



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy.stats import binom,geom,norm,ttest_1samp,ttest_ind,ttest_rel,chi2,chi2_contingency,chisquare,f_oneway,poisson,expon,probplot,shapiro,levene,kruskal
```

```
!gdown "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089"
```

```
Downloading...
From: https://d2beiqkhq929f0.cloudfront.net/public\_assets/assets/000/001/428/original/bike\_sharing.csv?1642089089
To: /content/bike_sharing.csv?1642089089
100% 648k/648k [00:00<00:00, 1.67MB/s]
```

```
df = pd.read_csv('bike_sharing.csv')
```

```
# Analysing the structure of the data
df
```

	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	0.0000	3	13	16	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1	
...	
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	

10886 rows x 12 columns

```
df.shape

(10886, 12)
```

```
df.ndim

2
```

```
df.dtypes
```

```

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

```

```

# There are no null values in the dataset
df.isna().sum()

```

```

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

```

```

# statistics of columns that have numerical data
df.describe()

```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

```
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
```

```
df[df.duplicated()]
# There are no duplicate rows in the dataset
```

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



```
sns.boxplot(df)
plt.xticks(rotation = 90)
# There are some outliers in the dataset

# We remove them when we plot the boxplots
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
 [Text(0, 0, 'season'),
  Text(1, 0, 'holiday'),
  Text(2, 0, 'workingday'),
  Text(3, 0, 'weather'),
  Text(4, 0, 'temp'),
  Text(5, 0, 'atemp'),
  Text(6, 0, 'humidity'),
```

```
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
```

```
df.head()
```

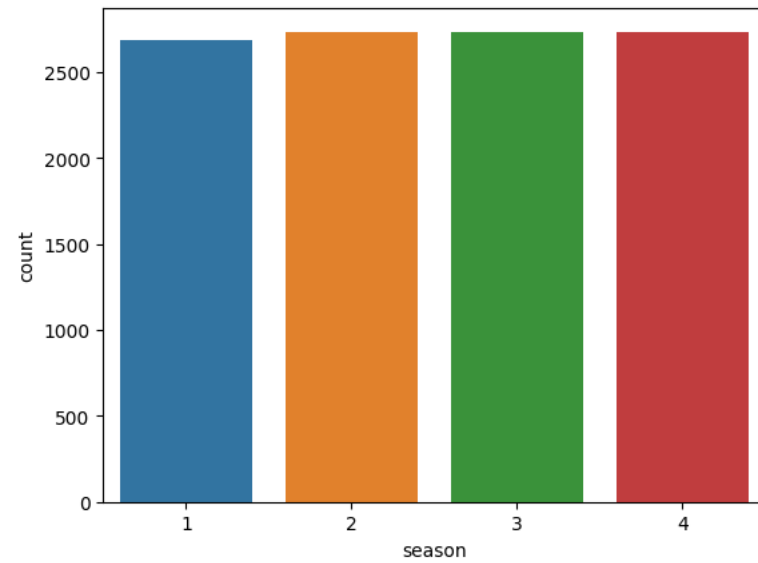
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84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

```
min_date = df['datetime'].min()
max_date = df['datetime'].max()
print("Min date of the data : ",min_date)
print("Max date of the data : ",max_date)
# So the data given to us is from Jan 1, 2011 to Dec 19, 2012
```

```
Min date of the data : 2011-01-01 00:00:00
Max date of the data : 2012-12-19 23:00:00
```

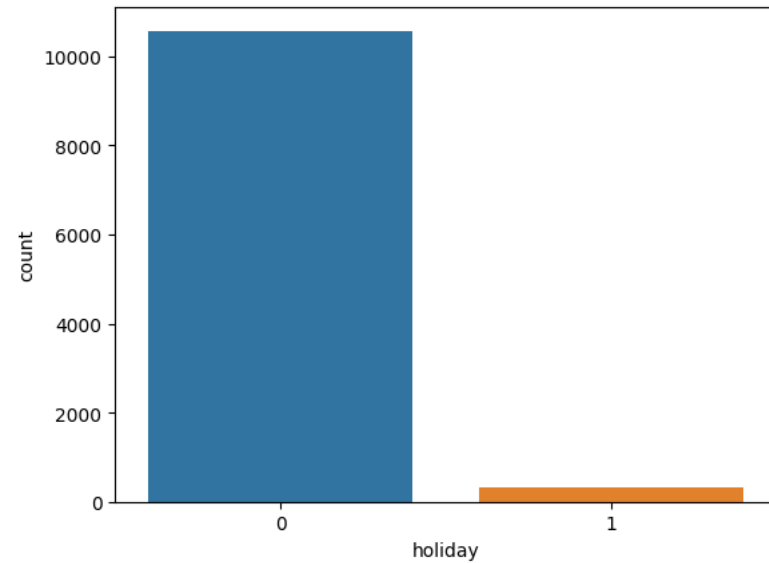
```
sns.countplot(data = df , x = 'season')
# A total of 4 seasons were present for our dataset
# (1: spring, 2: summer, 3: fall, 4: winter)
# Almost equal data points were given for all the 4 seasons
```

<Axes: xlabel='season', ylabel='count'>



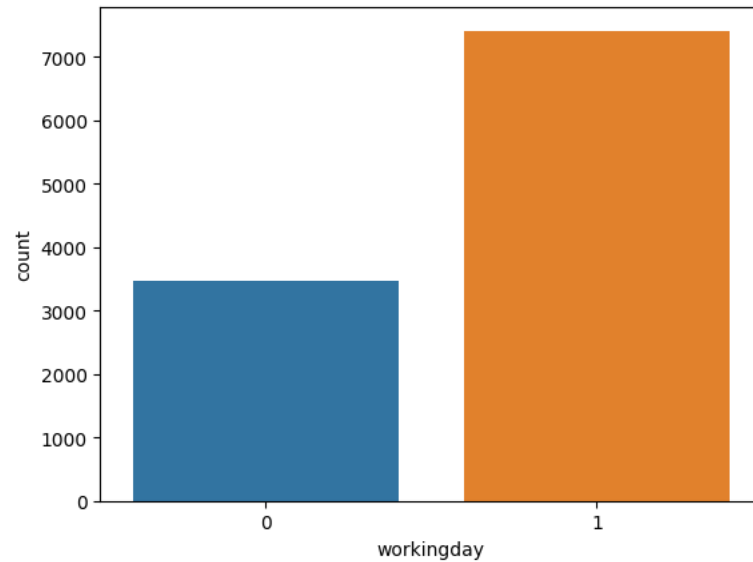
```
# Number of Holidays
sns.countplot(data = df, x = 'holiday')
# holiday = 1 is to indicate whether a day is holiday or not
# so we have data where number of holidays is very minimal
```

<Axes: xlabel='holiday', ylabel='count'>



```
sns.countplot(data = df, x = 'workingday')  
# number of working days is more in our dataset as we can see from the countplot  
# workingday = 1 is for working day and 0 for non working day which includes weekends too
```

<Axes: xlabel='workingday', ylabel='count'>



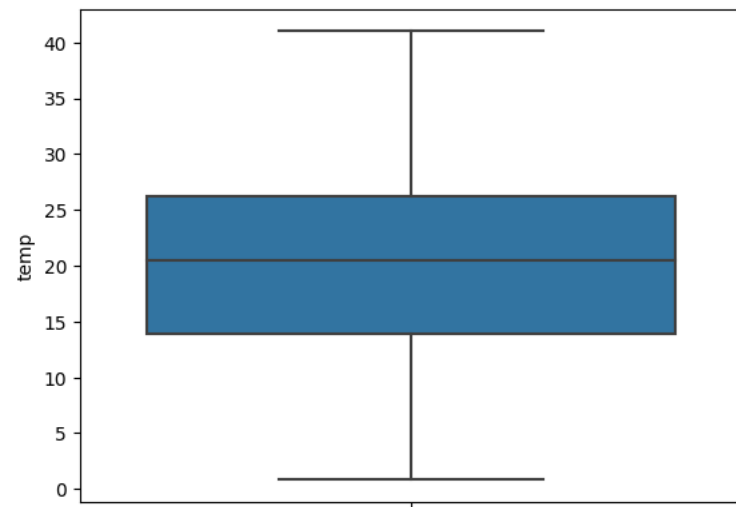
```
# weather  
sns.countplot(data = df, x = 'weather')  
# we have a total of 4 seasons  
# Inference :  
# More data points are available when the weather is partly cloudy, has few clouds, clear.  
# No data points were given for rainy days. We can clear infer that when it is raining no one prefers to use a bike because  
# it doesn't have any cover for the rider and so people do not tend to rent Yulu Bikes on a rainy day.  
  
# 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
```

<Axes: xlabel='weather', ylabel='count'>



```
sns.boxplot(data = df, y = 'temp')  
# box plot for temp in celsius  
# The median is around 20 degrees celsius  
# and there are no outliers
```

<Axes: ylabel='temp'>



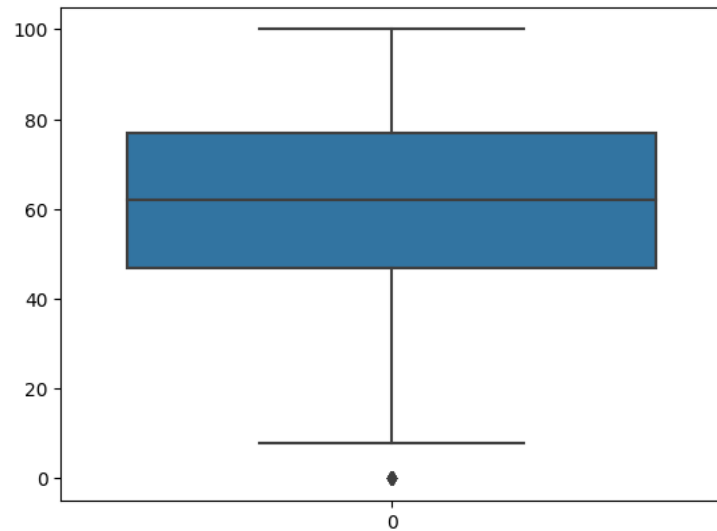
```
sns.boxplot(data = df, y = 'atemp')  
# atemp = feeling temperature  
# The median is around 25 degree celsius  
# and there are no outliers
```

<Axes: ylabel='atemp'>



```
sns.boxplot(data = df['humidity'])
# The variation of humidity can be seen from the box plot with median around 60 units
# and we see an outlier
```

<Axes: >



```
# Imputing the outliers
# use Boxplot,IQR to detect outliers
def imputing_outliers_with_mean(df,series_name):
    q1 = np.percentile(df[series_name],25)
    q3 = np.percentile(df[series_name],75)
    iqr = q3-q1

    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    if df[series_name].dtype == 'object':
        # Categorical series, impute with mode
        mode_value = df[series_name].mode().values[0]
        df.loc[df[series_name] < lower_bound] = mode_value
        df.loc[df[series_name] > upper_bound] = mode_value
    else:
        # Numerical series, impute with mean
```



```
mean_value = df[series_name].mean()
df.loc[df[series_name] < lower_bound] = mean_value
df.loc[df[series_name] > upper_bound] = mean_value
```

```
...
```

```
avg = df[series_name].mean()
```

```
df.loc[df[series_name] < lower_bound] = avg
```

```
df.loc[df[series_name] > upper_bound] = avg
```

```
...
```

```
return df
```

```
imputing_outliers_with_mean(df, 'humidity')
```

```
# we call the imputing_outliers_with_mean function to impute the outliers with mean of that series
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0000	3.0	13.0	16.0
1	2011-01-01 01:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	8.0	32.0	40.0
2	2011-01-01 02:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	5.0	27.0	32.0
3	2011-01-01 03:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	3.0	10.0	13.0
4	2011-01-01 04:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	0.0	1.0	1.0
...
10881	2012-12-19 19:00:00	4.0	0.0	1.0	1.0	15.58	19.695	50.0	26.0027	7.0	329.0	336.0
10882	2012-12-19 20:00:00	4.0	0.0	1.0	1.0	14.76	17.425	57.0	15.0013	10.0	231.0	241.0
10883	2012-12-19 21:00:00	4.0	0.0	1.0	1.0	13.94	15.910	61.0	15.0013	4.0	164.0	168.0
10884	2012-12-19 22:00:00	4.0	0.0	1.0	1.0	13.94	17.425	61.0	6.0032	12.0	117.0	129.0
10885	2012-12-19 23:00:00	4.0	0.0	1.0	1.0	13.12	16.665	66.0	8.9981	4.0	84.0	88.0



10886 rows × 12 columns

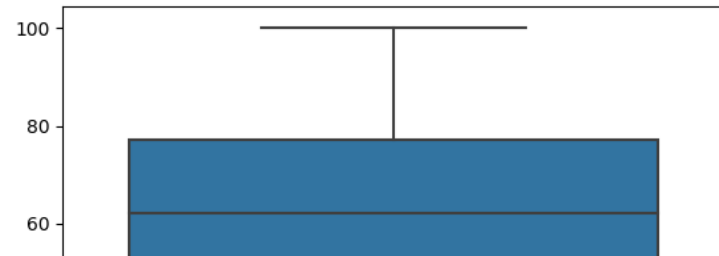
```
sns.boxplot(data = df['humidity'])
```

```
# Now the outliers are gone as they were imputed with mean of their series.
```

```
# The median still remained at notch above 60 units which in other words we can say
```

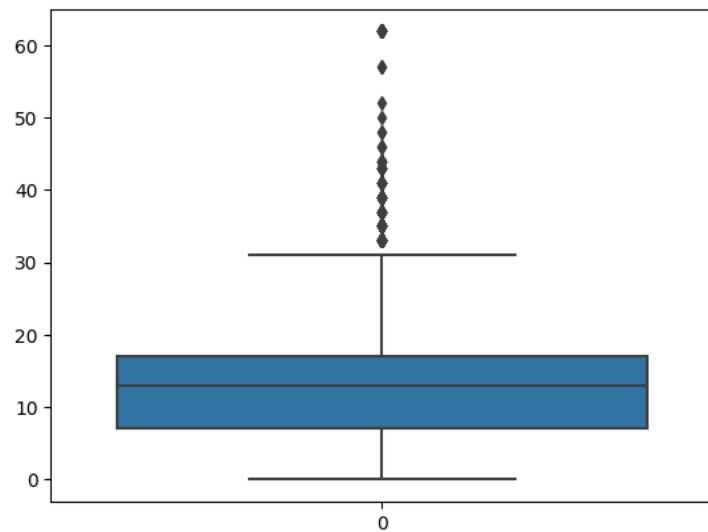
```
# Median is not affected by outlier treatment
```

<Axes: >



```
sns.boxplot(data = df['windspeed'])  
# The variation of humidity can be seen from the Box plot  
# and we see numerous outliers which can impact our analysis  
# so we impute them with mean  
# Here the median is around 15 units
```

<Axes: >



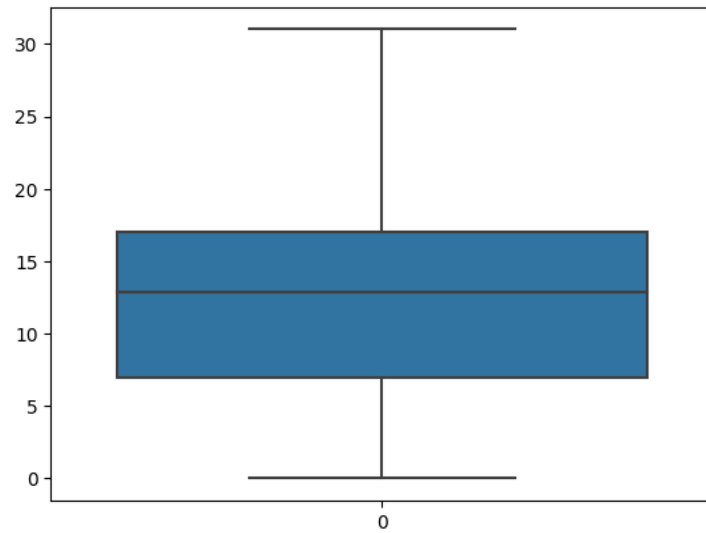
```
imputing_outliers_with_mean(df, 'windspeed')  
# we call the imputing_outliers_with_mean function to impute the outliers with mean of that series
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0000	3.0	13.0	16.0
1	2011-01-01 01:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	8.0	32.0	40.0
2	2011-01-01 02:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	5.0	27.0	32.0
3	2011-01-01 03:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	3.0	10.0	13.0
4	2011-01-01 04:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	0.0	1.0	1.0



```
sns.boxplot(data = df['windspeed'])
# we removed the outliers for the series windspeed.
# The same can be confirmed from the below Box Plot
```

<Axes: >



```
sns.boxplot(data = df['season'])
# There is an outlier in season series
# we impute this outlier with mode
```

<Axes: >



```
imputing_outliers_with_mean(df, 'season')
# removing outliers for season series
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0000	3.0	13.0	16.0
1	2011-01-01 01:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	8.0	32.0	40.0
2	2011-01-01 02:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0000	5.0	27.0	32.0
3	2011-01-01 03:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	3.0	10.0	13.0
4	2011-01-01 04:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0000	0.0	1.0	1.0
...
10881	2012-12-19 19:00:00	4.0	0.0	1.0	1.0	15.58	19.695	50.0	26.0027	7.0	329.0	336.0
10882	2012-12-19 20:00:00	4.0	0.0	1.0	1.0	14.76	17.425	57.0	15.0013	10.0	231.0	241.0
10883	2012-12-19 21:00:00	4.0	0.0	1.0	1.0	13.94	15.910	61.0	15.0013	4.0	164.0	168.0
10884	2012-12-19 22:00:00	4.0	0.0	1.0	1.0	13.94	17.425	61.0	6.0032	12.0	117.0	129.0
10885	2012-12-19 23:00:00	4.0	0.0	1.0	1.0	13.12	16.665	66.0	8.9981	4.0	84.0	88.0

10886 rows x 12 columns

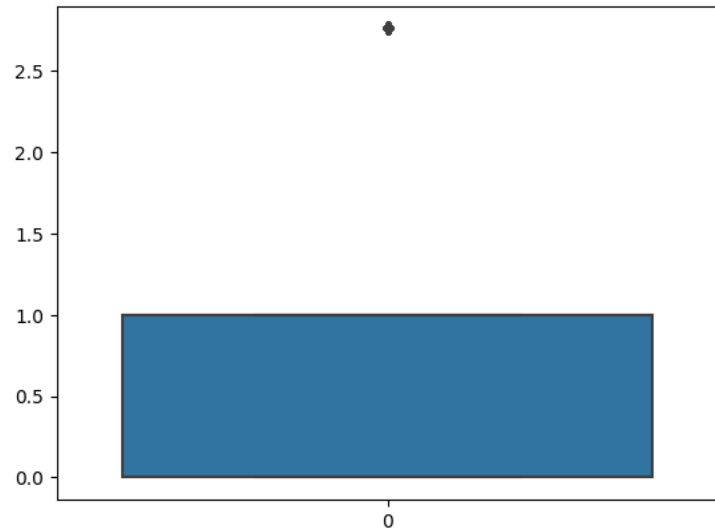
```
sns.boxplot(data = df['season'])
# we can see the outlier has been removed
```

<Axes: >



```
sns.boxplot(data = df['workingday'])  
# there are 1 outliers here
```

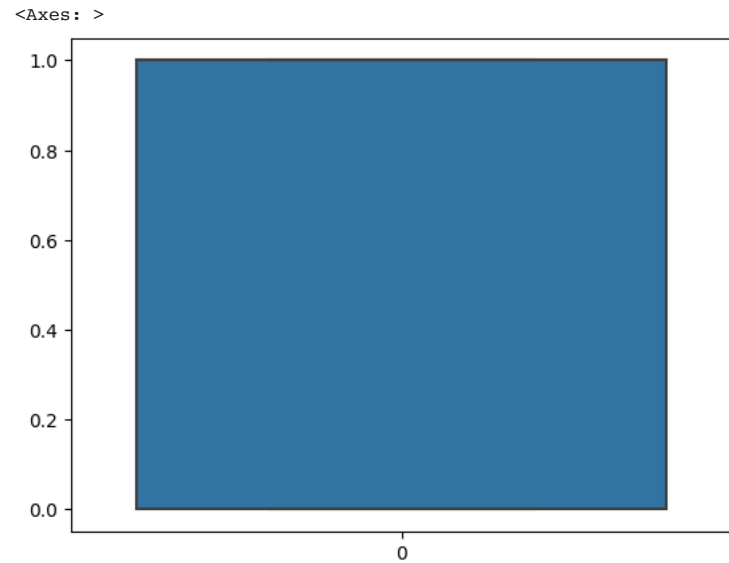
<Axes: >



```
imputing_outliers_with_mean(df, 'workingday')  
# removing the outlier in workingday column
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0000	3.0	13.0	16.0	

```
sns.boxplot(data = df['workingday'])  
# the outlier in workingday column has been removed
```



```
sns.boxplot(data = df['weather'])  
# Here 4 is not an outlier as weather with 4 value is Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
```

<Axes: >

df.head()

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0	3.0	13.0	16.0
1	2011-01-01 01:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0	8.0	32.0	40.0
2	2011-01-01 02:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0	5.0	27.0	32.0
3	2011-01-01 03:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0	3.0	10.0	13.0
4	2011-01-01 04:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0	0.0	1.0	1.0

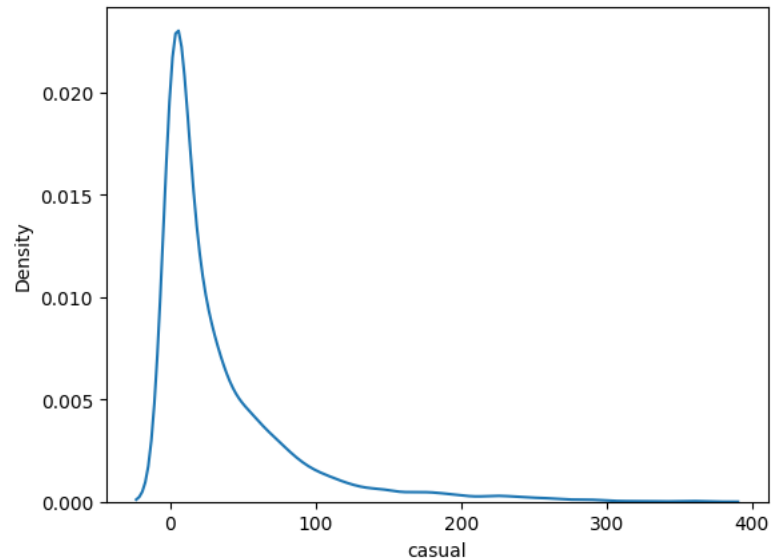


```
print("Average (mean) number of casual users per day : ",df['casual'].mean())
print("Average (median) number of casual users per day : ",df['casual'].median())
sns.kdeplot(data = df['casual'])
# we can see the distribution of casual users
# most of the days there are very few casual users who use Yulu Bikes
# On a given day the number of casual users is given by mean & median
```

Average (mean) number of casual users per day : 35.32838103628704

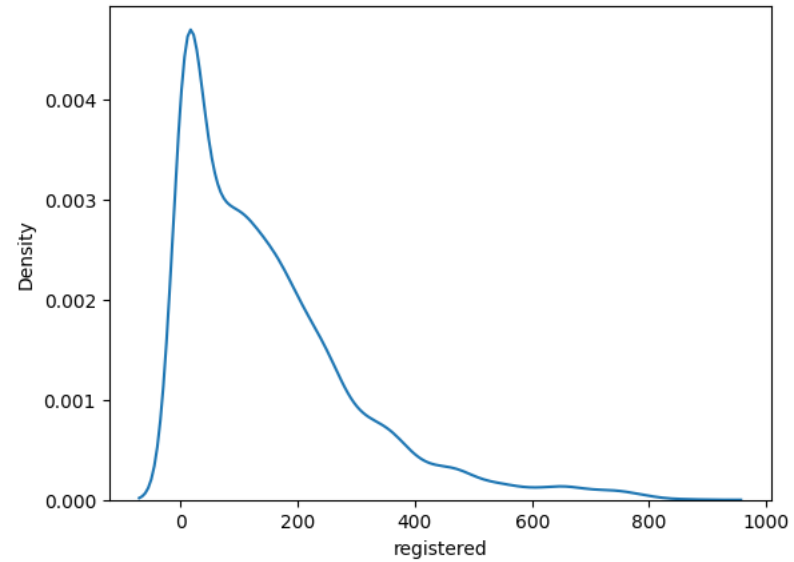
Average (median) number of casual users per day : 16.0

<Axes: xlabel='casual', ylabel='Density'>



```
print("Average (Mean) number of registered users per day : ",df['registered'].mean())
print("Average (Median) number of registered users per day : ",df['registered'].median())
sns.kdeplot(data = df['registered'])
# we can see the distribution of registered users
# On a given day the number of registered users is given by mean & median
```

```
Average (Mean) number of registered users per day : 152.3087227596014  
Average (Median) number of registered users per day : 115.0  
<Axes: xlabel='registered', ylabel='Density'>
```



```
print("Average (Mean) number of bikes used per day : ",df['count'].mean())  
print("Average (Median) number of bikes used per day : ",df['count'].median())  
sns.kdeplot(data = df['count'])  
# On a given day the number of bikes used is given by mean & median
```


Average (Mean) number of bikes used per day : 187.57420135596368
 Average (Median) number of bikes used per day : 140.0

```
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1.0	0.0	0.0	1.0	9.84	14.395	81.0	0.0	3.0	13.0	16.0
1	2011-01-01 01:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0	8.0	32.0	40.0
2	2011-01-01 02:00:00	1.0	0.0	0.0	1.0	9.02	13.635	80.0	0.0	5.0	27.0	32.0
3	2011-01-01 03:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0	3.0	10.0	13.0
4	2011-01-01 04:00:00	1.0	0.0	0.0	1.0	9.84	14.395	75.0	0.0	0.0	1.0	1.0

```
# line plot for season,holiday,workingday,weather,temp,atemp,humidity,windspeed vs casual,registered, count
```

```
# plotting numerical variables against count using scatterplot
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.lineplot(data=df, x=num_cols[index], y='count', ax=axis[row, col])
        index += 1
```

```
plt.show()
```

```
# Inferences from the plots
```

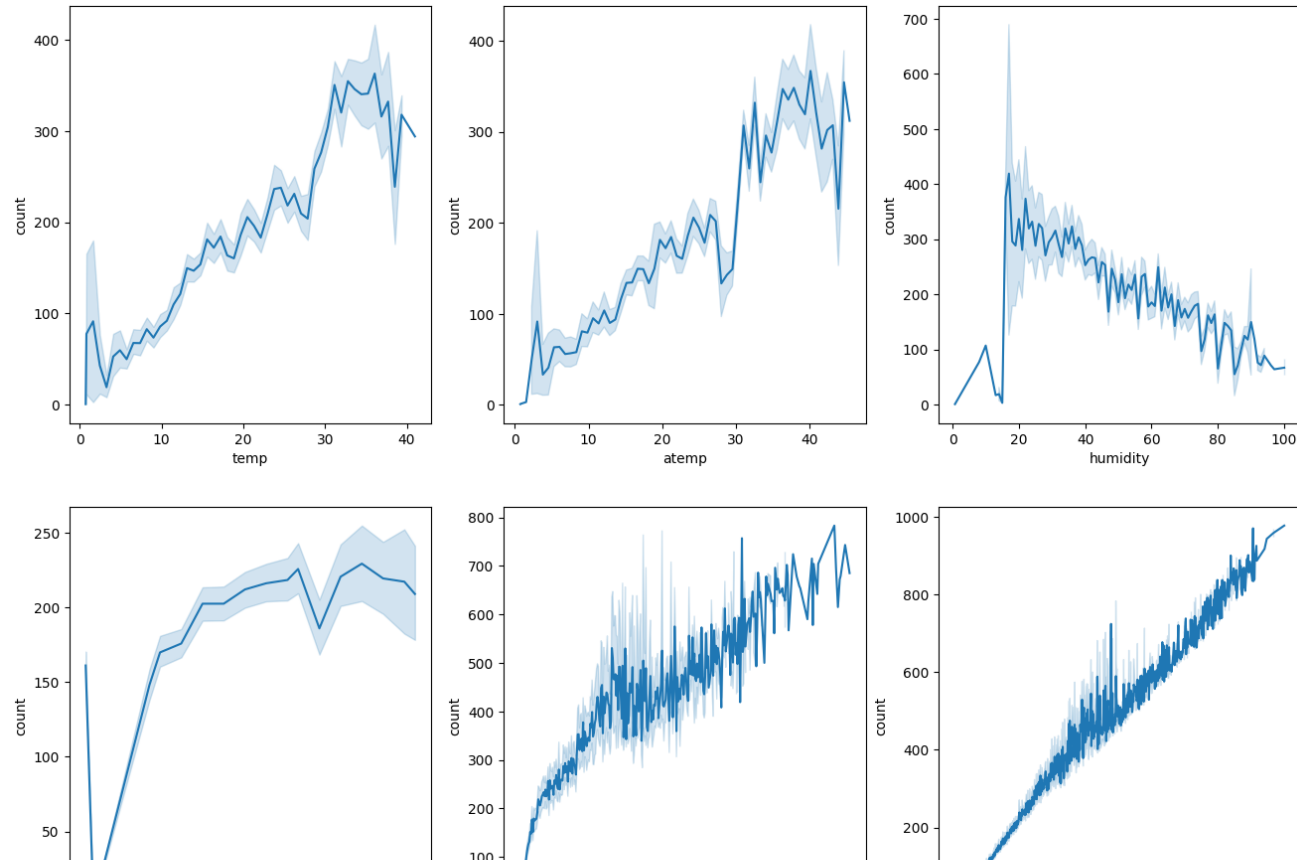
```
# as temp increases more number of bikes are in use
```

```
# as atemp increases more number of bikes are in use
```

```
# as humidity increases less number of bikes are in use
```

```
# when more bikes are in use we can see more registered and casual users tend to rent more bikes as this is obvious from the lineplots
```





```
# plotting categorical variables against count using scatterplot
cat_cols = ['season', 'holiday', 'workingday', 'weather']
```

```
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))
```

```
index = 0
for row in range(2):
    for col in range(2):
        sns.boxplot(data=df, x=cat_cols[index], y='count', ax=axis[row, col])
        index += 1
```

```
plt.show()
```

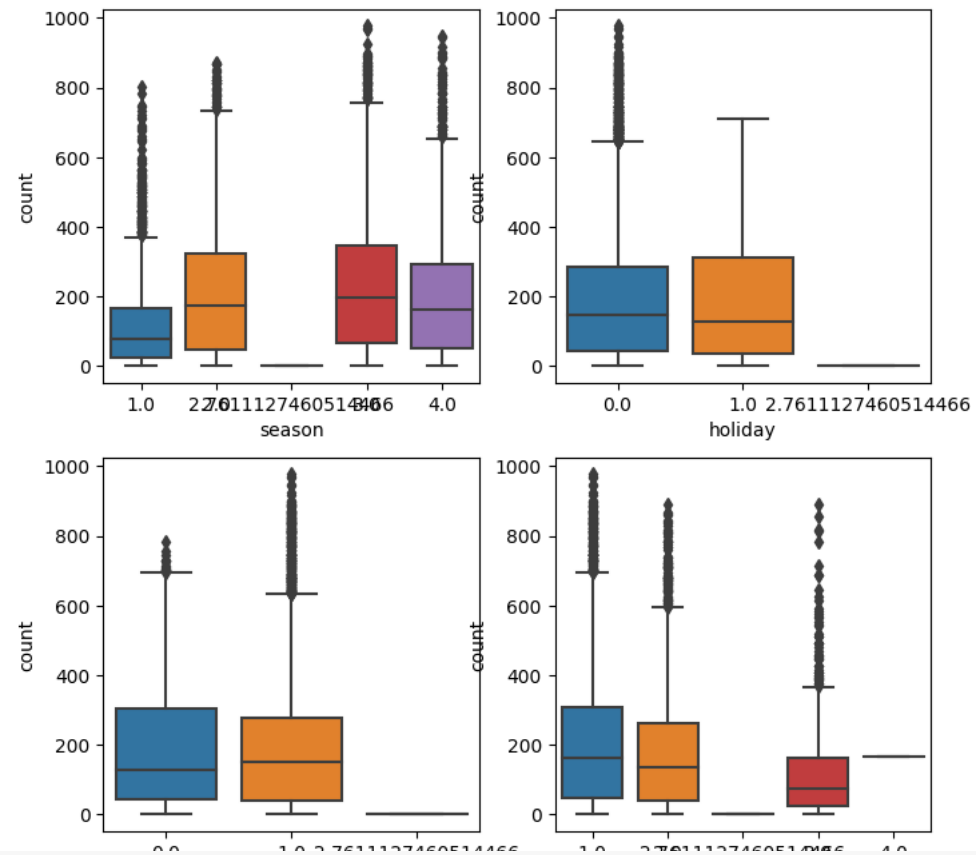
```
# Observations
```

```
# In summer and fall seasons more bikes are rented as compared to other seasons.
```

```
# Whenever its a holiday more bikes are rented.
```

```
# It is also clear from the workingday also that whenever day is holiday or weekend, slightly more bikes were rented.
```

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



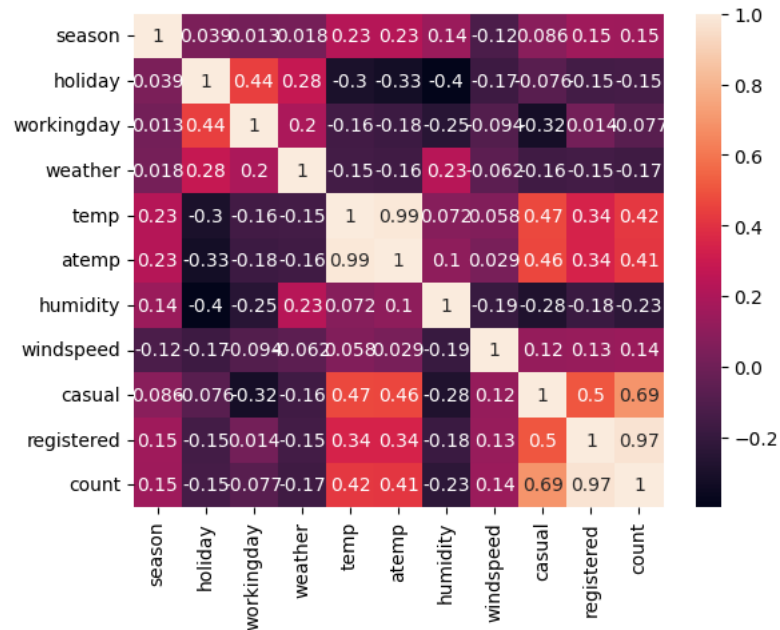
```
df.corr()
# correlation table between the columns of the dataset
```

```
<ipython-input-203-357a27145bdc>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only va
df.corr()
```

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	
season	1	0.000000	0.030338	0.012045	0.018281	0.227261	0.227070	0.140756	-0.121895	0.085583	0.154258	0.151010

```
sns.heatmap(df.corr(), annot=True)
plt.show()
# Heatmap for the columns of the dataset
# Observations
# Registered column is very strongly positively correlated to count
# Casual column is strongly positively correlated to count
# Casual is very negatively correlated to working day
```

```
<ipython-input-204-a7f823759449>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only va
sns.heatmap(df.corr(), annot=True)
```



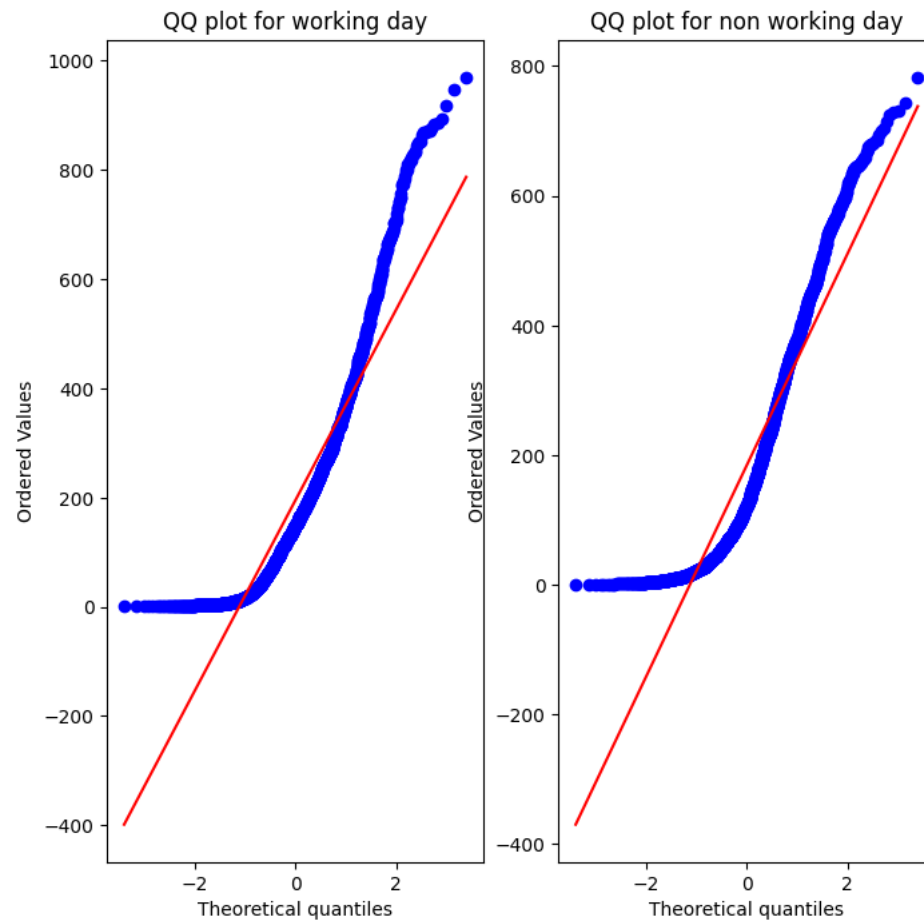
```
# Q-Q plots to check normality
# working day count of bikes vs non working day count of bikes

plt.figure(figsize = (8, 8))
plt.subplot(1, 2, 1)
plt.suptitle('QQ plots for the count of Yulu Bikes rented on workng and non working day')
probplot(df.loc[df['workingday'] == 1, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for working day')
plt.subplot(1, 2, 2)
probplot(df.loc[df['workingday'] == 0, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for non working day')
plt.plot()
```

```
# Observation
# the plot for working day does not follow the normal distribution as it is clearly visible from the QQ plot
```

```
[]
```

QQ plots for the count of Yulu Bikes rented on working and non working day



```
# Shapiro's test to test normality
# But the shapiro test won't give accurate results when sample size is too large

# H0 : The sample follows normal distribution
# Ha : The sample does not follows normal distribution

# Here we have taken a sample of 2000

print("For working day = 1")
test_stat, p_value = shapiro(df.loc[df['workingday'] == 1, 'count'].sample(2000))
```

```

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

print("For working day = 0")
test_stat, p_value = shapiro(df.loc[df['workingday'] == 0, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

# Observation :
# The sample does not follow normal distribution for both working and non working day

```

```

    For working day = 1
    p-value 3.7483430713117035e-38
    The sample does not follow normal distribution
    For working day = 0
    p-value 4.3026702088108806e-36
    The sample does not follow normal distribution

```

```

# Levene's test to test if difference in variance is significant

# H0 : Variances is equal
# H1 : Variances is not equal

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('Variances is not equal')
else:
    print('Variances is equal')

# Observation
# As per Levene's test both working day count of bikes and non working day count of bikes has equal variances

    p-value 0.9739090670332823
    Variances is equal

```

```

# Hypothesis Testing

# H0 : Working Day has no effect on the number of electric cycles rented
# Ha : Working Day has an effect on the number of electric cycles rented

# Working day vs count
working_day_0_count = df[df['workingday'] == 0]['count'].values
working_day_1_count = df[df['workingday'] == 1]['count'].values

t_stat, p_val= ttest_ind(working_day_0_count,working_day_1_count)

```

```
print("Test Statistic : ",t_stat)
print("P-value : ",p_val)

alpha = 0.05
# taking alpha as 0.05 significant level
if p_val < alpha :
    print("Reject H0")
    print("Working Day has an effect on the number of electric cycles rented")
else :
    print("We do not reject H0")
    print("Working Day has no effect on the number of electric cycles rented ")
```

```
Test Statistic : -1.229724557044485
P-value : 0.21882746848482007
We do not reject H0
Working Day has no effect on the number of electric cycles rented
```

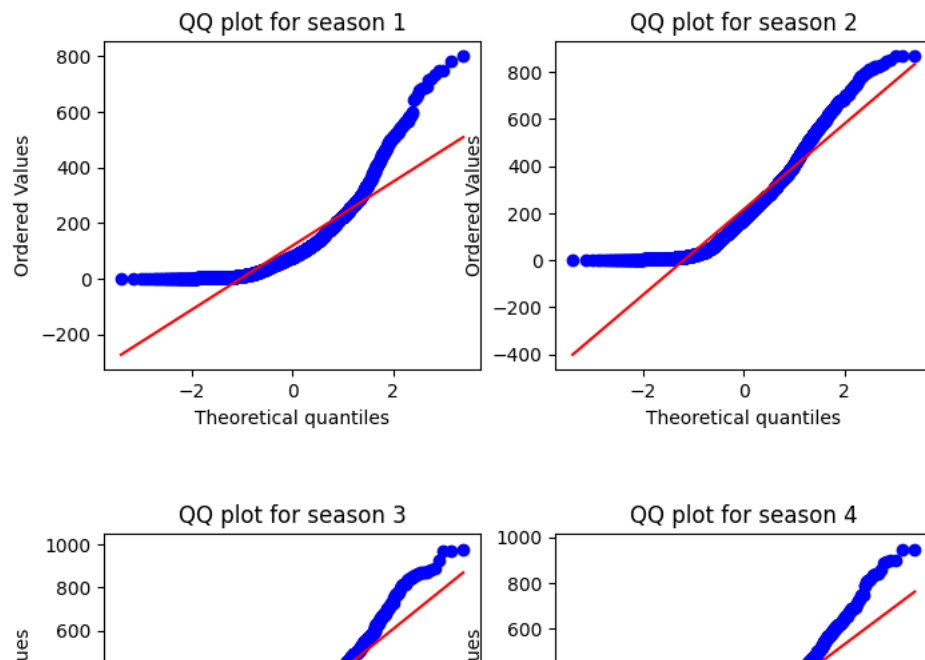
```
# Q-Q plots to check normality
# Count of bikes across all seasons

plt.figure(figsize = (8, 8))
plt.subplot(2, 2, 1)
plt.suptitle('QQ plots for the count of Yulu Bikes rented across seasons')
probplot(df.loc[df['season'] == 1, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for season 1')
plt.subplot(2, 2, 2)
probplot(df.loc[df['season'] == 2, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for season 2')
plt.plot()
plt.subplot(2, 2, 3)
probplot(df.loc[df['season'] == 3, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for season 3')
plt.plot()
plt.subplot(2, 2, 4)
probplot(df.loc[df['season'] == 4, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for season 4')
plt.plot()

plt.subplots_adjust(hspace=0.5) # adjusting the space vertically

# Observation
# the plots for seasons does not follow the normal distribution as it is clearly visible from the QQ plots
```

QQ plots for the count of Yulu Bikes rented across seasons



```
# Shapiro's test to test normality
# But the shapiro test won't give accurate results when sample size is too large

# H0 : The sample follows normal distribution
# Ha : The sample does not follows normal distribution

# Here we have taken a sample of 2000

print("For season = 1")
test_stat, p_value = shapiro(df.loc[df['season'] == 1, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

print("For season = 2")
test_stat, p_value = shapiro(df.loc[df['season'] == 2, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
```



```
print("For season = 3")
test_stat, p_value = shapiro(df.loc[df['season'] == 3, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

print("For season = 4")
test_stat, p_value = shapiro(df.loc[df['season'] == 4, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

# Observation :
# The samples does not follow normal distribution for all seasons
```

```
For season = 1
p-value 5.465064010866787e-44
The sample does not follow normal distribution
For season = 2
p-value 6.928039065267644e-34
The sample does not follow normal distribution
For season = 3
p-value 7.660134210261979e-32
The sample does not follow normal distribution
For season = 4
p-value 5.913557938354869e-35
The sample does not follow normal distribution
```

```
# Levene's test to test if difference in variance is significant
```

```
# H0 : Variances is equal
# H1 : Variances is not equal
```

```
test_stat, p_value = levene(df.loc[df['season'] == 1, 'count'].sample(2000),
                             df.loc[df['season'] == 2, 'count'].sample(2000),
                             df.loc[df['season'] == 3, 'count'].sample(2000),
                             df.loc[df['season'] == 4, 'count'].sample(2000))
```

```
print('p-value', p_value)
```

```
if p_value < 0.05:
    print('Variances is not equal')
else:
    print('Variances is equal')
```

```
# Observation
# As per Levene's test variances across all seasons for bike count is not equal
```

```
p-value 2.3353999333167597e-92
Variances is not equal
```

```
# Kruskal Wallis Test
# since the assumptions of ANOVA failed we perform Kruskal Wallis Test

# H0 : Mean no. of cycles rented is same for different seasons
# Ha : Mean no. of cycles rented is different for different seasons

alpha = 0.05

test_stat, p_value = kruskal(df.loc[df['season'] == 1, 'count'],
                             df.loc[df['season'] == 2, 'count'],
                             df.loc[df['season'] == 3, 'count'],
                             df.loc[df['season'] == 4, 'count'])

print('Test Statistic =', test_stat)
print('p value =', p_value)

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

# Observation
# Mean no. of cycles rented is different for different seasons

Test Statistic = 664.008132882048
p value = 1.3370428160922689e-143
Reject Null Hypothesis

# Hypothesis Testing ANOVA

# H0 : Number of cycles rented is same across all seasons
# Ha : Number of cycles rented is not same across all seasons

# season vs count

season_1 = df[df['season'] == 1]['count'].values
season_2 = df[df['season'] == 2]['count'].values
season_3 = df[df['season'] == 3]['count'].values
season_4 = df[df['season'] == 4]['count'].values

f_stat_season, p_val_anova_season = f_oneway(season_1, season_2, season_3, season_4)

print("F statistic : ", f_stat_season)
print("p_val_anova : ", p_val_anova_season)

if p_val_anova_season < alpha :
    print("We reject H0")
    print("Number of cycles rented is not same across all seasons")
else :
    print("We fail to reject H0")
    print("Number of cycles rented is same across all seasons")

F statistic : 224.1553523974667
p_val_anova : 5.131170054025797e-141
We reject H0
Number of cycles rented is not same across all seasons
```

```
# Q-Q plots to check normality
# Count of bikes across all seasons

plt.figure(figsize = (8, 8))
plt.subplot(2, 2, 1)
plt.suptitle('QQ plots for the count of Yulu Bikes rented across weather conditions')
probplot(df.loc[df['weather'] == 1, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for weather 1')
plt.subplot(2, 2, 2)
probplot(df.loc[df['weather'] == 2, 'count'].sample(2000), plot = plt, dist = 'norm')
plt.title('QQ plot for weather 2')
plt.plot()
plt.subplot(2, 2, 3)
probplot(df.loc[df['weather'] == 3, 'count'], plot = plt, dist = 'norm')
plt.title('QQ plot for weather 3')
plt.plot()
plt.subplot(2, 2, 4)
probplot(df.loc[df['weather'] == 4, 'count'], plot = plt, dist = 'norm')
plt.title('QQ plot for weather 4')
plt.plot()

plt.subplots_adjust(hspace=0.5) # adjusting the space vertically

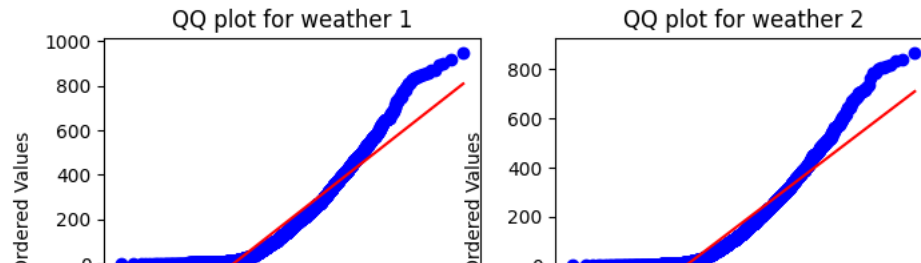
# Observation
# the plots for weather conditions does not follow the normal distribution as it is clearly visible from the QQ plots
# for weather = 4 there isn't much data available
```

```

/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_mstats_common.py:182: RuntimeWarning: invalid value encountered in double_scalars
  slope = ssxym / ssxm
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_mstats_common.py:196: RuntimeWarning: invalid value encountered in sqrt
  t = r * np.sqrt(df / ((1.0 - r + TINY)*(1.0 + r + TINY)))
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_mstats_common.py:199: RuntimeWarning: invalid value encountered in double_scalars
  slope_stderr = np.sqrt((1 - r**2) * ssym / ssxm / df)

```

QQ plots for the count of Yulu Bikes rented across weather conditions



```

# Shapiro's test to test normality
# But the shapiro test won't give accurate results when sample size is too large

# H0 : The sample follows normal distribution
# Ha : The sample does not follows normal distribution

# Here we have taken a sample of 2000

print("For weather = 1")
test_stat, p_value = shapiro(df.loc[df['weather'] == 1, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

print("For weather = 2")
test_stat, p_value = shapiro(df.loc[df['weather'] == 2, 'count'].sample(2000))

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

print("For weather = 3")
test_stat, p_value = shapiro(df.loc[df['weather'] == 3, 'count'])

print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')

'''

```

```
print("For weather = 4")
test_stat, p_value = shapiro(df.loc[df['weather'] == 4, 'count'])
```

```
print('p-value', p_value)
if p_value < 0.05:
    print('The sample does not follow normal distribution')
else:
    print('The sample follows normal distribution')
,,,
```

```
# Observation :
# The samples does not follow normal distribution for all weather conditions 1,2 & 3
# for weather condition 4 there isn't mcuh data and so we can't perform shapiro test
```

```
For weather = 1
p-value 1.1263513401182233e-34
The sample does not follow normal distribution
For weather = 2
p-value 1.06068177495515e-36
The sample does not follow normal distribution
For weather = 3
p-value 5.157817067869946e-32
The sample does not follow normal distribution
'\nprint("For weather = 4")\ntest_stat, p_value = shapiro(df.loc[df['weather'] == 4, 'count'])\n\nprint('\p-value', p_value)\nif p_value < 0.05:\n    print('\The sample does
not follow normal distribution')\nelse:\n    print('\The sample follows normal distribution')\n'
```

```
# Levene's test to test if difference in variance is significant
```

```
# H0 : Variances is equal
# H1 : Variances is not equal
```

```
test_stat, p_value = levene(df.loc[df['weather'] == 1, 'count'].sample(2000),
                             df.loc[df['weather'] == 2, 'count'].sample(2000),
                             df.loc[df['weather'] == 3, 'count'],
                             df.loc[df['weather'] == 4, 'count'])
```

```
print('p-value', p_value)
```

```
if p_value < 0.05:
    print('Variances is not equal')
else:
    print('Variances is equal')
```

```
# Observation
# As per Levene's test variances across all weather conditions for bike count is not equal
```

```
p-value 1.5030721480020107e-24
Variances is not equal
```

```
# Kruskal Wallis Test
# since the assumptions of ANOVA failed we perform Kruskal Wallis Test
```

```
# H0 : Mean no. of cycles rented is same for different weather conditions
# Ha : Mean no. of cycles rented is different for different weather conditions
```

```
alpha = 0.05
```

```

test_stat, p_value = kruskal(df.loc[df['weather'] == 1, 'count'],
                             df.loc[df['weather'] == 2, 'count'],
                             df.loc[df['weather'] == 3, 'count'],
                             df.loc[df['weather'] == 4, 'count'])
print('Test Statistic =', test_stat)
print('p value =', p_value)

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

# Observation
# Mean no. of cycles rented is different for different weather conditions

```

```

Test Statistic = 184.73490669792486
p value = 8.372812320181599e-40
Reject Null Hypothesis

```

```

# Hypothesis Testing ANOVA

# H0 : Number of cycles rented is same across all weather conditions
# Ha : Number of cycles rented is not same across all weather conditions

# weather vs count
weather_1 = df[df['weather'] == 1]['count'].values
weather_2 = df[df['weather'] == 2]['count'].values
weather_3 = df[df['weather'] == 3]['count'].values
weather_4 = df[df['weather'] == 4]['count'].values

f_stat_weather,p_val_anova_weather = f_oneway(weather_1,weather_2,weather_3,weather_4)

print("F statistic : ",f_stat_weather)
print("p_val_anova : ",p_val_anova_weather)

if p_val_anova_weather < alpha :
    print("We reject H0")
    print("Number of cycles rented is not same across all weather conditions")
else :
    print("We fail to reject H0")
    print("Number of cycles rented is same across all weather conditions")

F statistic : 61.31565239458179
p_val_anova : 2.697113772681018e-39
We reject H0
Number of cycles rented is not same across all weather conditions

```

```

# Hypothesis Testing Chi Square

# H0 : Weather is independent on season
# Ha : Weather is dependent on season

# weather vs season
contingency_table = pd.crosstab(df["season"], df["weather"])

```

```

print("Contingency Table : \n",contingency_table)

chi_test_stat, p_value_chi2, dof_chi2, expected_chi2 = chi2_contingency(contingency_table.values)

print("Test statistic : ",chi_test_stat)
print("P - value : ",p_value_chi2)

if p_value_chi2 < alpha :
    print("Reject H0")
    print("Weather is dependent on season")
else :
    print("We do not reject H0")
    print("Weather is independent on season")

```

```

Contingency Table :
  weather    1.000000    2.000000    2.761113    3.000000    4.000000
season
1.000000      1662         697          0         185          1
2.000000      1764         696          0         214          0
2.761113          0          0         248          0          0
3.000000      1921         599          0         188          0
4.000000      1692         801          0         218          0
Test statistic : 10939.073010236192
P - value : 0.0
Reject H0
Weather is dependent on season

```

```

# Observations
# 1. All the observations are given at the cell itself
# 2. By looking at the line plots of temp & atemp vs count of bikes that are being rented
#    we can see that around 35 degree Celsius more
#    number of bikes are being rented out.
# 3. As temp increases more number of bikes are in use
# 4. As atemp increases more number of bikes are in use
# 5. As humidity increases fewer bikes are in use
# 6. when more bikes are in use we can see more registered and casual users tend to rent
#    more bikes as this is obvious from the lineplots
# 7. In summer and fall seasons more bikes are rented as compared to other seasons
# 8. Whenever its a holiday more bikes are rented
# 9. It is also clear from the workingday also that whenever day is holiday or weekend, slightly
#    more bikes were rented
# 10. Whenever there is rain, thunderstorm, snow or fog, there were less bikes were rented
# 11. Weather and Season are dependent

```

```

# Recommendations
# 1. Since more bikes are being rented out at 35 degree celsius and India is
#    a tropical country with temperatures being around 35 degree celsius
#    on most of the days, we can advice Yulu to have more bikes in place during the summer months
# 2. We can advice Yulu to start operations in more hotter regions like
#    South India where temperatures easily cross 40 degree celsius.
# 3. We can advice Yulu to stop opeartions in cooler places like hill stations
# 4. For cities in Northern India where there is extremism in climate during summer
#    and winter months (like Delhi), we can advice Yulu to have more bikes
#    in place during summer months and fewer bikes in winter season.
# 5. By looking at Humidity vs Count plot we can advice Yulu to maintian fewer bikes in
#    coastal cities like Chennai,Mumbai etc., where there is more humidity

```

```
# 6. As per T Test between Working day count vs Non working day count we can advice Yulu
#    to maintain same number of bikes on working and non working days as
#    working Day has no effect on the number of electric cycles rented.
# 7. From ANOVA tests we can advice Yulu to maintain the number of bikes as per the season
#    or weather requirement. There is no need to maintain
#    same number of bikes across all seasons, weather conditions
```