YULU - Business CaseStudy

Hypothesis Testing

Features of the dataset:

- Column Profiling:

Feature	Description
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.
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 - Total_riders	count of total rental bikes including both casual and registered

- weather

Category	Details
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

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind,chi2_contingency,shapiro,ttest_rel,levene,kruskal
from statsmodels.stats.proportion import proportions_ztest
```

```
# reading csv file
data = pd.read_csv('bike_sharing.csv')
```

```
# first 5 rows of data
data.head()
```

	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 data | View recommended plots | New interactive sheet

```
# create a copy of data
df =data.copy()
```

```
# checking the dimension of dataframe
df.ndim
```

```
2
```

```
# checking the shape of dataframe
df.shape
```

```
(10886, 12)
```

```
# all columns in our dataframe
df.columns
```

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

```
# Cheking whether there is NULL values or not.
#  checking the datatype of each series
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
```

```
# converting object column to date and time for finding date related features
df['datetime'] = pd.to_datetime(df['datetime'])
```

```
# after convertion check
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  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: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

```
# finding most recent and oldest date in df
df['datetime'].max().date(),df['datetime'].min().date()
```

```
(datetime.date(2012, 12, 19), datetime.date(2011, 1, 1))
```

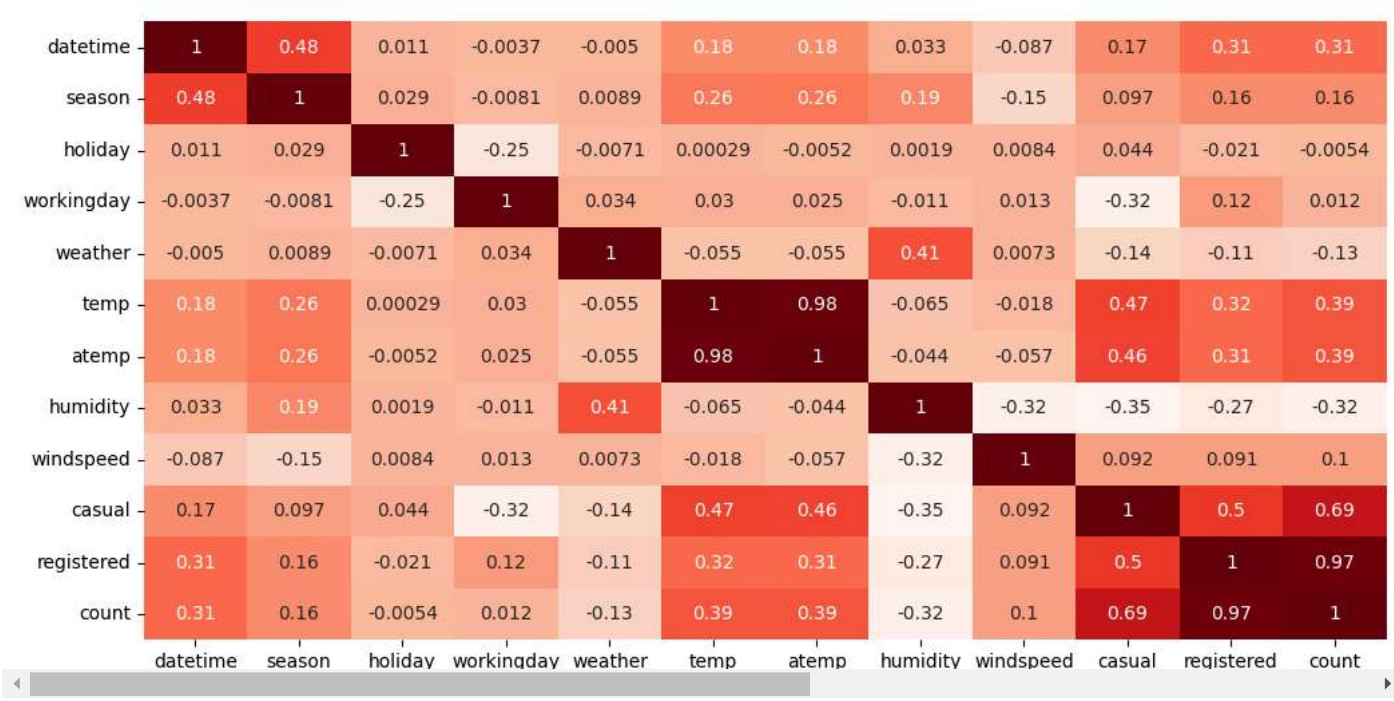
```
# Oldest and latest data
min_date=df["datetime"].min().date()
max_date=df["datetime"].max().date()

print("oldest date=",min_date,'\n')
print("Latest date=",max_date,'\n')
```

```
oldest date= 2011-01-01

Latest date= 2012-12-19
```

```
# correlation among all variables
df.corr()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
datetime	1.000000	0.480021	0.010988	-0.003658	-0.005048	0.180986	0.181823	0.032856	-0.086888	0.172728	0.314879	0.310187
season	0.480021	1.000000	0.029368	-0.008126	0.008879	0.258689	0.264744	0.190610	-0.147121	0.096758	0.164011	0.163439
holiday	0.010988	0.029368	1.000000	-0.250491	-0.007074	0.000295	-0.005215	0.001929	0.008409	0.043799	-0.020956	-0.005393
workingday	-0.003658	-0.008126	-0.250491	1.000000	0.033772	0.029966	0.024660	-0.010880	0.013373	-0.319111	0.119460	0.011594
weather	-0.005048	0.008879	-0.007074	0.033772	1.000000	-0.055035	-0.055376	0.406244	0.007261	-0.135918	-0.109340	-0.128655
temp	0.180986	0.258689	0.000295	0.029966	-0.055035	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.394454
atemp	0.181823	0.264744	-0.005215	0.024660	-0.055376	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.389784
humidity	0.032856	0.190610	0.001929	-0.010880	0.406244	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.317371
windspeed	-0.086888	-0.147121	0.008409	0.013373	0.007261	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.101369
casual	0.172728	0.096758	0.043799	-0.319111	-0.135918	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.690414
registered	0.314879	0.164011	-0.020956	0.119460	-0.109340	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.970948
count	0.310187	0.163439	-0.005393	0.011594	-0.128655	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.000000

```
# creating correlation heatmap
plt.figure(figsize=(15,6))
sns.heatmap(df.corr(),annot=True,cmap='Reds')
plt.show()
```



Observations

1. The correlation between "temp" and "atemp" is very high (0.985), indicating an almost perfect positive correlation. This suggests that these two variables are highly correlated and likely convey very similar information. It might be redundant to include both in certain analyses.
2. There is a positive correlation (0.259) between "season" and "temp." This suggests that as the season changes, there might be variations in temperature. For instance, certain seasons might be associated with higher or lower temperatures.
3. There is a positive correlation (0.119) between "workingday" and the number of "registered" users. This suggests that on working days, there may be more registered users utilizing the service compared to non-working days.
4. There is a positive correlation (0.406) between "weather" and "humidity." This suggests that higher values of the "weather" variable (potentially indicating worse weather conditions) are associated with higher humidity.

5. Both "casual" and "registered" users show strong positive correlations with the total "count" of users (0.690 for casual and 0.971 for registered). This indicates that both types of users significantly contribute to the overall user count.
6. Negative correlations are observed between "holiday" and "workingday" (-0.250) and between "humidity" and "windspeed" (-0.319). This suggests that these variables are inversely related. For example, holidays might be less likely to be working days, and higher humidity might be associated with lower windspeed.

```
# Changing datatypes of categorical columns
categorical_columns =['season','holiday','workingday','weather']

for col in categorical_columns:
  df[col] = df[col].astype('object')
```

Double-click (or enter) to edit

```
df.describe(include ='object')
```

	season	holiday	workingday	weather
count	10886	10886	10886	10886
unique	4	2	2	4
top	4	0	1	1
freq	2734	10575	7412	7192

```
df.describe()
```

	datetime	temp	atemp	humidity	windspeed	casual	registered	count
count	10886	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
min	2011-01-01 00:00:00	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2011-07-02 07:15:00	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	2012-01-01 20:30:00	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	2012-07-01 12:45:00	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	2012-12-19 23:00:00	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000
std	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454

```
# checking for null values
df.isna().sum()
```

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

```
# checking for duplicate values
df.duplicated().sum()
```

```
0
```

```
# distribution across the categorical variables
categorical_columns =['season','holiday','workingday','weather']

for x in categorical_columns:
 print(df[x].value_counts(),'\n')
 print('-'*20)
```

```
season
4    2734
2    2733
3    2733
1    2686
Name: count, dtype: int64

--------------------
holiday
0    10575
1      311
Name: count, dtype: int64

--------------------
workingday
1    7412
0    3474
Name: count, dtype: int64

--------------------
weather
1    7192
2    2834
3     859
4       1
Name: count, dtype: int64

--------------------
```

Observation

1. Almost similar hours of data for each season.
2. Each weather has different no of hours.
3. It is obvious that working day has more hours than non-working days

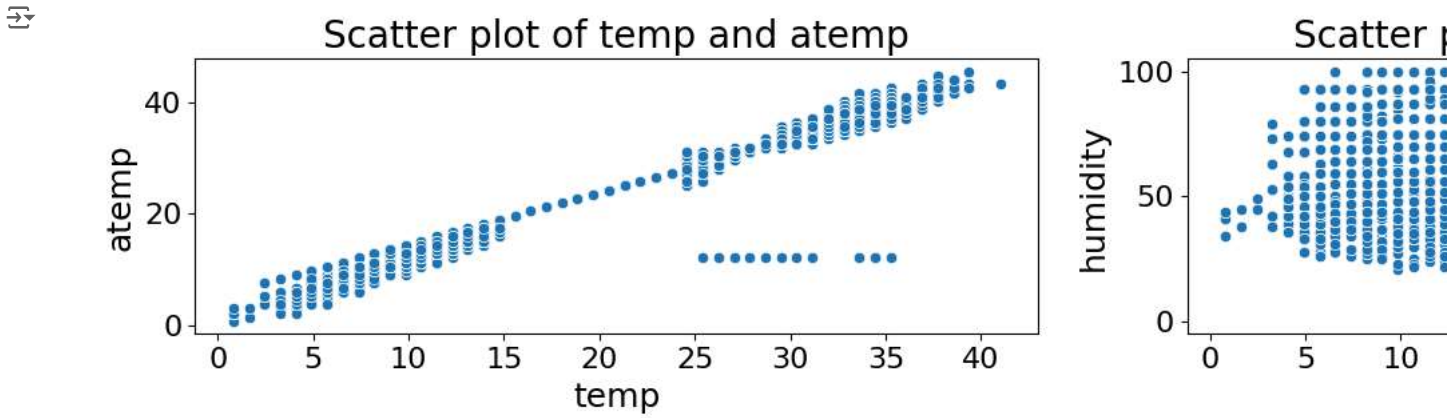
Univariate and Bivariate Graphical Analysis

** - Bivariate Analysis of Numerical Variables**

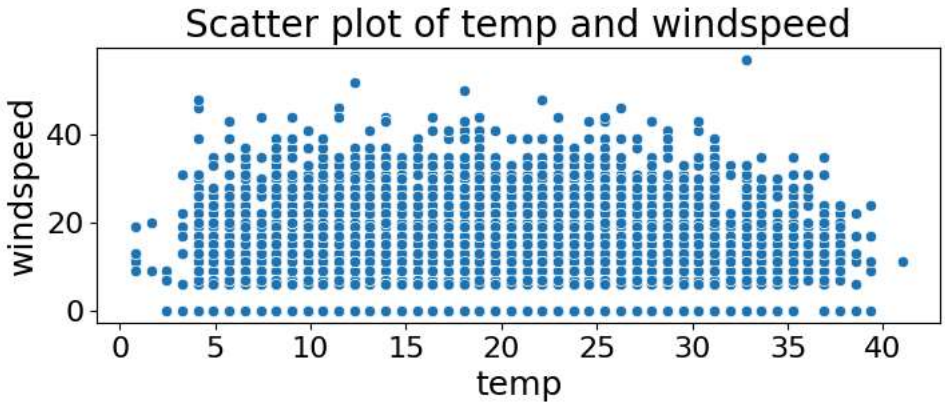
```
# numerical and categorical columns
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']
categorical_cols = ['season', 'holiday', 'workingday', 'weather']


# distribution of numerical variables using scatterplot

plt.figure(figsize=(15, 25))
plt.subplots_adjust(wspace=0.5,hspace=0.5)

# Scatter plot for 'temp' and 'atemp'
sub1 = plt.subplot(8, 2, 1)
sns.scatterplot(x=df['temp'], y=df['atemp'])
sub1.set_title('Scatter plot of temp and atemp', fontsize=20)
sub1.set_xlabel('temp', fontsize=18)
sub1.set_ylabel('atemp', fontsize=18)
sub1.tick_params(axis='both', labelsize=16)
```

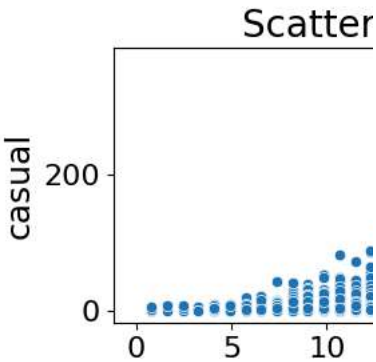
```
# Scatter plot for 'temp' and 'humidity'
sub2 = plt.subplot(8, 2, 2)
sns.scatterplot(x=df['temp'], y=df['humidity'])
sub2.set_title('Scatter plot of temp and humidity', fontsize=20)
sub2.set_xlabel('temp', fontsize=18)
sub2.set_ylabel('humidity', fontsize=18)
sub2.tick_params(axis='both', labelsize=16)

# Scatter plot for 'temp' and 'windspeed'
sub3 = plt.subplot(8, 2, 3)
sns.scatterplot(x=df['temp'], y=df['windspeed'])
sub3.set_title('Scatter plot of temp and windspeed', fontsize=20)
sub3.set_xlabel('temp', fontsize=18)
sub3.set_ylabel('windspeed', fontsize=18)
sub3.tick_params(axis='both', labelsize=16)

# Scatter plot for 'temp' and 'casual'
sub4 = plt.subplot(8, 2, 4)
sns.scatterplot(x=df['temp'], y=df['casual'])
sub4.set_title('Scatter plot of temp and casual', fontsize=20)
sub4.set_xlabel('temp', fontsize=18)
sub4.set_ylabel('casual', fontsize=18)
sub4.tick_params(axis='both', labelsize=16)

# Scatter plot for 'temp' and 'registered'
sub5 = plt.subplot(8, 2, 5)
sns.scatterplot(x=df['temp'], y=df['registered'])
sub5.set_title('Scatter plot of temp and registered', fontsize=20)
sub5.set_xlabel('temp', fontsize=18)
sub5.set_ylabel('registered', fontsize=18)
sub5.tick_params(axis='both', labelsize=16)

# Scatter plot for 'temp' and 'count'
sub6 = plt.subplot(8, 2, 6)
sns.scatterplot(x=df['temp'], y=df['count'])
sub6.set_title('Scatter plot of temp and count', fontsize=20)
sub6.set_xlabel('temp', fontsize=18)
sub6.set_ylabel('count', fontsize=18)
sub6.tick_params(axis='both', labelsize=16)

# Scatter plot for 'humidity' and 'windspeed'
sub7 = plt.subplot(8, 2, 7)
sns.scatterplot(x=df['humidity'], y=df['windspeed'])
sub7.set_title('Scatter plot of humidity and windspeed', fontsize=20)
sub7.set_xlabel('humidity', fontsize=18)
sub7.set_ylabel('windspeed', fontsize=18)
sub7.tick_params(axis='both', labelsize=16)

# Scatter plot for 'humidity' and 'casual'
sub8 = plt.subplot(8, 2, 8)
sns.scatterplot(x=df['humidity'], y=df['casual'])
sub8.set_title('Scatter plot of humidity and casual', fontsize=20)
sub8.set_xlabel('humidity', fontsize=18)
sub8.set_ylabel('casual', fontsize=18)
sub8.tick_params(axis='both', labelsize=16)

# Scatter plot for 'humidity' and 'registered'
sub9 = plt.subplot(8, 2, 9)
sns.scatterplot(x=df['humidity'], y=df['registered'])
sub9.set_title('Scatter plot of humidity and registered', fontsize=20)
sub9.set_xlabel('humidity', fontsize=18)
sub9.set_ylabel('registered', fontsize=18)
sub9.tick_params(axis='both', labelsize=16)

# Scatter plot for 'humidity' and 'count'
sub10 = plt.subplot(8, 2, 10)
sns.scatterplot(x=df['humidity'], y=df['count'])
sub10.set_title('Scatter plot of humidity and count', fontsize=20)
sub10.set_xlabel('humidity', fontsize=18)
sub10.set_ylabel('count', fontsize=18)
sub10.tick_params(axis='both', labelsize=16)

# Scatter plot for 'windspeed' and 'casual'
sub11 = plt.subplot(8, 2, 11)
sns.scatterplot(x=df['windspeed'], y=df['casual'])
sub11.set_title('Scatter plot of windspeed and casual', fontsize=20)
sub11.set_xlabel('windspeed', fontsize=18)
```

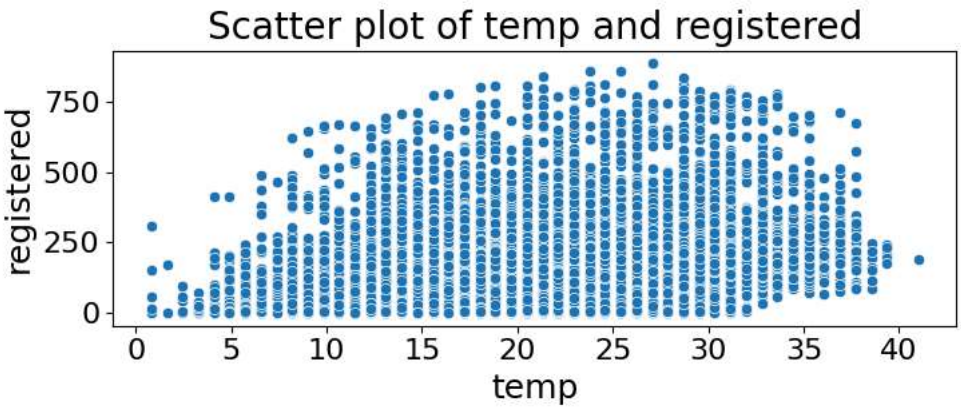
```
sub11.set_ylabel('casual', fontsize=18)
sub11.tick_params(axis='both', labelsize=16)

# Scatter plot for 'windspeed' and 'registered'
sub12 = plt.subplot(8, 2, 12)
sns.scatterplot(x=df['windspeed'], y=df['registered'])
sub12.set_title('Scatter plot of windspeed and registered', fontsize=20)
sub12.set_xlabel('windspeed', fontsize=18)
sub12.set_ylabel('registered', fontsize=18)
sub12.tick_params(axis='both', labelsize=16)

# Scatter plot for 'windspeed' and 'count'
sub13 = plt.subplot(8, 2, 13)
sns.scatterplot(x=df['windspeed'], y=df['count'])
sub13.set_title('Scatter plot of windspeed and count', fontsize=20)
sub13.set_xlabel('windspeed', fontsize=18)
sub13.set_ylabel('count', fontsize=18)
sub13.tick_params(axis='both', labelsize=16)

# Scatter plot for 'casual' and 'registered'
sub14 = plt.subplot(8, 2, 14)
sns.scatterplot(x=df['casual'], y=df['registered'])
sub14.set_title('Scatter plot of casual and registered', fontsize=20)
sub14.set_xlabel('casual', fontsize=18)
sub14.set_ylabel('registered', fontsize=18)
sub14.tick_params(axis='both', labelsize=16)

# Scatter plot for 'casual' and 'count'
sub15 = plt.subplot(8, 2, 15)
sns.scatterplot(x=df['casual'], y=df['count'])
sub15.set_title('Scatter plot of casual and count', fontsize=20)
sub15.set_xlabel('casual', fontsize=18)
sub15.set_ylabel('count', fontsize=18)
sub15.tick_params(axis='both', labelsize=16)

# Scatter plot for 'registered' and 'count'
sub15 = plt.subplot(8, 2, 16)
sns.scatterplot(x=df['registered'], y=df['count'])
sub15.set_title('Scatter plot of registered and count', fontsize=20)
sub15.set_xlabel('registered', fontsize=18)
sub15.set_ylabel('count', fontsize=18)
sub15.tick_params(axis='both', labelsize=16)

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

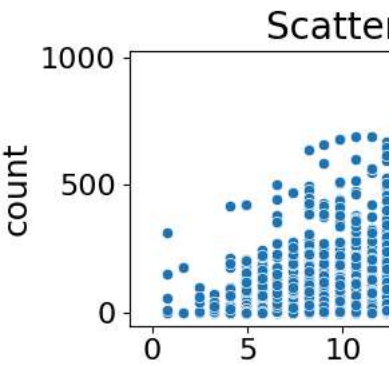

Scatter plot of temp and atemp
atemp
temp
0
20
40
0
5
10
15
20
25
30
35
40

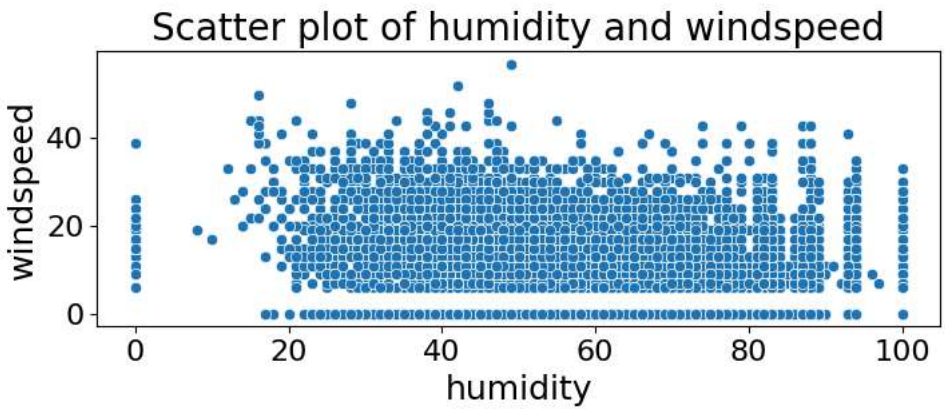
Scatter
humidity
0
50
100
0
5
10

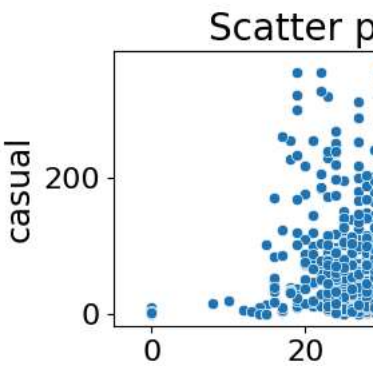

Scatter plot of temp and windspeed
windspeed
temp
0
20
40
0
5
10
15
20
25
30
35
40

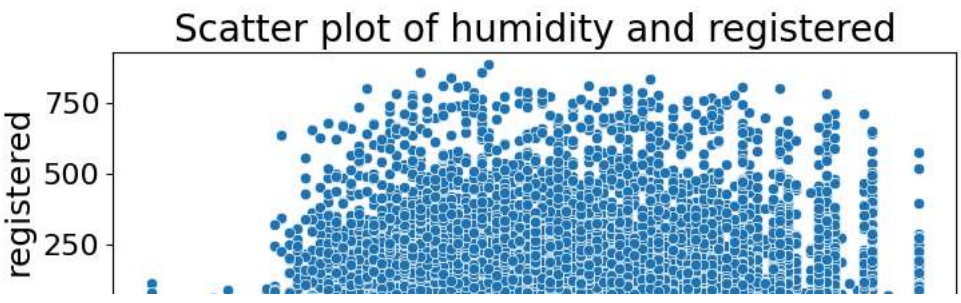

Scatter
casual
0
200
0
5
10

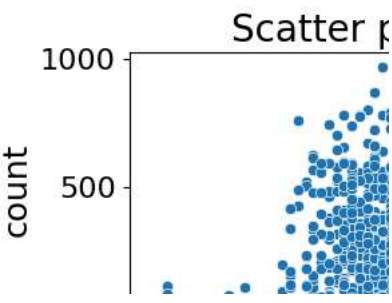

Scatter plot of temp and registered
registered
temp
0
250
500
750
0
5
10
15
20
25
30
35
40


Scatte
count
0
500
1000
0
5
10


Scatter plot of humidity and windspeed
windspeed
humidity
0
20
40
0
20
40
60
80
100

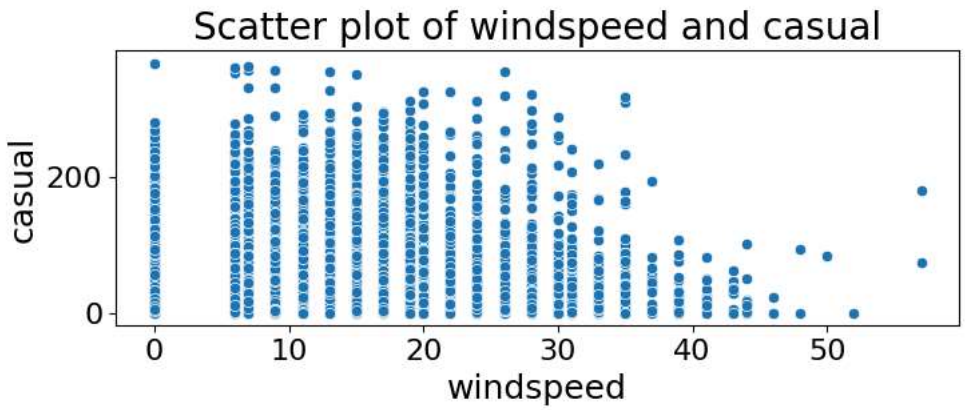

Scatter p
casual
0
200
0
20

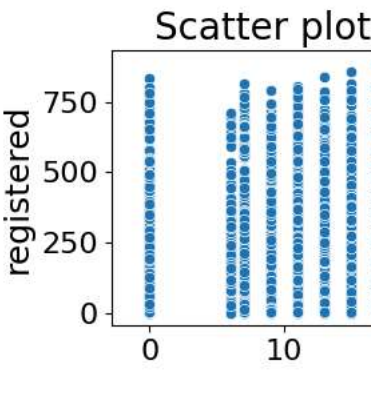

Scatter plot of humidity and registered
registered
0
250
500
750

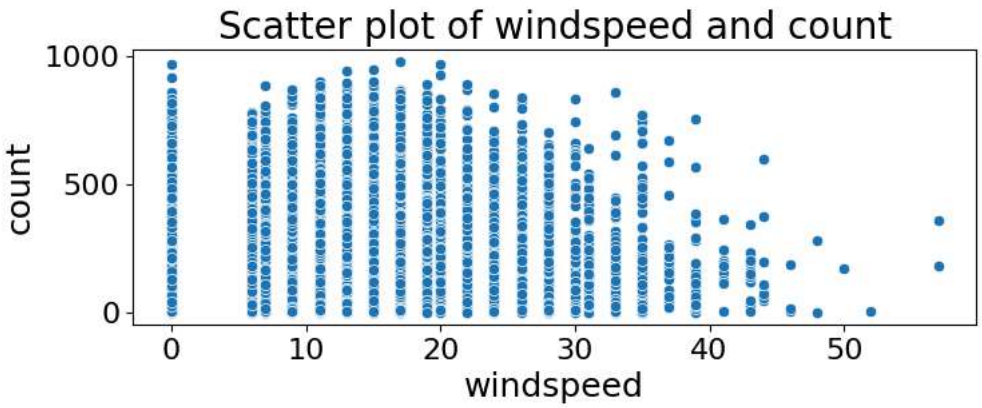

Scatter p
count
0
500
1000

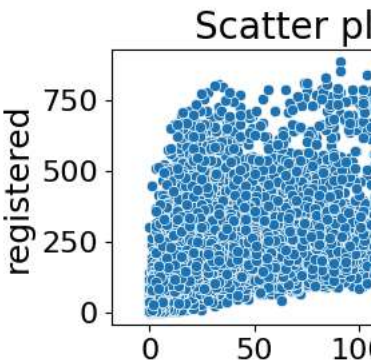

0
0
20
40
60
80
100
humidity

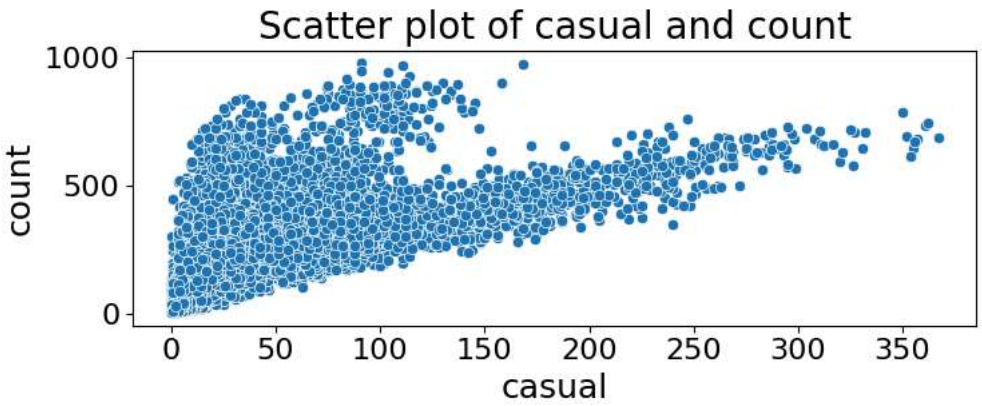

0
0
20

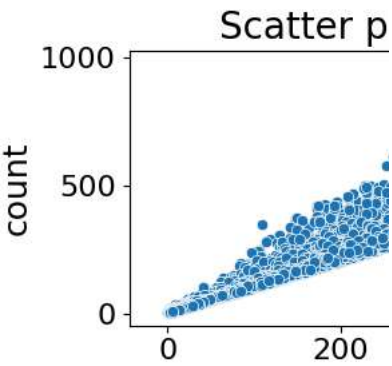

Scatter plot of windspeed and casual
casual
0
200
0
10
20
30
40
50
windspeed


Scatter plot
registered
0
250
500
750
0
10


Scatter plot of windspeed and count
count
0
500
1000
0
10
20
30
40
50
windspeed

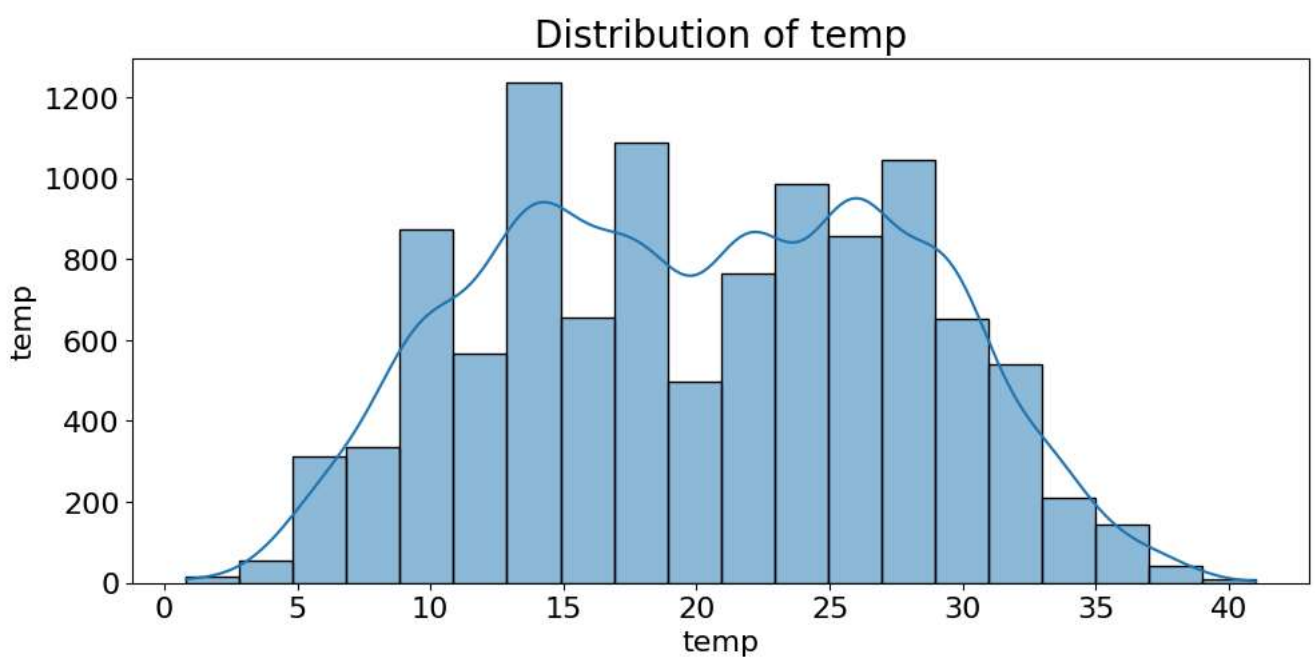

Scatter pl
registered
0
250
500
750
0
50

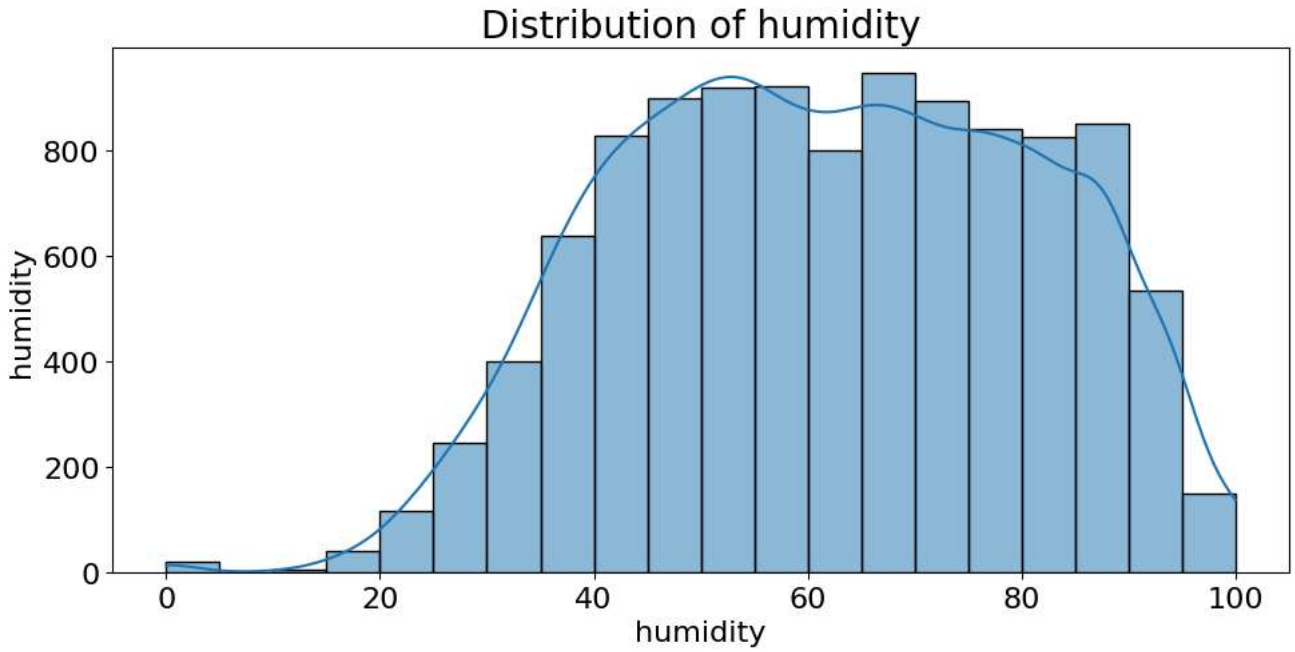

Scatter plot of casual and count
count
0
500
1000
0
50
100
150
200
250
300
350
casual

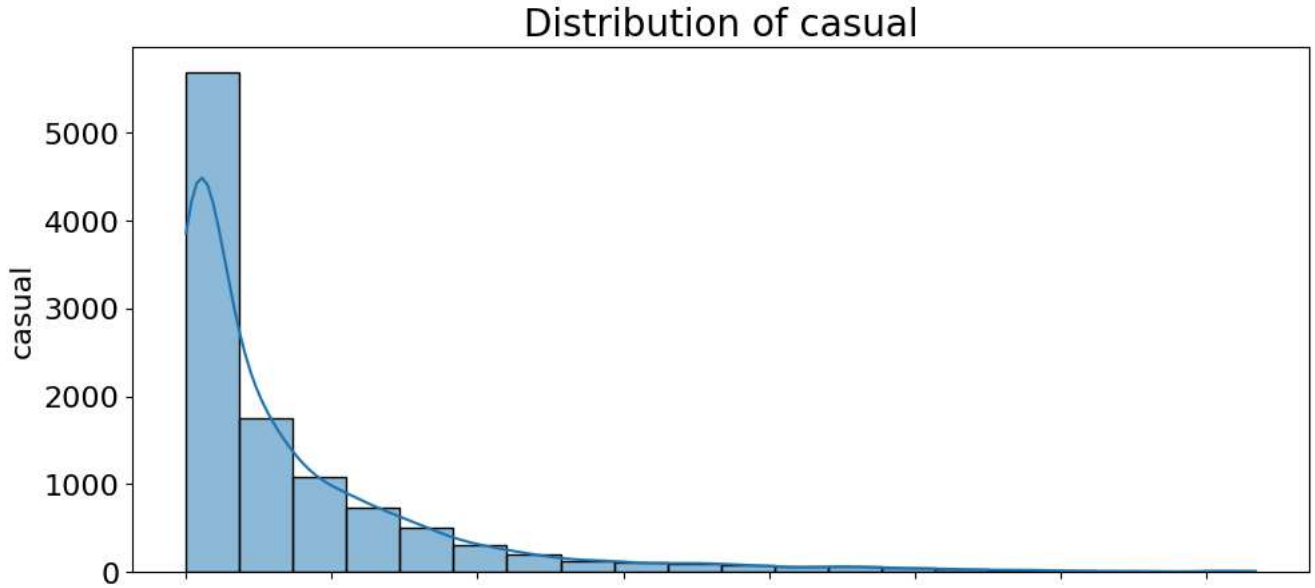

Scatter p
count
0
500
1000
0
200

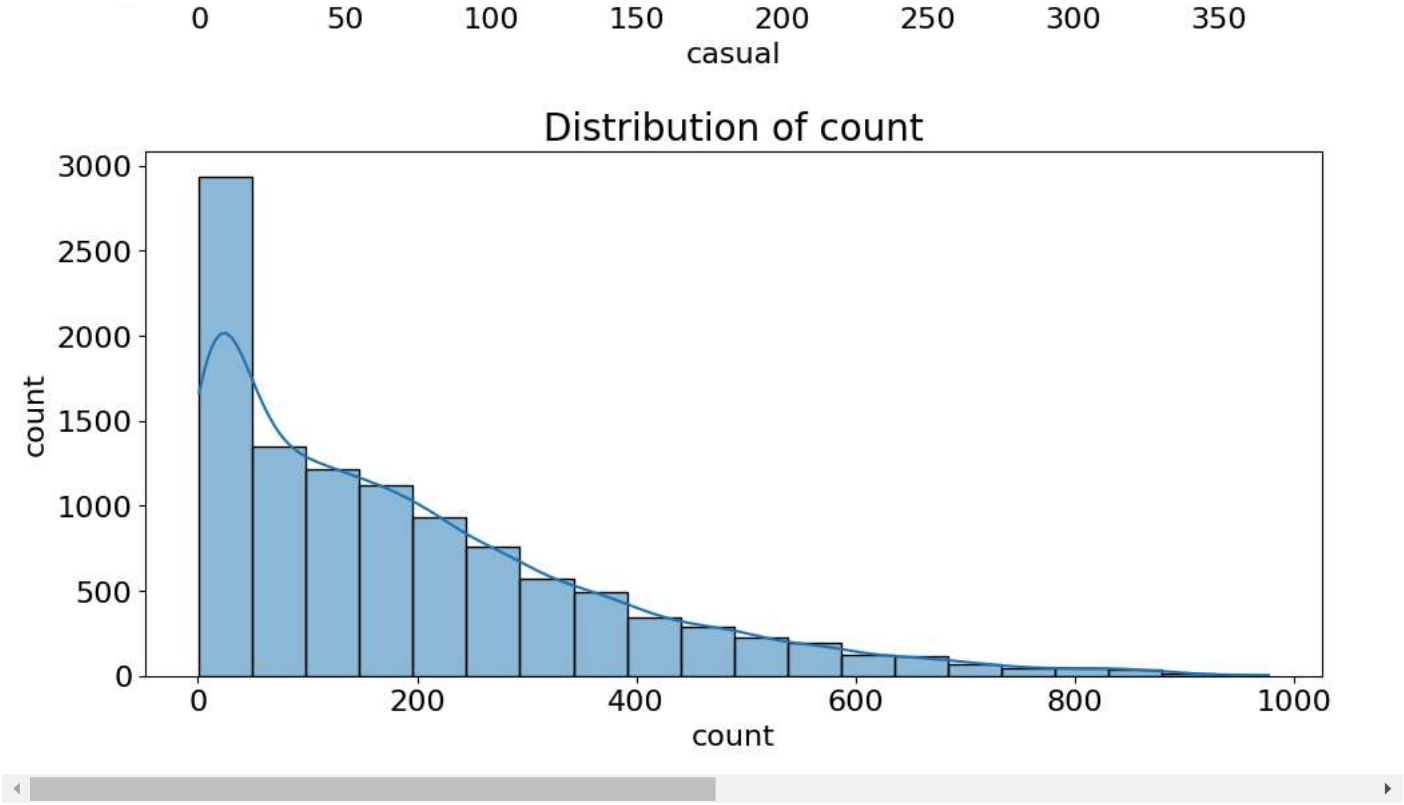
- Univariate Analysis of Numerical Variables:

```
# distribution of numerical variables using histogram

plt.figure(figsize=(25, 25))

plt.subplot(4, 2, 1)
sns.histplot(df['temp'], bins=20, kde=True)
plt.title('Distribution of temp', fontsize=20)
plt.xlabel('temp', fontsize=16)
plt.ylabel('temp', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 2)
sns.histplot(df['atemp'], bins=20, kde=True)
plt.title('Distribution of atemp', fontsize=20)
plt.xlabel('atemp', fontsize=16)
plt.ylabel('atemp', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 3)
sns.histplot(df['humidity'], bins=20, kde=True)
plt.title('Distribution of humidity', fontsize=20)
plt.xlabel('humidity', fontsize=16)
plt.ylabel('humidity', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 4)
sns.histplot(df['windspeed'], bins=20, kde=True)
plt.title('Distribution of windspeed', fontsize=20)
plt.xlabel('windspeed', fontsize=16)
plt.ylabel('windspeed', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 5)
sns.histplot(df['casual'], bins=20, kde=True)
plt.title('Distribution of casual', fontsize=20)
plt.xlabel('casual', fontsize=16)
plt.ylabel('casual', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 6)
sns.histplot(df['registered'], bins=20, kde=True)
plt.title('Distribution of registered', fontsize=20)
plt.xlabel('registered', fontsize=16)
plt.ylabel('registered', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 7)
sns.histplot(df['count'], bins=20, kde=True)
plt.title('Distribution of count', fontsize=20)
plt.xlabel('count', fontsize=16)
plt.ylabel('count', fontsize=16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplots_adjust(hspace=0.3)

plt.show()
```


Distribution of temp
temp
0
200
400
600
800
1000
1200
0
5
10
15
20
25
30
35
40
temp

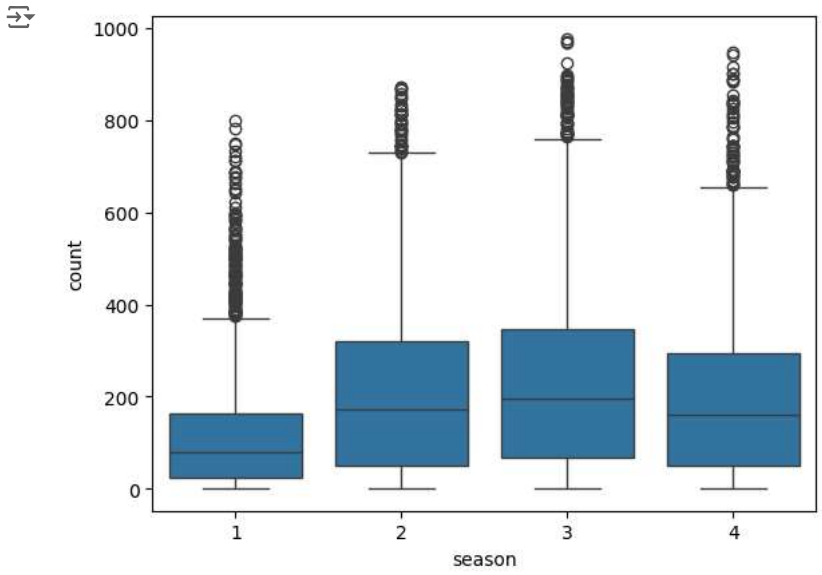

Distribution of humidity
humidity
0
200
400
600
800
0
20
40
60
80
100
humidity


Distribution of casual
casual
0
1000
2000
3000
4000
5000



3.3 Bivariate Analysis of Categorical and Numerical variable

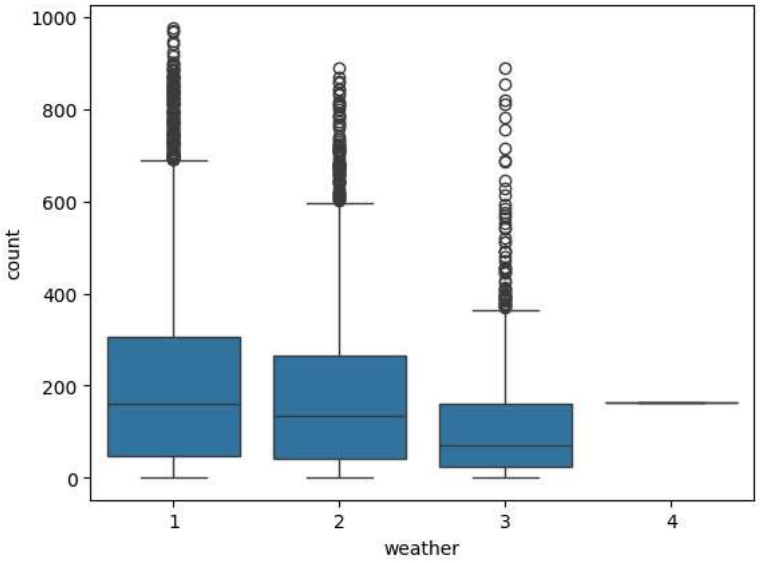
```
sns.boxplot(x='season', y='count', data=df)
plt.show()
```



Observation

1. Median of season 3 is highest followed by 2,4,1.
2. Count of cycles is dependnt on season.
3. Highest no of outliers are obsereved in Season 1 followed by 4,3,2.

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



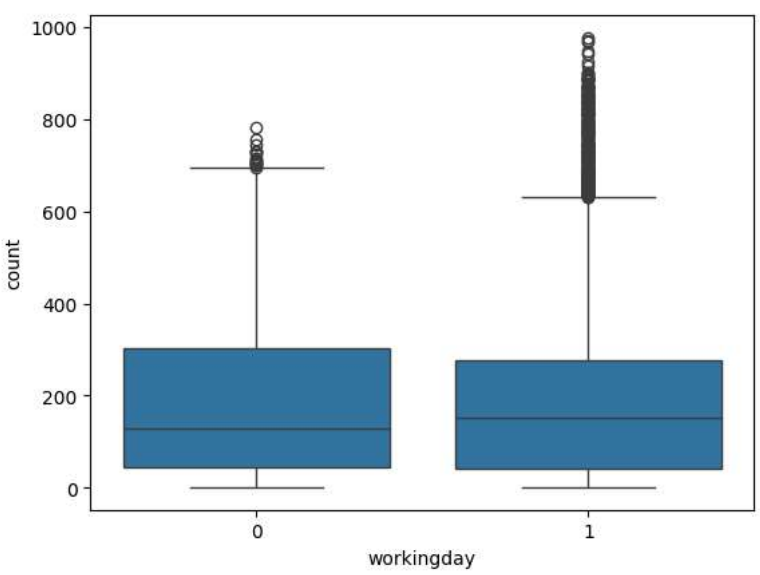
Median for weather 1 is more than other weathers.

Weather and count of cycles are dependent.

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

Highest outliers are observed for weather 3.

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



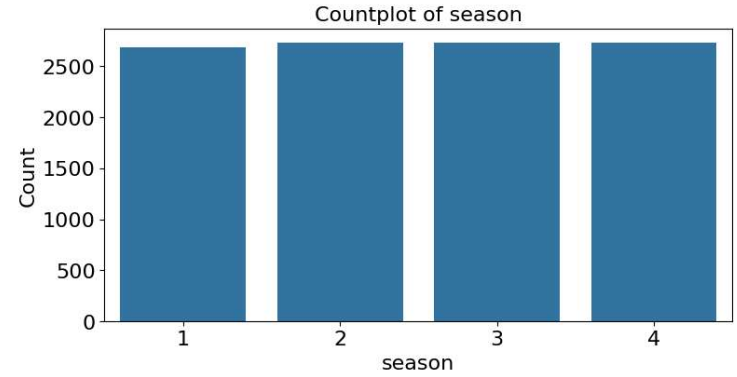
Observation

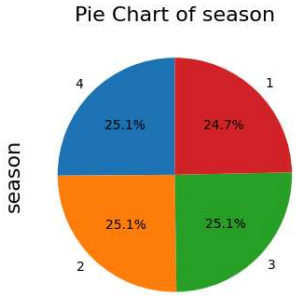
1. Median of workingday and non-workingday for the count of cycle are almost same.
2. Working day has more outliers as compared to no working day.

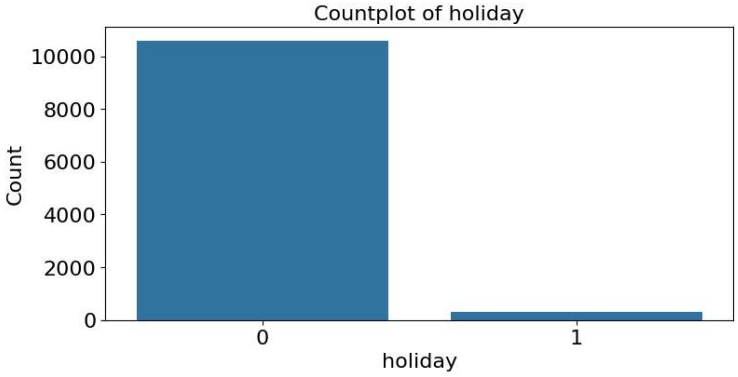
3.4 Univariate Analysis of Categorical Variables

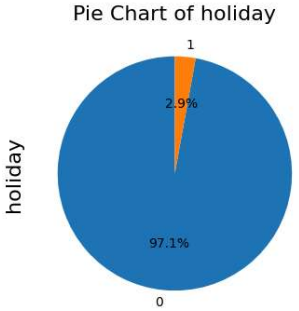
```
plt.figure(figsize=(20, 20))

plt.subplot(4, 2, 1)
sns.countplot(x='season', data=df, linewidth=1)
```

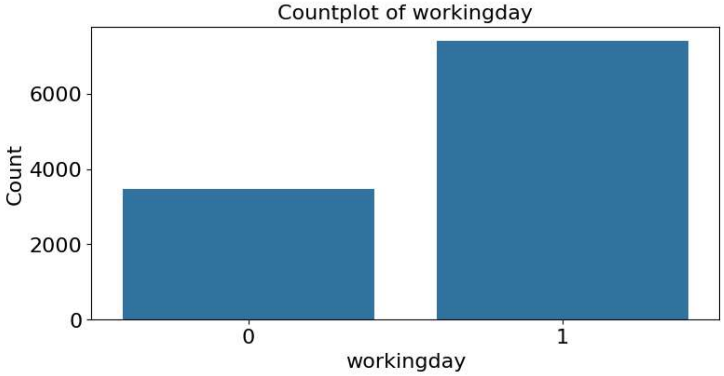
```
plt.title('Countplot of season', fontsize = 16)
plt.xlabel('season', fontsize = 16)
plt.ylabel('Count', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

# Pie chart for better visualization of proportions
plt.subplot(4, 2, 2)
df['season'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart of season', fontsize = 16)
plt.ylabel('season', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 3)
sns.countplot(x='holiday', data=df, linewidth=1)
plt.title('Countplot of holiday', fontsize = 16)
plt.xlabel('holiday', fontsize = 16)
plt.ylabel('Count', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

# Pie chart for better visualization of proportions
plt.subplot(4, 2, 4)
df['holiday'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart of holiday', fontsize = 16)
plt.ylabel('holiday', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 5)
sns.countplot(x='workingday', data=df, linewidth=1)
plt.title('Countplot of workingday', fontsize = 16)
plt.xlabel('workingday', fontsize = 16)
plt.ylabel('Count', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

# Pie chart for better visualization of proportions
plt.subplot(4, 2, 6)
df['workingday'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart of workingday', fontsize = 16)
plt.ylabel('workingday', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

plt.subplot(4, 2, 7)
sns.countplot(x='weather', data=df, linewidth=1)
plt.title('Countplot of weather', fontsize = 16)
plt.xlabel('weather', fontsize = 16)
plt.ylabel('Count', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)

# Pie chart for better visualization of proportions
plt.subplot(4, 2, 8)
df['weather'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title('Pie Chart of weather', fontsize = 16)
plt.ylabel('weather', fontsize = 16)
ax = plt.gca()
ax.tick_params(axis='both', labelsize=16)


plt.subplots_adjust(wspace=0.3)
plt.subplots_adjust(hspace=0.3)


plt.show()
```

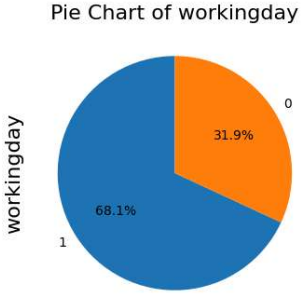

Countplot of season
Count
season
0
500
1000
1500
2000
2500
1
2
3
4

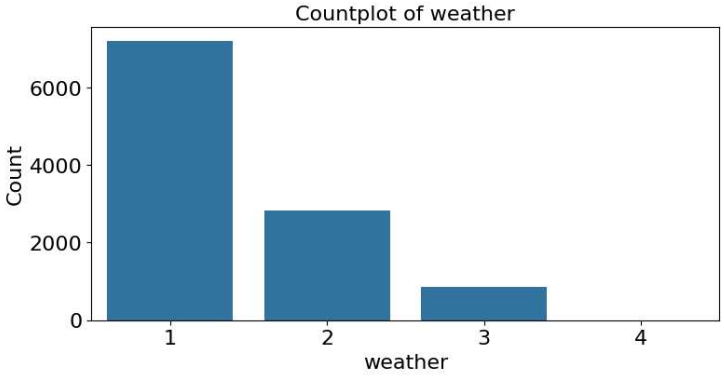

Pie Chart of season
season
25.1%
24.7%
25.1%
25.1%
1
2
3
4

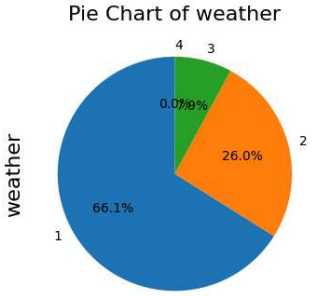

Countplot of holiday
Count
holiday
0
2000
4000
6000
8000
10000
0
1


Pie Chart of holiday
holiday
2.9%
97.1%
1
0


Countplot of workingday
Count
workingday
0
2000
4000
6000
0
1


Pie Chart of workingday
workingday
31.9%
68.1%
0
1


Countplot of weather
Count
weather
0
2000
4000
6000
1
2
3
4


Pie Chart of weather
weather
0.07%
26.0%
66.1%
1
2
3
4

Observations

1. Count of all four seasons is almost equally distributed.
2. 2.9% of the data is for holiday, rest all is for weekend.
3. 68.1% of data is for working day, rest all is for weekend or holiday.
4. 66.1 % of data is for weather 1, 26% of the data is for weather 2, 9% of data is for weather 3 and 0% is for weatehr 4.

** Hypothesis Testing**

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

H0 = The count of bicycles rented during non-working day <= the count on working day

Ha = The count of bicycles rented during non-working day > the count on working day

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

# variance is not same
```

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

```
non_working = df[df['workingday']==0]['count']
working = df[df['workingday']==1]['count']


# using ttest_ind to find relation

test_stats, p_value= ttest_ind(non_working, working, equal_var=False, alternative='greater')

alpha = 0.05

if p_value > alpha:
    print("failed to reject H0")
else:
    print("reject H0")
```

```
failed to reject H0
```

- As the p_value is greater than alpha, NULL hypothesis can't be rejected.
- There is no sufficient evidence that working day has more probability of more count of ride than non working day.

Recommendations:

1. Since the analysis indicates a higher demand on working days, Yulu should focus on strengthening its services during these days. This could involve optimizing fleet distribution, ensuring bike availability during peak working hours, and providing promotions or incentives to attract more users on weekdays.
2. While the analysis doesn't show a significant difference in demand, Yulu can still explore opportunities to increase bicycle rentals on non-working days. This may involve targeted marketing campaigns, special promotions, or events during weekends to encourage more users

to rent bicycles on those days.

4.2 Checking if the demand of bicycles on rent is the same for different Weather conditions?

H0 = The count of bicycles rented during different weather is similar

Ha = The count of bicycles rented during differnt weather is different

```
df.groupby('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

```
# count of hours and variance are different
# Taking the equal no of samples i.e 850

w1= df[df['weather']==1]['count'].sample(850)
w2= df[df['weather']==2]['count'].sample(850)
w3= df[df['weather']==3]['count'].sample(850)
```

Conditions to use ANOVA:-

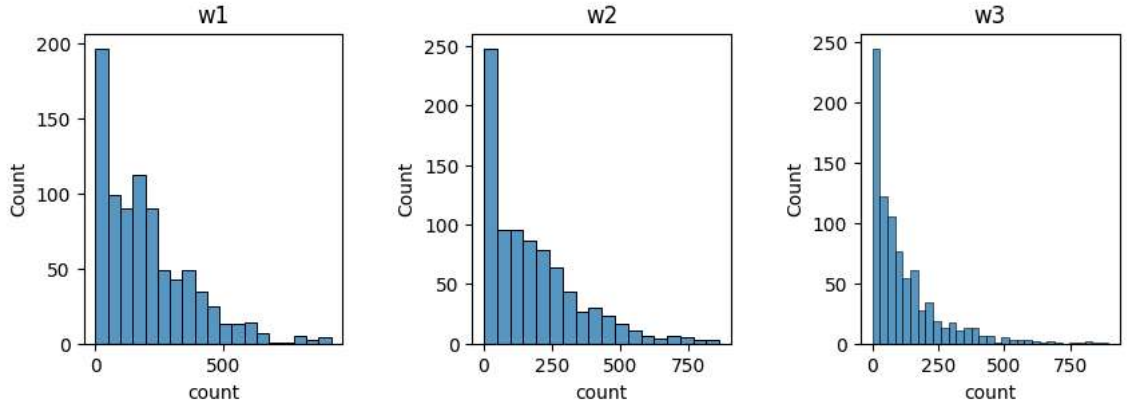
1. Distribution should be normal

2. Variance should be equal

4.2.1 Checking if distribution normal by means of visual and statistical analysis

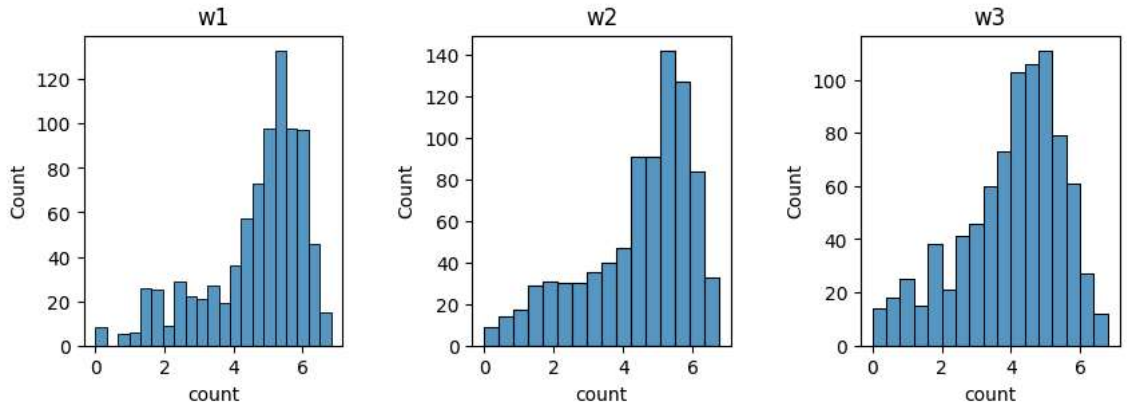
```
plt.figure(figsize=(10, 3))

plt.subplots_adjust(wspace=0.5)
plt.subplot(1,3,1)
sns.histplot(w1)
plt.title('w1')

plt.subplot(1,3,2)
sns.histplot(w2)
plt.title('w2')

plt.subplot(1,3,3)
sns.histplot(w3)
plt.title('w3')
```

```
Text(0.5, 1.0, 'w3')
```



None of the distributions for weather is normal. next step is to check by applying transformation

```
# checking by applying log-transformation

plt.figure(figsize=(10, 3))

plt.subplots_adjust(wspace=0.5)
plt.subplot(1,3,1)
sns.histplot(np.log(w1))
plt.title('w1')

plt.subplot(1,3,2)
sns.histplot(np.log(w2))
plt.title('w2')

plt.subplot(1,3,3)
sns.histplot(np.log(w3))
plt.title('w3')

plt.show()
```



After applying transformation, the distribution does not look normal by visual

```
# statistical test to check normality of data
# H0: The Series is Normal distribution
# Ha: The Series is not Normal distribution

test_stats_1,p_value_1 = shapiro(w1)
test_stats_2,p_value_2 = shapiro(w2)
test_stats_3,p_value_3 = shapiro(w3)

alpha=0.05

if p_value_1 > alpha:
    print("for W1 failed to reject H0")
else:
    print("for W1 reject H0")
```

```
if p_value_2 > alpha:
    print("for W2 failed to reject H0")
else:
    print("for W2 reject H0")


if p_value_3 > alpha:
    print("for W2 failed to reject H0")
else:
    print("for W2 reject H0")
```

```
for W1 reject H0
for W2 reject H0
for W2 reject H0
```

Observation for distribution

1. p_values for all Weathers are less than alpha so the NULL hypothesis is rejected.
2. statistically it's proven that distribution is not Normal.

```
# H0: Variance is same
# Ha: Variance is not same

test_stats, p_value = levene(w1,w2,w3)

alpha = 0.05


if p_value > alpha:
    print("failed to reject H0")
else:
    print("reject H0")
```

```
reject H0
```

Observation

1. p_values for all Weathers are less than alpha so the NULL hypothesis is rejected.
2. Statistically it's proven that variance for all weather are differenct.
3. As distribution is not normal and variance are different, ANOVA cannot be used, so Kruskal Wallis Test is to be preferred.

** Kruskal Wallis test**

```
# H0 = The count of bicycles rented during different weather is similar
# Ha = The count of bicycles rented during different weather is different


# Performing Kruskal-Wallis test
statistic, p_value = kruskal(w1, w2, w3)

print("Kruskal-Wallis Statistic:", statistic)
print()
print("P-value:", p_value)
print()

alpha = 0.05
if p_value < alpha:
    print("Reject the NULL hypothesis")
else:
    print("Can't reject NULL hypothesis")
```

```
Kruskal-Wallis Statistic: 118.68130419688478

P-value: 1.6930985629832562e-26
```

```
Reject the NULL hypothesis
```

Recommendation:

1. Implement a dynamic pricing strategy that takes weather conditions into account. For instance, Yulu could offer discounted rates during adverse weather to encourage usage and adjust prices during high-demand weather conditions.
2. Adjust the fleet composition based on weather patterns. For example, introduce weather-resistant bicycles or cycles with features suitable for specific weather conditions to enhance user comfort and attract more users.
3. Collaborate with weather forecasting platforms to access real-time weather data. This partnership can enable Yulu to proactively adjust operations, marketing, and promotions based on accurate and up-to-date weather information.
4. Educate users about the suitability of bicycles in different weather conditions. Provide tips on safe riding practices during adverse weather.

Observation

The number of bicycles rented during different weather is different, so the bicycles rented was affected by weather condition

** Check if the demand of bicycles on rent is the same for different Seasons?**

H0 = The count of bicycles rented during different seasons is similar

Ha = The count of bicycles rented during differnt seasons is different

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

season	count	mean	std	min	25%	50%	75%	max
1	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0
2	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0
3	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0
4	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0

Conditions to use ANOVA:-

1. Distribution should be normal

2. Variance should be equal

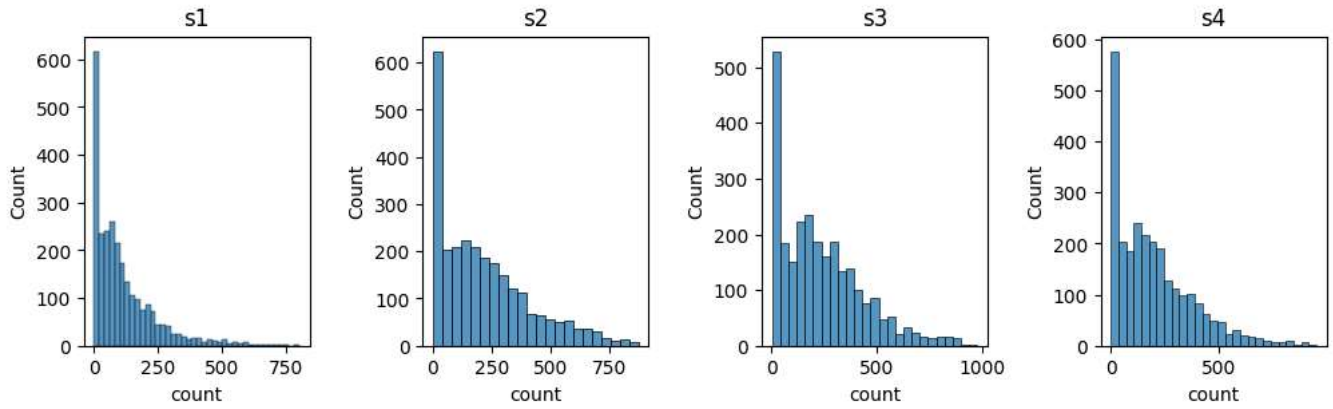
```
s1 = df[df['season']==1]['count'].sample(2650)
s2 = df[df['season']==2]['count'].sample(2650)
s3 = df[df['season']==3]['count'].sample(2650)
s4 = df[df['season']==4]['count'].sample(2650)
```

Checking if distribution normal by means of visual and statistical analysis**

```
plt.figure(figsize=(12, 3))

plt.subplots_adjust(wspace=0.5)
plt.subplot(1,4,1)
sns.histplot(s1)
plt.title('s1')

plt.subplot(1,4,2)
sns.histplot(s2)
plt.title('s2')
```

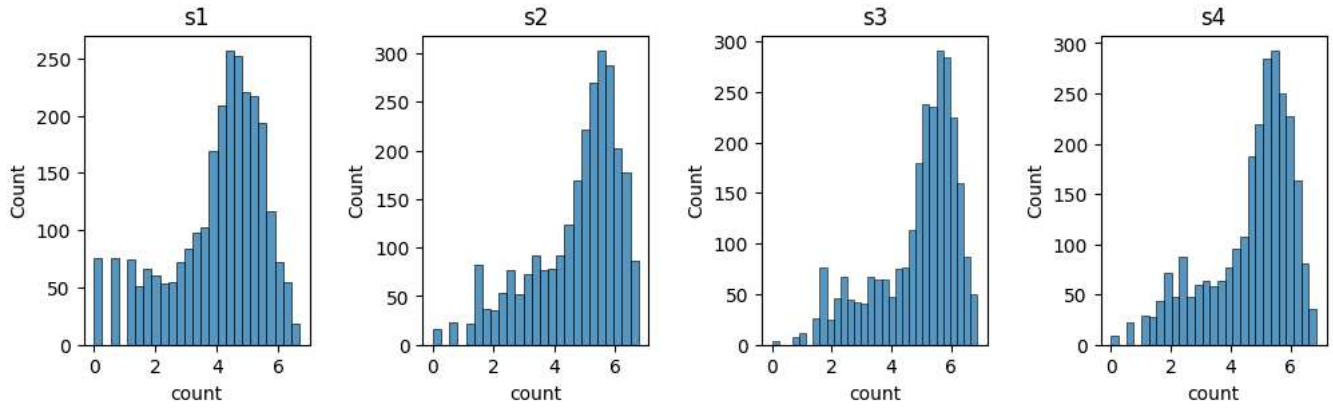
```
plt.subplot(1,4,3)
sns.histplot(s3)
plt.title('s3')

plt.subplot(1,4,4)
sns.histplot(s4)
plt.title('s4')
```

```
Text(0.5, 1.0, 's4')
```



None of the distributions for weather is normal. next step is to check by applying transformation

```
# checking by applying transformation

plt.figure(figsize=(12, 3))

plt.subplots_adjust(wspace=0.5)
plt.subplot(1,4,1)
sns.histplot(np.log(s1))
plt.title('s1')

plt.subplot(1,4,2)
sns.histplot(np.log(s2))
plt.title('s2')

plt.subplot(1,4,3)
sns.histplot(np.log(s3))
plt.title('s3')

plt.subplot(1,4,4)
sns.histplot(np.log(s4))
plt.title('s4')

plt.show()
```



After applying the transformation, the distribution is not normal

```
# checking normal distribution statistically
```

```
# H0: The Series is Normal distribution
# Ha: The Series is not Normal distribution

stats_1,p_val_1 = shapiro(s1)
stats_2,p_val_2 = shapiro(s2)
stats_3,p_val_3 = shapiro(s3)
stats_4,p_val_4 = shapiro(s4)

alpha=0.05

if p_val_1 > alpha:
    print("for s1 failed to reject H0")
else:
    print("for s1 reject H0")

if p_val_2 > alpha:
    print("for s2 failed to reject H0")
else:
    print("for s2 reject H0")

if p_val_3 > alpha:
    print("for s3 failed to reject H0")
else:
    print("for s3 reject H0")

if p_val_4 > alpha:
    print("for s4 failed to reject H0")
else:
    print("for s4 reject H0")
```

```
for s1 reject H0
for s2 reject H0
for s3 reject H0
for s4 reject H0
```

Observation

1. p_values for all seasons are less than alpha so the NULL hypothesis is rejected.
2. Statistically it's proven that distribution is not Normal

Checking if variance is same by means of visual and statistical analysis**

```
# H0: Variance is same
# Ha: Variance is not same

test_stats, p_value = levene(s1,s2,s3,s4)
print(test_stats,p_value)

alpha = 0.05

if p_value > alpha:
    print("failed to reject H0")
else:
    print("reject H0")
```

```
189.21386404985432 1.5664227742005114e-119
reject H0
```

observation for distribution

1. p_values for all seasons are less than alpha so the NULL hypothesis is rejected.
2. statistically it's proven that variance for all season are differenct.
3. As distribution is not normal and variance are different, ANOVA cannot be used, so Kruskal Wallis Test is to be preferred.

Kruskal Wallis Test**

```
# H0 = The count of bicycles rented during different seasons is similar
# Ha = The count of bicycles rented during different seasons is different


# Performing Kruskal-Wallis test
statistic_s, p_value_kru = kruskal(w1, w2, w3)

print("Kruskal-Wallis Statistic:", statistic_s)
print()
print("P-value:", p_value_kru)
print()

alpha = 0.05
if p_value_kru < alpha:
    print("Reject the NULL hypothesis")
else:
    print("Can't reject NULL hypothesis")
```

```
Kruskal-Wallis Statistic: 118.68130419688478

P-value: 1.6930985629832562e-26

Reject the NULL hypothesis
```

Observation:

The number of bicycles rented during differnt season is different, so the bicycles rented was affected by season.

Recommendations:

1. Observing the trends, it is evident that the demand for bicycles is significantly higher during the fall season, followed by summer, winter, and spring. This information underscores the importance of tailoring strategies to accommodate the seasonal variations in sales.
2. In light of the seasonal sales data, it is recommended that Yulu focuses on promoting the usage of bicycles during the seasons with comparatively lower sales, namely winter and spring. Implementing targeted marketing campaigns, special promotions, and innovative features during these periods can potentially stimulate demand and enhance overall sales performance.

Checking if the Weather conditions are significantly different during different Seasons?

Both the variables are categorical so chisquare test is to be used

H0: Weather and Season are Independent

Ha: Weather and Season are not Independent

```
# creating crosstab

data = pd.crosstab(df['weather'], df['season'])
data
```

season	1	2	3	4
weather				
1	1759	1801	1930	1702
2	715	708	604	807
3	211	224	199	225
4	1	0	0	0

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

```
chi2_stat, p_value_chi, dof, expected = chi2_contingency(data)

alpha = 0.05
if p_value_chi > alpha:
    print("failed to reject H0")
else:
    print("reject H0")
```

```
reject H0
```

Observation:

Weather and seasons are dependent.

Recommendation:

1. Develop season-specific strategies to address varying weather patterns. For example, during monsoons or extreme weather, Yulu can implement additional safety measures or promote alternative transportation options.
2. During seasons with challenging weather conditions, consider temporary service adaptations. This might include adjusting bike availability, introducing weather-specific features, or providing users with real-time weather alerts and recommendations.
3. Collaborate with weather services to integrate accurate and real-time weather information into Yulu's platform.

Business Insights

- Maximum bike rentals occur during summer, while the minimum is observed in winter.
- Clear weather is associated with the highest bike rental counts, whereas rentals sharply decrease in rain, thunderstorm, snow, or fog.