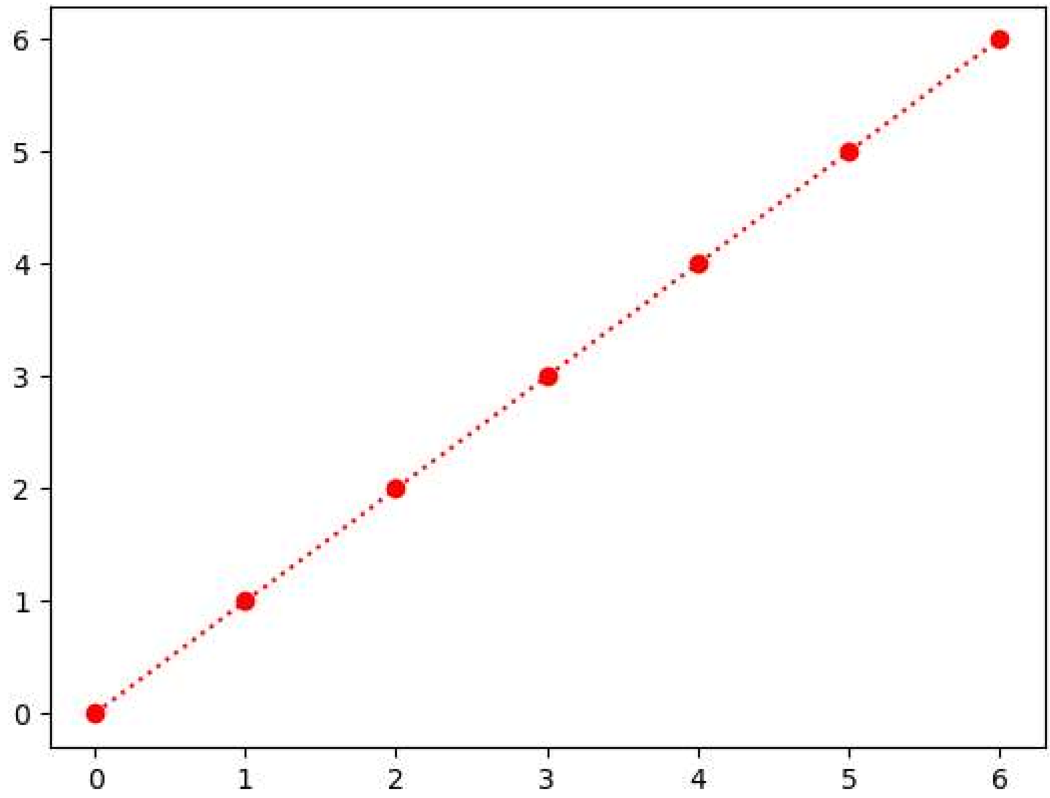
In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

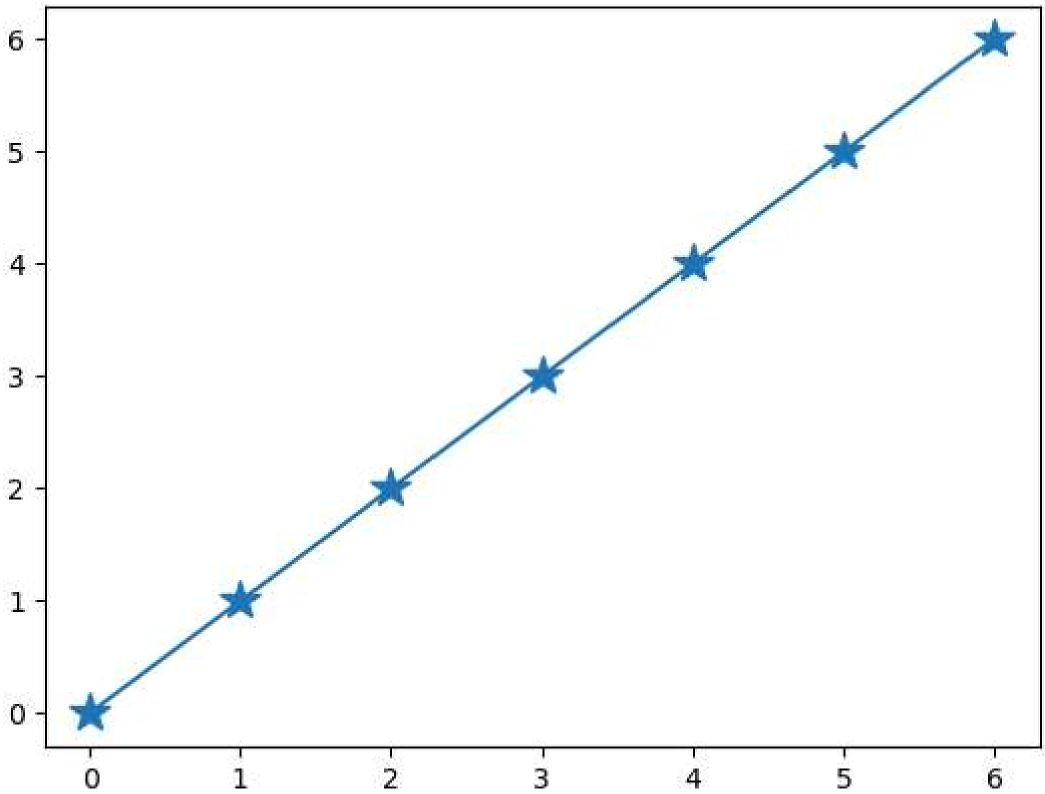
In [8]:

```python
x=np.array([0,1,2,3,4,5,6])
y=np.linspace(0,6,7)
plt.plot(x,y,'o:r')  # 'o:r'= o is marker,,, : is dotted line,,, r is color
plt.show()
```

