About Yulu

1) 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.

2) 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!

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

Problem Statement

The company wants to know:

Which variables are significant in predicting the demand for shared electric cycles in the Indian market?

How well those variables describe the electric cycle demands

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

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

```
df.shape
```

```
(10886, 12)
```

```
df.head() # top 5 of data frame
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail() # last 5 of df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

```
df.sample(5) #randomly selected 5 of df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
793	2011-02-16 12:00:00	1	0	1	1	15.58	19.695	32	22.0028	14	72	86
4597	2011-11-04 15:00:00	4	0	1	1	18.86	22.725	44	26.0027	45	192	237
10567	2012-12-06 17:00:00	4	0	1	1	12.30	15.910	45	7.0015	31	586	617
2745	2011-07-03 07:00:00	3	0	0	2	26.24	28.790	89	12.9980	3	22	25
3940	2011-09-15 05:00:00	3	0	1	1	24.60	28.790	78	0.0000	1	30	31

```
df.columns
```

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

```
df.dtypes
```

	0
datetime	object
season	int64
holiday	int64
workingday	int64
weather	int64
temp	float64
atemp	float64
humidity	int64
windspeed	float64
casual	int64
registered	int64
count	int64

Every datatpe is correct except datetime.

we need to change object to datetime.

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

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

```
df.dtypes
```

	0
datetime	datetime64[ns]
season	int64
holiday	int64
workingday	int64
weather	int64
temp	float64
atemp	float64
humidity	int64
windspeed	float64
casual	int64
registered	int64
count	int64

```
df.isna()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...
10881	False	False	False	False	False	False	False	False	False	False	False	False
10882	False	False	False	False	False	False	False	False	False	False	False	False
10883	False	False	False	False	False	False	False	False	False	False	False	False
10884	False	False	False	False	False	False	False	False	False	False	False	False
10885	False	False	False	False	False	False	False	False	False	False	False	False

10886 rows × 12 columns

```
np.any(df.isna())# no null values
```

```
False
```

```
df.duplicated()
```

	0
0	False
1	False
2	False
3	False
4	False
...	...
10881	False
10882	False
10883	False
10884	False
10885	False

10886 rows × 1 columns

```
np.any(df.duplicated()) # no duplicate values
```

```
False
```

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

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

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

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

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

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

start date--> 2011-01-01

End date--> 2012-12-19

total of 718 days data

```
df['day'] = df['datetime'].dt.day_name() # adding day column in last to know the day analysis
```

```
df['hour']=df['datetime'].dt.hour
```

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	Saturday	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	Saturday	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	Saturday	2
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	Saturday	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	Saturday	4

Next steps: Generate code with df View recommended plots New interactive sheet

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

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	hour
0	2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	0.0	3	13	16	Saturday	0
1	2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	0.0	8	32	40	Saturday	1
2	2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	0.0	5	27	32	Saturday	2
3	2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	0.0	3	10	13	Saturday	3
4	2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	0.0	0	1	1	Saturday	4

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
```

```
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
 12  day         10886 non-null  object
 13  hour        10886 non-null  int32
dtypes: datetime64[ns](1), float64(3), int32(1), int64(7), object(2)
memory usage: 1.1+ MB
```

```
df[['temp','atemp','humidity','windspeed','casual','registered','count']].describe()
```

	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

```
def workingday(x):
    if x == 1:
        return 'workingday'
    else:
        return 'holiday'
df['day_of_workingday'] = df['workingday'].apply(workingday)
```

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	hour	day_of_workingday
0	2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	0.0	3	13	16	Saturday	0	holiday
1	2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	0.0	8	32	40	Saturday	1	holiday
2	2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	0.0	5	27	32	Saturday	2	holiday
3	2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	0.0	3	10	13	Saturday	3	holiday
4	2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	0.0	0	1	1	Saturday	4	holiday

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	hour	day_of_workingday
0	2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	0.0	3	13	16	Saturday	0	holiday
1	2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	0.0	8	32	40	Saturday	1	holiday
2	2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	0.0	5	27	32	Saturday	2	holiday
3	2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	0.0	3	10	13	Saturday	3	holiday
4	2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	0.0	0	1	1	Saturday	4	holiday

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['holiday'].value_counts()
```

holiday	count
0	10575
1	311

```
df['day_of_workingday'].value_counts()
```

day_of_workingday	count
workingday	7412
holiday	3474

```
vc_of_days=df['day'].value_counts()
vc_of_days.sort_values()
```

day	count
Friday	1529
Tuesday	1539
Monday	1551
Wednesday	1551
Thursday	1553
Sunday	1579
Saturday	1584

```
df['weather'].value_counts()
```

weather	count
1	7192
2	2834
3	859
4	1

```
df.dtypes
```

	0
datetime	datetime64[ns]
season	object
holiday	int64
workingday	int64
weather	int64
temp	float64
atemp	float64
humidity	int64
windspeed	float64
casual	int64
registered	int64
count	int64
day	object
hour	int32
day_of_workingday	object

```
df['holiday'] = df['holiday'].astype('object')
df['workingday'] = df['workingday'].astype('object')
df['weather'] = df['weather'].astype('object')
df['season'] = df['season'].astype('object')
```

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	day	hour	day_of_workingday
0	2011-01-01 00:00:00	spring	0	0	1	9.84	14.395	81	0.0	3	13	16	Saturday	0	holiday
1	2011-01-01 01:00:00	spring	0	0	1	9.02	13.635	80	0.0	8	32	40	Saturday	1	holiday
2	2011-01-01 02:00:00	spring	0	0	1	9.02	13.635	80	0.0	5	27	32	Saturday	2	holiday
3	2011-01-01 03:00:00	spring	0	0	1	9.84	14.395	75	0.0	3	10	13	Saturday	3	holiday
4	2011-01-01 04:00:00	spring	0	0	1	9.84	14.395	75	0.0	0	1	1	Saturday	4	holiday

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df['season'] = df['season'].astype('category')
df['holiday'] = df['holiday'].astype('category')
df['workingday'] = df['workingday'].astype('category')
df['weather'] = df['weather'].astype('category')
df['temp'] = df['temp'].astype('float32')
df['atemp'] = df['atemp'].astype('float32')
```

```
df['humidity'] = df['humidity'].astype('int8')
df['windspeed'] = df['windspeed'].astype('float32')
```

```
df.describe()
```

	datetime	temp	atemp	humidity	windspeed	casual	registered	count	hour
count	10886	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	20.230862	23.655085	61.886460	12.799396	36.021955	155.552177	191.574132	11.541613
min	2011-01-01 00:00:00	0.820000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
25%	2011-07-02 07:15:00	13.940000	16.665001	47.000000	7.001500	4.000000	36.000000	42.000000	6.000000
50%	2012-01-01 20:30:00	20.500000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000	12.000000
75%	2012-07-01 12:45:00	26.240000	31.059999	77.000000	16.997900	49.000000	222.000000	284.000000	18.000000
max	2012-12-19 23:00:00	41.000000	45.455002	100.000000	56.996899	367.000000	886.000000	977.000000	23.000000
std	NaN	7.791600	8.474654	19.245033	8.164592	49.960477	151.039033	181.144454	6.915838

```
df.dtypes
```

	0
datetime	datetime64[ns]
season	category
holiday	category
workingday	category
weather	category
temp	float32
atemp	float32
humidity	int8
windspeed	float32
casual	int64
registered	int64
count	int64
day	object
hour	int32
day_of_workingday	object

dtype: object

Define Problem Statement and perform Exploratory Data Analysis

Definition of problem (as per given problem statement with additional views) Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.

Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

Bivariate Analysis (Relationships between important variables such as workday and count, season and count, weather and count.

Illustrate the **insights based on EDA**

Comments on range of attributes, outliers of various attributes

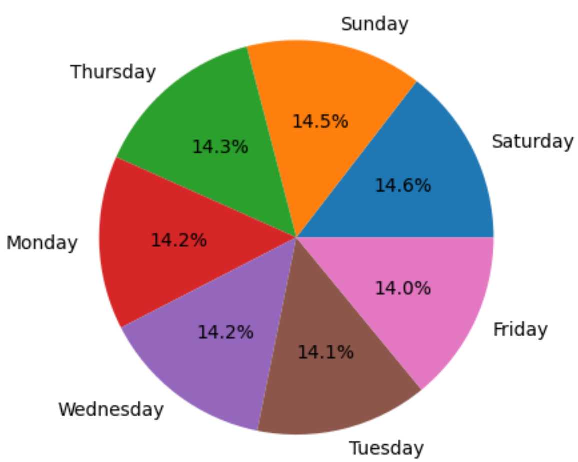
Comments on the distribution of the variables and relationship between them

Comments for each univariate and bivariate plots

UNIVARIATE

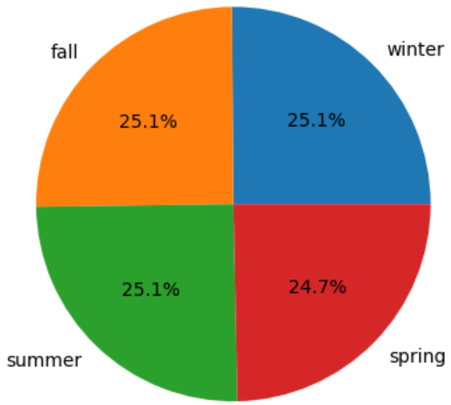
```
plt.pie(df['day'].value_counts(),labels=df['day'].value_counts().index,autopct='%1.1f%%')
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



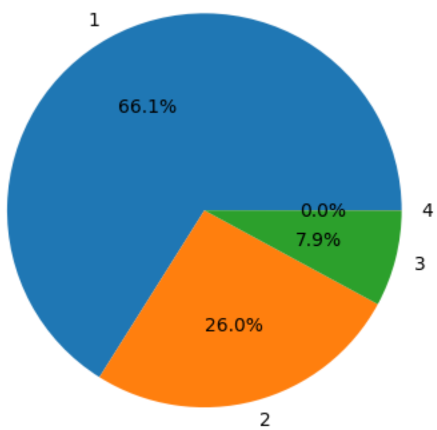

```
plt.pie(df['season'].value_counts(),labels=df['season'].value_counts().index,autopct='%1.1f%%')
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



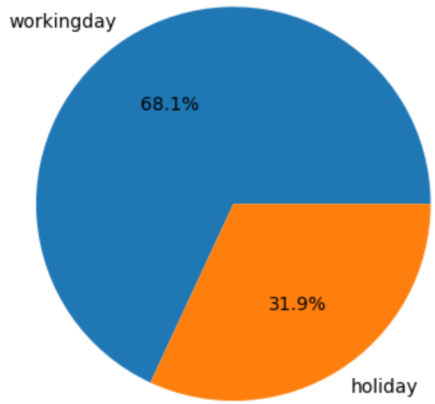

```
plt.pie(df['weather'].value_counts(),labels=df['weather'].value_counts().index,autopct='%1.1f%%')
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



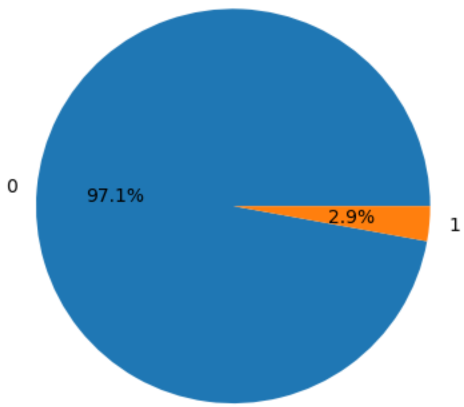

```
plt.pie(df['day_of_workingday'].value_counts(),labels=df['day_of_workingday'].value_counts().index,autopct='%1.1f%%')
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```




```
plt.pie(df['holiday'].value_counts(),labels=df['holiday'].value_counts().index,autopct='%1.1f%%')
plt
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```



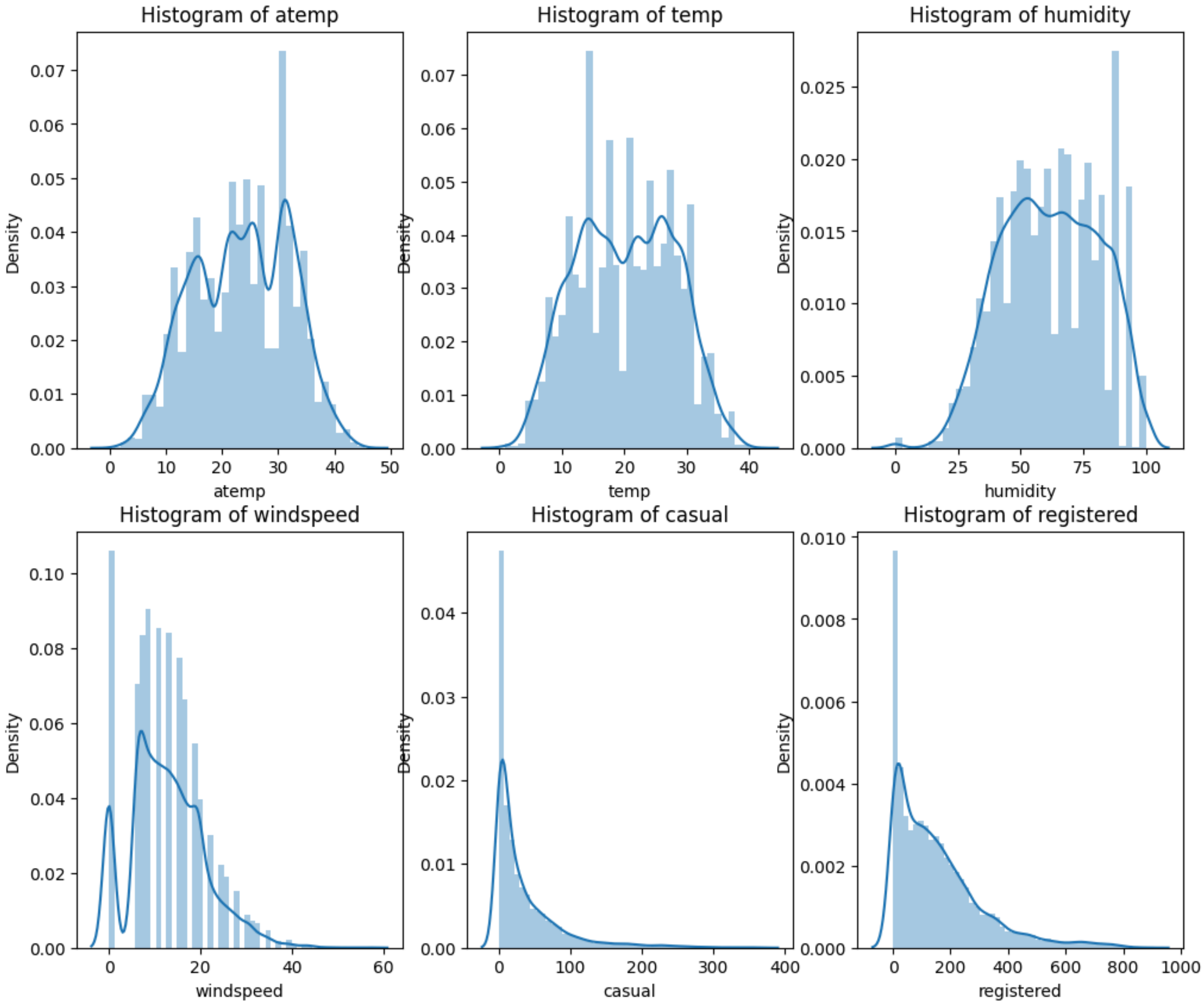

```
plt.figure(figsize=(12,10))
plt.subplot(2,3,1)
sns.distplot(df['atemp'])
plt.title('Histogram of atemp')
plt.subplot(2,3,2)
sns.distplot(df['temp'])
plt.title('Histogram of temp')
plt.subplot(2,3,3)
sns.distplot(df['humidity'])
```

```
plt.title('Histogram of humidity')
plt.subplot(2,3,4)
sns.distplot(df['windspeed'])
plt.title('Histogram of windspeed')
plt.subplot(2,3,5)
sns.distplot(df['casual'])
plt.title('Histogram of casual')
plt.subplot(2,3,6)
sns.distplot(df['registered'])
plt.title('Histogram of registered')
plt.show()
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['atemp'])
<ipython-input-43-27c6edf93127>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['temp'])
<ipython-input-43-27c6edf93127>:9: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['humidity'])
<ipython-input-43-27c6edf93127>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['windspeed'])
<ipython-input-43-27c6edf93127>:15: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['casual'])
<ipython-input-43-27c6edf93127>:18: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```

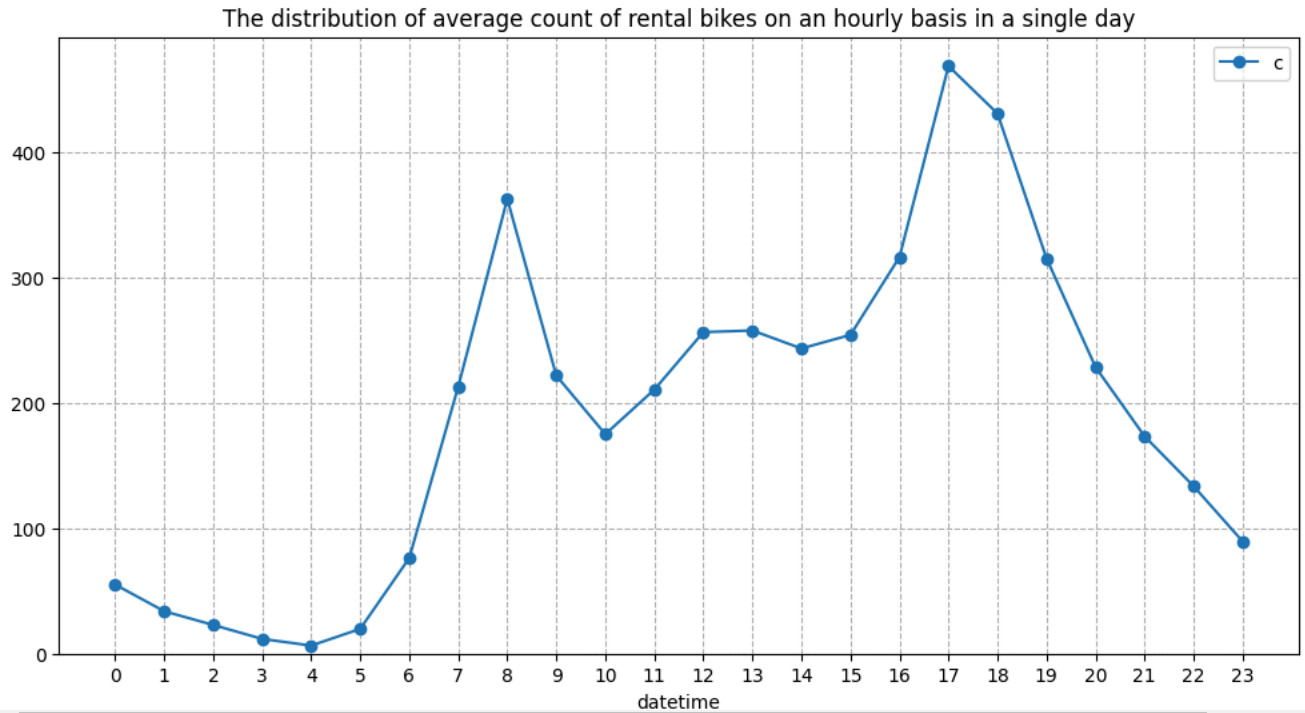
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

```
sns.distplot(df['registered'])
```



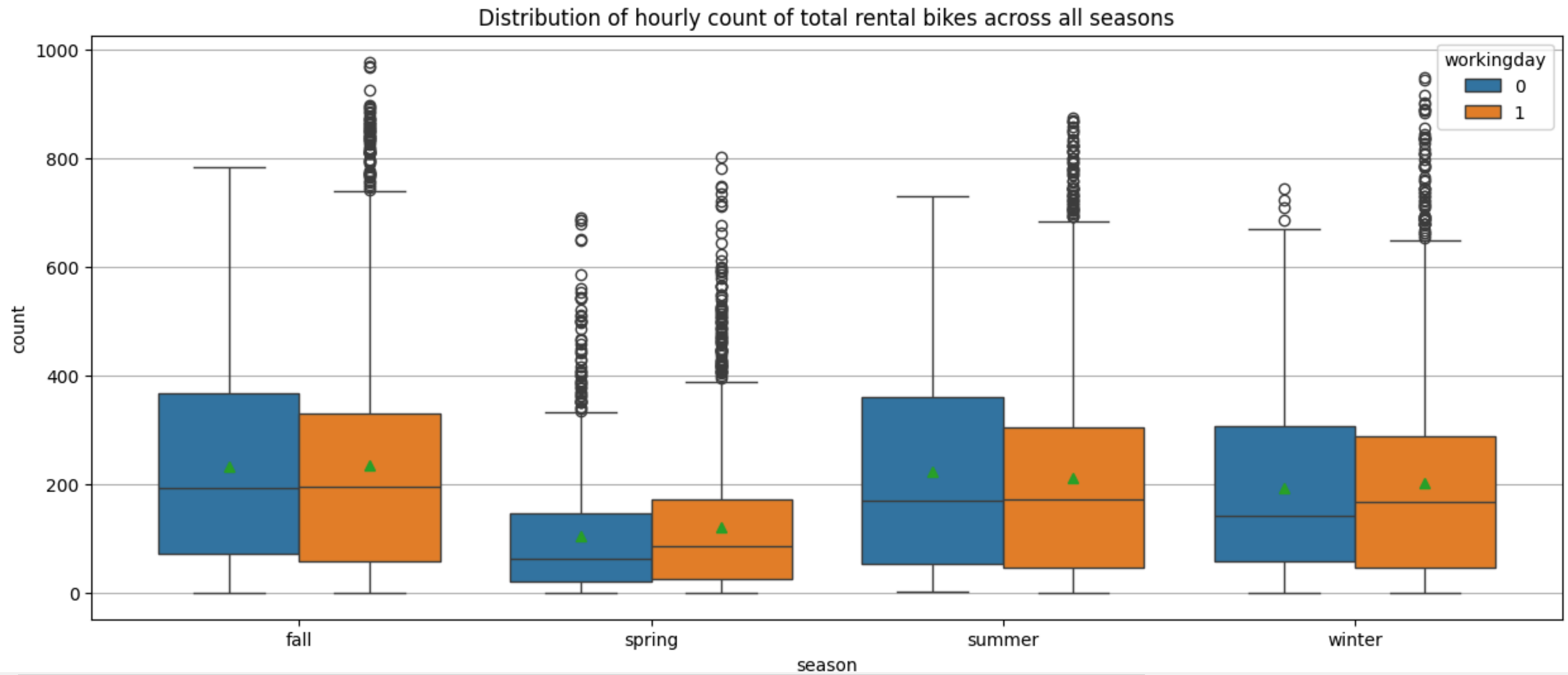
```
plt.figure(figsize = (12, 6))
plt.title("The distribution of average count of rental bikes on an hourly basis in a single day")
df.groupby(by = df['datetime'].dt.hour)['count'].mean().plot(kind = 'line', marker = 'o')
plt.ylim(0,)
plt.xticks(np.arange(0, 24))
plt.legend('count')
plt.grid(axis = 'both',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

