In [1]:
```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from matplotlib import pyplot
import pylab as py

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Converting the txt file to csv file

df = pd.read_csv(r'bike_sharing.txt')
df.to_csv (r'bike_sharing.csv', index=None)
```

In [3]:
```
df
```

Out[3]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	ca
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	
...	...	...	...	...	...	...	...	...	...	
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	

10886 rows × 12 columns

1. Define Problem Statement and perform Exploratory Data Analysis.

1.a. Definition of problem (as per given problem statement with additional views)
=> 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.
=> To know when do people use our bikes the max and what we should do to increase our business.
=> Who are our target customers? => Which all variables are dependent on each other?

1.b. Observations on shape of data, data types of all the attributes, conversion of categorical attributes to 'category' (If required) , missing value detection, statistical summary.

In [4]:
```
df.shape
```

Out[4]:
```
(10886, 12)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [6]:
```
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 [7]:
```
# conversion of categorical attributes to 'category' and datetime attribute to 'd

df['season'] = df.season.astype('category')
df['holiday'] = df.holiday.astype('category')
df['workingday'] = df.workingday.astype('category')
df['weather'] = df.weather.astype('category')
df['datetime'] = pd.to_datetime(df['datetime'])

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  category
 2   holiday     10886 non-null  category
 3   workingday  10886 non-null  category
 4   weather     10886 non-null  category
 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: category(4), datetime64[ns](1), float64(3), int64(4)
memory usage: 723.7 KB
```

In [8]:
```
numerical_cols = ['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered
categorical_cols = ['season', 'holiday', 'workingday', 'weather']
```

In [9]:
```
df[numerical_cols].describe()
```

Out[9]:

	temp	atemp	humidity	windspeed	casual	registered	
count	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.(
mean	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.5
std	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.1
min	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.(
25%	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.(
50%	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.(
75%	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.(
max	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.(

In [10]:
```
df[categorical_cols].describe()
```

Out[10]:

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

In [11]:
```
# Count the number of null values in each columns

df.isna().sum()
```

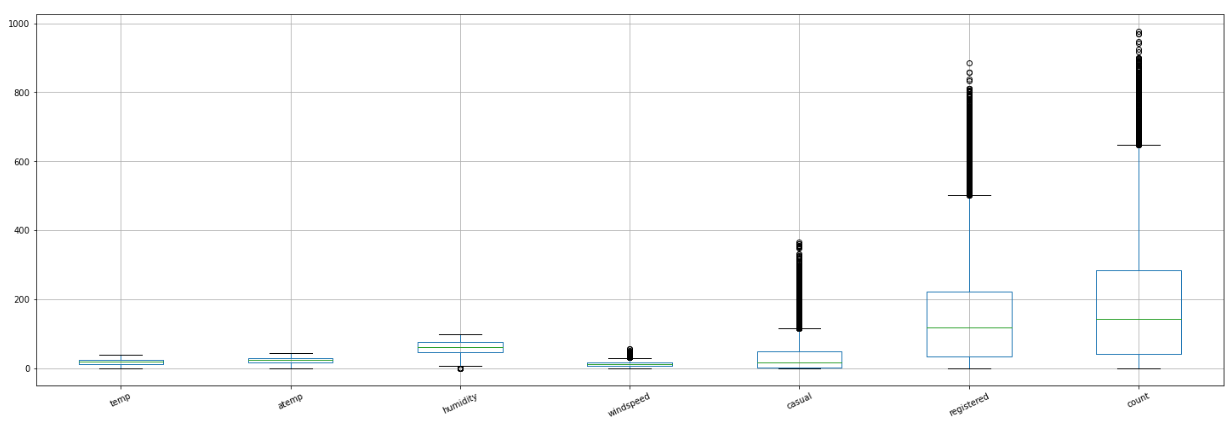
Out[11]:
```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

Inference: There are no missing values in the dataset.

Handling Outliers

In [12]:
```
df[numerical_cols].boxplot(rot=25, figsize=(25,8))
```

Out[12]: <AxesSubplot:>



In [13]:
```
Q1 = df[numerical_cols].quantile(0.25)
Q3 = df[numerical_cols].quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
temp           12.3000
atemp          14.3950
humidity       30.0000
windspeed       9.9964
casual         45.0000
registered    186.0000
count         242.0000
dtype: float64
```

In [14]:
```
df = df[~((df[numerical_cols] < (Q1 - 1.5 * IQR)) |(df[numerical_cols] > (Q3 + 1.
df = df.reset_index(drop=True)
```

In [15]:
```
df
```

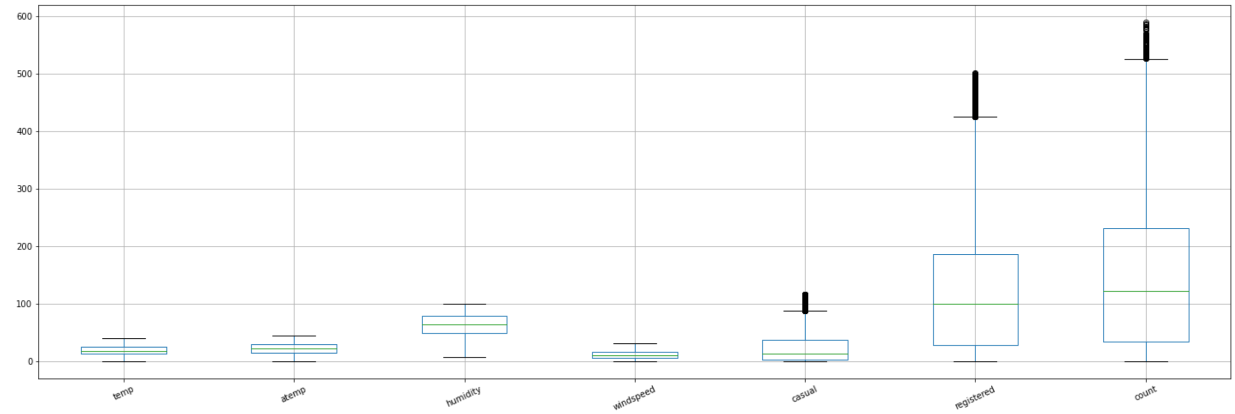
Out[15]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0
...	...	...	...	...	...	...	...	...	...	...
9513	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7
9514	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10
9515	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4
9516	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12
9517	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4

9518 rows × 12 columns

In [16]:
```
df[numerical_cols].boxplot(rot=25, figsize=(25,8))
```

Out[16]: <AxesSubplot:>



1.c. Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplots of all the categorical variables)

In [17]:

```
df
```

Out[17]:

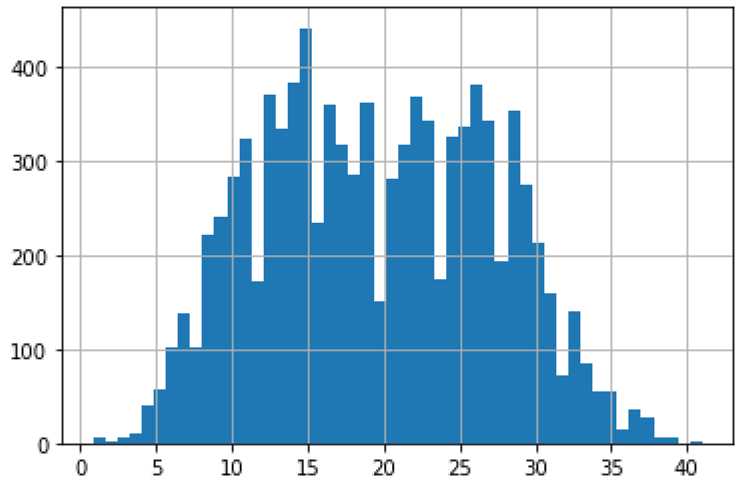
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0
...	...	...	...	...	...	...	...	...	...	...
9513	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7
9514	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10
9515	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4
9516	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12
9517	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4

9518 rows × 12 columns

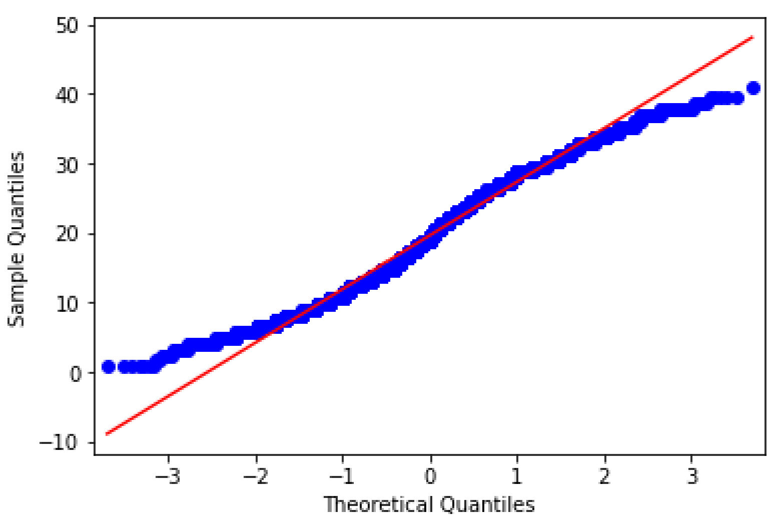
Numerical columns

In [18]:
```
df["temp"].hist(bins=50)
```

Out[18]: <AxesSubplot:>



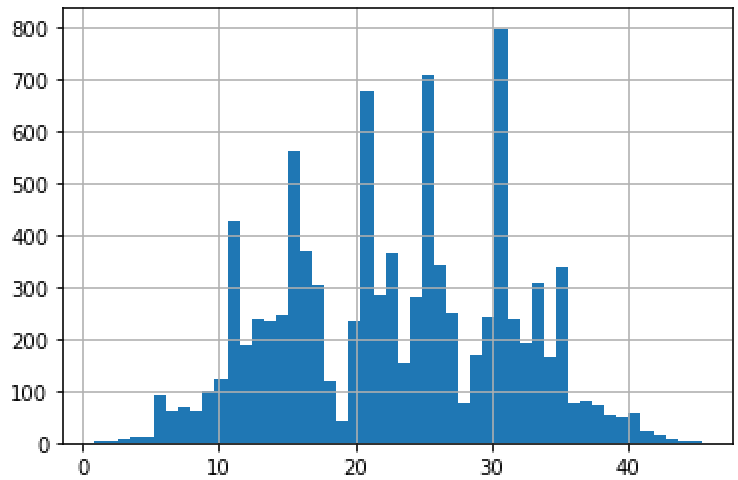
In [19]:
```
sm.qqplot(df["temp"], line ='s')
py.show()
```



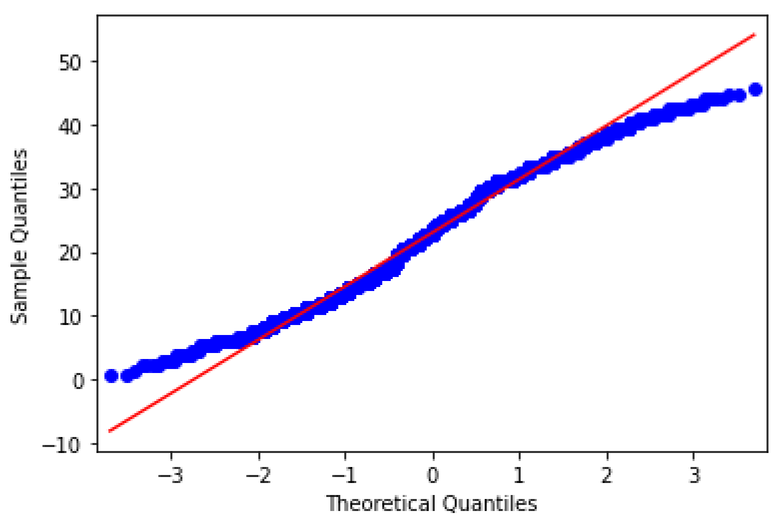
Inference: temp is almost following Normal Distribution

In [20]:
```
df["atemp"].hist(bins=50)
```

Out[20]: <AxesSubplot:>



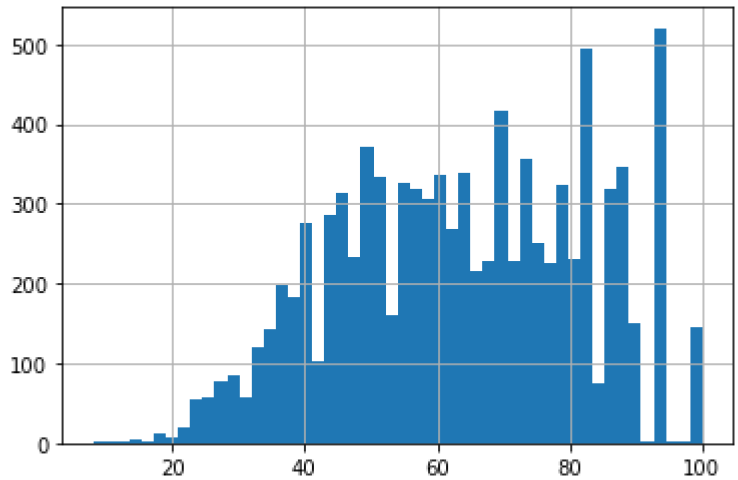
In [21]:
```
sm.qqplot(df["atemp"], line ='s')
py.show()
```



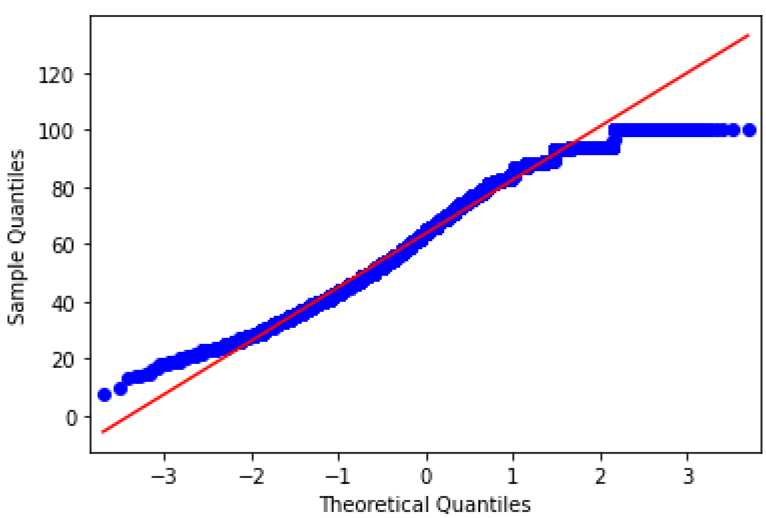
Inference: atemp is almost following Normal distribution

In [22]:
```
df["humidity"].hist(bins=50)
```

Out[22]: <AxesSubplot:>



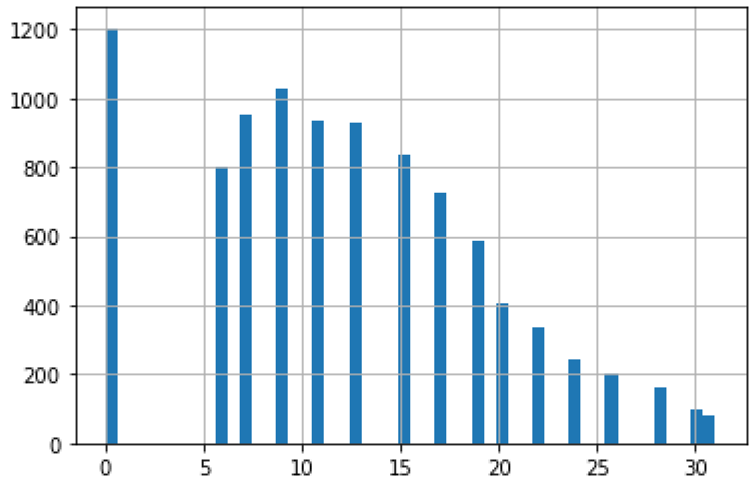
In [23]:
```
sm.qqplot(df["humidity"], line ='s')
py.show()
```



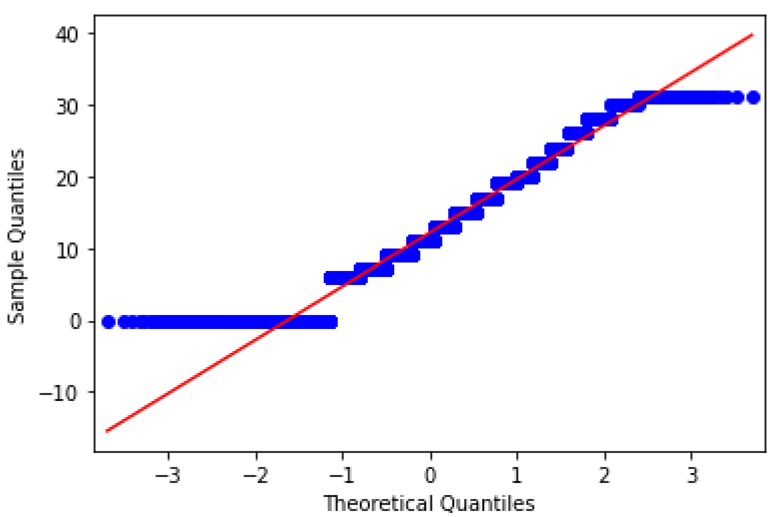
Inference: humidity is almost following Normal distribution

In [24]:
```
df["windspeed"].hist(bins=50)
```

Out[24]: <AxesSubplot:>



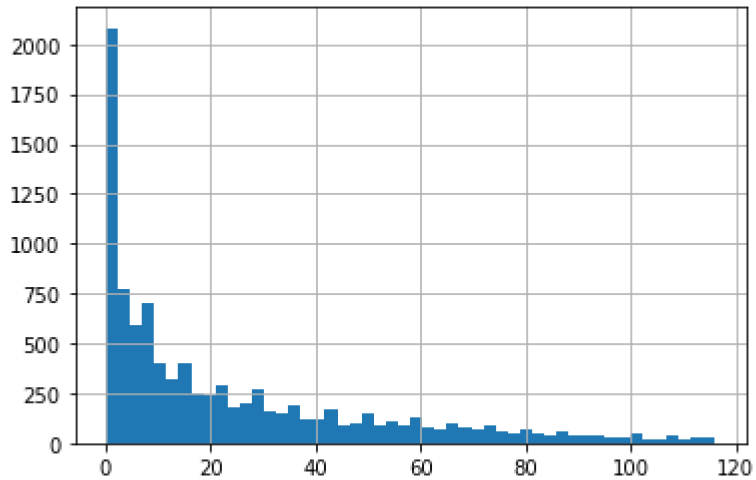
In [25]:
```
sm.qqplot(df["windspeed"], line ='s')
py.show()
```



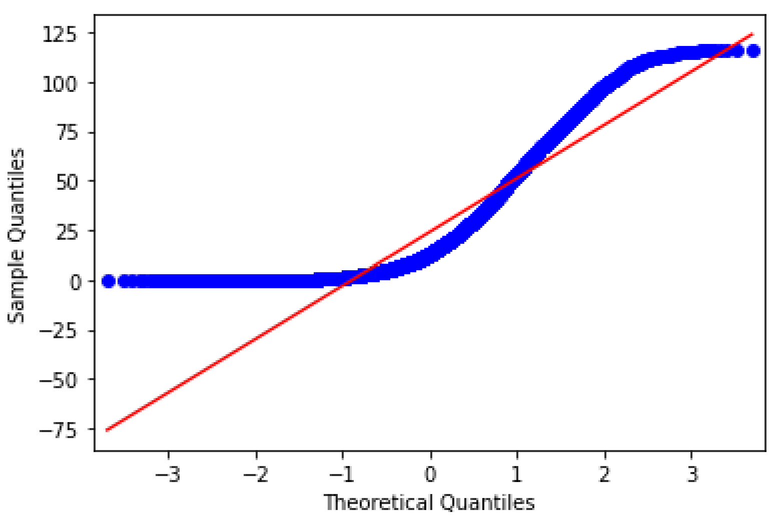
Inference: windspeed is almost following Normal distribution

In [26]:
```
df["casual"].hist(bins=50)
```

Out[26]: <AxesSubplot:>



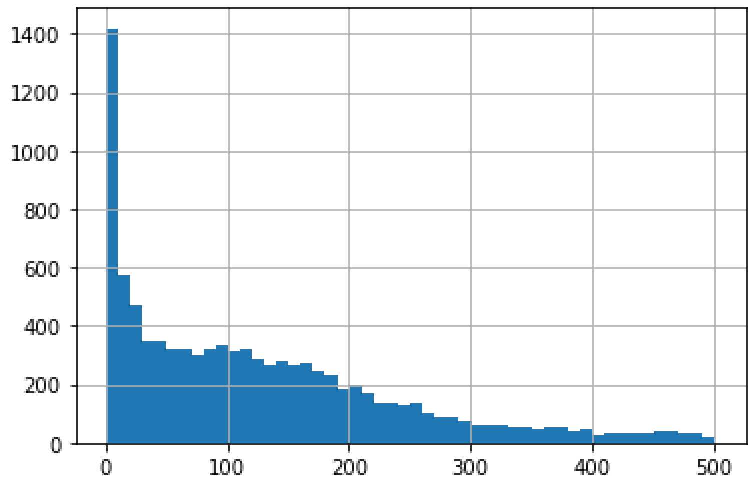
In [27]:
```
sm.qqplot(df["casual"], line ='s')
py.show()
```



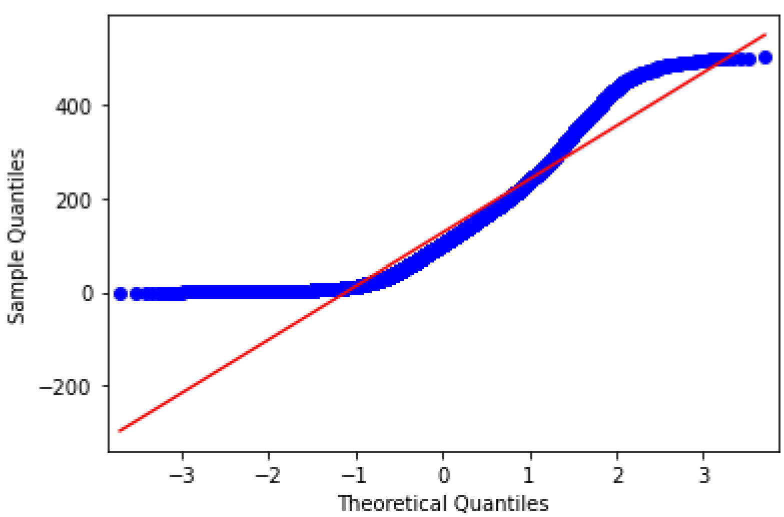
Inference: count of casuals are not following Normal distribution

In [28]:
```
df["registered"].hist(bins=50)
```

Out[28]: <AxesSubplot:>



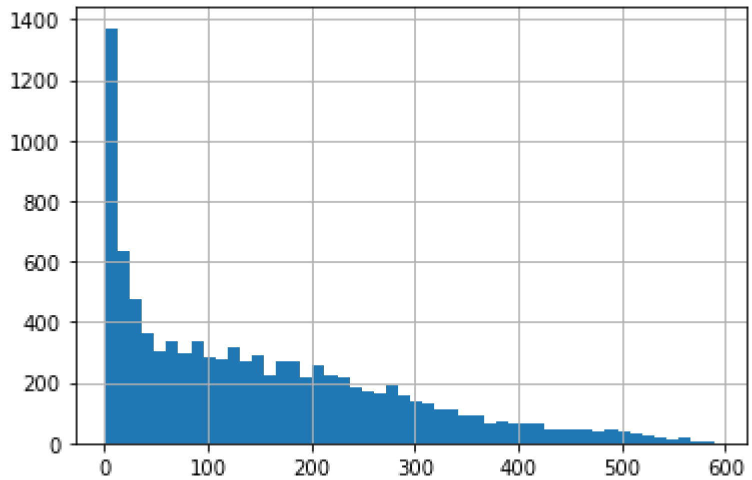
In [29]:
```
sm.qqplot(df["registered"], line ='s')
py.show()
```



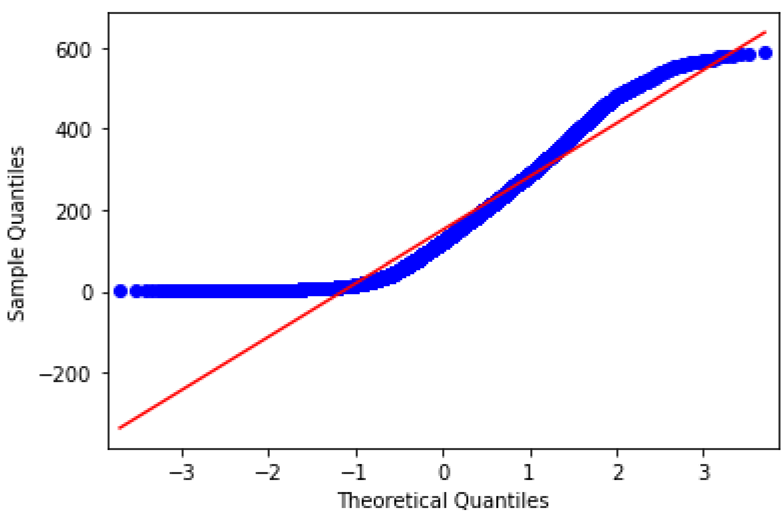
Inference: count of registered are not following Normal distribution

In [30]:
```
df["count"].hist(bins=50)
```

Out[30]: <AxesSubplot:>



In [31]:
```
sm.qqplot(df["count"], line ='s')
py.show()
```



Inference: count(both casual and registered) are not following Normal distribution

Categorical columns

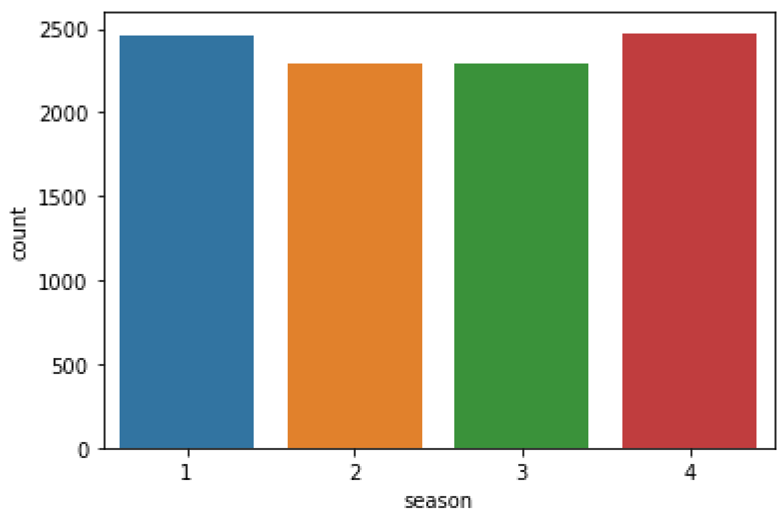
season: season (1: spring, 2: summer, 3: fall, 4: winter) holiday : weather day is a holiday or not

workingday : if day is neither weekend nor holiday is 1, otherwise is 0.

weather:
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

In [32]:
```
sns.countplot(x="season", data=df)
```

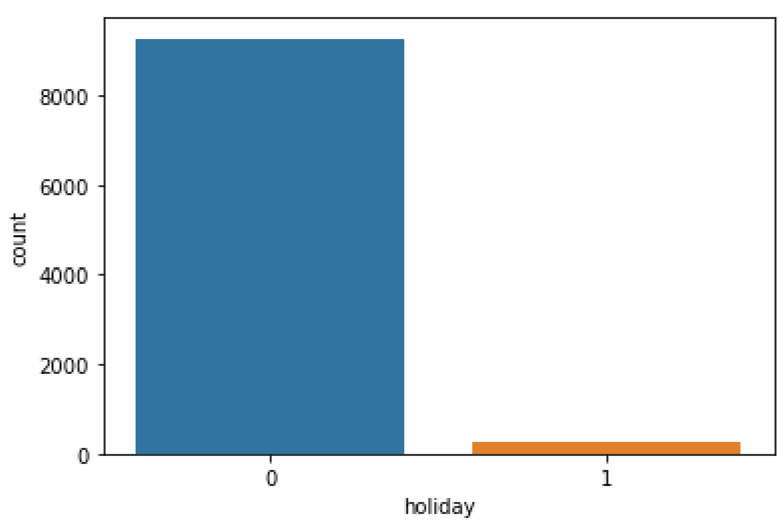
Out[32]: <AxesSubplot:xlabel='season', ylabel='count'>



Inference: in all the seasons, number of rides taken used are the same.

In [33]:
```
sns.countplot(x="holiday", data=df)
```

Out[33]: <AxesSubplot:xlabel='holiday', ylabel='count'>

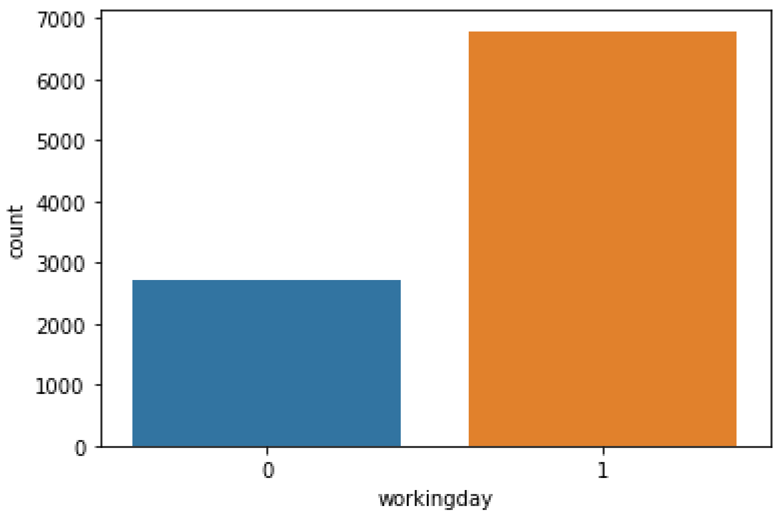


Inference: number of rides in the non-holidays(weekdays and weekends) are more than holidays(weekends not included).

Recommendation: That means during holidays, not much rental bikes are used, so you can use this time to service and maintenace of the bikes.

In [34]:
```
sns.countplot(x="workingday", data=df)
```

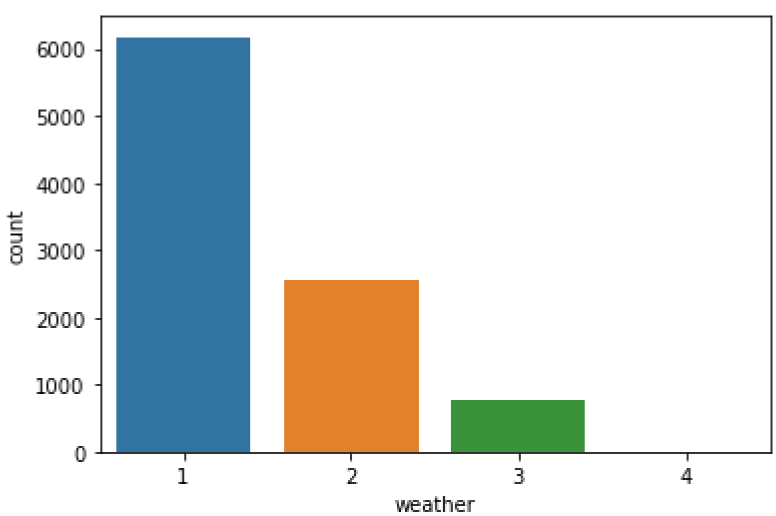
Out[34]: <AxesSubplot:xlabel='workingday', ylabel='count'>



Inference: In weekends also, people use rental bikes but its lesser than that during workingdays.
Recommendation: You can leave less bikes during non-working days or more bikes in working hours.

In [35]:
```
sns.countplot(x="weather", data=df)
```

Out[35]: <AxesSubplot:xlabel='weather', ylabel='count'>



Inference: During heavy rains(category 4), none of the bikes are used. Most of the bikes are used during category 1 time. And the count reduces as the influence of sun reduces.
Recommendation: During category 4, you can introduce few electric cars.

1.d. Bivariate Analysis (Relationships between important variables such as workday and count, season and count, weather and count.

In [36]:
```
df
```

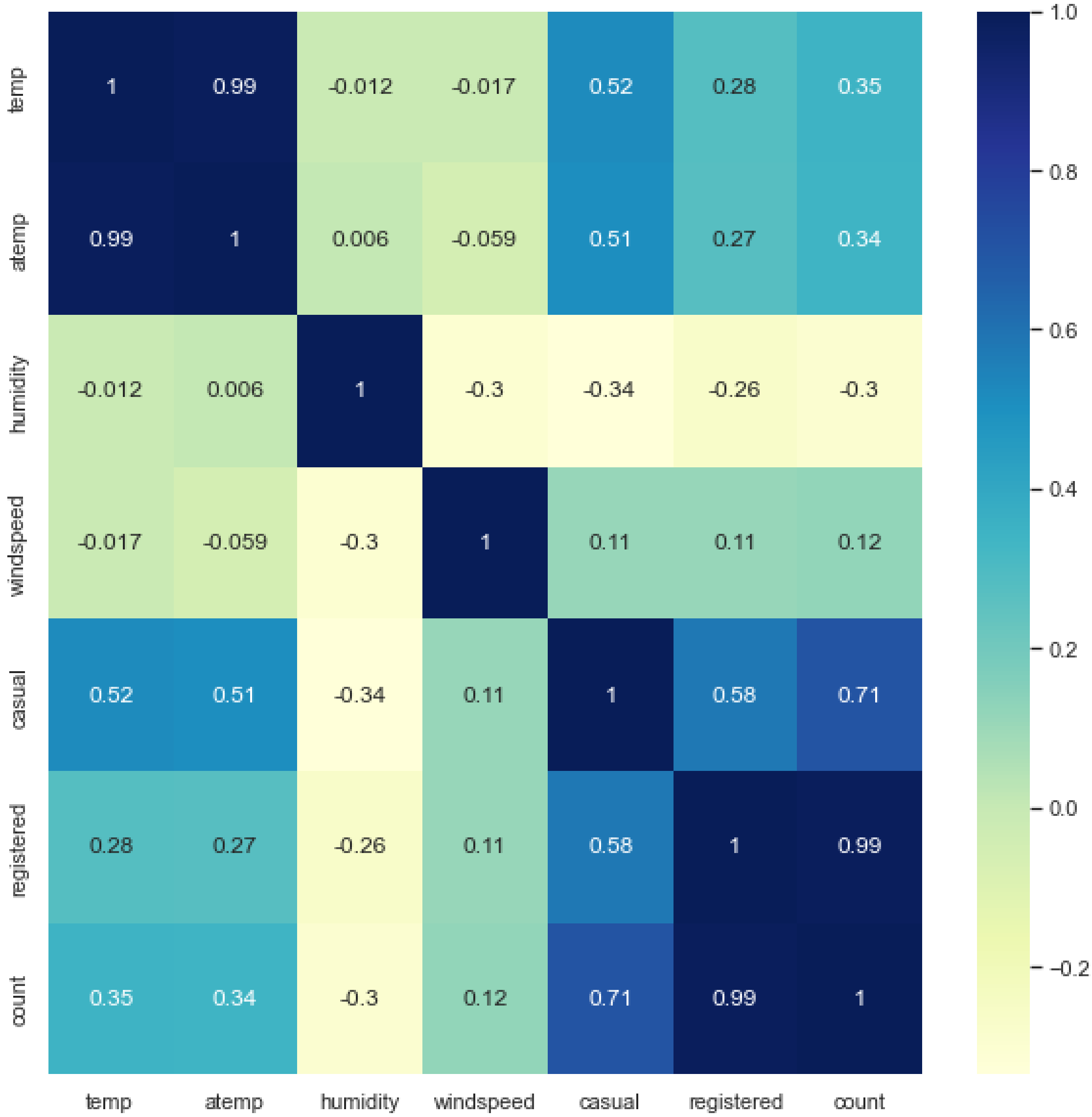
Out[36]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0
...	...	...	...	...	...	...	...	...	...	...
9513	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7
9514	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10
9515	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4
9516	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12
9517	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4

9518 rows × 12 columns

In [37]:
```
sns.set(rc = {'figure.figsize':(10,10)})
sns.heatmap(df[numerical_cols].corr(), cmap="YlGnBu", annot=True)
```

Out[37]: <AxesSubplot:>



In [38]:
```
df.describe()
```

Out[38]:

	temp	atemp	humidity	windspeed	casual	registered	count
count	9518.000000	9518.000000	9518.000000	9518.000000	9518.000000	9518.000000	9518.000000
mean	19.589971	22.987399	63.737025	12.133336	23.955033	126.181025	150.136058
std	7.686871	8.361526	18.693175	7.437481	26.956046	114.116911	131.586548
min	0.820000	0.760000	8.000000	0.000000	0.000000	0.000000	1.000000
25%	13.120000	15.910000	49.000000	7.001500	3.000000	28.250000	34.000000
50%	18.860000	22.725000	64.500000	11.001400	13.000000	101.000000	122.000000
75%	26.240000	30.305000	79.000000	16.997900	37.000000	187.000000	231.000000
max	41.000000	45.455000	100.000000	31.000900	116.000000	501.000000	590.000000

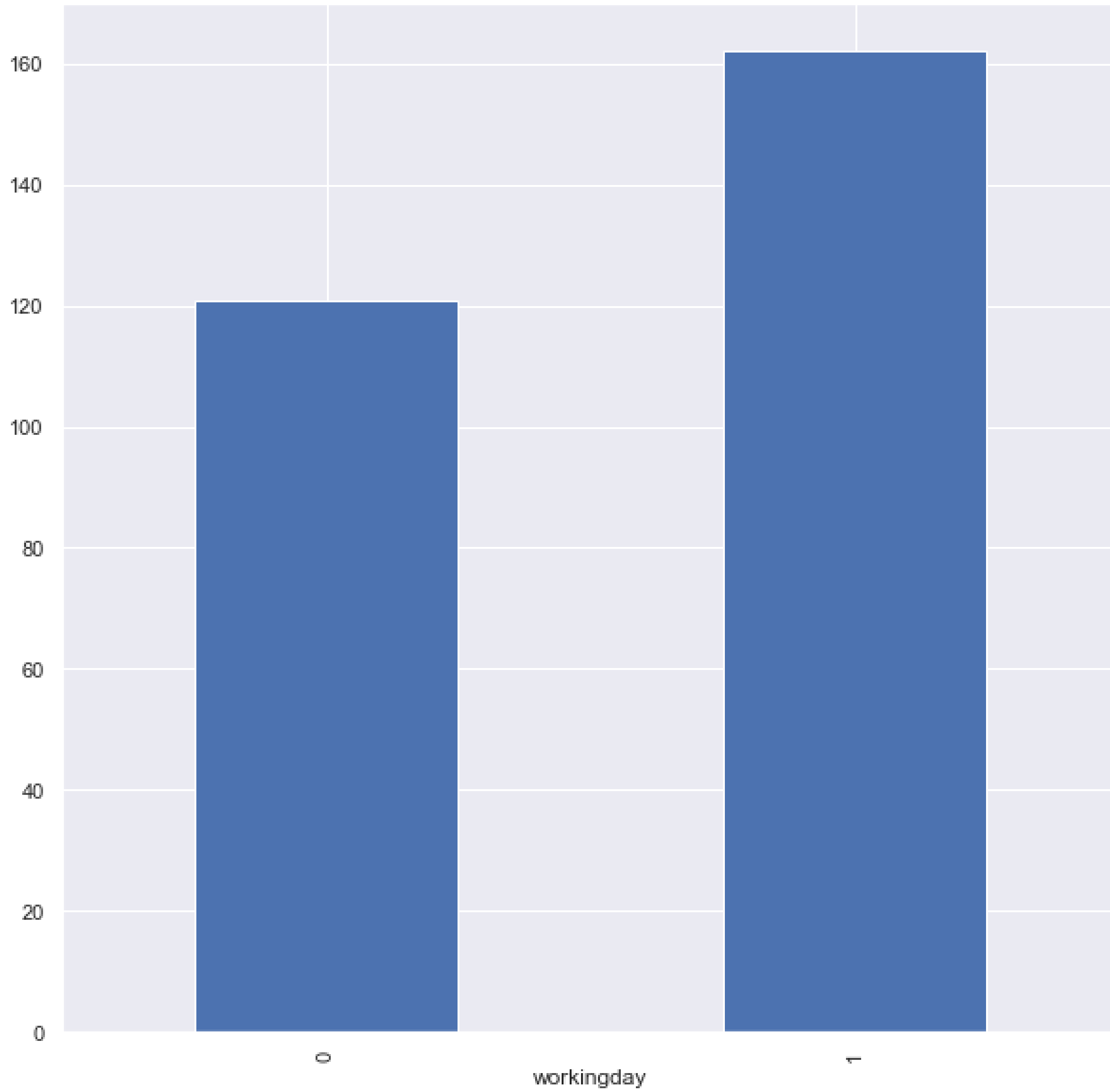
workday v/s count

In [39]:
```
df.groupby("workingday")["count"].mean()
```

Out[39]:
```
workingday
0    120.681085
1    161.970103
Name: count, dtype: float64
```

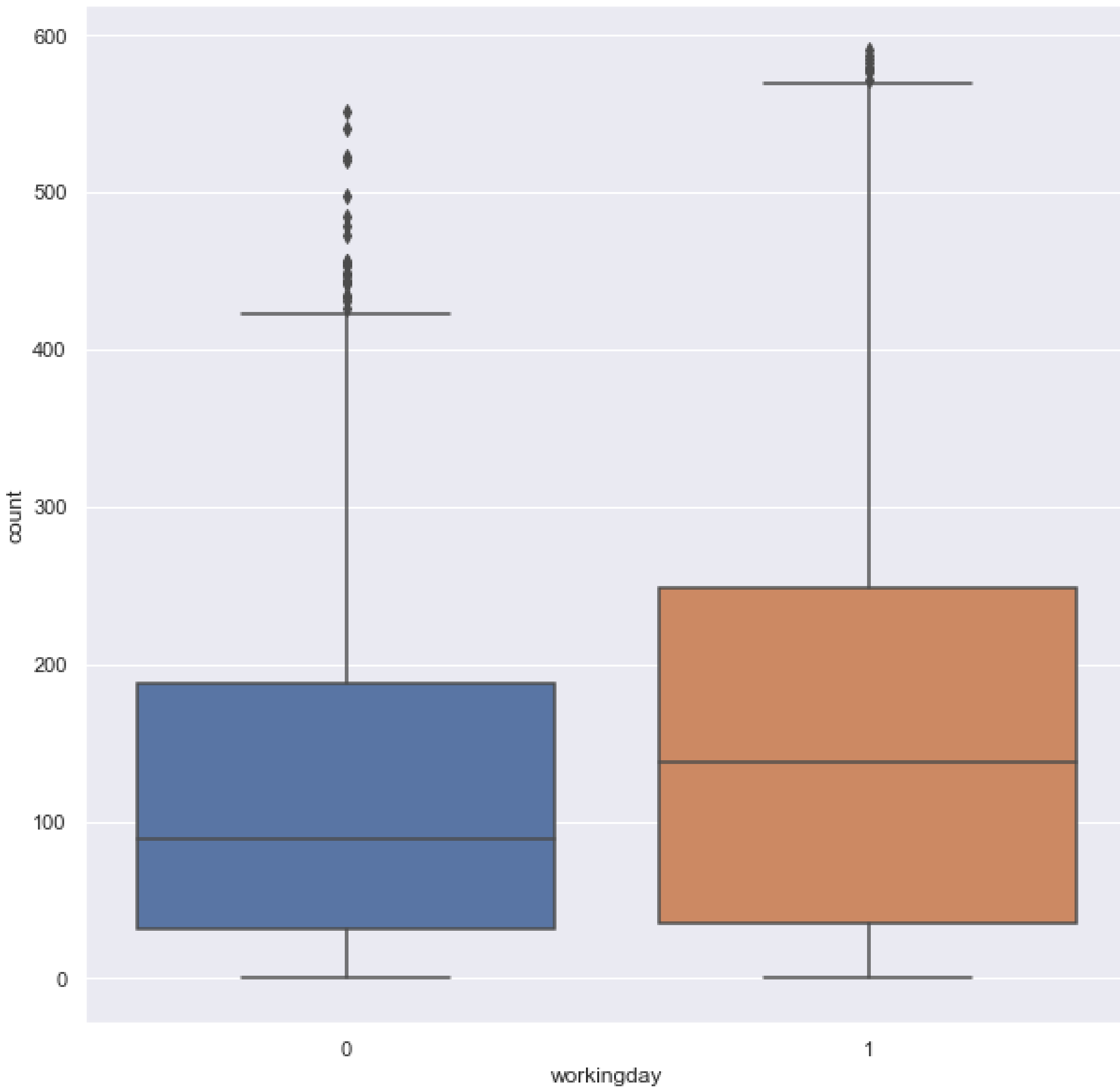
In [40]:
```
df.groupby("workingday")["count"].mean().plot.bar()
```

Out[40]: <AxesSubplot:xlabel='workingday'>



In [41]:
```
sns.boxplot(data = df, x = "workingday", y = "count")
```

Out[41]: <AxesSubplot:xlabel='workingday', ylabel='count'>



Inference: On average, bikes are used workingdays is ~1.5times more than non-workingdays per hour.

In [42]:
```
df.groupby("workingday")["count", 'datetime', 'casual', 'registered'].max()
```

Out[42]:

workingday	count	datetime	casual	registered
0	551	2012-12-16 23:00:00	116	490
1	590	2012-12-19 23:00:00	116	501

Inference: Especially after 9pm, metro & other modes of public transports will stopped, and most of the taxis and autos will be demanding for high amounts, so especially people working at constructions and IT will prefer for electric solo and shared bikes which will reduce the cost of the customer's ride drastically.

Recommendation: In the morning time due to availability of public transport, lot of middle class and one-day wage workers use public transport to reach to their work place. So our main business hours is from 9pm to 12am, we can keep more vehicles in those places where our customers work.

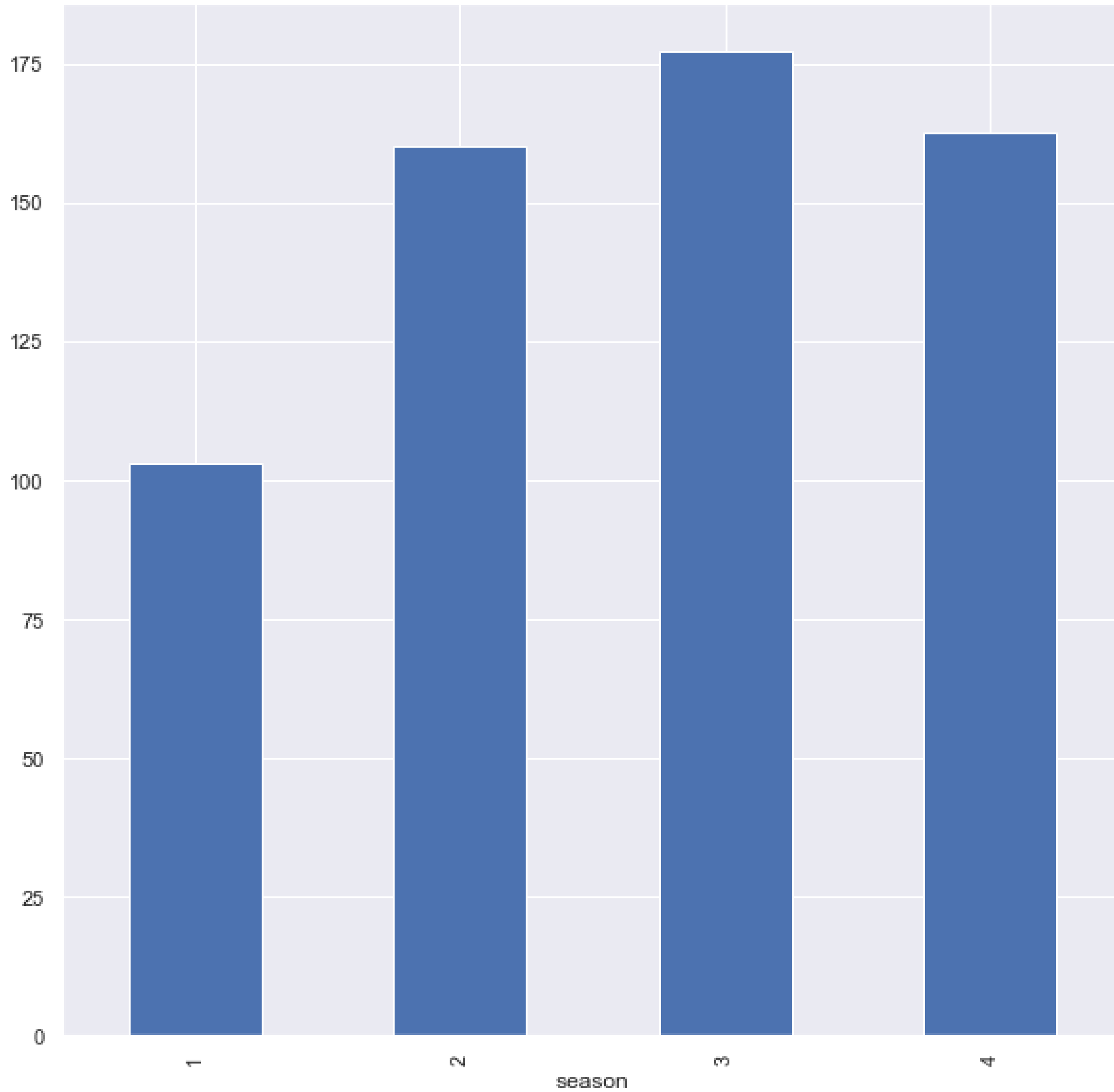
season v/s count

In [43]:
```
df.groupby("season")["count"].mean()
```

Out[43]:
```
season
1    103.164028
2    160.360820
3    177.151661
4    162.437172
Name: count, dtype: float64
```

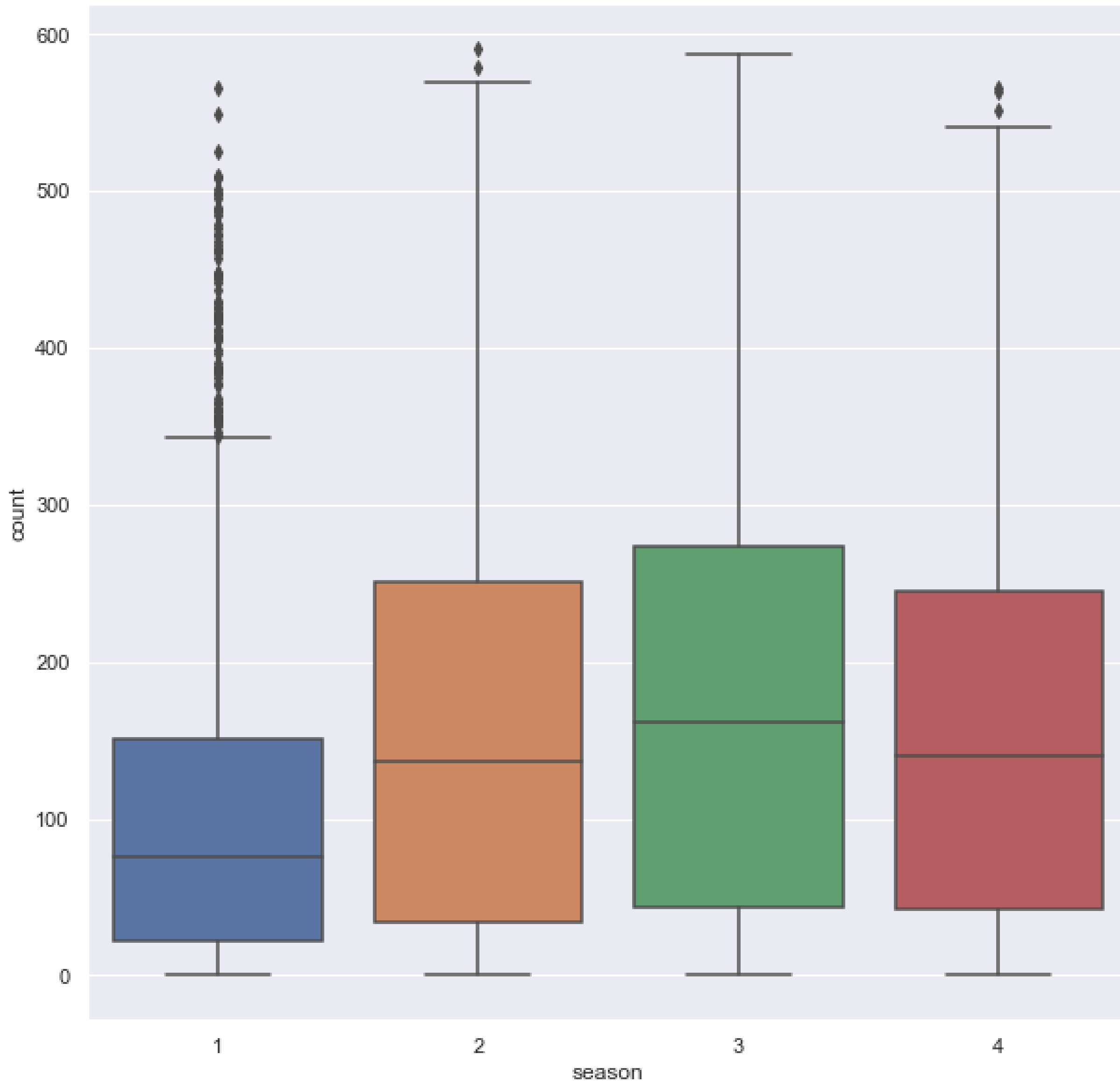
In [44]:
```
df.groupby("season")["count"].mean().plot.bar()
```

Out[44]: <AxesSubplot:xlabel='season'>



In [45]:
```
sns.boxplot(data = df, x = "season", y = "count")
```

Out[45]: <AxesSubplot:xlabel='season', ylabel='count'>



In [46]:
```
df.groupby("season")["count", 'datetime', 'casual', 'registered'].max()
```

Out[46]:

season	count	datetime	casual	registered
1	566	2012-03-19 23:00:00	115	498
2	590	2012-06-19 23:00:00	116	500
3	587	2012-09-19 23:00:00	116	500
4	566	2012-12-19 23:00:00	116	501

Inference: In all the seasons, the peak hours is between 9pm to 12am.

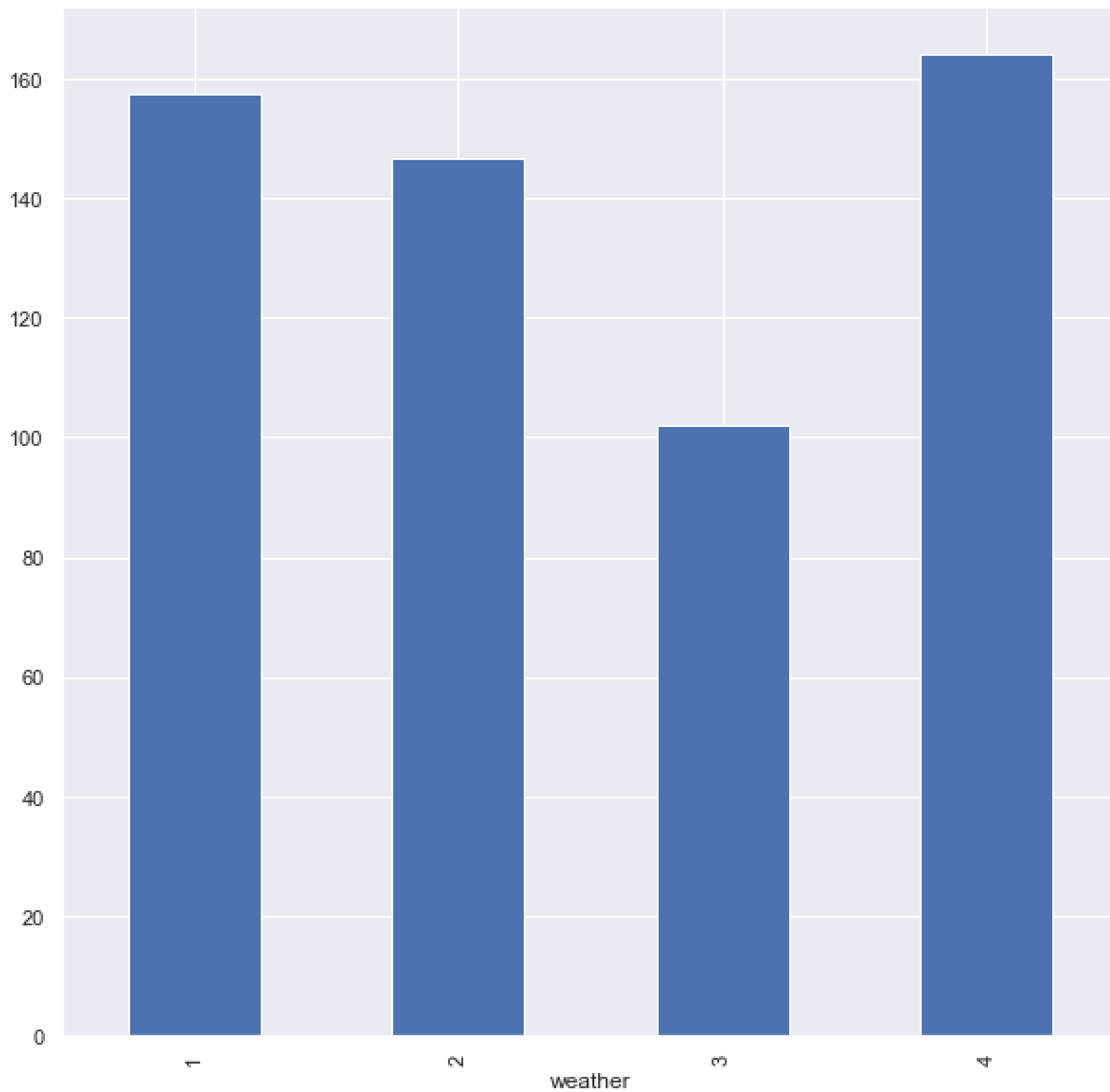
weather v/s count

In [47]:
```
df.groupby("weather")["count"].mean()
```

Out[47]:
```
weather
1    157.522021
2    146.805685
3    102.170763
4    164.000000
Name: count, dtype: float64
```

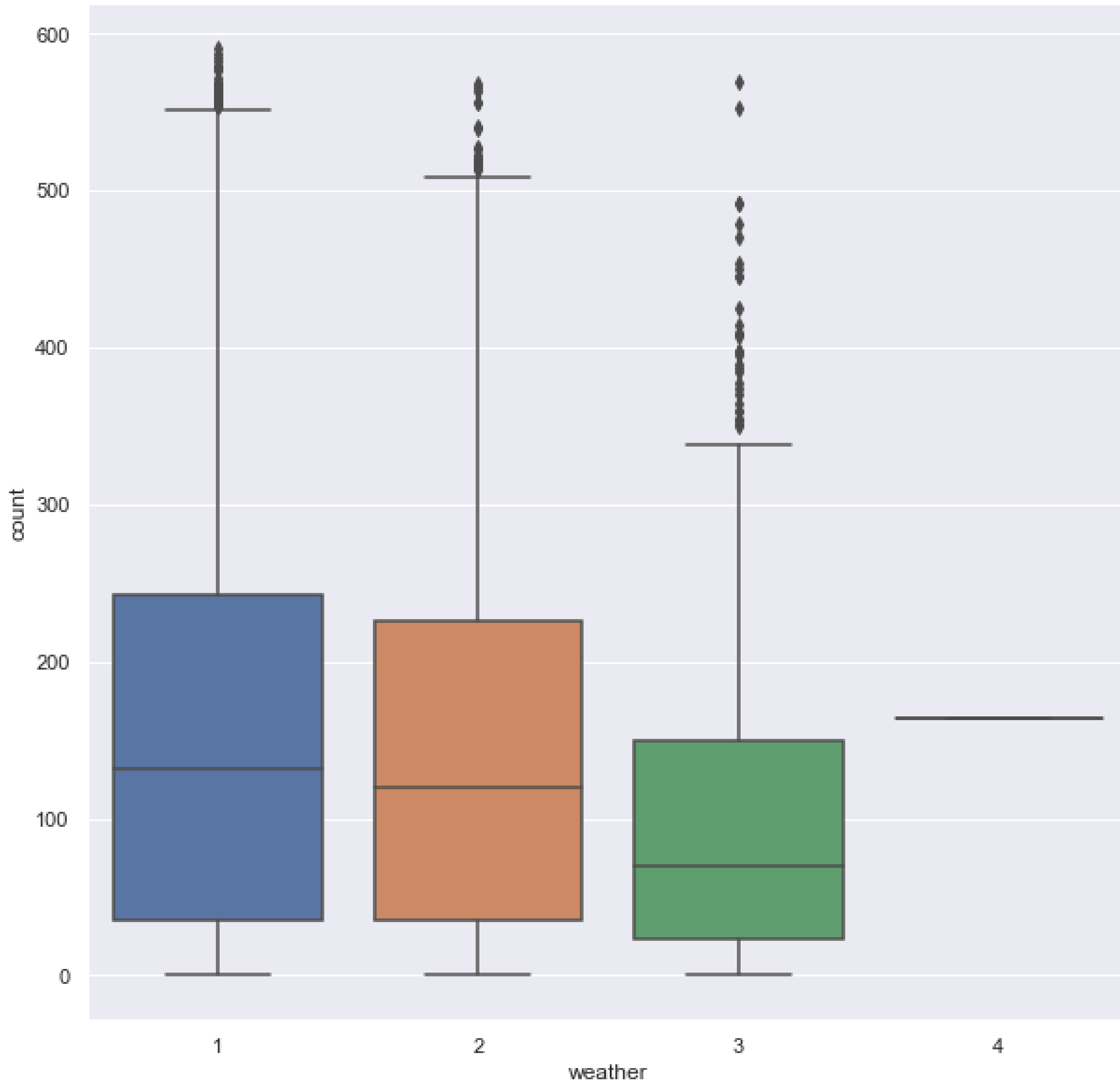
In [48]:
```
df.groupby("weather")["count"].mean().plot.bar()
```

Out[48]: <AxesSubplot:xlabel='weather'>



In [49]:
```
sns.boxplot(data = df, x = "weather", y = "count")
```

Out[49]: <AxesSubplot:xlabel='weather', ylabel='count'>



In [50]:
```
df.groupby("season")["count", 'datetime', 'casual', 'registered'].max()
```

Out[50]:

season	count	datetime	casual	registered
1	566	2012-03-19 23:00:00	115	498
2	590	2012-06-19 23:00:00	116	500
3	587	2012-09-19 23:00:00	116	500
4	566	2012-12-19 23:00:00	116	501

Inference: There are few rides per hour during rainy weather but in the peak hours you can see the same number of rides happening as usual. So during rainy weather, we can only keep the bikes during the peak hours.

2. Hypothesis Testing

2.a. 2- Sample T-Test to check if Working Day has an effect on the number of electric cycles rented (10 points)

ho: count on working day < count on non-working day
ha: count on working day >= count on non-working day

In [51]:
```
workingday_sample = df["workingday"].sample(n=1000)
count_sample = df["count"].sample(n=1000)

stats.ttest_ind(workingday_sample, count_sample, alternative="less")
```

Out[51]:
```
Ttest_indResult(statistic=-36.54376351521165, pvalue=1.183422965693989e-224)
```

Inference: pvalue<<0.05, we reject our Null hypothesis and accept ha.

Therefore, count on working day >= count on non-working day

2.b. ANNOVA to check if No. of cycles rented is similar or different in different 1. weather 2. season (10 points)

2.b.1. weather

ho: count is same in different weathers
h1: count is not same in different weathers

In [52]:
```
df.weather.value_counts()
```

Out[52]:
```
1    6176
2    2568
3     773
4       1
Name: weather, dtype: int64
```

In [53]:
```
df.groupby('weather').agg({'count':'sum'}).reset_index(drop=True)
```

Out[53]:

	count
0	972856
1	376997
2	78978
3	164

In [54]:
```
weather_1 = df[df["weather"]==1]["count"].sample(n=500)
weather_2 = df[df["weather"]==2]["count"].sample(n=500)
weather_3 = df[df["weather"]==3]["count"].sample(n=500)

weather_1 = weather_1.reset_index(drop=True)
weather_2 = weather_2.reset_index(drop=True)
weather_3 = weather_3.reset_index(drop=True)

dataset = pd.DataFrame({"1":weather_1, "2":weather_2, "3":weather_3})
dataset
```

Out[54]:

	1	2	3
0	308	112	312
1	1	339	20
2	103	106	194
3	69	52	132
4	138	180	19
...	...	...	...
495	99	418	1
496	70	237	2
497	29	258	71
498	67	382	407
499	334	497	96

500 rows × 3 columns

In [55]:
```
f, pval = stats.f_oneway(dataset["1"], dataset["2"], dataset["3"])
print(f, pval)
```

```
32.08987980588854 2.2595026108213704e-14
```

Inference: Since pvalue<0.05, we reject ho and go with ha.

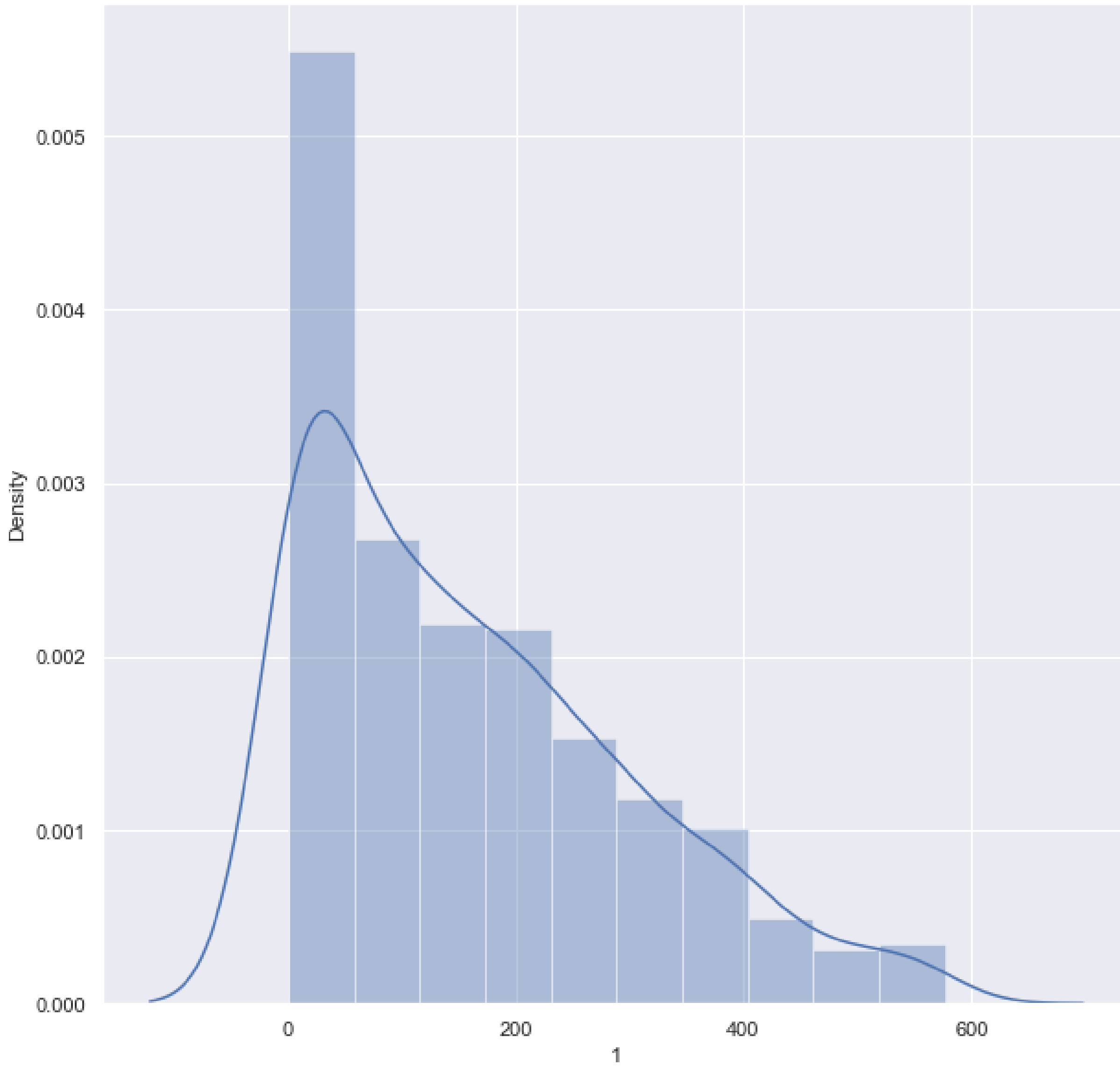
In [56]:
```
mean_1 = dataset['1'].mean()
mean_2 = dataset['2'].mean()
mean_3 = dataset['3'].mean()

print(mean_1, mean_2, mean_3)
```

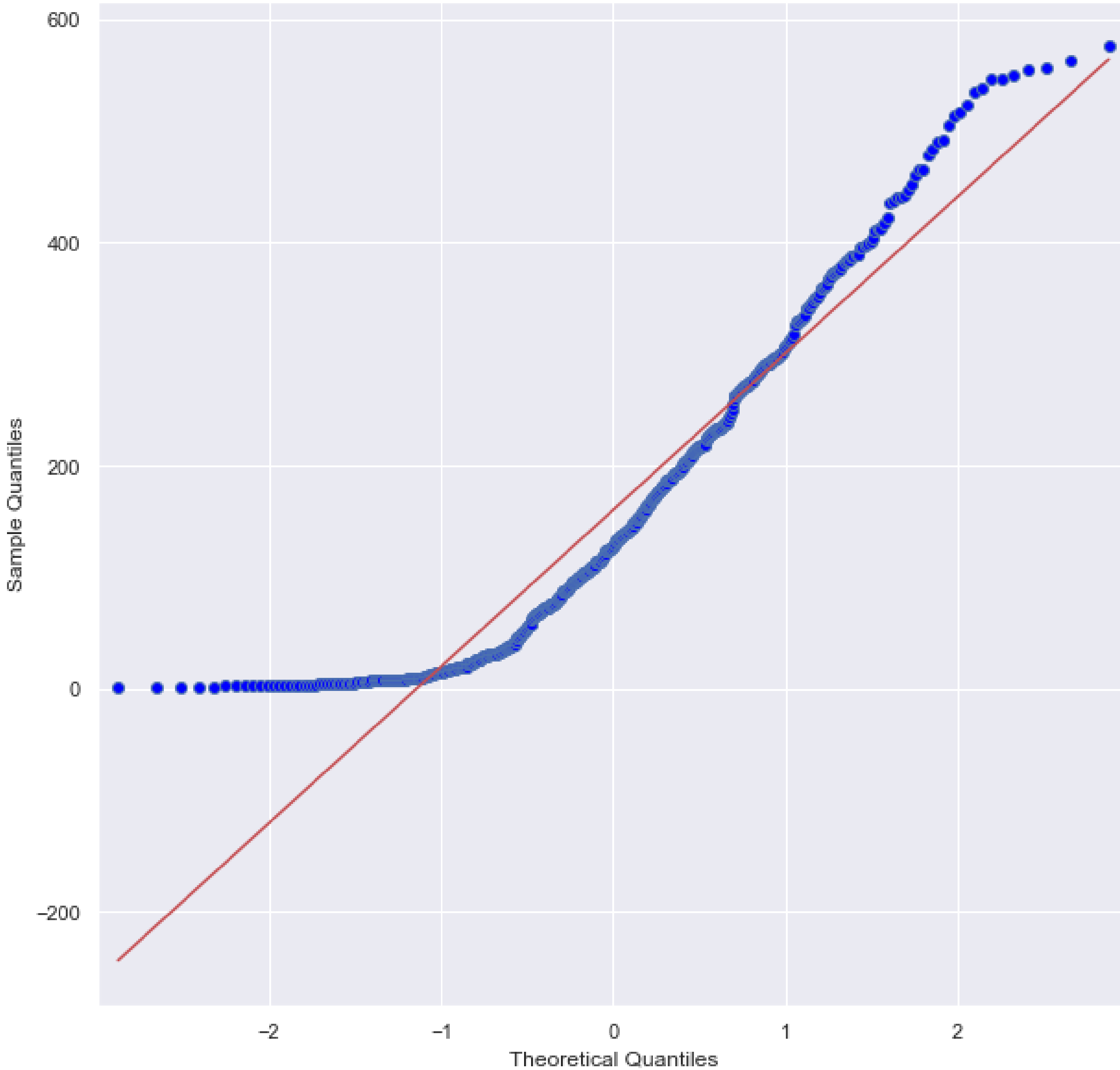
```
160.26 139.146 99.026
```

In [57]:
```
sns.distplot(dataset['1'], bins = 10)
```

Out[57]: <AxesSubplot:xlabel='1', ylabel='Density'>



In [58]:
```
sm.qqplot(dataset['1'], line = "s")
py.show()
```



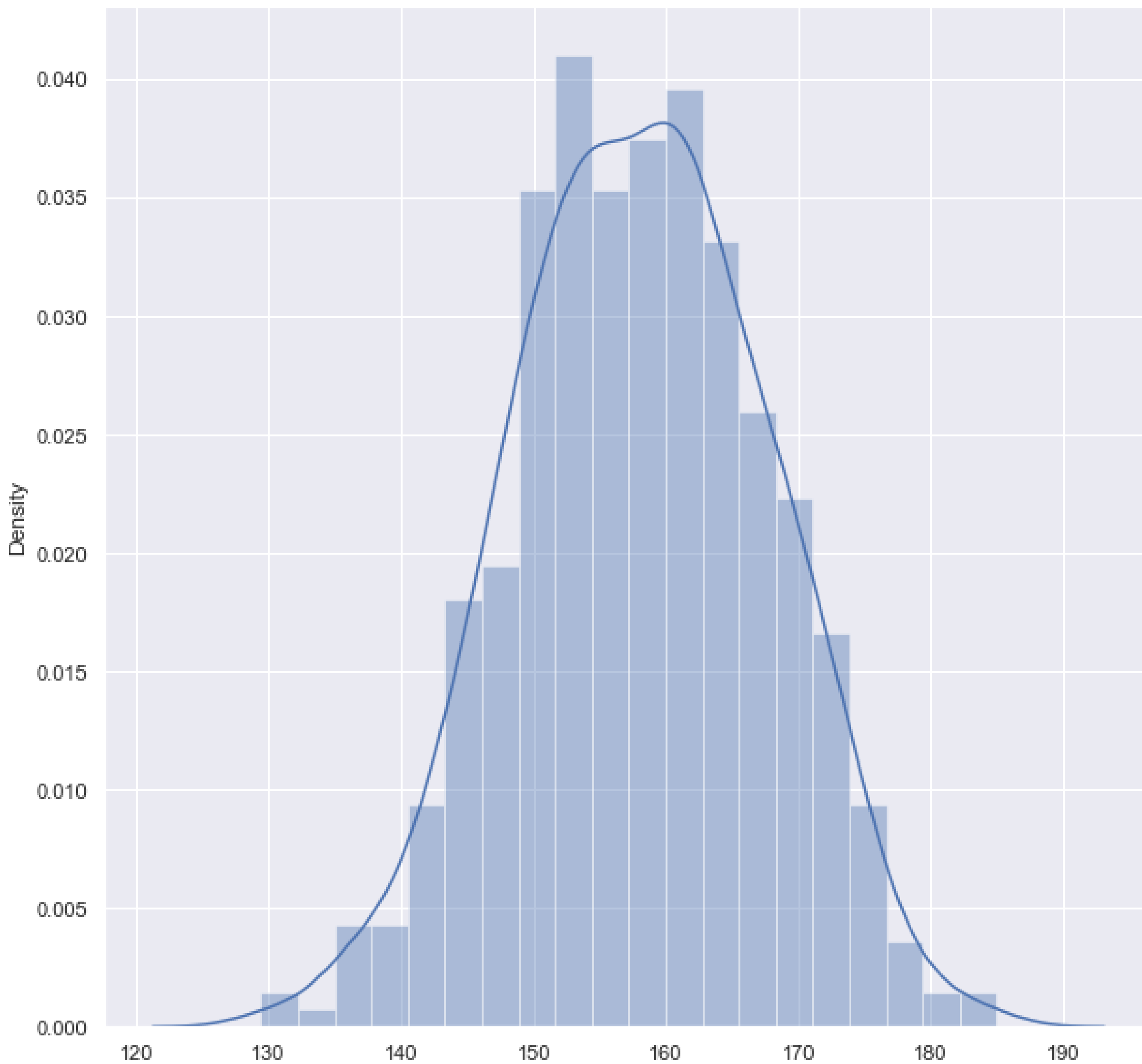
Inference: weather_1 is not following Normal distribution.

Lets try to do Bootstrapping and see if Sampling distribution follow Normal distribution or not. Irrespective of how your distribution is, if your smapling distribution follows a Normal distribution, we can approximate from CLT.

In [59]:
```
weather_1_samp = []
for i in range(500):
    w1 = df[df["weather"]==1]["count"].sample(n=200)
    av = w1.mean()
    weather_1_samp.append(av)
```

In [60]:
```
sns.distplot(weather_1_samp, bins = 20)
```

Out[60]: <AxesSubplot:ylabel='Density'>



Inference: This is following Normal Distribution.

In [61]:
```
np.mean(weather_1_samp)
```

Out[61]: 158.044

In [62]:
```
mu = (mean_1*500 + mean_2*500 + mean_3*500)/1500
mu
```

Out[62]: 132.81066666666666

In [63]:
```
ssw = sum((weather_1-mean_1)**2 + (weather_2-mean_2)**2 + (weather_3-mean_3)**2)
ssw
```

Out[63]: 22567115.20399998

In [64]:
```
ssg = ((mean_1 - mu)**2)*500 + ((mean_2 - mu)**2)*500 + ((mean_3 - mu)**2)*500
ssg
```

Out[64]: 967503.0253333332

In [65]:
```
msg = ssg/2
msg
```

Out[65]: 483751.5126666666

In [66]:
```
msw = ssw/(1500-3)
msw
```

Out[66]: 15074.89325584501

In [67]:
```
fstat = msg/msw
fstat
```

Out[67]: 32.08987980588857

Then to get Fcritical values look at the F-distribution table.

F - Distribution (α = 0.01 in the Right Tail)

df_2 \ df_1	1	2	3	4	5	6	7	8	9
1	4052.2	4999.5	5403.4	5624.6	5763.6	5859.0	5928.4	5981.1	6022.5
2	98.503	99.000	99.166	99.249	99.299	99.333	99.356	99.374	99.388
3	34.116	30.817	29.457	28.710	28.237	27.911	27.672	27.489	27.345
4	21.198	18.000	16.694	15.977	15.522	15.207	14.976	14.799	14.659
5	16.258	13.274	12.060	11.392	10.967	10.672	10.456	10.289	10.158
6	13.745	10.925	9.7795	9.1483	8.7459	8.4661	8.2600	8.1017	7.9761
7	12.246	9.5466	8.4513	7.8466	7.4604	7.1914	6.9928	6.8400	6.7188
8	11.259	8.6491	7.5910	7.0061	6.6318	6.3707	6.1776	6.0289	5.9106
9	10.561	8.0215	6.9919	6.4221	6.0569	5.8018	5.6129	5.4671	5.3511
10	10.044	7.5594	6.5523	5.9943	5.6363	5.3858	5.2001	5.0567	4.9424
11	9.6460	7.2057	6.2167	5.6683	5.3160	5.0692	4.8861	4.7445	4.6315
12	9.3302	6.9266	5.9525	5.4120	5.0643	4.8206	4.6395	4.4994	4.3875
13	9.0738	6.7010	5.7394	5.2053	4.8616	4.6204	4.4410	4.3021	4.1911
14	8.8616	6.5149	5.5639	5.0354	4.6950	4.4558	4.2779	4.1399	4.0297
15	8.6831	6.3589	5.4170	4.8932	4.5556	4.3183	4.1415	4.0045	3.8948
16	8.5310	6.2262	5.2922	4.7726	4.4374	4.2016	4.0259	3.8896	3.7804
17	8.3997	6.1121	5.1850	4.6690	4.3359	4.1015	3.9267	3.7910	3.6822
18	8.2854	6.0129	5.0919	4.5790	4.2479	4.0146	3.8406	3.7054	3.5971
19	8.1849	5.9259	5.0103	4.5003	4.1708	3.9386	3.7653	3.6305	3.5225
20	8.0960	5.8489	4.9382	4.4307	4.1027	3.8714	3.6987	3.5644	3.4567
21	8.0166	5.7804	4.8740	4.3688	4.0421	3.8117	3.6396	3.5056	3.3981
22	7.9454	5.7190	4.8166	4.3134	3.9880	3.7583	3.5867	3.4530	3.3458
23	7.8811	5.6637	4.7649	4.2636	3.9392	3.7102	3.5390	3.4057	3.2986
24	7.8229	5.6136	4.7181	4.2184	3.8951	3.6667	3.4959	3.3629	3.2560
25	7.7698	5.5680	4.6755	4.1774	3.8550	3.6272	3.4568	3.3239	3.2172
26	7.7213	5.5263	4.6366	4.1400	3.8183	3.5911	3.4210	3.2884	3.1818
27	7.6767	5.4881	4.6009	4.1056	3.7848	3.5580	3.3882	3.2558	3.1494
28	7.6356	5.4529	4.5681	4.0740	3.7539	3.5276	3.3581	3.2259	3.1195
29	7.5977	5.4204	4.5378	4.0449	3.7254	3.4995	3.3303	3.1982	3.0920
30	7.5625	5.3903	4.5097	4.0179	3.6990	3.4735	3.3045	3.1726	3.0665
40	7.3141	5.1785	4.3126	3.8283	3.5138	3.2910	3.1238	2.9930	2.8876
60	7.0771	4.9774	4.1259	3.6490	3.3389	3.1187	2.9530	2.8233	2.7185
120	6.8509	4.7865	3.9491	3.4795	3.1735	2.9559	2.7918	2.6629	2.5586
∞	6.6349	4.6052	3.7816	3.3192	3.0173	2.8020	2.6393	2.5113	2.4073

Numerator Degrees of Freedom (df_1); Denominator Degrees of Freedom (df_2)

fcrit~=3.0

Since fcrit<<45.198, we reject ho and accept ha.

2.b.2. season

ho: count is not same in different seasons
h1: count is same in different seasons

In [68]:
```
df.season.value_counts()
```

Out[68]:
```
4    2475
1    2463
2    2292
3    2288
Name: season, dtype: int64
```

In [69]:
```
df.groupby('season').agg({'count':'sum'}).reset_index(drop=True)
```

Out[69]:

	count
0	254093
1	367547
2	405323
3	402032

In [70]:
```
season_1 = df[df["season"]==1]["count"].sample(n=500)
season_2 = df[df["season"]==2]["count"].sample(n=500)
season_3 = df[df["season"]==3]["count"].sample(n=500)
season_4 = df[df["season"]==3]["count"].sample(n=500)

season_1 = season_1.reset_index(drop=True)
season_2 = season_2.reset_index(drop=True)
season_3 = season_3.reset_index(drop=True)
season_4 = season_4.reset_index(drop=True)

dataset = pd.DataFrame({"1":season_1, "2":season_2, "3":season_3, "4":season_4})
dataset
```

Out[70]:

	1	2	3	4
0	83	92	236	223
1	78	13	441	343
2	71	5	284	7
3	40	486	129	268
4	82	388	152	14
...	...	...	...	...
495	17	59	140	207
496	161	52	22	7
497	57	135	7	161
498	18	286	266	176
499	202	274	318	52

500 rows × 4 columns

In [71]:
```
f, pval = stats.f_oneway(dataset["1"], dataset["2"], dataset["3"], dataset["4"])
print(f, pval)
```

```
38.25960961101622 4.8977501313665634e-24
```

Inference: pvalue<0.05, that means we reject ho and accept ha

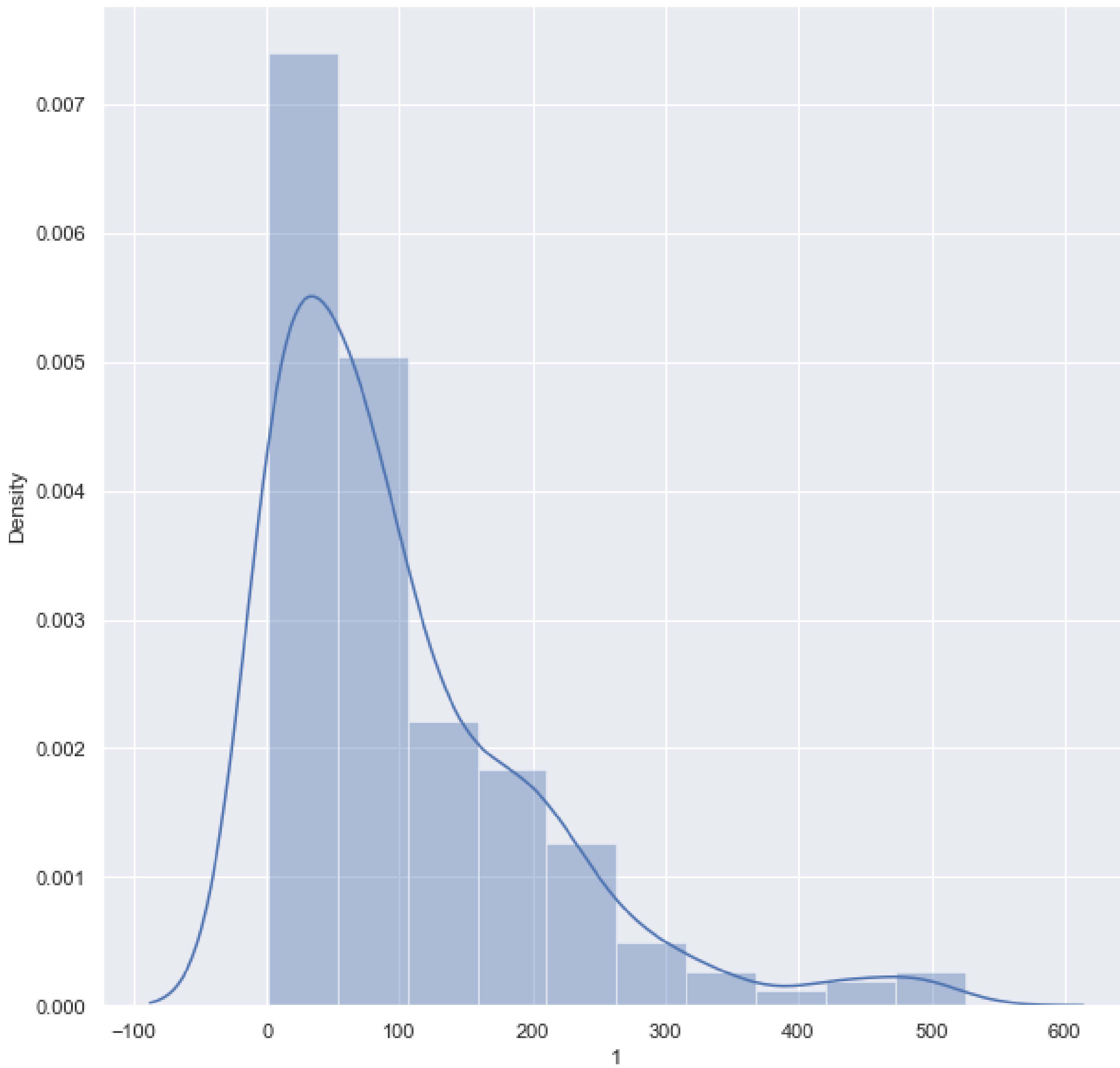
In [72]:
```
mean_1 = dataset['1'].mean()
mean_2 = dataset['2'].mean()
mean_3 = dataset['3'].mean()
mean_4 = dataset['4'].mean()

print(mean_1, mean_2, mean_3, mean_4)
```

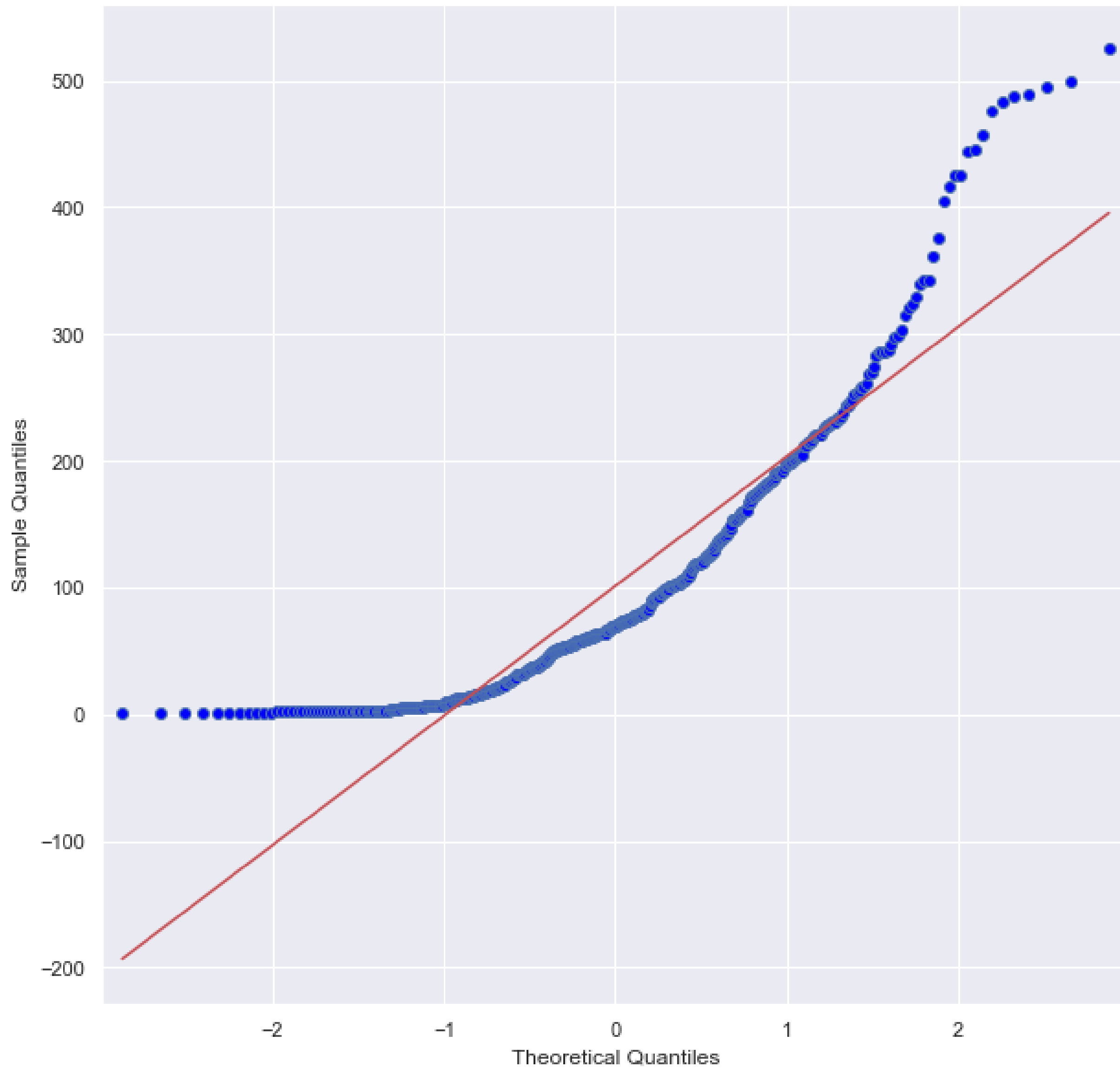
```
101.176 161.15 180.278 173.446
```

In [73]:
```
sns.distplot(dataset['1'], bins = 10)
```

Out[73]:
```
<AxesSubplot:xlabel='1', ylabel='Density'>
```



In [74]:
```
sm.qqplot(dataset['1'], line = "s")
py.show()
```



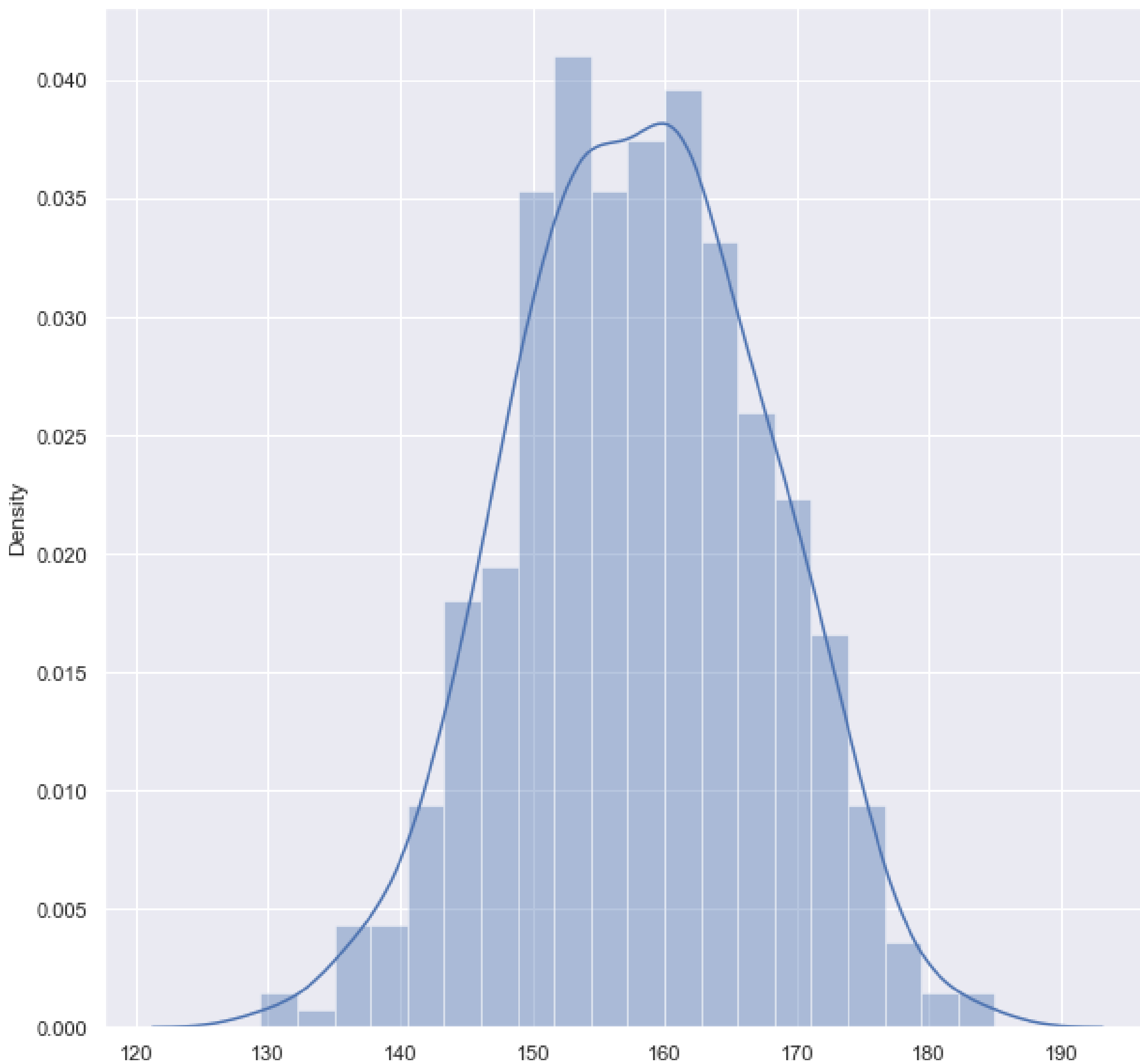
Inference: season_1 is not following Normal distribution.

Lets try to do Bootstrapping and see if Sampling distribution follow Normal distribution or not. Irrespective of how your distribution is, if your smapling distribution follows a Normal distribution, we can approximate from CLT.

In [75]:
```
season_1_samp = []
for i in range(500):
    s1 = df[df["season"]==1]["count"].sample(n=200)
    av = s1.mean()
    season_1_samp.append(av)
```

In [76]:
```
sns.distplot(weather_1_samp, bins = 20)
```

Out[76]: <AxesSubplot:ylabel='Density'>



Inference: This is following Normal distribution

In [77]:
```
np.mean(weather_1_samp)
```

Out[77]: 158.044

In [78]:
```
mu = (mean_1*500 + mean_2*500 + mean_3*500 + mean_4*500)/2000
mu
```

Out[78]: 154.0125

In [79]:
```
ssw = sum((season_1-mean_1)**2 + (season_2-mean_2)**2 + (season_3-mean_3)**2 + (s
ssw
```

Out[79]: 33998924.162000015

In [80]:
```
ssg = ((mean_1 - mu)**2)*500 + ((mean_2 - mu)**2)*500 + ((mean_3 - mu)**2)*500 +
ssg
```

Out[80]: 1955088.5254999998

In [81]:
```
msg = ssg/3
msg
```

Out[81]: 651696.1751666666

In [82]:
```
msw = ssw/(2000-4)
msw
```

Out[82]: 17033.529139278566

In [83]:
```
fstat = msg/msw
fstat
```

Out[83]: 38.259609611016195

Then to get Fcritical values look at the F-distribution table.

F - Distribution (α = 0.01 in the Right Tail)

Denominator Degrees of Freedom df_2 \ df_1 Numerator Degrees of Freedom	1	2	3	4	5	6	7	8	9
1	4052.2	4999.5	5403.4	5624.6	5763.6	5859.0	5928.4	5981.1	6022.5
2	98.503	99.000	99.166	99.249	99.299	99.333	99.356	99.374	99.388
3	34.116	30.817	29.457	28.710	28.237	27.911	27.672	27.489	27.345
4	21.198	18.000	16.694	15.977	15.522	15.207	14.976	14.799	14.659
5	16.258	13.274	12.060	11.392	10.967	10.672	10.456	10.289	10.158
6	13.745	10.925	9.7795	9.1483	8.7459	8.4661	8.2600	8.1017	7.9761
7	12.246	9.5466	8.4513	7.8466	7.4604	7.1914	6.9928	6.8400	6.7188
8	11.259	8.6491	7.5910	7.0061	6.6318	6.3707	6.1776	6.0289	5.9106
9	10.561	8.0215	6.9919	6.4221	6.0569	5.8018	5.6129	5.4671	5.3511
10	10.044	7.5594	6.5523	5.9943	5.6363	5.3858	5.2001	5.0567	4.9424
11	9.6460	7.2057	6.2167	5.6683	5.3160	5.0692	4.8861	4.7445	4.6315
12	9.3302	6.9266	5.9525	5.4120	5.0643	4.8206	4.6395	4.4994	4.3875
13	9.0738	6.7010	5.7394	5.2053	4.8616	4.6204	4.4410	4.3021	4.1911
14	8.8616	6.5149	5.5639	5.0354	4.6950	4.4558	4.2779	4.1399	4.0297
15	8.6831	6.3589	5.4170	4.8932	4.5556	4.3183	4.1415	4.0045	3.8948
16	8.5310	6.2262	5.2922	4.7726	4.4374	4.2016	4.0259	3.8896	3.7804
17	8.3997	6.1121	5.1850	4.6690	4.3359	4.1015	3.9267	3.7910	3.6822
18	8.2854	6.0129	5.0919	4.5790	4.2479	4.0146	3.8406	3.7054	3.5971
19	8.1849	5.9259	5.0103	4.5003	4.1708	3.9386	3.7653	3.6305	3.5225
20	8.0960	5.8489	4.9382	4.4307	4.1027	3.8714	3.6987	3.5644	3.4567
21	8.0166	5.7804	4.8740	4.3688	4.0421	3.8117	3.6396	3.5056	3.3981
22	7.9454	5.7190	4.8166	4.3134	3.9880	3.7583	3.5867	3.4530	3.3458
23	7.8811	5.6637	4.7649	4.2636	3.9392	3.7102	3.5390	3.4057	3.2986
24	7.8229	5.6136	4.7181	4.2184	3.8951	3.6667	3.4959	3.3629	3.2560
25	7.7698	5.5680	4.6755	4.1774	3.8550	3.6272	3.4568	3.3239	3.2172
26	7.7213	5.5263	4.6366	4.1400	3.8183	3.5911	3.4210	3.2884	3.1818
27	7.6767	5.4881	4.6009	4.1056	3.7848	3.5580	3.3882	3.2558	3.1494
28	7.6356	5.4529	4.5681	4.0740	3.7539	3.5276	3.3581	3.2259	3.1195
29	7.5977	5.4204	4.5378	4.0449	3.7254	3.4995	3.3303	3.1982	3.0920
30	7.5625	5.3903	4.5097	4.0179	3.6990	3.4735	3.3045	3.1726	3.0665
40	7.3141	5.1785	4.3126	3.8283	3.5138	3.2910	3.1238	2.9930	2.8876
60	7.0771	4.9774	4.1259	3.6490	3.3389	3.1187	2.9530	2.8233	2.7185
120	6.8509	4.7865	3.9491	3.4795	3.1735	2.9559	2.7918	2.6629	2.5586
∞	6.6349	4.6052	3.7816	3.3192	3.0173	2.8020	2.6393	2.5113	2.4073

fcrit~=2.609

Since fcrit<<33.58, we reject ho and accept ha.

Inference:

1. count is different in different weather.
2. count is same in different seasons.

2.c. Chi-square test to check if Weather is dependent on the season (10 points)

Hypothesis Testing:
ho: There is no relationship btw weather and season.
ha: There is relationship btw weather and season.

In [84]:
```python
ctab = pd.crosstab(df['weather'], df['season'])
ctab
```

Out[84]:

season	1	2	3	4
weather				
1	1595	1473	1598	1510
2	683	614	517	754
3	184	205	173	211
4	1	0	0	0

In [85]:
```python
from scipy.stats import chi2_contingency
```

In [86]:
```python
stat, p, dof, expected = chi2_contingency(ctab)
```

In [87]:
```python
print("Chi-square stat:",stat)
print("pvalue :",p)
```

```
Chi-square stat: 49.956607077688595
pvalue : 1.0976664201931212e-07
```

dof = (r-1)*(c-1) = 9

alpha = 0.05

chi-stats = 49.95

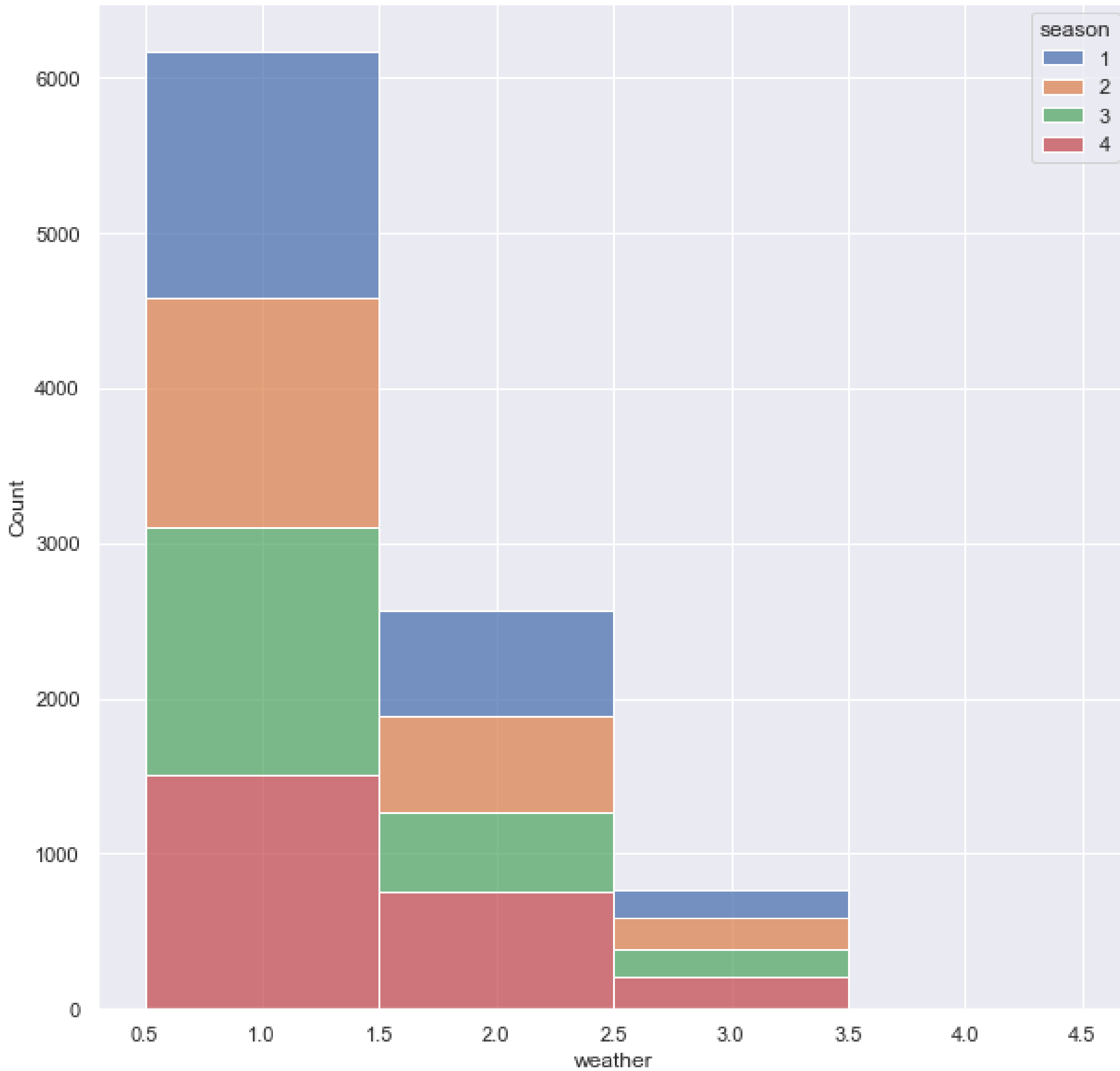
See the Chi-square distribution table to know the chi-crit.
chi-crit = 16.919

chi-crit < chi-stats, so we reject ho and accepts ha.

Therefore, there is relationship btw weather and season ie weather and season are dependent on each other.

In [88]:
```
sns.histplot(binwidth=0.5, x="weather", hue="season", data=df, stat="count", mult
```

Out[88]: <AxesSubplot:xlabel='weather', ylabel='Count'>



In [ ]: