About Yulu

- Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.
- Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!
- Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
import warnings
warnings.filterwarnings('ignore')


df = pd.read_csv("Yulu.txt")
df
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000
...	...	...	...	...	...	...	...	...	...

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

```
df.shape
```

```
(10886, 12)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  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.keys()
```

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

```
df.describe()
```

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

```
df["season"].unique()
```

```
array([1, 2, 3, 4])
```

```
df["season"].nunique()
```

```
4
```

```
df["weather"].unique()
```

```
array([1, 2, 3, 4])
```

```
df["weather"].nunique()
```

```
4
```

```
df["temp"].unique()
```

```
array([ 9.84,  9.02,  8.2 , 13.12, 15.58, 14.76, 17.22, 18.86, 18.04,
       16.4 , 13.94, 12.3 , 10.66,  6.56,  5.74,  7.38,  4.92, 11.48,
        4.1 ,  3.28,  2.46, 21.32, 22.96, 23.78, 24.6 , 19.68, 22.14,
       20.5 , 27.06, 26.24, 25.42, 27.88, 28.7 , 30.34, 31.16, 29.52,
       33.62, 35.26, 36.9 , 32.8 , 31.98, 34.44, 36.08, 37.72, 38.54,
        1.64,  0.82, 39.36, 41.  ])
```

```
df["temp"].nunique()
```

```
49
```

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

	datetime
0	2011-01-01 00:00:00
1	2011-01-01 01:00:00
2	2011-01-01 02:00:00
3	2011-01-01 03:00:00
4	2011-01-01 04:00:00
...	...
10881	2012-12-19 19:00:00
10882	2012-12-19 20:00:00
10883	2012-12-19 21:00:00
10884	2012-12-19 22:00:00
10885	2012-12-19 23:00:00

10886 rows × 1 columns

dtype: datetime64[ns]

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().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

dtype: int64

```
df.head()
```

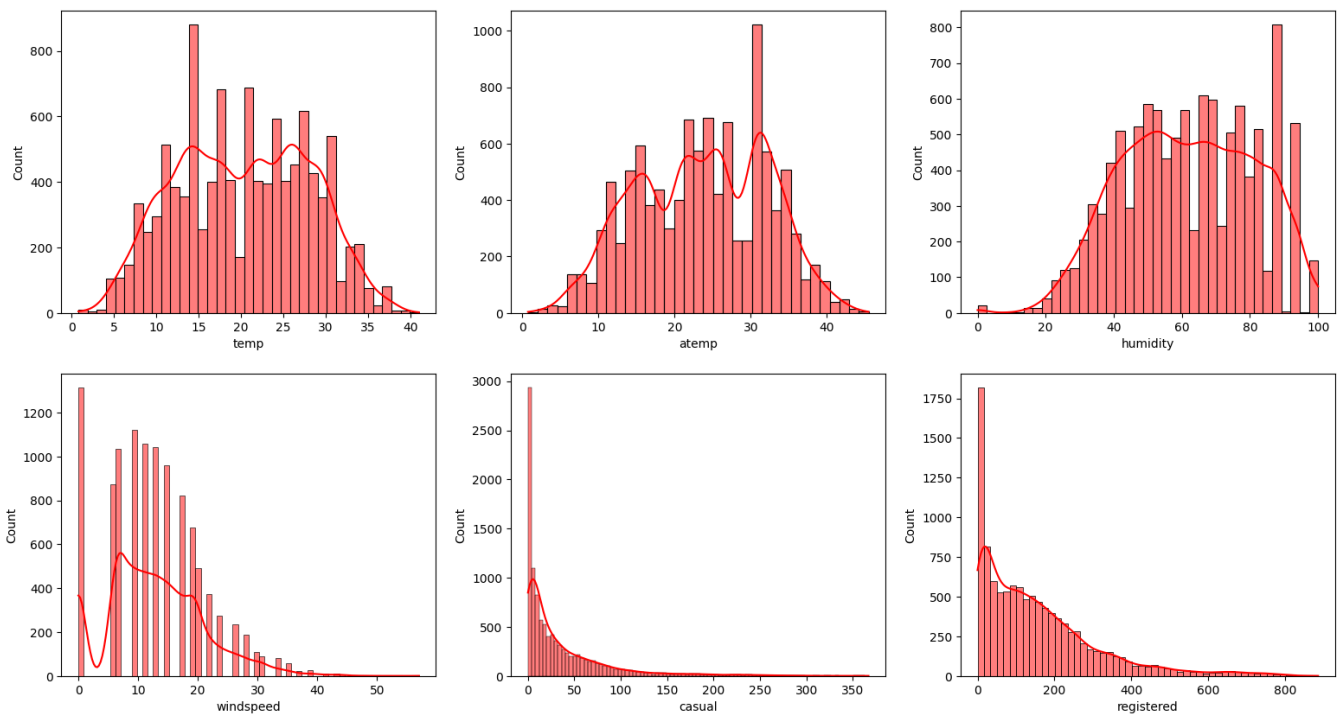
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	cas
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	
	2011-01									

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

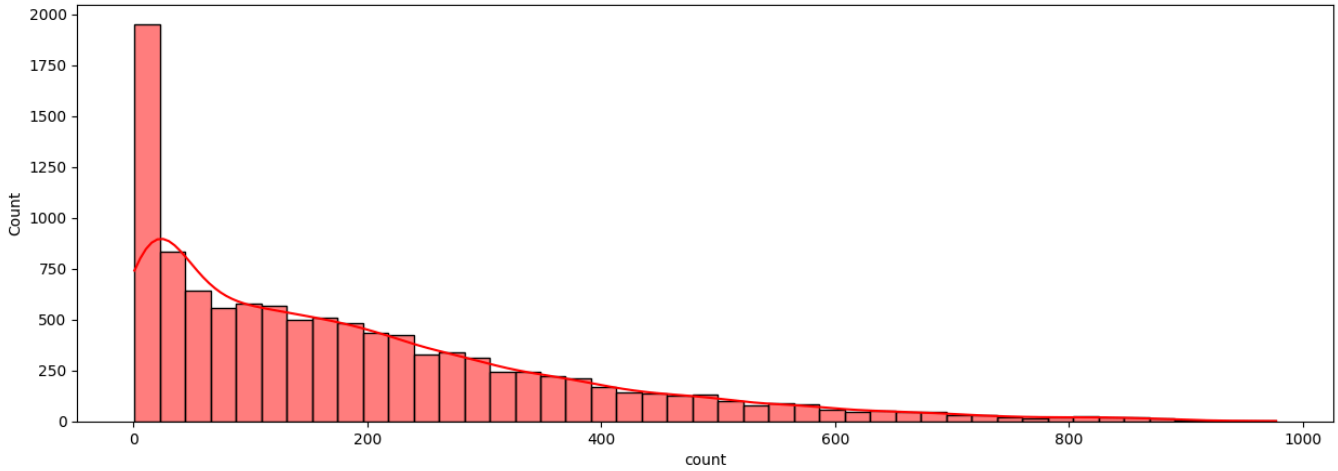
```
columns_cat=['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']

fig,axis=plt.subplots(nrows=2,ncols=3,figsize=(19,10))

index=0
for row in range(2):
  for col in range(3):
    sns.histplot(df[columns_cat[index]],ax=axis[row,col],kde=True, color = "red")
    index += 1
plt.show()
```



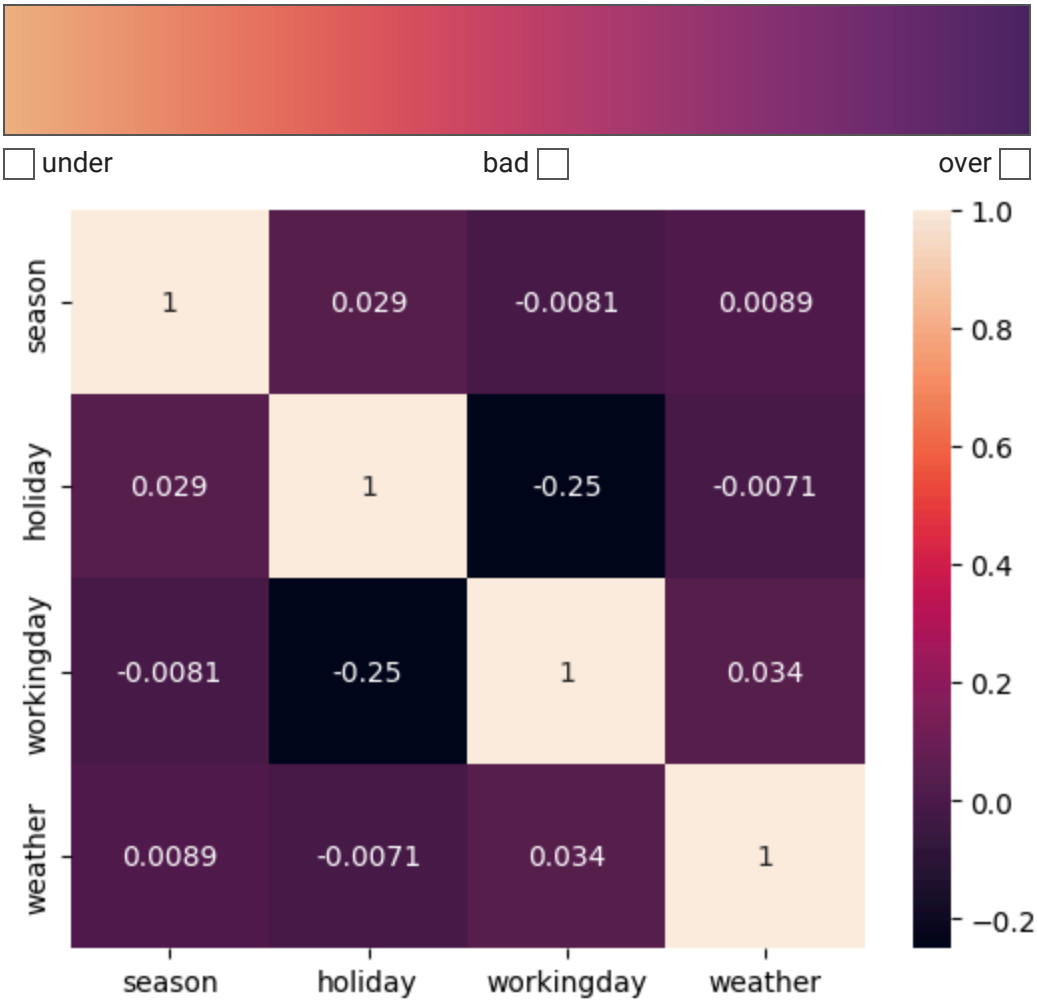

```
fig,axis=plt.subplots(nrows=1,ncols=1,figsize=(15,5))
sns.histplot(df[columns_cat[-1]], kde=True, color = "red")
plt.show()
```



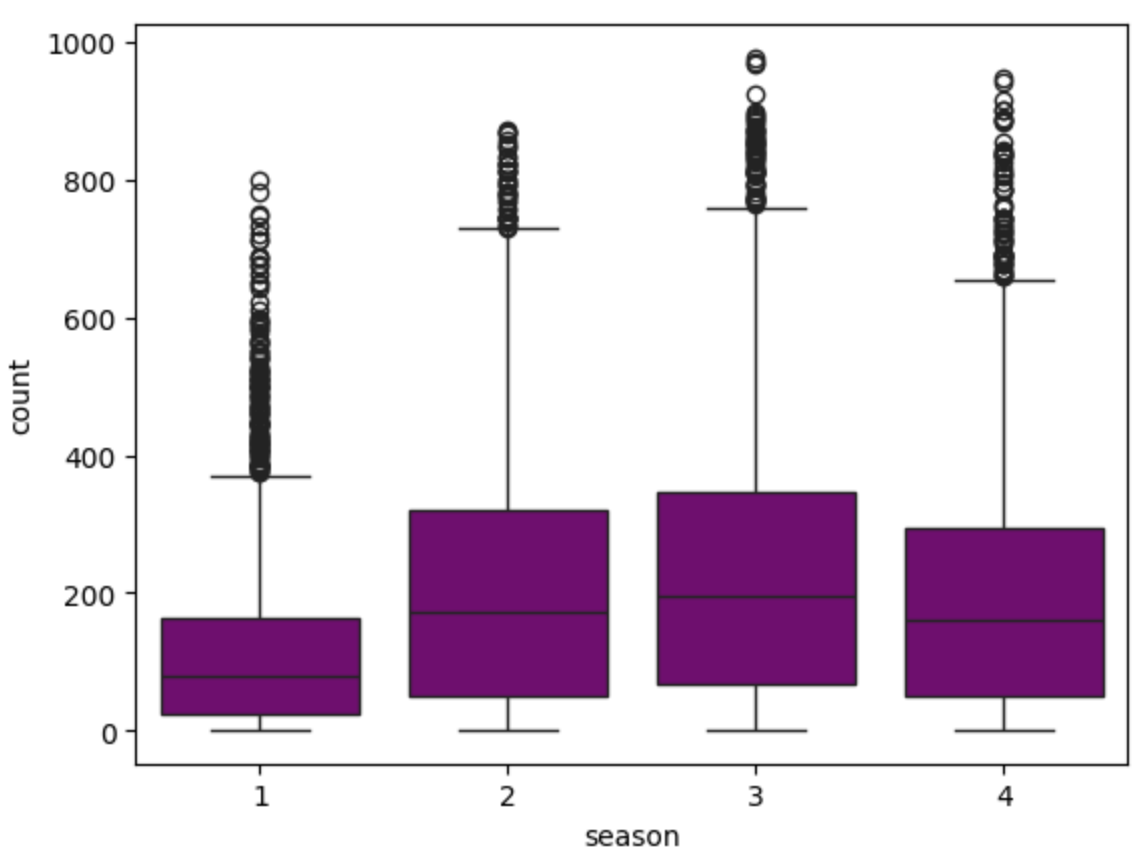

```
df1 = df[["season","holiday","workingday","weather"]]
```

```
sns.heatmap(df1.corr(),annot=True)
sns.color_palette("flare", as_cmap=True)
```

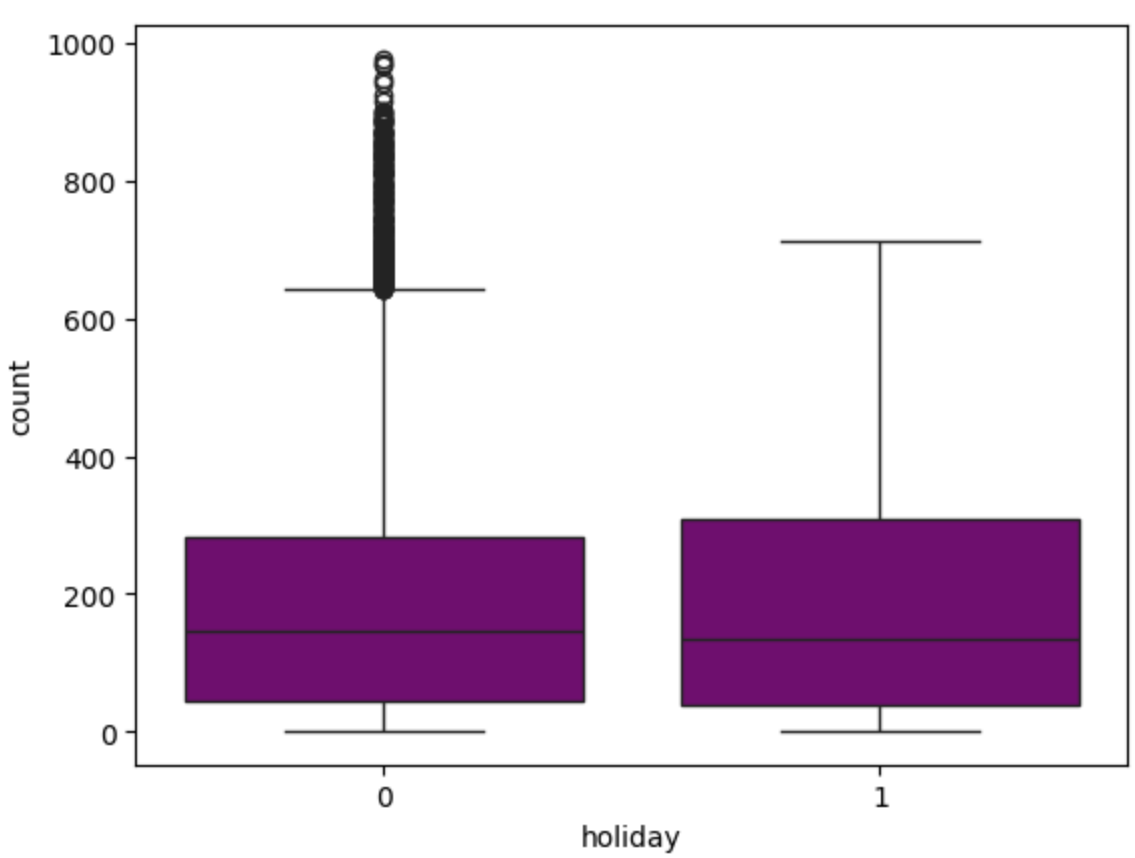
flare



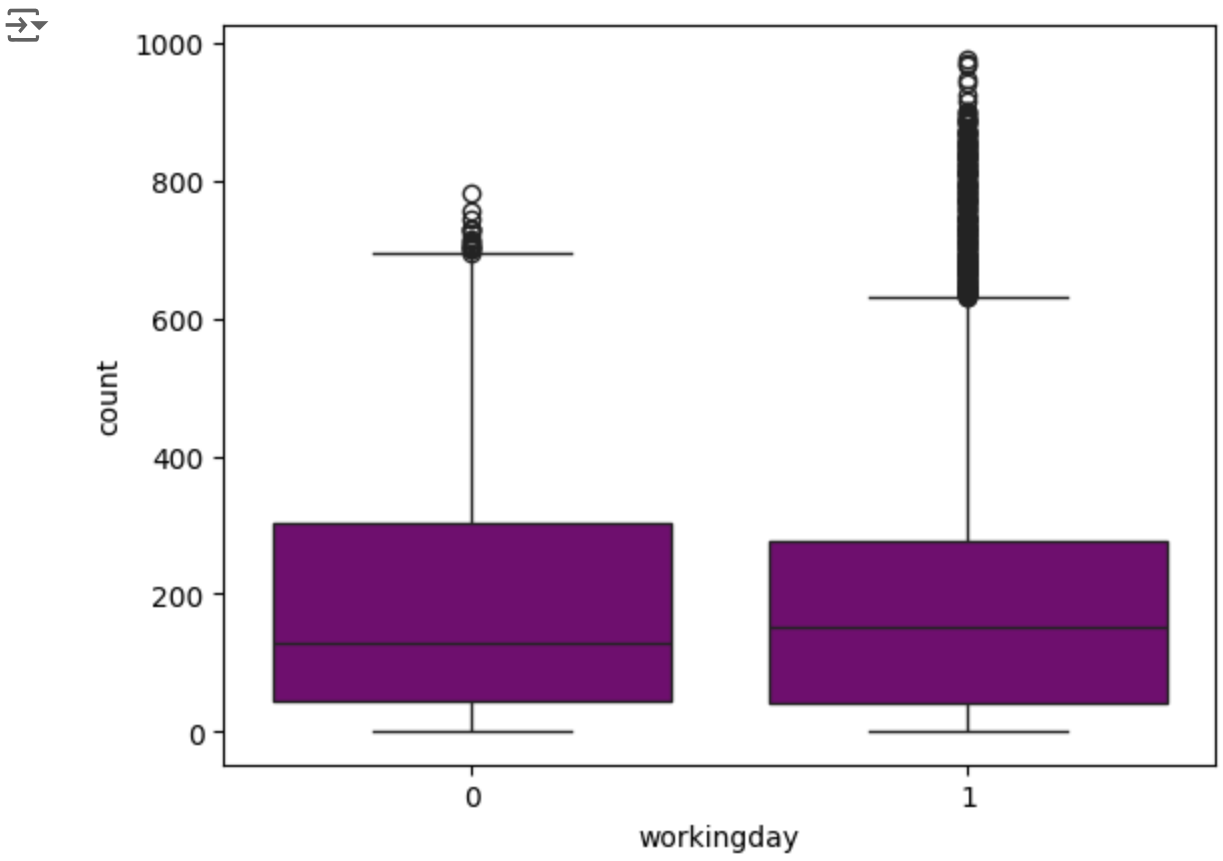

```
sns.boxplot(x="season",y="count",data=df,color = "purple")
plt.show()
```



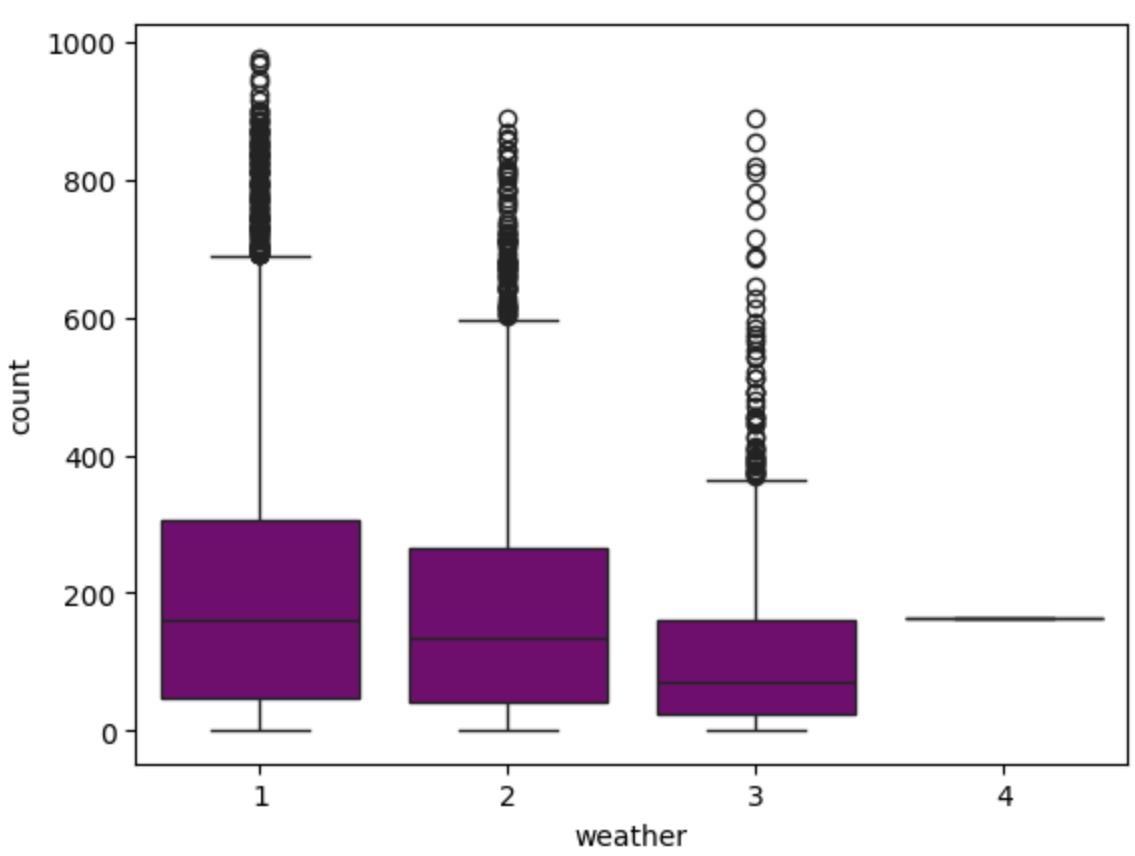

```
sns.boxplot(x="holiday",y="count",data=df,color = "purple")
plt.show()
```



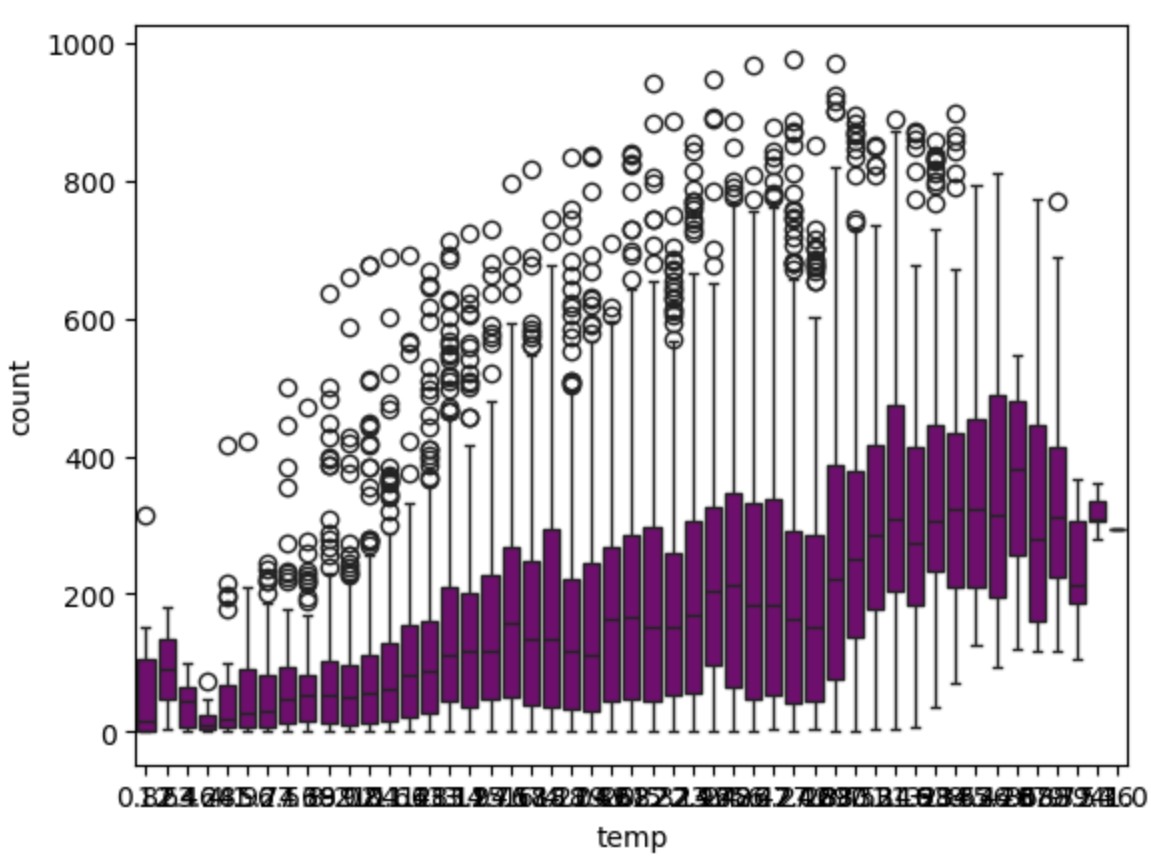
```
sns.boxplot(x="workingday",y="count",data=df,color = "purple")
plt.show()
```



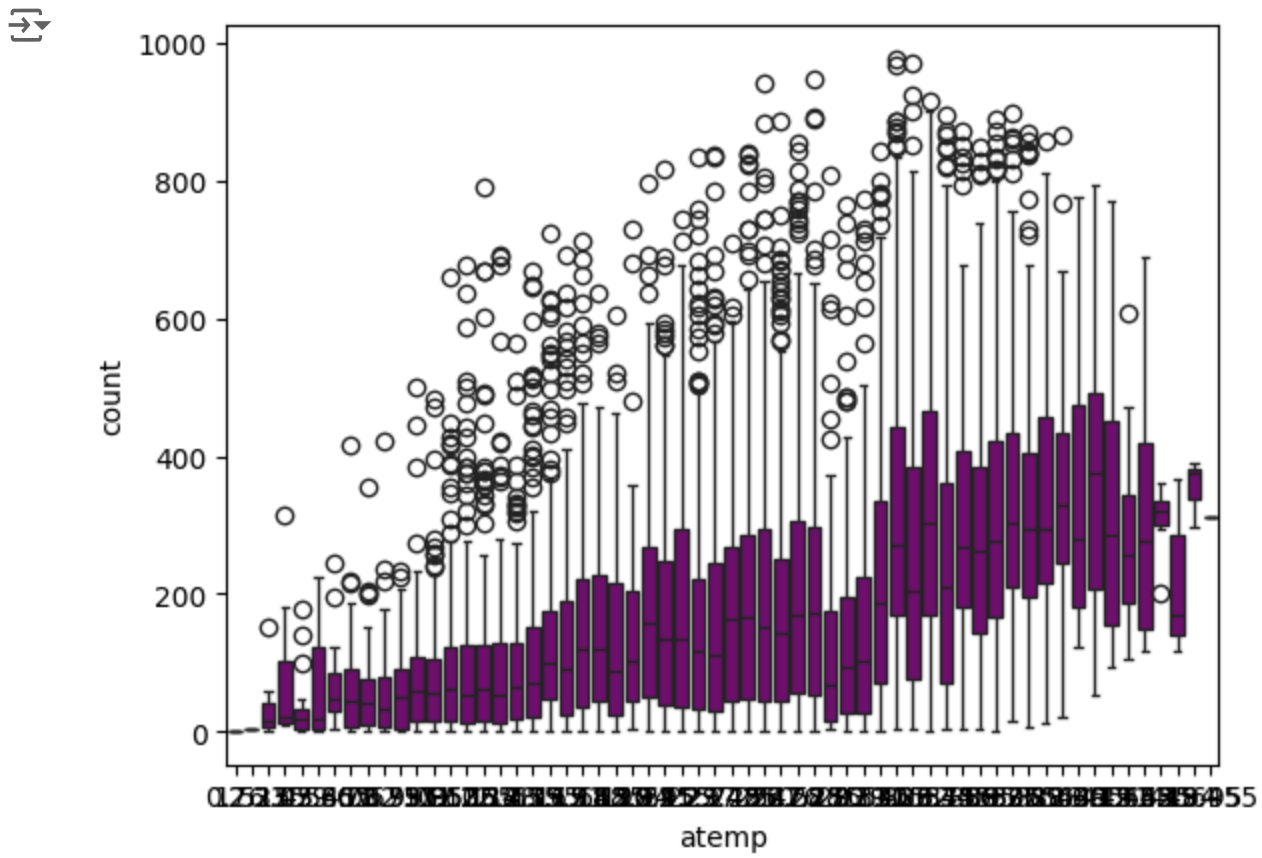

```
sns.boxplot(x="weather",y="count",data=df,color = "purple")
plt.show()
```



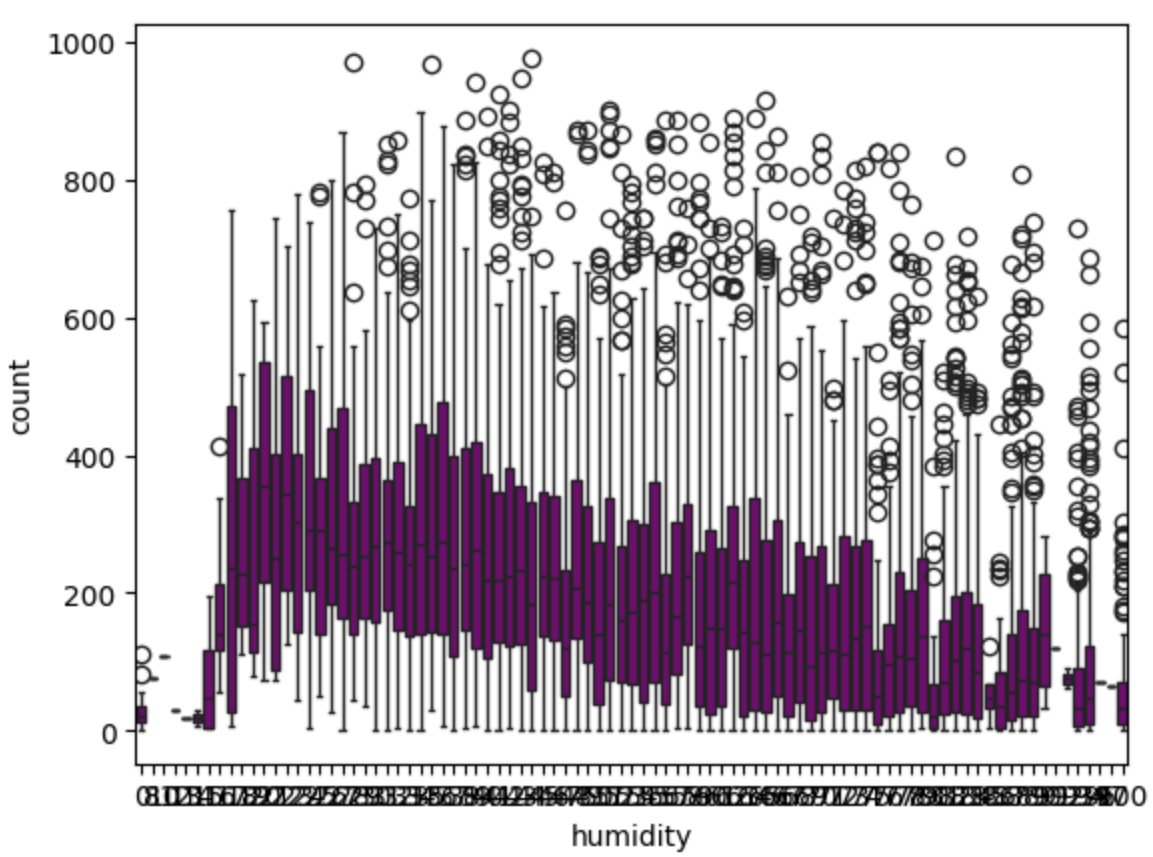

```
sns.boxplot(x="temp",y="count",data=df,color = "purple")
plt.show()
```



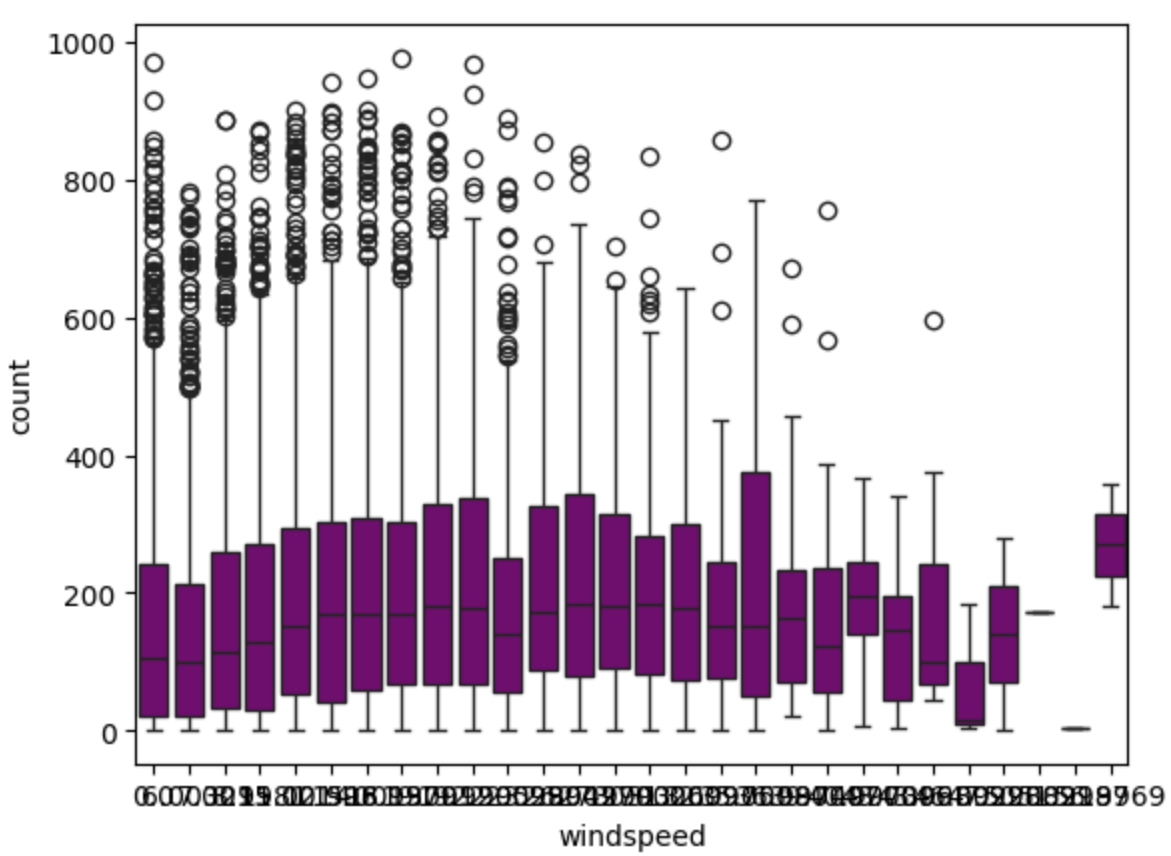
```
sns.boxplot(x="atemp",y="count",data=df,color = "purple")
plt.show()
```




```
sns.boxplot(x="humidity",y="count",data=df,color = "purple")
plt.show()
```




```
sns.boxplot(x="windspeed",y="count",data=df,color = "purple")
plt.show()
```




```
df["season"].value_counts()
```

season	count
4	2734
2	2733
3	2733
1	2686

dtype: int64

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

holiday	count
0	10575
1	311

dtype: int64

```
df["workingday"].value_counts()
```

workingday	count
1	7412
0	3474

dtype: int64

```
df["weather"].value_counts(normalize= True)*100
```

weather	proportion
1	66.066507
2	26.033437
3	7.890869
4	0.009186

dtype: float64

```
df["temp"].value_counts()
```

temp	count
14.76	467
26.24	453
28.70	427
13.94	413
18.86	406
22.14	403
25.42	403
16.40	400
22.96	395
27.06	394
24.60	390
12.30	385
21.32	362
17.22	356
13.12	356
29.52	353
10.66	332
18.04	328
20.50	327
30.34	299
9.84	294
15.58	255
9.02	248
31.16	242
8.20	229
27.88	224
23.78	203
32.80	202
11.48	181

19.68	170
6.56	146
33.62	130
5.74	107
7.38	106
31.98	98
34.44	80
35.26	76
4.92	60
36.90	46
4.10	44
37.72	34
36.08	23
3.28	11
0.82	7
38.54	7
39.36	6
2.46	5
1.64	2
41.00	1

dtype: int64

```
df["atemp"].value_counts()
```

atemp	count
31.060	671
25.760	423
22.725	406
20.455	400
26.515	395
16.665	381
25.000	365
33.335	364
21.210	356
30.305	350
15.150	338
21.970	328
24.240	327
17.425	314
31.820	299
34.850	283
27.275	282
32.575	272
11.365	271
14.395	269
29.545	257
19.695	255
15.910	254
12.880	247
13.635	237
34.090	224
12.120	195
28.790	175
23.485	170

10.605	166
35.605	159
9.850	127
18.180	123
36.365	123
37.120	118
9.090	107
37.880	97
28.030	80
7.575	75
38.635	74
6.060	73
39.395	67
6.820	63
8.335	63
18.940	45
40.150	45
40.910	39
5.305	25
42.425	24
41.665	23
3.790	16
4.545	11
3.030	7
43.940	7
2.275	7
43.180	7
44.695	3
0.760	2
1.515	1
45.455	1

dtype: int64

```
df["datetime"].value_counts()
```

datetime	count
2011-01-01 00:00:00	1
2012-05-01 21:00:00	1
2012-05-01 13:00:00	1
2012-05-01 14:00:00	1
2012-05-01 15:00:00	1
...	...
2011-09-02 04:00:00	1
2011-09-02 05:00:00	1
2011-09-02 06:00:00	1
2011-09-02 07:00:00	1
2012-12-19 23:00:00	1

10886 rows × 1 columns

dtype: int64

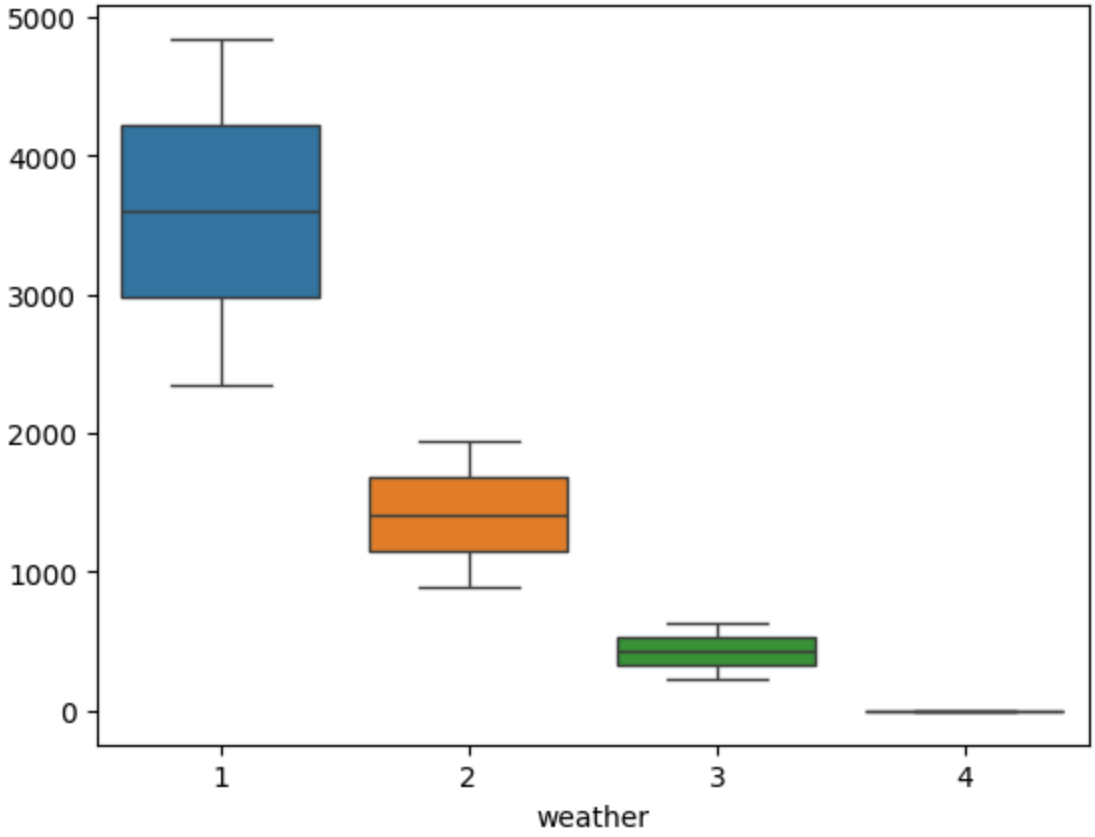
```
df1 = pd.crosstab(df["workingday"],df["weather"])
df1
```

workingday \ weather	1	2	3	4
0	2353	897	224	0
1	4839	1937	635	1

Next steps: Generate code with df1 | View recommended plots | New interactive sheet

```
sns.boxplot(df1)
```

```
<Axes: xlabel='weather'>
```




```
df.groupby(["workingday"])[["casual","registered","count"]].sum()
```

workingday	casual	registered	count
0	206037	448835	654872
1	186098	1244506	1430604

```
df.groupby(["weather"])[["casual","registered","count"]].sum()
```

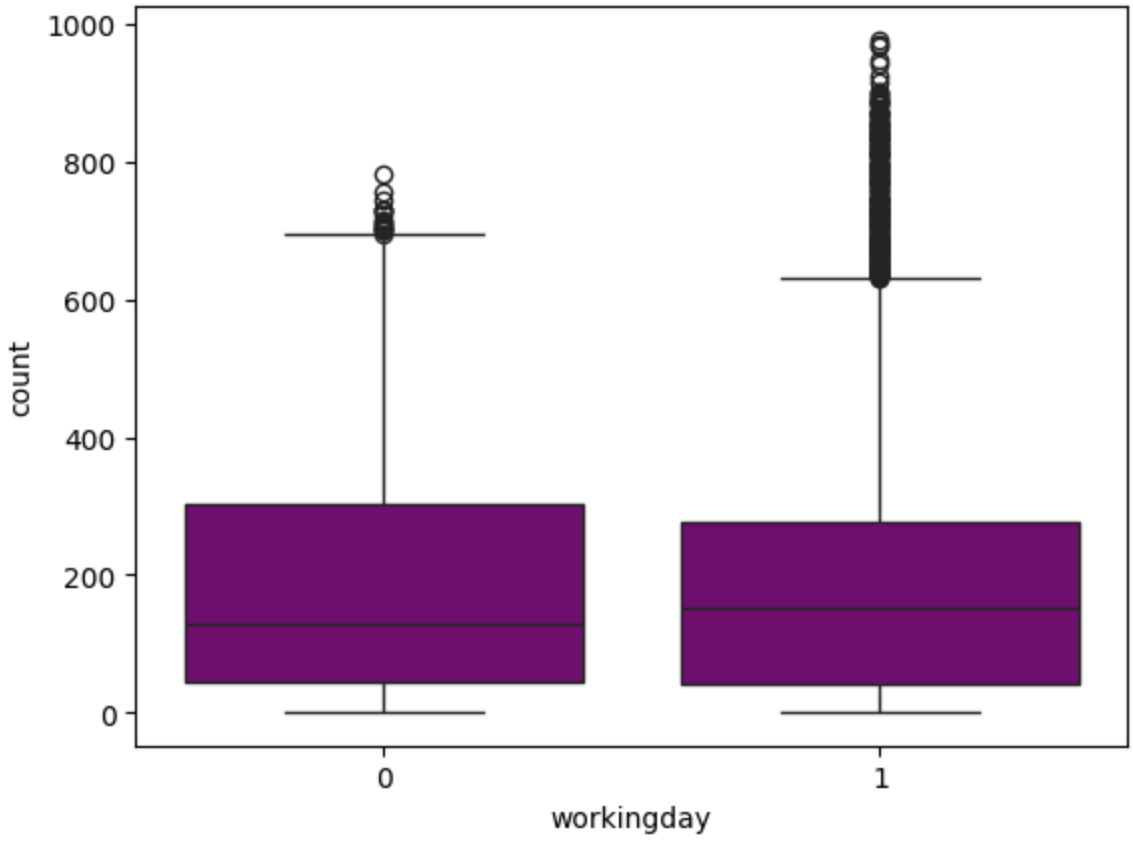
weather	casual	registered	count
1	289900	1186163	1476063
2	87246	419914	507160
3	14983	87106	102089
4	6	158	164

```
df.groupby(["season"])[["casual","registered","count"]].sum()
```

season	casual	registered	count
1	41605	270893	312498
2	129672	458610	588282
3	142718	497944	640662
4	78140	465894	544034

```
sns.boxplot(data=df,x=df["workingday"],y=df["count"],color="purple")
```

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



1) HYPOTHESIS TESTING

- Ho : Working Day has no effect on bike rentals
- Ha: Working day has effect on bike rentals Since the data we work is numerical vs categorical, and how dependent each category(week day, weekend) on the rentals. appropriate test - ttest_independent significance level alpha - 0.05

```
from scipy.stats import ttest_ind
test1=df[df["workingday"]==1][["count"]]
```

```
test2=df[df["workingday"]==0][["count"]]
ttest,pvalue=ttest_ind(test1,test2,equal_var=False,alternative="greater")
alpha=0.05
print(alpha,ttest,pvalue)
if pvalue<alpha:
  print("Result : Reject null hypothesis, Working day has effect on bike rentals")
  print("Result : Accept alternate hypothesis, Working day has no effect on bike rentals")
else:
  print("Result : Accept null hypothesis, Working day has no effect on bike rentals")
```

```
0.05 [1.23625804] [0.10820156]
Result : Accept null hypothesis, Working day has no effect on bike rentals
```

2) HYPOTHESIS TESTING

- Ho : Weather has no effect on bike rentals
- Ha: Weather has effect on bike rentals Since the data we work is numerical vs >2 categorical, and how dependent rentals on each weather category appropriate test - ANOVA significance level alpha - 0. Since , it is anova test.the normality(shapiro,qqplot,kstest) and equal variances(levenes test) should be tested

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

weather	count
1	7192
2	2834
3	859
4	1

dtype: int64

```
df_new=df[~(df["weather"]==4)]
test_1=df_new[df_new["weather"]==1][["count"]]
test_2=df_new[df_new["weather"]==2][["count"]]
test_3=df_new[df_new["weather"]==3][["count"]]
```

NORMALITY TEST ,

we are taking shapiro test

```
from scipy.stats import shapiro
sstat,pvalue=shapiro(df_new["count"].sample(4999))
print(pvalue)
```

```
1.2586928300333067e-52
```

```
if pvalue<0.05:
  print("its gaussian(normal distribution)")
else:
  print("Its not gaussian(not normally distributed)")
```

```
its gaussian(normal distribution)
```

NORMALITY TEST

we are taking kstest

```
from scipy.stats import kstest
kstat,pvalue=kstest(test_1["count"],test_2["count"],test_3["count"])
print(pvalue)
```

```
2.0232588507344455e-07
```

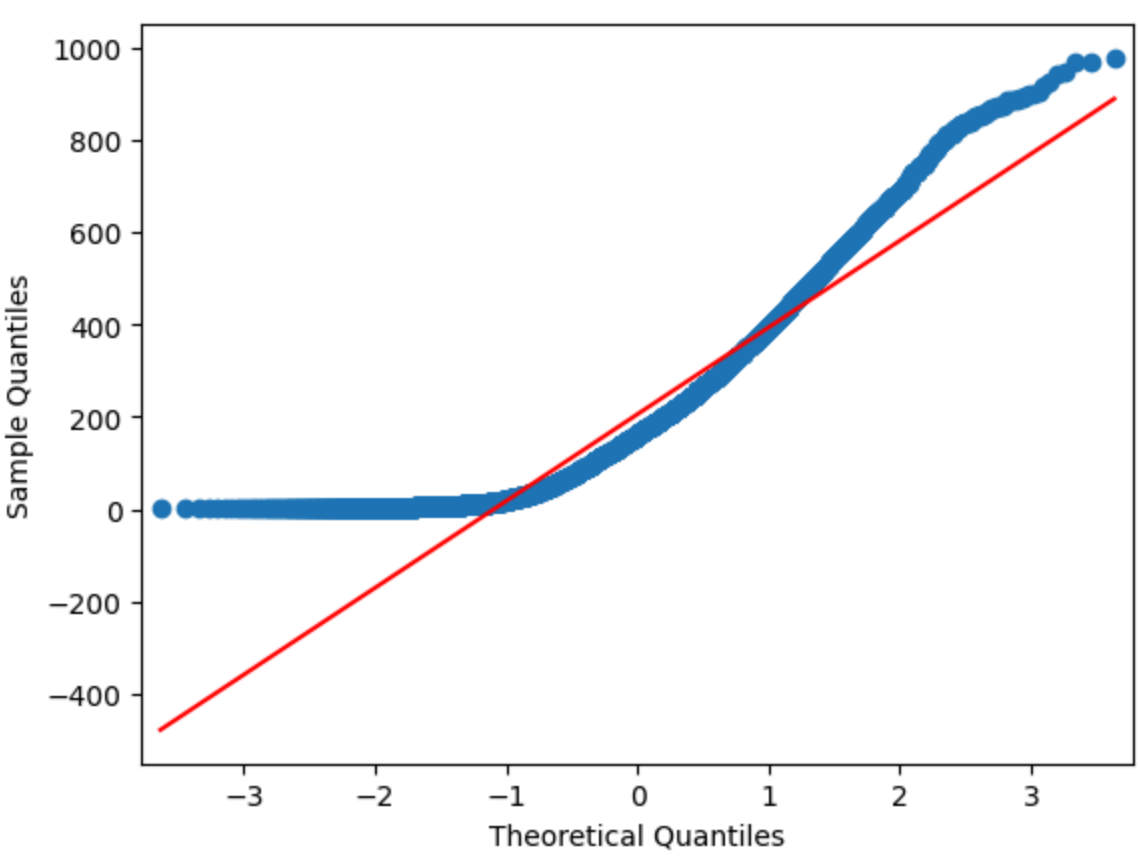
```
if pvalue<0.05:
  print("its gaussian(normal distribution)")
else:
  print("Its not gaussian(not normally distributed)")
```

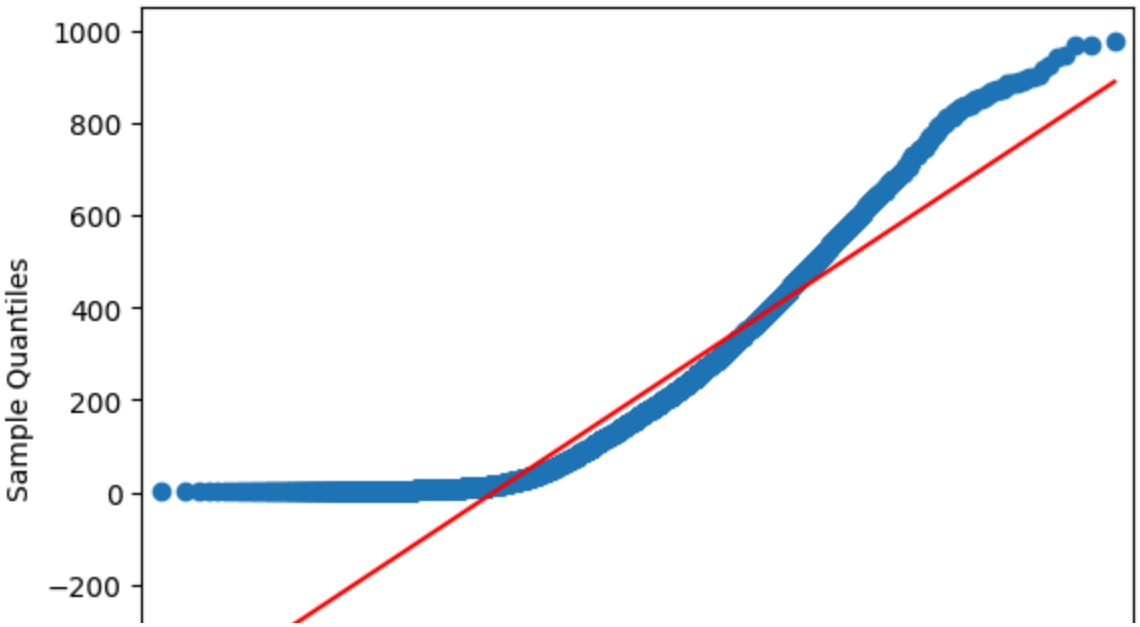
```
its gaussian(normal distribution)
```

NORMALITY TEST

- we will be using QQ Plot
- This output shows it is not gaussian distribution , when the percentile of sample is not

```
from statsmodels.graphics.gofplots import qqplot
qqplot(test_1["count"],line="s")
```






```
qqplot(test_2["count"],line="s")
```