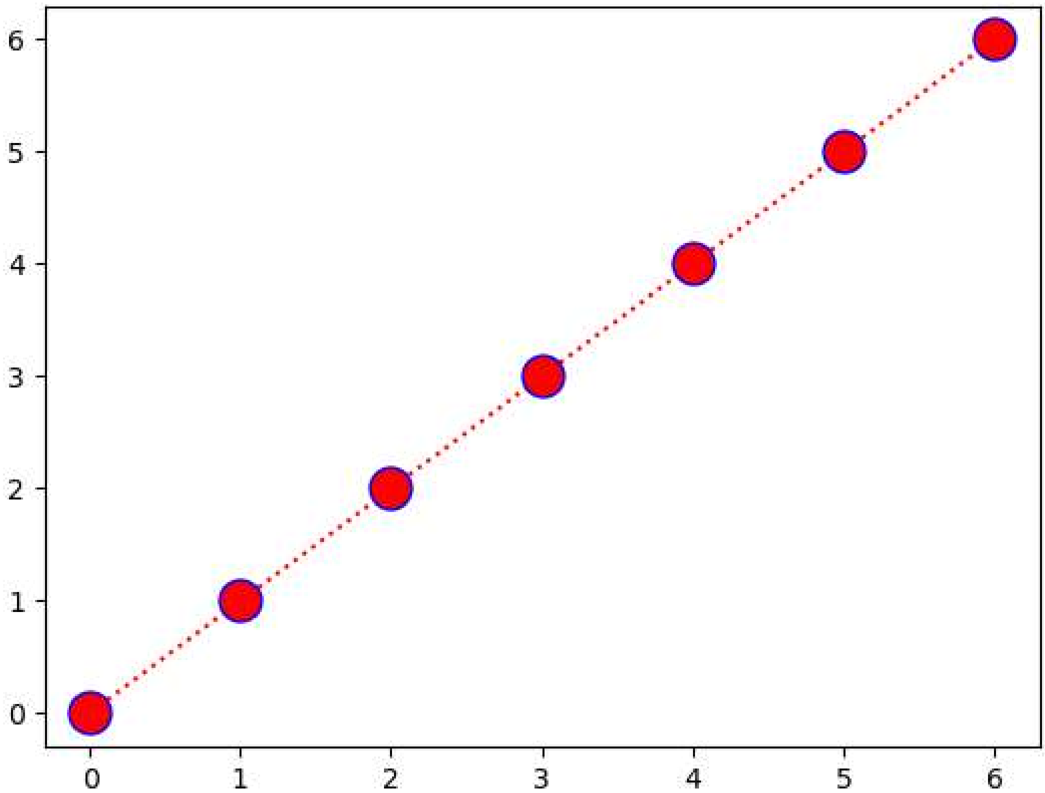


In [10]:

```python
plt.plot(x,y, marker='*', ms=15) # marker with marker size which is given by ms
plt.show()
```

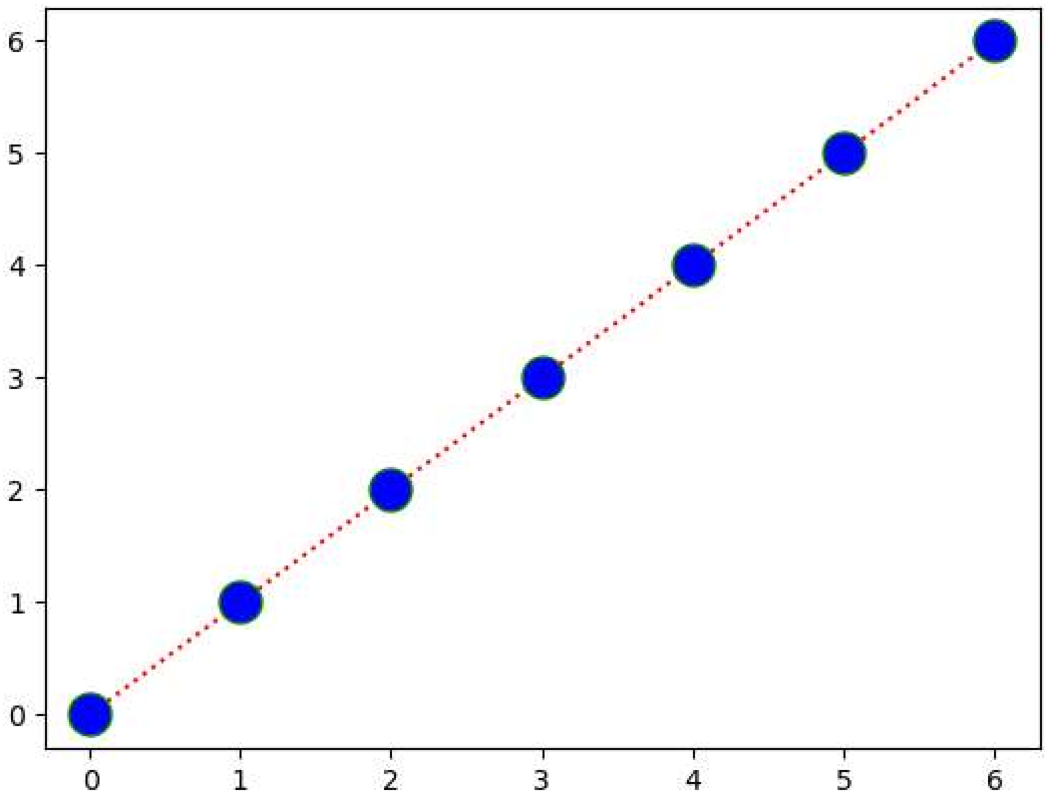


In [14]:
```
# code to execute marker edge color   mec= marker edge color
plt.plot(x,y, 'o:r',ms=15,mec='b')
plt.show()
```

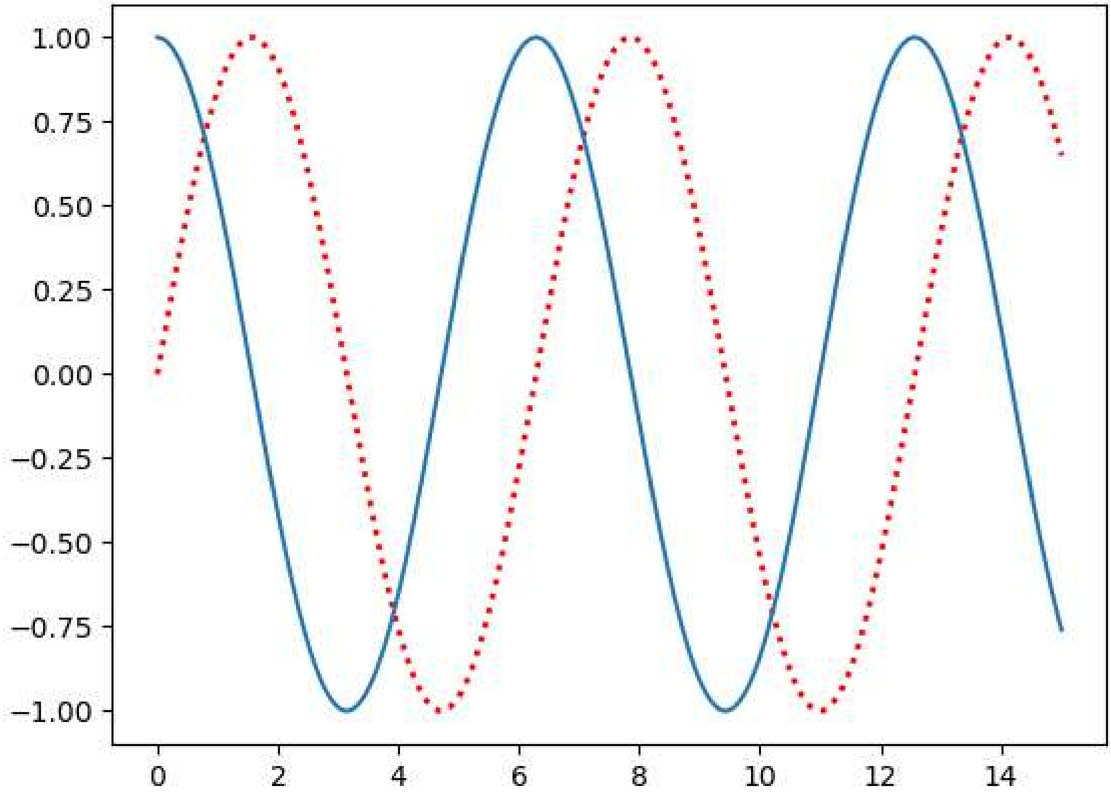


In [15]:
```
# marker face color  mfc= marker face color
```

In [16]:
```
plt.plot(x,y,'o:r', ms=15, mec='g',mfc='b')
plt.show()
```

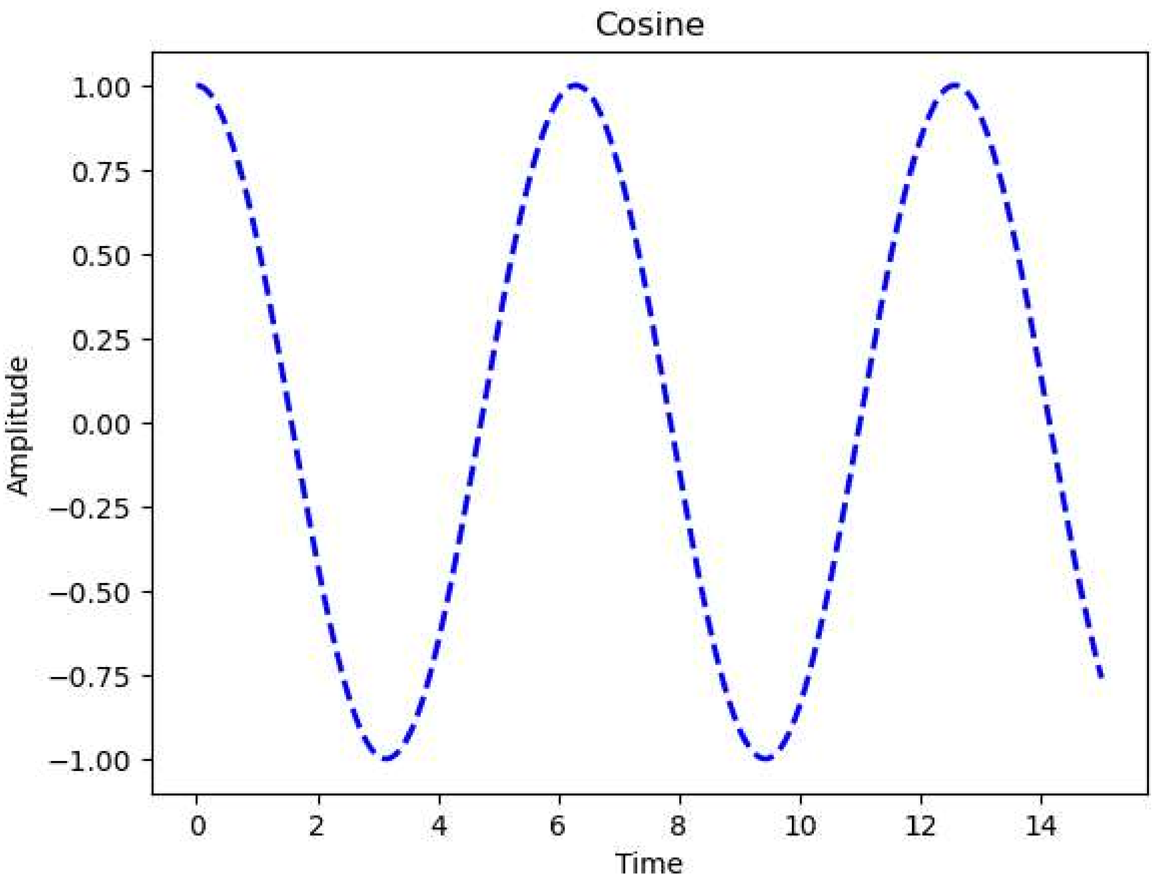


In [39]:
```
xsin=np.linspace(0,15,1000)
ysin=np.sin(xsin)
plt.plot(xsin,ysin,color='red',linestyle='dotted',linewidth='2')
plt.plot(xsin,np.cos(xsin))
plt.show()
```



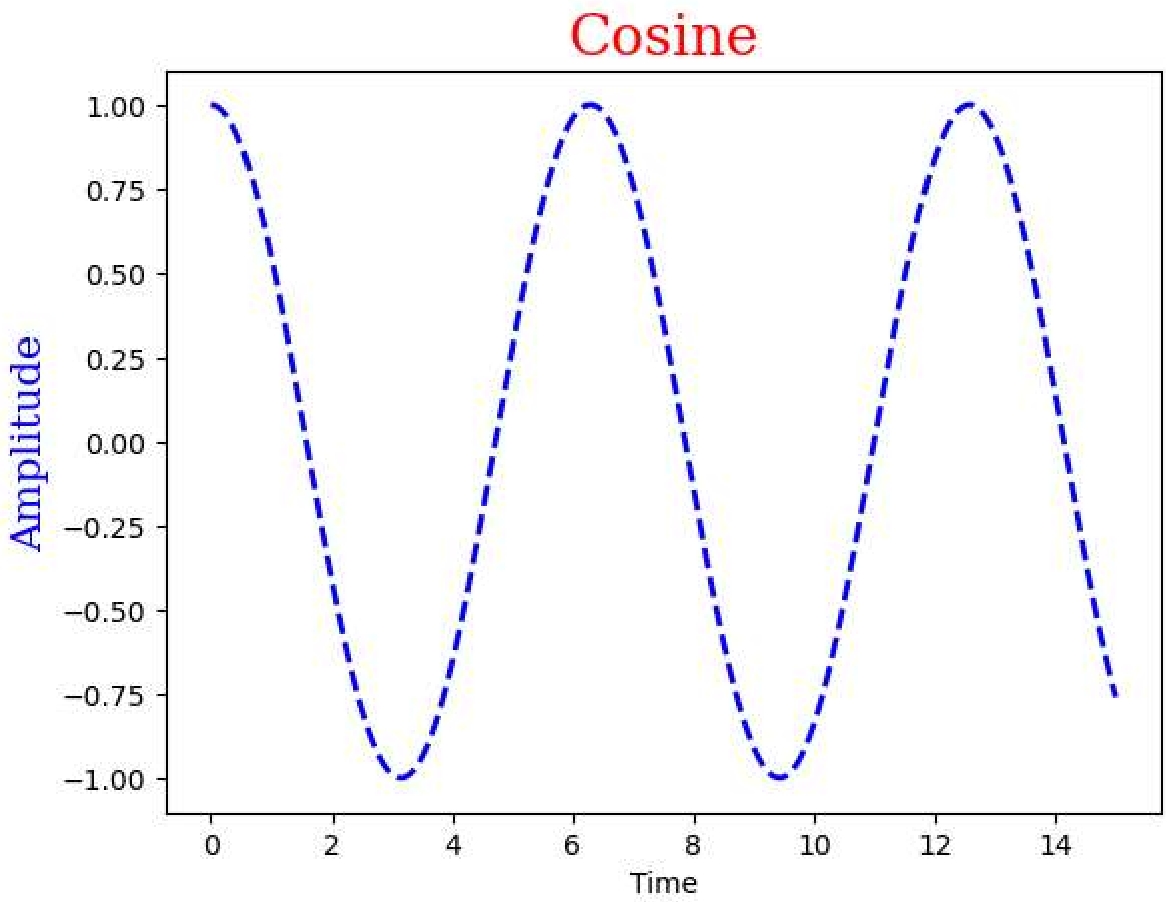
Title, labels

In [42]:
```
plt.plot(xsin,np.cos(xsin),color='b',linestyle='dashed',linewidth='2')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Cosine')
plt.show()
```

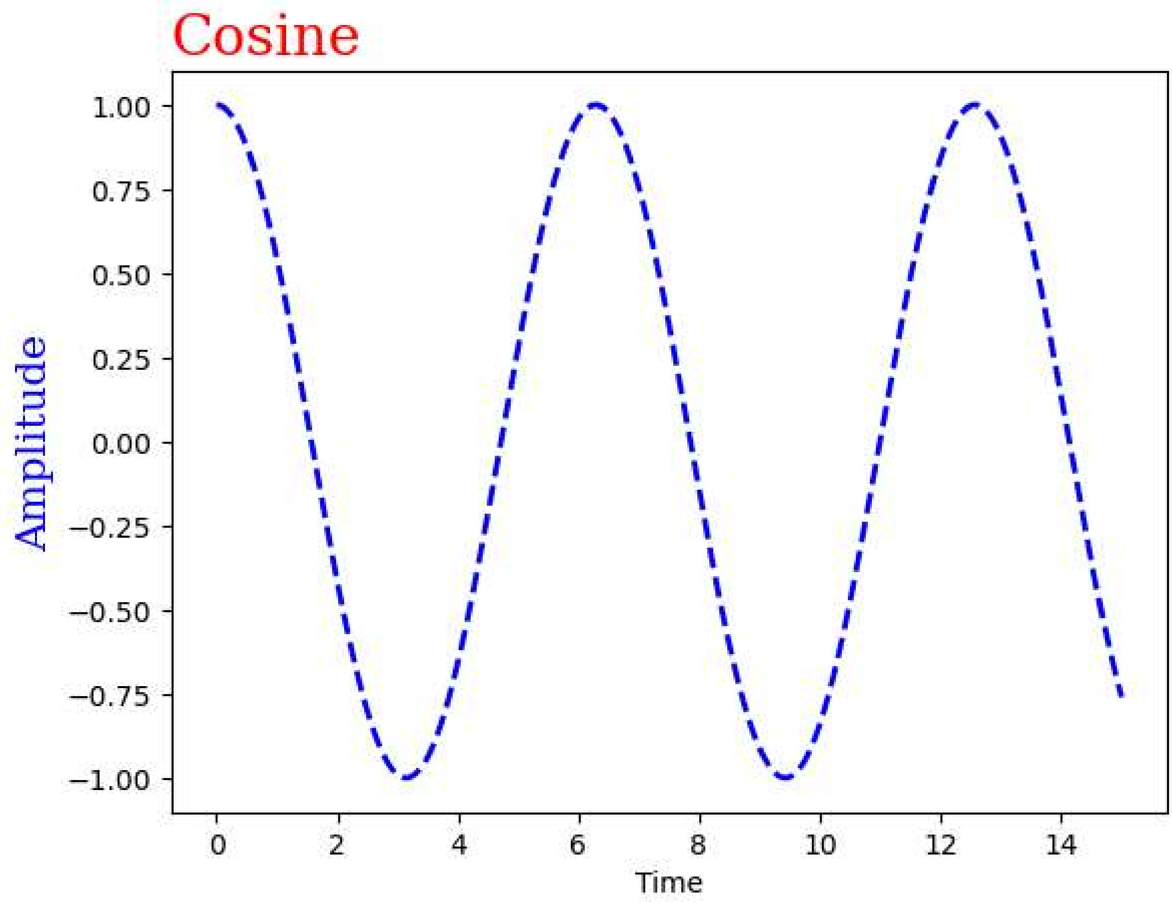


In [43]:
```
## adding variable font in the plot
```

In [46]:
```
font1={'family':'Serif','color':'red','size':20}
font2={'family':'Serif','color':'blue','size':15}
plt.plot(xsin,np.cos(xsin),color='b',linestyle='dashed',linewidth='2')
plt.xlabel('Time')
plt.ylabel('Amplitude',fontdict=font2)
plt.title('Cosine',fontdict=font1)
plt.show()
```



In [89]:
```
# shifing the location of the title
font1={'family':'Serif','color':'red','size':20}
font2={'family':'Serif','color':'blue','size':15}
plt.plot(xsin,np.cos(xsin),color='b',linestyle='dashed',linewidth='2')
plt.xlabel('Time')
plt.ylabel('Amplitude',fontdict=font2)
plt.title('Cosine',fontdict=font1,loc='left')  # left shifting in the title positic
plt.show()
```



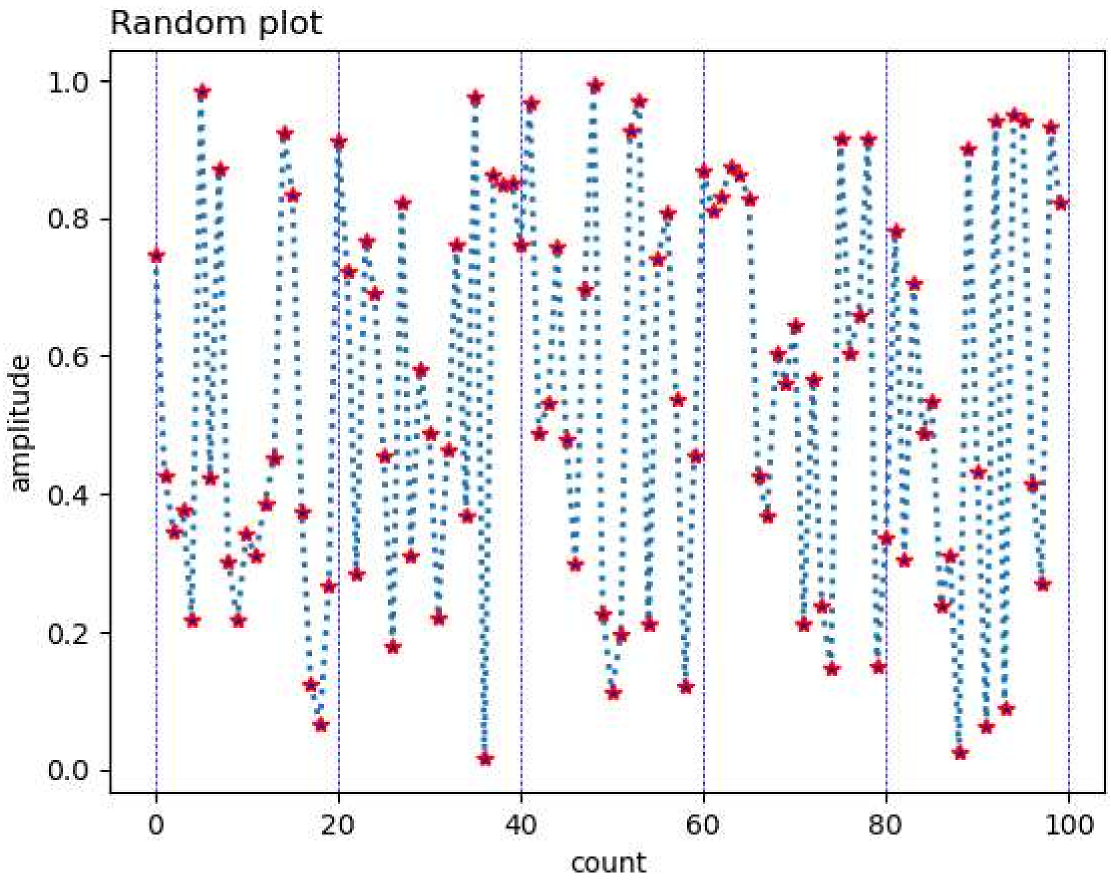
grid lines addition

In [ ]:

```
ms= marker size,, mec= marker edge color,,, mfc= marker facecolor ,,, linestyle, li
```

In [111…

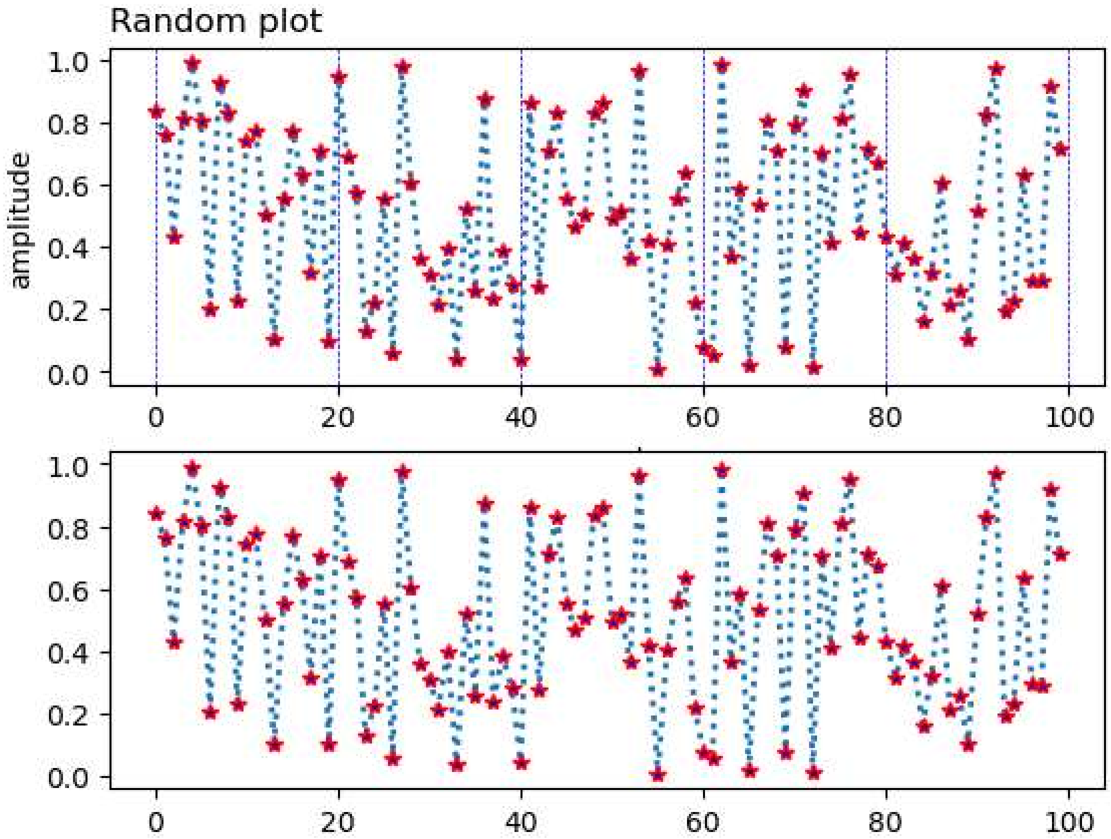
```
import random
val=[]
count=[]
for i in range(0,100):
    val.append(np.random.random())
    count.append(i)

plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.xlabel('count')
plt.ylabel('amplitude')
plt.title('Random plot',loc='left')
plt.grid(axis='x',color='b',linestyle='dashed', linewidth=0.5)
plt
plt.show()
```



Mutiple plots

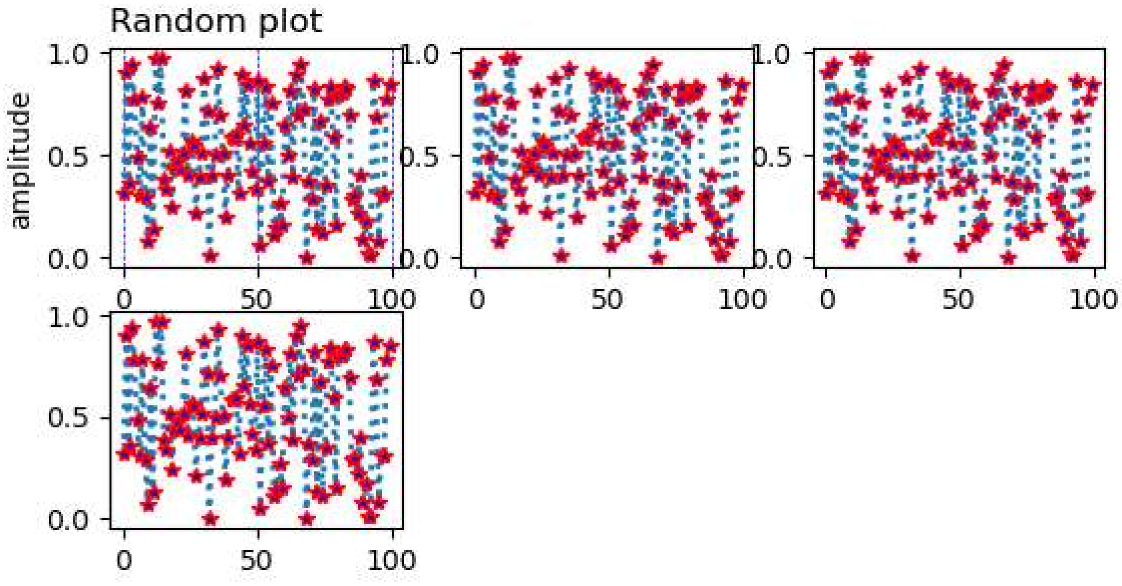
In [119…

```
import random
val=[]
count=[]
for i in range(0,100):
    val.append(np.random.random())
    count.append(i)
plt.subplot(2,1,1)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.xlabel('count')
plt.ylabel('amplitude')
plt.title('Random plot',loc='left')
plt.grid(axis='x',color='b',linestyle='dashed', linewidth=0.5)
plt.subplot(2,1,2)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.show()
```



In [121…

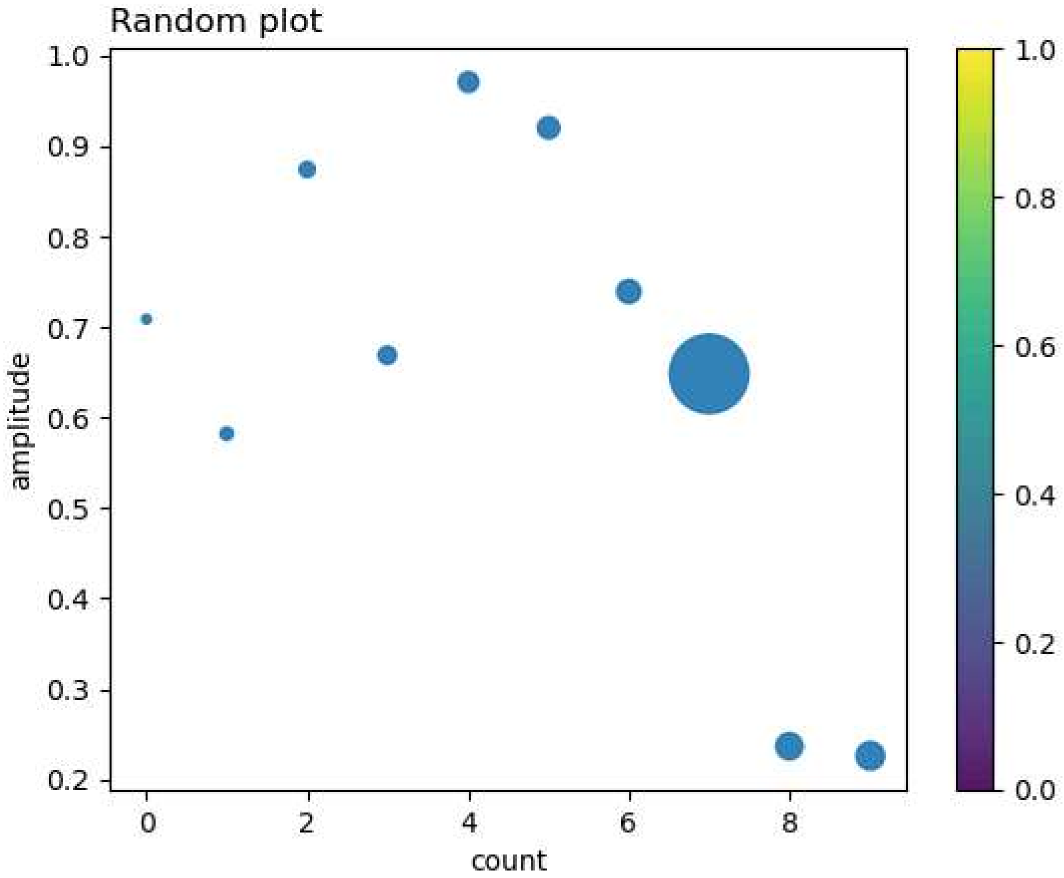
```
import random
val=[]
count=[]
for i in range(0,100):
    val.append(np.random.random())
    count.append(i)
plt.subplot(3,3,1)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.xlabel('count')
plt.ylabel('amplitude')
plt.title('Random plot',loc='left')
plt.grid(axis='x',color='b',linestyle='dashed', linewidth=0.5)
plt.subplot(3,3,2)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.subplot(3,3,3)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.subplot(3,3,4)
plt.plot(count,val,linestyle='dotted',marker='*',mec='r',mfc='b',linewidth='2')
plt.show()
```



Scatter plot

In [135…

```
import random
val=[]
count=[]
for i in range(0,10):
    val.append(np.random.random())
    count.append(i)
sizes=np.array([10,20,30,40,50,60,70,800,90,100])
plt.scatter(count,val,s=sizes,alpha=0.9),#mec='r',mfc='b',linewidth='2')
plt.colorbar()
plt.xlabel('count')
plt.ylabel('amplitude')
plt.title('Random plot',loc='left')
plt.show()
```



In [ ]:

In [ ]:

In [90]:
```
import numpy as np
from scipy.optimize import minimize
```

In [61]:
```
# hydro typical year monthly generation data
```

In [85]:
```
hydro_2021=np.array([285,185,183,55,61,83,45,84,129,219,280,265])
#solar energy data from 100 to 1 MW
solar_data={
    100: np.array([3,4,6,14,12,11,13,10,5,3,2,1]),
    150: np.array([5,5,10,20,18,17,19,14,8,5,3,2]),
    200: np.array([6,7,13,27,23,23,25,19,11,6,5,2]),
    250: np.array([8,9,16,34,29,29,32,24,13,8,6,3]),
    300: np.array([10,11,19,41,35,34,38,29,16,9,7,3]),
    350: np.array([11,12,23,48,41,40,44,33,19,11,8,4]),
    400: np.array([13,14,26,54,47,46,50,38,21,12,9,5]),
    450: np.array([14,16,29,61,53,51,57,43,24,14,10,5]),
    500: np.array([16,18,32,68,58,57,63,48,27,15,11,6]),
    550: np.array([18,19,36,75,64,63,69,52,29,17,12,6]),
    600: np.array([19,21,39,82,70,69,76,57,32,18,14,7]),
    650: np.array([21,23,42,88,76,74,82,62,35,20,15,7]),
    700: np.array([22,25,45,95,82,80,88,67,37,21,16,8]),
    750: np.array([24,26,48,102,88,86,95,71,40,23,17,9]),
    800: np.array([25,28,52,109,94,91,101,76,43,25,18,9]),
    850: np.array([27,30,55,116,99,97,107,81,45,26,19,10]),
    900: np.array([29,32,58,122,105,103,113,86,48,28,28,20]),
    950: np.array([30,34,61,129,111,109,120,90,51,29,22,11]),
    1000: np.array([32,35,65,136,117,114,126,95,53,31,23,12])
}
```

In [63]:
```
# calcuation of energy gap
```

In [86]:
```
def energy_gap(pv_capacity,hydro_energy,grid_limit=720):
    # solar energy monthly data
    solar_energy_generated=solar_data[pv_capacity]

    # summing the total energy
    total_energy=hydro_energy+solar_energy_generated

    # defining curtailment
    curtailment=np.minimum(total_energy, grid_limit)
    total_delivered=curtailment.sum()

    #minimizng the gap: target-(hydro+solar-curtailment)
    return(grid_limit*len(hydro_energy)-total_delivered)**2
```

In [65]:
```
# minimize the energy gap by selecting the optimal capacity from
#the selected range
```

In [88]:
```
def find_optimal_capacity(hydro_2021):
    # list of solar capcities
    capacities=list(solar_data.keys())
    results=[]

    # perfomr opitimization
    for capacity in capacities:
        gap=energy_gap(capacity,hydro_2021)
        results.append((capacity,gap))

    # find the capacity with the minimum energy gap
    optimal_capacity=min(results,key=lambda x : x[1])[0]
    return optimal_capacity

# find the optimla pv capcity
optimal_pv_capacity=find_optimal_capacity(hydro_2021)
print(f"Optimal pv capacity :{optimal_pv_capacity} kW")
```

```
Optimal pv capacity :1000 kW
```

In [ ]: