DISTRIBUTIONS IN PYTHON

1) Random variable (RV):-

- Maps outcome of a sample space to a real line such that there is a unique real number corresponding to every outcome of sample space
- Eg: Coin toss experiment where outcomes [H,T] are mapped to [0,1].If outcomes are already real valued then no need to map them on real line (throw of a dice). These are examples of discrete random variables
- Continuous RV maps outcomes of a continuous phenomena to intervals in a real line (Eg: sensor readings)

2) Probability mass/density function:-

- For a discrete RV, the probability mass function assigns a probability measure to every discrete outcome of sample space
- Coin toss sample space [H T] mapped to [0 1]. X is discrete RV whose outcome can be 0 or 1
  P(X = 0) = 0.5, P(X = 1) = 0.5
- For a continuous RV, the probability density function assigns a probability measure to every interval on real line
  $\mathrm{P}(a < x < b) = \int_a^b f(x)dx \ (\text{Area under the curve of } f(x)\big)$

3) Cumulative distribution/density function:-

- This is the probability that the RV 'x' lies in the interval $-\infty < x < b$
  $\mathrm{F(b)} = \mathrm{P}(-\infty < x < b) = \int_{-\infty}^{b} f(x) dx$

Every distribution in Python has four functionalities

1. ‘rvs’: generating random numbers from a distribution
2. ‘pdf’:probability density function
3. ‘cdf’:cumulative distribution function
4. ‘ppf’: percentile point function (inverse cumulative distribution function)

Distribution root name
norm- normal distribution
binom- binomial distribution

In this lecture we are going to work with standard normal distributions i.e.
mean=0
standard deviation=1

NORMAL DISTRIBUTION

In [1]:

```
# Importing numpy for numerical operations
import numpy as np

# Importing matplotlib and seaborn for visualization
import matplotlib.pyplot as plt
import seaborn as sns


# Importing scipy.stats for statistical computations
import scipy.stats
```

Setting figure size

In [2]:
```
plt.rcParams["figure.figsize"] = (7,7)
```

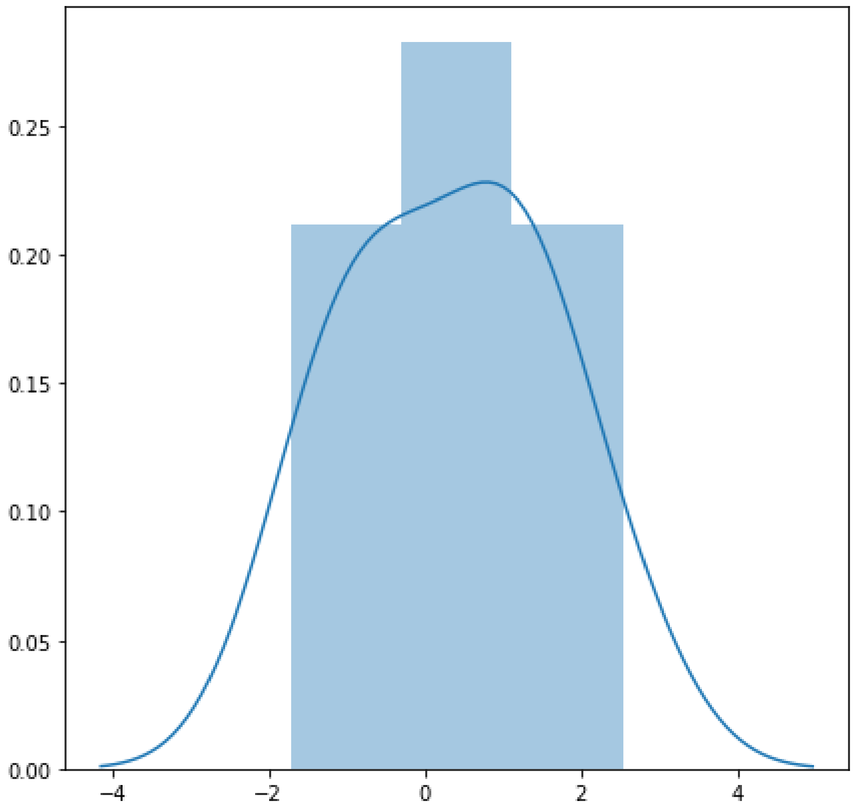
GENERATING RANDOM NUMBERS

In [3]:

```
v1= scipy.stats.norm.rvs(loc=0,scale=1,size=10)
print('Mean',v1.mean())
print('Std Dev',v1.std())
```

```
Mean 0.3253569359912848
Std Dev 1.2820175997433112
```

In [4]:
```
sns.distplot(v1)
```
Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cfe3a42048>
```



In [5]:

```python
v2= scipy.stats.norm.rvs(loc=0,scale=1,size=100)
print('Mean',v2.mean())
print('Std Dev',v2.std())
```

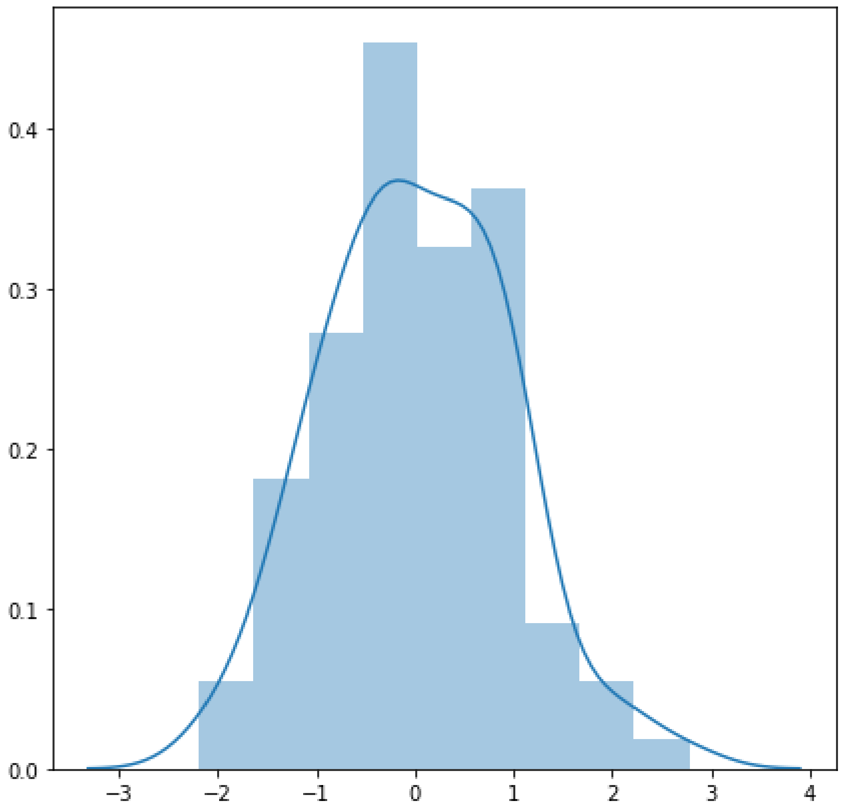
```
Mean 0.020713894103135778
Std Dev 0.9415726759437645
```

In [6]:

```python
sns.distplot(v2)
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cfe6b01648>
```

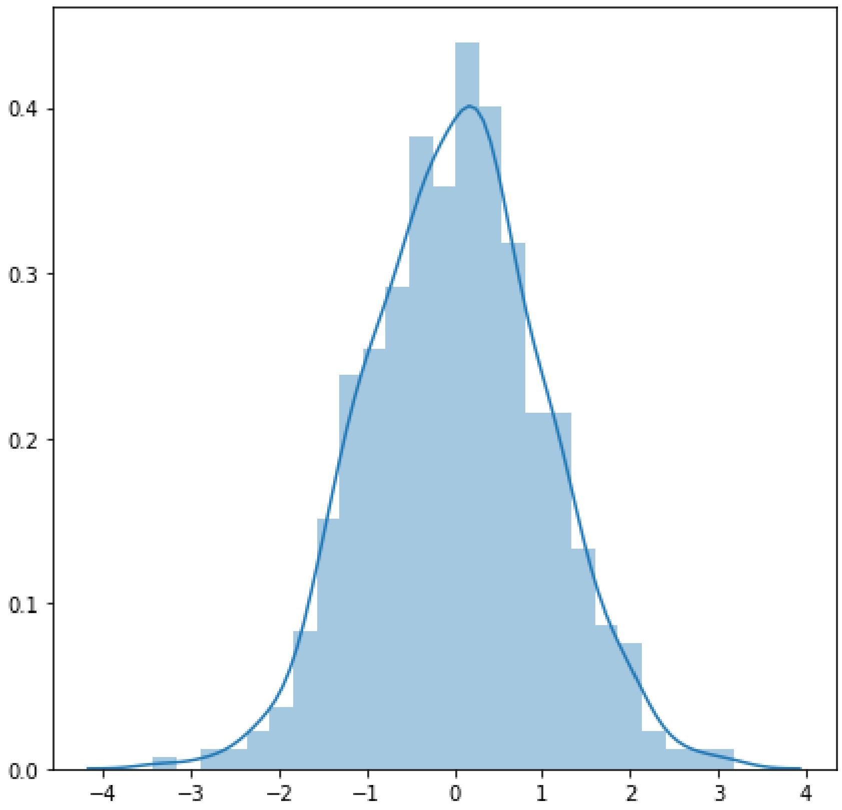


In [7]:
```
v3= scipy.stats.norm.rvs(loc=0,scale=1,size=1000)
print('Mean',v3.mean())
print('Std Dev',v3.std())
```

```
Mean 0.02103248207417688
Std Dev 0.9865003013211904
```

In [8]:
```
sns.distplot(v3)
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cfe6b87e48>
```



PROBABILITY DENSITY FUNCTION

In [9]:
```
scipy.stats.norm.pdf(-1)
```

Out[9]:
```
0.24197072451914337
```

In [10]:
```
scipy.stats.norm.pdf(np.arange(-3,-1,0.01),loc = 0,scale= 1)
```

Out[10]:
```
array([0.00443185, 0.00456659, 0.00470496, 0.00484703, 0.0049929 ,
       0.00514264, 0.00529634, 0.0054541 , 0.00561598, 0.0057821 ,
       0.00595253, 0.00612738, 0.00630673, 0.00649068, 0.00667932,
       0.00687277, 0.0070711 , 0.00727444, 0.00748287, 0.00769651,
       0.00791545, 0.00813981, 0.00836969, 0.0086052 , 0.00884645,
       0.00909356, 0.00934664, 0.0096058 , 0.00987115, 0.01014283,
       0.01042093, 0.0107056 , 0.01099694, 0.01129507, 0.01160014,
       0.01191224, 0.01223153, 0.01255811, 0.01289213, 0.0132337 ,
       0.01358297, 0.01394006, 0.01430511, 0.01467825, 0.01505962,
       0.01544935, 0.01584758, 0.01625445, 0.0166701 , 0.01709467,
       0.0175283 , 0.01797113, 0.01842331, 0.01888498, 0.01935628,
       0.01983735, 0.02032836, 0.02082943, 0.02134071, 0.02186237,
       0.02239453, 0.02293735, 0.02349099, 0.02405557, 0.02463127,
       0.02521822, 0.02581658, 0.02642649, 0.0270481 , 0.02768157,
       0.02832704, 0.02898466, 0.02965458, 0.03033696, 0.03103193,
       0.03173965, 0.03246027, 0.03319392, 0.03394076, 0.03470094,
       0.03547459, 0.03626187, 0.03706291, 0.03787786, 0.03870686,
       0.03955004, 0.04040755, 0.04127953, 0.04216611, 0.04306742,
       0.0439836 , 0.04491477, 0.04586108, 0.04682264, 0.04779957,
       0.04879202, 0.04980009, 0.0508239 , 0.05186358, 0.05291923,
       0.05399097, 0.0550789 , 0.05618314, 0.05730379, 0.05844094,
       0.05959471, 0.06076517, 0.06195242, 0.06315656, 0.06437766,
       0.06561581, 0.06687109, 0.06814357, 0.06943331, 0.07074039,
       0.07206487, 0.07340681, 0.07476626, 0.07614327, 0.07753789,
       0.07895016, 0.08038011, 0.08182778, 0.08329319, 0.08477636,
       0.08627732, 0.08779607, 0.08933262, 0.09088698, 0.09245913,
       0.09404908, 0.0956568 , 0.09728227, 0.09892547, 0.10058637,
       0.10226492, 0.1039611 , 0.10567483, 0.10740608, 0.10915477,
       0.11092083, 0.11270421, 0.1145048 , 0.11632253, 0.1181573 ,
       0.120009  , 0.12187754, 0.12376279, 0.12566464, 0.12758295,
       0.1295176 , 0.13146843, 0.1334353 , 0.13541806, 0.13741654,
       0.13943057, 0.14145997, 0.14350455, 0.14556413, 0.1476385 ,
       0.14972747, 0.1518308 , 0.15394829, 0.1560797 , 0.15822479,
       0.16038333, 0.16255506, 0.16473972, 0.16693704, 0.16914676,
       0.17136859, 0.17360225, 0.17584743, 0.17810384, 0.18037116,
```

CUMULATIVE DISTRIBUTION FUNCTION

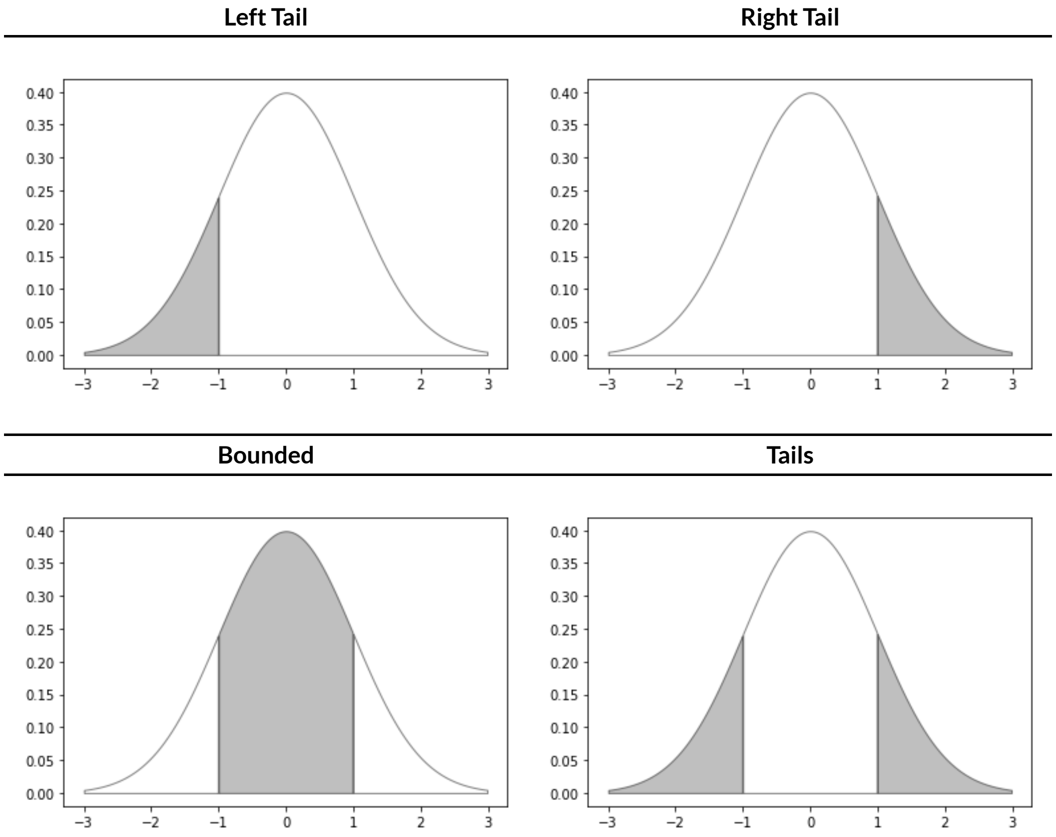
In [11]:
```
scipy.stats.norm.cdf(x=-1,loc = 0,scale= 1)
```

Out[11]:
```
0.15865525393145707
```

In [12]:
```
1-scipy.stats.norm.cdf(x=-1,loc = 0,scale= 1)
```

Out[12]:
```
0.8413447460685429
```

VISUALIZING DISTRIBUTIONS



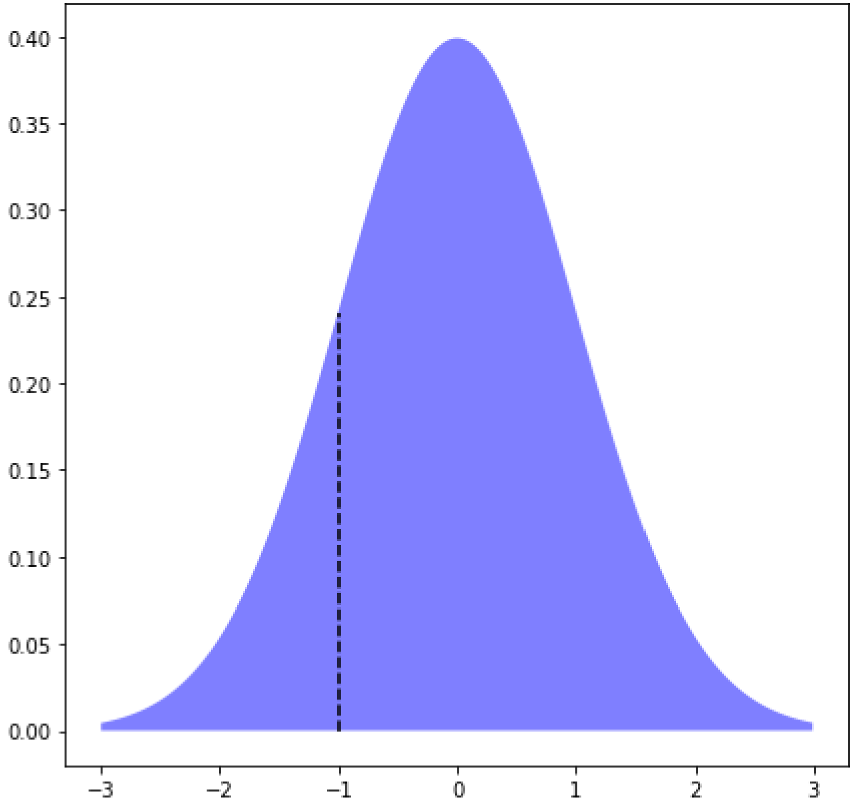
LEFT TAIL

In [13]:

```
plt.fill_between(x=np.arange(-3,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,3,0.01)),
                 facecolor='blue',
                 alpha=0.5)
plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
```

Out[13]:

```
<matplotlib.collections.LineCollection at 0x1cfe6c49388>
```



```
In [14]: scipy.stats.norm.cdf(x=-1,loc = 0,scale= 1)

Out[14]: 0.15865525393145707
```

cdf to left of -1 i.e. Z<=-1

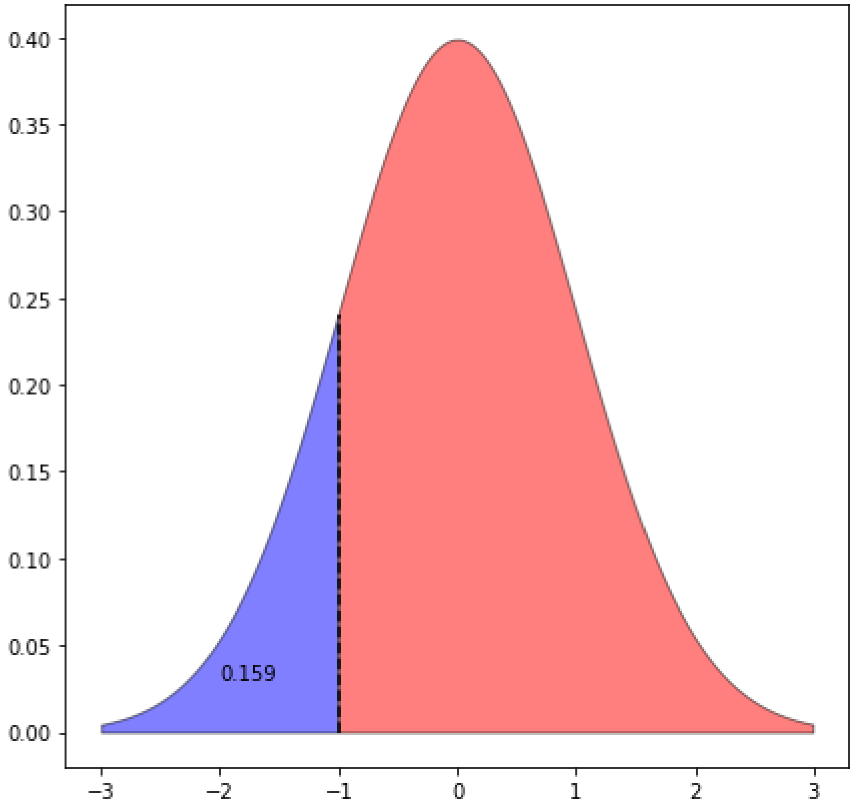
```
In [15]: prob_under_neg1 =scipy.stats.norm.cdf(x=-1,loc = 0,scale= 1)
         print(prob_under_neg1)

0.15865525393145707
```

Plotting the cdf value on the normal distribution

In [16]:
```
plt.fill_between(x=np.arange(-3,-1,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,-1,0.01)) ,
                 facecolor='blue',edgecolor='black',
                 alpha=0.5)
plt.fill_between(x=np.arange(-1,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-1,3,0.01)) ,
                 facecolor='red',edgecolor='black',
                 alpha=0.5)
plt.text(x=-2, y=0.03, s= round(prob_under_neg1,3))
plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
```

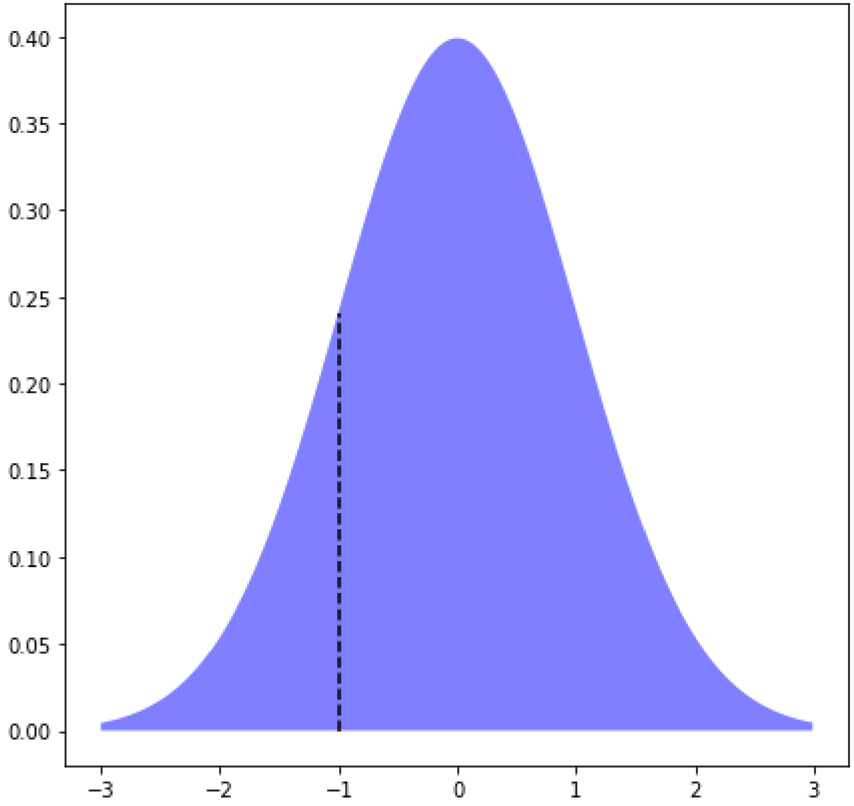
Out[16]:
```
<matplotlib.collections.LineCollection at 0x1cfe6d08e88>
```



RIGHT TAIL

In [17]:
```
plt.fill_between(x=np.arange(-3,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,3,0.01)) ,
                 facecolor='blue',
                 alpha=0.5)
plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
```

Out[17]:
```
<matplotlib.collections.LineCollection at 0x1cfe6cce408>
```



cdf to right of -1 i.e. Z>=-1

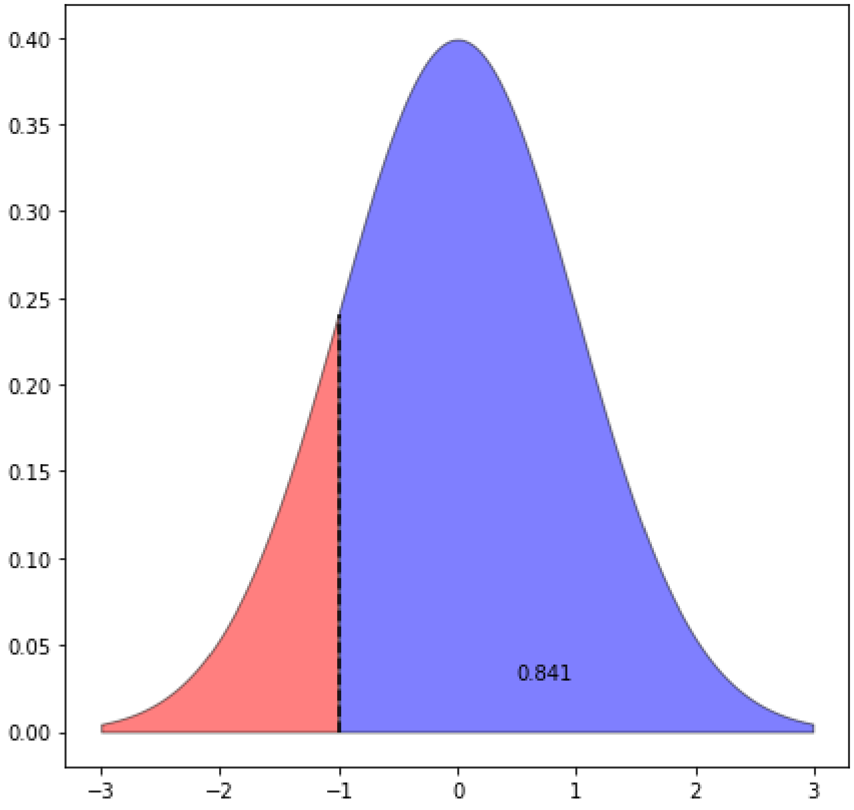
In [18]:

```
prob_over_neg1 =1-prob_under_neg1
print(prob_over_neg1)
```

```
0.8413447460685429
```

In [19]:
```
plt.fill_between(x=np.arange(-3,-1,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,-1,0.01)) ,
                 facecolor='red',edgecolor='black',
                 alpha=0.5)
plt.fill_between(x=np.arange(-1,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-1,3,0.01)) ,
                 facecolor='blue',edgecolor='black',
                 alpha=0.5)
plt.text(x=0.5, y=0.03, s= round(prob_over_neg1,3))
plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
```

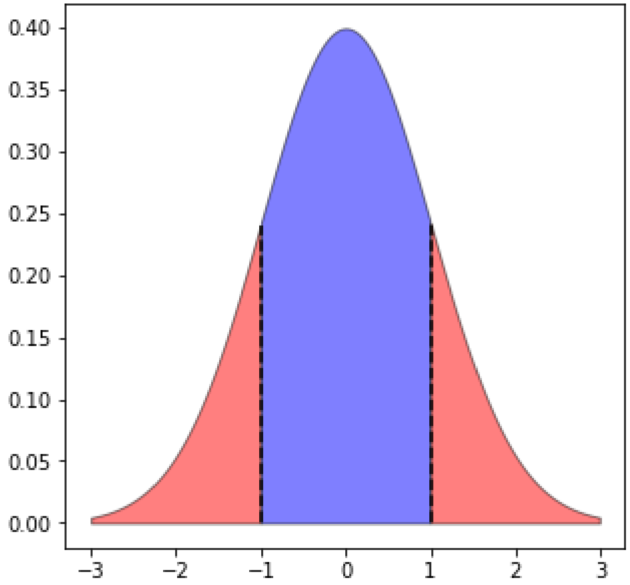
Out[19]:
```
<matplotlib.collections.LineCollection at 0x1cfe6dffdc8>
```



BOUNDED

In [20]:
```
plt.rcParams["figure.figsize"] = (5,5)
plt.fill_between(x=np.arange(-3,-1,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,-1,0.01)) ,
                 facecolor='red',edgecolor='black',
                 alpha=0.5)
plt.fill_between(x=np.arange(-1,1,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-1,1,0.01)) ,
                 facecolor='blue',edgecolor='black',
                 alpha=0.5)
plt.fill_between(x=np.arange(1,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(1,3,0.01)) ,
                 facecolor='red',edgecolor='black',
                 alpha=0.5)
plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
plt.vlines(x=1,ymin=0,ymax=0.24,linestyles='dashed')
```

Out[20]:
```
<matplotlib.collections.LineCollection at 0x1cfe6e836c8>
```



In [21]:

```
prob_over_pos1 =1-scipy.stats.norm.cdf(x=1,loc = 0,scale= 1)
print(prob_over_pos1)
print(prob_under_neg1)
between_prob=1-(prob_under_neg1+prob_over_pos1)
print(between_prob)
```

```
0.15865525393145707
0.15865525393145707
0.6826894921370859
```

TAILS

In [22]:
```
tails_prob=prob_under_neg1+prob_over_pos1
print(tails_prob)
```

```
0.31731050786291415
```

INVERSE CUMULATIVE DISTRIBUTION FUNCTION

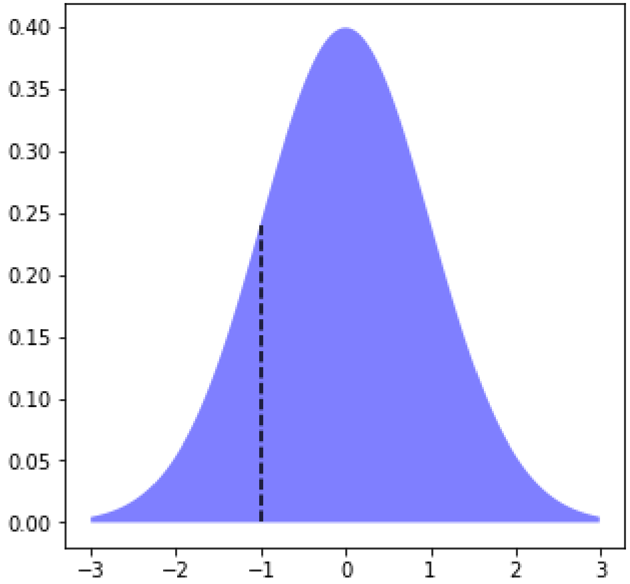
In [23]:

```
plt.fill_between(x=np.arange(-3,3,0.01),
                 y1= scipy.stats.norm.pdf(np.arange(-3,3,0.01)) ,
                 facecolor='blue',
                 alpha=0.5)

plt.vlines(x=-1,ymin=0,ymax=0.24,linestyles='dashed')
```

Out[23]:

```
<matplotlib.collections.LineCollection at 0x1cfe6e3fc88>
```



In [24]:
```
q_val=scipy.stats.norm.cdf(x=-1,loc = 0,scale= 1)
print(q_val)
```

```
0.15865525393145707
```

In [25]:
```
scipy.stats.norm.ppf(q=q_val,loc = 0,scale= 1)
```

Out[25]:
```
-1.0
```

BINOMIAL DISTRIBUTION

- No. of successes in 10 tosses of a fair coin
- n- no.of tosses
- p- probability of success

In [26]:
```
scipy.stats.binom.rvs(n=10,p=0.5)
```

Out[26]:
```
5
```

A coin is tossed 10 times and the experiment is repeated for 5 times.In each experiment the number of successes are recorded

In [27]:
```
scipy.stats.binom.rvs(size=5,n=10,p=0.5,random_state=0)
```

Out[27]:
```
array([5, 6, 5, 5, 5])
```

Now let us repeat the experiment for 15 times

In [28]:
```
scipy.stats.binom.rvs(size=15,n=10,p=0.5,random_state=0)
```

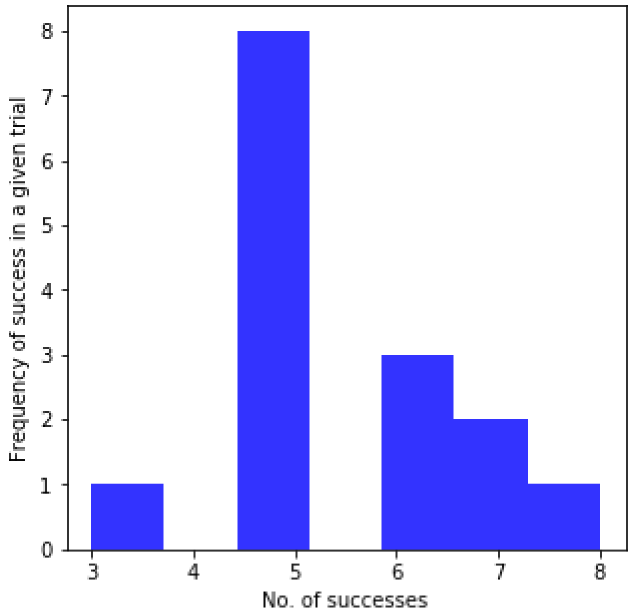
Out[28]:
```
array([5, 6, 5, 5, 5, 6, 5, 7, 8, 5, 6, 5, 5, 7, 3])
```

Let us visualize this distribution

In [29]:
```
ax=sns.distplot(scipy.stats.binom.rvs(size=15,n=10,p=0.5,random_state=0),kde=False,hist_
kws={"color": 'b','alpha':0.8})
# default color is blue

# Labels for x and y axis
ax.set(xlabel='No. of successes',ylabel='Frequency of success in a given trial')
```

Out[29]:
```
[Text(0, 0.5, 'Frequency of success in a given trial'),
 Text(0.5, 0, 'No. of successes')]
```



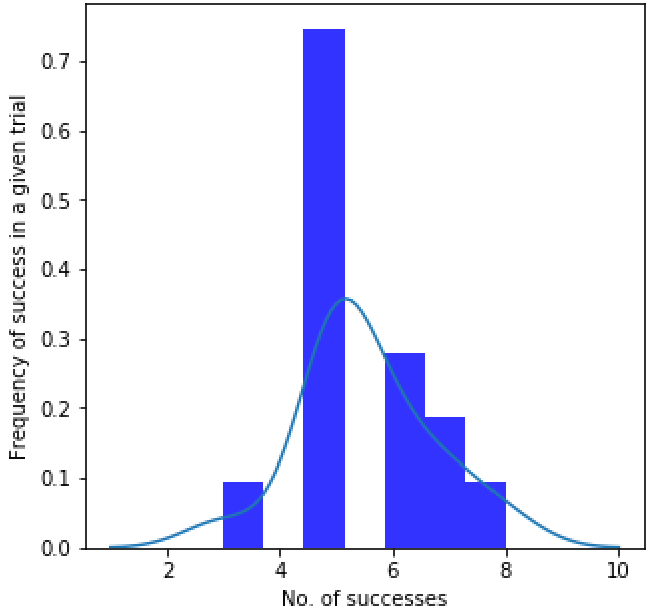
In [30]:

```
ax=sns.distplot(scipy.stats.binom.rvs(size=15,n=10,p=0.5,random_state=0),kde=True,hist_k
ws={"color": 'b','alpha':0.8})
# default color is blue

# Labels for x and y axis
ax.set(xlabel='No. of successes',ylabel='Frequency of success in a given trial')
```

Out[30]:

```
[Text(0, 0.5, 'Frequency of success in a given trial'),
 Text(0.5, 0, 'No. of successes')]
```

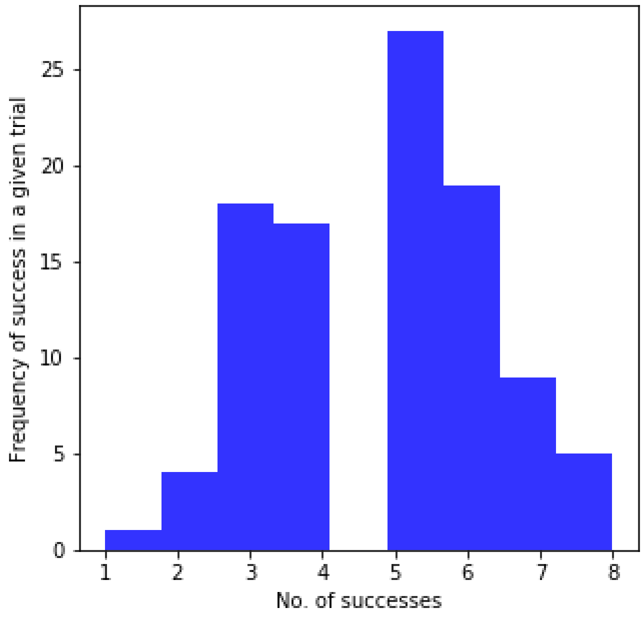


Now increase the no. of experiments to 100 and then to 500

In [31]:
```
ax=sns.distplot(scipy.stats.binom.rvs(size=100,n=10,p=0.5,random_state=0),kde=False,hist
_kws={"color": 'b','alpha':0.8})
# default color is blue

# Labels for x and y axis
ax.set(xlabel='No. of successes',ylabel='Frequency of success in a given trial')
```

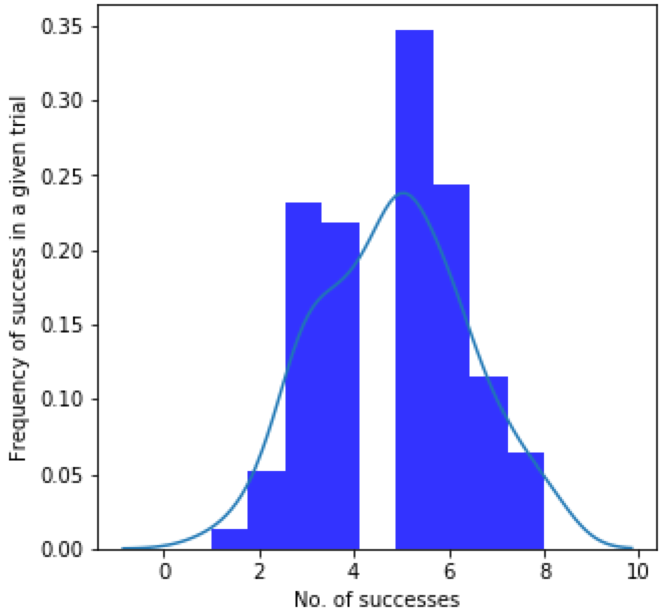
Out[31]:
```
[Text(0, 0.5, 'Frequency of success in a given trial'),
 Text(0.5, 0, 'No. of successes')]
```



In [32]:
```
ax=sns.distplot(scipy.stats.binom.rvs(size=100,n=10,p=0.5,random_state=0),kde=True,hist_
kws={"color": 'b','alpha':0.8})
# default color is blue

# Labels for x and y axis
ax.set(xlabel='No. of successes',ylabel='Frequency of success in a given trial')
```

Out[32]:
```
[Text(0, 0.5, 'Frequency of success in a given trial'),
 Text(0.5, 0, 'No. of successes')]
```



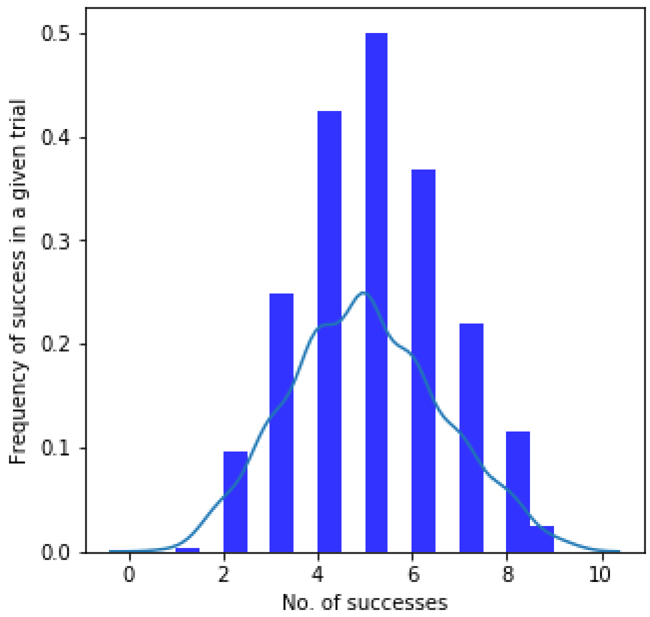
In [33]:

```
ax=sns.distplot(scipy.stats.binom.rvs(size=500,n=10,p=0.5,random_state=0),kde=True,hist_
kws={"color": 'b','alpha':0.8})
# default color is blue

# Labels for x and y axis
ax.set(xlabel='No. of successes',ylabel='Frequency of success in a given trial')
```

Out[33]:

```
[Text(0, 0.5, 'Frequency of success in a given trial'),
 Text(0.5, 0, 'No. of successes')]
```



PROBABILITY MASS FUNCTION

Probability of seeing a given number of success

Probability of seeing exactly nine heads i.e.X=9

In [34]:
```
scipy.stats.binom.pmf(n=20,p=0.5,k=9)
```

Out[34]:
```
0.16017913818359344
```

Probability of seeing nine or lesser heads i.e. X<=9

In [35]:
```
scipy.stats.binom.pmf(n=20,p=0.5,k=0)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=1)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=2)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=3)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=4)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=5)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=6)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=7)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=8)+\
scipy.stats.binom.pmf(n=20,p=0.5,k=9)
```

Out[35]:
```
0.4119014739990231
```

In [36]:
```
k_range=np.arange(0,10)

scipy.stats.binom.pmf(n=20,p=0.5,k=k_range)

sum(scipy.stats.binom.pmf(n=20,p=0.5,k=k_range))
```

Out[36]:
```
0.4119014739990231
```

This is the cumulative distribution function

In [37]:
```
scipy.stats.binom.cdf(n=20,p=0.5,k=9)
```

Out[37]:
```
0.41190147399902316
```