

Data wrangling is the process of converting and formatting data from its raw form to usable format.in order to analyse it.

without proper data wrangling,we cannot perform analysis and model building. A good data wrangling ensures a better accuracy of the model/algoritm

Problem Statement:

Lyft, Inc. is a transportation network company based in San Francisco, California and operating in 640 cities in the United States and 9 cities in Canada. It develops, markets, and operates the Lyft mobile app, offering car rides, scooters, and a bicycle-sharing system. It is the second largest rideshare company in the world, second to only Uber.

Lyft's bike-sharing service is also among the largest in the USA. Being able to anticipate demand is extremely important for planning of bicycles, stations, and the personnel required to maintain these. This demand is sensitive to a lot of factors like season, humidity, rain, weekdays, holidays, and more. To enable this planning, Lyft needs to rightly predict the demand according to these factors.

Data Dictionary

Attribute Information:

date = date of the ride

season - 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday - whether the day is considered a holiday

workingday - whether the day is neither a weekend nor holiday

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

temp - temperature in Celsius

atemp - "feels like" temperature in Celsius

humidity - relative humidity

windspeed - wind speed

casual - number of non-registered user rentals initiated

registered - number of registered user rentals initiated

count - number of total rentals

You are a data scientist and you are assigned a task of performing data wrangling on a set of datasets. These datasets may have ambiguities. You have to identify these ambiguities, and apply different data wrangling techniques to get the datasets ready for usage

```
In [1]: import numpy as np
import pandas as pd
```

```
In [2]: #Loading the data into a dataframe:
data1=pd.read_csv('dataset_1.csv')
data1.head()
```

```
Out[2]:    instant      dteday  season  yr  mnth  hr  holiday  weekday  weathersit  temp
          0   1 01-01-2011    1  0     1  0    False       6        1  0.24
          1   2 01-01-2011    1  0     1  1    False       6        1  0.22
          2   3 01-01-2011    1  0     1  2    False       6        1  0.22
          3   4 01-01-2011    1  0     1  3    False       6        1  0.24
          4   5 01-01-2011    1  0     1  4    False       6        1  0.24
```

```
In [3]: data1.tail()
```

```
Out[3]:    instant      dteday  season  yr  mnth  hr  holiday  weekday  weathersit  temp
          605   606 28-01-2011    1  0     1  11   False       5        3  0.18
          606   607 28-01-2011    1  0     1  12   False       5        3  0.18
          607   608 28-01-2011    1  0     1  13   False       5        3  0.18
          608   609 28-01-2011    1  0     1  14   False       5        3  0.22
          609   610 28-01-2011    1  0     1  15   False       5        2  0.20
```

```
In [4]: data1.shape
```

```
Out[4]: (610, 10)
```

```
In [5]: #to check only column names
data1.columns
```

```
Out[5]: Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday',
       'weathersit', 'temp'],
       dtype='object')
```

```
In [6]: data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 610 entries, 0 to 609
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   instant     610 non-null    int64  
 1   dteday      610 non-null    object  
 2   season      610 non-null    int64  
 3   yr          610 non-null    int64  
 4   mnth        610 non-null    int64  
 5   hr          610 non-null    int64  
 6   holiday     610 non-null    bool   
 7   weekday     610 non-null    int64  
 8   weathersit  610 non-null    int64  
 9   temp         610 non-null    float64 
dtypes: bool(1), float64(1), int64(7), object(1)
memory usage: 43.6+ KB
```

```
In [7]: data1.isnull().sum()
```

```
out[7]: instant      0
dteday       0
season       0
yr          0
mnth        0
hr          0
holiday     0
weekday     0
weathersit  0
temp         0
dtype: int64
```

```
In [8]: data2=pd.read_excel('dataset_2.xlsx')
data2.head()
```

```
Out[8]:    Unnamed: 0 instant atemp hum windspeed casual registered cnt
      0         0      1  0.2879  0.81        0.0     3     13    16
      1         1      2  0.2727  0.80        0.0     8     32    40
      2         2      3  0.2727  0.80        0.0     5     27    32
      3         3      4  0.2879  0.75        0.0     3     10    13
      4         4      5  0.2879  0.75        0.0     0      1     1
```

```
In [9]: data2.tail()
```

```
Out[9]:    Unnamed: 0 instant atemp hum windspeed casual registered cnt
      605       605     606  0.2121  0.93      0.1045     0     30    30
      606       606     607  0.2121  0.93      0.1045     1     28    29
      607       607     608  0.2121  0.93      0.1045     0     31    31
      608       608     609  0.2727  0.80      0.0000     2     36    38
      609       609     610  0.2576  0.86      0.0000     1     40    41
```

```
In [10]: data2.columns
```

```
Out[10]: Index(['Unnamed: 0', 'instant', 'atemp', 'hum', 'windspeed', 'casual',
                'registered', 'cnt'],
               dtype='object')
```

```
In [11]: data2.drop(['Unnamed: 0'], axis=1, inplace=True)
```

```
In [12]: data2.columns
```

```
Out[12]: Index(['instant', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt'], dtype='object')
```

```
In [13]: data2.shape
```

```
Out[13]: (610, 7)
```

```
In [14]: ##combining both the files and storing it into another dataframe:
```

```
combined_data=pd.merge(data1,data2,on='instant')
combined_data.head()
```

```
Out[14]:
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	01-01-2011	1	0	1	0	False	6	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	01-01-2011	1	0	1	1	False	6	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	01-01-2011	1	0	1	2	False	6	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	01-01-2011	1	0	1	3	False	6	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	01-01-2011	1	0	1	4	False	6	1	0.24	0.2879	0.75	0.0	0	1	1

```
In [15]: combined_data.shape
```

```
Out[15]: (610, 16)
```

```
In [16]: data3=pd.read_csv('dataset_3.csv')
data3.head()
```

```
Out[16]:
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	620	29-01-2011	1	0	1	1	False	6	1	0.22	0.2273	0.64	0.1940	0	20	20
1	621	29-01-2011	1	0	1	2	False	6	1	0.22	0.2273	0.64	0.1642	0	15	15
2	622	29-01-2011	1	0	1	3	False	6	1	0.20	0.2121	0.64	0.1343	3	5	8
3	623	29-01-2011	1	0	1	4	False	6	1	0.16	0.1818	0.69	0.1045	1	2	3
4	624	29-01-2011	1	0	1	6	False	6	1	0.16	0.1818	0.64	0.1343	0	2	2

```
In [17]: data3.tail()
```

Out[17]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
385	615	28-01-2011	1	0	1	20	False	5	2	0.24	0.2273	0.70	0.1940	1	61	62
386	616	28-01-2011	1	0	1	21	False	5	2	0.22	0.2273	0.75	0.1343	1	57	58
387	617	28-01-2011	1	0	1	22	False	5	1	0.24	0.2121	0.65	0.3582	0	26	26
388	618	28-01-2011	1	0	1	23	False	5	1	0.24	0.2273	0.60	0.2239	1	22	23
389	619	29-01-2011	1	0	1	0	False	6	1	0.22	0.1970	0.64	0.3582	2	26	28

In [18]: `data3=data3.sort_values(by='instant')`

In [19]: `data3.head()`

Out[19]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
381	611	28-01-2011	1	0	1	16	False	5	1	0.22	0.2727	0.80	0.0000	10	70	80
382	612	28-01-2011	1	0	1	17	False	5	1	0.24	0.2424	0.75	0.1343	2	147	149
383	613	28-01-2011	1	0	1	18	False	5	1	0.24	0.2273	0.75	0.1940	2	107	109
384	614	28-01-2011	1	0	1	19	False	5	2	0.24	0.2424	0.75	0.1343	5	84	89
385	615	28-01-2011	1	0	1	20	False	5	2	0.24	0.2273	0.70	0.1940	1	61	62

In [20]: `data3.tail()`

Out[20]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
376	996	14-02-2011	1	0	2	3	False	1	1	0.34	0.3182	0.46	0.2239	1	1	2
377	997	14-02-2011	1	0	2	4	False	1	1	0.32	0.3030	0.53	0.2836	0	2	2
378	998	14-02-2011	1	0	2	5	False	1	1	0.32	0.3030	0.53	0.2836	0	3	3
379	999	14-02-2011	1	0	2	6	False	1	1	0.34	0.3030	0.46	0.2985	1	25	26
380	1000	14-02-2011	1	0	2	7	False	1	1	0.34	0.3030	0.46	0.2985	2	96	98

In [21]: `data3.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 390 entries, 381 to 380
Data columns (total 16 columns):
 #   Column      Non-Null Count Dtype  
--- 
 0   instant     390 non-null    int64  
 1   dteday      390 non-null    object  
 2   season      390 non-null    int64  
 3   yr          390 non-null    int64  
 4   mnth        390 non-null    int64  
 5   hr          390 non-null    int64  
 6   holiday     390 non-null    bool   
 7   weekday     390 non-null    int64  
 8   weathersit  390 non-null    int64  
 9   temp         390 non-null    float64 
 10  atemp       390 non-null    float64 
 11  hum          390 non-null    float64 
 12  windspeed   390 non-null    float64 
 13  casual       390 non-null    int64  
 14  registered  390 non-null    int64  
 15  cnt          390 non-null    int64  
dtypes: bool(1), float64(4), int64(10), object(1)
memory usage: 49.1+ KB
```

```
In [22]: data3.columns
```

```
Out[22]: Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday',
       'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual',
       'registered', 'cnt'],
      dtype='object')
```

```
In [23]: combined_data.columns==data3.columns
```

```
Out[23]: array([ True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True])
```

```
In [24]: #####check if the data has duplicate rows:
duplicates=data3[data3.duplicated()]
duplicates
```

```
Out[24]: instant  dteday  season  yr  mnth  hr  holiday  weekday  weathersit  temp  atemp  hum  windspeed  casual  registered  cnt
```

```
In [25]: combined_data.shape
```

```
Out[25]: (610, 16)
```

```
In [26]: data3.shape
```

```
Out[26]: (390, 16)
```

```
In [27]: final_data=pd.concat([combined_data,data3])
final_data
```

```
Out[27]:
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	01-01-2011	1	0	1	0	False	6	1	0.24	0.2879	0.81	0.0000	3	13	16
1	2	01-01-2011	1	0	1	1	False	6	1	0.22	0.2727	0.80	0.0000	8	32	40
2	3	01-01-2011	1	0	1	2	False	6	1	0.22	0.2727	0.80	0.0000	5	27	32
3	4	01-01-2011	1	0	1	3	False	6	1	0.24	0.2879	0.75	0.0000	3	10	13
4	5	01-01-2011	1	0	1	4	False	6	1	0.24	0.2879	0.75	0.0000	0	1	1
...	
376	996	14-02-2011	1	0	2	3	False	1	1	0.34	0.3182	0.46	0.2239	1	1	2
377	997	14-02-2011	1	0	2	4	False	1	1	0.32	0.3030	0.53	0.2836	0	2	2
378	998	14-02-2011	1	0	2	5	False	1	1	0.32	0.3030	0.53	0.2836	0	3	3
379	999	14-02-2011	1	0	2	6	False	1	1	0.34	0.3030	0.46	0.2985	1	25	26
380	1000	14-02-2011	1	0	2	7	False	1	1	0.34	0.3030	0.46	0.2985	2	96	98

1000 rows × 16 columns

```
In [28]: final_data.shape
```

```
Out[28]: (1000, 16)
```

```
In [29]: final_data.isna().sum()
```

```
Out[29]: instant      0  
          dteday      0  
          season      0  
          yr          0  
          mnth       0  
          hr          0  
          holiday     0  
          weekday     0  
          weathersit   0  
          temp         0  
          atemp        11  
          hum          0  
          windspeed    0  
          casual       0  
          registered   0  
          cnt          0  
          dtype: int64
```

```
In [30]: ### if the missing value % is less 20% --->drop missing values  
### if the missing value % is between 20-70 % ---> impute missing values  
#### if missing value % is greater than 70%-----> drop the variable with missing values.
```

```
In [31]: 11/1000
```

```
Out[31]: 0.011
```

```
In [32]: #####as the missing value % is lesser than 20% we drop the rows with missing values.  
final_data.dropna(inplace=True)
```

```
In [33]: final_data.shape
```

```
Out[33]: (989, 16)
```

```
In [34]: #### check for duplicates:  
duplicate_rows=final_data[final_data.duplicated()]  
duplicate_rows
```

```
Out[34]: instant dteday season yr mnth hr holiday weekday weathersit temp atemp hum windspeed casual registered cnt
```

Sanity checks on data- checks the logical correctness of the data

```
In [35]: np.sum(final_data['casual']+final_data['registered']-final_data['cnt'])
```

```
Out[35]: 0
```

```
In [36]: ###rename the variable names to more logically relevant ones-:
final_data.rename(columns={'dteday':'Date','yr':'Year','mnth':'Month','hr':'Hour','weathersit':'Weather','temp':'Temperature',
'hum':'Humidity','cnt':'Count'},inplace=True)
```

```
In [37]: final_data.columns
```

```
Out[37]: Index(['instant', 'Date', 'season', 'Year', 'Month', 'Hour', 'holiday',
'weekday', 'Weather', 'Temperature', 'atemp', 'Humidity', 'windspeed',
'casual', 'registered', 'Count'],
dtype='object')
```

```
In [38]: np.sum(final_data['casual']+final_data['registered']-final_data['Count'])
```

```
Out[38]: 0
```

```
In [39]: ###checking if month and hour have discrepancies:
```

```
print(np.unique(final_data.Hour))
print(np.unique(final_data.Month))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[1 2]
```

```
In [40]: ###segregate the numeric and categorical variables:
```

```
numeric_variables=final_data.select_dtypes(include='number')
categorical_variables=final_data.select_dtypes(exclude='number')
```

```
In [41]: numeric_variables.columns
```

```
Out[41]: Index(['instant', 'season', 'Year', 'Month', 'Hour', 'weekday', 'Weather',
'Temperature', 'atemp', 'Humidity', 'windspeed', 'casual', 'registered',
'Count'],
dtype='object')
```

```
In [42]: categorical_variables.shape
```

```
Out[42]: (989, 2)
```

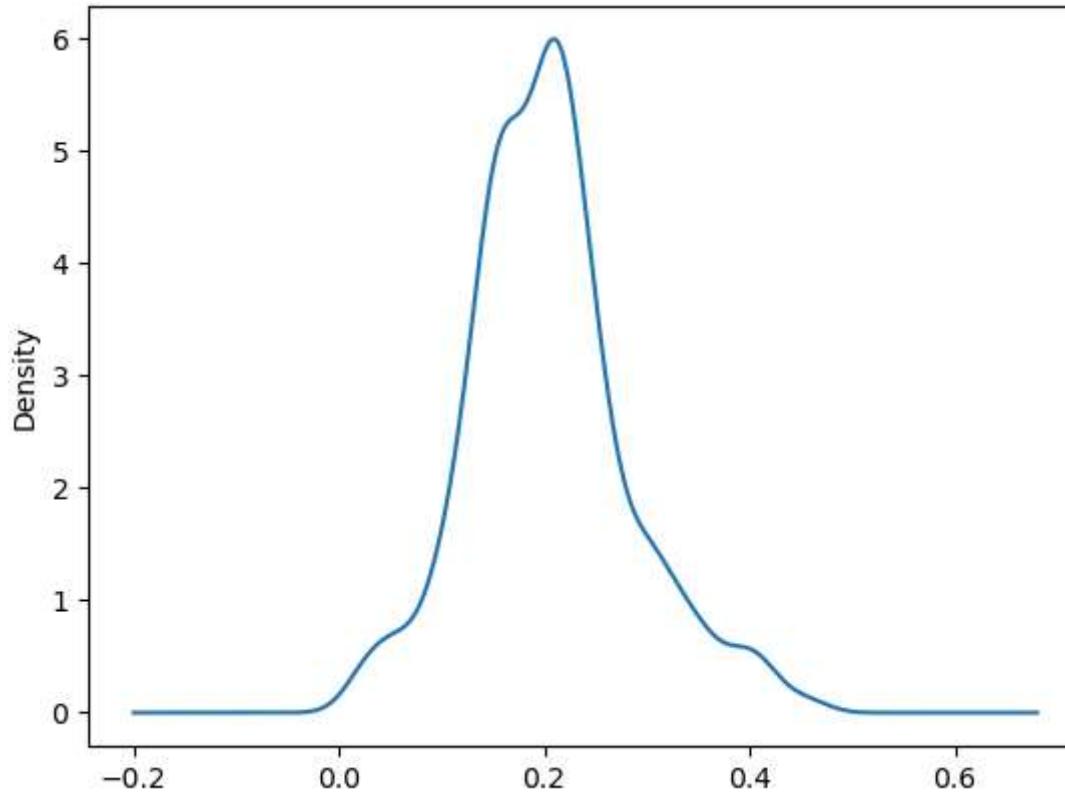
```
In [43]: categorical_variables.columns  
Out[43]: Index(['Date', 'holiday'], dtype='object')
```

```
In [44]: final_data.shape  
Out[44]: (989, 16)
```

when we join two data tables we perform the join based on row indexes, merging of two tables can be done based on a primary key(matching column or row) while concat can be used to stack dataframes either horizontally or vertically without any commonality.

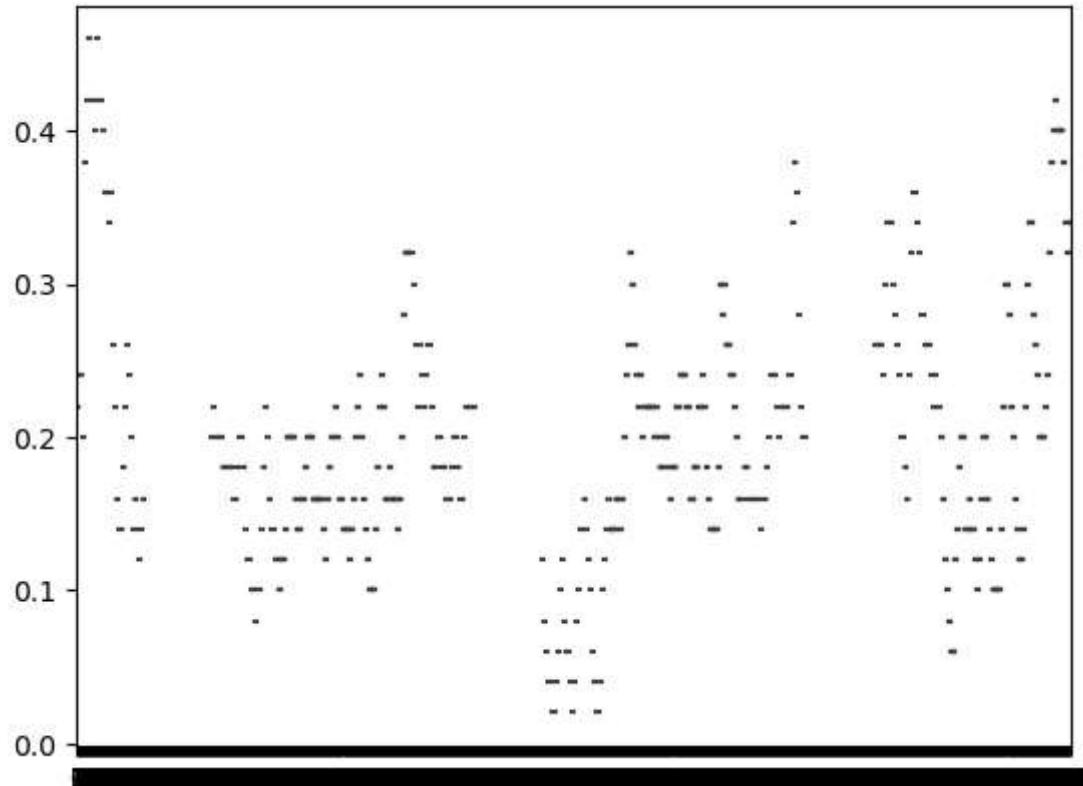
```
In [45]: from matplotlib import pyplot as plt  
import seaborn as sns
```

```
In [46]: final_data.Temperature.plot.density()  
Out[46]: <Axes: ylabel='Density'>
```



All continuous variables should follow a normal distribution.

```
In [47]: sns.boxplot(final_data.Temperature)  
plt.show()
```



```
In [48]: final_data.describe() #describe gives us the summary stats for all the numeric variables in the data
```

Out[48]:

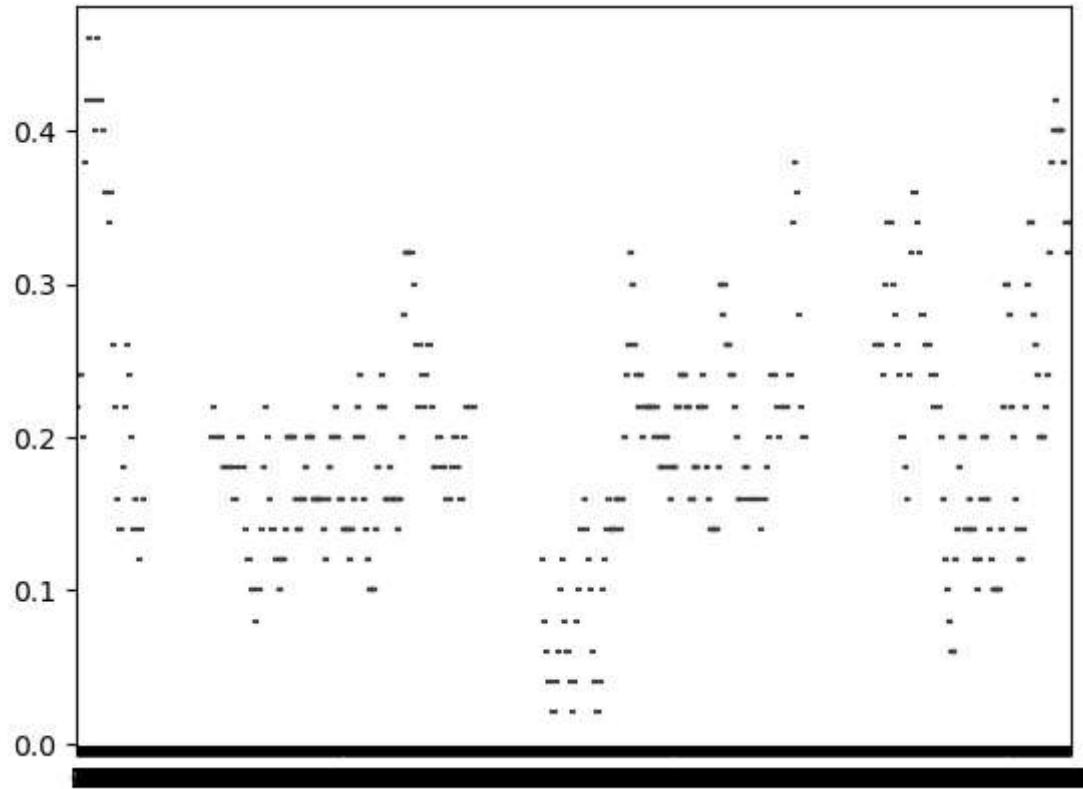
	instant	season	Year	Month	Hour	weekday	Weather	Temperature	atemp	Humidity	windspeed	casual	registered
count	989.000000	989.0	989.0	989.000000	989.000000	989.000000	989.000000	989.000000	989.000000	989.000000	989.000000	989.000000	989.000000
mean	505.622851	1.0	0.0	1.315470	11.753286	2.991911	1.479272	0.204712	0.211958	0.581769	0.194609	4.921132	53.689
std	286.274765	0.0	0.0	0.464938	6.891129	2.084727	0.651085	0.077789	0.076703	0.187706	0.129225	7.666231	48.019
min	1.000000	1.0	0.0	1.000000	0.000000	0.000000	1.000000	0.020000	0.000000	0.210000	0.000000	0.000000	0.000000
25%	259.000000	1.0	0.0	1.000000	6.000000	1.000000	1.000000	0.160000	0.166700	0.440000	0.104500	0.000000	15.000
50%	506.000000	1.0	0.0	1.000000	12.000000	3.000000	1.000000	0.200000	0.212100	0.550000	0.164200	3.000000	46.000
75%	753.000000	1.0	0.0	2.000000	18.000000	5.000000	2.000000	0.240000	0.257600	0.700000	0.283600	6.000000	75.000
max	1000.000000	1.0	0.0	2.000000	23.000000	6.000000	4.000000	0.460000	0.454500	1.000000	0.582100	62.000000	247.000

In [49]: *### box plots is a five number summary, which includes minimum, Q1, median, Q3, maximum.*

Interquartile range is Q3-Q1, represents 50% of the data

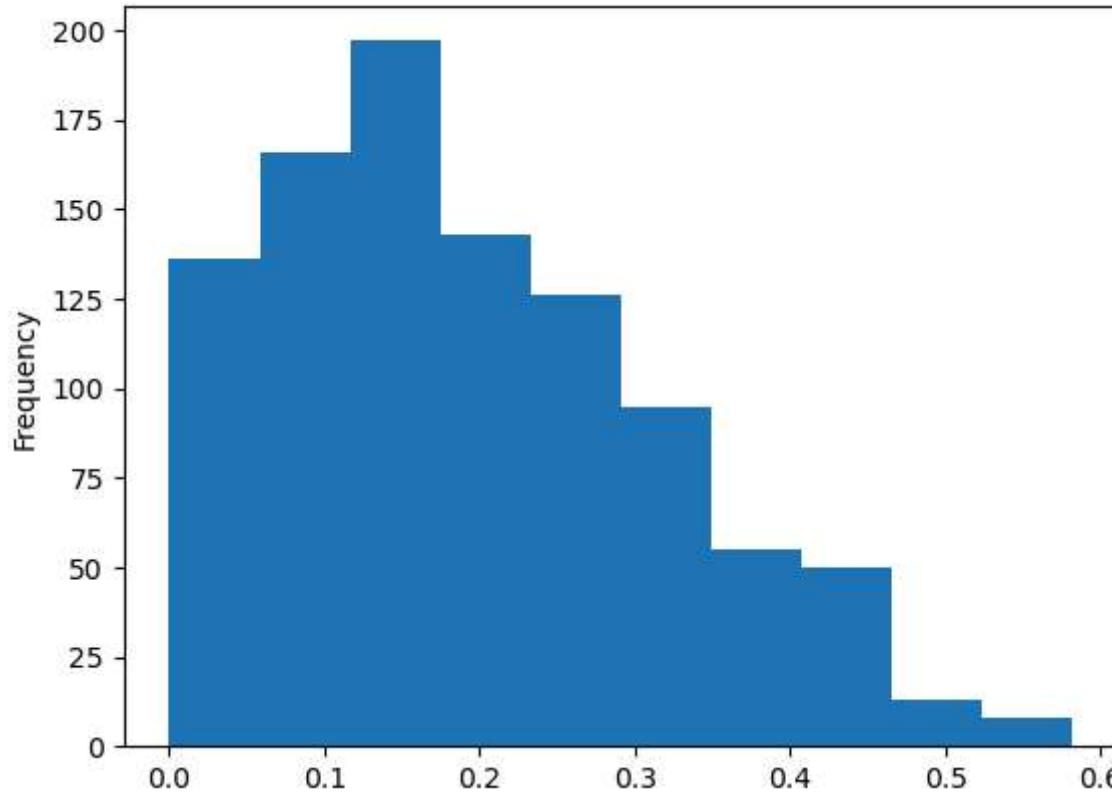
the values below Q1-1.5IQR and above Q3+1.5IQR are called the negative and positive outliers.

In [50]: `sns.boxplot(final_data.Temperature)
plt.show()`



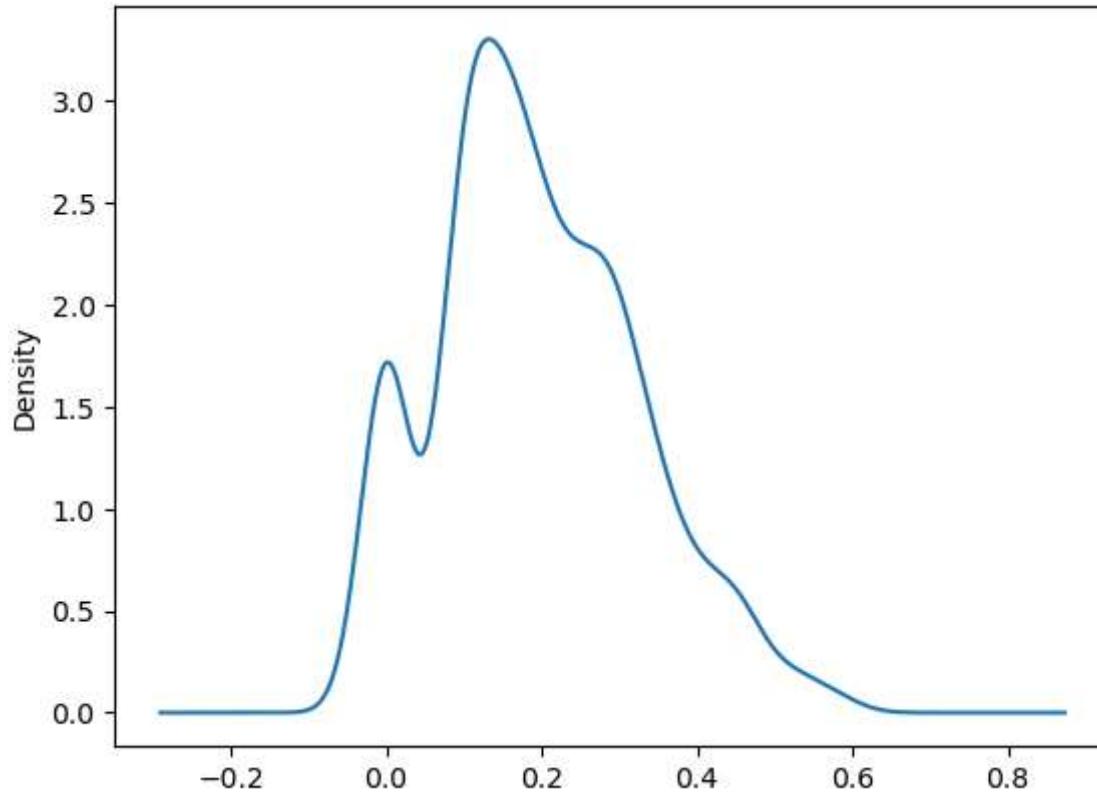
```
In [51]: #####check the distribution for windspeed.  
final_data.windspeed.plot.hist()
```

```
Out[51]: <Axes: ylabel='Frequency'>
```



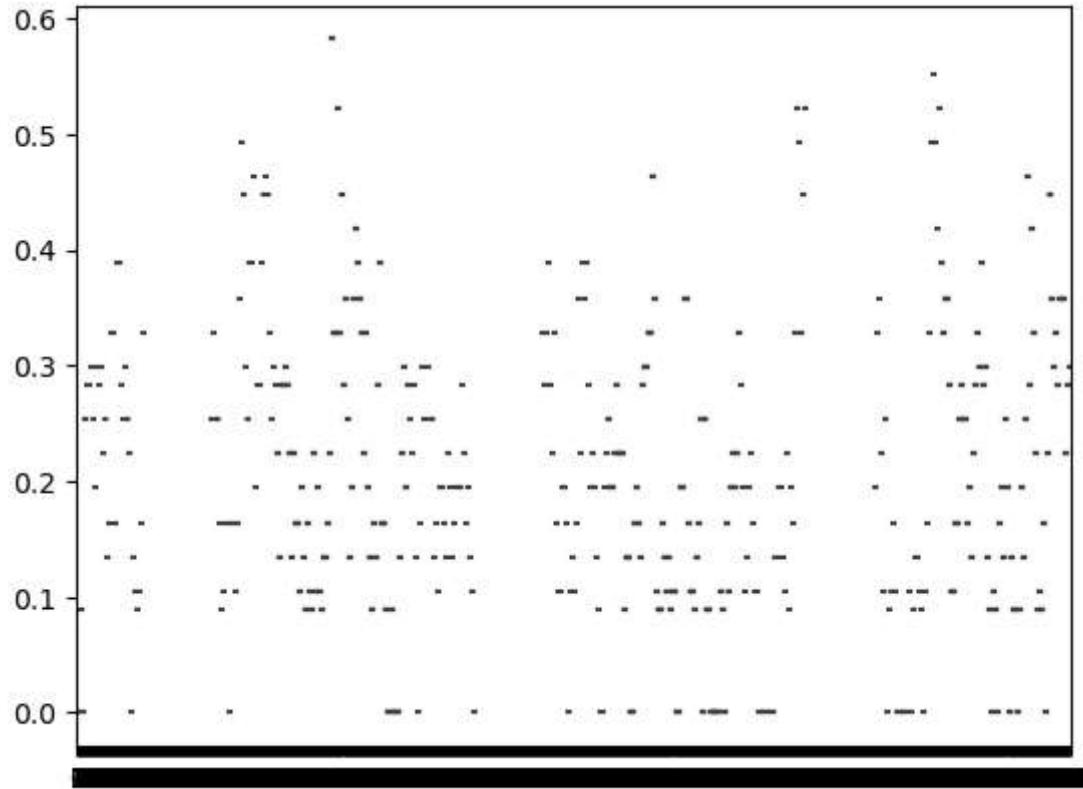
```
In [52]: final_data.windspeed.plot.density()
```

```
Out[52]: <Axes: ylabel='Density'>
```

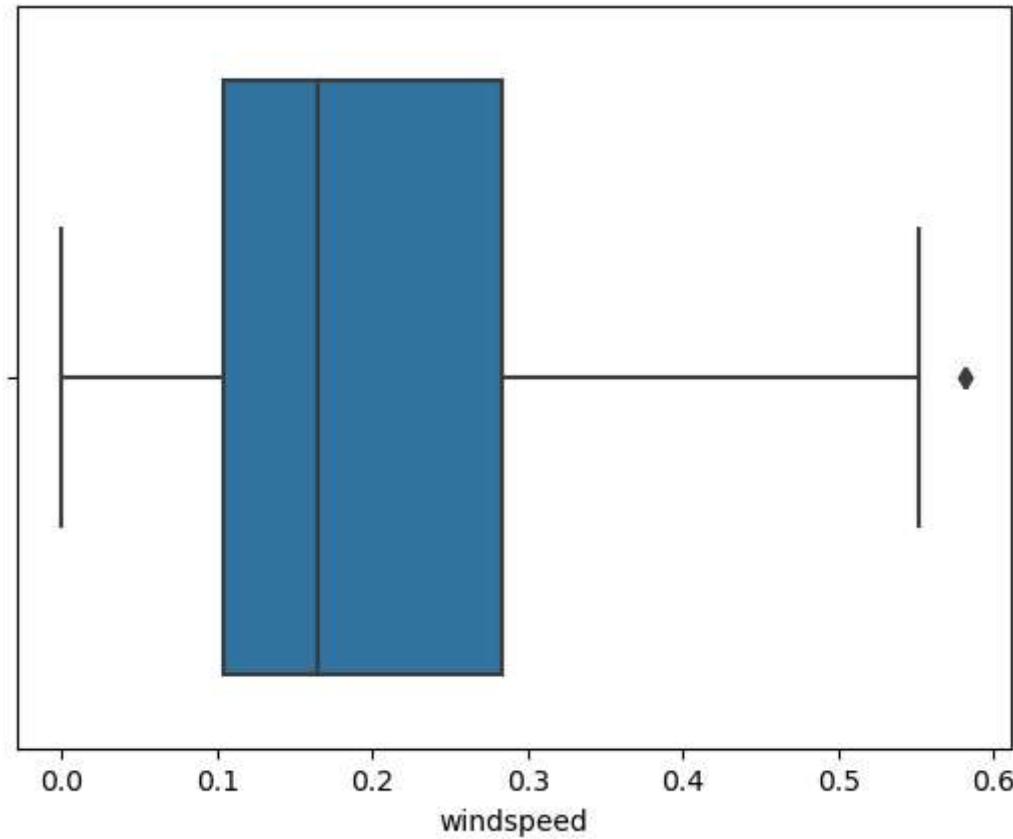


```
In [53]: sns.boxplot(final_data.windspeed)
```

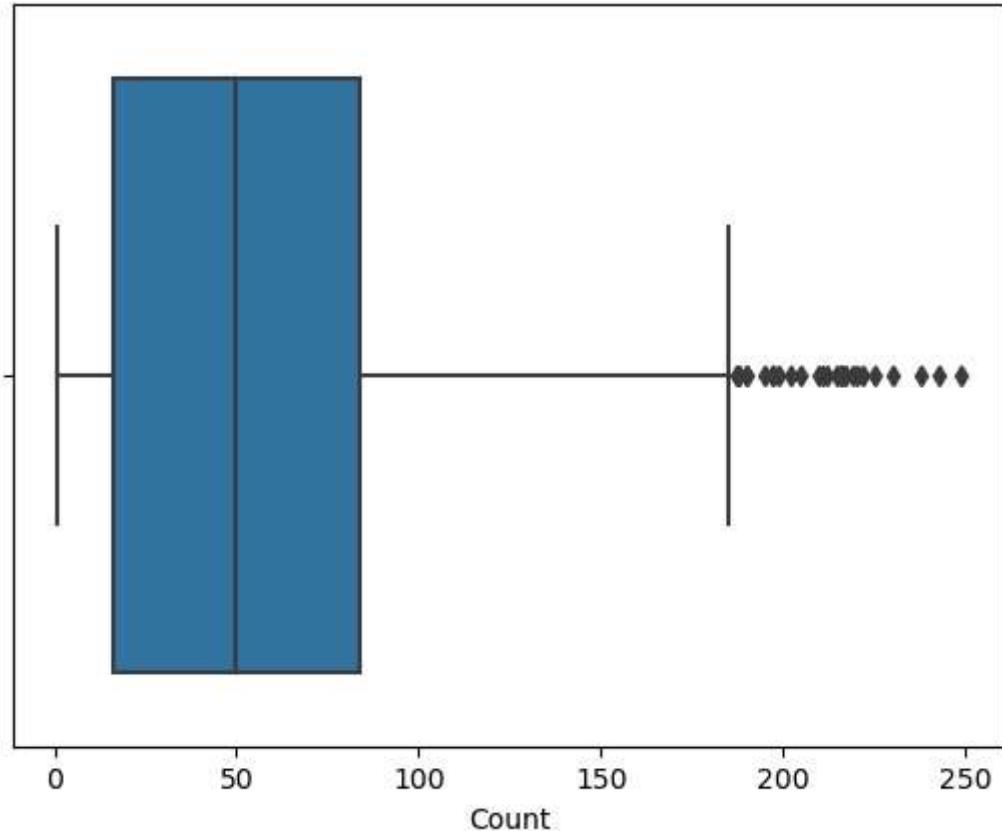
```
Out[53]: <Axes: >
```



```
In [54]: sns.boxplot(x=final_data['windspeed'])
plt.show()
```



```
In [55]: sns.boxplot(x=final_data['Count'])
plt.show()
```



```
In [56]: final_data['Count'].quantile([0.25,0.50,0.75,0.90,0.95,0.99])
```

```
Out[56]:
```

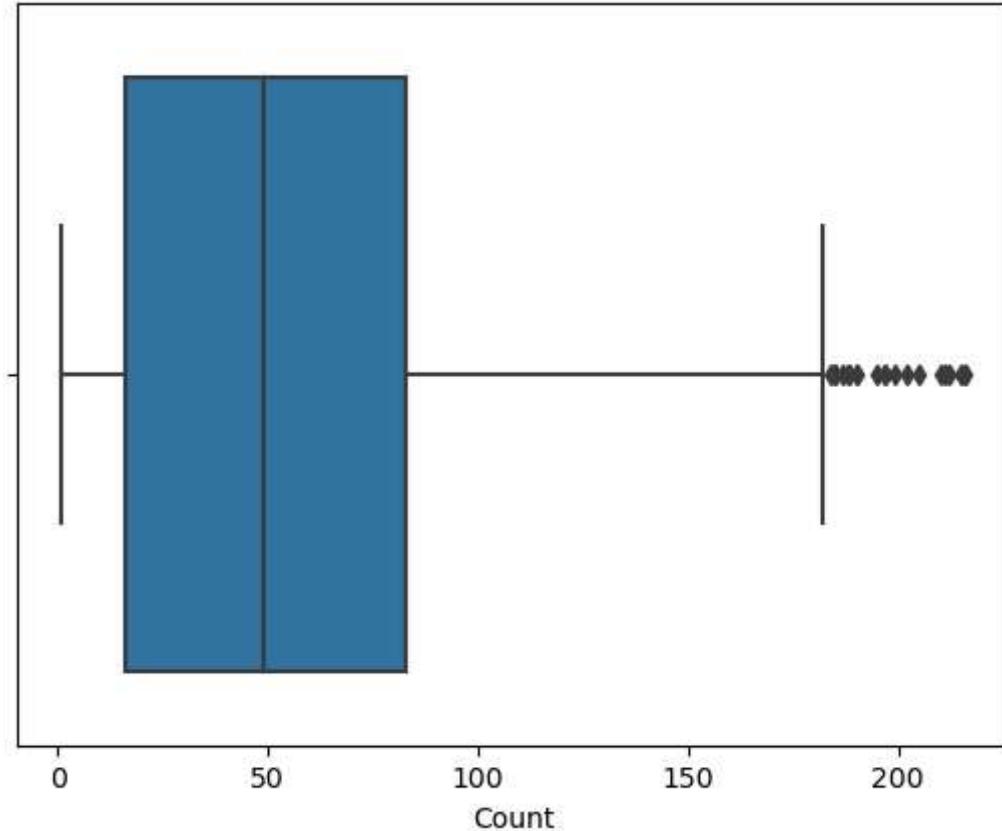
0.25	16.00
0.50	50.00
0.75	84.00
0.90	130.40
0.95	166.00
0.99	216.12

Name: Count, dtype: float64

```
In [57]: #####216.12 is the 99th percentile,only 1% of the data is above it.
```

```
In [58]: final_data=final_data[final_data['Count']<217].copy()
```

```
In [59]: sns.boxplot(x=final_data['Count'])
plt.show()
```

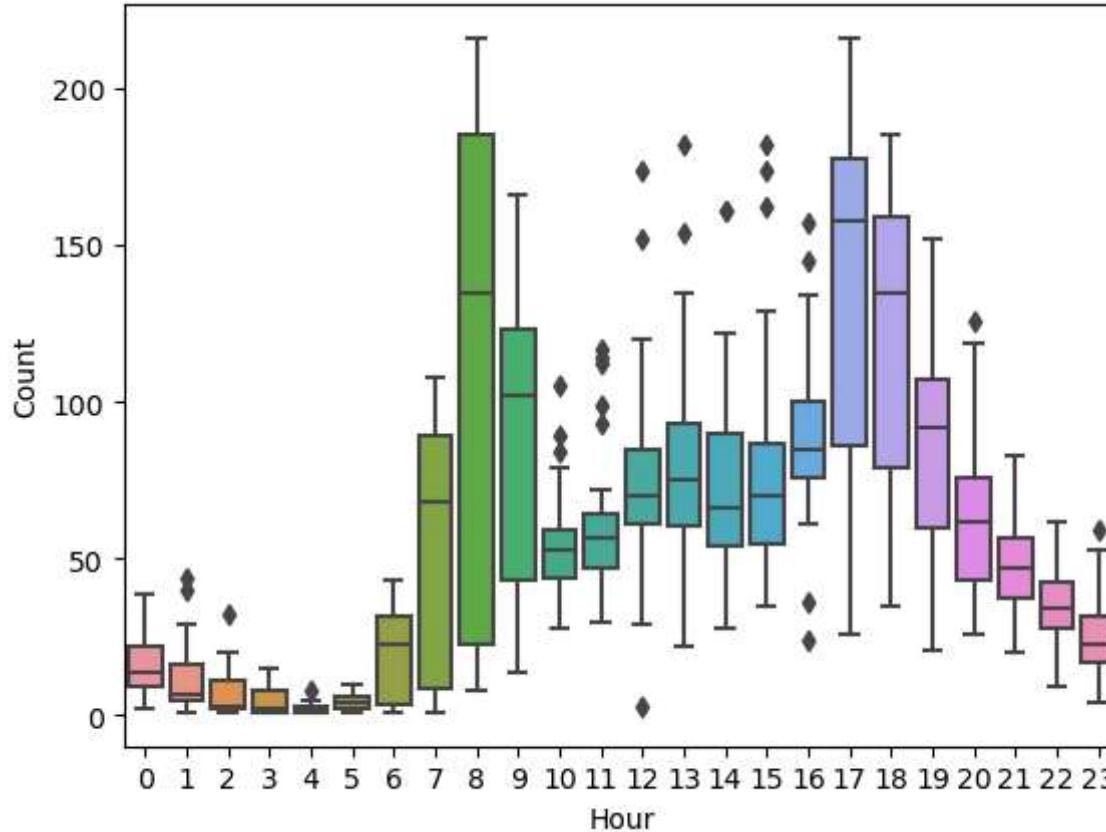


```
In [60]: final_data.shape
```

```
Out[60]: (979, 16)
```

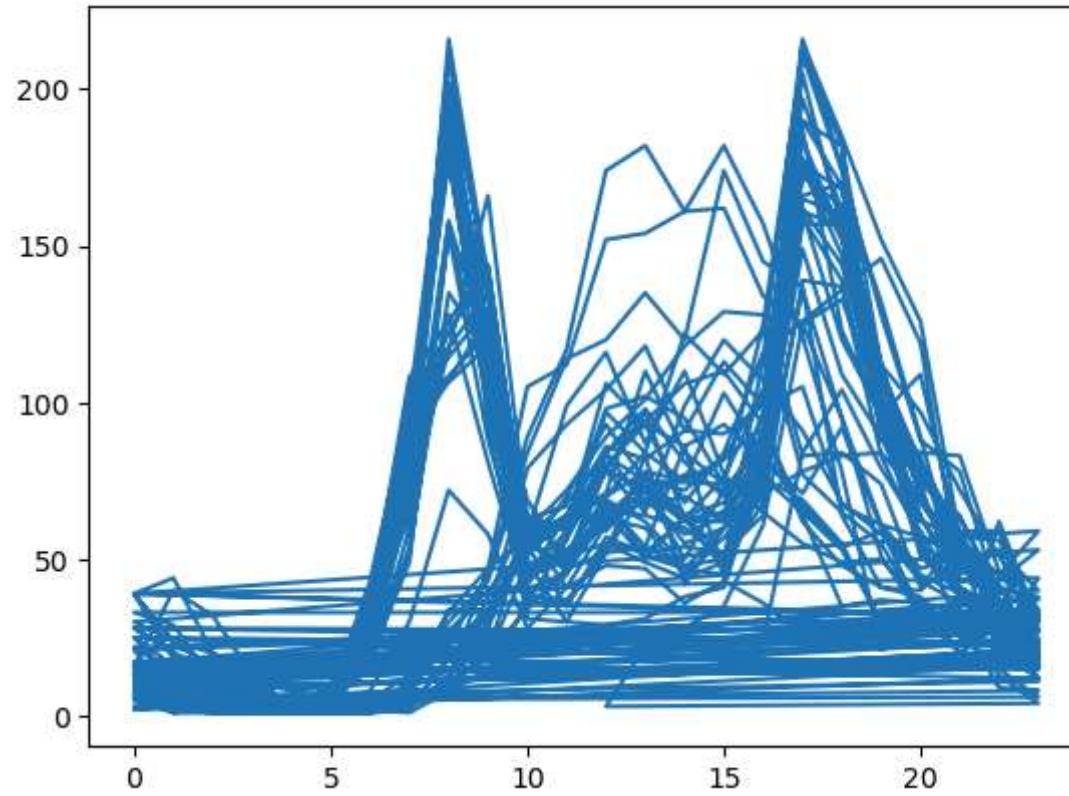
```
In [61]: ###Q) make a boxplot for count vs hr and give your observations on the same.
```

```
In [62]: sns.boxplot(x='Hour',y='Count',data=final_data)
plt.show()
```