```
[]
```




```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across all seasons')
sns.boxplot(data = df, x = 'season', y = 'count', hue = 'workingday', showmeans = True)
plt.grid(axis = 'y', linestyle = '-')
plt.plot()
```

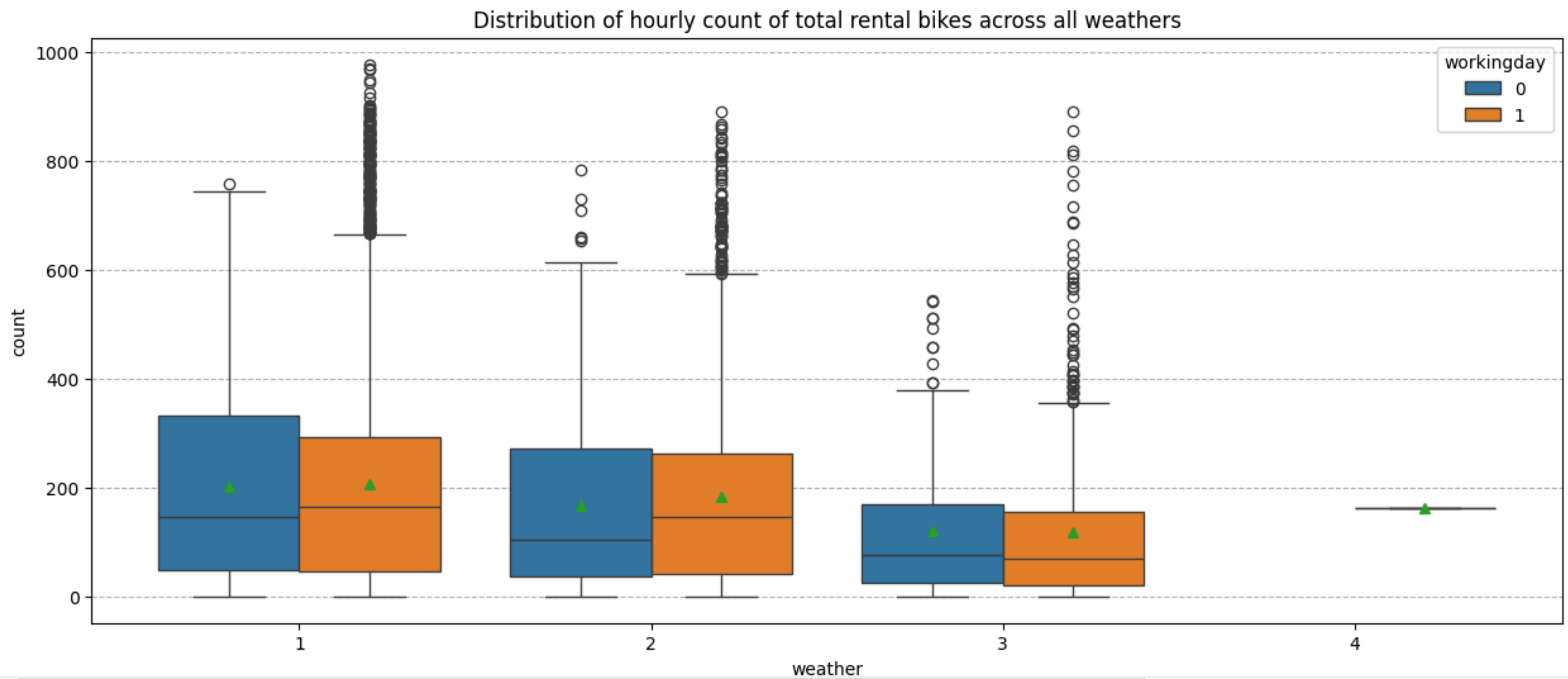
[]



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

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across all weathers')
sns.boxplot(data = df, x = 'weather', y = 'count', hue = 'workingday', showmeans = True)
plt.grid(axis = 'y', linestyle = '--')
plt.plot()
```

[]



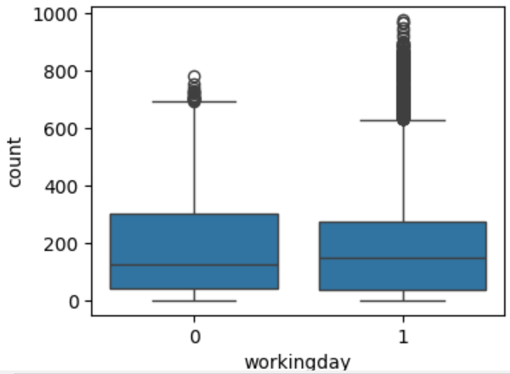
The hourly count of total rental bikes is higher in the clear and cloudy weather, followed by the misty weather and rainy weather. There are very few records for extreme weather conditions.

if Working Day has an effect on the number of electric cycles rented**

```
# To do this we need to know the count of working day
df.groupby(by = 'workingday')['count']
```

```
<ipython-input-47-ee01f37e91f1>:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain
  df.groupby(by = 'workingday')['count']
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7d7d6cfc2560>
```

```
plt.figure(figsize = (4, 3))
sns.boxplot(data = df, x = 'workingday', y = 'count')
plt.plot()
```



Step-1

Setup Null Hypothesis

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

.Alternate Hypothesis(Ha):- Working day does have effect on number of electric cycles rented.

Step-2

Chceking homogentiy if variance and how the data is distributed using **QQ plot, Levene's Test**

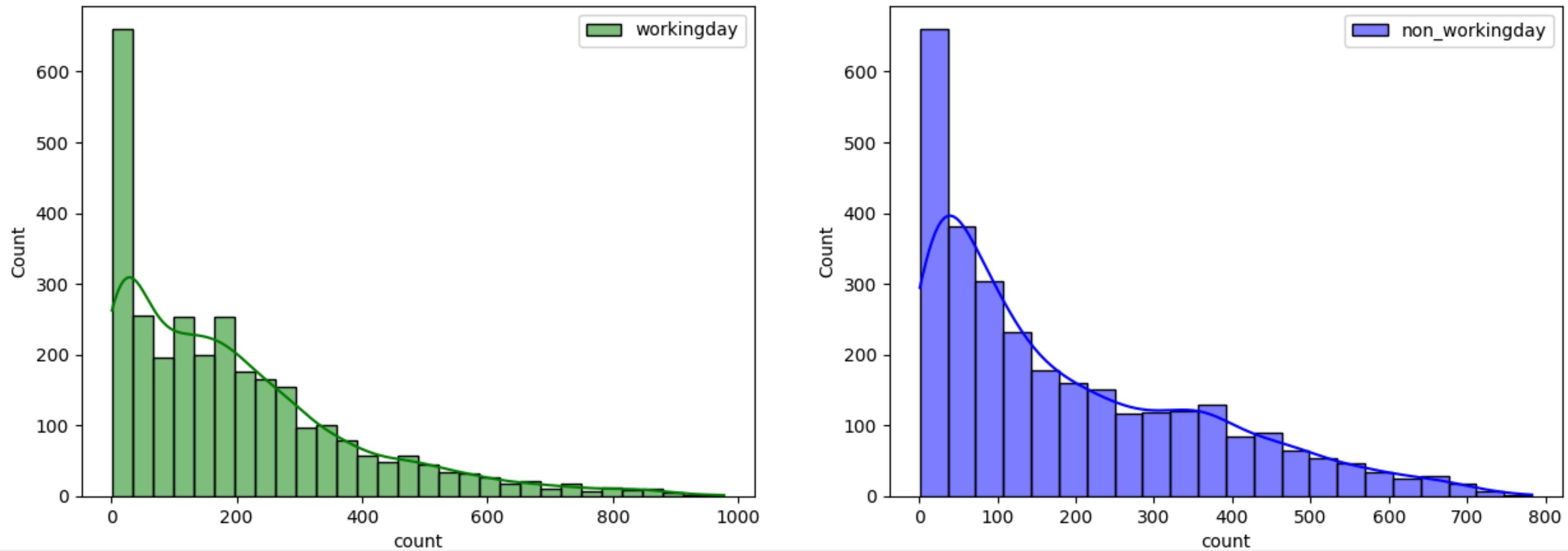
step-3 Define test statistics

step-4 Compute **P-value** and fix value of **alpha** and compare them

step-5 Based on comparision we need to know to **reject H0 or fail to reject H0**

```
# lets check if samples follow normal distribution
plt.figure(figsize = (15, 5))
plt.subplot(1, 2, 1)
sns.histplot(df.loc[df['workingday'] == 1, 'count'].sample(3000),
            color = 'green', kde = True, label = 'workingday')
plt.legend()
plt.subplot(1, 2, 2)
sns.histplot(df.loc[df['workingday'] == 0, 'count'].sample(3000),
             color = 'blue', kde = True, label = 'non_workingday')
plt.legend()
plt.plot()
```

[]

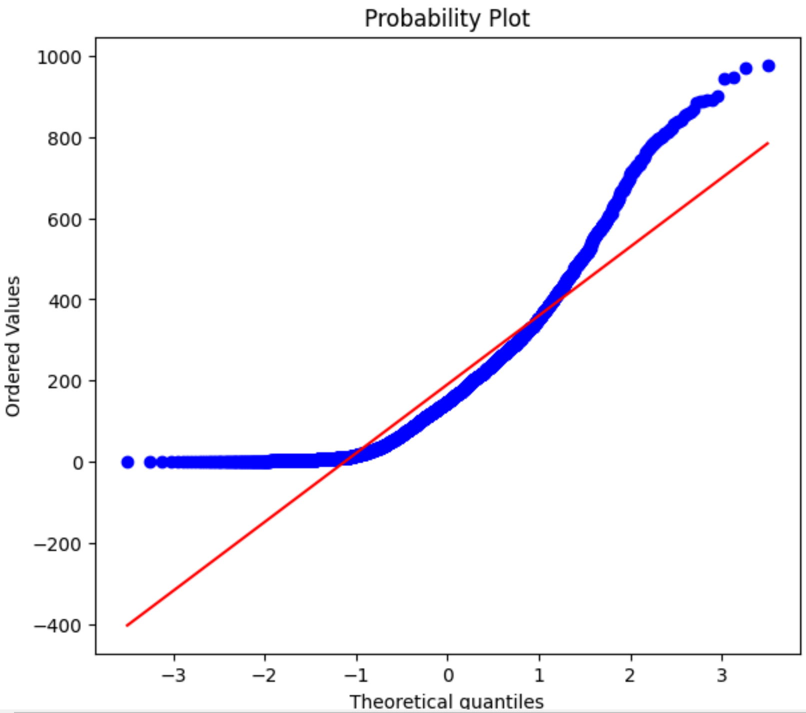


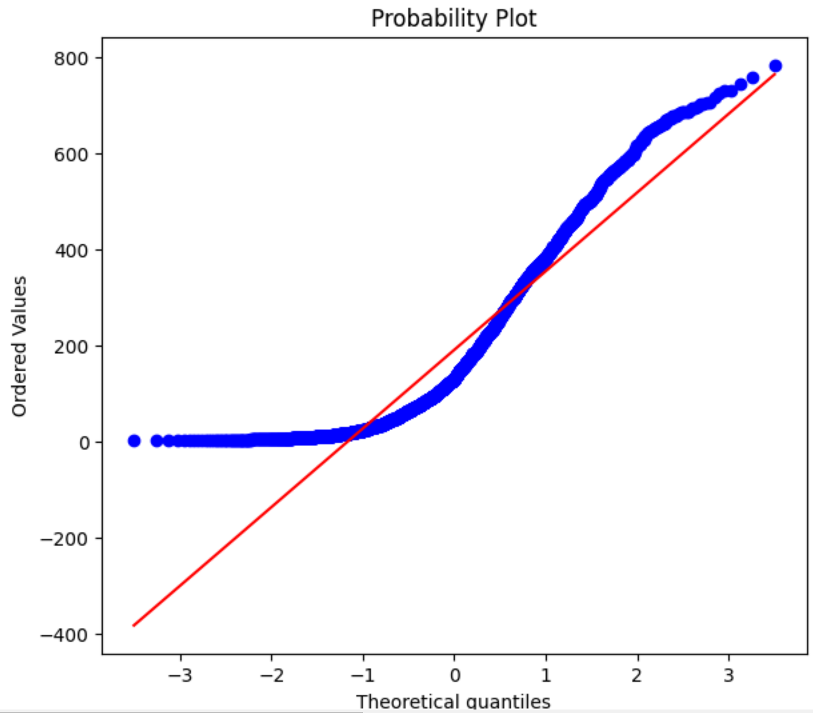
*Not normally distributed *

How ever lets us check the distribution with QQ plot to confirm the how thedistribution is done

```
plt.figure(figsize = (15, 6))
plt.subplot(1, 2, 1)
spy.probplot(df.loc[df['workingday'] == 1, 'count'].sample(3000), plot = plt, dist = 'norm')
plt.subplot(1, 2, 2)
spy.probplot(df.loc[df['workingday'] == 0, 'count'].sample(3000), plot = plt, dist = 'norm')
plt.plot()
```

[]





We can confirm that it does not follow normal distribution

Let's try applying shapiro wilk test

H0- sample follows normal distribution

H1- sample doesn't follow nomral distribution

alpha=0.05

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] == 1, 'count'].sample(3000))
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 2.06748443654708O2e-45
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(df.loc[df['workingday'] == 0, 'count'].sample(3000))
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 2.0152945810217397e-42
The sample does not follow normal distribution
```

** Each of the "workingday" and "non_workingday" data, the samples do not follow normal distribution.**

Let's check homogenity of varinace

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(df.loc[df['workingday'] == 1, 'count'].sample(3000),
                                df.loc[df['workingday'] == 0, 'count'].sample(3000))
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 0.622131460359391
The samples have Homogenous Variance
```

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

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

test_stat, p_value = spy.mannwhitneyu(df.loc[df['workingday'] == 1, 'count'],
                                      df.loc[df['workingday'] == 0, 'count'])
print('P-value :',p_value)
```

```
P-value : 0.9679139953914079
```

Has P-value is greater than 0.05 we fail to reject null hypothesis

So there is no significant diffirence in rent of electric cycles in wokring and non-wokring days

3b) No. of cycles rented similar or different in different seasons?

Step-1

Setup Null Hypothesis

.**Null Hypothesis(H0)**:- seasons does not have any effect on the number of electric cycles rented

.**Alternate Hypothesis(Ha)**:- seasons does have effect on number of electric cycles rented.

Step-2

Chceking homogentiy if variance and how the data is distributed using **QQ plot, Levene's Test**

step-3 Define test statistics

step-4 Compute **P-value** and fix value of **alpha** and compare them

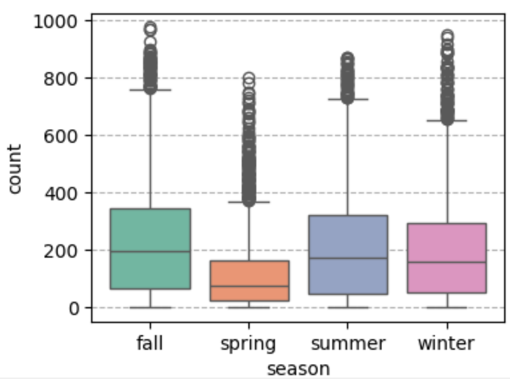
step-5 Based on comparision we need to know to** reject H0 or fail to reject H0**

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

```
<ipython-input-55-2aa6330c0f18>:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain
  df.groupby(by = 'season')['count']
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7d7d6cc2f3a0>
```

```
plt.figure(figsize = (4, 3))
sns.boxplot(data = df, x = 'season', y = 'count',palette="Set2")
plt.grid(axis = 'y', linestyle = '--')
plt.plot()
```

```
<ipython-input-56-af80bce12084>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data = df, x = 'season', y = 'count',palette="Set2")
[]
```



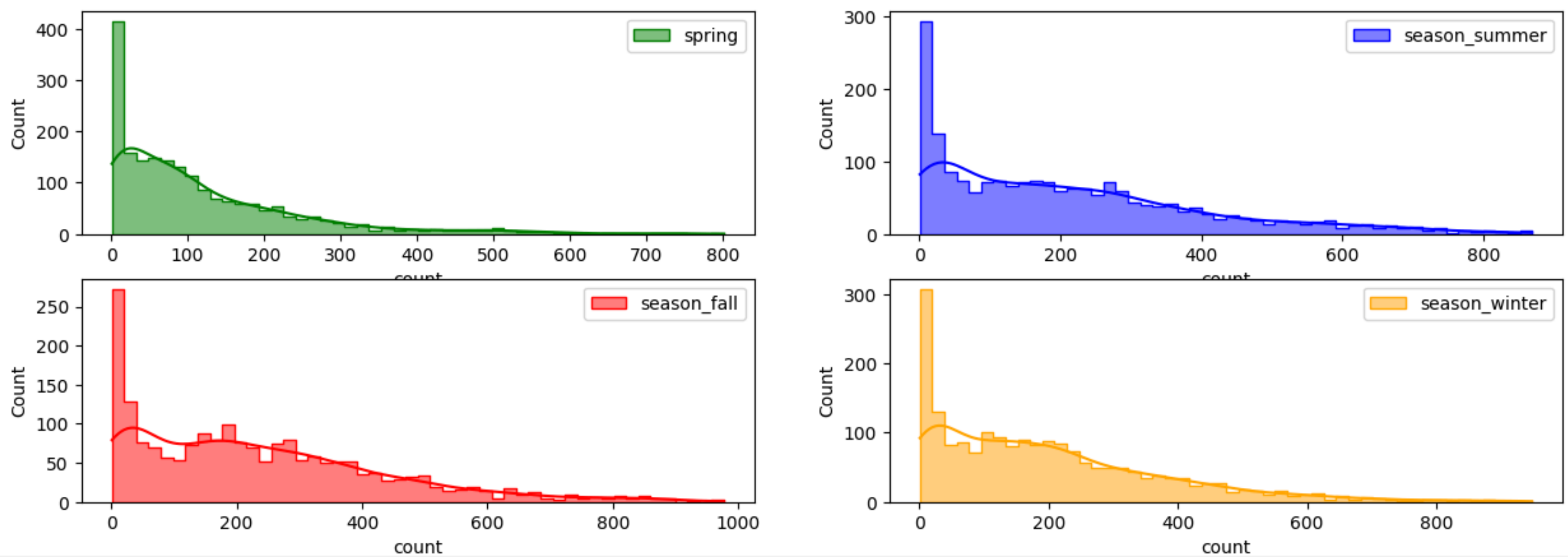
most of them were rented in fall, follwed by summer and winter and least rented during spring.

```
spring=df.loc[df['season']=='spring','count']
summer=df.loc[df['season']=='summer','count']
fall=df.loc[df['season']=='fall','count']
winter=df.loc[df['season']=='winter','count']
```

```
# lets check if samples follow normal distribution
plt.figure(figsize = (15, 5))
plt.subplot(2, 2, 1)
```

```
sns.histplot(spring.sample(2000),bins=50,element='step' ,
            color = 'green', kde = True, label = 'spring')
plt.legend()
plt.subplot(2, 2, 2)
sns.histplot(summer.sample(2000), bins = 50,
             element = 'step', color = 'blue', kde = True, label = 'season_summer')
plt.legend()
plt.subplot(2, 2, 3)
sns.histplot(fall.sample(2000), bins = 50,
             element = 'step', color = 'red', kde = True, label = 'season_fall')
plt.legend()
plt.subplot(2, 2, 4)
sns.histplot(winter.sample(2000), bins = 50,
             element = 'step', color = 'orange', kde = True, label = 'season_winter')
plt.legend()
plt.plot()
```

[]



seems like it right tailed not normally distrbuted.

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

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

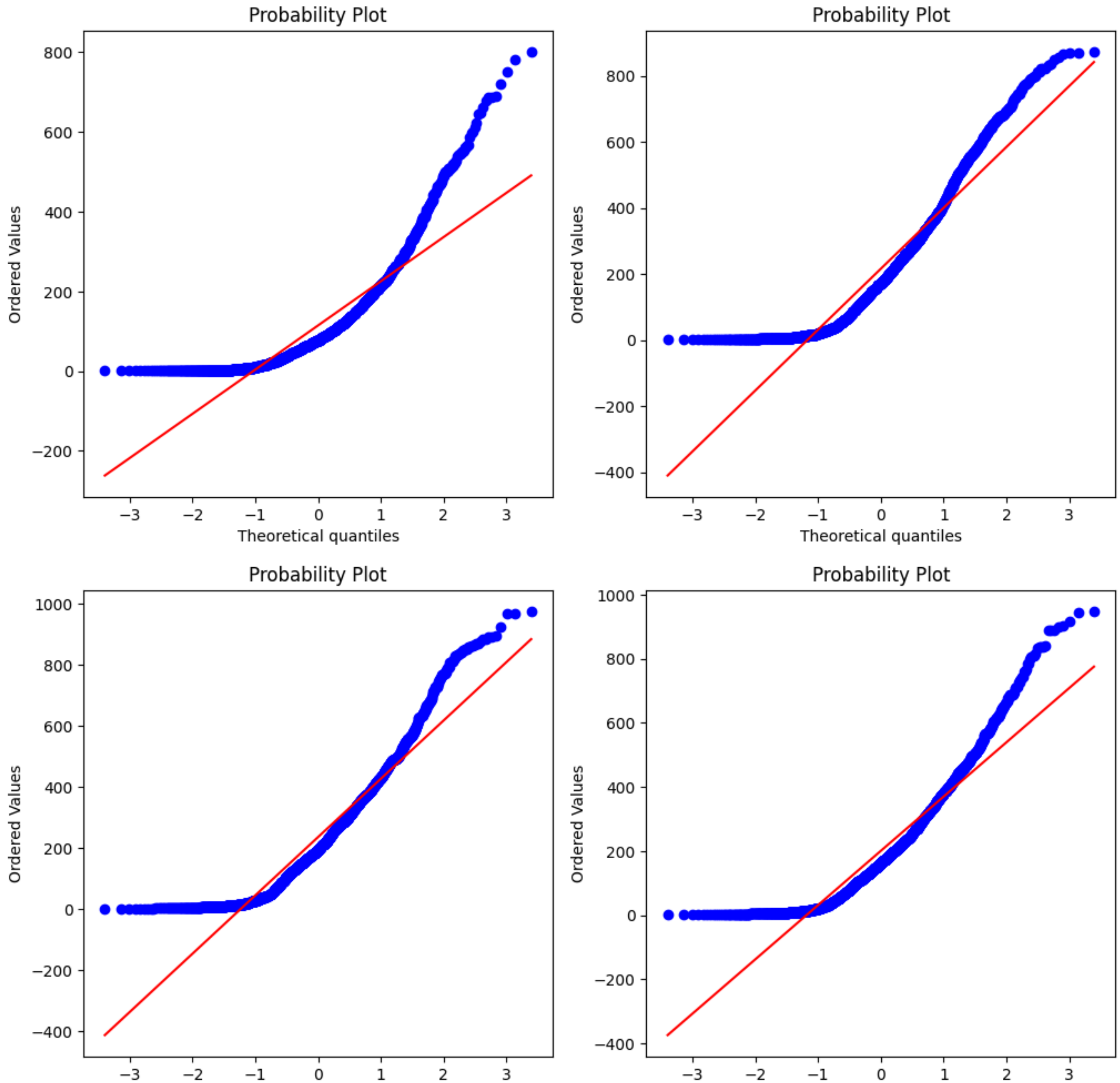
where, μ = group mean and k = number of groups.

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.

QQ PLOT

```
plt.figure(figsize = (12, 12))
plt.subplot(2, 2, 1)
plt.suptitle('QQ plots for the count of electric vehicles rented in different seasons')
spy.probplot(spring.sample(2000), plot = plt, dist = 'norm')

plt.subplot(2, 2, 2)
spy.probplot(summer.sample(2000), plot = plt, dist = 'norm')


plt.subplot(2, 2, 3)
spy.probplot(fall.sample(2000), plot = plt, dist = 'norm')

plt.subplot(2, 2, 4)
spy.probplot(winter.sample(2000), plot = plt, dist = 'norm')

plt.plot()
```

QQ plots for the count of electric vehicles rented in different seasons



Theoretical quantiles Theoretical quantiles

seems like nothing nomrally distrbuted.

lets apply shapiro-wilk test for normality

```
test_stat, p_value = spy.shapiro(spring.sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 2.9479235808246546e-43
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(summer.sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 9.291564831846448e-34
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(fall.sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 5.749169021484996e-32
The sample does not follow normal distribution
```

```
test_stat, p_value = spy.shapiro(winter.sample(2000))
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```

```
p-value 9.405475804949815e-35
The sample does not follow normal distribution
```

Homogeneity of Variances using Levene's test

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

# Alternate Hypothesis(HA) - Non Homogenous Variance

test_stat, p_value = spy.levene(spring.sample(2000),
                                summer.sample(2000),
```

```
                                fall.sample(2000),
                                winter.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 6.519198995319998e-83
The samples do not have  Homogenous Variance
```

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

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

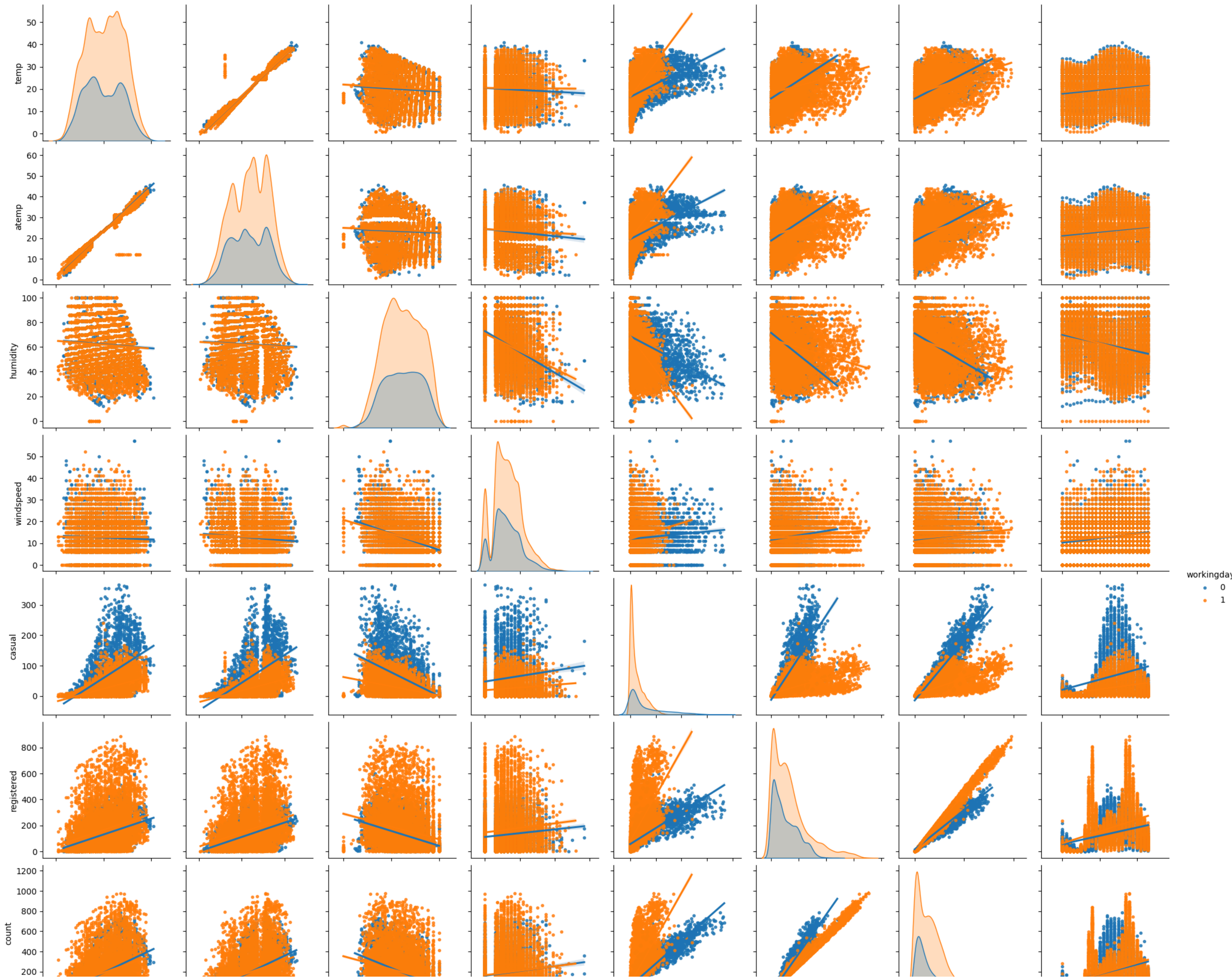
```
Test Statistic = 699.6668548181988
p value = 2.479008372608633e-151
```

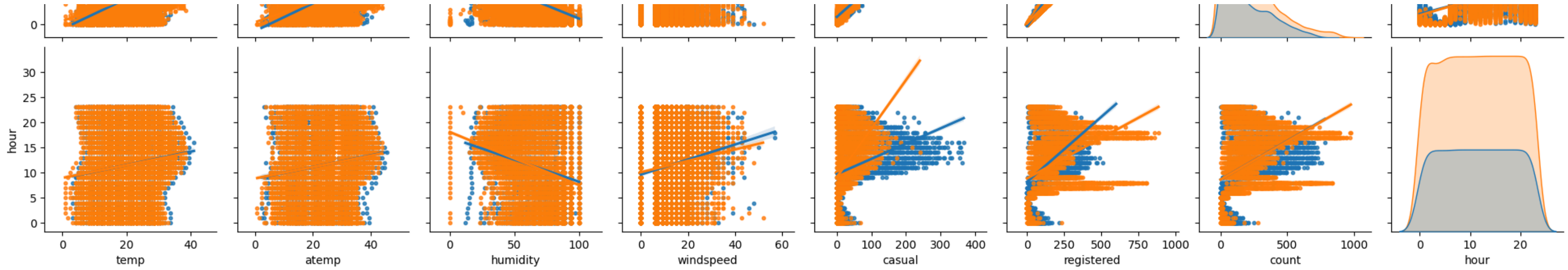
```
if p_value < alpha:
    print('Reject Null Hypothesis')
else:
    print('Failed to reject Null Hypothesis')
```

```
Reject Null Hypothesis
```

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

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


temp
atemp
humidity
windspeed
casual
registered
count
workingday
0
1

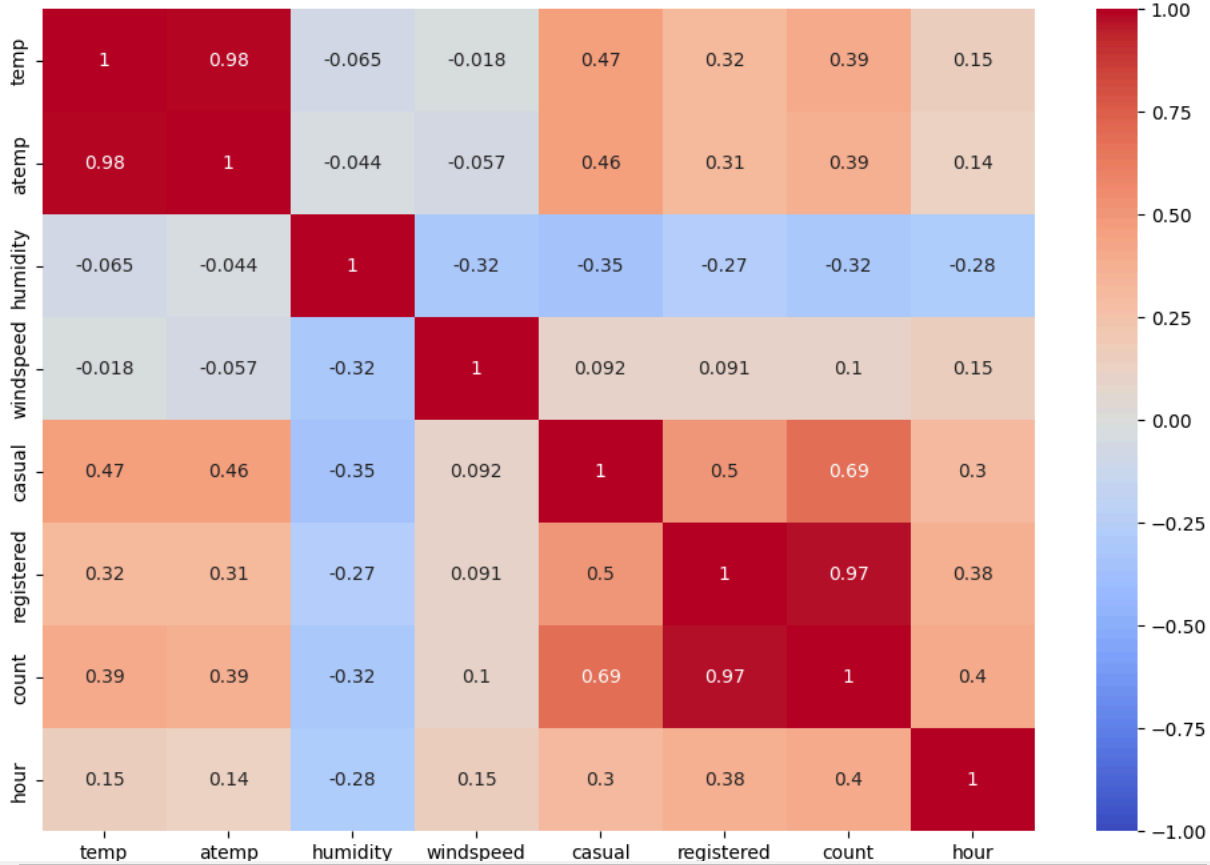

hour
temp
atemp
humidity
windspeed
casual
registered
count
hour

```
numerical_df = df.select_dtypes(include=['number'])
corr_data = numerical_df.corr()

corr_data
```

	temp	atemp	humidity	windspeed	casual	registered	count	hour
temp	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.394454	0.145430
atemp	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.389784	0.140343
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.317371	-0.278011
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.101369	0.146631
casual	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.690414	0.302045
registered	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.970948	0.380540
count	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.000000	0.400601
hour	0.145430	0.140343	-0.278011	0.146631	0.302045	0.380540	0.400601	1.000000

Next steps: Generate code with `corr_data` View recommended plots New interactive sheet

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



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],['hour,registred],[hour,count]

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.

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

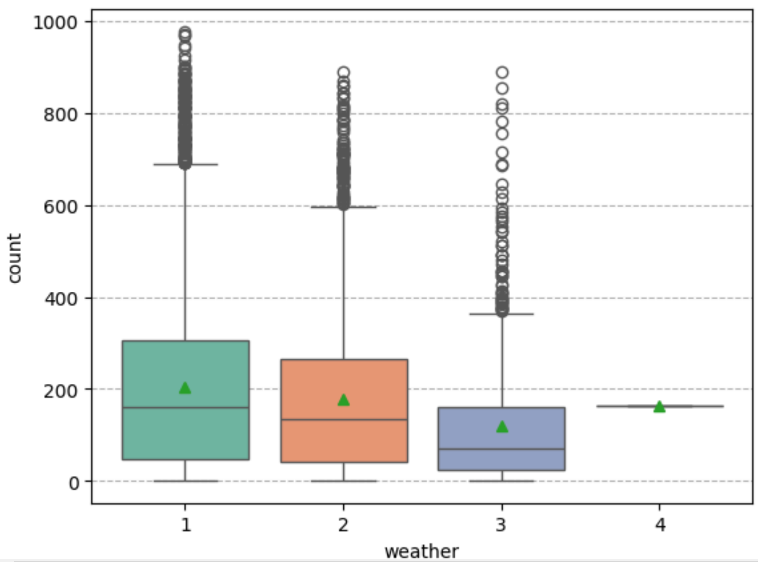
```
df.groupby(by = 'weather')['count'].describe()
```

```
<ipython-input-70-85159499a46f>:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain
  df.groupby(by = 'weather')['count'].describe()
```

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

```
sns.boxplot(data = df, x = 'weather', y = 'count', showmeans = True,palette="Set2")
plt.grid(axis = 'y', linestyle = '--')
plt.plot()
```

```
<ipython-input-71-7760d1a0f4f6>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(data = df, x = 'weather', y = 'count', showmeans = True,palette="Set2")
[]
```



we have only one datapoint for weather 4 so we cant perform any test there so eliminating weather 4 from dataset

```
weather1 = df.loc[df['weather'] == 1]
weather2 = df.loc[df['weather'] == 2]
weather3 = df.loc[df['weather'] == 3]
```

Step-1

Setup Null Hypothesis

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

.Alternate Hypothesis(Ha):- weather does have effect on number of electric cycles rented.

Step-2

Chceking homogentiy if variance and how the data is distributed using QQ plot, Levene's Test

step-3 Define test statistics

step-4 Compute P-value and fix value of alpha and compare them

step-5 Based on comparision we need to know to** reject H0 or fail to reject H0**

```
plt.figure(figsize = (15, 5))
plt.subplot(1, 3, 1)
sns.histplot(weather1.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'green', kde = True, label = 'weather1')
plt.legend()
plt.subplot(1, 3, 2)
sns.histplot(weather2.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'red', kde = True, label = 'weather2')
plt.legend()
plt.subplot(1, 3, 3)
sns.histplot(weather3.loc[:, 'count'].sample(500), bins = 40,
             element = 'step', color = 'blue', kde = True, label = 'weather3')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7d7d69add060>
```