncols=2,figsize=(16,12))
index=0
for row in range(2):
    for col in range(2):
        sns.boxplot(data=df,x=cat_col[index],y="count",ax=axis[row,col])
        index+=1
plt.show()
```

During the seasons of summer and fall, we see more bikes being rented.

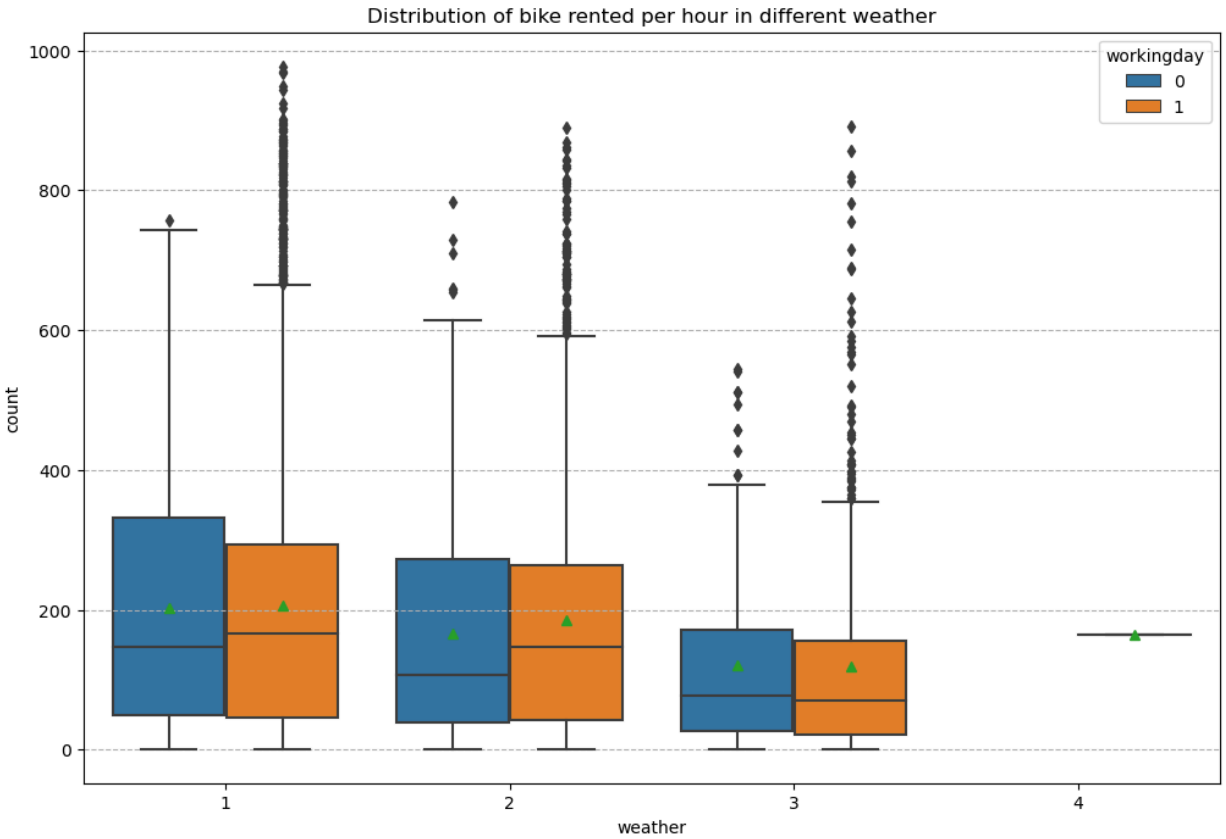
More bikes are rented on holidays.

Also non-working days have more bikes being rented.

Whenever there is rain, thunderstorm, snow or fog, less bikes were rented.

In [63]:
```
# Analysis of bike rented per hour in different season
plt.figure(figsize=(12,8))
plt.title("Distribution of bike rented per hour in different weather")
sns.boxplot(x="weather",y="count",hue="workingday",data=df,showmeans=True)
```

```
plt.grid(axis="y",linestyle="--")
plt.show()
```



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

In [ ]:
```
# plotting numerical variables against count using scatterplot
num_col=["temp","atemp","humidity","windspeed","casual","registered","count"]
fig,axis=plt.subplots(nrows=2,ncols=3,figsize=(12,8))
index=0
for row in range(2):
    for col in range(3):
        sns.scatterplot(x=num_col[index],y="count",data=df,ax=axis[row,col])
        index+=1
plt.show()
```

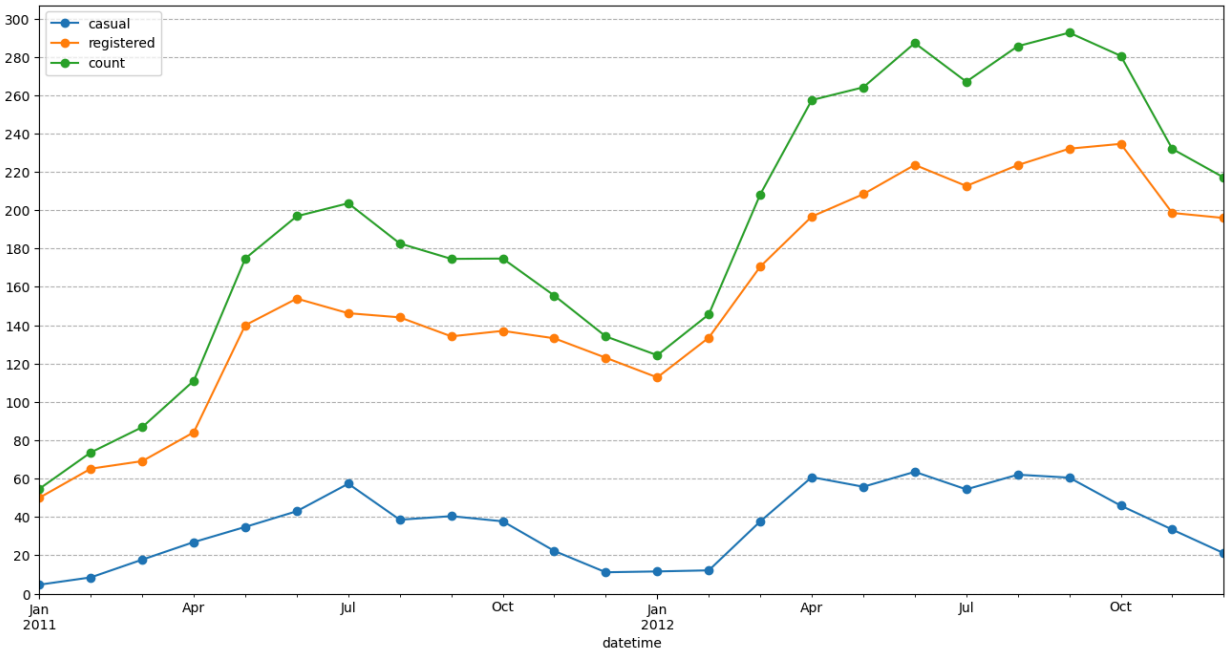
When humidity was less that 20, Bikes rented were very low.

When temp and atemp were less than 10 degree celcius, the no of bikes rented were very less.

When the windspeed was more that 35, no of bikes rented were less.

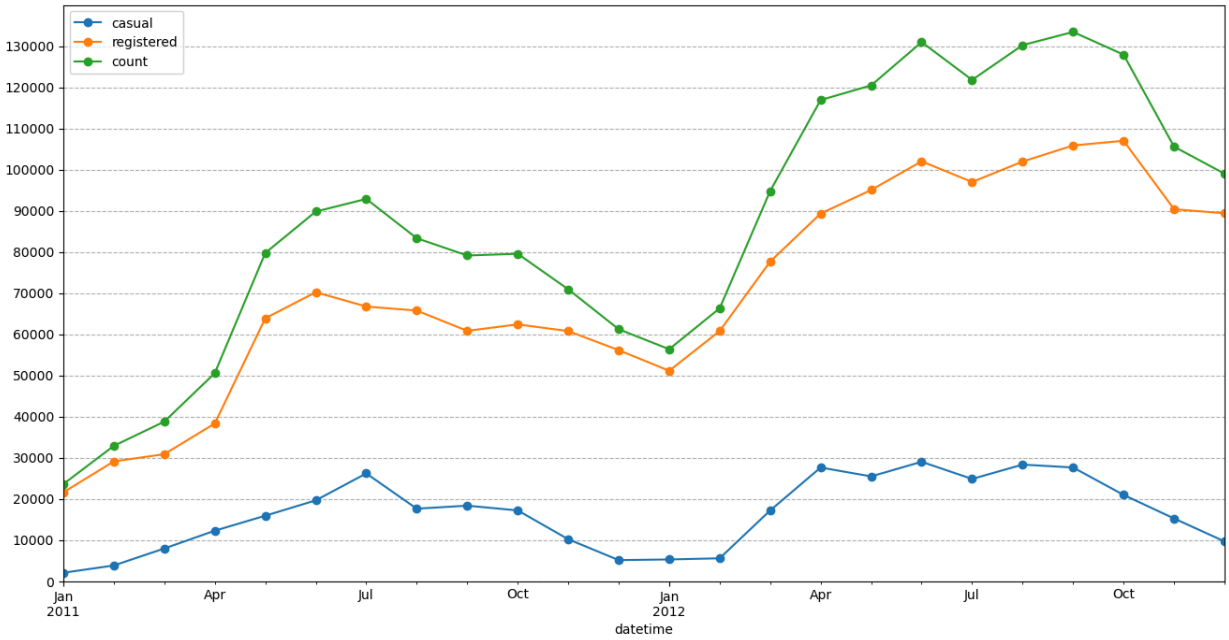
In [67]:
```
# The below code visualizes the trend of the monthly average values for the 'casual',
 # and 'total' variables.

plt.figure(figsize = (16, 8))
# plotting a lineplot by resampling the data on a monthly basis, and calculating the m
 # of 'casual', 'registered' and 'total' users for each month
df.resample('M')['casual'].mean().plot(kind = 'line', legend = 'casual', marker = 'o')
df.resample('M')['registered'].mean().plot(kind = 'line', legend = 'registered', marke
df.resample('M')['count'].mean().plot(kind = 'line', legend = 'total', 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()
```



Monthly average users suddenly raised from Dec 2011 and rise was significant upto March 2012.

In [68]:
```
# The below code visualizes the trend of the monthly total values for the 'casual', 'r
 # and 'count' variables, allowing for easy comparison and analysis of their patterns
plt.figure(figsize = (16, 8))
# plotting a lineplot by resampling the data on a monthly basis, and calculating the s
 # 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 = '--')
plt.yticks(np.arange(0, 130001, 10000))
plt.ylim(0,) # setting the lower y-axis limit to 0
plt.show()
```



From the graph we can observed that there is a bit slow down in the customer behaviour for rental bike service from December 2011 to jan 2012 but after jan 2012 there is sudden increase in demand of rental bike by registered user but the behaviour of casual user remains same till feb 2012.

In [77]:
```
# To see how average hourly count of rental bikes varies across different months.
df.reset_index(drop=True,inplace = True)
# Grouping the DataFrame by the month
df1 = df.groupby(by = df['datetime'].dt.month)['count'].mean().reset_index()
df1.rename(columns = {'datetime' : 'month'}, inplace = True)
# Create a new column 'prev_count' by shifting the 'count' column one position up
 # to compare the previous month's count with the current month's count
df1['prev_count'] = df1['count'].shift(1)
# Calculating the growth percentage of 'count' with respect to the 'count' of previous
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1.set_index('month', inplace = True)
df1
```

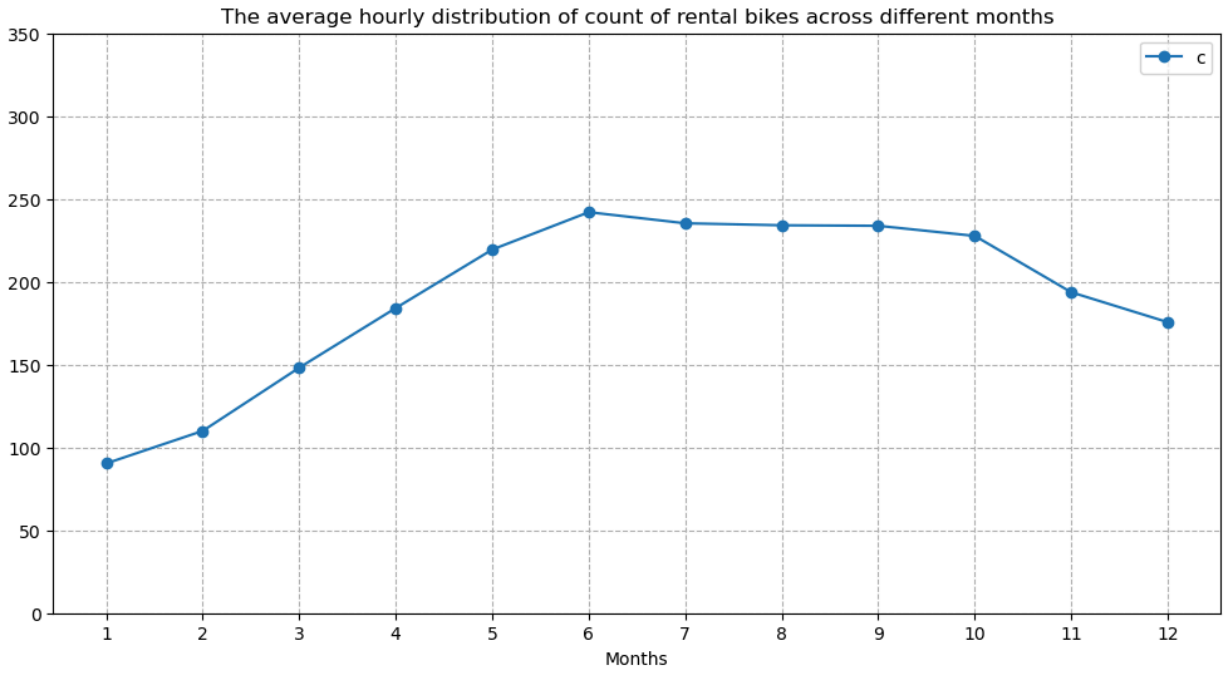
Out[77]:

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

Only in first 2 months of the year i.e. in Jan to March, the growth of average hourly count of rental bikes was positive and significant. In the months of March to May the growth was in decreasing phase. From June to december growth was negative.

In [78]:

```
# The resulting plot visualizes the average hourly distribution of the count of rental
 # 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 count of rental bikes across different m
# Grouping the DataFrame by the month and calculating the mean of the 'count' column f
 # 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 =
plt.xlabel('Months')
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 1
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.show()
```



In [ ]:
```
The average hourly count of rental bikes is the highest in the month of June followed

The average hourly count of rental bikes is the lowest in the month of January followe

Overall, these trends suggest a seasonal pattern in the count of rental bikes, with hi
summer months, a slight decline in the winter, and a further decrease in the spring se

It could be useful for the rental bike company to consider these patterns for resource
and operational planning throughout the year.
```

In [79]:
```
# 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
df1['growth_percent'] = (df1['count'] - df1['prev_count']) * 100 / df1['prev_count']
df1
```

Out[79]:

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

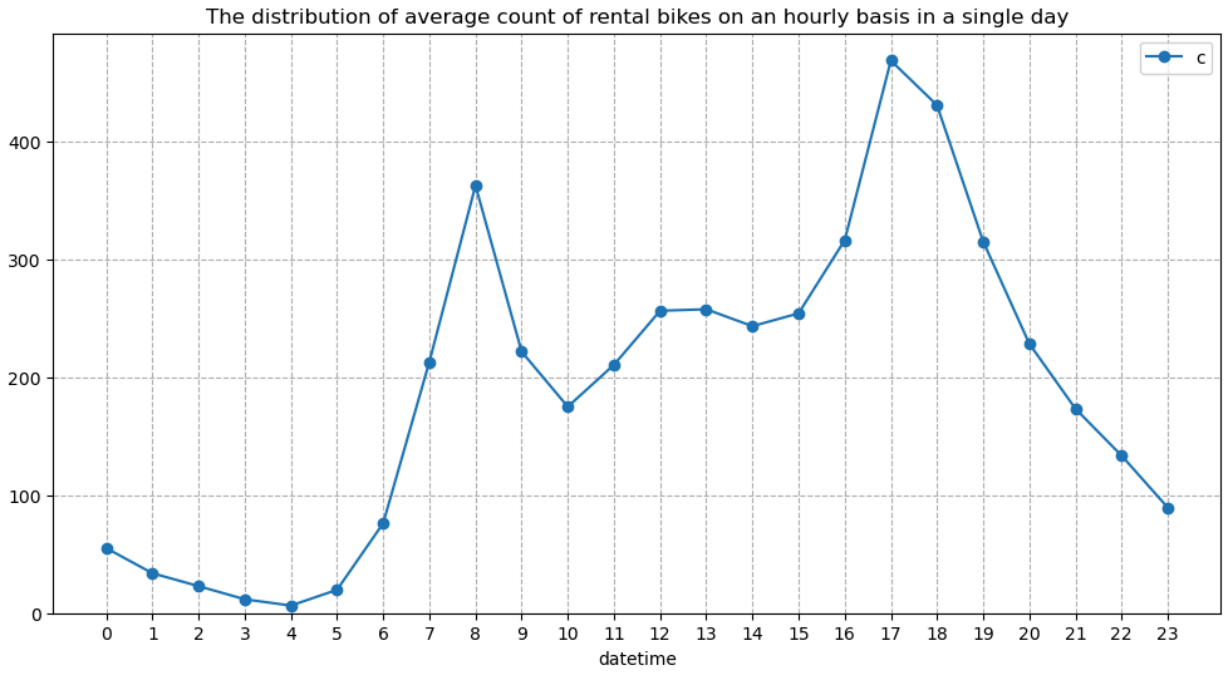
During the early morning hours (hours 0 to 5), there is a significant decrease in the count, with negative growth percentages ranging from -38.59% to -48.66% which is expected.

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 with slight fluctuations 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 [80]:
```
# plotting the above trend of hourly variation.
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on an hourly basis in a s
df.groupby(by = df['datetime'].dt.hour)['count'].mean().plot(kind = 'line', marker = '
plt.ylim(0,)
plt.xticks(np.arange(0, 24))
plt.legend('count')
plt.grid(axis = 'both', linestyle = '--')
plt.show()
```



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 night.

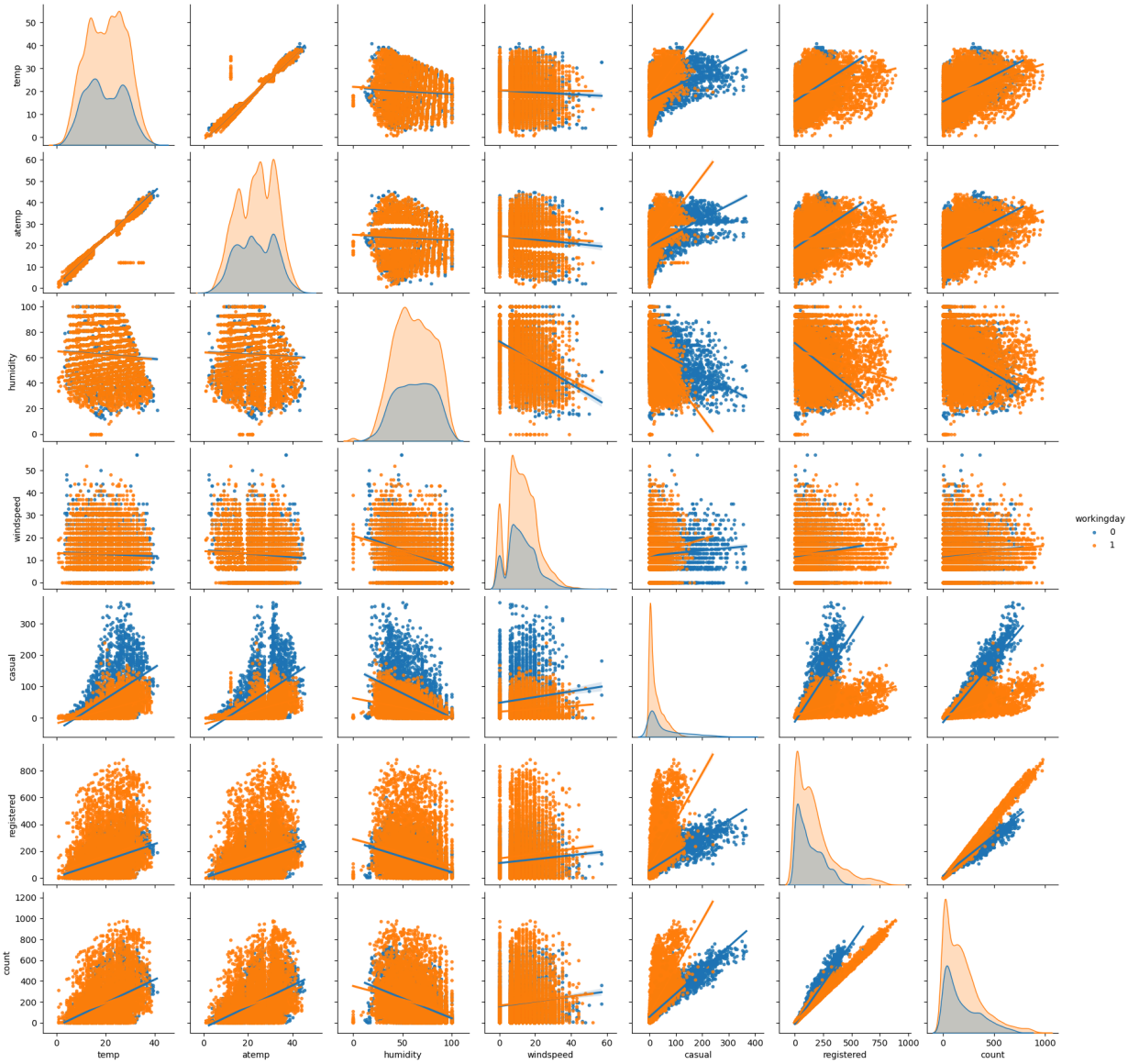
In [81]:
```
df.head()
```

Out[81]:

	level_0	index	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspe
0	0	0	2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	
1	1	1	2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	
2	2	2	2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	
3	3	3	2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	
4	4	4	2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	

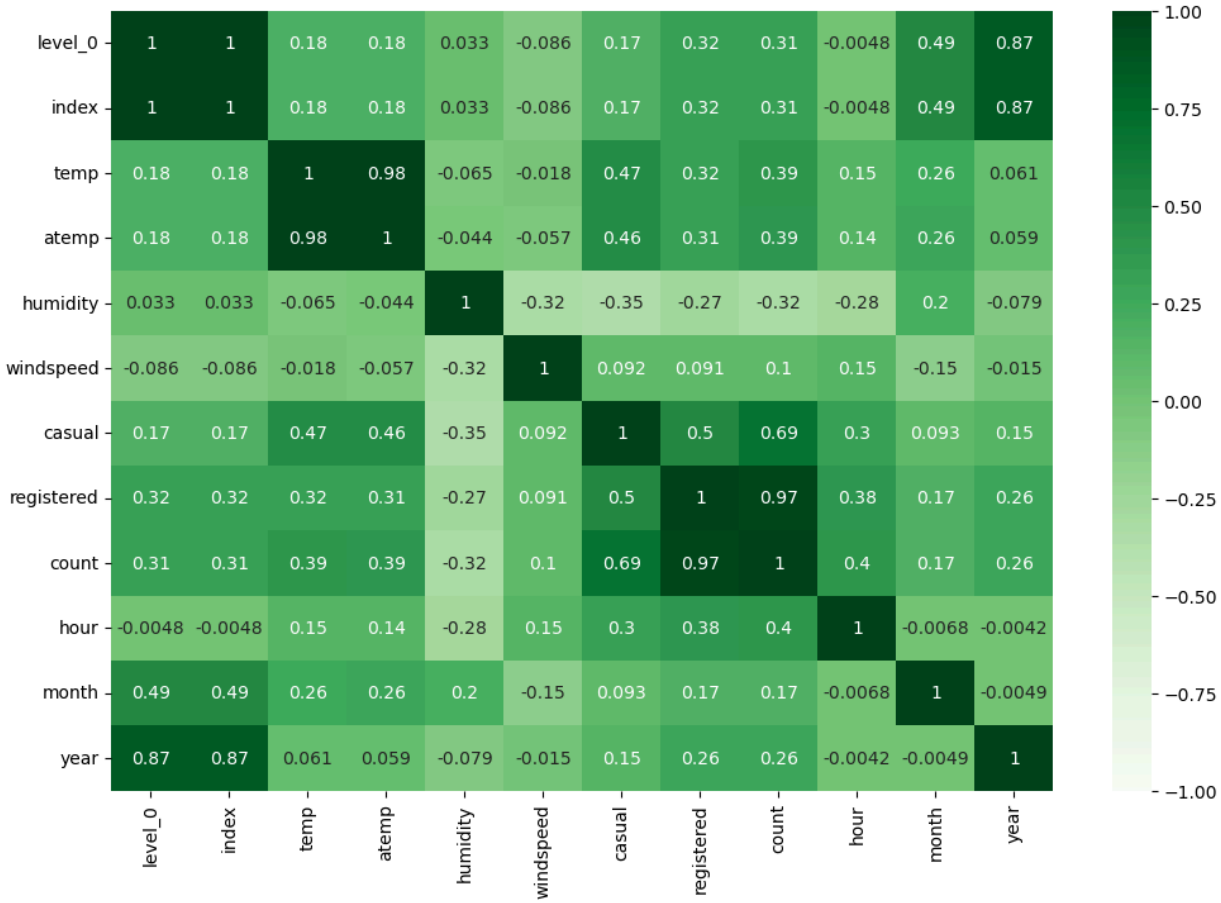
In [85]:

```
# Generating a pair plot to see the relation between different numerical attributes.
df1 = df[['temp','atemp','humidity','windspeed','casual','registered','count','working
sns.pairplot(data = df1,
 kind = 'reg',
 hue = 'workingday',
 markers="." )
plt.show()
```



In [87]:
```
# Correlation matrix for all the numerical attributes.
corr_data = df.corr(numeric_only=True)
plt.figure(figsize = (12, 8))
sns.heatmap(data = corr_data, cmap = 'Greens', annot = True, vmin = -1, vmax = 1)
plt.plot()
```

Out[87]:
```
[]
```



Very High Correlation (> 0.9) exists between columns [atemp, temp] and [count, registered]

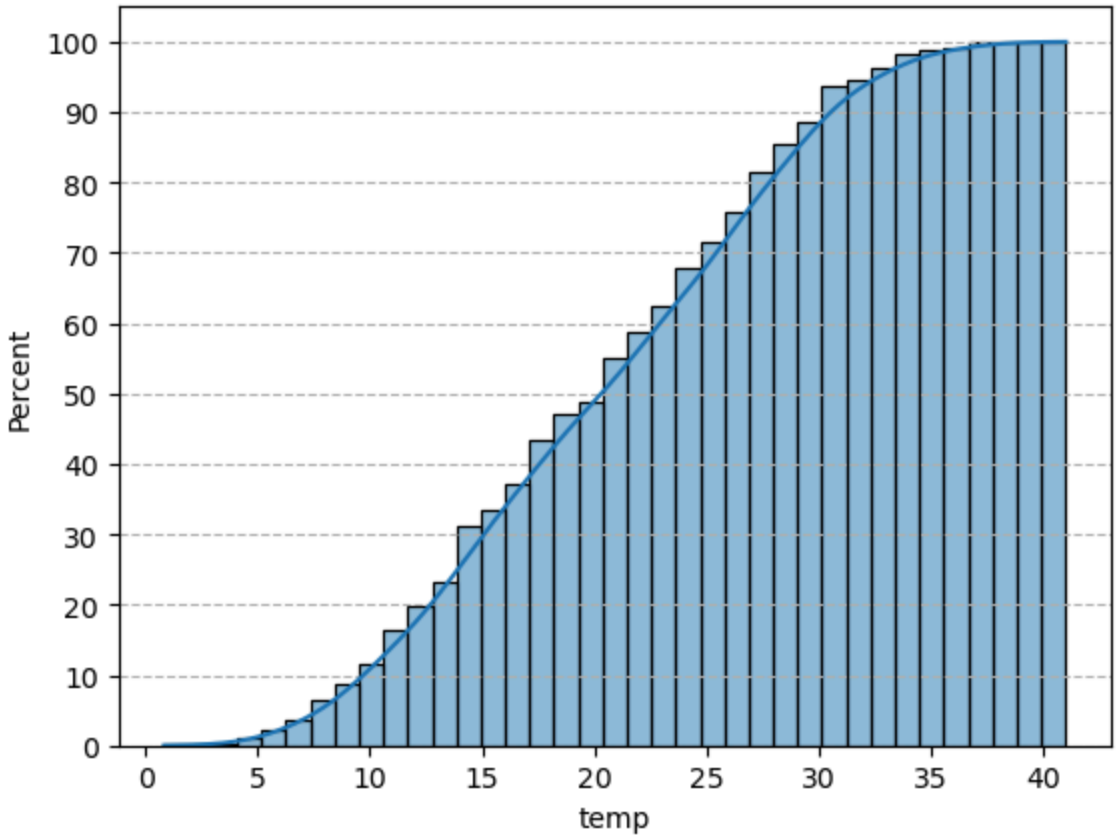
High positive/negative correlation (0.7 - 0.9) does not exist between any other columns.

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.

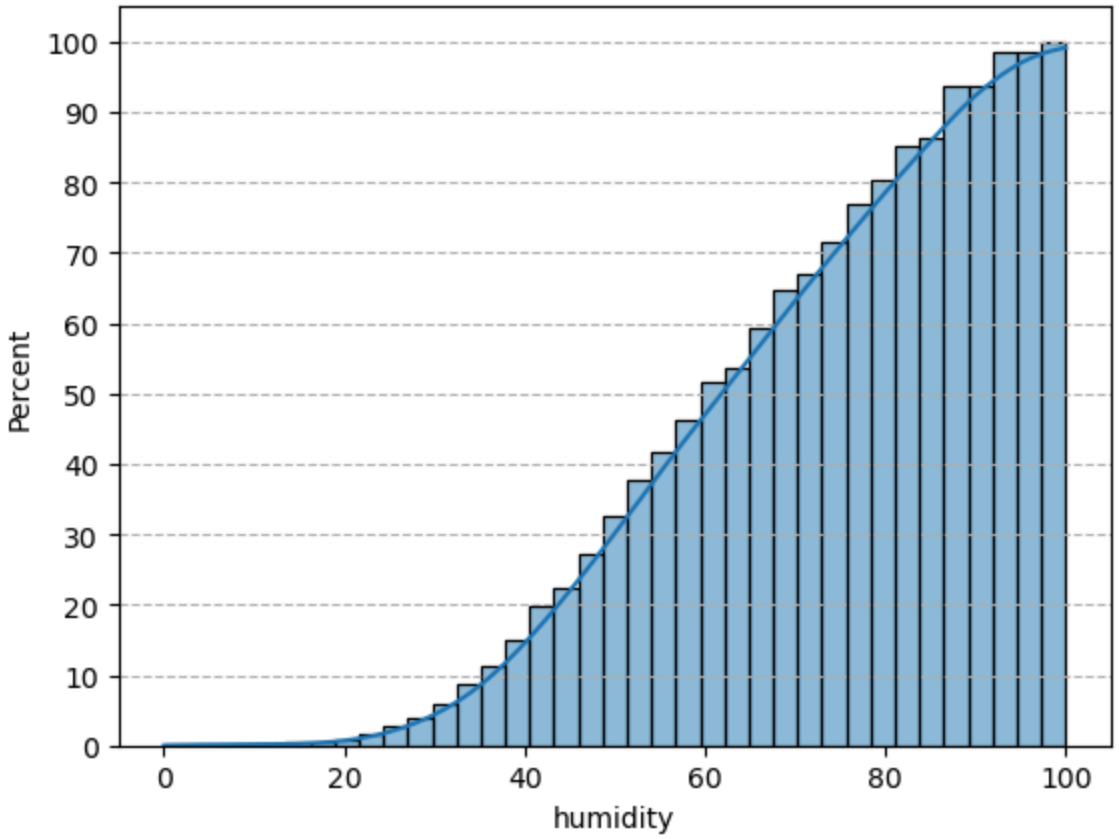
In [88]:
```
# Generating a histogram plot for the 'temp' feature, showing the cumulative distribut
# of temperature values in the dataset.
sns.histplot(data = df, x = 'temp', kde = True, cumulative = True, stat = 'percent')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.show()
```



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

In [89]:
```
# Generating a histogram plot for the 'humidity' feature, showing the cumulative
 # distribution of humidity values in the dataset.
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
```

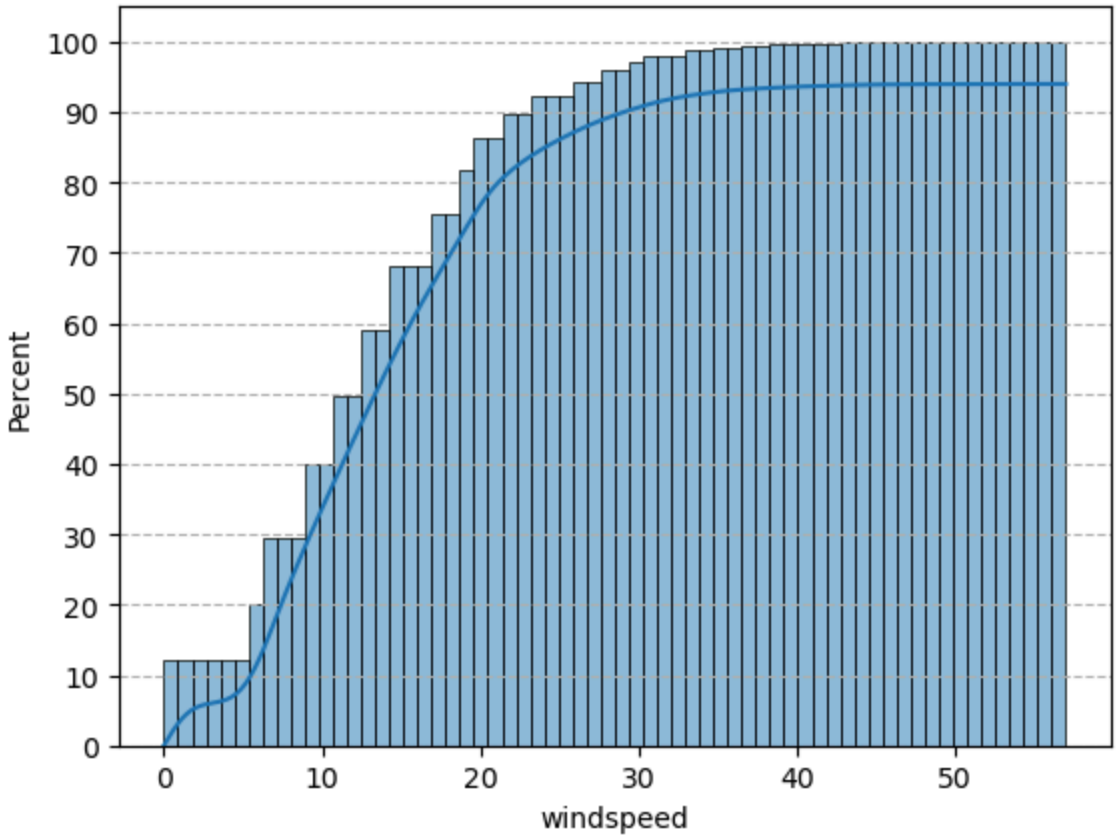
Out[89]:
```
[]
```



More than 80 % of the time, the humidity value is greater than 42. Thus for most of the time, humidity level varies from optimum to too moist.

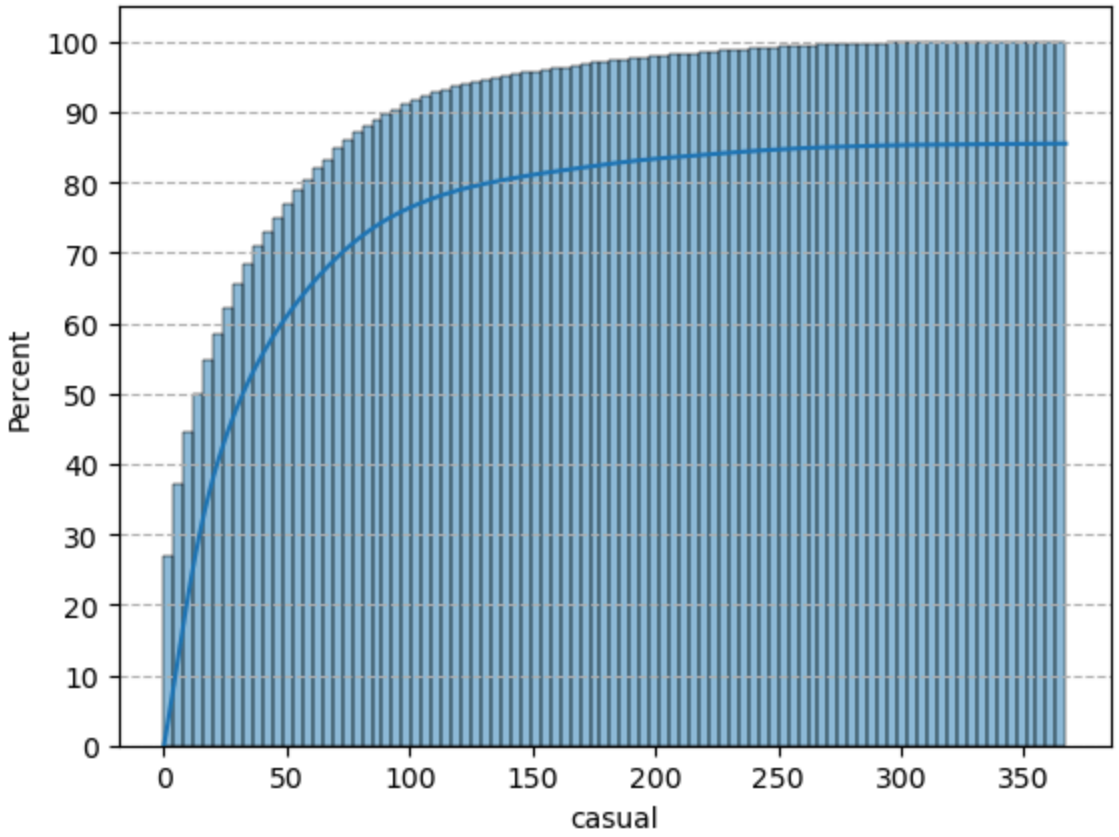
In [91]:
```
# Generating a histogram plot for the 'windspeed' feature, showing the cumulative
 # distribution of windspeed values in the dataset.
sns.histplot(data = df, x = 'windspeed', kde = True, cumulative = True, stat = 'percer
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.plot()
```

Out[91]:
```
[]
```



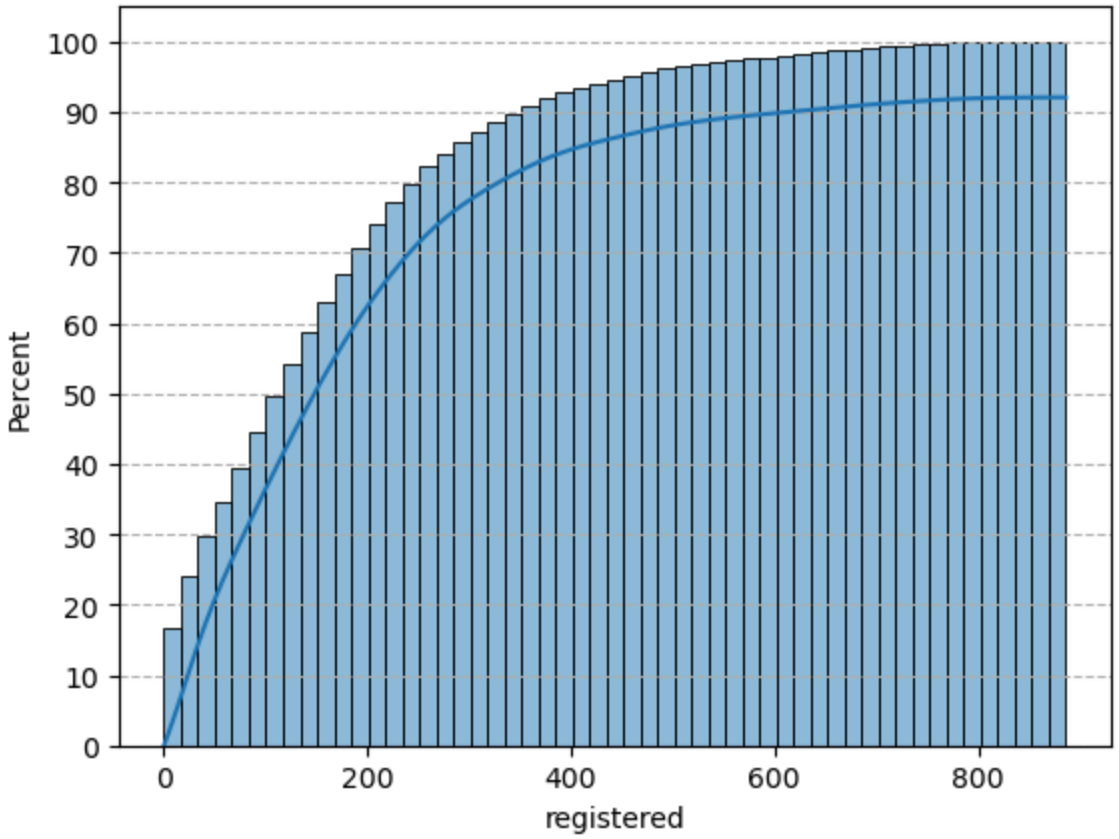
More than 85 % of the total windspeed data has a value of less than 20.

In [92]:
```
# Generating a histogram plot for the 'casual users', showing the cumulative
 # distribution of casual users values in the dataset.
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.show()
```



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

In [93]:
```
# Generating a histogram plot for the 'registered users', showing the cumulative
 # distribution of registered users values in the dataset.
sns.histplot(data = df, x = 'registered', kde = True, cumulative = True, stat = 'perce
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.show()
```



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

4. Hypothesis testing

4.1 Checking if there any significant difference between the no. of bike rides on Weekdays and Weekends.

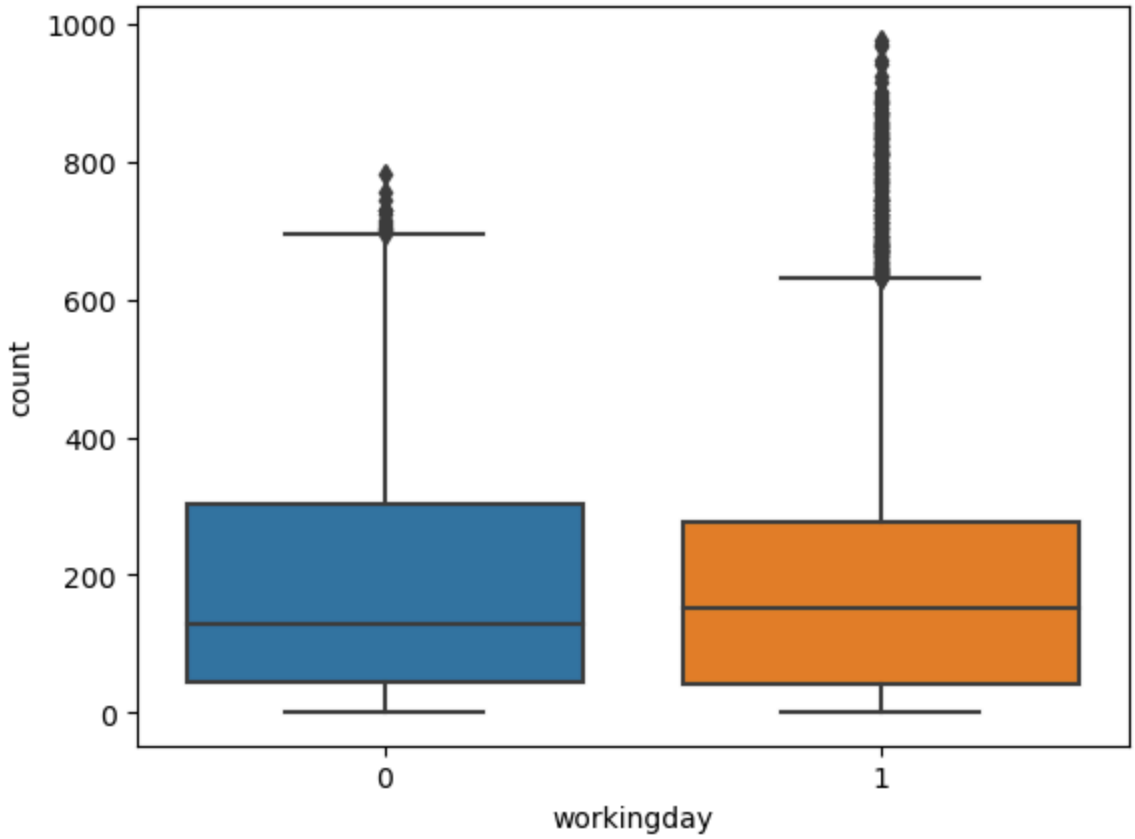
In [94]:
```
df.groupby(by = 'workingday')['count'].describe()
```

Out[94]:

workingday	count	mean	std	min	25%	50%	75%	max
0	3474.0	188.506621	173.724015	1.0	44.0	128.0	304.0	783.0
1	7412.0	193.011873	184.513659	1.0	41.0	151.0	277.0	977.0

In [95]:
```
sns.boxplot(data = df, x = 'workingday', y = 'count')
plt.plot()
```

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



STEP-1 : Defining Null and Alternate Hypothesis.

```
    Null Hypothesis(Ho) - 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 assumptions for the hypothesis:

```
   Distribution Check for QQ plot

   Homogeneity of variances using levene's test.
```

step-3: Defining 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: Computing p value and fixing alpha value.

```
     We set Alpha=0.05
```

step-5: Compairing p value and alpha

Based on pvalue and alpha,we will accept or reject H0

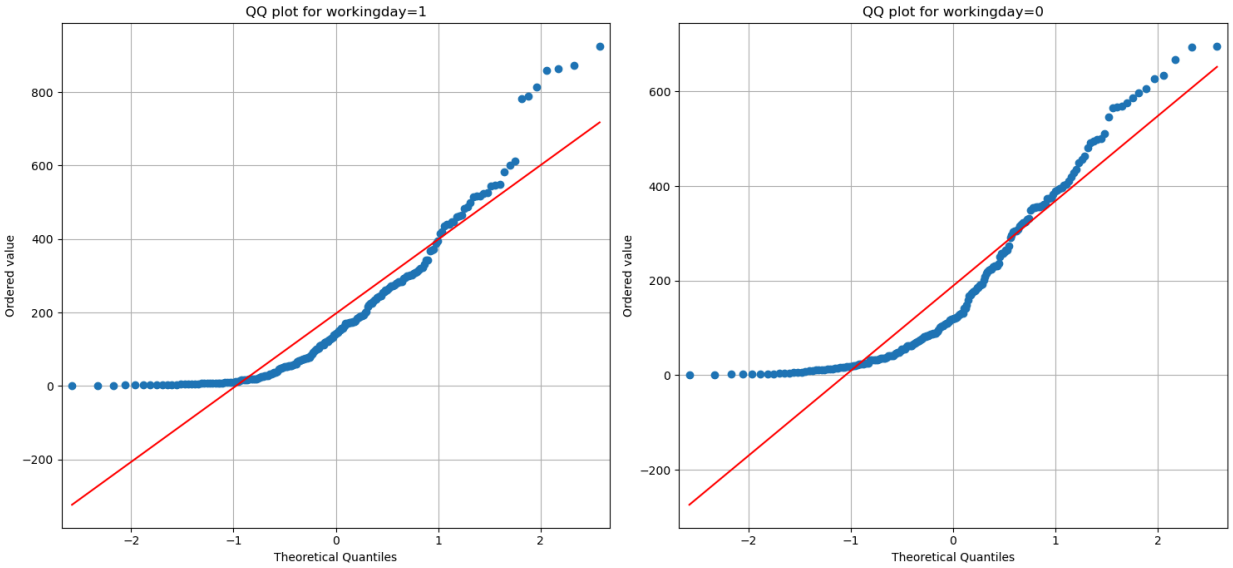
If p value > alpha then accept H0

If p value < alpha then reject H0

Checking for normal Distribution using QQ plot

In [97]:
```
# Checking the normality of the distribution using QQ-plots
# QQ plot weather
from statsmodels.graphics.gofplots import qqplot
df1 = df.loc[df['workingday'] == 1, 'count'].sample(200)
df2 = df.loc[df['workingday'] == 0, 'count'].sample(200)
plt.figure(figsize=(6,7))
# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 7))
# Plot QQ plots for each weather category
for i, (data, workday) in enumerate(zip([df1, df2], [1, 0])):
    ax = axes[i]
    qqplot(data, line='s', ax=ax)
    ax.set_title(f'QQ plot for workingday={workday}')
    ax.set_ylabel('Ordered value')
    ax.grid(True)
plt.tight_layout()
plt.show()
```

```
<Figure size 600x700 with 0 Axes>
```



From the above plot it is clear that distribution is not normal.

Checking for homogeneous variance

In [98]:
```
# Checking Homogeneity of Variances using Lavene's test.
from scipy.stats import levene
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance
```

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

```
p-value 0.3872326025377243
The samples have Homogenous Variance
```

p-value 0.5689524063282737

The samples have Homogenous Variance

So we can see that samples have homogenous variance but dont follow a normal distrubution. Since t-test assumes normal distribution and homogeneity of variances, we have atleast 1 condition fulfilling here.

Although t-test is not very suitable here and we can use non- parametric tests like Mann-Whitney U test but using t-test would still give us satisfactory results.

In [99]:
```
# Assuming normal distribution we can perform 2 sample independent t-test
# to check if the working day is having any effect on the no of users or not.
from scipy.stats import ttest_ind
# 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_stat, p_value = ttest_ind(df.loc[df['workingday'] == 1, 'count'],
 df.loc[df['workingday'] == 0, 'count'])
print('P-value :',p_value)
if p_value < 0.05:
 print('Mean no.of electric cycles rented is not same for working and non-working days
else:
 print('Mean no.of electric cycles rented is same for working and non-working days')
```

```
P-value : 0.22644804226361348
Mean no.of electric cycles rented is same for working and non-working days
```

P-value : 0.22644804226361348

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 .

4.2 Checking if there any significant difference between the no. of bike rides on Holidays and non-holidays.

In [100…
```
df.groupby(by = 'holiday')['count'].describe()
```

Out[100]:

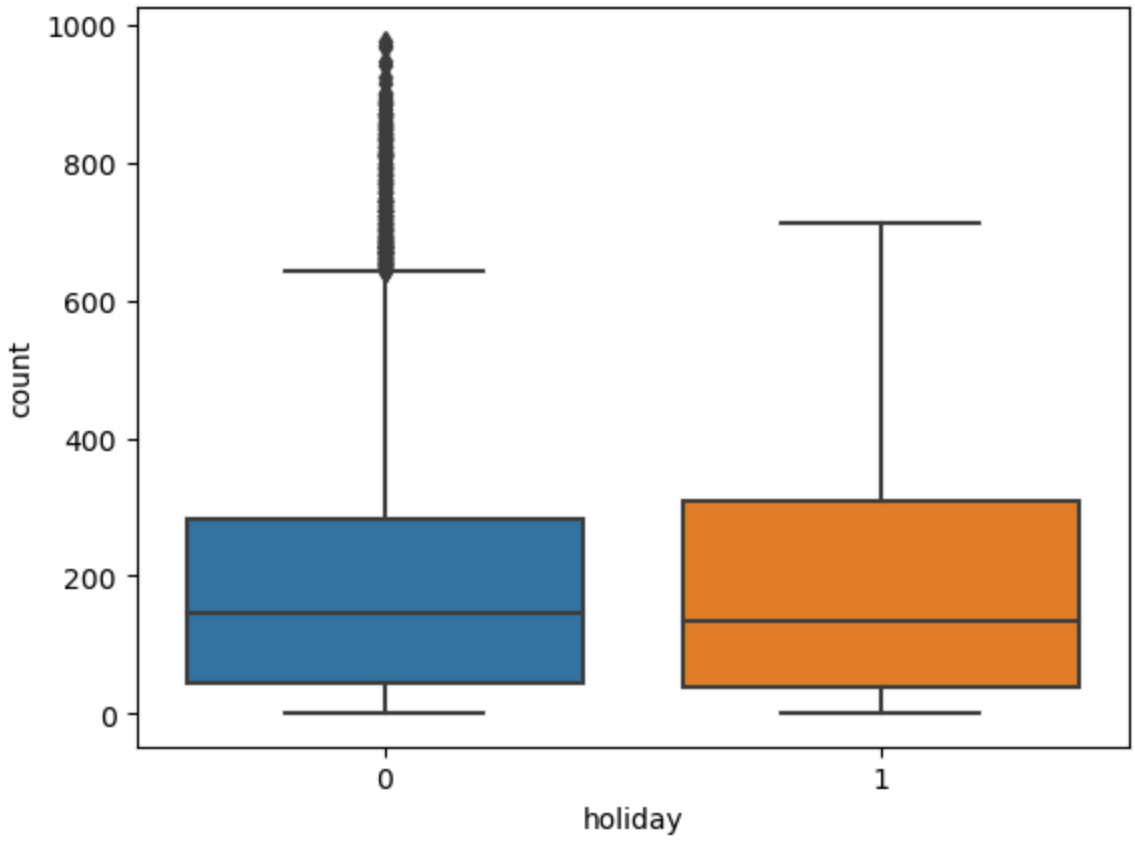
holiday	count	mean	std	min	25%	50%	75%	max
0	10575.0	191.741655	181.513131	1.0	43.0	145.0	283.0	977.0
1	311.0	185.877814	168.300531	1.0	38.5	133.0	308.0	712.0

In [101…

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

Out[101]:

```
[]
```



STEP-1 : Defining Null and Alternate Hypothesis.

```
Null Hypothesis(Ho) - 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 assumptions for the hypothesis:

Distribution Check for QQ plot

Homogeneity of variances using levene's test.

step-3: Defining 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: Computing p value and fixing alpha value.

```
We set Alpha=0.05
```

step-5: Compairing p value and alpha

Based on pvalue and alpha,we will accept or reject H0

If p value > alpha then accept H0

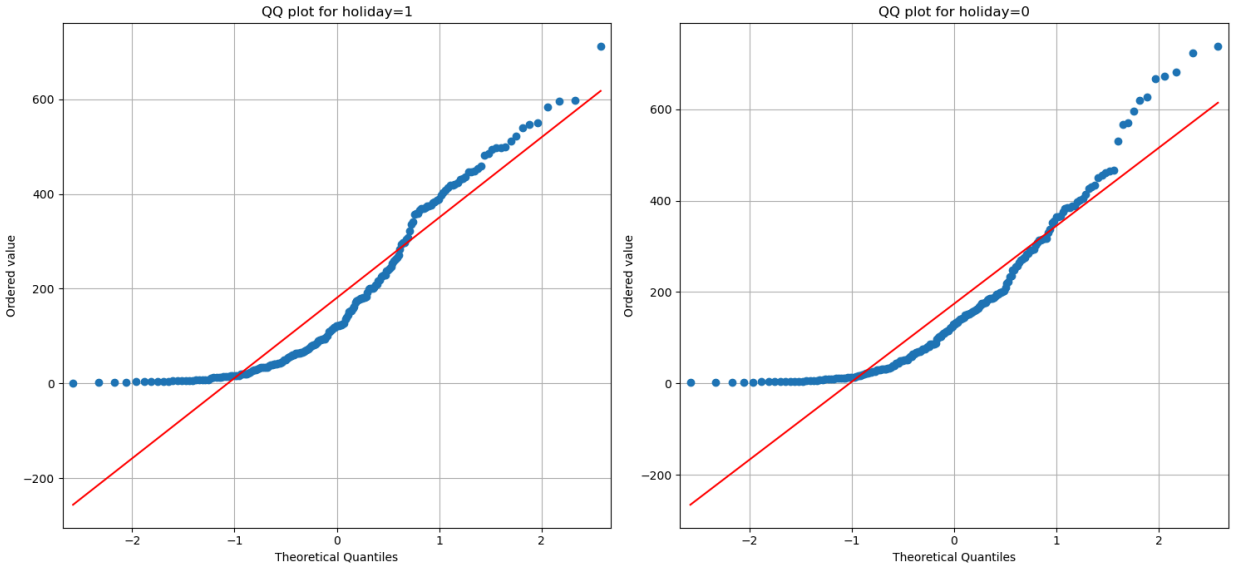
If p value < alpha then reject H0

Checking for normal Distribution using QQ plot

In [104…

```
 # Checking the normality of the distribution using QQ-plots
# QQ plot weather
from statsmodels.graphics.gofplots import qqplot
dt1 = df.loc[df['holiday'] == 1, 'count'].sample(200)
dt2 = df.loc[df['holiday'] == 0, 'count'].sample(200)
plt.figure(figsize=(6,7))
# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 7))
# Plot QQ plots for each weather category
for i, (data, holidayy) in enumerate(zip([dt1, dt2], [1, 0])):
    ax = axes[i]
    qqplot(data, line='s', ax=ax)
    ax.set_title(f'QQ plot for holiday={holidayy}')
    ax.set_ylabel('Ordered value')
    ax.grid(True)

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

```
<Figure size 600x700 with 0 Axes>
```



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

Checking for normality using Shapiro test

In [105…

```
# Applying Shapiro-Wilk test toc check the normality.
from scipy.stats import shapiro
# Ho: The sample follows normal distribution
# Ha: The sample does not follow normal distribution
alpha = 0.05
# For holiday
test_stat, p_value = shapiro(df.loc[df['holiday'] == 1, 'count'].sample(200))
print('p-value', p_value)
if p_value < 0.05:
 print('The sample does not follow normal distribution')
else:
 print('The sample follows normal distribution')

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

```
p-value 7.868279812417445e-12
The sample does not follow normal distribution
p-value 3.615824584191074e-10
The sample does not follow normal distribution
```

so the distribution is not normal.

Checking for Homogeneity of variances using Levene's test

In [106…

```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance
test_stat, p_value = levene(df.loc[df['holiday'] == 0, 'count'].sample(200),
 df.loc[df['holiday'] == 1, 'count'].sample(200))
print('p-value', p_value)
if p_value < 0.05:
```

```
 print('The samples do not have Homogenous Variance')
else:
 print('The samples have Homogenous Variance ')
```

```
p-value 0.9955614730923873
The samples have Homogenous Variance
```

So we can see that samples have homogenous variance but dont follow a normal distrubution. Since t-test assumes normal distribution and homogeneity of variances, we have atleast 1 condition fulfilling here.

Although t-test is not very suitable here and we can use non- parametric tests like Mann-Whitney U test but using t-test would still give us satisfactory results.

In [107…

```
# Assuming normal distribution we can perform 2 sample independent t-test
# to check if the holiday is having any effect on the no of users or not.
from scipy.stats import ttest_ind
# Ho : No of electric cycles rented is same for working and non-working days
# Ha : No of electric cycles rented is not same for working and non-working days
# Assuming significance Level to be 0.05
test_stat, p_value = ttest_ind(df.loc[df['holiday'] == 0, 'count'].sample(200),
 df.loc[df['holiday'] == 1, 'count'].sample(200))
print('P-value :',p_value)
if p_value < 0.05:
 print('No of electric cycles rented is not similar for holidays and non-holidays')
else:
 print('No of electric cycles rented is similar for holidays and non-holidays')
```

```
P-value : 0.7319836987224306
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.

4.3. Checking if the weather has any significant effect on the no. of bikes rented.

In [108…

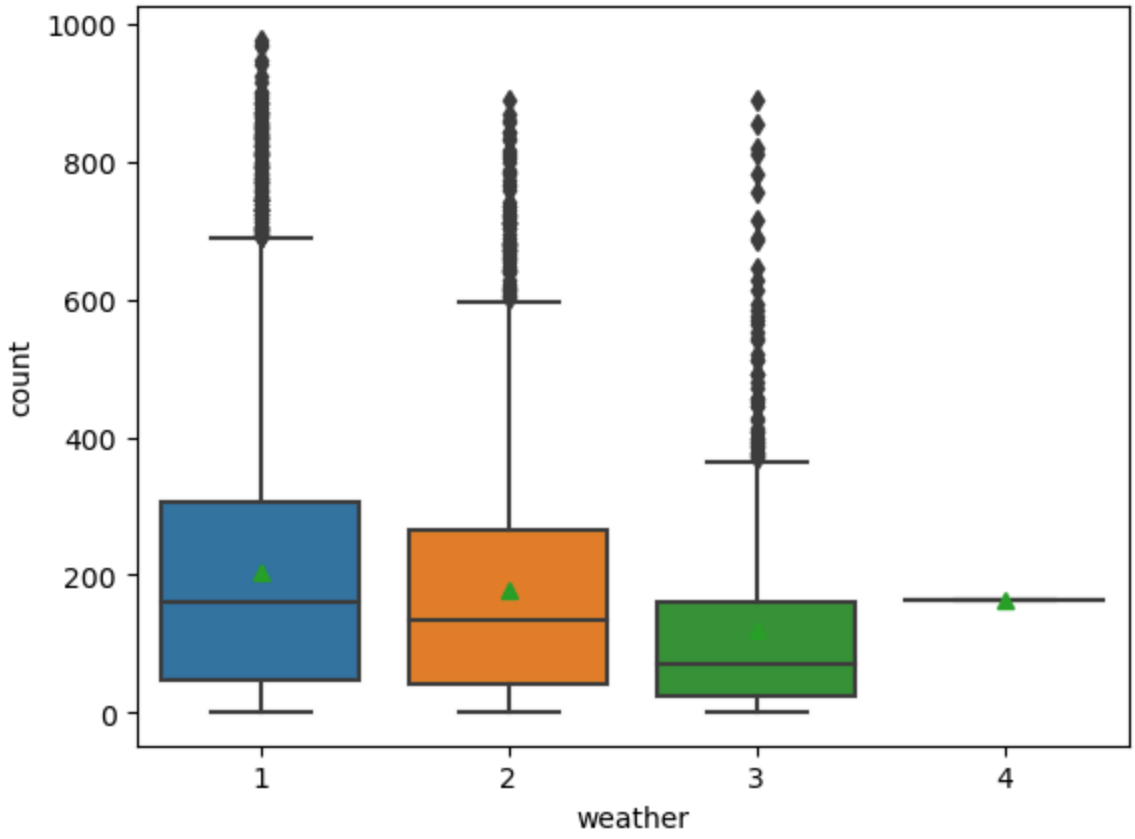
```
df.groupby(by = 'weather')['count'].describe()
```

Out[108]:

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 [109…

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



In [110…
```
df_weather1 = df.loc[df['weather'] == 1]
df_weather2 = df.loc[df['weather'] == 2]
df_weather3 = df.loc[df['weather'] == 3]
df_weather4 = df.loc[df['weather'] == 4]
len(df_weather1), len(df_weather2), len(df_weather3), len(df_weather4)
```

Out[110]:
```
(7192, 2834, 859, 1)
```

Since the weather condition 4 is having only 1 record, we can ignore that.

STEP-1 : Defining Null and Alternate Hypothesis.

Null Hypothesis(Ho) - 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 assumptions for the hypothesis:

Distribution Check for QQ plot

Homogeneity of variances using levene's test.

step-3: Defining 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: Computing p value and fixing alpha value.

We set Alpha=0.05

step-5: Compairing p value and alpha

Based on pvalue and alpha,we will accept or reject H0

If p value > alpha then accept H0

If p value < alpha then reject H0

Checking for normality using Kurtosis

In [111…
```
# firstly checking the normality using Kurtosis.
from scipy.stats import kurtosis
for i in range(1,4):
# Calculation of kurtosis
 data_kurtosis = kurtosis(df[df['weather'] == i]['count'])
 # Print the kurtosis value
 print(f'Kurtosis for weather={i}:', data_kurtosis)
```

```
Kurtosis for weather=1: 0.9632151489948488
Kurtosis for weather=2: 1.5835130178554868
Kurtosis for weather=3: 5.961191782478394
```

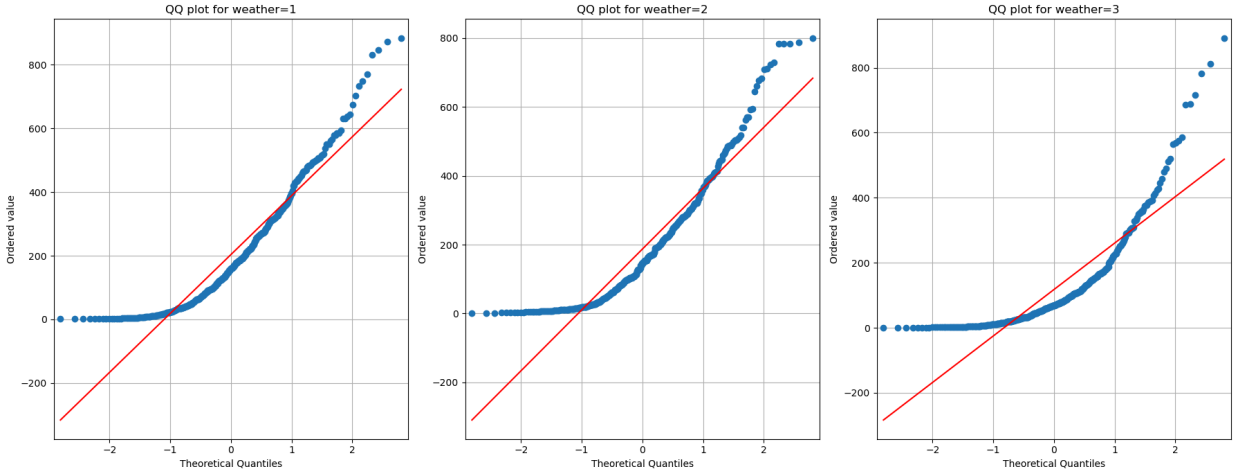
Positive kurtosis signals that the distribution is more peaked (leptokurtic) than a normal distribution.

Tails of the distribution are heavier and there are more outliers.

Checking for normality using QQ-plots

In [114…
```
# Checking the distribution of count using QQ plot.
# QQ plot weather
from statsmodels.graphics.gofplots import qqplot
dt1 = df.loc[df['weather'] == 1, 'count'].sample(400)
dt2 = df.loc[df['weather'] == 2, 'count'].sample(400)
dt3 = df.loc[df['weather'] == 3, 'count'].sample(400)
plt.figure(figsize=(6,7))
# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 7))
# Plot QQ plots for each weather category
for i, (data, weather) in enumerate(zip([dt1, dt2, dt3], [1, 2, 3])):
    ax = axes[i]
    qqplot(data, line='s', ax=ax)
    ax.set_title(f'QQ plot for weather={weather}')
    ax.set_ylabel('Ordered value')
    ax.grid(True)
plt.tight_layout()
plt.show()
```

```
<Figure size 600x700 with 0 Axes>
```

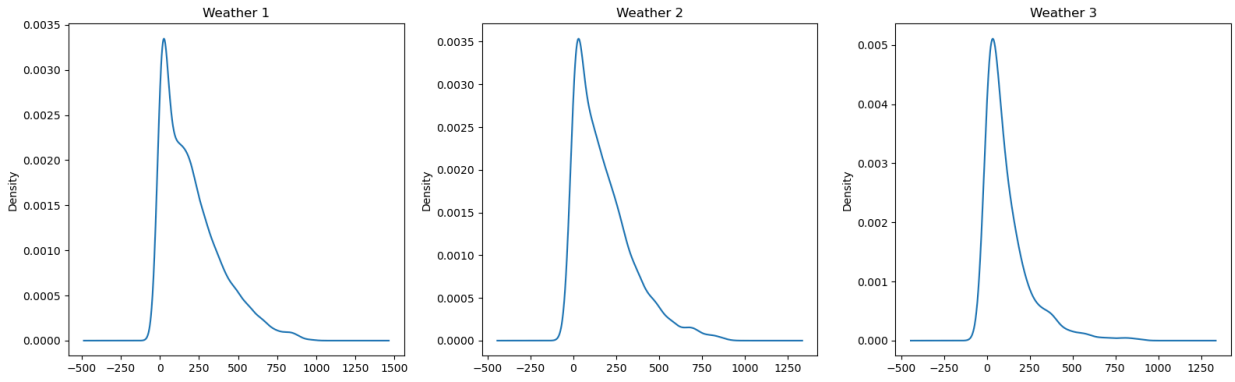


It can be inferred from the above plots that all three weather are having the distributions that do not follow normal distribution pattern.

Checking for normality using Skewness

In [115…

```
# Checking for normality of count for different weather conditions using Skewness.
data1 = df.loc[df['weather'] == 1, 'count']
data2 = df.loc[df['weather'] == 2, 'count']
data3 = df.loc[df['weather'] == 3, 'count']
plt.figure(figsize=(16, 5))
plt.subplot(1, 3, 1)
data1.plot(kind='density', sharex=False)
plt.title('Weather 1')
plt.subplot(1, 3, 2)
data2.plot(kind='density', sharex=False)
plt.title('Weather 2')
plt.subplot(1, 3, 3)
data3.plot(kind='density', sharex=False)
plt.title('Weather 3')
plt.tight_layout()
plt.show()
```



For all three weather patterns the dirtribution is positive skewed or right skewed which indicated the distribution is not normal.

Checking for Homogeneity of Variances using Levene's test

In [116…
```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance
test_stat, p_value = levene(df_weather1.loc[:, 'count'].sample(400),
                      df_weather2.loc[:, 'count'].sample(400),
                      df_weather3.loc[:, 'count'].sample(400))
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 6.682676577585303e-12
The samples do not have Homogenous Variance
```

Since the samples are not normally distributed and do not have the Homogenous variance, One-way ANOVA test should not be performed here, we can perform its non parametric equivalent test i.e., Kruskal-Wallis test for independent samples. But we can still perform ANOVA assuming the normal distribution.

Performing One-way ANOVA test assuming conditions satisfied

In [117…
```
# Performing One-way ANOVA assuming normal distribution.
# Assuming significance= 0.05
from scipy.stats import f_oneway # Numeric Vs categorical for many categories.
data1 = df.loc[df['weather'] == 1, 'count']
data2 = df.loc[df['weather'] == 2, 'count']
data3 = df.loc[df['weather'] == 3, 'count']
# Ho: All groups have the same mean
# Ha: One or more groups have different mean
f_stats, p_value = f_oneway(data1, data2, data3)
print("test statistic:",f_stats)
print("p_value:",p_value)
if p_value < 0.05:
    print("The mean number of rental bikes is statistically different for different we
else:
    print("The average number of rental bikes is statistically similar for different w
```

```
test statistic: 98.28356881946706
p_value: 4.976448509904196e-43
The mean number of rental bikes is statistically different for different weathers.
```

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

4.4. Checking if the season has any significant effect on the no. of bikes rented.

In [118…
```
df.groupby(by = 'season')['count'].describe()
```

Out[118]:

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

In [121…

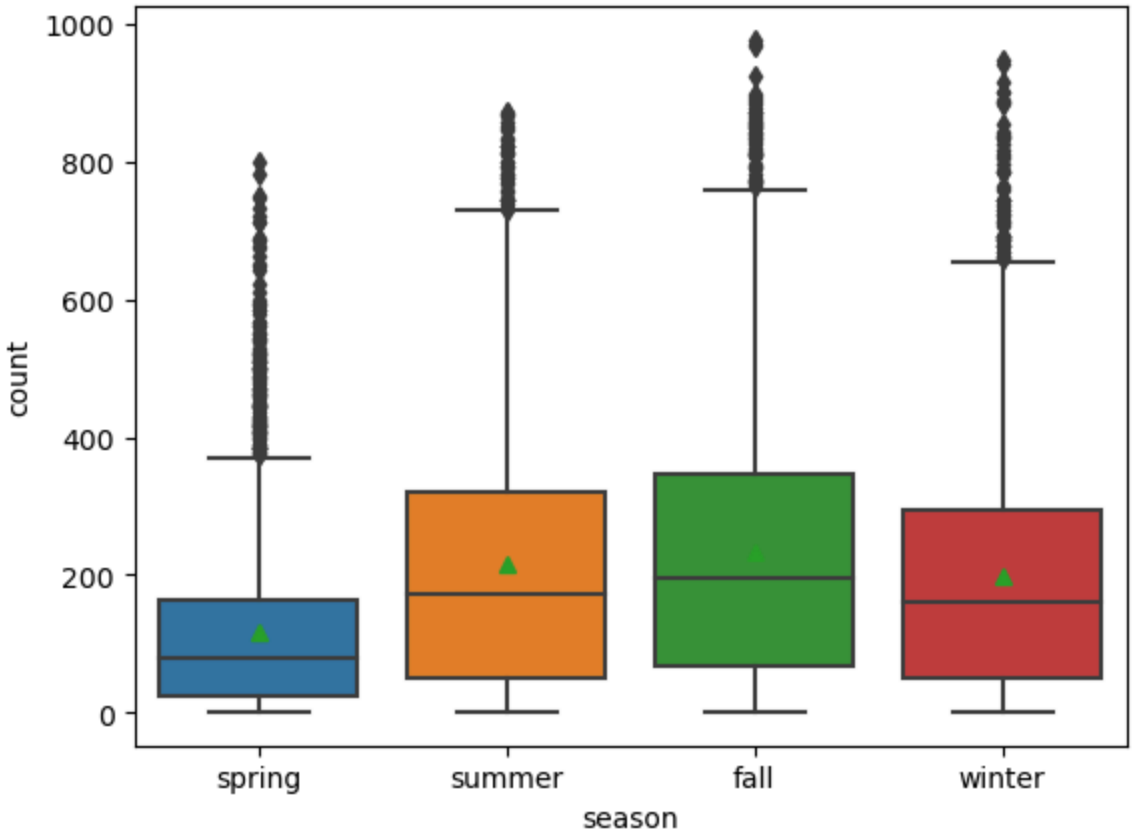
```
df_season_spring = df.loc[df['season'] == 'spring', 'count']
df_season_summer = df.loc[df['season'] == 'summer', 'count']
df_season_fall = df.loc[df['season'] == 'fall', 'count']
df_season_winter = df.loc[df['season'] == 'winter', 'count']
len(df_season_spring), len(df_season_summer), len(df_season_fall),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

Out[121]:

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

In [120…

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



STEP-1 : Defining Null and Alternate Hypothesis.

Null Hypothesis(Ho) - Mean no. of cycles rented per hour is same for season 1,2,3 and 4.

Alternate Hypothesis(Ha) - Mean no. of cycles rented per hour is different for season 1,2,3 and 4.

Step2: Checking for basic assumptions for the hypothesis.

Distribution check using QQ Plot or kurtosis or Skewness.

Homogeneity of variances using Levene's test

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

If the assumptions of One-way ANOVA are met then we can proceed performing One-way ANOVA or else we can assume the -->

conditions are met and can still perform One-way ANOVA test.

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

- We set our alpha to be 0.05

STEP-5 : Comparing p-value and alpha.

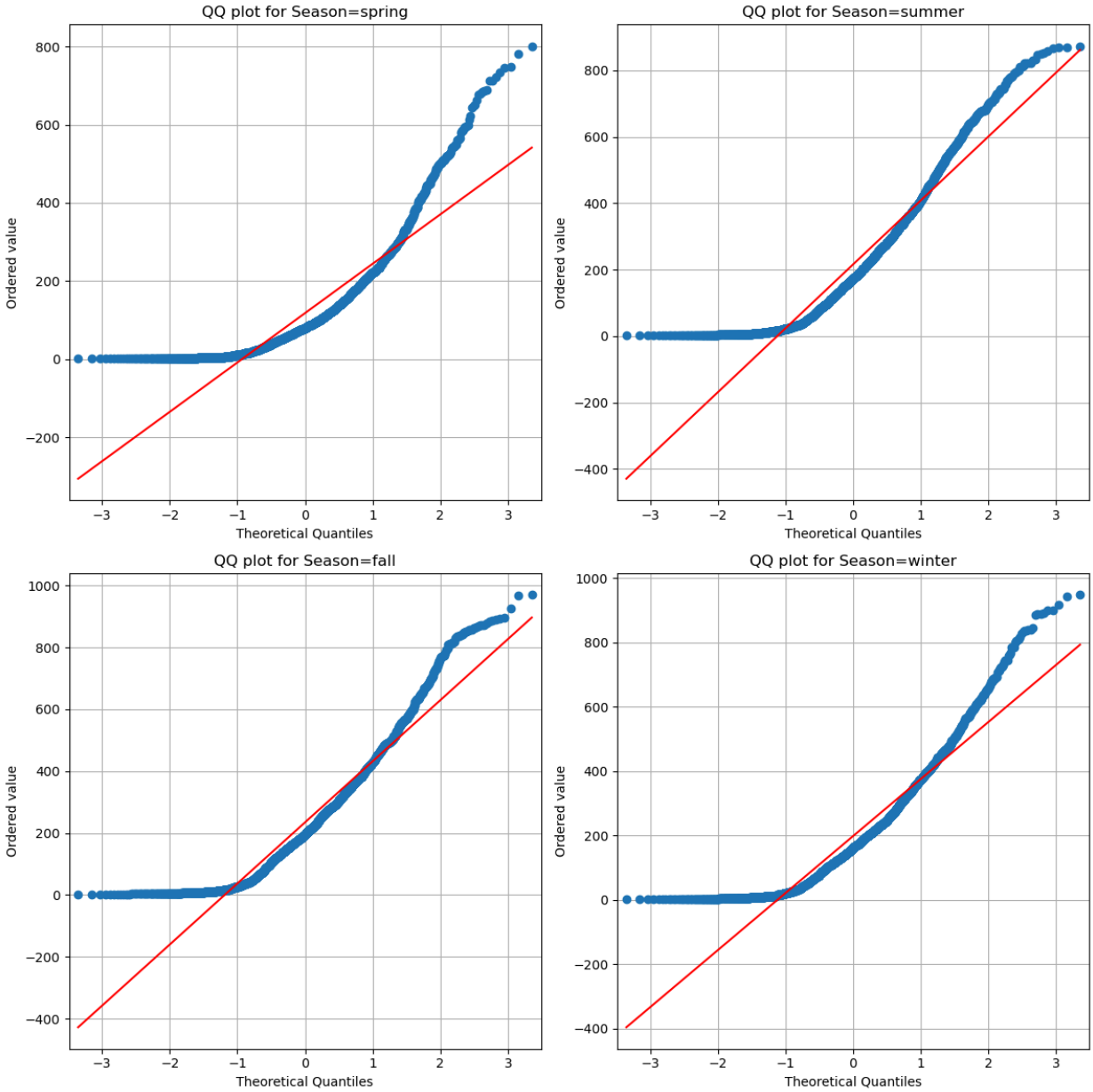
Based on p-value, we will accept or reject H0.

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

Checking for normality using QQ plot

In [123…

```
# Checking the normality by using QQ-plots.
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
# Plot QQ plots for each season category
for ax, (data, season) in zip(axes.ravel(), [(df_season_spring.sample(2500), 'spring')
                                             (df_season_summer.sample(2500), 'summer')
                                             (df_season_fall.sample(2500), 'fall'),
                                             (df_season_winter.sample(2500), 'winter')
    qqplot(data, line='s', ax=ax)
    ax.set_title(f'QQ plot for Season={season}')
    ax.set_ylabel('Ordered value')
    ax.grid(True)
plt.tight_layout()
plt.show()
```



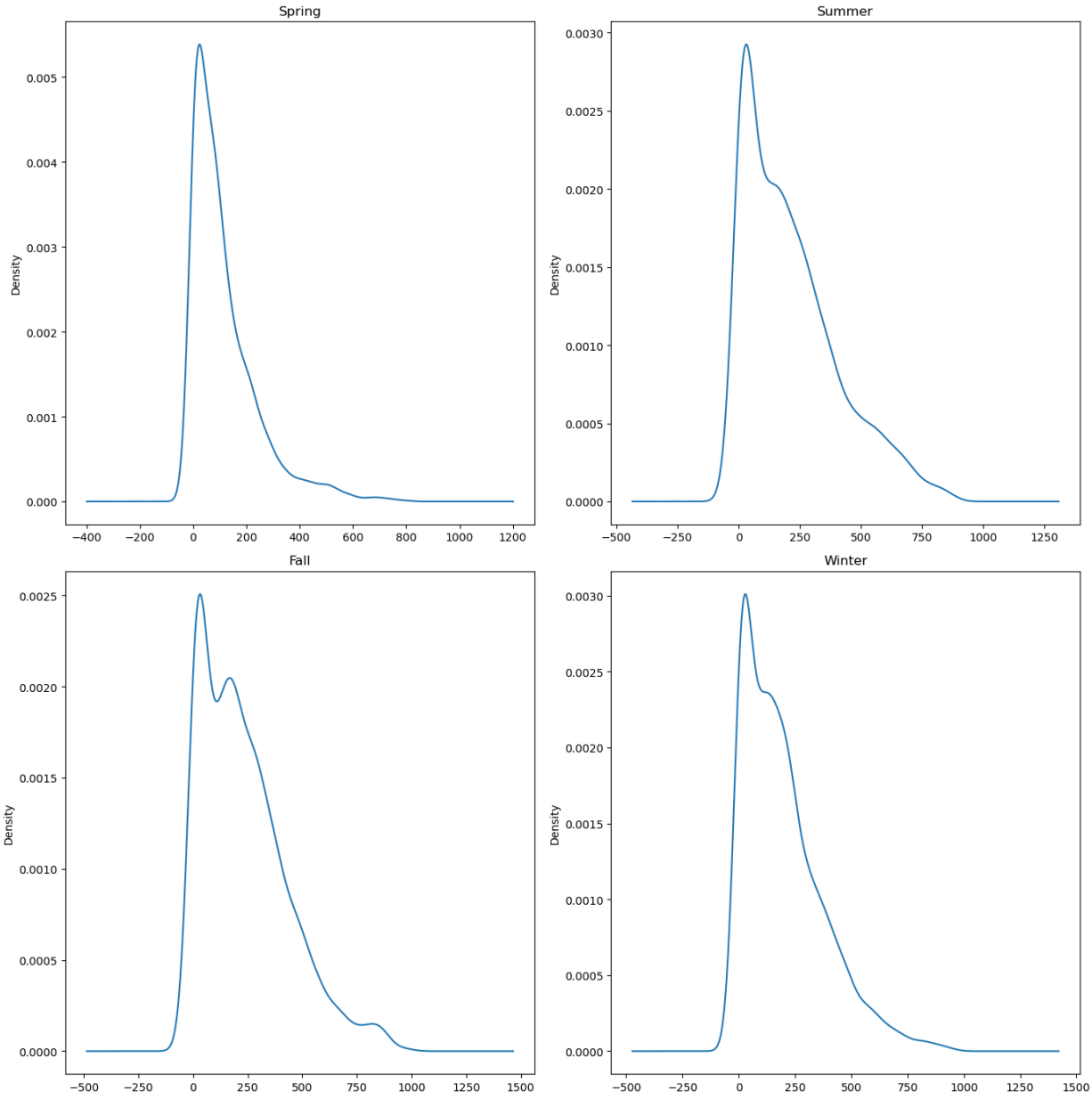
It can be inferred from the above plots that the distributions do not follow normal distribution.

Checking for normality using Skewness

In [124…

```
# Checking for normality of count for different seasons using Skewness.
df_season_spring = df.loc[df['season'] == 'spring', 'count']
df_season_summer = df.loc[df['season'] == 'summer', 'count']
df_season_fall = df.loc[df['season'] == 'fall', 'count']
df_season_winter = df.loc[df['season'] == 'winter', 'count']
plt.figure(figsize=(14, 14))
plt.subplot(2, 2, 1)
df_season_spring.plot(kind='density', sharex=False)
plt.title('Spring')
plt.subplot(2, 2, 2)
df_season_summer.plot(kind='density', sharex=False)
plt.title('Summer')
plt.subplot(2, 2, 3)
df_season_fall.plot(kind='density', sharex=False)
plt.title('Fall')
plt.subplot(2, 2, 4)
```

```
df_season_winter.plot(kind='density', sharex=False)
plt.title('Winter')
plt.tight_layout()
plt.show()
```



For all four seasons the distribution is positive skewed or right skewed which indicated the distribution is not normal

Checking for normality using Kurtosis

In [126…

```
# Checking for normality using Kurtosis.
lst=['spring','summer','fall','winter']
for seasonn in lst:
# Calculation of kurtosis
 data_kurtosis = kurtosis(df[df['season'] == seasonn]['count'])
 # Print the kurtosis value
 print(f'Kurtosis for {seasonn} season:', data_kurtosis)
```

```
Kurtosis for spring season: 4.30449666648592
Kurtosis for summer season: 0.4222412657621657
Kurtosis for fall season: 0.6959091337333851
Kurtosis for winter season: 1.2689637849725477
```

Positive kurtosis signals that the distribution is more peaked (leptokurtic) than a normal distribution.

Tails of the distribution are heavier and there are more outliers

Checking for normality using Shapiro test.

In [128…

```
# Checking normality using shapiro test.
from scipy.stats import shapiro, kruskal
d = {}
d[1] = df_season_spring.sample(2500)
d[2] = df_season_summer.sample(2500)
d[3] = df_season_fall.sample(2500)
d[4] = df_season_winter.sample(2500)
for i in range(1, 5):
    test_stat, p_value = shapiro(d[i])
    print('p-value:', p_value)
    if p_value < 0.05:
        print('The sample for does not follow a normal distribution')
    else:
        print('The sample follows a normal distribution')
```

```
p-value: 0.0
The sample for does not follow a normal distribution
p-value: 9.77625402465766e-38
The sample for does not follow a normal distribution
p-value: 1.480914960733062e-35
The sample for does not follow a normal distribution
p-value: 2.1948228159758994e-38
The sample for does not follow a normal distribution
```

It can be interpreted from the above observation that non of the distributions is normal.

Checking the homogeneity of the variances using Levene's test.

In [129…

```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance
test_stat, p_value = levene(df_season_spring.sample(2500),
 df_season_summer.sample(2500),
 df_season_fall.sample(2500),
 df_season_winter.sample(2500))
print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

```
p-value 9.052917501430857e-115
The samples do not have Homogenous Variance
```

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

Wallis H-test for independent samples.

In [130…
```
# Ho : Mean no. of cycles rented is same for different seasons.
# Ha : Mean no. of cycles rented is different for different seasons.
# Assuming significance Level to be 0.05
alpha = 0.05
test_stat, p_value = kruskal(df_season_spring, df_season_summer, df_season_fall,df_sea
print('Test Statistic =', test_stat)
print('p value =', p_value)
if p_value < alpha:
    print('Mean no. of cycles rented is different for different seasons')
else:
    print('Mean no. of cycles rented is same for different seasons.')
```

```
Test Statistic = 699.6668548181988
p value = 2.479008372608633e-151
Mean no. of cycles rented is different for different seasons
```

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

4.5 Checking if Weather conditions are significantly different during different seasons.

In [131…
```
df[['weather', 'season']].describe()
```

Out[131]:

	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 : Defining Null and Alternate Hypothesis.

Null Hypothesis(Ho) - Weather is independent of the season.

Alternate Hypothesis(Ha) - Weather is significantly dependent of the season.

step2: Checking for the basic assumptions of the hypothesis

1.The data in the cells should be frequencies, or counts of cases.

1. 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.
2. There are 2 variables, and both are measured as categories.
3. Value of the cell expected should be 5 or more e in at least 80% of the cells, and no cell should have an expected of less than one (3)

step3: Defining 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-4 : Computing 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 : Comparing 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 [134…
```
# First, finding the contingency table such that each value is the total number of tot
 # 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[134]:

season	1	2	3	4
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 [135…
```
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[135]:
```
array([[470116, 139386,  31160],
       [223009,  76406,  12919],
       [426350, 134177,  27755],
       [356588, 157191,  30255]], dtype=int64)
```

In [136…
```
from scipy.stats import chi2_contingency
chi_test_stat, p_value, dof, expected = chi2_contingency(observed = cross_table)
print('Test Statistic =', chi_test_stat)
print('p value =', p_value)
print('-' * 65)
print("Expected : '\n'", expected)
```

```
Test Statistic = 10838.372332480214
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 ]]
```

In [137…
```
 # Comparing p value with significance level
alpha = 0.05
if p_value < alpha:
    print('Weather is significantly dependent of the season.')
else:
    print('Weather is independent of the season.')
```

```
Weather is significantly dependent of the season.
```

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

5. Insights

Mean temp = 20.23 degree celsius & Standart deviation of temp = 7.79

Mean Humidity = 61.89 & Standart deviation of humidity = 19.24

Mean Windspeed = 12.8 & Standart deviation of windspeed = 8.16

Out of every 100 users, around 19 are casual users and 81 are registered users.

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

Weather patters is mostly Clear or Few clouds or partly cloudy or partly cloudy which is expected in a tropical country like India.

More bikes are rented on holidays and non-working days.

There is a seasonal pattern in the count of rental bikes, with higher demand during the fall and summer months, a slight decline in the winter, and a further decrease in the spring months.

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

The average hourly count of rental bikes is the lowest in the month of January followed by February and March.

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

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

More than 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.

6. Recommendations

Seasonal Marketing:

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

Time-based Pricing:

Take advantage of the hourly fluctuation in bike rental counts throughout the day. Consider implementing time-based pricing where rental rates are lower during off-peak hours and higher during peak hours. This can encourage customers to rent bikes during less busy times, balancing out the demand and optimizing the resources.

Weather-based Promotions:

Recognize the impact of weather on bike rentals. Create weather-based promotions that target customers during clear and cloudy weather, as these conditions show the highest rental counts. Yulu can offer weather-specific discounts to attract more customers during these favorable weather conditions.

User Segmentation:

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

Optimize Inventory:

Analyze the demand patterns during different months and adjust the inventory accordingly. During months with lower rental counts such as January, February, and March, Yulu can

optimize its inventory levels to avoid excess bikes. On the other hand, during peak months, ensure having sufficient bikes available to meet the higher demand.

Improve Weather Data Collection:

Given the lack of records for extreme weather conditions, consider improving the data collection process for such scenarios. Having more data on extreme weather conditions can help to understand customer behavior and adjust the operations accordingly, such as offering specialized bike models for different weather conditions or implementing safety measures during extreme weather.

Customer Comfort:

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

Collaborations with Weather Services:

Consider collaborating with weather services to provide real-time weather updates and forecasts to potential customers. Incorporate weather information into your marketing campaigns or rental app to showcase the ideal biking conditions and attract users who prefer certain weather conditions.

Seasonal Bike Maintenance:

Allocate resources for seasonal bike maintenance. Before the peak seasons, conduct thorough maintenance checks on the bike fleet to ensure they are in top condition. Regularly inspect and service bikes throughout the year to prevent breakdowns and maximize customer satisfaction.

Customer Feedback and Reviews:

Encourage customers to provide feedback and reviews on their biking experience. Collecting feedback can help identify areas for improvement, understand customer preferences,and tailor the services to better meet customer expectations.

Social Media Marketing:

Leverage social media platforms to promote the electric bike rental services. Share captivating visuals of biking experiences in different weather conditions, highlight customer testimonials, and engage with potential customers through interactive posts and contests. Utilize targeted advertising campaigns to reach specific customer segments and drive more bookings.

Special Occasion Discounts:

Since Yulu focusses on providing a sustainable solution for vehicular pollution, it should give special discounts on the occassions like Zero Emissions Day (21st September), Earth day (22nd April), World Environment Day (5th June) etc in order to attract new users.