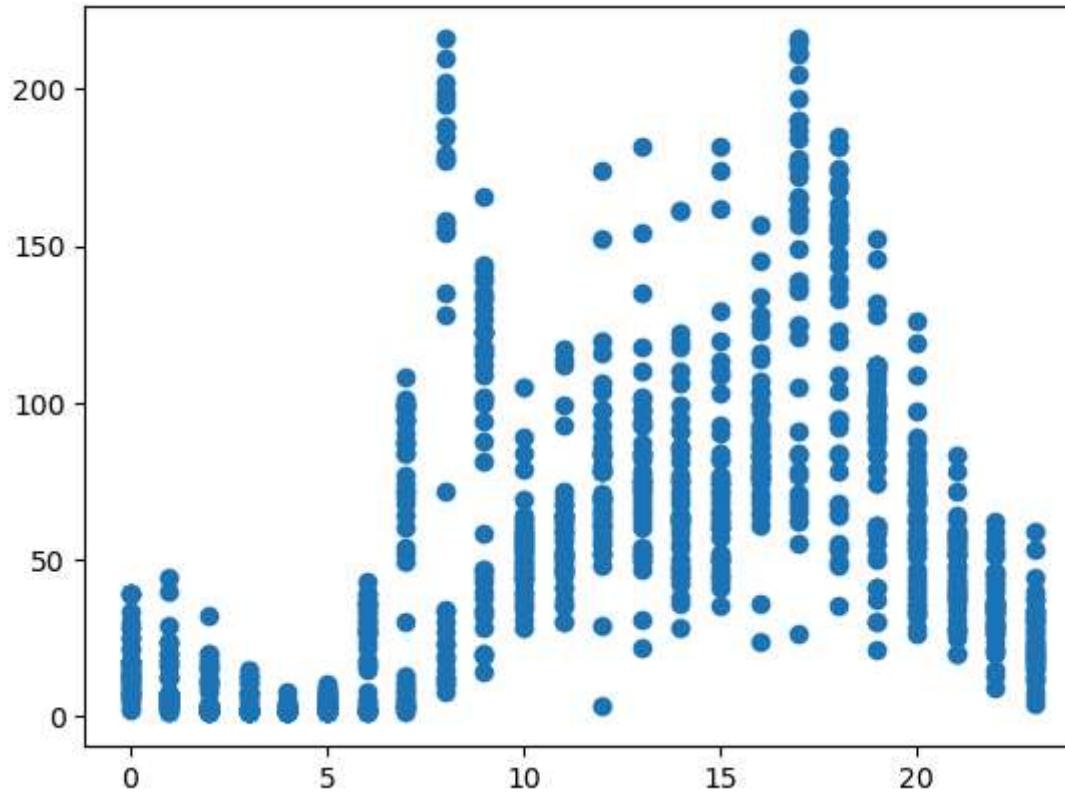


- demand increases early in the morning 6am-8.30am,starts falling from 9am,remains stable throughout the day and increases in the evening,recording the highest at 5pm

```
In [63]: plt.plot(final_data['Hour'],final_data['Count'])
plt.show()
```



```
In [64]: plt.scatter(final_data['Hour'],final_data['Count'])
plt.show()
```



```
In [65]: ### the points in a scatterplot joined by lines is a lineplot
```

```
In [66]: salary=pd.read_csv('salary_data.csv')
salary
```

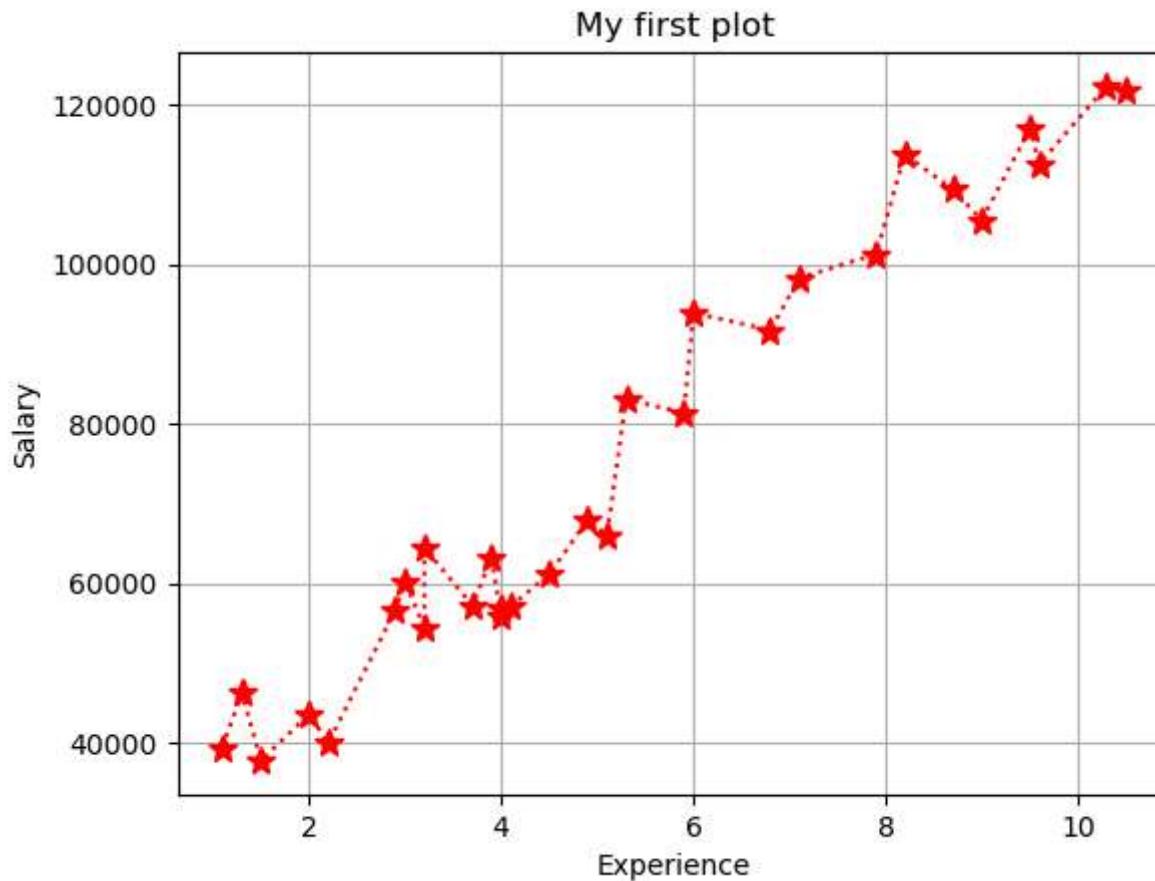
Out[66]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0

	YearsExperience	Salary
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

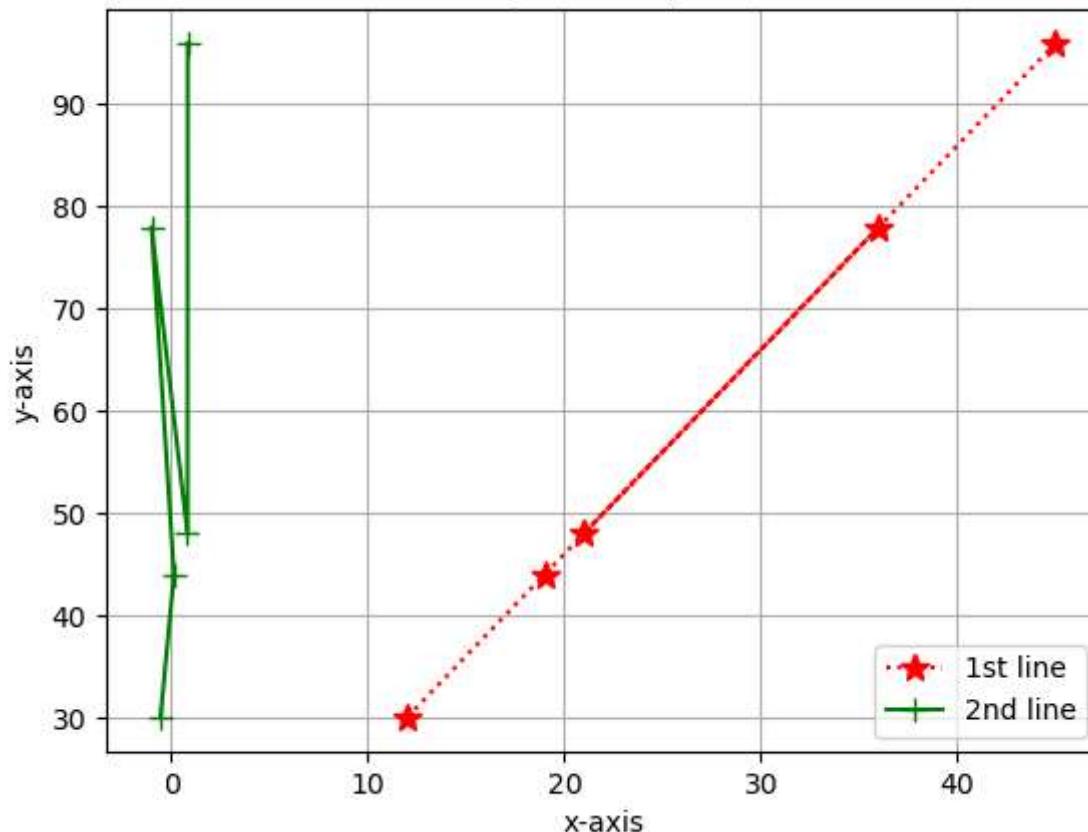
```
In [67]: x=np.array([12,19,36,21,45])
y=np.sin(x)
y1 =2*x+6

plt.plot(salary[ 'YearsExperience'],salary[ 'Salary'],color='red',markersize=10,marker='*',linestyle='dotted')
#plt.plot(x,y,color='green',markersize=8,marker='+',label='2nd Line')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.title('My first plot')
plt.grid()
plt.show()
```



```
In [68]: plt.plot(x,y1,color='red',markersize=10,marker='*',linestyle='dotted',label='1st line')
plt.plot(y,y1,color='green',markersize=8,marker='+',label='2nd line')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('My second plot')
plt.grid()
plt.legend()
plt.show()
```

My second plot



```
In [69]: #?plt.plot
```

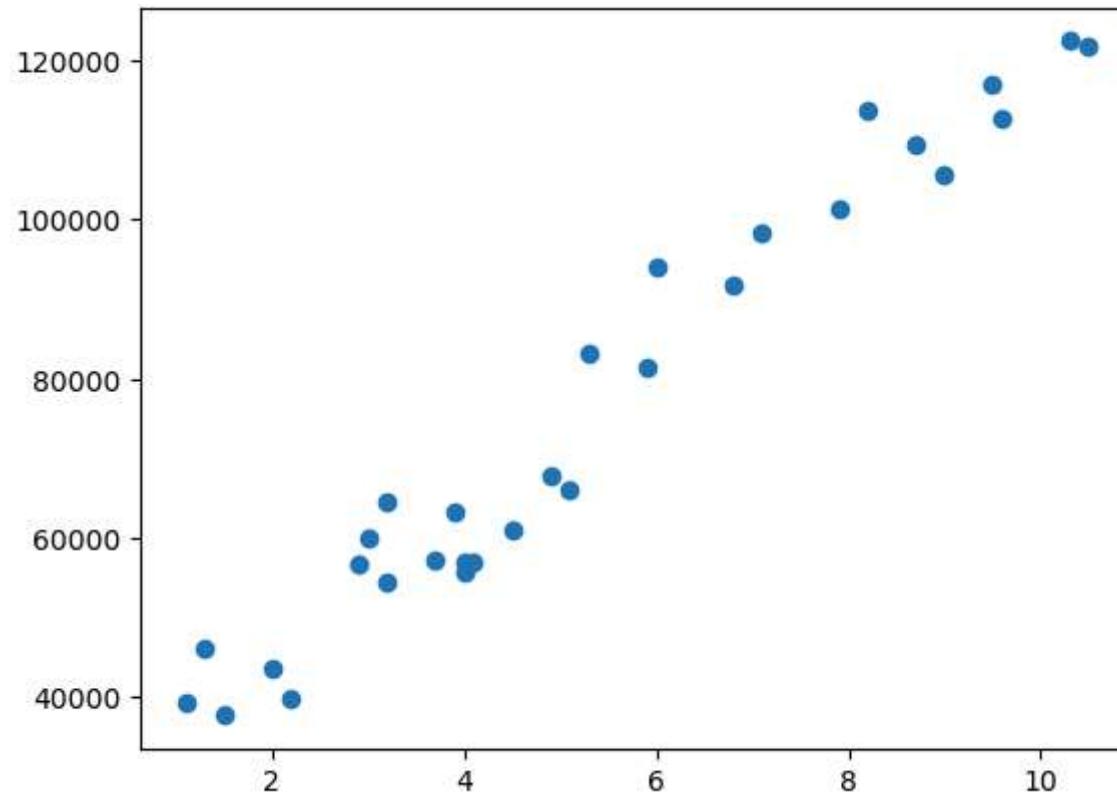
```
In [70]: ###as we see above there is a clear positive relationship, let us give a value to this relationship.  
salary.corr()
```

```
Out[70]:
```

	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

```
In [71]: ### when years of experience increases by 1 unit, the salary increases by 0.978242 units.  
#### 0.978242 is called pearsons correlation coefficient.
```

```
In [72]: plt.scatter(salary['YearsExperience'], salary['Salary'])
plt.show()
```

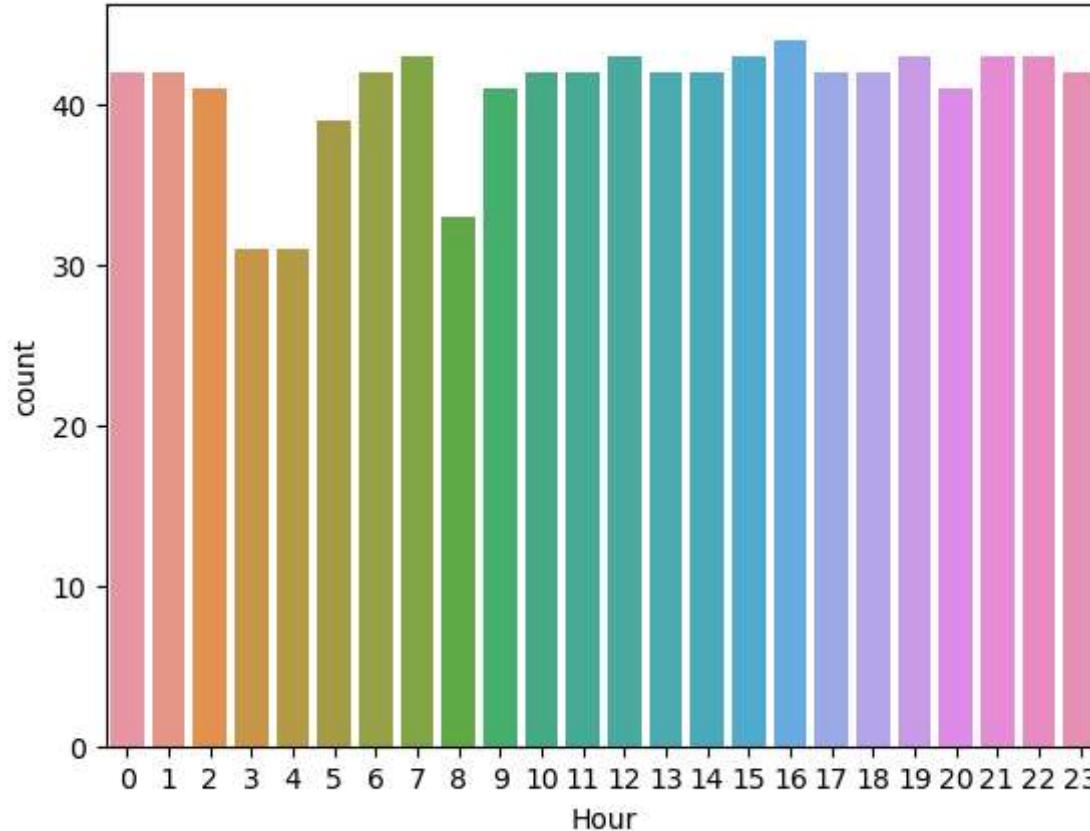


```
In [73]: ###explore a categorical variable we use barplots or countplots.
sns.countplot(x='season',data=final_data)
plt.show()
```

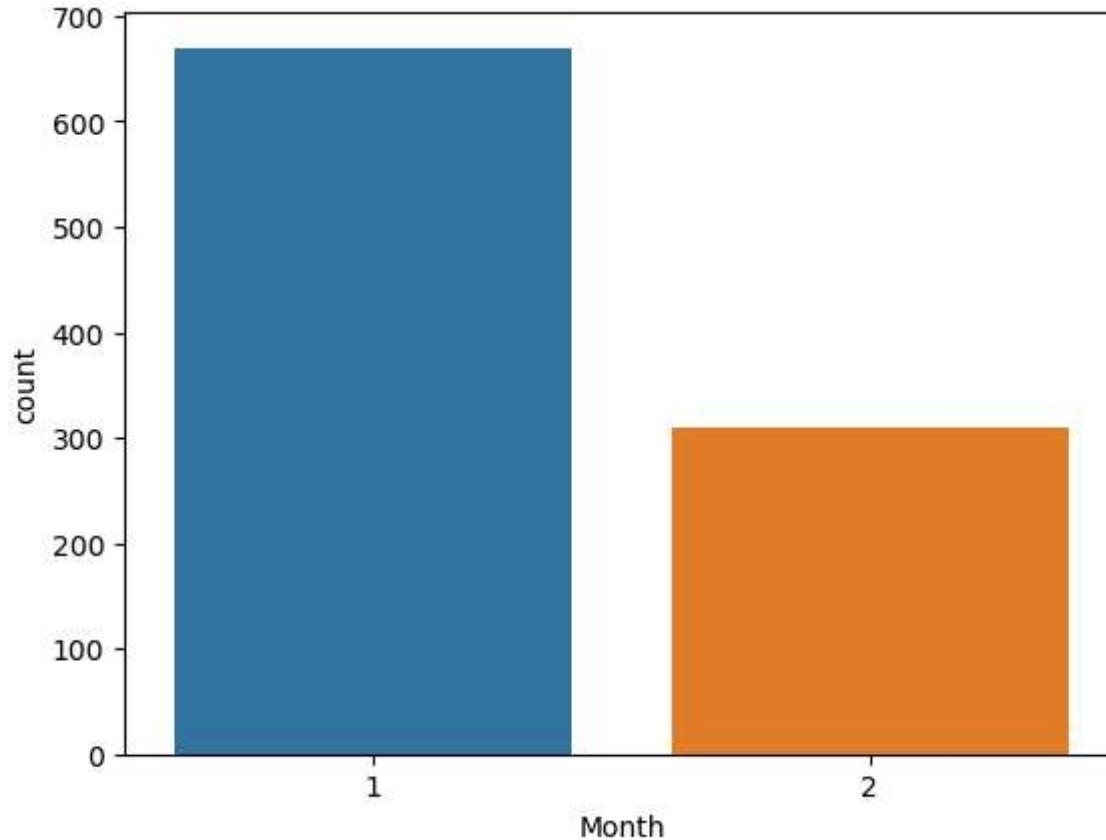
```
print(np.unique(final_data.season))
```

```
[1]
```

```
#print(np.unique(final_data.Hour))
sns.countplot(x='Hour',data=final_data)
plt.show()
```



```
In [75]: sns.countplot(x='Month', data=final_data)
plt.show()
```



```
In [76]: ##### most of the rides are recorded in jan, and at 4pm
```

```
In [77]: x=np.arange(10)
x
```

```
Out[77]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [78]: y1=x**2
y2=x*3-5
```

```
In [79]: y1
```

```
Out[79]: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
In [80]: y2
```

```
Out[80]: array([-5, -2,  1,  4,  7, 10, 13, 16, 19, 22])
```

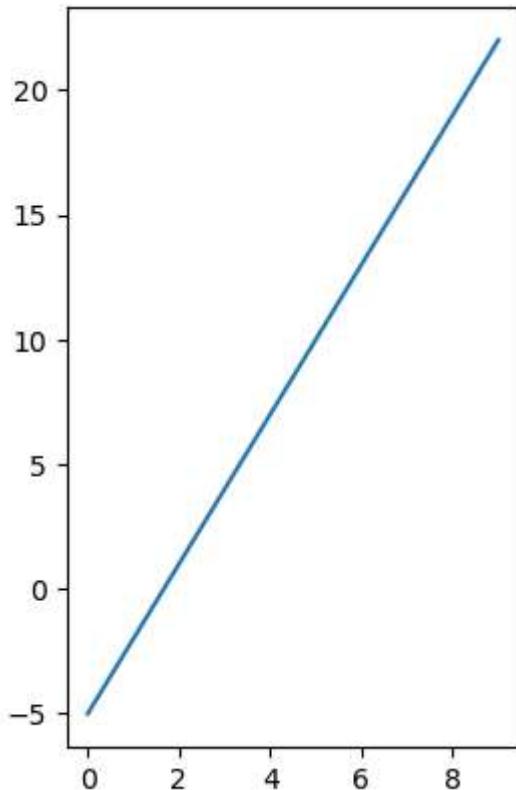
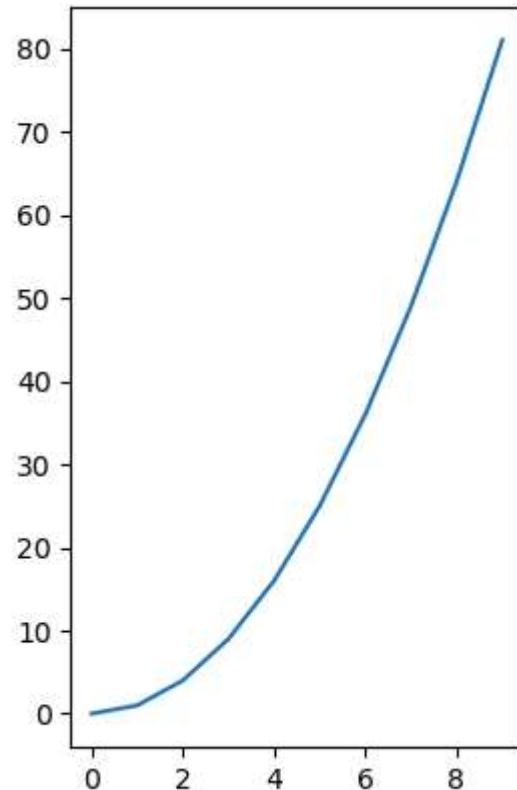
```
In [81]: plt.subplot(1,2,1)
```

```
plt.plot(x,y1)
```

```
plt.subplot(1,2,2)
```

```
plt.plot(x,y2)
```

```
plt.show()
```

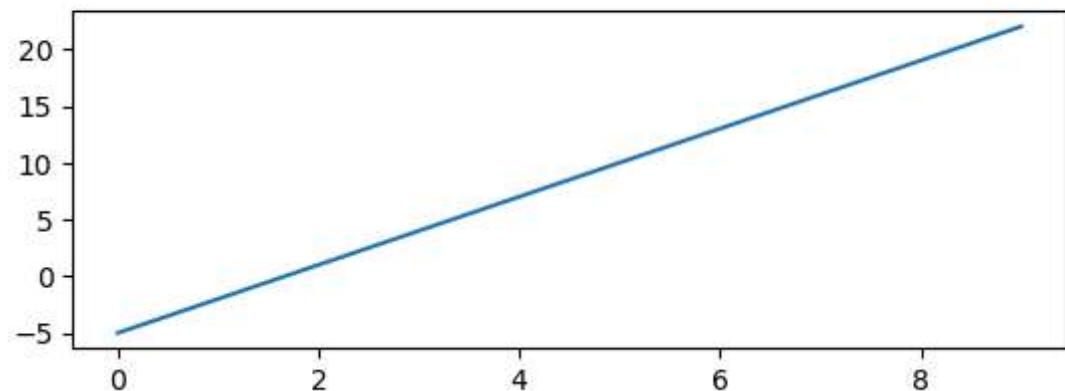
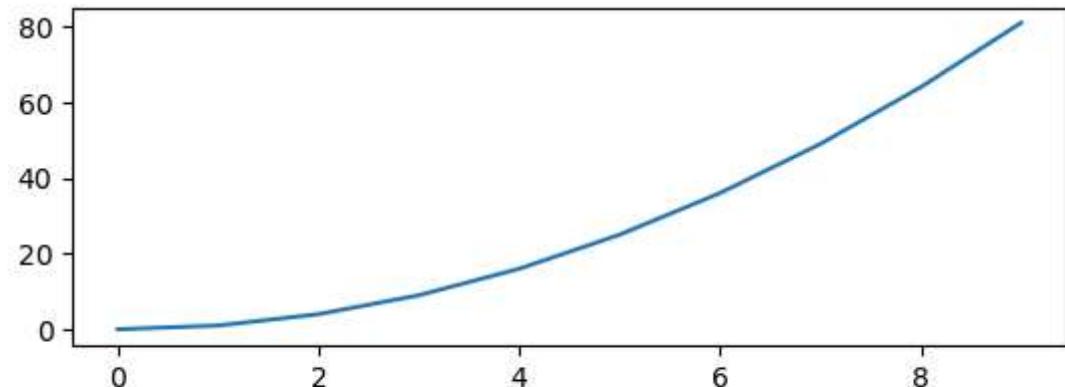


```
In [82]: plt.subplot(2,1,1)
```

```
plt.plot(x,y1)
```

```
plt.subplot(2,1,2)
```

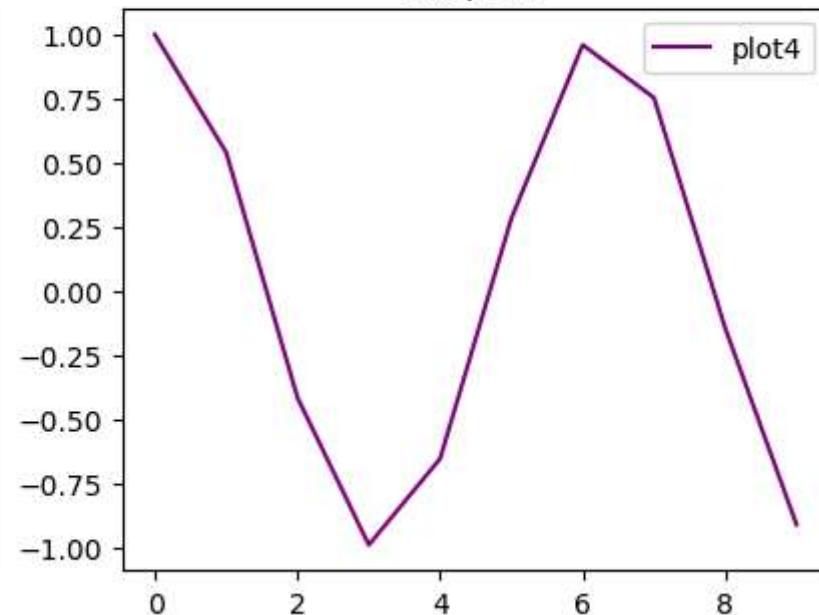
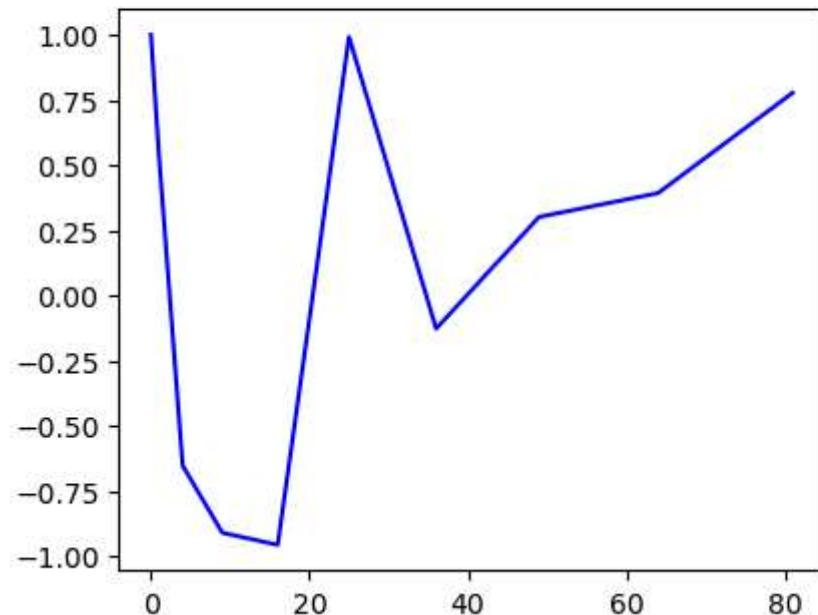
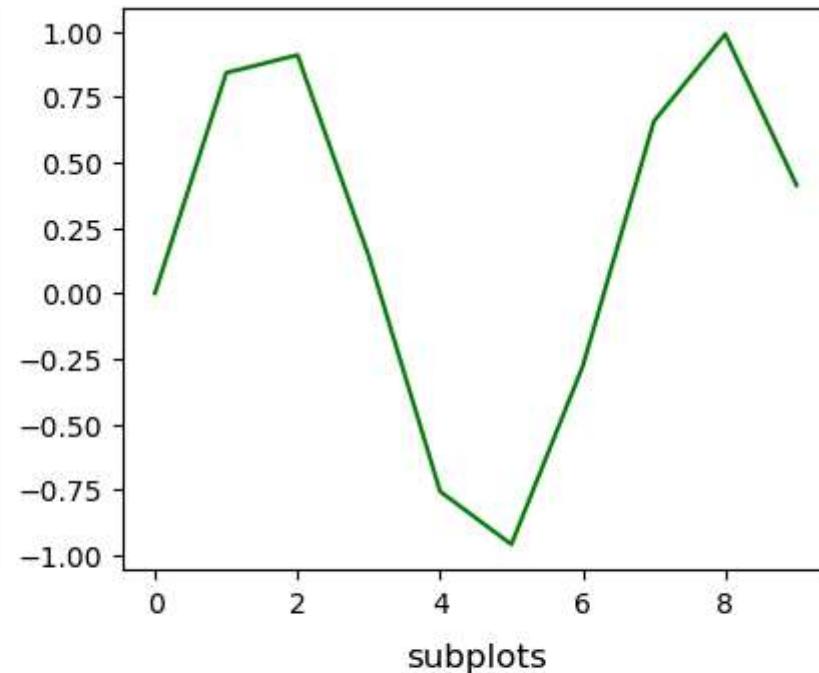
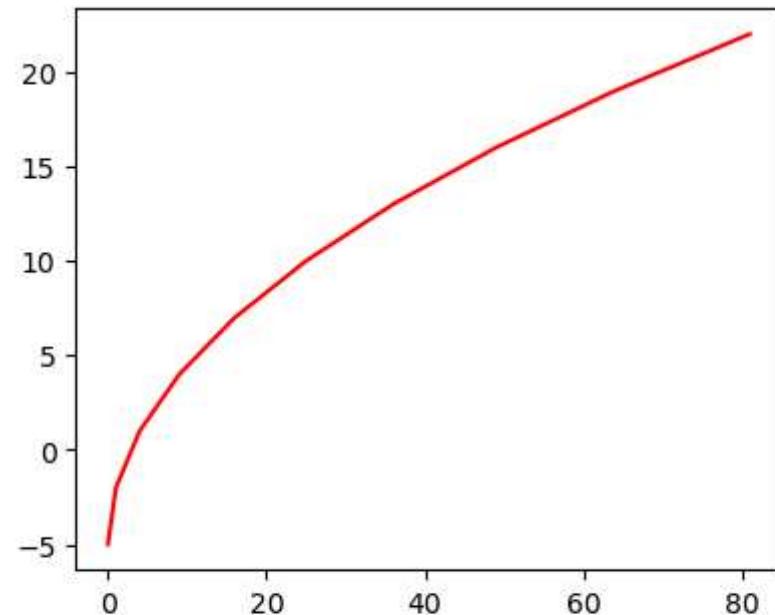
```
plt.plot(x,y2)  
plt.show()
```



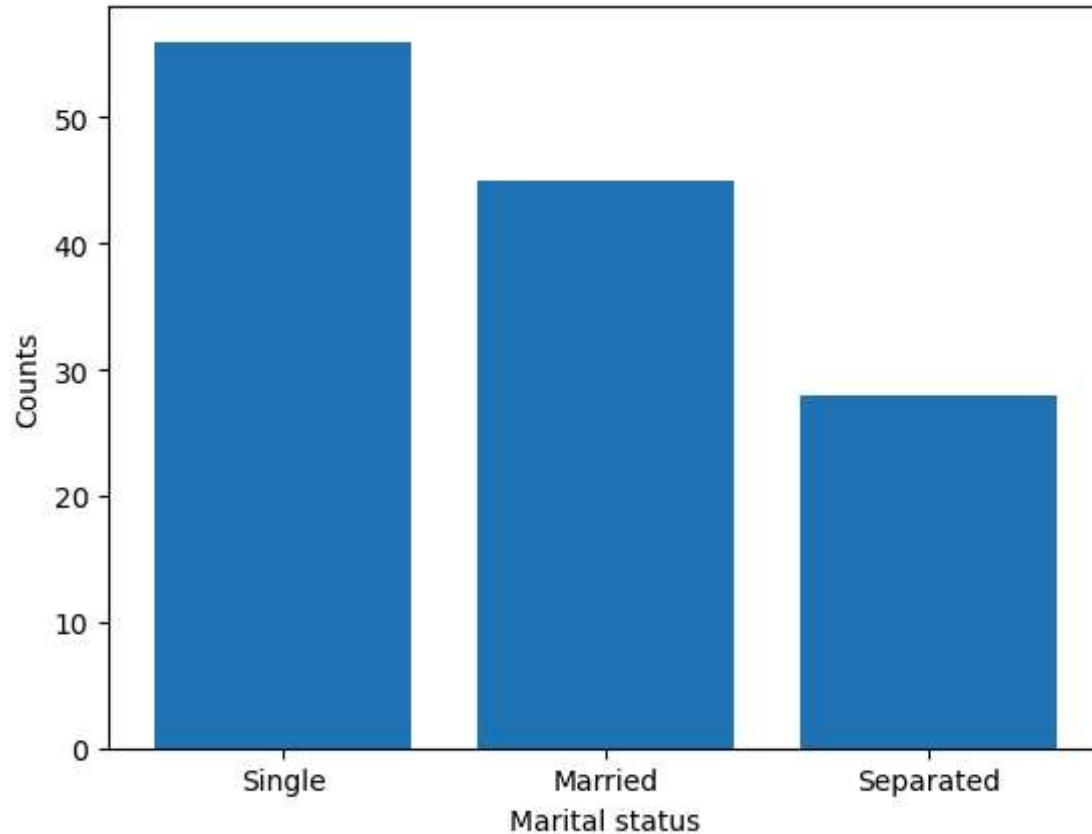
In [83]: *#default plotting area-6.4 and 4.8 inches.*

```
#plt.figure(figsize=(8,5))  
  
fig,ax=plt.subplots(nrows=2,ncols=2,figsize=(10,8))  
  
ax[0,0].plot(y1,y2,color='r',label='plot1')  
ax[0,1].plot(x,np.sin(x),color='g',label='plot2')  
ax[1,0].plot(y1,np.cos(y1),color='b',label='plot3')  
ax[1,1].plot(x,np.cos(x),color='purple',label='plot4')  
  
plt.legend()  
plt.title('subplots')
```

```
plt.show()
```

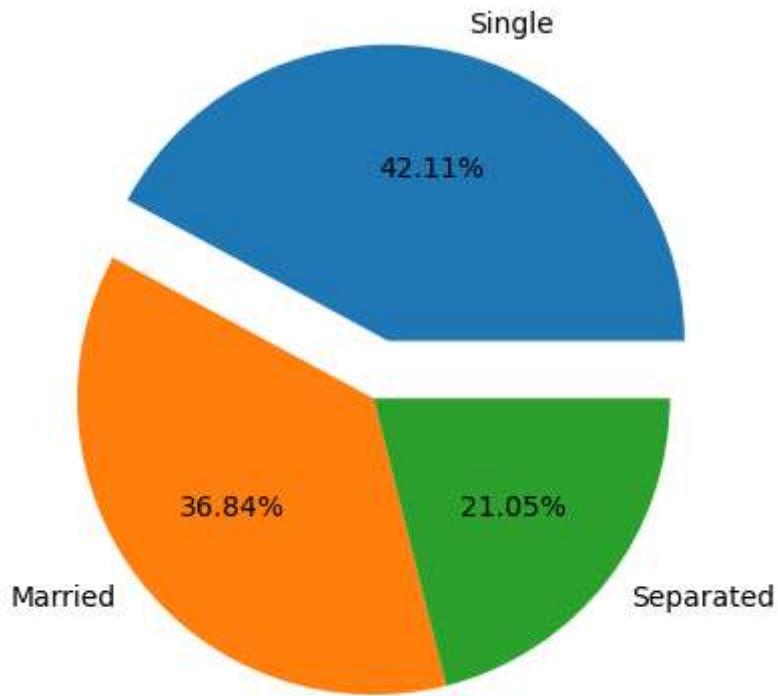


```
In [84]: x=np.array(['Single','Married','Separated'])
y=np.array([56,45,28])
plt.bar(x,y)
plt.xlabel('Marital status')
plt.ylabel('Counts')
plt.show()
```



```
In [85]: labels=['Single','Married','Separated']
y=np.array([56,49,28])
myexplode=[0.2,0,0]

plt.pie(y,labels=labels,explode=myexplode,autopct='%1.2f%%')
plt.show()
```



In [86]: `final_data.head()`

	instant	Date	season	Year	Month	Hour	holiday	weekday	Weather	Temperature	atemp	Humidity	windspeed	casual	registered	Count
0	1	01-01-2011	1	0	1	0	False	6	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	01-01-2011	1	0	1	1	False	6	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	01-01-2011	1	0	1	2	False	6	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	01-01-2011	1	0	1	3	False	6	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	01-01-2011	1	0	1	4	False	6	1	0.24	0.2879	0.75	0.0	0	1	1

```
In [87]: final_data.columns
```

```
Out[87]: Index(['instant', 'Date', 'season', 'Year', 'Month', 'Hour', 'holiday',
   'weekday', 'Weather', 'Temperature', 'atemp', 'Humidity', 'windspeed',
   'casual', 'registered', 'Count'],
  dtype='object')
```

```
In [88]: final_data['Year'].value_counts()
```

```
Out[88]: Year
0    979
Name: count, dtype: int64
```

```
In [89]: final_data['holiday'].value_counts()
```

```
Out[89]: holiday
False    955
True     24
Name: count, dtype: int64
```

```
In [90]: final_data.drop(['Year','instant','Date','season'],axis=1,inplace=True)
```

```
In [91]: final_data.shape
```

```
Out[91]: (979, 12)
```

```
In [92]: #remove duplicates:
print(final_data.drop_duplicates(keep=False))
```

	Month	Hour	holiday	weekday	Weather	Temperature	atemp	Humidity	\
0	1	0	False	6	1	0.24	0.2879	0.81	
1	1	1	False	6	1	0.22	0.2727	0.80	
2	1	2	False	6	1	0.22	0.2727	0.80	
3	1	3	False	6	1	0.24	0.2879	0.75	
4	1	4	False	6	1	0.24	0.2879	0.75	
..
376	2	3	False	1	1	0.34	0.3182	0.46	
377	2	4	False	1	1	0.32	0.3030	0.53	
378	2	5	False	1	1	0.32	0.3030	0.53	
379	2	6	False	1	1	0.34	0.3030	0.46	
380	2	7	False	1	1	0.34	0.3030	0.46	
	windspeed	casual	registered	Count					
0	0.0000	3	13	16					
1	0.0000	8	32	40					
2	0.0000	5	27	32					
3	0.0000	3	10	13					
4	0.0000	0	1	1					
..					
376	0.2239	1	1	2					
377	0.2836	0	2	2					
378	0.2836	0	3	3					
379	0.2985	1	25	26					
380	0.2985	2	96	98					

[979 rows x 12 columns]

In [93]: `final_data.shape`

Out[93]: (979, 12)

In [94]: `filtered_df=final_data.query('holiday==True')`
`filtered_df`

Out[94]:

	Month	Hour	holiday	weekday	Weather	Temperature	atemp	Humidity	windspeed	casual	registered	Count
372	1	0	True	1	2	0.20	0.1970	0.47	0.2239	1	16	17
373	1	1	True	1	2	0.20	0.1970	0.44	0.1940	1	15	16
374	1	2	True	1	2	0.18	0.1667	0.43	0.2537	0	8	8
375	1	3	True	1	2	0.18	0.1818	0.43	0.1940	0	2	2
376	1	4	True	1	2	0.18	0.1970	0.43	0.1343	1	2	3
377	1	5	True	1	2	0.18	0.1970	0.43	0.1642	0	1	1
378	1	6	True	1	2	0.18	0.1818	0.43	0.1940	0	5	5
379	1	7	True	1	2	0.16	0.1818	0.50	0.1343	4	9	13
380	1	8	True	1	2	0.16	0.1515	0.47	0.2239	3	30	33
381	1	9	True	1	2	0.16	0.1515	0.47	0.2239	8	39	47
382	1	10	True	1	2	0.16	0.1515	0.50	0.2537	7	50	57
383	1	11	True	1	2	0.16	0.1515	0.55	0.1940	9	55	64
384	1	12	True	1	2	0.18	0.1970	0.47	0.1343	10	70	80
385	1	13	True	1	2	0.18	0.1970	0.47	0.1343	13	80	93
386	1	14	True	1	2	0.18	0.2121	0.43	0.1045	12	74	86
387	1	15	True	1	2	0.20	0.2121	0.47	0.1642	21	72	93
388	1	16	True	1	2	0.20	0.2121	0.47	0.1642	6	76	82
389	1	17	True	1	1	0.20	0.1970	0.51	0.1940	4	67	71
390	1	18	True	1	2	0.18	0.1667	0.55	0.2537	7	85	92
391	1	19	True	1	3	0.18	0.1818	0.59	0.1940	2	58	60
392	1	20	True	1	3	0.16	0.1515	0.80	0.1940	4	29	33
393	1	21	True	1	3	0.16	0.1515	0.80	0.1940	3	24	27
394	1	22	True	1	3	0.14	0.1212	0.93	0.2537	0	13	13
395	1	23	True	1	3	0.16	0.1364	0.86	0.2836	1	3	4

```
In [95]: final_data['holiday'].value_counts()
```

```
Out[95]: holiday
      False    955
      True     24
Name: count, dtype: int64
```

```
In [96]: #holiday has to be encoded:
cat_col=['holiday']
final_data=pd.get_dummies(final_data,columns=cat_col,drop_first=True)
final_data.head()
```

```
Out[96]:   Month  Hour  weekday  Weather  Temperature  atemp  Humidity  windspeed  casual  registered  Count  holiday_True
0       1      0        6         1       0.24  0.2879     0.81      0.0       3      13      16      False
1       1      1        6         1       0.22  0.2727     0.80      0.0       8      32      40      False
2       1      2        6         1       0.22  0.2727     0.80      0.0       5      27      32      False
3       1      3        6         1       0.24  0.2879     0.75      0.0       3      10      13      False
4       1      4        6         1       0.24  0.2879     0.75      0.0       0       1       1      False
```

```
In [111...]: final_data['holiday_True'].value_counts()
```

```
Out[111]: holiday_True
      False    955
      True     24
Name: count, dtype: int64
```

```
In [97]: final_data.loc[372,:]
```

```
Out[97]:   Month  Hour  weekday  Weather  Temperature  atemp  Humidity  windspeed  casual  registered  Count  holiday_True
372      1      0        1         2       0.2  0.1970     0.47      0.2239      1      16      17      True
372      2     23        0         1       0.4  0.4091     0.35      0.3582      3      27      30      False
```

```
In [112...]: final_data.dtypes
```

```
Out[112]:
```

Month	int64
Hour	int64
weekday	int64
Weather	int64
Temperature	float64
atemp	float64
Humidity	float64
windspeed	float64
casual	int64
registered	int64
Count	int64
holiday_True	bool
	dtype: object

```
In [113...]: #splitting the data into feature set(X) and DV(y)  
X=final_data.drop('Count',axis=1)  
y=final_data['Count']
```

```
In [116...]: X
```

```
Out[116]:
```

	Month	Hour	weekday	Weather	Temperature	atemp	Humidity	windspeed	casual	registered	holiday_True
0	1	0	6	1	0.24	0.2879	0.81	0.0000	3	13	False
1	1	1	6	1	0.22	0.2727	0.80	0.0000	8	32	False
2	1	2	6	1	0.22	0.2727	0.80	0.0000	5	27	False
3	1	3	6	1	0.24	0.2879	0.75	0.0000	3	10	False
4	1	4	6	1	0.24	0.2879	0.75	0.0000	0	1	False
...
376	2	3	1	1	0.34	0.3182	0.46	0.2239	1	1	False
377	2	4	1	1	0.32	0.3030	0.53	0.2836	0	2	False
378	2	5	1	1	0.32	0.3030	0.53	0.2836	0	3	False
379	2	6	1	1	0.34	0.3030	0.46	0.2985	1	25	False
380	2	7	1	1	0.34	0.3030	0.46	0.2985	2	96	False

979 rows × 11 columns

```
In [117...]: from sklearn.model_selection import train_test_split  
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=50)
```

```
In [101...]: ### random state is used to fix the selection of the training set and testing set.
```

```
In [118...]: from sklearn.linear_model import LinearRegression  
lr=LinearRegression()
```

```
In [119...]: #train the model  
lr.fit(X_train,y_train)
```

```
Out[119]: LinearRegression  
LinearRegression()
```

```
In [120...]: ##test the model:  
predictions=lr.predict(X_test)
```

```
In [125...]: ###evaluating the model:  
from sklearn.metrics import r2_score
```

```
In [126...]: r2=r2_score(y_test,predictions)  
print(r2)
```

```
1.0
```

```
In [107...]: ###The model gives us 100% accuracy,or in other words-actuals==predictions.
```

```
In [108...]: X_test
```

Out[108]:

	Month	Hour	weekday	Weather	Temperature	atemp	Humidity	windspeed	casual	registered	holiday_True
245	2	11	2	1	0.22	0.1818	0.47	0.4627	1	49	False
303	2	23	4	1	0.12	0.1515	0.42	0.1045	1	33	False
586	1	17	3	3	0.20	0.1818	0.93	0.3582	0	26	False
395	1	23	1	3	0.16	0.1364	0.86	0.2836	1	3	True
508	1	7	0	1	0.08	0.1061	0.58	0.1642	1	1	False
...
314	2	12	5	1	0.22	0.2273	0.47	0.1343	7	64	False
159	2	21	5	2	0.24	0.2424	0.70	0.1642	2	41	False
341	1	17	6	2	0.32	0.3030	0.36	0.2836	29	54	False
93	1	1	3	1	0.16	0.1970	0.74	0.0896	0	6	False
171	2	9	6	3	0.20	0.2273	1.00	0.0896	3	17	False

245 rows × 11 columns

In [128...]: y_test

Out[128]:

245	50
303	34
586	26
395	4
508	2
	..
314	71
159	43
341	83
93	6
171	20

Name: Count, Length: 245, dtype: int64

In [129...]: predictions

```
Out[129]: array([ 50.,  34.,  26.,  4.,  2.,  23.,  4.,  139.,  3.,  50.,  2.,
   30.,  59.,  20.,  1.,  53.,  7.,  30.,  20.,  86.,  3.,  1.,
   2., 114.,  46.,  40.,  49.,  7.,  95.,  34.,  76., 129., 119.,
   3.,  3.,  51.,  40.,  68.,  29.,  34.,  78.,  70.,  41.,  84.,
   3.,  3.,  62.,  18.,  63.,  1.,  64.,  61.,  64.,  75.,  5.,
 112.,  11.,  2., 103.,  2.,  55.,  11.,  15.,  3.,  57.,  79.,
  74.,  53.,  96.,  44.,  69.,  11., 112.,  59.,  26.,  32.,  43.,
  69.,  45.,  9.,  18.,  2.,  12.,  56.,  36.,  56.,  34.,  53.,
  59., 120.,  31., 149., 123.,  58.,  69.,  51.,  5.,  51., 120.,
   7.,  69.,  33., 109.,  48.,  35.,  35.,  6., 155.,  32.,  2.,
   1.,  49., 110.,  7.,  77.,  55.,  17.,  75.,  78.,  7.,  52.,
  78.,  53., 144.,  1.,  13.,  60.,  7.,  3.,  51.,  74., 100.,
  29.,  8.,  6.,  59.,  84., 112.,  38.,  37.,  30.,  17.,  58.,
 169.,  2.,  17., 152.,  33.,  28.,  13., 176.,  4.,  86.,  36.,
 145.,  52.,  23.,  76., 132.,  49.,  26.,  54.,  28.,  88.,  1.,
  27.,  17.,  28., 104.,  91.,  56.,  8.,  62.,  2., 101.,  13.,
   3.,  10.,  31.,  69.,  9.,  7.,  15.,  58.,  40.,  51.,  45.,
  54.,  20.,  16.,  12.,  20.,  32.,  47., 134.,  65.,  1.,  74.,
  39.,  40.,  30.,  41.,  72.,  17.,  48., 127.,  64.,  3.,  70.,
   5.,  60.,  30., 175.,  17.,  4.,  56., 101., 103., 110.,  36.,
   2.,  36.,  21.,  84.,  52.,  65.,  41.,  77.,  52.,  41.,  59.,
  65.,  84.,  54., 128.,  2.,  18.,  27.,  28.,  2.,  71.,  43.,
  83.,  6.,  20.])
```

In []: