In [51]:

```
import pandas as pd
df = pd.read_csv('C:/Users/maheshmangaonkar/Desktop/Yulu.csv')
```

In [52]:

```
df
```

Out[52]:

	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 × 12 columns

In [53]:

```
#The data consists of 10886 rows × 12 columns.
```

In [54]:

```
df.shape
```

Out[54]:

```
(10886, 12)
```

In [55]:

```
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
```

In [56]:

```
df.describe()
```

Out[56]:

	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	1088
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	19
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	18
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	4
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	14
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	28
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	97

In [57]:

```
#The data describes the following insights:
    1.
```

```
  Input In [57]
    1.
    ^
IndentationError: unexpected indent
```

In [58]:

```
df.nunique()
```

Out[58]:

```
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
```

In [59]:

```
#missing_values
df.isna().sum()
```

Out[59]:

```
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
```

In [ ]:

```
# No missing data in df
```

In [60]:

```
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
```

In [ ]:

```
#Converting season ,holiday ,workingday, weather into category
```

In [61]:

```
df["season"] = df["season"].astype("category")
df["holiday"] = df["holiday"].astype("category")
df["workingday"] = df["workingday"].astype("category")
df["weather"] = df["weather"].astype("category")
```

In [62]:

```
df
```

Out[62]:

	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 × 12 columns

In [7]:

```
# Univariate Analysis
```

In [63]:

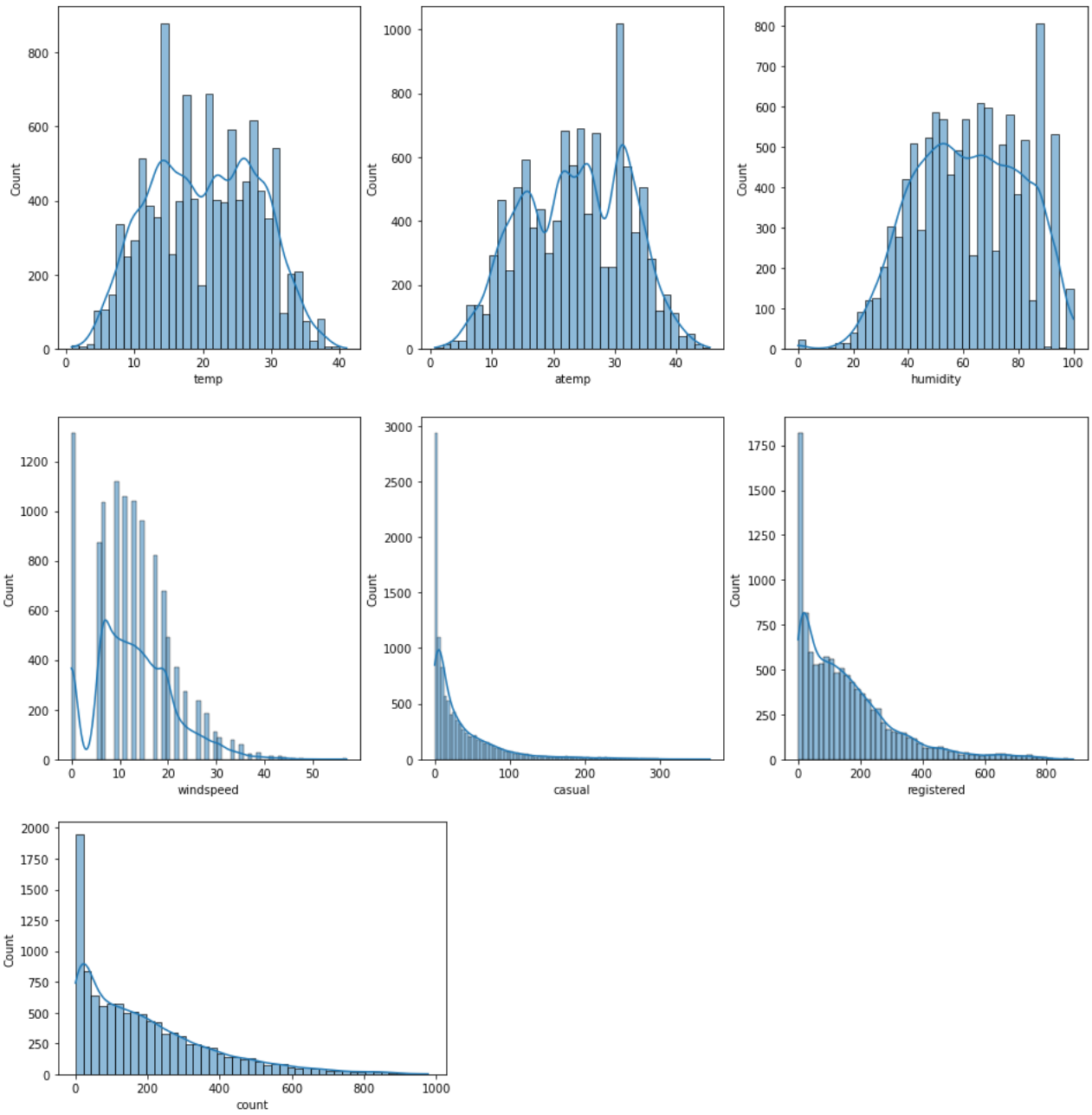
```
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered','count']
```

