In [ ]:
```
#e(a)=p(a)*n
#a.p(A)+b.p(b)+...numerival outcomes
#expected values req for future outccomes
#range is 0 to 1
#ensuring correct expected outcome (by PFD) freq/posible outcomes of sample space
#central tendency --.mean median mode
#ex:weather forecasting intervals used
```

Axioms

p(A)>=0 p(S)=1 p(A)+p(b) = sum of prob =1

more than 1 -->>ex.1.5 bayes theorem used #counted twice

Combinatorics: #can be used to find out favorable outcomes variation : mix of permutation and combination with repetition n^p without repetition : n!/(n-p)!

Intro to probability

In [2]:
```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
#p(e) = favourable/all
#comb prob is by multiplying
#5 coin tosses
#p=10/2^n
from math import factorial
```

In [6]:
```
def coinflip_prob(n, k):
    n_choose_k = factorial(n)/(factorial(k)*factorial(n-k))
```

```
    return n_choose_k/2**n
```

In [7]:
```
coinflip_prob(5, 3) #calling the function to calc prob #getting 3 heads
```

Out[7]:
```
0.3125
```

In [8]:
```
coinflip_prob(100, 7)
```

Out[8]:
```
1.2627738903068384e-20
```

In [9]:
```
#values from 0 to n-1 range #h is the heads
[coinflip_prob(5, h) for h in range(6)] #here we get 0,1,2,3,4,5
```

Out[9]:
```
[0.03125, 0.15625, 0.3125, 0.3125, 0.15625, 0.03125]
```

In [11]:
```
#factorial() not defined for negative values
```

In [ ]:
```
#seed va
#random.seed
```

In [26]:
```
ns = np.array([2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]) #exponential increase of
#no.of trials
```

In [27]:
```
np.random.seed(42) #seed value is passed to get different random numbers
```

In [28]:
```
np.random.binomial(1, 0.5) #1st parameter is no.of trials , 2nd is prbabil of each trial,
#3rd para is size
```

Out[28]:
```
0
```

In [29]:
```
heads_count = [np.random.binomial(n, 0.5) for n in ns]
heads_count
```

Out[29]:
```
[2, 3, 4, 6, 13, 27, 54, 136, 258, 509, 1038, 2094]
```

In [22]:
```
heads_count1 = [np.random.binomial(n, 0.7) for n in ns]
heads_count1
```

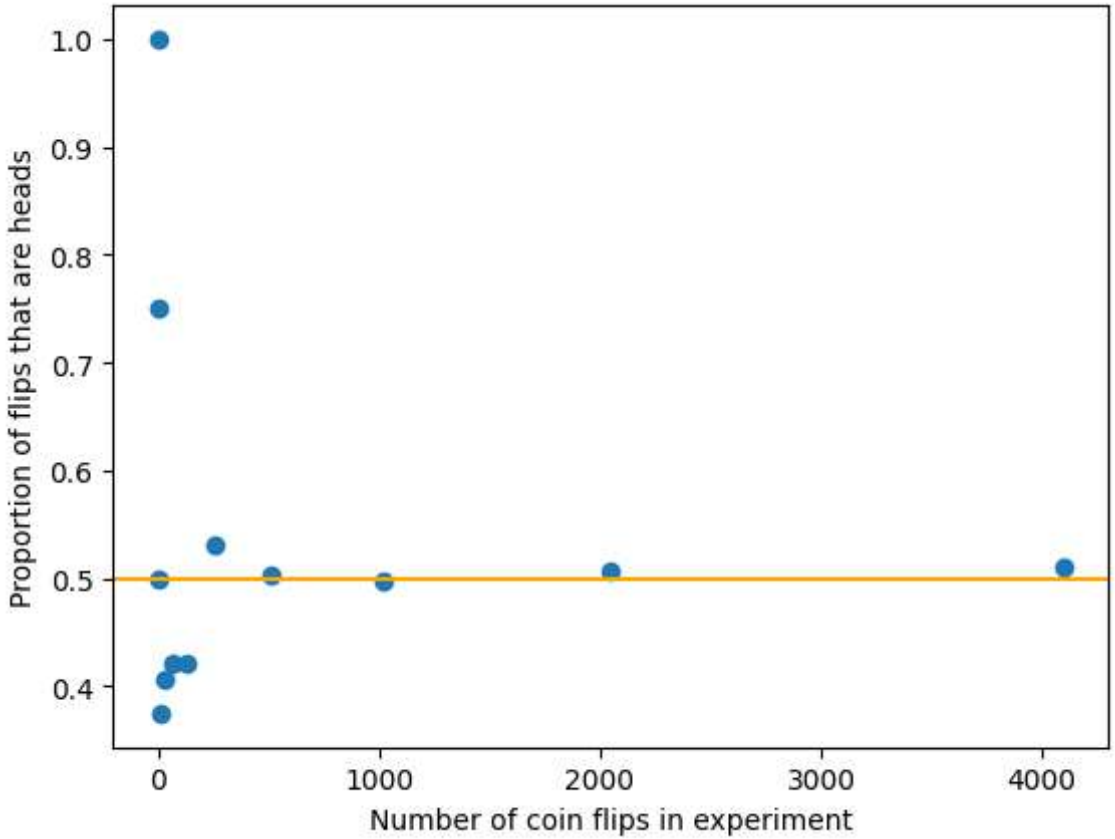
Out[22]:
```
[2, 3, 7, 9, 22, 46, 91, 180, 346, 715, 1427, 2833]
```

In [30]:
```
proportion_heads = heads_count/ns #2/2=1 means 100% probability, 3/4, 6/8,...>heads/trial
proportion_heads
```

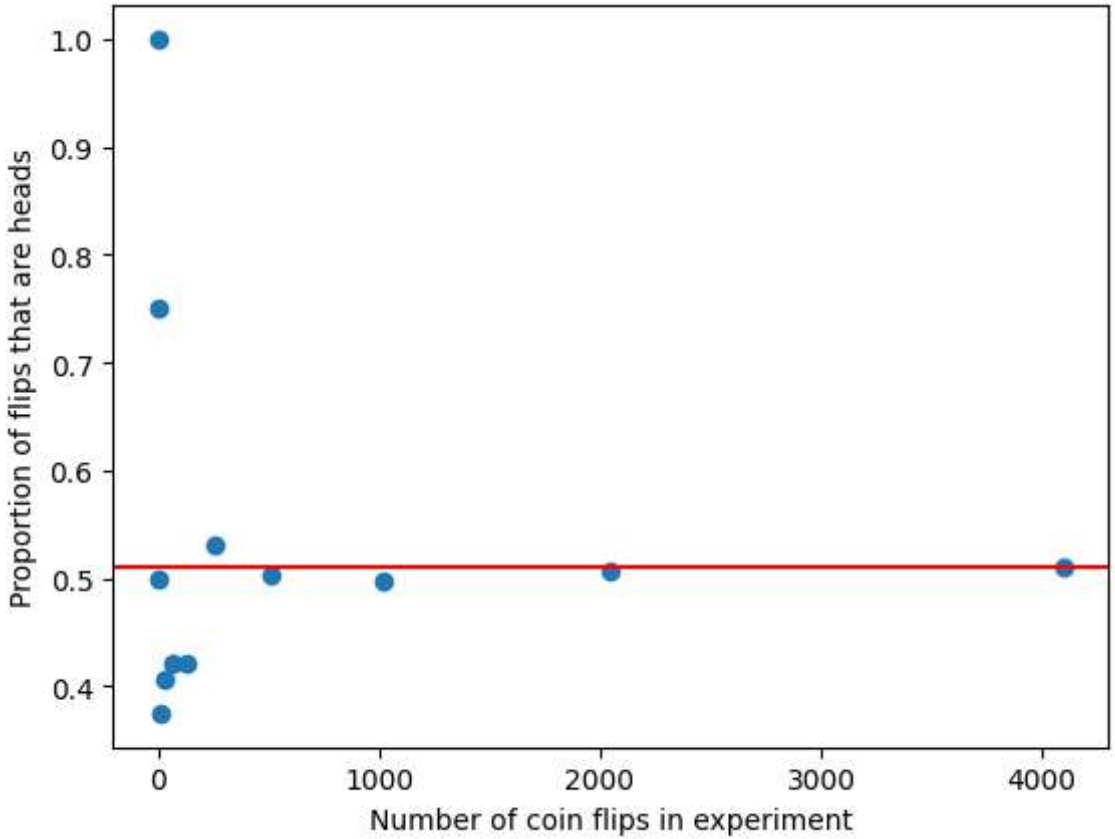
Out[30]:
```
array([1.        , 0.75      , 0.5       , 0.375     , 0.40625   ,
       0.421875  , 0.421875  , 0.53125   , 0.50390625, 0.49707031,
       0.50683594, 0.51123047])
```

In [ ]:
```
#plotting a graph
#subplot: can hANdle new data
#fig:frame axis:to plot data
```

In [32]:
```
fig, ax= plt.subplots()
plt.xlabel('Number of coin flips in experiment')
plt.ylabel('Proportion of flips that are heads')
plt.axhline(0.5, color='orange')
_=ax.scatter(ns, proportion_heads)
```



In [34]:
```
fig, ax= plt.subplots()
plt.xlabel('Number of coin flips in experiment')
plt.ylabel('Proportion of flips that are heads')
plt.axhline(0.51123047, color='red') #0.511 is the proportion
_=ax.scatter(ns, proportion_heads)
```



In [35]:
```
#5 coin flip exp:
# n=1000
n_experiments = 1000
heads_count = np.random.binomial(5, 0.5, n_experiments)
heads_count
```

Out[35]:
```
array([2, 2, 2, 3, 2, 3, 3, 1, 3, 1, 1, 4, 4, 3, 2, 1, 3, 2, 1, 2, 1, 4,
       2, 3, 2, 3, 3, 1, 5, 3, 4, 4, 3, 4, 1, 2, 1, 2, 2, 2, 4, 2, 2, 3,
       1, 3, 1, 5, 3, 2, 0, 4, 3, 3, 3, 1, 2, 1, 4, 3, 2, 1, 2, 2, 3, 3,
       4, 2, 1, 3, 3, 3, 3, 2, 3, 2, 0, 1, 1, 3, 2, 3, 4, 2, 2, 3, 2, 1,
       2, 1, 4, 3, 3, 4, 3, 1, 4, 3, 3, 4, 2, 1, 2, 2, 4, 4, 0, 3, 2, 2,
       1, 2, 4, 2, 3, 3, 2, 5, 4, 2, 2, 2, 2, 1, 3, 3, 1, 2, 4, 2, 1, 2,
       5, 2, 3, 3, 2, 3, 2, 3, 3, 3, 1, 4, 2, 1, 1, 3, 3, 0, 3, 2, 3, 1,
       3, 2, 4, 1, 2, 1, 4, 4, 2, 3, 4, 3, 3, 2, 1, 4, 4, 3, 2, 2, 3, 4,
       4, 3, 3, 1, 1, 4, 3, 0, 1, 3, 0, 1, 3, 3, 3, 2, 3, 2, 2, 3, 3, 4,
       3, 3, 1, 2, 2, 2, 5, 2, 4, 3, 3, 3, 3, 2, 2, 3, 2, 0, 3, 1, 4, 4,
       4, 2, 0, 4, 2, 4, 4, 4, 2, 2, 4, 2, 1, 3, 4, 3, 3, 1, 3, 5, 1, 3,
       4, 3, 3, 3, 2, 2, 3, 3, 4, 4, 3, 3, 3, 3, 3, 3, 4, 2, 2, 1, 3, 1,
       2, 3, 2, 3, 0, 1, 4, 2, 1, 3, 3, 2, 3, 1, 1, 3, 3, 3, 3, 5, 3, 2,
       3, 2, 2, 1, 0, 4, 4, 3, 2, 1, 1, 2, 3, 3, 3, 2, 4, 3, 3, 3, 2, 2,
       2, 3, 0, 1, 1, 1, 4, 3, 2, 1, 2, 2, 1, 2, 2, 3, 3, 1, 2, 3, 3, 4,
       3, 1, 1, 3, 0, 3, 4, 3, 2, 3, 2, 3, 4, 2, 4, 4, 2, 1, 1, 0, 1, 3,
       1, 2, 4, 0, 4, 2, 1, 3, 3, 4, 3, 3, 2, 1, 3, 3, 5, 2, 2, 3, 2, 4,
       4, 2, 3, 3, 1, 4, 3, 4, 2, 4, 2, 0, 4, 1, 2, 4, 4, 3, 3, 2, 2, 2,
       3, 3, 3, 3, 1, 2, 1, 3, 2, 4, 2, 1, 1, 3, 3, 1, 1, 3, 1, 4, 3, 1,
       1, 5, 2, 2, 4, 4, 5, 3, 2, 1, 3, 3, 2, 4, 1, 2, 0, 2, 1, 1, 1, 3,
       3, 3, 4, 2, 2, 4, 2, 4, 0, 5, 1, 4, 3, 5, 1, 3, 5, 3, 3, 3, 2, 3,
       3, 4, 1, 2, 4, 4, 2, 3, 2, 2, 2, 2, 3, 1, 5, 5, 3, 3, 2, 4, 3, 1,
       4, 4, 4, 3, 3, 2, 4, 4, 1, 0, 2, 3, 5, 1, 3, 2, 5, 4, 4, 2, 2, 2,
       1, 4, 4, 5, 5, 3, 3, 4, 4, 2, 2, 1, 4, 3, 2, 3, 3, 2, 1, 3, 3, 3,
       3, 4, 3, 3, 4, 2, 1, 0, 3, 3, 3, 2, 1, 0, 2, 3, 2, 2, 4, 2, 3, 3,
       2, 3, 4, 4, 1, 4, 2, 2, 2, 5, 2, 2, 3, 2, 1, 1, 1, 1, 1, 3, 1, 2,
       4, 2, 3, 1, 2, 1, 1, 2, 1, 1, 1, 2, 2, 2, 3, 3, 1, 3, 3, 1, 4, 4,
       1, 2, 3, 3, 1, 2, 2, 2, 3, 2, 2, 3, 1, 2, 3, 4, 3, 3, 1, 2, 3, 2,
       2, 2, 0, 2, 2, 2, 1, 4, 3, 3, 3, 2, 1, 3, 3, 3, 2, 1, 2, 2, 3, 3,
       2, 5, 3, 2, 1, 1, 2, 1, 1, 2, 1, 4, 1, 3, 2, 5, 1, 2, 5, 4, 4, 2,
       1, 3, 4, 3, 3, 2, 3, 1, 2, 2, 3, 2, 1, 3, 2, 3, 1, 2, 3, 1, 2, 1,
       1, 5, 2, 3, 2, 3, 3, 3, 2, 2, 2, 4, 4, 4, 1, 3, 4, 1, 1, 3, 3, 4,
       1, 3, 2, 1, 1, 4, 3, 3, 2, 4, 2, 4, 2, 2, 2, 2, 3, 3, 2, 4, 2, 3,
       4, 3, 1, 4, 3, 2, 1, 3, 3, 3, 4, 3, 1, 2, 2, 3, 1, 3, 3, 4, 2, 4,
       1, 4, 1, 2, 3, 1, 2, 3, 3, 3, 3, 3, 2, 2, 1, 4, 3, 2, 2, 4, 1, 3,
       3, 3, 0, 4, 4, 3, 3, 4, 3, 1, 3, 3, 2, 3, 4, 4, 2, 1, 5, 4, 1, 4,
       4, 3, 3, 2, 1, 2, 3, 0, 2, 2, 3, 4, 2, 2, 3, 2, 2, 2, 1, 1, 2, 2,
       4, 2, 3, 3, 0, 3, 1, 4, 1, 2, 1, 5, 3, 3, 1, 3, 2, 4, 2, 2, 1, 2,
       3, 1, 3, 2, 3, 2, 2, 3, 1, 1, 4, 4, 2, 2, 3, 2, 0, 1, 3, 3, 0, 2,
       2, 3, 0, 1, 3, 1, 3, 2, 1, 2, 3, 4, 4, 2, 3, 2, 2, 4, 0, 1, 2, 0,
       1, 3, 2, 4, 4, 2, 2, 2, 3, 2, 3, 2, 3, 2, 2, 4, 3, 1, 4, 2, 4, 3,
```

```
2, 0, 2, 3, 3, 2, 1, 4, 5, 3, 1, 2, 0, 4, 1, 3, 2, 3, 3, 4, 2, 0,
1, 4, 4, 3, 3, 3, 1, 4, 2, 2, 3, 1, 1, 3, 1, 3, 2, 2, 2, 2, 3, 2,
3, 2, 3, 4, 3, 2, 0, 2, 3, 1, 2, 3, 2, 3, 1, 1, 3, 2, 3, 2, 2, 4,
3, 3, 2, 3, 2, 1, 4, 1, 4, 2, 1, 3, 4, 3, 3, 3, 3, 4, 2, 2, 2, 5,
4, 1, 3, 4, 1, 3, 4, 1, 3, 2])
```

In [45]:
```
#unique funcn used
heads, event_count = np.unique(heads_count, return_counts=True) #true indicates returning of unique
#values in a sorted array.
```

In [46]:
```
heads
```

Out[46]:
```
array([0, 1, 2, 3, 4, 5])
```

In [47]:
```
event_count #tells about how many times 0,1,2,3,4,5 has occured
```

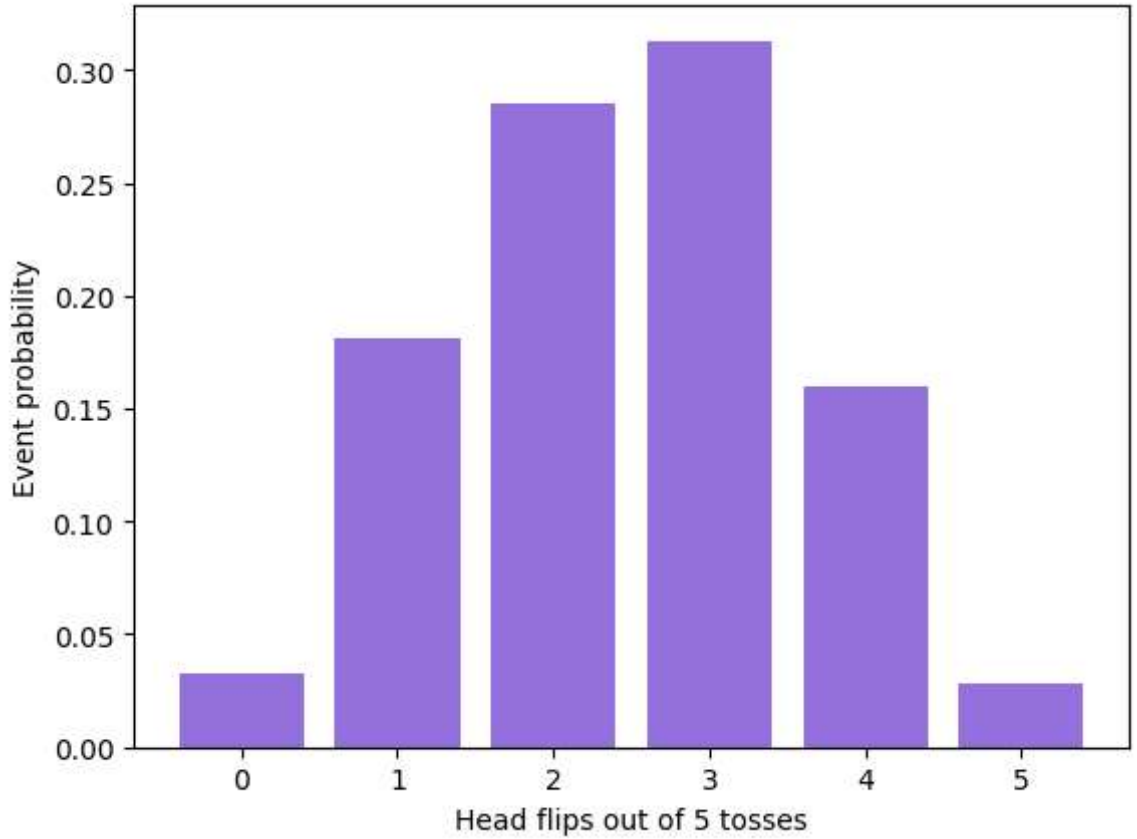
Out[47]:
```
array([ 33, 181, 285, 313, 160,  28], dtype=int64)
```

In [49]:
```
event_proba = event_count/n_experiments
event_proba
```

Out[49]:
```
array([0.033, 0.181, 0.285, 0.313, 0.16 , 0.028])
```

In [ ]:
```
#plotting a bar graph is the next step
#for categorical outcome:
```

In [77]:
```
plt.bar(heads, event_proba, color='mediumpurple')
plt.xlabel('Head flips out of 5 tosses')
_=plt.ylabel('Event probability')
```



Expected value

In [81]:
```
P=[coinflip_prob(5, x) for x in range(6)]
P
```

Out[81]:
```
[0.03125, 0.15625, 0.3125, 0.3125, 0.15625, 0.03125]
```

In [82]:
```
E = sum([P[x]*x for x in range(6)])
E
```

Out[82]:
```
2.5
```

In [ ]:
```
#heads =1 and tails=0 >>numer outcome
```

In [85]:
```
A = 1
B =0
E = sum([P[x]*x for x in range(6)])
E
```

Out[85]: 2.5

In [ ]:
```
#measures of central tendency mean median and mode has built in fun
```

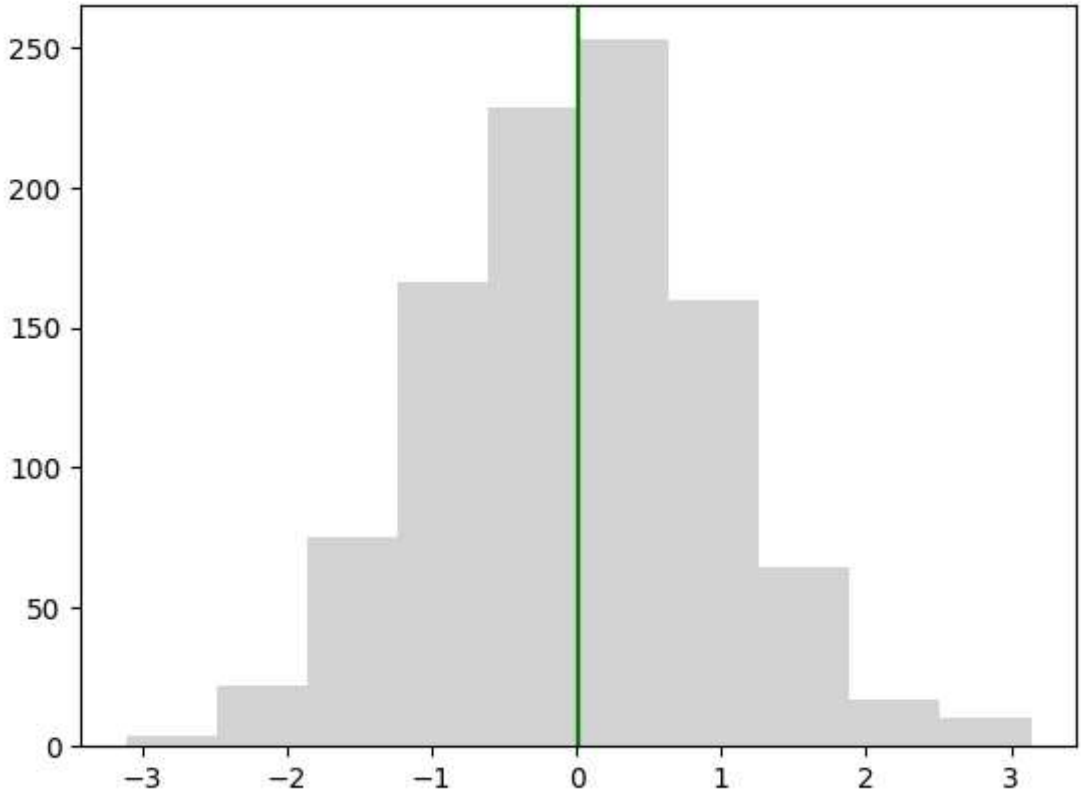
In [86]:
```
np.median(heads_count)
```

Out[86]: 3.0

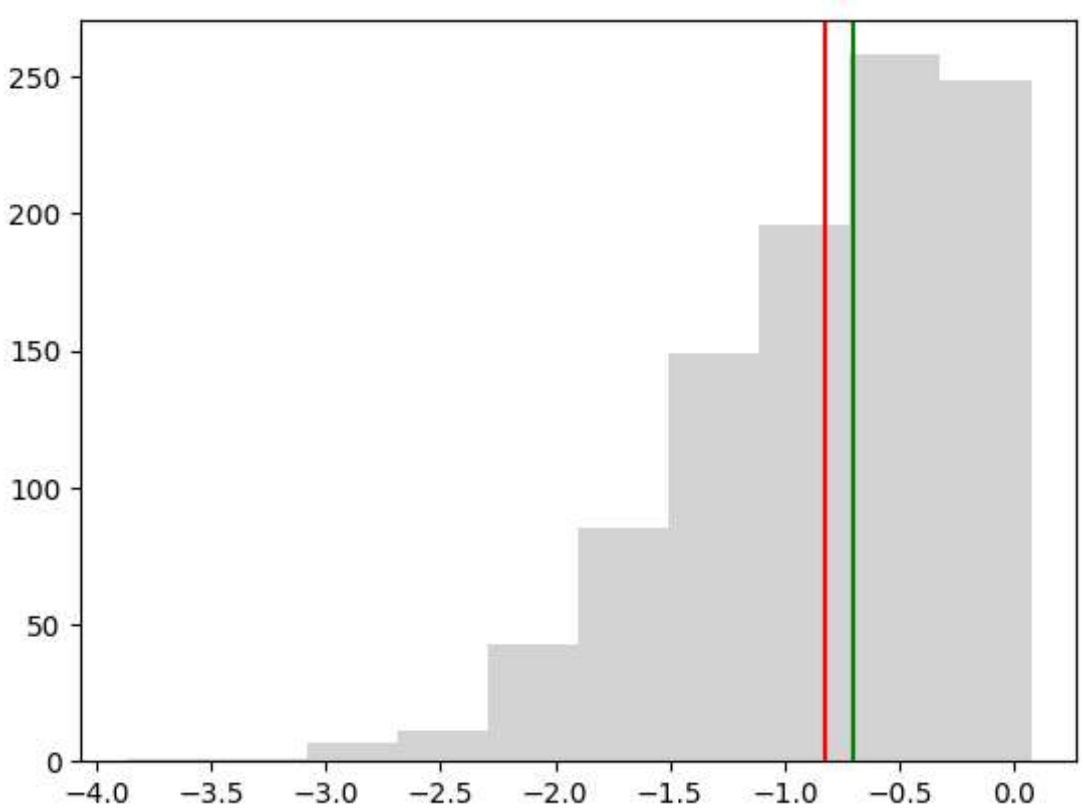
In [ ]:
```
#right skew is +ve and mean>mode and vice_versa foe left skew
#mean>median>mode
```

In [87]:
```
#skewness =0;mean~median #rvs is
#axv is for drawing a vertical line
x= st.skewnorm.rvs(0, size=1000)
#first arg is skewness : 0 has no skew >>normal distribution
#array,array of bins, patches
#mode cannot be applied on pdf can be aplied on dicrete distributions
```

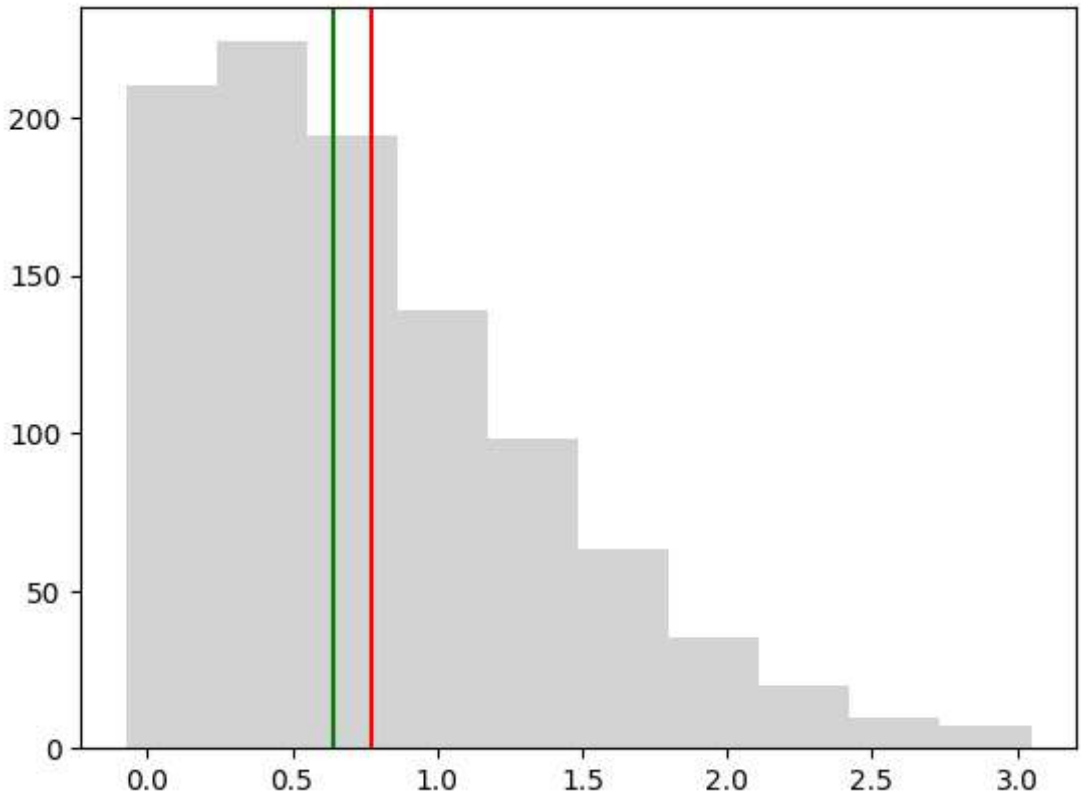
In [89]:
```
fig, ax=plt.subplots()
plt.axvline(x =np.mean(x), color= 'red')
plt.axvline(x =np.median(x), color= 'green')
_=plt.hist(x,color ='lightgray')
```



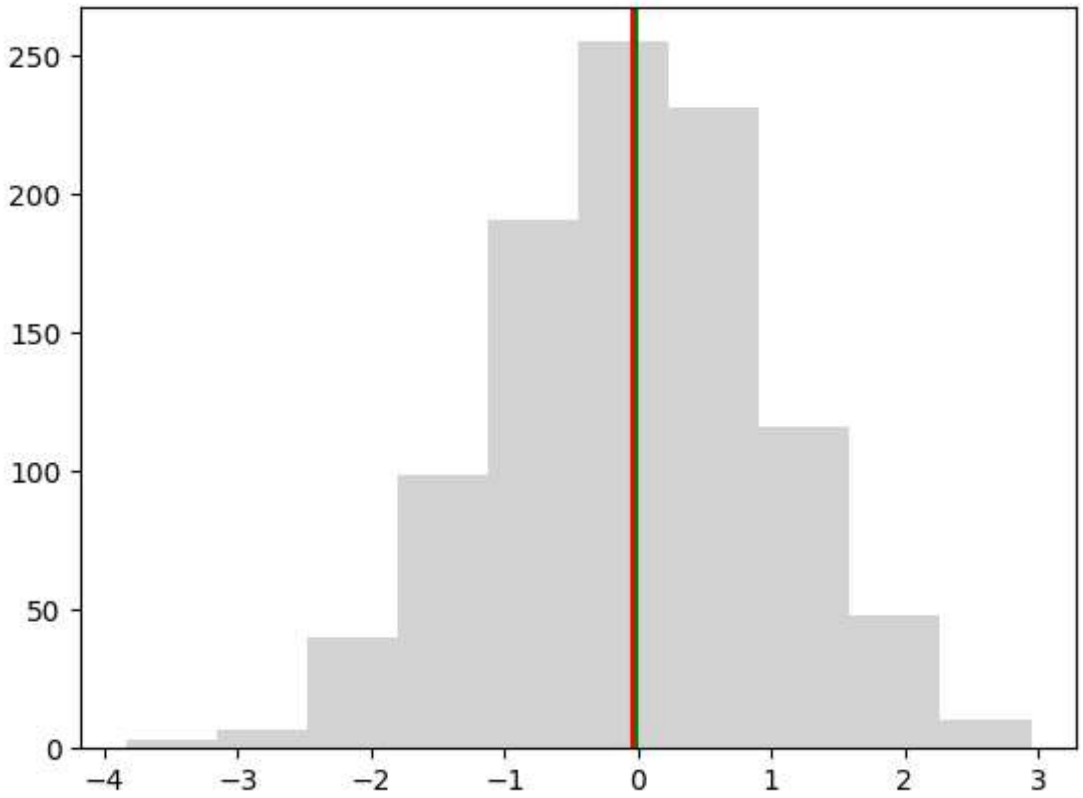
In [90]:
```
x= st.skewnorm.rvs(-20, size=1000)
fig, ax=plt.subplots()
plt.axvline(x =np.mean(x), color= 'red')
plt.axvline(x =np.median(x), color= 'green')
_=plt.hist(x,color ='lightgray') #for left skew
```



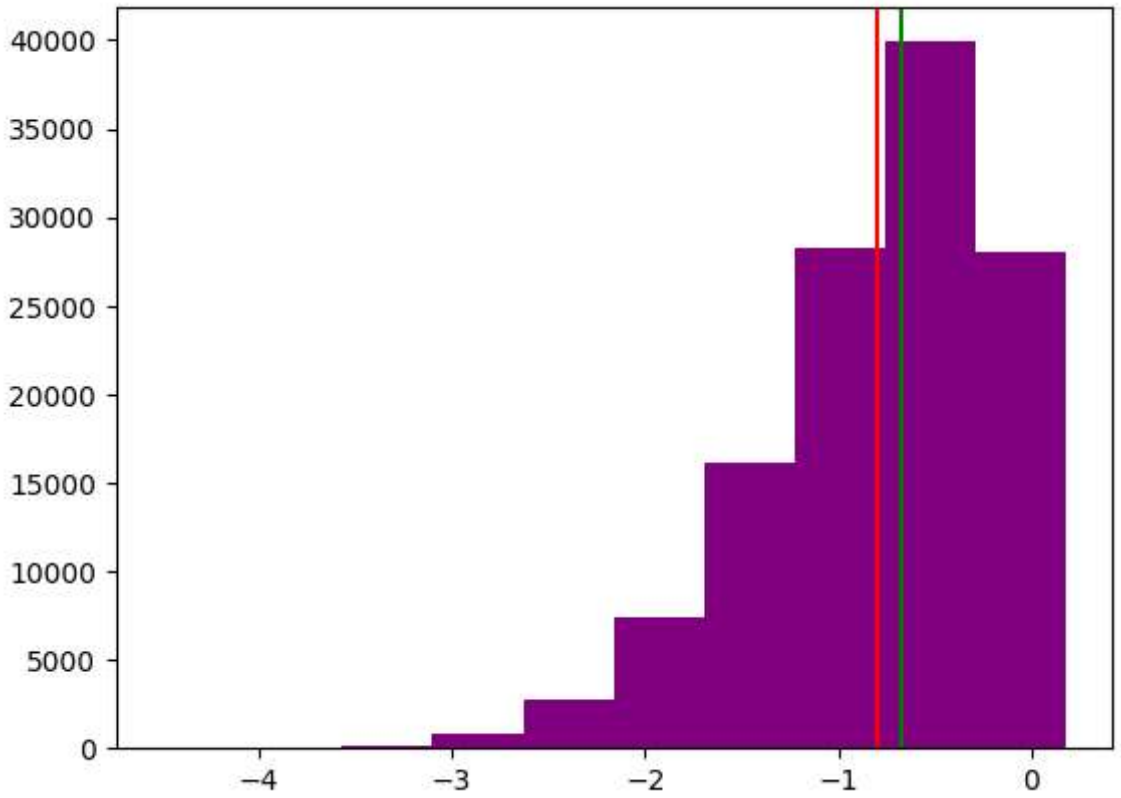
In [91]:
```
x= st.skewnorm.rvs(20, size=1000)
fig, ax=plt.subplots()
plt.axvline(x =np.mean(x), color= 'red')
plt.axvline(x =np.median(x), color= 'green')
_=plt.hist(x,color ='lightgray') #for right skew
#tail towards right
```



In [92]:
```
x= st.skewnorm.rvs(0, size=1000) #normal distribution
fig, ax=plt.subplots()
plt.axvline(x =np.mean(x), color= 'red')
plt.axvline(x =np.median(x), color= 'green')
_=plt.hist(x,color ='lightgray')
```

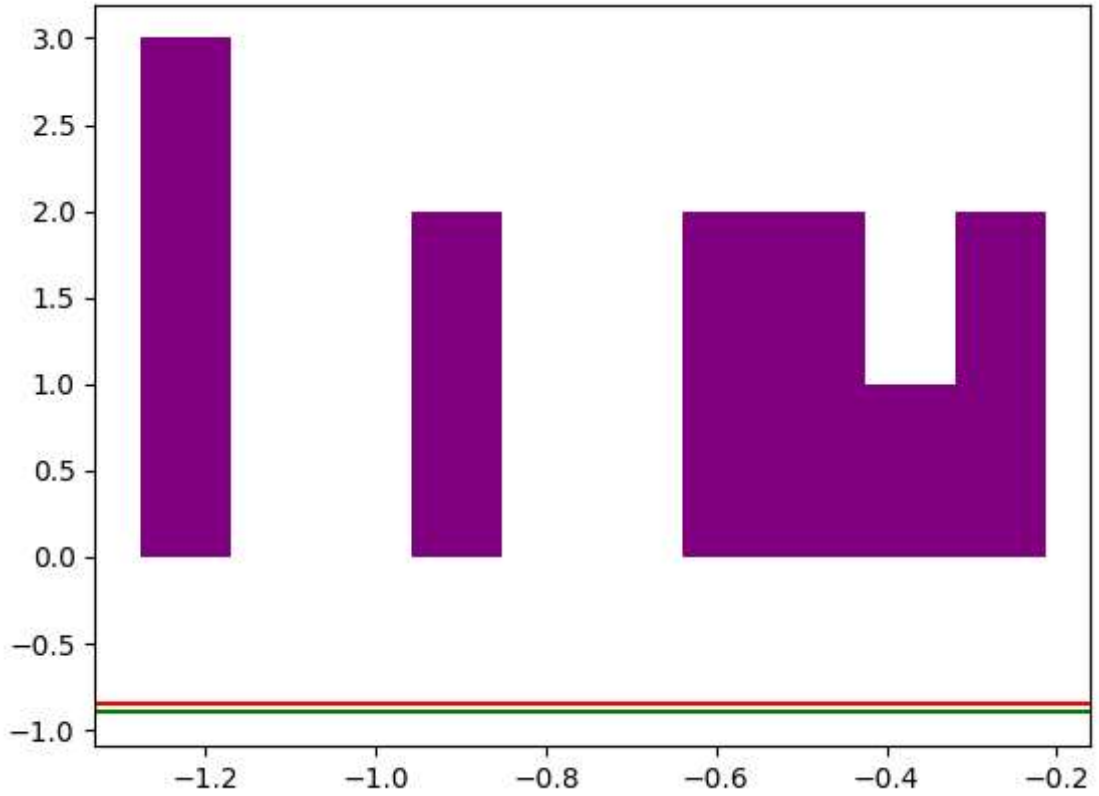


In [98]:
```
x= st.skewnorm.rvs(-20, size = 123409)
fig, ax=plt.subplots()
plt.axvline(x =np.mean(x), color= 'red')
plt.axvline(x =np.median(x), color= 'green')
_=plt.hist(x,color ='purple')
```



In [116…

```
y= st.skewnorm.rvs(-67, size = 12) #size cant be negative
#changing x to y and changing axv to axh line
#changing sizes
fig, ax=plt.subplots()
plt.axhline(y =np.mean(x), color= 'red')
plt.axhline(y =np.median(x), color= 'green')
_=plt.hist(y,color ='purple')
```



theoretical and expected

In [ ]:

Probability-2

P(A/B)=P(B/A)*P(A)/P(B) P(B)>0 P(A/B)=P(A intrn B)/P(B) conditional prob based on past data random variables(X): used to compute other events easily >> X:U-->R(MAPPING)>> U has no range every feature is a random var..can be categorical or numerical>> discrete>>countable and continuous>>intervals and can be anything >ex:salary,height prob distribution of random variable:P(x) lies between 0 and 1 mean/expectation:mu/E(x)=summation i=1 to n(xiPi) most expected value mu=0 variance:varX = summation(xi^2*Pi-mu^2) standard deviation =sqrt(varx)

Probability Distributions

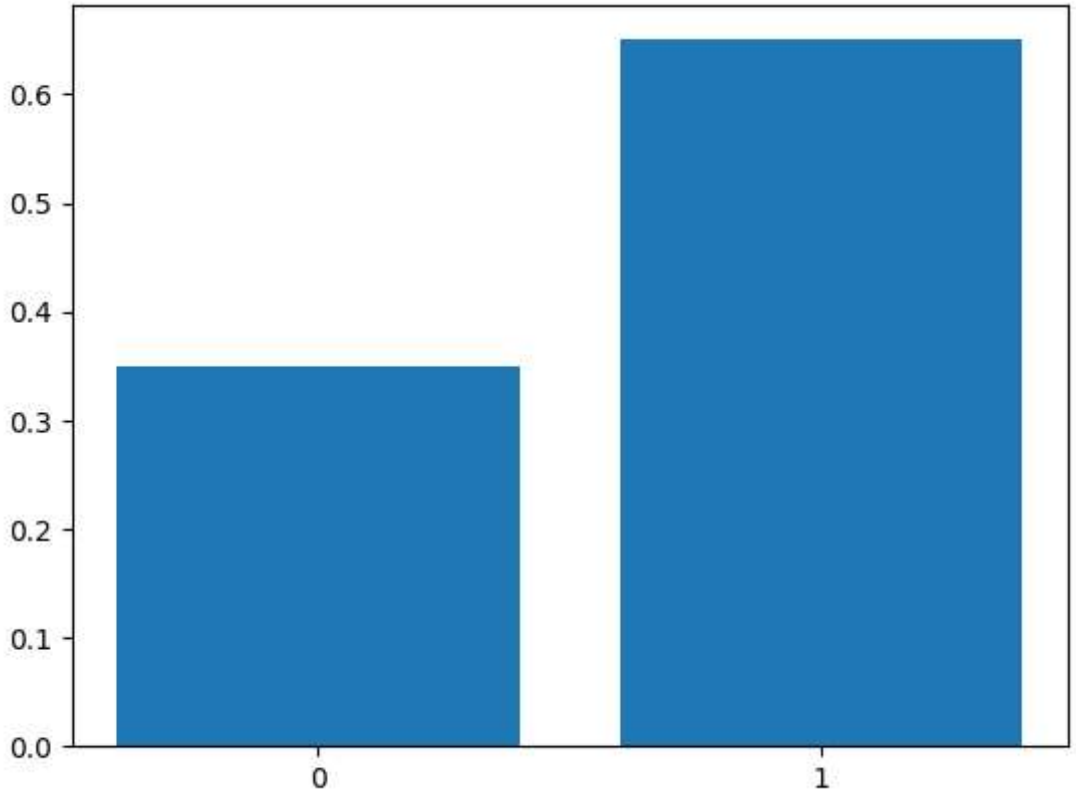
1.DISCRETE: PROB MASS FUNCTION(PMF) 2.CONTINUOUS :PROB DENSITY FUNCTION(PDF) 3.CUMULATIVE : CDF(belongs to both discrete and continuous

DISCRETE

BERNOULLI: p=1=success q=0=failure=1-p 0<p<1 p=q=1/2 equally likely

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.bar(['0','1'],[0.35,0.65]) #1st para: category #2nd para:value for category
```

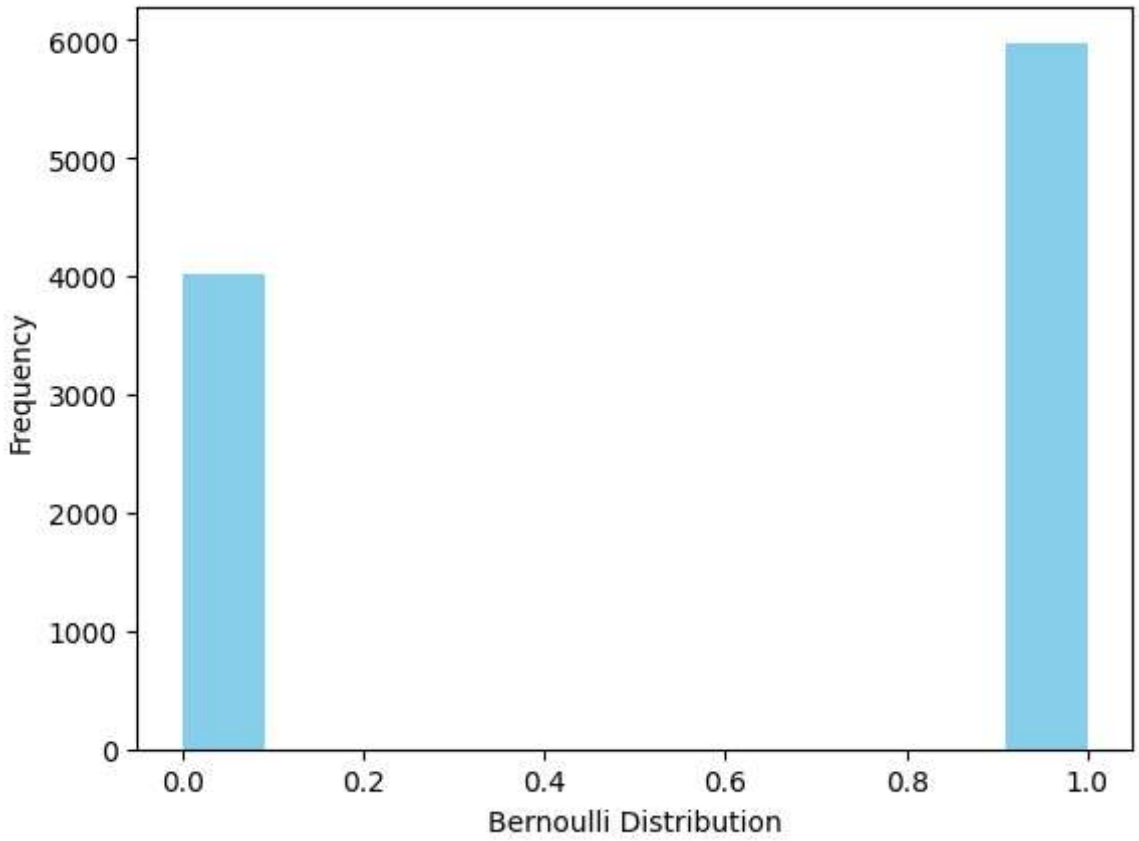
Out[3]:
```
<BarContainer object of 2 artists>
```



In [5]:
```
from scipy.stats import bernoulli
#gives sorted 1-D array >.data bern
data_bern = bernoulli.rvs(size=10000, p=0.6)
import seaborn as sns
ax=sns.distplot(data_bern,
                kde=False,
                color='skyblue',
                hist_kws={"linewidth":15,'alpha':1})
ax.set(xlabel='Bernoulli Distribution',
      ylabel='Frequency')
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_14304\4264797599.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax=sns.distplot(data_bern,
```

Out[5]: [Text(0.5, 0, 'Bernoulli Distribution'), Text(0, 0.5, 'Frequency')]

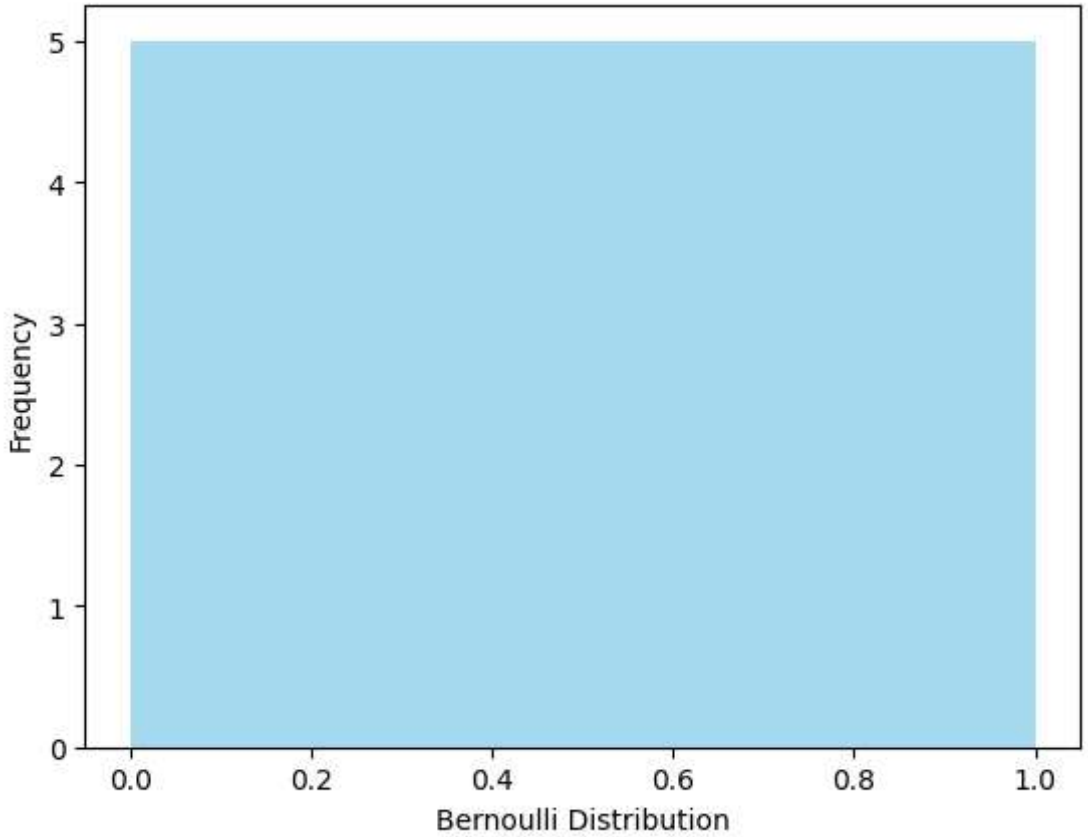


In [21]:
```
from scipy.stats import bernoulli
#gives sorted 1-D array >.data bern
```

```
data_bern = bernoulli.rvs(size=10, p=0.56)
import seaborn as sns
ax=sns.distplot(data_bern,
                kde=False,
                color='skyblue',
                hist_kws={"linewidth":-2375,'alpha':0.75})
ax.set(xlabel='Bernoulli Distribution',
      ylabel='Frequency')
#alpha is between 0 and 1
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_14304\2647963275.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax=sns.distplot(data_bern,
```

Out[21]: [Text(0.5, 0, 'Bernoulli Distribution'), Text(0, 0.5, 'Frequency')]



all trials are independent and random

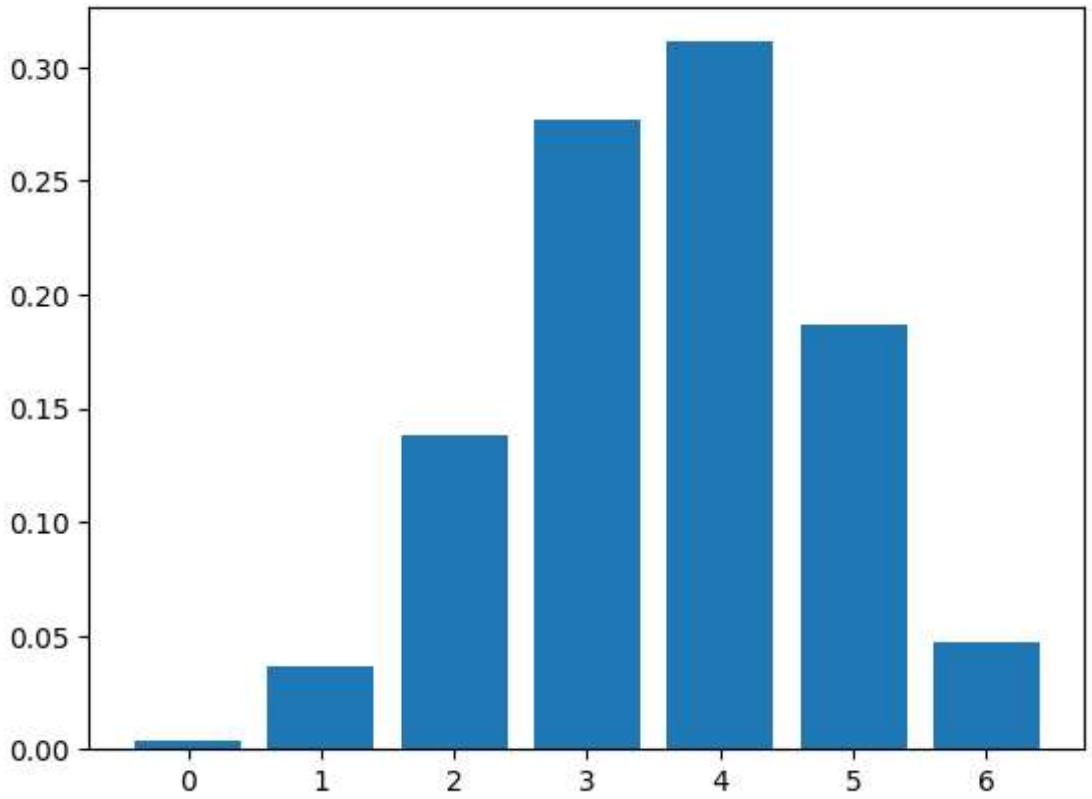
-ve binomial distribution : no fixed trials: cannot be predicted

success or failure are the outcomes

0 to n-1: range

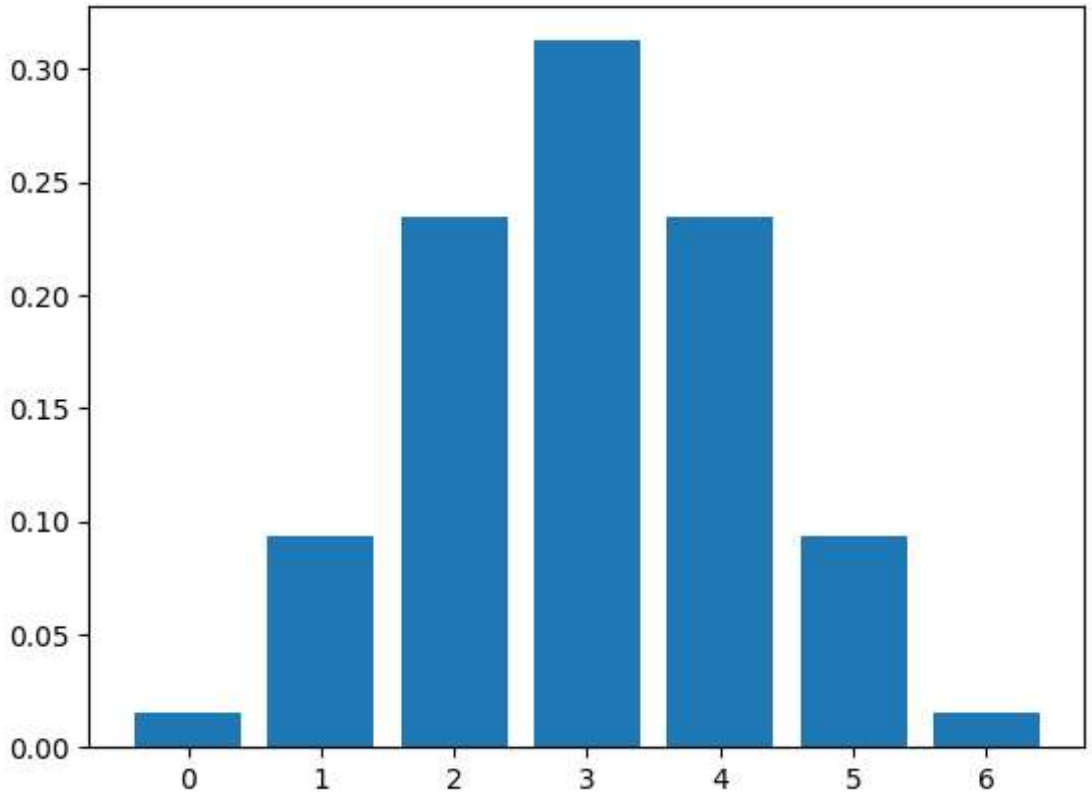
In [22]:
```
from scipy.stats import binom
```

```
import matplotlib.pyplot as plt
#setting the values of n and p
n=6
p=0.6
#defining list of r values
r_values= list(range(n+1))
#list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values]
#plotting the graph
plt.bar(r_values, dist)
plt.show()
```



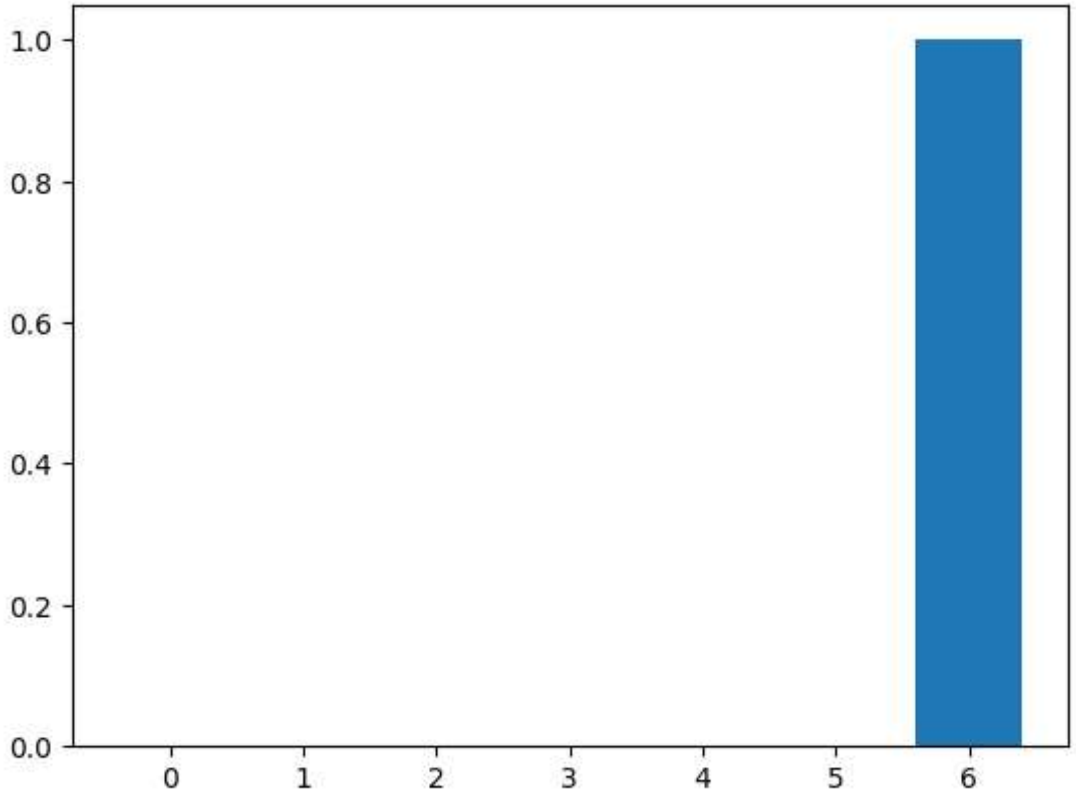
In [23]:
```
from scipy.stats import binom
import matplotlib.pyplot as plt
#setting the values of n and p
n=6
p=0.5
```

```
#defining list of r values
r_values= list(range(n+1))
#list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values]
#plotting the graph
plt.bar(r_values, dist)
plt.show() #normal distribution on changing the value of p to 0.5
```



In [26]:
```
from scipy.stats import binom
import matplotlib.pyplot as plt
#setting the values of n and p
n=6
p=1
#defining list of r values
r_values= list(range(n+1))
#list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values]
```

```
#plotting the graph
plt.bar(r_values, dist)
plt.show()
```



success for each trial-->binomial

Binomial and bernoulli on random variable

In [2]:
```
from scipy import stats
X = stats.binom(10, 0.8)
#declare X to be a binomial random variable
X.pmf(3) #givinf value as 3
```

Out[2]: 0.0007864319999999988

In [28]:
```
#P(X<=4)
X.cdf(4)
```

Out[28]: 0.006369382399999991

In [29]:
```
#E[X]
X.mean()
```

Out[29]: 8.0

In [30]:
```
#Var(x)
X.var()
```

Out[30]: 1.5999999999999996

In [31]:
```
#get a random sample from
X.rvs() #if no parameter is passed then it generates only one random sample
```

Out[31]: 5

In [3]:
```
X.rvs(10)
#get 10 random samples
```

Out[3]: array([9, 7, 10, 10, 10, 5, 7, 9, 9, 7], dtype=int64)

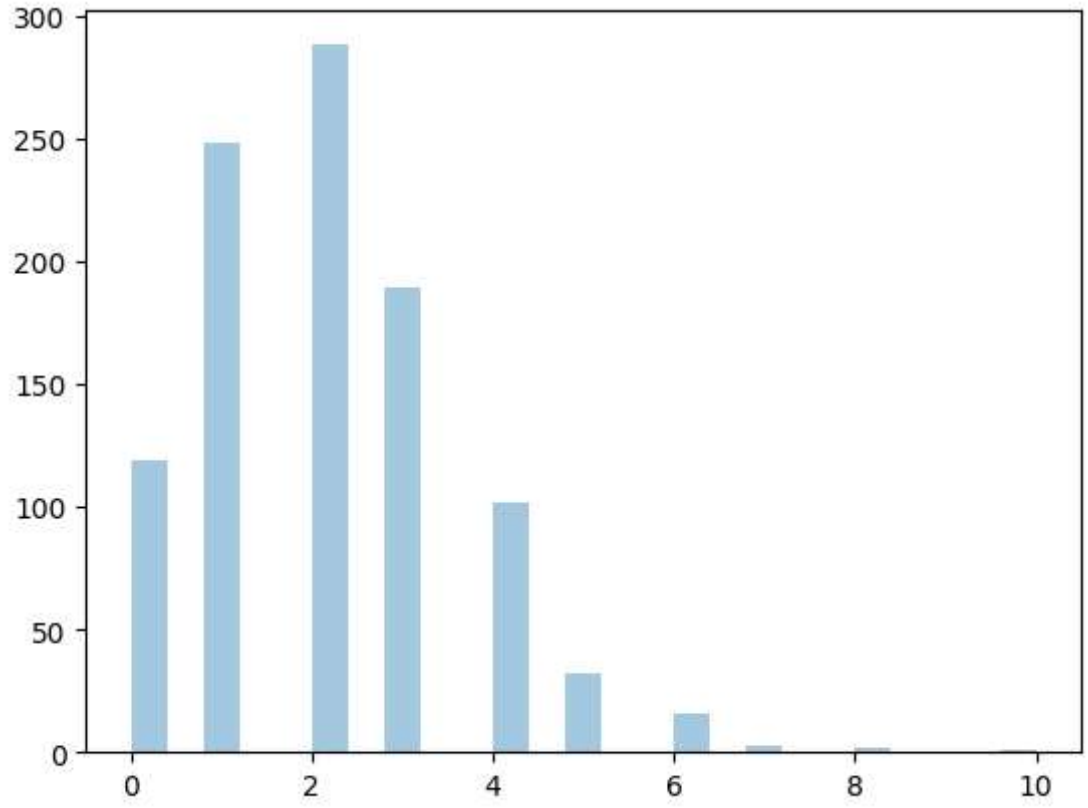
In [4]:
```
#generandom 1*10 distribution for occurence 2:
from numpy import random
x = random.poisson(lam=2, size=10)
print(x)
```

```
[1 0 1 4 3 1 1 1 7 0]
```

In [6]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.poisson(lam=2, size=1000), kde=False) #1st parameter is te number of occurence ,
#size is any random 1000 values
```

```
#in dist plot kde is set to true by default
plt.show()
```

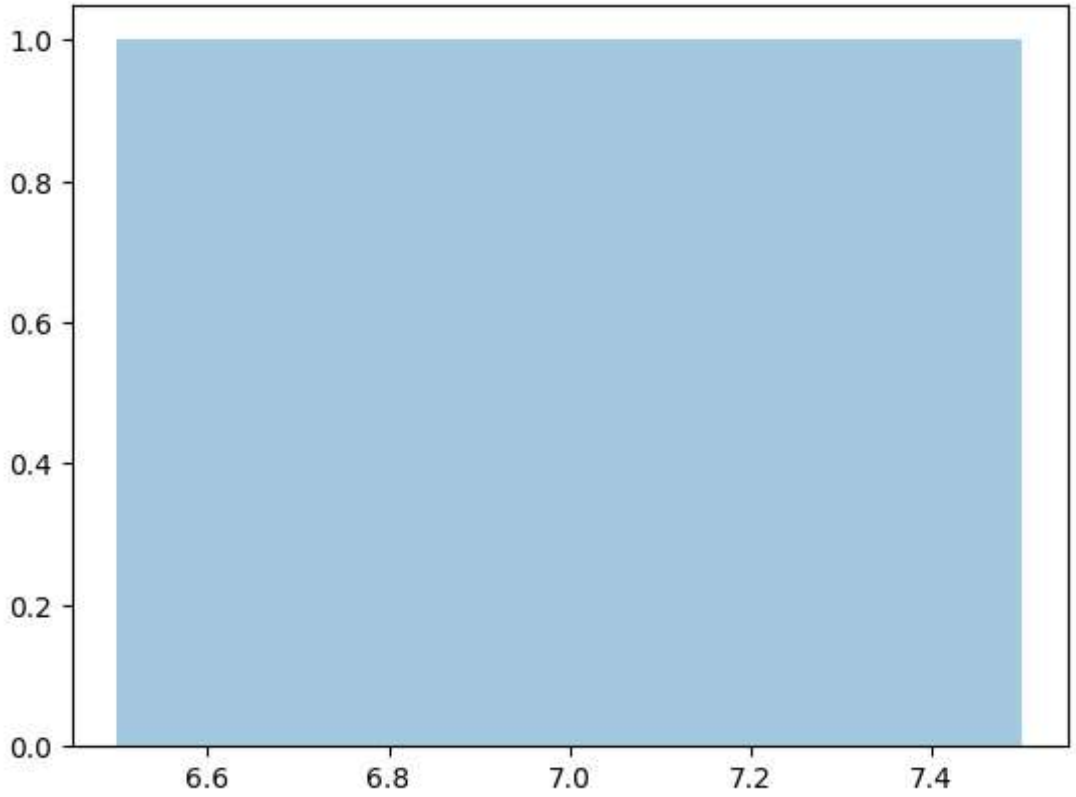
```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\1772104155.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.poisson(lam=2, size=1000), kde=False) #1st parameter is te number of occurence ,
```



In [7]:
```
from numpy import random
import matplotlib.pyplot as plt
```

```
import seaborn as sns
sns.distplot(random.poisson(lam=9, size=1), kde=False) #1st parameter is te number of occurence ,
#size is any random 1000 values
#in dist plot kde is set to true by default
plt.show()
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\4028215891.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.poisson(lam=9, size=1), kde=False) #1st parameter is te number of occurence ,
```



Call center simulation

In [53]:
```
#initialisation and print
#start code
#n = None
#p = None
#revenue = None
#employees = None
#wage = None
#size = employees
#Task 2:
#conversion,revenue expense and profits
#Task 1:
from scipy import stats

employees= 100
p=0.04
wage=200
avg_revenue = 100
n = 50
from scipy import stats
X = stats.binom(50, 0.04)
#declare X to be a binomial random variable
X.pmf(3)
```

Out[53]:
```
0.18415520493576068
```

In [62]:
```
m = X.mean() #mean =2.03
m
```

Out[62]:
```
2.0
```

In [19]:
```
X.var() #var
```

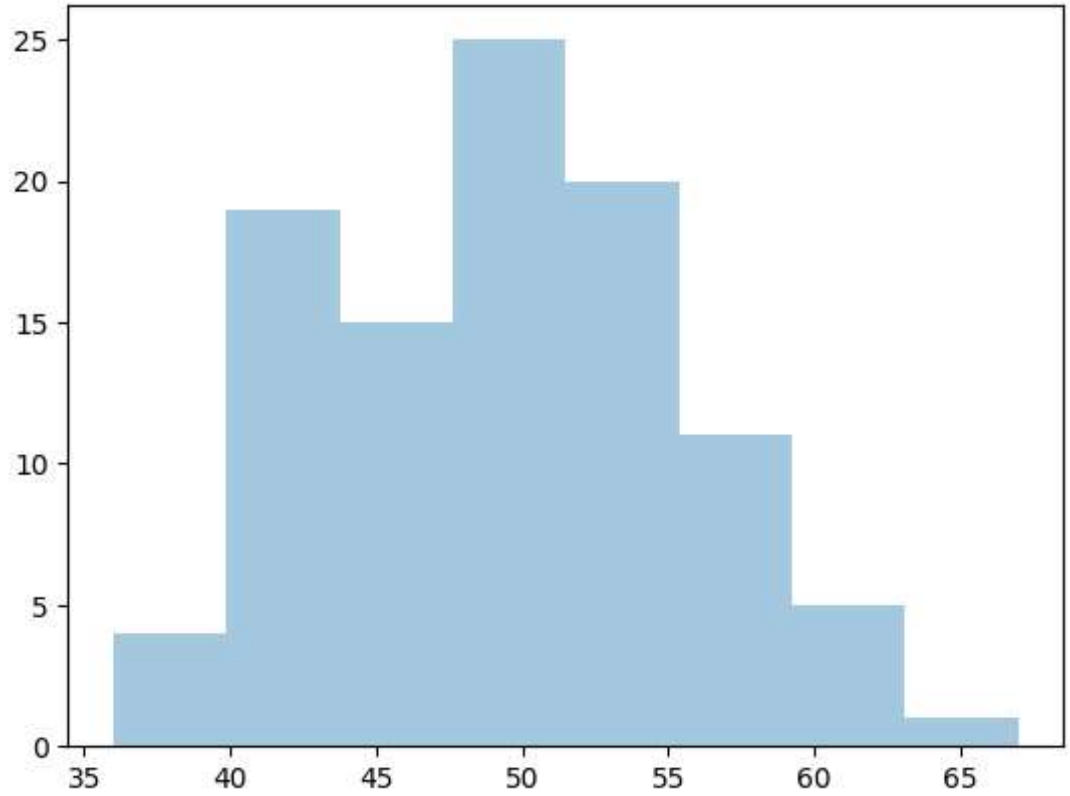
Out[19]:
```
1.92
```

In [21]:
```
1.92**0.5 #sd
```

Out[21]:
```
1.3856406460551018
```

In [23]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(random.poisson(lam=50, size=100), kde=False) #1st parameter is te number of occurence ,
#size is any random 1000 values
#in dist plot kde is set to true by default
plt.show()
#size=employees
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\2619215753.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.poisson(lam=50, size=100), kde=False) #1st parameter is te number of occurence ,
```



Task 3:

sims = 1000 calc sims_conversions # for var in range(sims) calc sims_profits # use sims conversion xlabel:profits ylabel= Frequency

results from task 2:

total conversions:203 total revenues: 20300 total expense: 20000 total profits:300

conversion is the sum of all

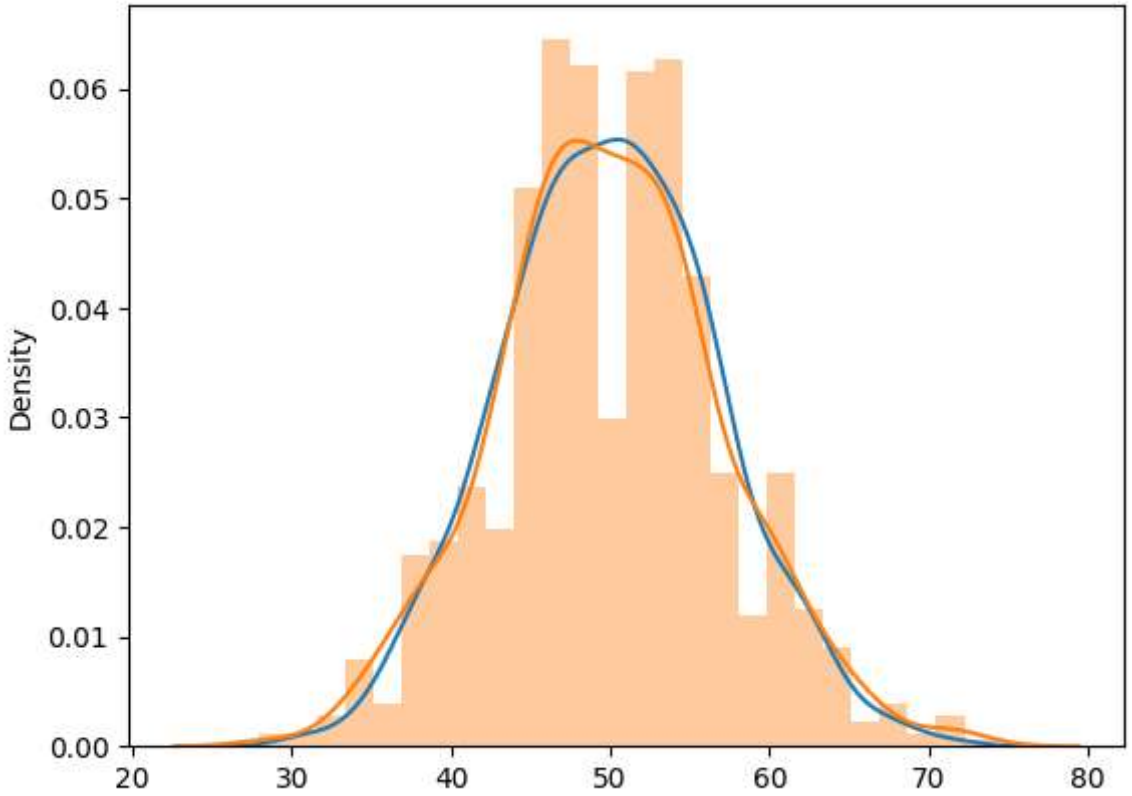
In [ ]:

In [31]:
```
from scipy import stats
Y= stats.poisson(2) #declare Y to be a poissom random variable
Y.pmf(3)
```

Out[31]:
```
0.18044704431548356
```

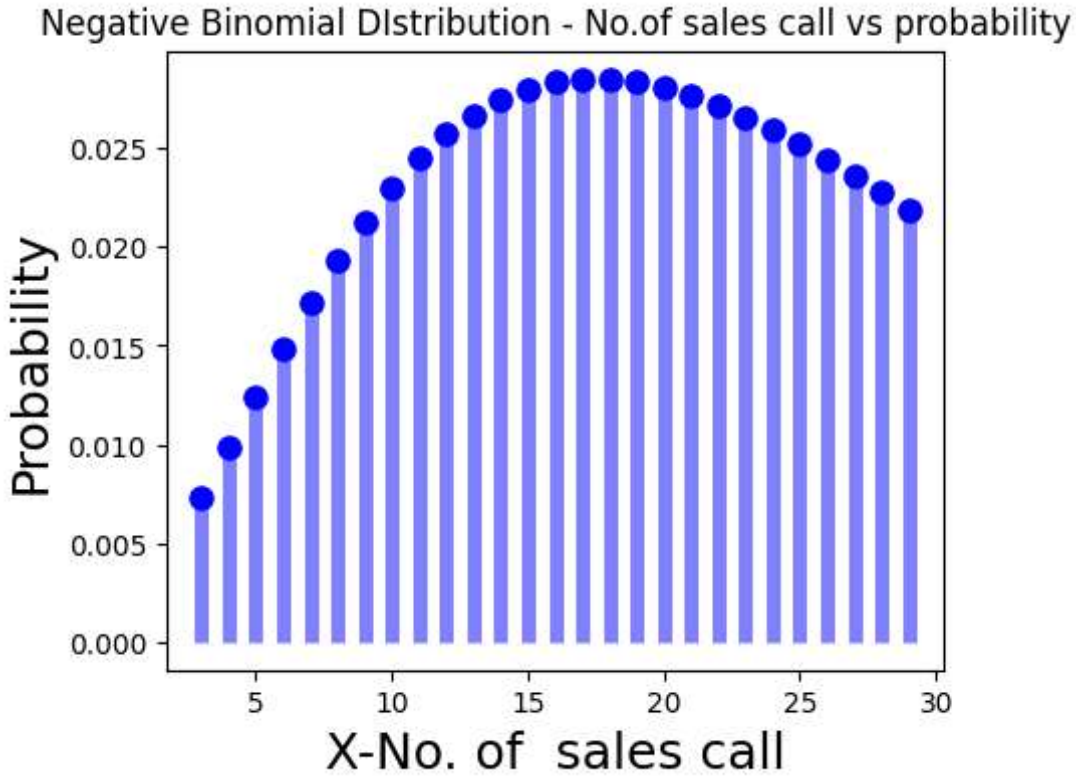
In [32]:
```
sns.distplot(random.normal(loc=50, scale=7, size=1000),hist = False, label='normal') #1st 2 parameters are mean and standad devia
sns.distplot(random.poisson(lam=50, size=1000),hist=True, label='Poisson')
plt.show()
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\1622824568.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.normal(loc=50, scale=7, size=1000),hist = False, label='normal') #1st 2 parameters are mean and standad de
viation
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\1622824568.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(random.poisson(lam=50, size=1000),hist=True, label='Poisson')
```



In [41]:
```
#flipping till we get the results >> no fixed results
from scipy.stats import nbinom
X = np.arange(3,30)
p=0.1
r=3
#X = discrete negative binomial random variable
#calculate geometric probability distribution
fig, ax= plt.subplots(1, 1, figsize=(5,4))
nbinom_pd = nbinom.pmf(X, r, p)
ax.plot(X, nbinom_pd, 'bo', ms=8, label='nbinom pmf')
plt.ylabel('Probability', fontsize='18')
plt.xlabel('X-No. of  sales call', fontsize='18')
plt.title('Negative Binomial DIstribution - No.of sales call vs probability')
ax.vlines(X,0, nbinom_pd, colors='b',lw=5, alpha=0.5)
```

Out[41]:
```
<matplotlib.collections.LineCollection at 0x25295924790>
```



Continuous distribution:

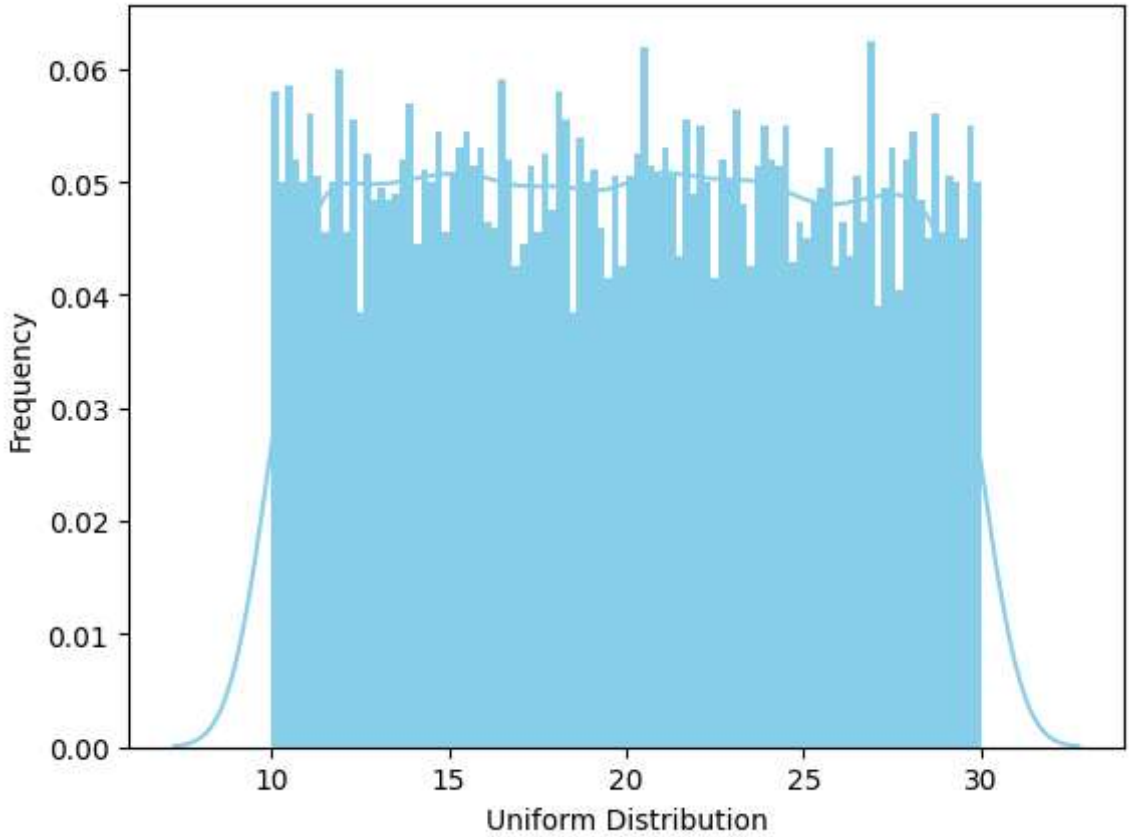
1.uniform: straight hor line

In [42]:
```
#loc+scale 10-30 that is 10 to 10+20 width as scale >>uniformly distributed
from scipy.stats import uniform
#random numbers from uniform distribution
n = 10000
start = 10
width = 20
data_uniform = uniform.rvs(size=n, loc=start, scale= width)
ax = sns.distplot(data_uniform,
                  bins =100,
                  kde= True,
```

```
                color ='skyblue',
                hist_kws={'linewidth':15,'alpha':1})
ax.set(xlabel='Uniform Distribution', ylabel='Frequency')
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\4001009652.py:8: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(data_uniform,
```

Out[42]: [Text(0.5, 0, 'Uniform Distribution'), Text(0, 0.5, 'Frequency')]



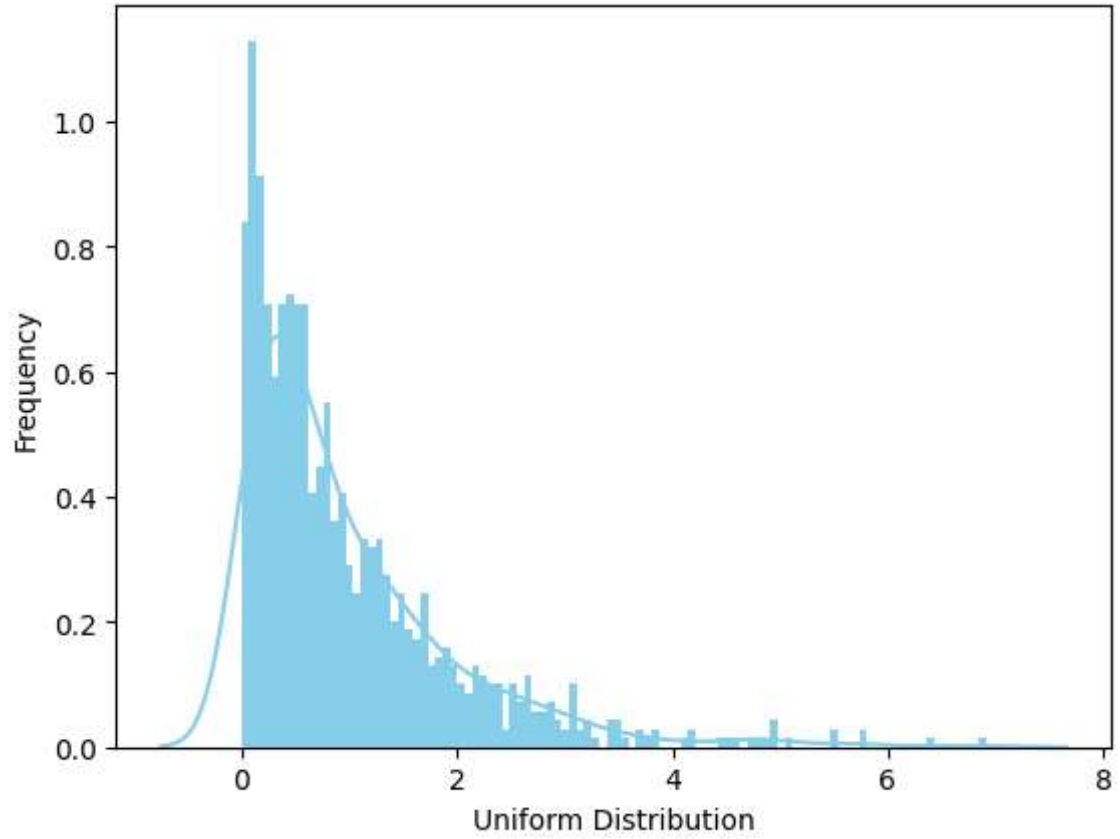
exponential distribution

rate parameter (lambda)

In [43]:
```
from scipy.stats import expon

data_expon = expon.rvs(scale=1, loc=0, size=1000)
ax = sns.distplot(data_expon,
                  bins =100,
                  kde= True,
                  color ='skyblue',
                  hist_kws={'linewidth':15,'alpha':1})
ax.set(xlabel='Uniform Distribution', ylabel='Frequency')
```

```
C:\Users\Anusha\AppData\Local\Temp\ipykernel_8440\147123030.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(data_expon,
```

Out[43]: [Text(0.5, 0, 'Uniform Distribution'), Text(0, 0.5, 'Frequency')]



In [ ]:

In [ ]: