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.

Define Problem Statement and perform Exploratory Data Analysis

The company wants to know:

- Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
- How well those variables describe the electric cycle demands

Let's import the data and dependencies for exploration

```
# importing dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

```
# importing file
df = pd.read_csv('yulu_bike_sharing.csv')
```

```
# analysing basic info about data
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

```
df.tail()
```

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

```
df.sample()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
4565	2011-11-03 07:00:00	4	0	1	2	13.12	16.665	81	6.0032	12	280	292

Total Number of rows and columns present in the data set

```
print(f'Number of rows:{df.shape[0]}  \nNumber of columns:{df.shape[1]}')
```

```
Number of rows:10886
Number of columns: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
```

Unique elements present in each columns

```
# number of unique values in each column
df.nunique()
```

```
datetime      10886
season            4
holiday           2
workingday        2
weather           4
temp             49
atemp            60
humidity         89
windspeed        28
casual          309
registered      731
count           822
dtype: int64
```

Datatype of following attributes needs to change to proper data type

- datetime - to datetime
- season - to categorical
- holiday - to categorical
- workingday - to categorical
- weather - to categorical

```
# convert date time data type
df['datetime'] = pd.to_datetime(df['datetime'])
# convert category columns data type from int to object
cat_cols = ['season', 'holiday', 'workingday', 'weather']
for col in cat_cols:
    df[col] = df[col].astype('object')
```

```
df.info() # after converted few data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  object
 2   holiday     10886 non-null  object
 3   workingday  10886 non-null  object
 4   weather     10886 non-null  object
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(4), object(4)
memory usage: 1020.7+ KB
```

Statistical Summary of the entire data set

```
df.describe(include = 'all').T
```

	count	unique	top	freq	mean	min	25%	50%	75%	max	std
datetime	10886	NaN	NaN	NaN	2011-12-27 05:56:22.399411968	2011-01-01 00:00:00	2011-07-02 07:15:00	2012-01-01 20:30:00	2012-07-01 12:45:00	2012-12-19 23:00:00	NaN
season	10886.0	4.0	4.0	2734.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
holiday	10886.0	2.0	0.0	10575.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
workingday	10886.0	2.0	1.0	7412.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
weather	10886.0	4.0	1.0	7192.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
temp	10886.0	NaN	NaN	NaN	20.23086	0.82	13.94	20.5	26.24	41.0	7.79159
atemp	10886.0	NaN	NaN	NaN	23.655084	0.76	16.665	24.24	31.06	45.455	8.474601
humidity	10886.0	NaN	NaN	NaN	61.88646	0.0	47.0	62.0	77.0	100.0	19.245033
windspeed	10886.0	NaN	NaN	NaN	12.799395	0.0	7.0015	12.998	16.9979	56.9969	8.164537
casual	10886.0	NaN	NaN	NaN	36.021955	0.0	4.0	17.0	49.0	367.0	49.960477
registered	10886.0	NaN	NaN	NaN	155.552177	0.0	36.0	118.0	222.0	886.0	151.039033
count	10886.0	NaN	NaN	NaN	191.574132	1.0	42.0	145.0	284.0	977.0	181.144454

- There are no missing values in the dataset.
- casual and registered attributes might have outliers because their mean and median are
  very far away from one another and the value of standard deviation is also high which tells us that there is high variance in the data of these attributes.

Let’s find out is there any missing values or duplicate records in the data set

```
# detecting missing values
df.isnull().sum() # no missing values in the df
```

```
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
```

```
# detecting duplicate values
df.duplicated().sum() # no duplicate records in the df
```

```
0
```

- There are no missing values and no duplicate records in the data set

Let’s understand the minimum and maximum dates of bike rented

```
# minimum date and maximum date
print('Minimum Date:', df['datetime'].min(), '\nMaximum Date:', df['datetime'].max())
```

```
Minimum Date: 2011-01-01 00:00:00
Maximum Date: 2012-12-19 23:00:00
```

- The entire data set has nearly 2 years of records

Let’s understand Categorical columns and unique records of each column

```
# number of unique values in each catagorical columns
df[cat_cols].melt().groupby(['variable','value'])[['value']].count()
```

		value
variable	value	
holiday	0	10575
	1	311
season	1	2686
	2	2733
	3	2733
	4	2734
weather	1	7192
	2	2834
	3	859
	4	1
workingday	0	3474
	1	7412

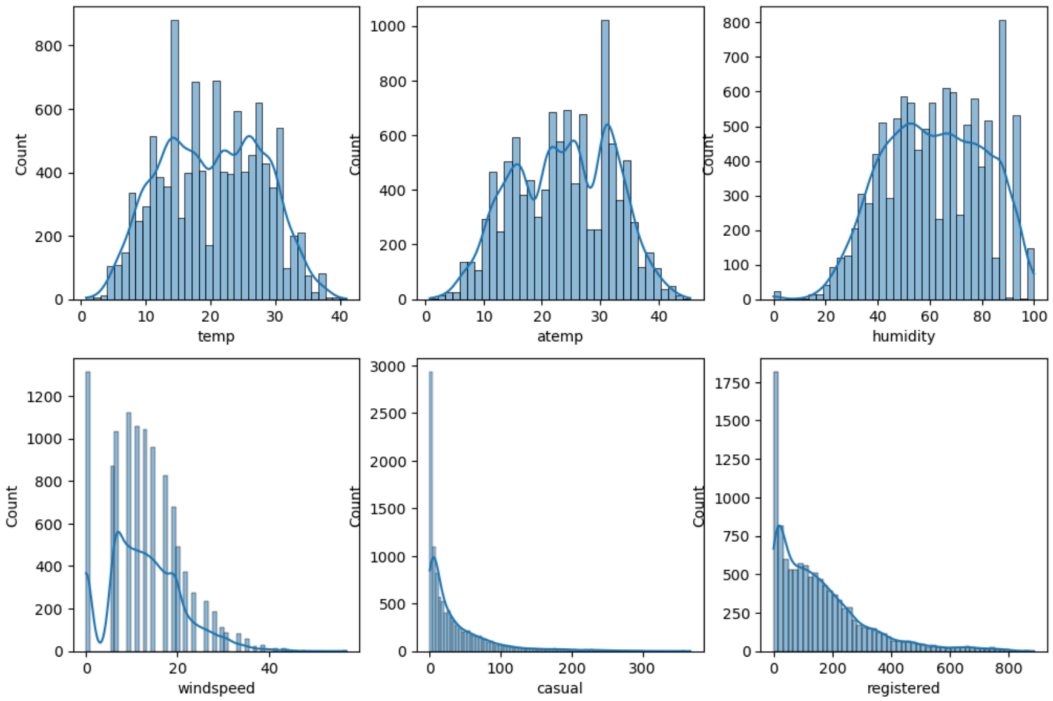
Univariate & Bi-variate analysis

((Try establishing a relation between the dependent and independent variable (Dependent "Count" & Independent: Workingday, Weather, Season etc))

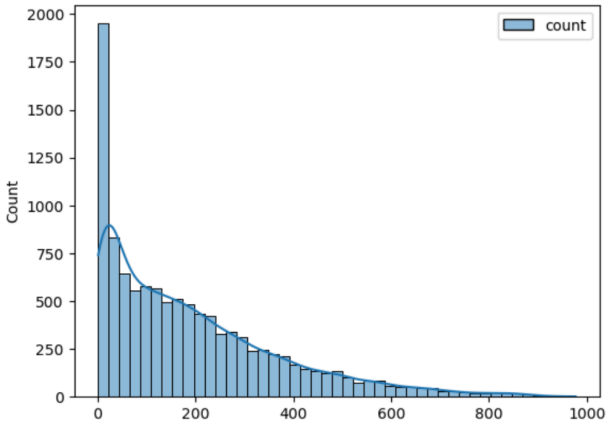
Let's understand the numerical columns and their distributions of data using countplot

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

```
# histogram plot to analyse the distribution of numerical columns

fig, axis = plt.subplots(nrows = 2, ncols = 3, figsize=(12,8))
index = 0
for row in range(2):
    for col in range(3):
        sns.histplot(df[num_cols[index]] , ax = axis[row,col], kde = True)
        index = index + 1
plt.show()
```



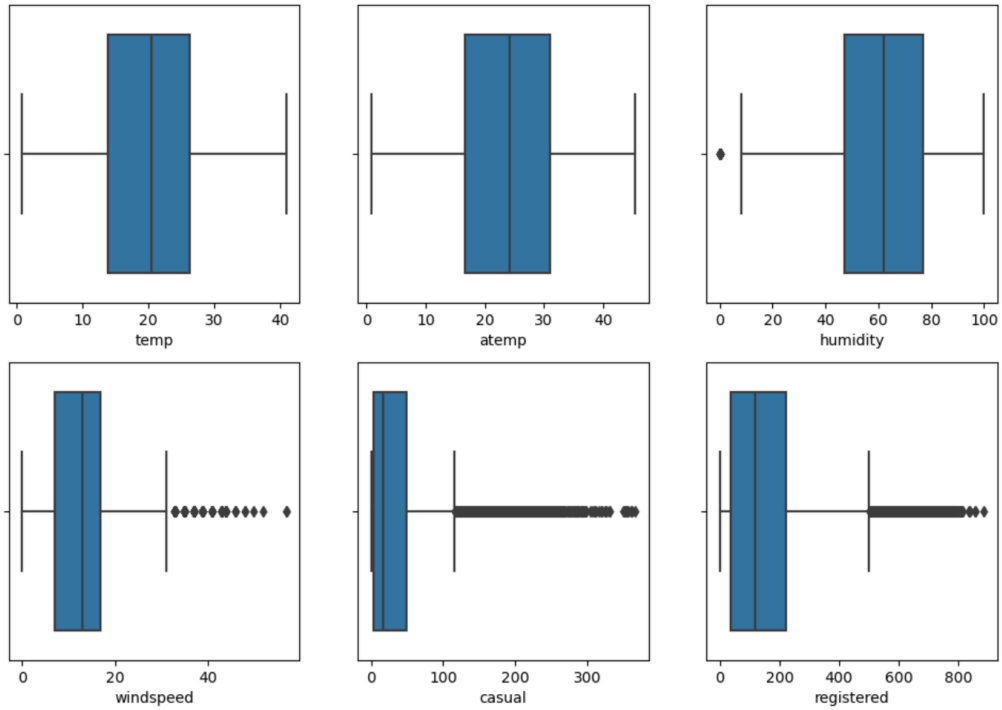

```
sns.histplot([df['count']], kde = True)
plt.show()
```



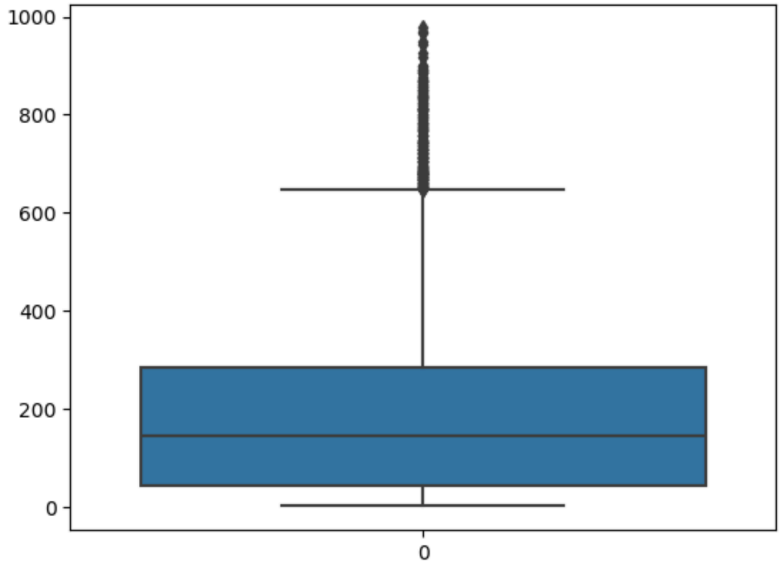
- temp, atemp and humidity looks like they follows the Normal Distribution
- casual, registered and count somewhat looks like Log Normal Distribution
- windspeed follows the binomial distribution

Let's detect is there any outliers in our data set using boxplot

```
# let's use boxplot to detect outliers present in numerical columns

fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12,8))
index = 0
for row in range(2):
    for col in range(3):
        sns.boxplot(x= df[num_cols[index]], ax=axis[row,col])
        index = index + 1
plt.show()
```



```
sns.boxplot([df['count']])
plt.show()
```



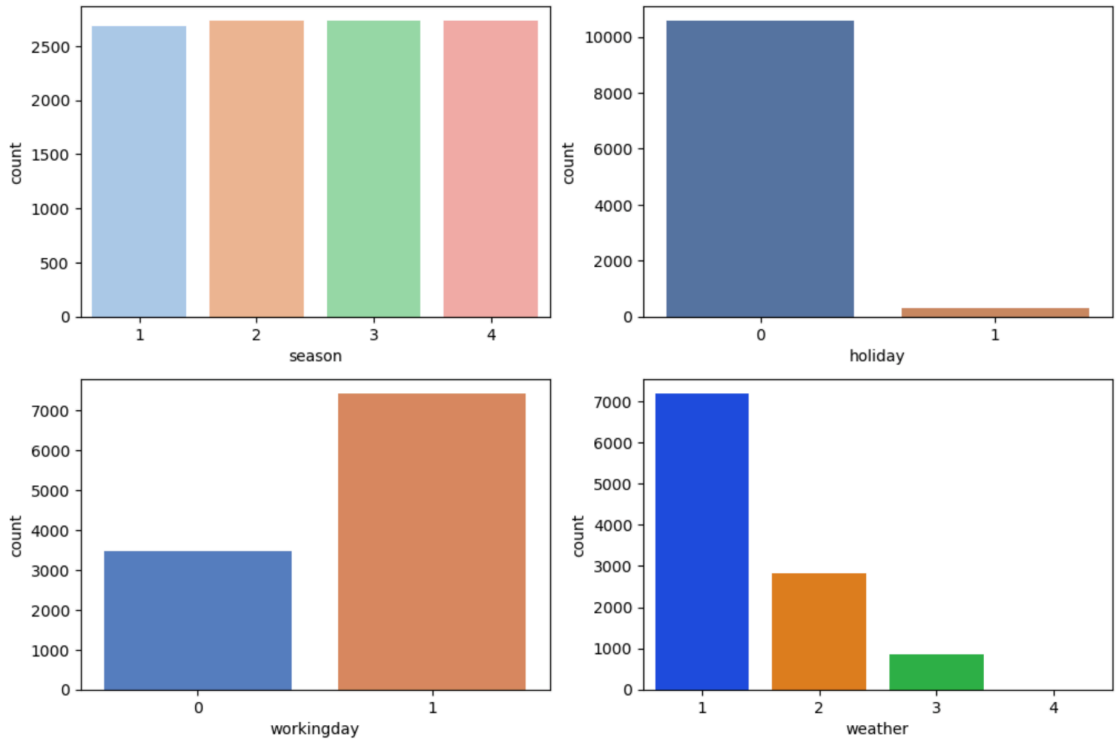
- Looks like humidity, casual, registered and count have outliers in the data.

Let's use countplot for each categorical column

```
cat_cols
```

```
['season', 'holiday', 'workingday', 'weather']
```

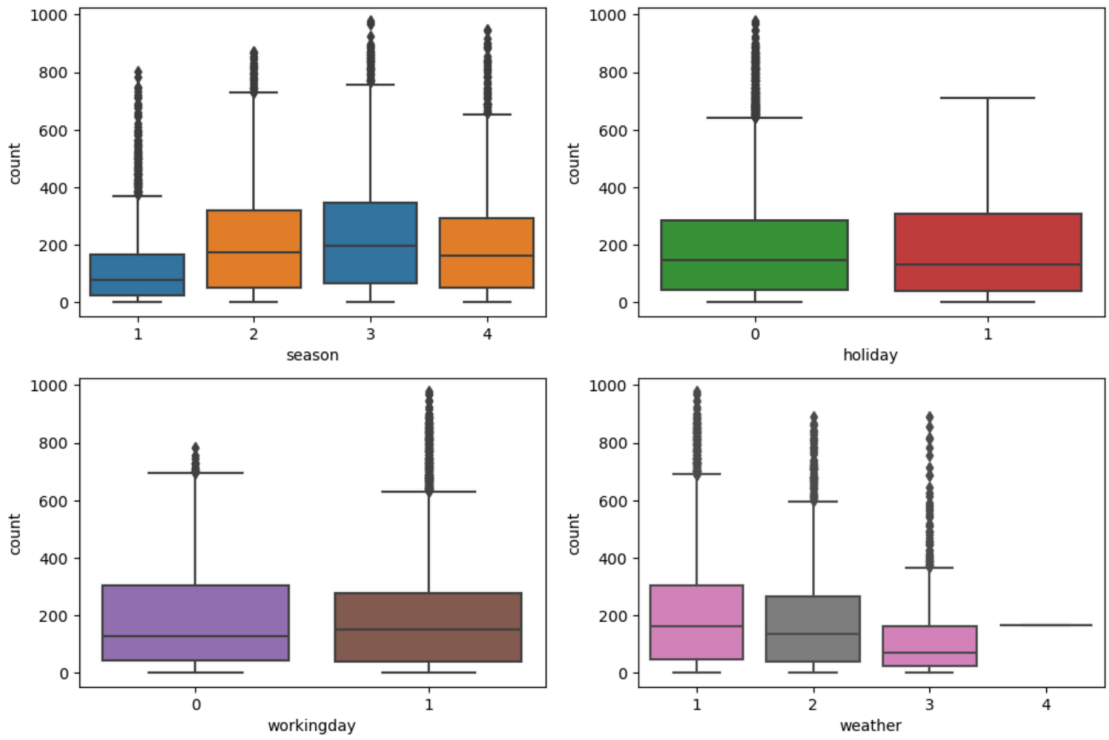
```
# countplot for each categorical column
fig, axis = plt.subplots(nrows = 2, ncols =2, figsize=(12,8))
index = 0
palette_list = ['pastel', 'deep', 'muted', 'bright']
for row in range(2):
    for col in range(2):
        sns.countplot(x=cat_cols[index], ax=axis[row,col], data=df, palette=palette_list[index])
        index = index + 1
plt.show()
```



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

Let's use boxplot for each categorical column

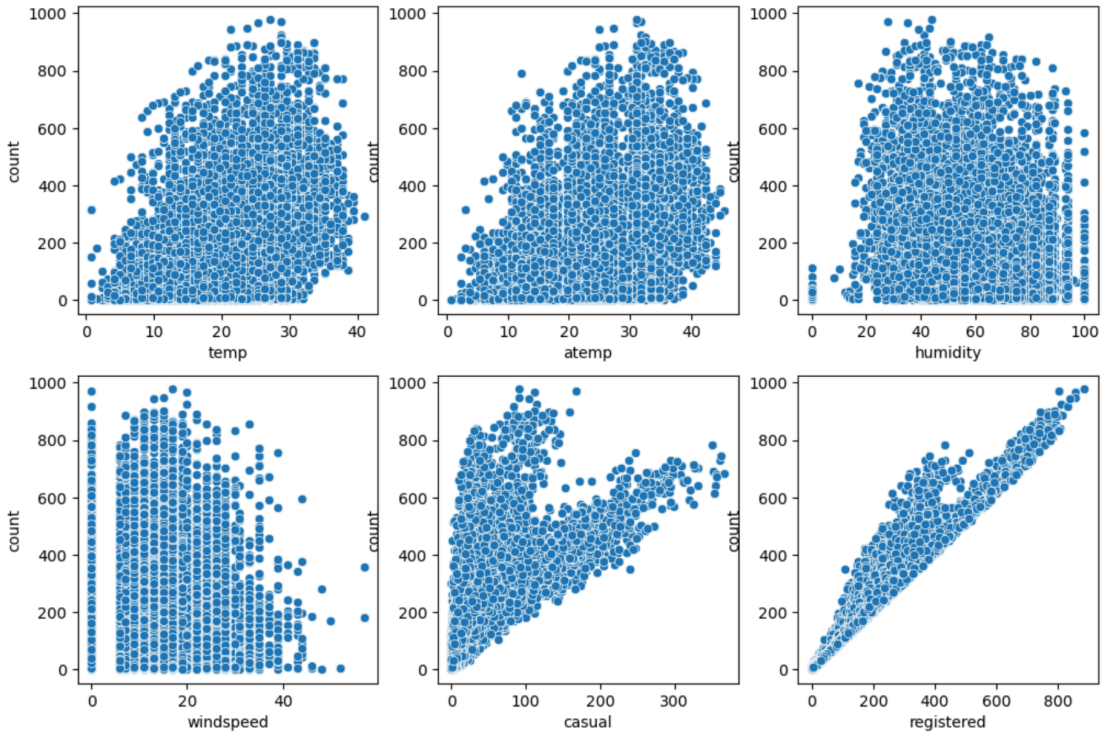
```
# plotting categorical columns against count
fig, axis = plt.subplots(nrows = 2, ncols = 2, figsize=(12,8))
index = 0
palette_list = [['#1f77b4', '#ff7f0e'], ['#2ca02c', '#d62728'], ['#9467bd', '#8c564b'], ['#e377c2', '#7f7f7f'
for row in range(2):
    for col in range(2):
        sns.boxplot(x=cat_cols[index], y='count', ax=axis[row,col], data=df, palette=palette_list[index])
        index = index + 1
plt.show()
```



- In summer and fall seasons more bikes are rented as compared to other seasons.
- Whenever it's a holiday more bikes are rented.
- It is also clear from the working day also that whenever day is holiday or weekend, slightly
  more bikes were rented.
- Whenever there is rain, thunderstorm, snow or fog, fewer bikes are rented.

Let’s use scatterplot for each categorical column

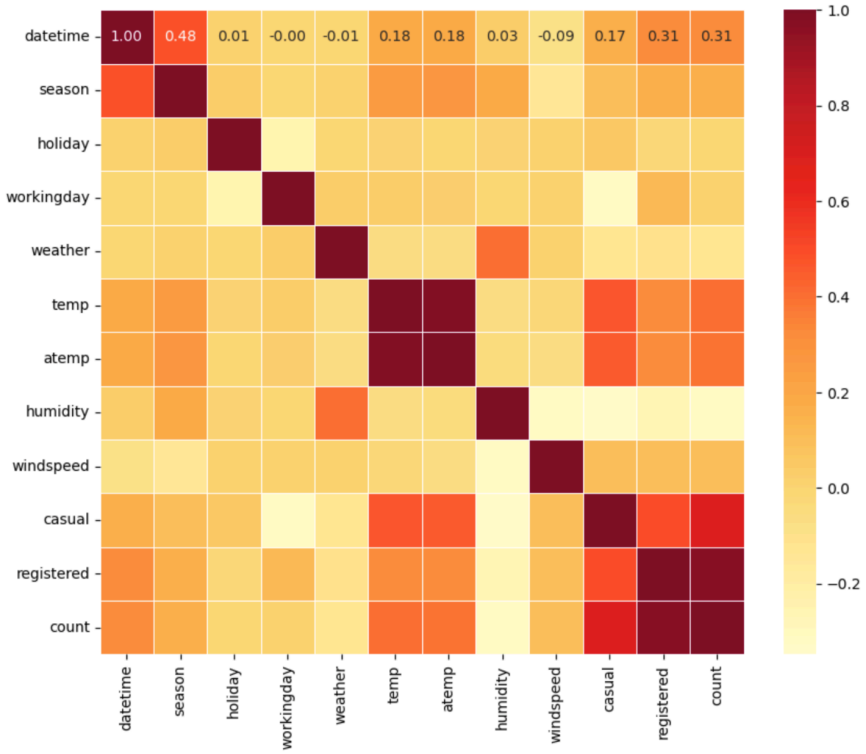
```
# plotting numerical variables against count using scatterplot
fig, axis = plt.subplots(nrows=2, ncols=3, figsize=(12,8))
index = 0
for row in range(2):
    for col in range(3):
        sns.scatterplot(data=df, x=num_cols[index], y='count', ax=axis[row,col])
        index = index + 1
plt.show()
```



- Whenever the humidity is less than 20, the number of bikes rented is very very low.
- Whenever the temperature is less than 10, the number of bikes rented is less.
- Whenever the windspeed is greater than 35, number of bikes rented is less.

Let's understand the correlation between count and numerical variables

```
# understanding the correlation between count and numerical variables
df.corr()['count']
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="YlOrRd", square=True, cbar=True, linewidths=0.5)
plt.show()
```



Hypothesis Testing

2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented

```
# 2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented
```

```
data_group1 = df[df['workingday']==0]['count'].values
data_group2 = df[df['workingday']==1]['count'].values

print(np.var(data_group1), np.var(data_group2))
np.var(data_group2)// np.var(data_group1)
```

```
30171.346098942427 34040.69710674686
1.0
```

```
t_stat, p_value = stats.ttest_ind(a=data_group1, b=data_group2, equal_var=True)
```

```
# Display Results
print(f"T-Statistic: {t_stat:.4f}")
print(f"P-Value: {p_value:.4f}")

# Interpretation
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis: Working Day does not affect the number of electric cycles rented.")
else:
    print("Fail to reject the null hypothesis: Working Day does affect the number of electric cycles rented.")
```

```
T-Statistic: -1.2096
P-Value: 0.2264
Fail to reject the null hypothesis: Working Day does affect the number of electric cycles rented.
```

ANNOVA to check if No. of cycles rented is similar or different in different (1. weather 2.Season)

```
# ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season
```

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

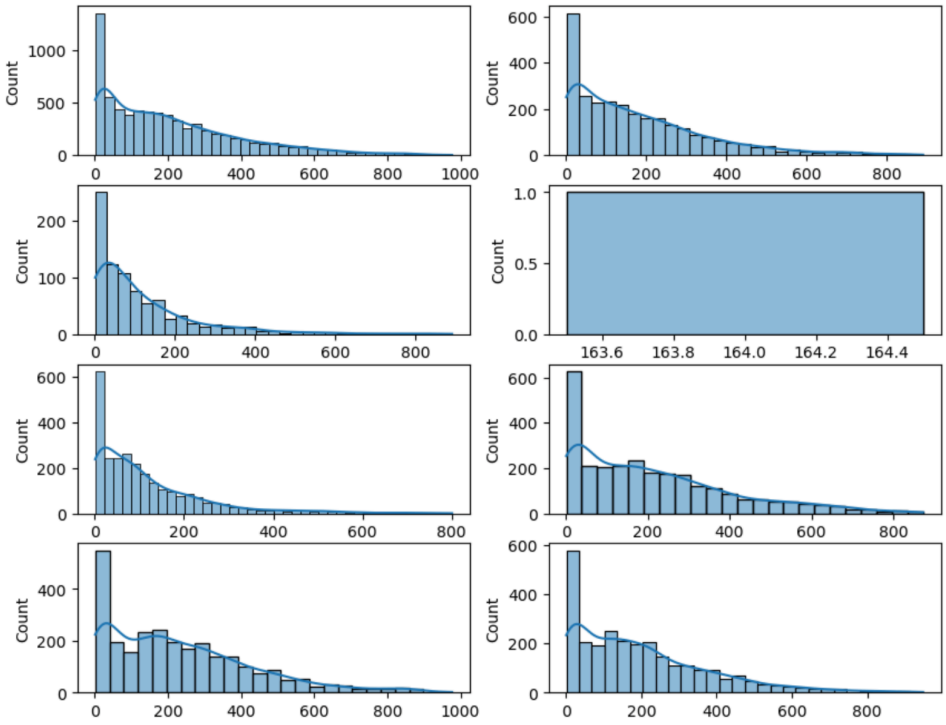
```
array([1, 2, 3, 4], dtype=object)
```

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

```
array([1, 2, 3, 4], dtype=object)
```

```
# seperating weather and season groups with its unique values for analysis
W1 = df[df['weather']==1]['count'].values
W2 = df[df['weather']==2]['count'].values
W3 = df[df['weather']==3]['count'].values
W4 = df[df['weather']==4]['count'].values
S1 = df[df['season']==1]['count'].values
S2 = df[df['season']==2]['count'].values
S3 = df[df['season']==3]['count'].values
S4 = df[df['season']==4]['count'].values
groups = [W1,W2,W3,W4,S1,S2,S3,S4]
```

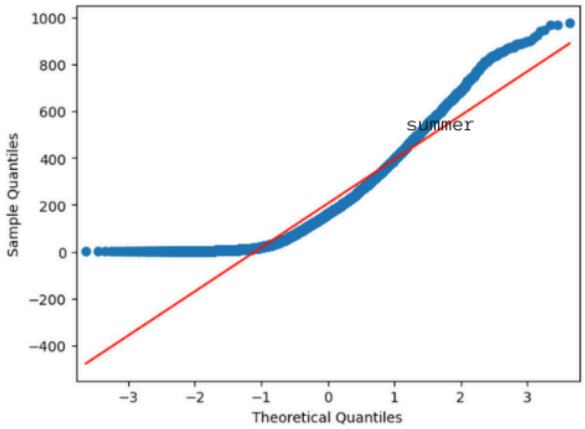
```
fig, axis = plt.subplots(nrows=4 , ncols=2 , figsize=(10,8))
index = 0
for row in range(4):
    for col in range(2):
        sns.histplot(groups[index], ax=axis[row,col], kde=True)
        index = index + 1
plt.show()
```

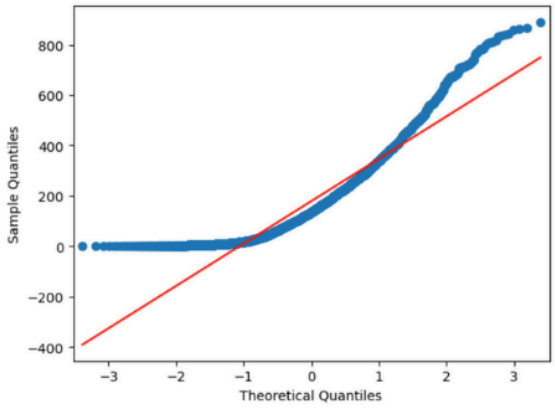


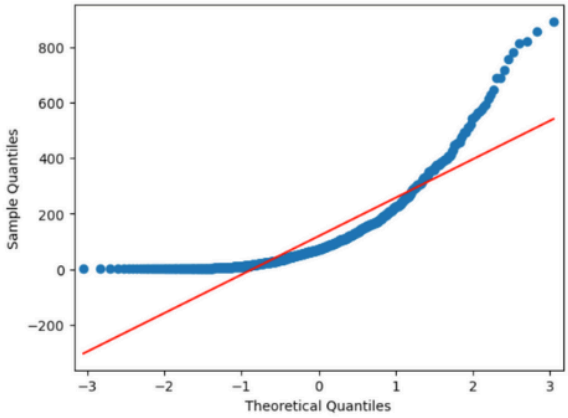
Let’s use qq-plot

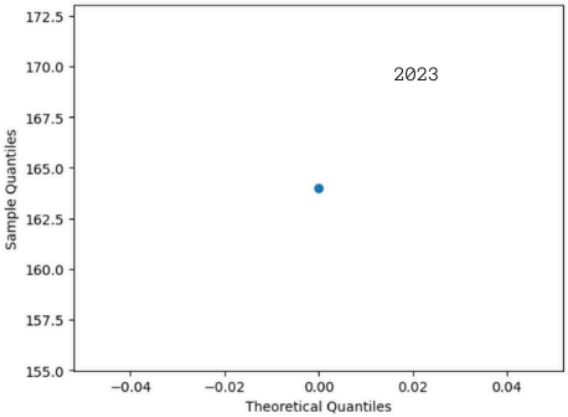
```
from statsmodels.graphics.gofplots import qqplot
```

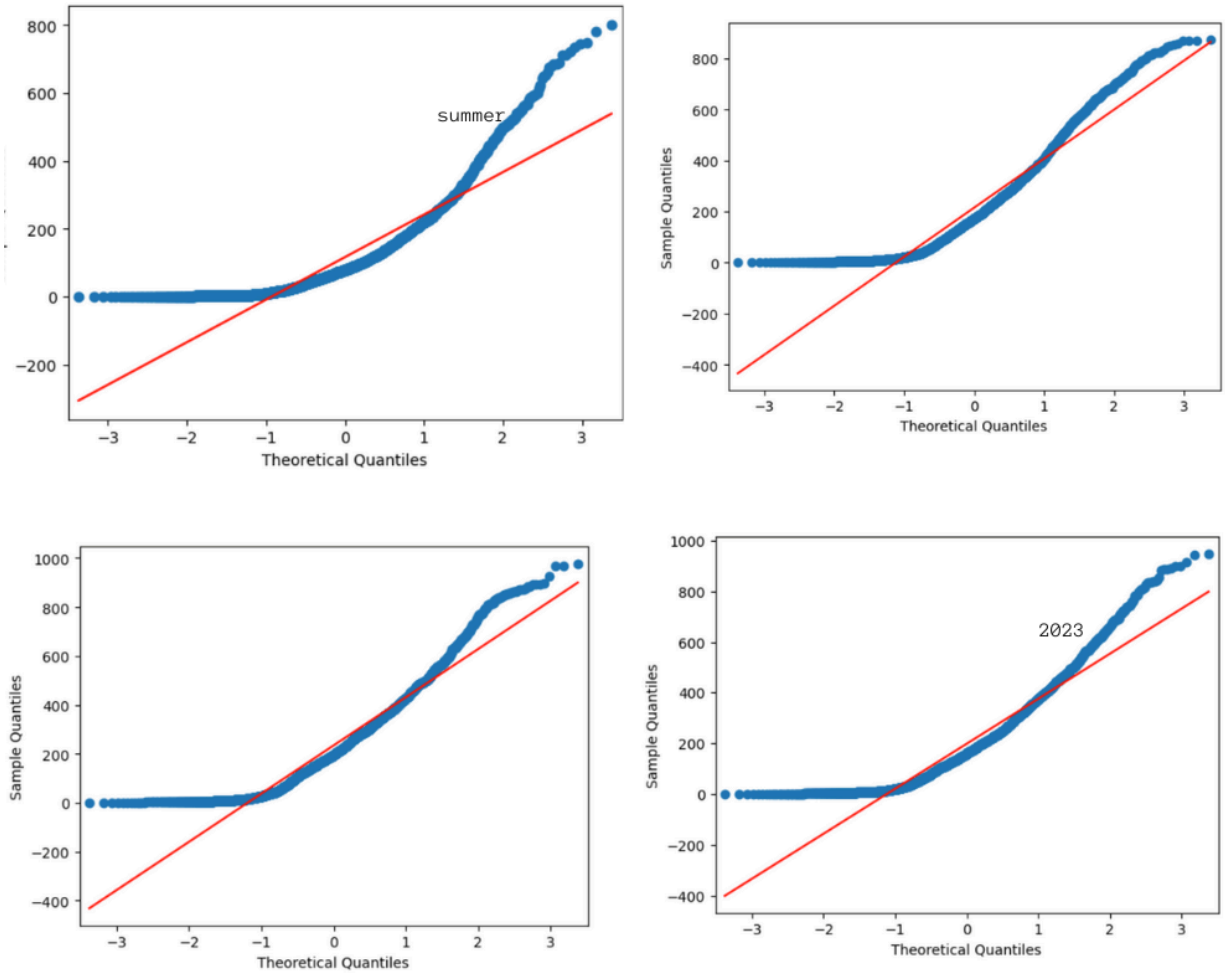
```
index = 0
for row in range(4):
    for col in range(2):
        qqplot(groups[index], line="s")
        index += 1

plt.show()
```











- As per above graphs, all groups are not following Gaussian distribution

Chi-square test to check if Weather is dependent on the season

```
val = stats.chi2_contingency(data_table)
print(val)
expected_values = val[3]
print(expected_values)
nrows, ncols = 4, 4
dof = (nrows-1)*(ncols-1)
print("degrees of freedom: ", dof)
alpha = 0.05


chi_sqr = sum([(o-e)**2/e for o, e in zip(data_table.values, expected_values)])
chi_sqr_statistic = chi_sqr[0] + chi_sqr[1]
print("chi-square test statistic: ", chi_sqr_statistic)

critical_val = stats.chi2.ppf(q=1-alpha, df=dof)
print(f"critical value: {critical_val}")

p_val = 1-stats.chi2.cdf(x=chi_sqr_statistic, df=dof)
print(f"p-value: {p_val}")

if p_val <= alpha:
    print("\nSince p-value is less than the alpha 0.05, We reject the Null Hypothesis. Meaning that\
    Weather is dependent on the season.")
else:
    print("Since p-value is greater than the alpha 0.05, We do not reject the Null Hypothesis")
```

```
Chi2ContingencyResult(statistic=49.158655596893624, pvalue=1.549925073686492e-07, dof=9, expected_freq=arra
y([[1.77454639e+03, 6.99258130e+02, 2.11948742e+02, 2.46738931e-01],
       [1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-01],
       [1.80559765e+03, 7.11493845e+02, 2.15657450e+02, 2.51056403e-01],
       [1.80625831e+03, 7.11754180e+02, 2.15736359e+02, 2.51148264e-01]]))
[[1.77454639e+03 6.99258130e+02 2.11948742e+02 2.46738931e-01]
 [1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
 [1.80559765e+03 7.11493845e+02 2.15657450e+02 2.51056403e-01]
 [1.80625831e+03 7.11754180e+02 2.15736359e+02 2.51148264e-01]]
degrees of freedom:  9
chi-square test statistic:  44.09441248632364
critical value: 16.918977604620448
p-value: 1.3560001579371317e-06

Since p-value is less than the alpha 0.05, We reject the Null Hypothesis. Meaning that    Weather is depende
nt on the season.
```

```
#assumptions of ANOVA don't hold, we need Kruskal Wallis
kruskal_stat, p_value = stats.kruskal(W1,W2,W3,W4,S1,S2,S3,S4)
print("p_value===",p_value)
if p_value<0.05:
  print("Since p-value is less than 0.05, we reject the null hypothesis")
```

```
p_value=== 4.614440933900297e-191
Since p-value is less than 0.05, we reject the null hypothesis
```

```
plt.show()

#Null Hypothesis: Variances are similar in different weather and season
#Alternate Hypothesis: Variances are not similar in different weather and season.

#Significance level (alpha): 0.05
levene_stat, p_value = stats.levene(W1,W2,W3,W4,S1,S2,S3,S4)
print(p_value)
if p_value < 0.05:
    print("Reject the Null hypothesis : Variances are not equal")
else:
  print("Fail to Reject the Null hypothesis : Variances are  equal")
```

Insights

- In summer and fall seasons more bikes are rented as compared to other seasons.
- Whenever it's 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, fewer bikes are rented.
- Whenever the humidity is less than 20, the number of bikes rented is very very low.
- Whenever the temperature is less than 10, the number of bikes rented is less.
- Whenever the windspeed is greater than 35, the number of bikes rented is less.

Recommendations

- In summer and fall seasons the company should have more bikes in stock to be rented. Because the demand in these seasons is higher as compared to other seasons.
- With a significance level of 0.05, workingday has no effect on the number of bikes being rented.
- In very low humid days, companies should have less bikes in the stock to be rented.
- Whenever the temperature is less than 10 or on very cold days, companies should have fewer bikes.
- Whenever the windspeed is greater than 35 or in thunderstorms, companies should have less bikes in stock to be rented.