In [9]:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt



fig, axis = plt.subplots(nrows=2 , ncols=3, figsize = (16,12))
index = 0
for row in range (2):
    for columns in range (3):
        sns.histplot(df[numerical_cols[index]], ax = axis[row,columns], kde = True)
        index+=1
plt.show()
sns.histplot(data = df, x = 'count',kde = True)
plt.show()
```



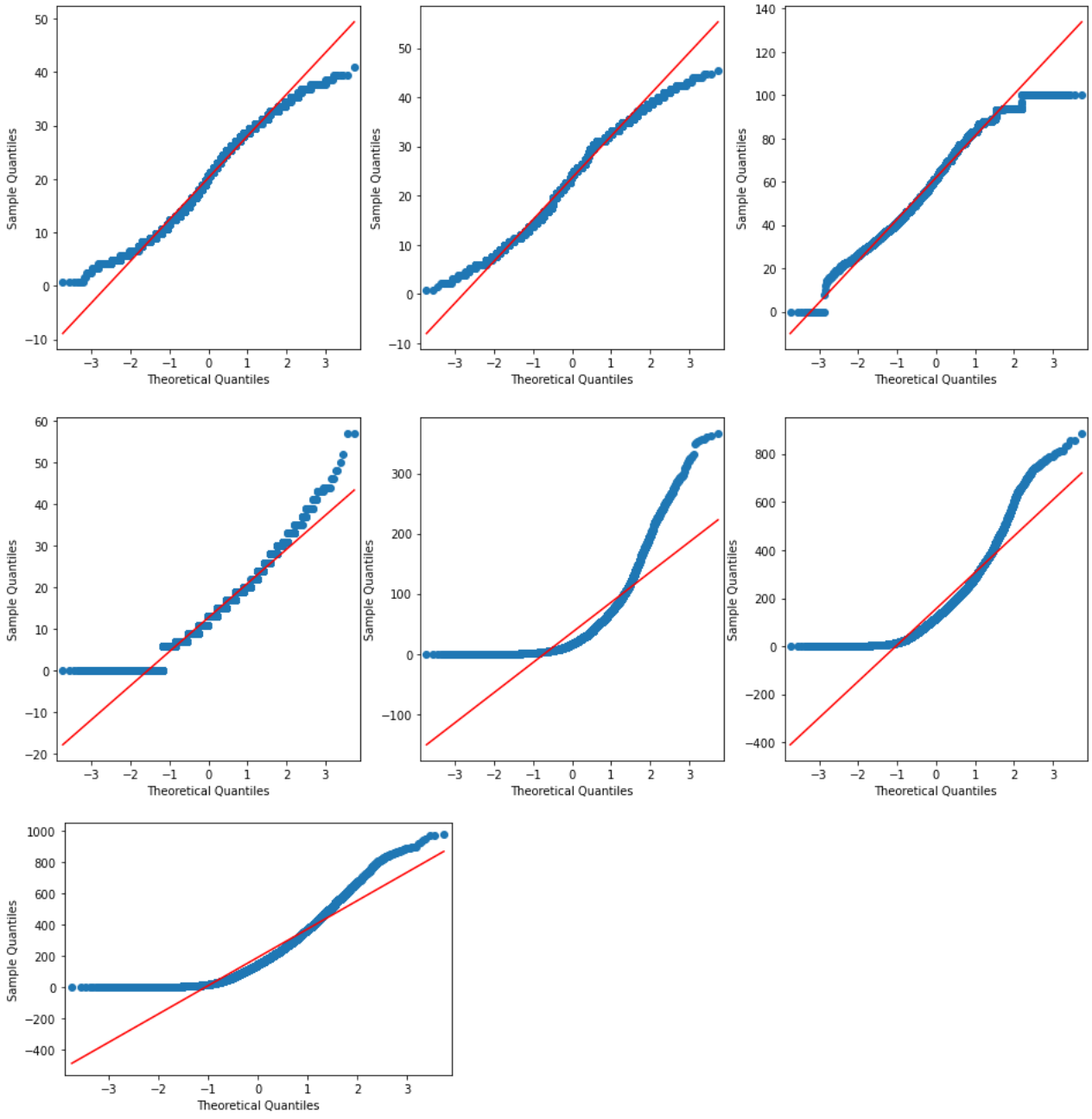
In [10]:

```
#1.casual, registered and count  is like Log Normal Distribution
#2.temp, atemp and humidity follows the Normal Distribution
#3.windspeed follows the binomial distribution
```

In [64]:

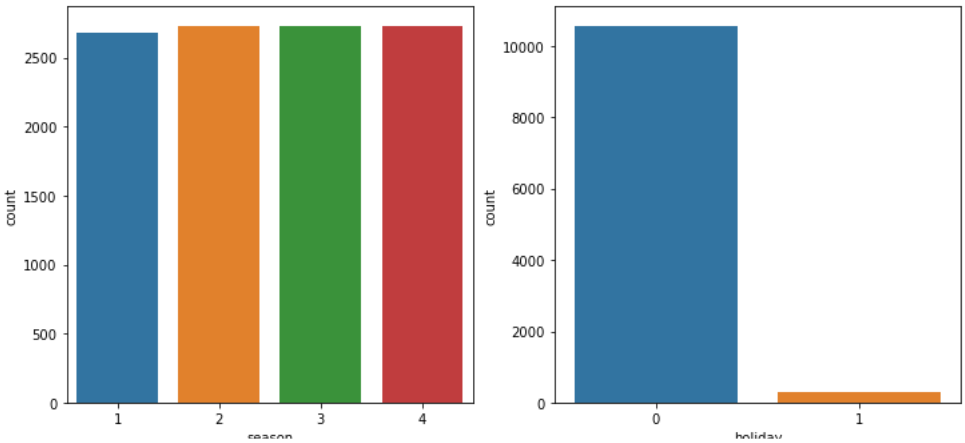
```
import statsmodels.api as sm
import pylab as py
import warnings
warnings.filterwarnings('ignore')

fig, axis = plt.subplots(nrows=2 , ncols=3, figsize = (16,12))
index = 0
for row in range (2):
    for columns in range (3):
        sm.qqplot(df[numerical_cols[index]], ax = axis[row,columns], line = 's')
        index+=1
py.show()
sm.qqplot(df[numerical_cols[-1]],line="s")
py.show()
```



In [13]:

```
# Categorical variable analysis
categorical= ['season', 'holiday', 'workingday', 'weather']
fig, axis = plt.subplots(nrows=2 , ncols=2, figsize = (12,12))
index = 0
for row in range (2):
    for columns in range (2):
        sns.countplot(df[categorical[index]], ax = axis[row,columns])
        index+=1
plt.show()
```



In [14]:

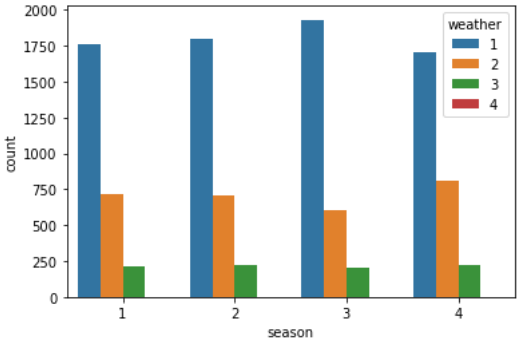
```
#Bivariate analysis
```

In [65]:

```
sns.countplot(df['season'],hue=df['weather'],data=df)
```

Out[65]:

```
<AxesSubplot:xlabel='season', ylabel='count'>
```

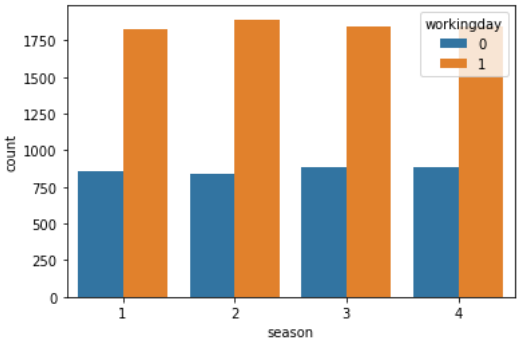


In [66]:

```
sns.countplot(df['season'],hue=df['workingday'],data=df)
```

Out[66]:

```
<AxesSubplot:xlabel='season', ylabel='count'>
```



In [17]:

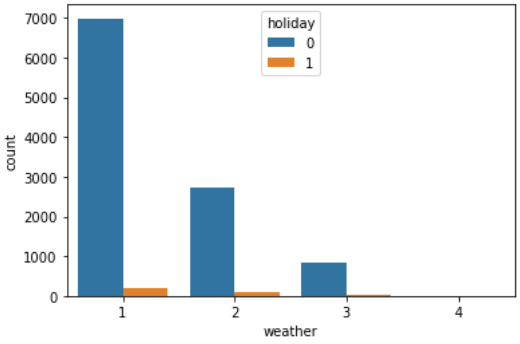
```
#In all 4 seasons during workingday only the count is high
```

In [67]:

```
sns.countplot(df['weather'],hue=df['holiday'],data=df)
```

Out[67]:

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```



In [19]:

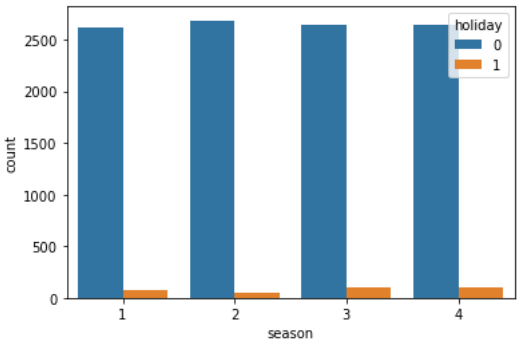
```
#count is high when there is no holiday
```

In [68]:

```
sns.countplot(df['season'],hue=df['holiday'],data=df)
```

Out[68]:

```
<AxesSubplot:xlabel='season', ylabel='count'>
```



In [21]:

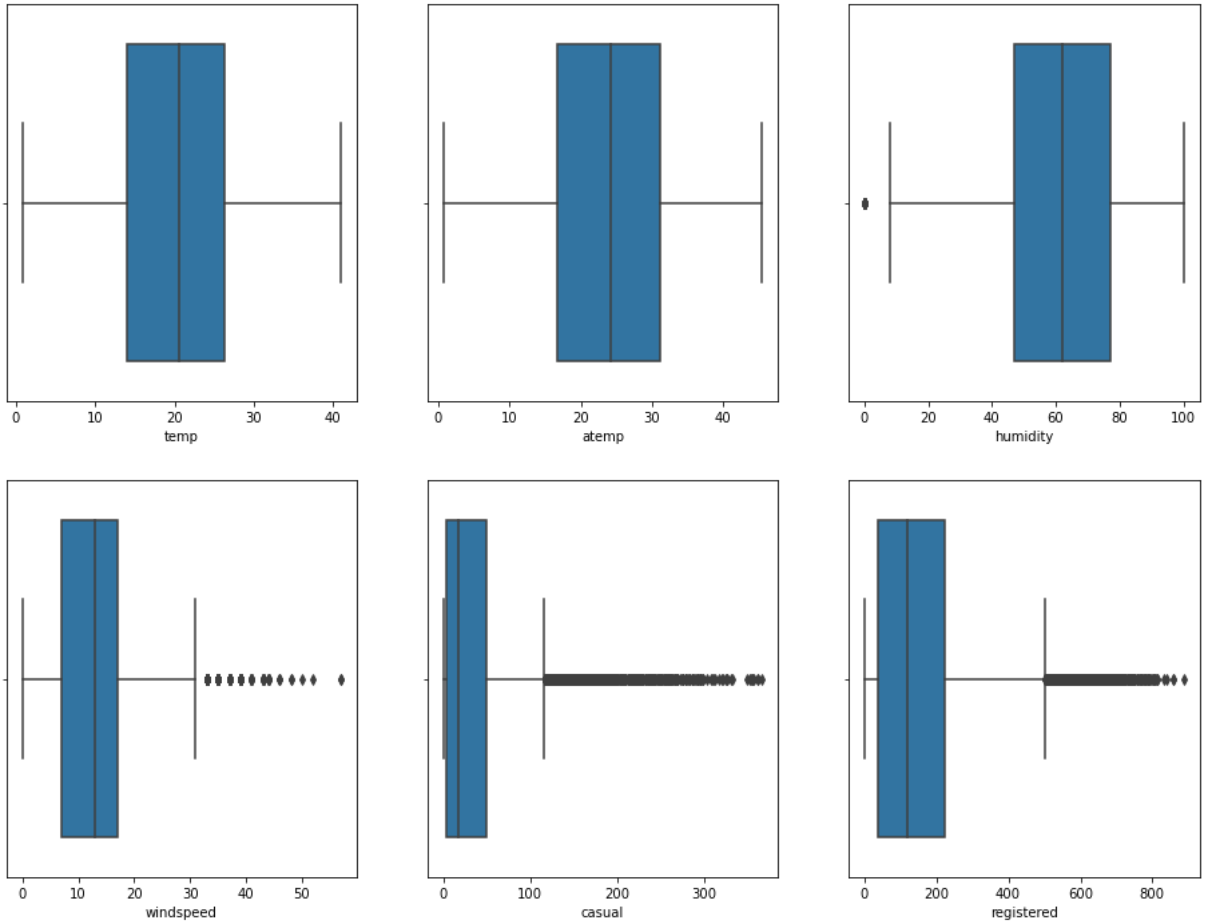
```
#Boxplot
```

In [69]:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt



fig, axis = plt.subplots(nrows=2 , ncols=3, figsize = (16,12))
index = 0
for row in range (2):
    for columns in range (3):
        sns.boxplot(df[numerical_cols[index]], ax = axis[row,columns])
        index+=1
plt.show()
```



In [23]:

```
#Outliers are present in windspeed, casual, registered
```

In [24]:

```
#2 sample t test
```

In [25]:

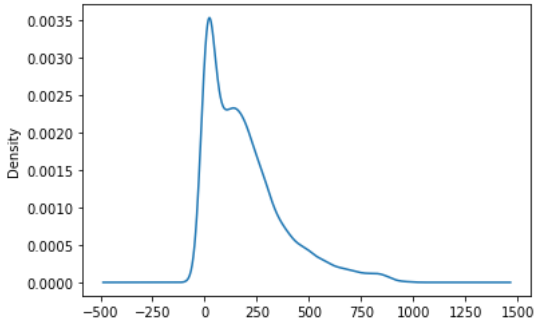
```
#significant level(alpha) as 0.05 for all test
```

In [26]:

```
df.loc[df[('workingday')] == 1]['count'].plot(kind ='kde')
```

Out[26]:

```
<AxesSubplot:ylabel='Density'>
```



In [77]:

```
import numpy as np
import pandas as pd
df1 = df.loc[df[('workingday')] == 1]['count'].reset_index
```

In [78]:

```
df1
```

Out[78]:

```
<bound method Series.reset_index of 47         5
48         2
49         1
50         3
51        30
        ...
10881    336
10882    241
10883    168
10884    129
10885     88
Name: count, Length: 7412, dtype: int64>
```

In [80]:

```
df1.drop(index ='workingday' , axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [80], in <cell line: 1>()
----> 1 df1.drop(index ='workingday' , axis=1, inplace=True)

AttributeError: 'function' object has no attribute 'drop'
```

In [36]:

```
df2= df.loc[df[('workingday')] == 0]['count'].reset_index
```

In [81]:

```
df2
```

Out[81]:

```
<bound method Series.reset_index of 0         16
1         40
2         32
3         13
4          1
        ...
10809    109
10810    122
10811    106
10812     89
10813     33
Name: count, Length: 3474, dtype: int64>
```

In [82]:

```
df2.drop(['index'],axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [82], in <cell line: 1>()
----> 1 df2.drop(['index'],axis=1,inplace=True)

AttributeError: 'function' object has no attribute 'drop'
```

In [42]:

```

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [42], in <cell line: 1>()
----> 1 ttest,p_value = ttest_ind(df1,df2)

NameError: name 'ttest_ind' is not defined
```

In [83]:

```
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
import scipy.stats as stats
from scipy.stats import ttest_ind, ttest_1samp, ttest_rel ,chi2_contingency,f_oneway,chisquare,levene,shapiro,boxcox
%matplotlib inline
import os

ttest,p_value = ttest_ind (df1,df2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [83], in <cell line: 8>()
      5 get_ipython().run_line_magic('matplotlib', 'inline')
      6 import os
----> 8 ttest,p_value = ttest_ind (df1,df2)

File ~\anaconda3\lib\site-packages\scipy\stats\stats.py:6133, in ttest_ind(a, b, axis, equal_var, nan_policy, permutations,
random_state, alternative, trim)
   6130 n2 = b.shape[axis]
   6132 if trim == 0:
-> 6133     v1 = np.var(a, axis, ddof=1)
   6134     v2 = np.var(b, axis, ddof=1)
   6135     m1 = np.mean(a, axis)

File <__array_function__ internals>:5, in var(*args, **kwargs)

File ~\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3723, in var(a, axis, dtype, out, ddof, keepdims, where)
   3720     else:
   3721         return var(axis=axis, dtype=dtype, out=out, ddof=ddof, **kwargs)
-> 3723 return _methods._var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
   3724                      **kwargs)

File ~\anaconda3\lib\site-packages\numpy\core\_methods.py:222, in _var(a, axis, dtype, out, ddof, keepdims, where)
    220     div = rcount.reshape(arrmean.shape)
    221 if isinstance(arrmean, mu.ndarray):
--> 222     arrmean = um.true_divide(arrmean, div, out=arrmean, casting='unsafe',
    223                              subok=False)
    224 else:
    225     arrmean = arrmean.dtype.type(arrmean / rcount)

TypeError: unsupported operand type(s) for /: 'method' and 'int'
```

In [84]:

```
t_stat, p_value = levene(df["count"],df["workingday"])
p_value
alpha = 0.5
```

In [85]:

```
p_value
```

Out[85]:

```
0.0
```

In [ ]:

```
#Hypothesis Testing (30 Points):
#2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented (10 points)
#ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season (10 points)
#Chi-square test to check if Weather is dependent on the season (10 points)
```

In [86]:

```
t_stat, p_value = levene(df["count"],df["workingday"])
p_value
alpha = 0.5
```

In [ ]:

```
#2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented (10 points)
#H0 = There is no effect of Working Day on the number of electic cycles rented.
#Ha = There is an effect of WorkingDay on the number of electric cycles rented.
#Right/Left/Two_tailed Test_statistic
#Using ttest_ind
```

In [87]:

```
ttest_ind(df["count"], df["workingday"])
```

Out[87]:

```
Ttest_indResult(statistic=109.95076974934595, pvalue=0.0)
```

In [88]:

```
population_mean_count = df["count"].mean()
population_mean_count
```

Out[88]:

```
191.57413191254824
```

In [89]:

```
#1.Working Day has effect on number of electric cycles rented
population_mean_count = df["count"].mean()
population_mean_count
```

Out[89]:

```
191.57413191254824
```

In [91]:

```
df_workingday_count = df[df["workingday"] == 1]["count"]
df_workingday_count.mean()
```

Out[91]:

```
193.01187263896384
```

In [93]:

```
df_nworkingday_count = df[df["workingday"] == 0]["count"]
df_nworkingday_count.mean()
```

Out[93]:

```
188.50662061024755
```

In [ ]:

```
#ANOVA testing
```

In [ ]:

```
#H0 = Working day does not have any effect on number of cycles rented.
#HA = Working day has an positive effect on number of cycles rented.
# take 99% confidence level
```

In [94]:

```
alpha = 0.01
f_stat, p_value = f_oneway(df_workingday_count,df_nworkingday_count)
print(f"Test statistic = {f_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
Test statistic = 1.4631992635777575 pvalue = 0.22644804226428558
Fail to reject Null Hypothesis
```

In [ ]:

```
#We can say that Working day has an positive effect on number of cycles rented
```

In [ ]:

```
#Using ttest
```

In [ ]:

```
#H0 = Working day does not have any effect on number of cycles rented.
#HA = Working day has an positive effect on number of cycles rented.
# take 99% confidence level
```

In [96]:

```
alpha = 0.01
t_stat, p_value = ttest_ind(df_workingday_count,df_nworkingday_count, alternative = "greater")
print(f"Test statistic = {t_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
Test statistic = 1.2096277376026694 pvalue = 0.11322402113180674
Fail to reject Null Hypothesis
```

In [ ]:

```
#We can say that Working day has an positive effect on number of cycles rented
```

In [97]:

```
# 2.No. of cycles rented similar or different in different seasons
df_season1_spring = df[df["season"] == 1]["count"]
df_season1_spring_subset = df_season1_spring.sample(100)
```

In [98]:

```
df_season2_summer =df[df["season"] == 2]["count"]
df_season2_summer_subset = df_season2_summer.sample(100)
```

In [99]:

```
df_season3_fall = df[df["season"] == 3]["count"]
df_season3_fall_subset = df_season3_fall.sample(100)
```

In [100]:

```
df_season4_winter = df[df["season"] == 4]["count"]
df_season4_winter_subset = df_season4_winter.sample(100)
```

In [101]:

```
#H0 = All samples have equal variance
#HA = At least one sample will have different variance
t_stat, p_value = levene(df_season1_spring, df_season2_summer, df_season3_fall, df_season4_winter)
p_value
```

Out[101]:

```
1.0147116860043298e-118
```

In [102]:

```
#H0 = All samples have equal variance
#HA = At least one sample will have different variance
t_stat, p_value = levene(df_season1_spring, df_season2_summer, df_season3_fall, df_season4_winter)
p_value
```

Out[102]:

```
1.0147116860043298e-118
```

In [103]:

```
#H0 = Sample is drawn from NormalDistribution
#HA = Sample is not from Normal Distribution
##Here we are considering alpha (significance value as ) 0.05
t_stat, pvalue = shapiro(df_season1_spring_subset)
if pvalue < 0.05:
 print("Reject H0 Data is not Gaussian")
else:
 print("Fail to reject Data is Gaussian")
```

```
Reject H0 Data is not Gaussian
```

In [104]:

```
t_stat, pvalue = shapiro(df_season2_summer_subset)
if pvalue < 0.05:
 print("Reject H0 Data is not Gaussian")
else:
 print("Fail to reject Data is Gaussian")
```

```
Reject H0 Data is not Gaussian
```

In [105]:

```
t_stat, pvalue = shapiro(df_season3_fall_subset)
if pvalue < 0.05:
 print("Reject H0 Data is not Gaussian")
else:
 print("Fail to reject Data is Gaussian")
```

```
Reject H0 Data is not Gaussian
```

In [106]:

```
t_stat, pvalue = shapiro(df_season4_winter_subset)
if pvalue < 0.05:
 print("Reject H0 Data is not Gaussian")
else:
 print("Fail to reject Data is Gaussian")
```

```
Reject H0 Data is not Gaussian
```

In [107]:

```
#H0 = season does not have any effect on number of cycles rented.
#HA = At least one season out of four (1:spring, 2:summer,3:fall, 4:winter) has an effect on number of cycles rented.
#Righ Tailed /Left/Two
#Test Statistic and p_value
#We will consider alpha as 0.01 significance value. i.e 99% confidence
alpha = 0.01
f_stat, p_value = f_oneway(df_season1_spring, df_season2_summer, df_season3_fall, df_season4_winter)
print(f"Test statistic = {f_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
Test statistic = 236.94671081032106 pvalue = 6.164843386499654e-149
Reject Null Hypothesis
```

In [108]:

```
#3.No. of cycles rented similar or different in different weather
df_weather1_clear = df[df["weather"] == 1]["count"]
df_weather1_clear.mean()
```

Out[108]:

```
205.23679087875416
```

In [109]:

```
df_weather2_Mist = df[df["weather"] == 2]["count"]
df_weather2_Mist.mean()
```

Out[109]:

```
178.95553987297106
```

In [110]:

```
df_weather3_LightSnow = df[df["weather"] == 3]["count"]
df_weather3_LightSnow.mean()
```

Out[110]:

```
118.84633294528521
```

In [111]:

```
df_weather4_HeavyRain = df[df["weather"] == 4]["count"]
df_weather4_HeavyRain.mean()
```

Out[111]:

```
164.0
```

In [118]:

```
df_weather4_HeavyRain = df[df["weather"] == 4]["count"]
df_weather4_HeavyRain
```

Out[118]:

```
5631    164
Name: count, dtype: int64
```

In [112]:

```
t_stat, p_value = levene(df_weather1_clear, df_weather2_Mist, df_weather3_LightSnow, df_weather4_HeavyRain)
p_value
```

Out[112]:

```
3.504937946833238e-35
```

In [ ]:

```
#Shapiro == Test for normality
```

In [113]:

```
#H0 = Sample is drawn from NormalDistribution
#HA = Sample is not from Normal Distribution
##Here we are considering alpha (significance value as ) 0.05
shapiro(df_weather1_clear)
```

Out[113]:

```
ShapiroResult(statistic=0.8909225463867188, pvalue=0.0)
```

In [114]:

```
shapiro(df_weather2_Mist)
```

Out[114]:

```
ShapiroResult(statistic=0.8767688274383545, pvalue=9.781063280987223e-43)
```

In [115]:

```
shapiro(df_weather3_LightSnow)
```

Out[115]:

```
ShapiroResult(statistic=0.7674333453178406, pvalue=3.876134581802921e-33)
```

In [116]:

```
shapiro(df_weather4_HeavyRain)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [116], in <cell line: 1>()
----> 1 shapiro(df_weather4_HeavyRain)

File ~\anaconda3\lib\site-packages\scipy\stats\morestats.py:1749, in shapiro(x)
   1747 N = len(x)
   1748 if N < 3:
-> 1749     raise ValueError("Data must be at least length 3.")
   1751 a = zeros(N, 'f')
   1752 init = 0

ValueError: Data must be at least length 3.
```

In [ ]:

```
#Using ANOVA
```

In [119]:

```
#H0 = weather does not have any effect on number of cycles rented.
#HA = At least one weather out of four (1: clear, 2: Mist, 3:Light snow, 4:Heavy Rain) has an effect on number of cycles re
#Righ Tailed /Left/Two
#Test Statistic and p_value
#We will consider alpha as 0.01 significance value. i.e 99% confidence
alpha= 0.01
f_stat, p_value = f_oneway(df_weather1_clear, df_weather2_Mist, df_weather3_LightSnow, df_weather4_HeavyRain)
print(f"Test statistic = {f_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
Test statistic = 65.53024112793271 pvalue = 5.482069475935669e-42
Reject Null Hypothesis
```

In [120]:

```
#H0 = weather does not have any effect on number of cycles rented.
#HA = At least one weather out of four (1: clear, 2: Mist, 3:Light snow, 4:Heavy Rain) has an effect on number of cycles re
#Righ Tailed /Left/Two
#Test Statistic and p_value
#We will consider alpha as 0.01 significance value. i.e 99% confidence
alpha= 0.01
f_stat, p_value = f_oneway(df_weather1_clear, df_weather2_Mist, df_weather3_LightSnow)
print(f"Test statistic = {f_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
Test statistic = 98.28356881946706 pvalue = 4.976448509904196e-43
Reject Null Hypothesis
```

In [ ]:

```
#It shows that weather has effect on the number of cycles rented
```

In [ ]:

```
#4. Chi-square test to check if Weather is dependent on the season (10 points)
```

In [121]:

```
val = pd.crosstab(index= df["weather"],columns = df["season"])
```

In [122]:

```
print(val)
chisquare (val)
```

```
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
```

Out[122]:

```
Power_divergenceResult(statistic=array([2749.33581534, 2821.39590194, 3310.63995609, 2531.07388442]), pvalue=array([0., 0.,
0., 0.]))
```

In [123]:

```
#H0 = Weather is not dependent on season
#Ha = Weather is dependent on season
chi_stat,p_value,df,confusion_matrix =chi2_contingency(val)
print(f"chi_stat = {chi_stat} pvalue = {p_value}")
if (p_value < alpha):
 print("Reject Null Hypothesis")
else:
 print("Fail to reject Null Hypothesis")
```

```
chi_stat = 49.15865559689363 pvalue = 1.5499250736864862e-07
Reject Null Hypothesis
```

In [ ]:

```
#Weather is dependent on season
```