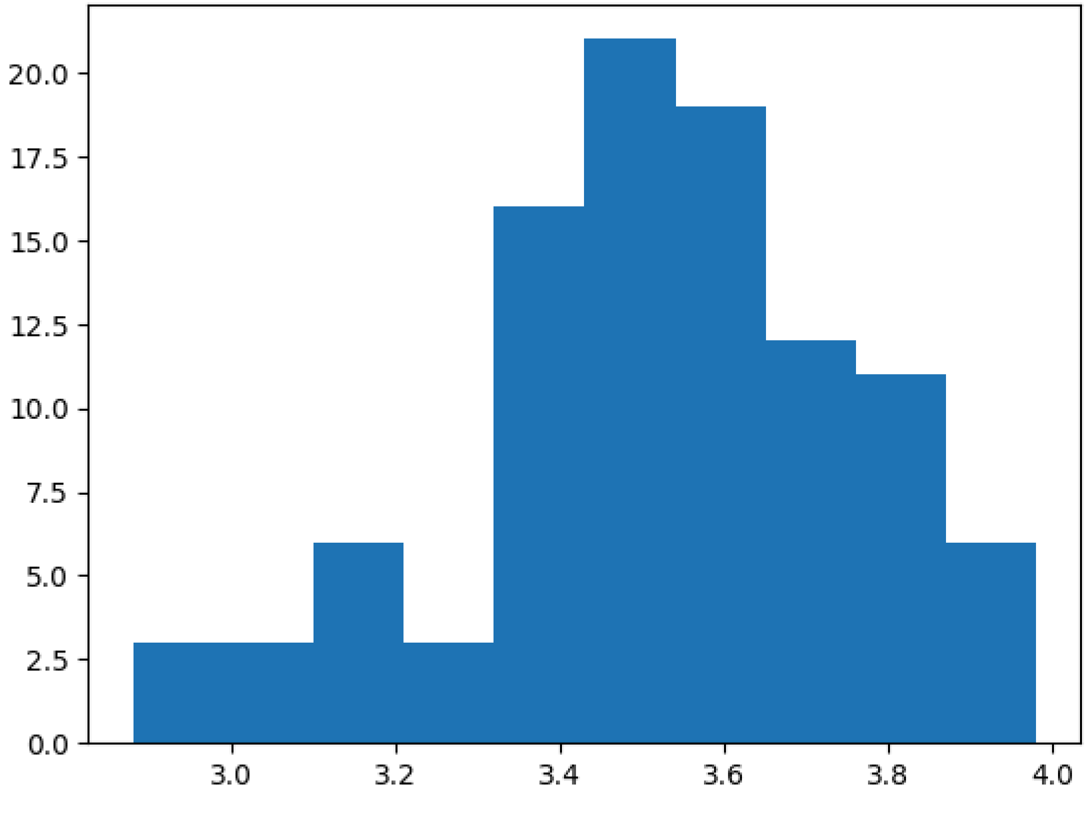
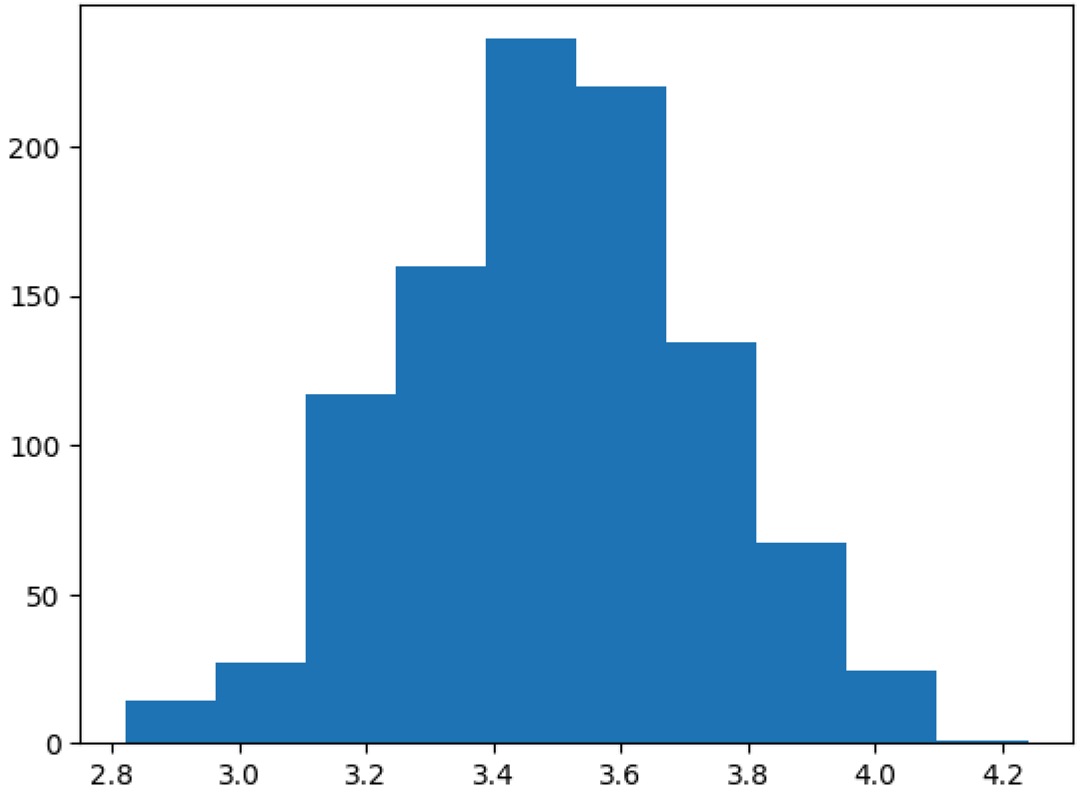
In [2]:
```
# demonstration of the central limit theorem
from numpy.random import seed
from numpy.random import randint
from numpy import mean
from matplotlib import pyplot
```

In [3]:
```
# seed the random number generator
seed(1)

def plot_clt(n):
    # calculate the mean of 50 dice rolls n times
    means = [mean(randint(1, 7, 50)) for _ in range(n)]

    # plot the distribution of sample means
    pyplot.hist(means)
    pyplot.show()
```

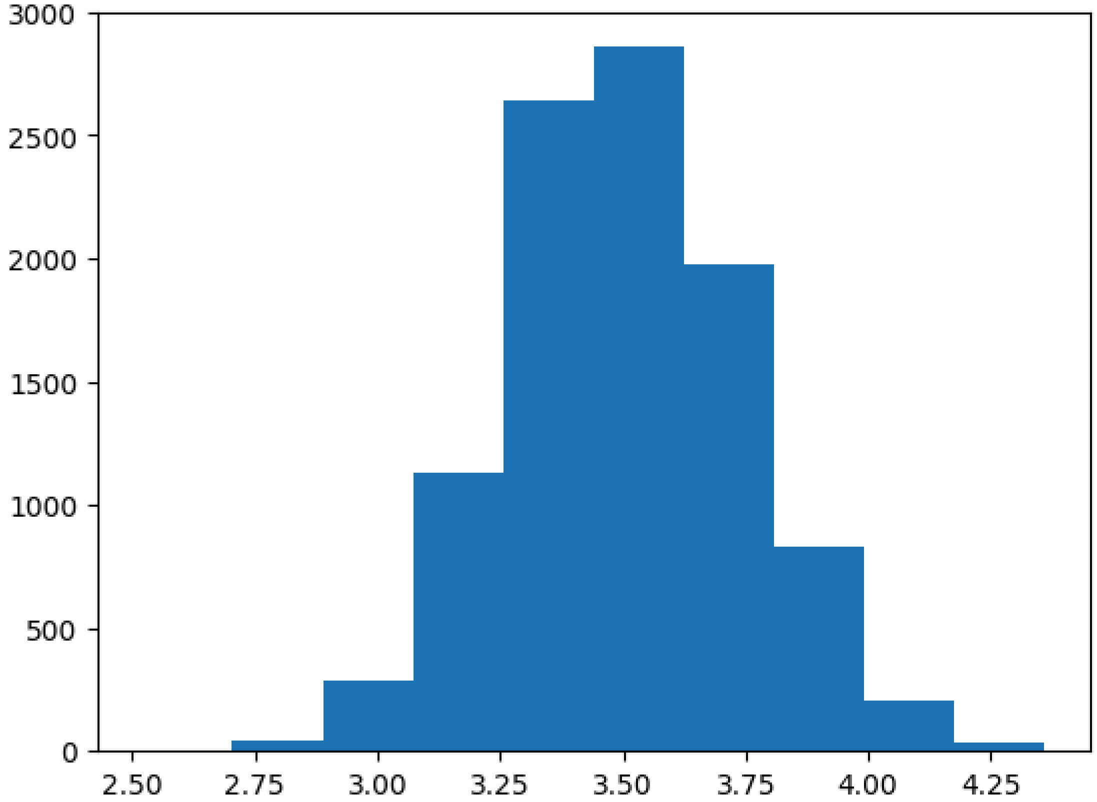
In [4]:
```
plot_clt(100)
```



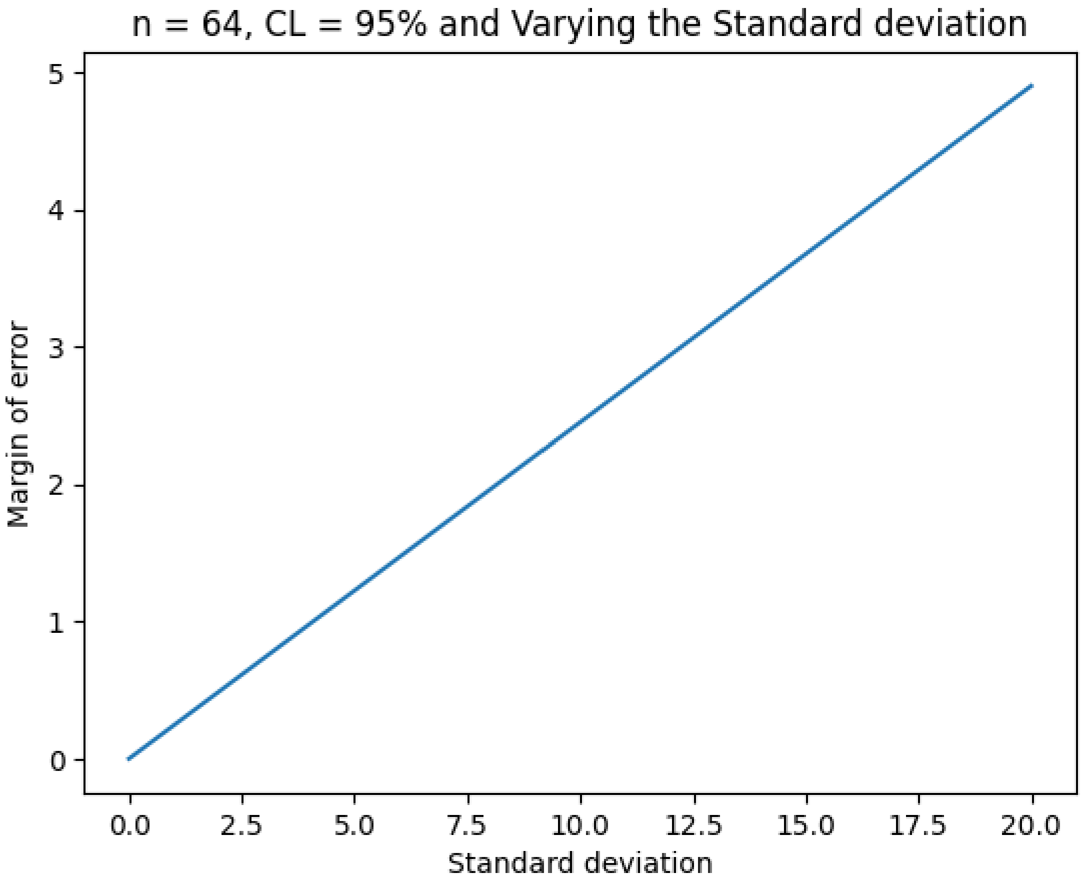
In [5]:
```
plot_clt(1000)
```




```
In [6]: plot_clt(10000)
```

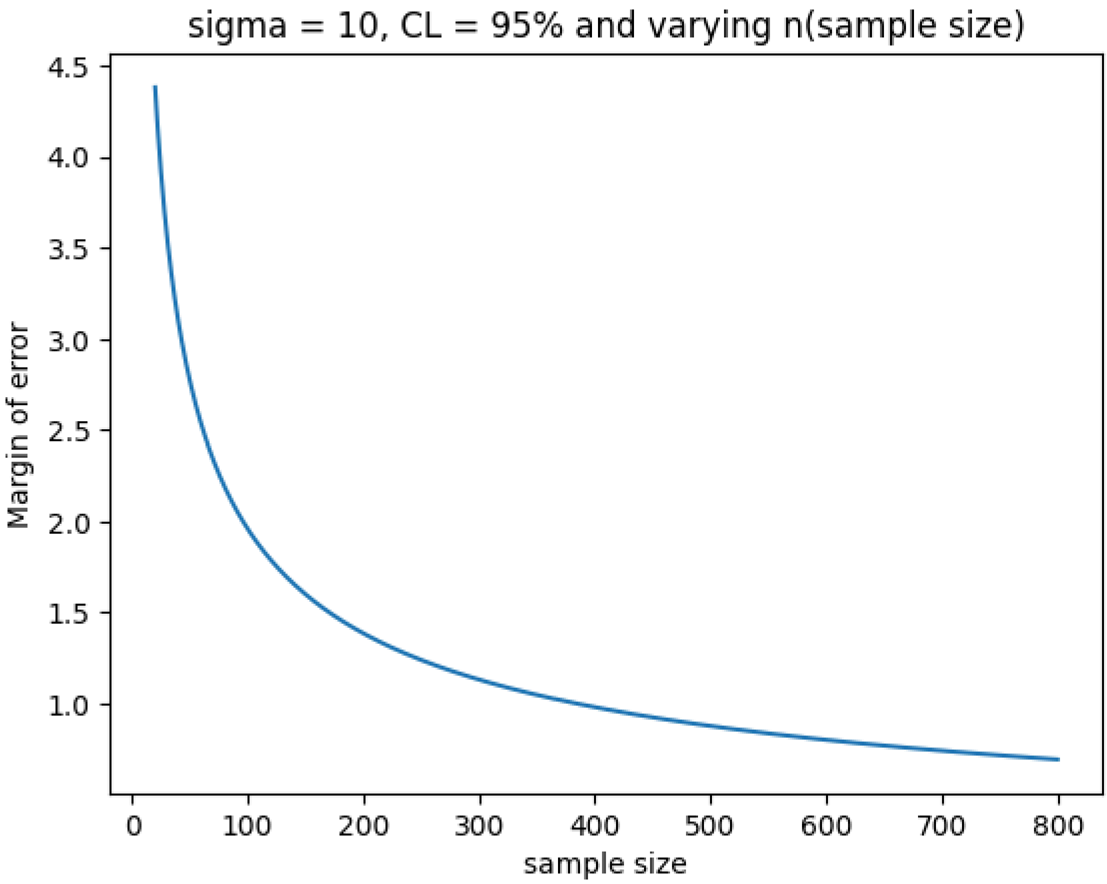


In [2]:
```
#Effect of Standard deviation on Margin of error
#MOE = z_alpha * sigma/sqrt(n)
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.stats import norm

#sigma = 10
z_score = 1.96 # 95% CL
n = 64

moe = []  #list of margin of errors
x = range(0,21) # Varying SD from 0 to 20

for sd in x:
    moe.append(z_score * sd/math.sqrt(n))

plt.plot(x, moe)
plt.title('n = 64, CL = 95% and Varying the Standard deviation')
plt.ylabel('Margin of error')
plt.xlabel('Standard deviation')
plt.show()
```



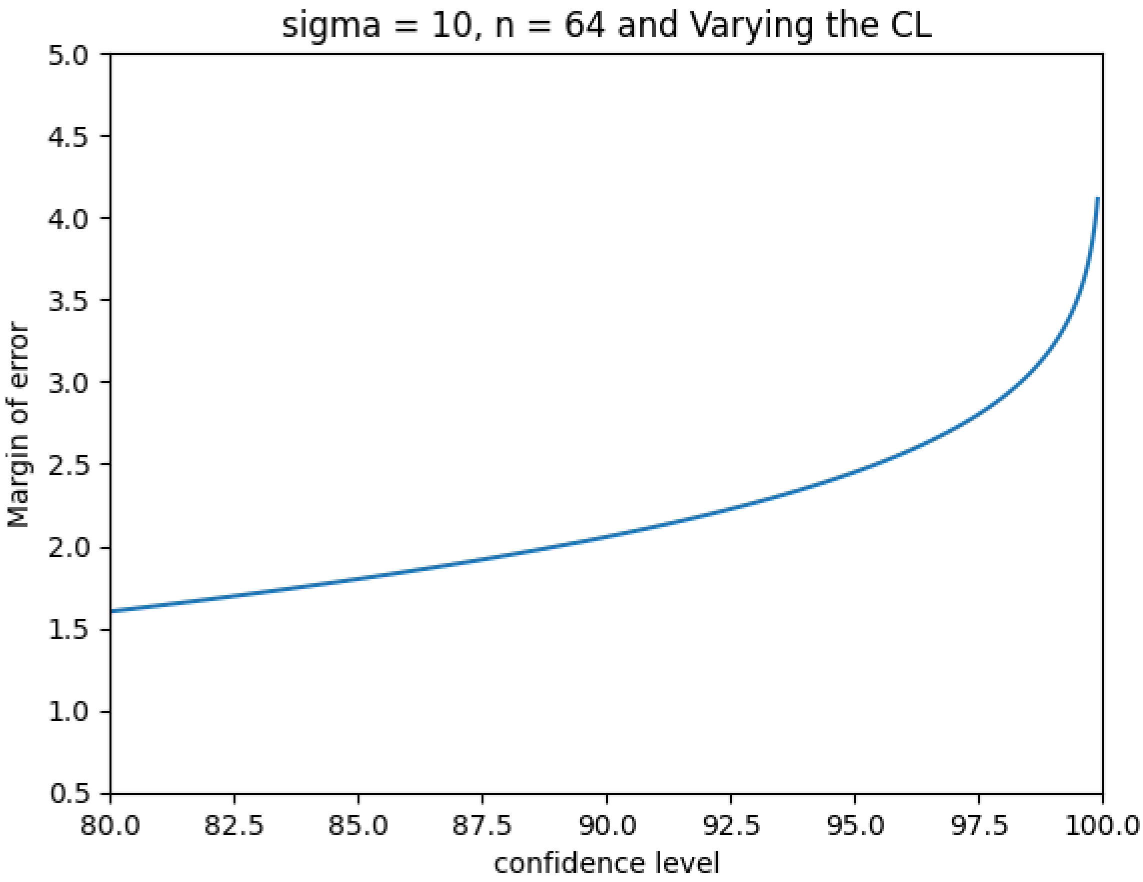
In [3]:
```
#Effect of sample size on Margin of error
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.stats import norm
```

```
sigma = 10
z_score = 1.96 # 95% CL

moe = []
x = range(20,801) # varying sample size from 20 to 800

for n in x:
    moe.append(1.96 * sigma/math.sqrt(n))

plt.plot(x, moe)
plt.title('sigma = 10, CL = 95% and varying n(sample size)')
plt.ylabel('Margin of error')
plt.xlabel('sample size')
plt.show()
```



In [1]:
```
#Effect of Confidence Level on Margin of error
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.stats import norm


sigma = 10
#z_score = 1.96 # 95% CL
n = 64

def prob(z1,z2):
       return (norm.cdf(z2) - norm.cdf(z1))


z_alphaby2 = np.linspace(norm.ppf(0.10), norm.ppf(0.0005), 100)
```

```
z_alphaby2 = z_alphaby2 * -1

#calculating margin of error
moe = z_alphaby2 * sigma/math.sqrt(n)

#getting the CL
x = []
for i in range(0, len(z_alphaby2)):
    x.append(prob(-1 * z_alphaby2[i], z_alphaby2[i]))

#getting the CL in %
x = np.array(x) * 100

plt.plot(x, moe)
plt.ylabel('Margin of error')
plt.ylim(0.5,5)
plt.xlim(80,100)
plt.title('sigma = 10, n = 64 and Varying the CL')
plt.xlabel('confidence level')
plt.show()
```



In [ ]:

In [1]:
```python
from scipy.stats import norm
from math import sqrt


def two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    actual_z = abs(norm.ppf(alpha/2))
    hypo_z = (sample_mean - pop_mean) / (std_dev/sqrt(sample_size))
    print('actual z value :', actual_z)
    print('hypothesis z value :', hypo_z, '\n')
    if hypo_z >= actual_z or hypo_z <= -(actual_z):
        return True
    else:
        return False


alpha = 0.05
sample_mean = 585
pop_mean = 558
sample_size =  100
std_dev = 139

print('H0 : μ =', pop_mean)
print('H1 : μ !=', pop_mean)
print('alpha value is :', alpha, '\n')

reject = two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject:
    print('Reject NULL hypothesis')
else:
    print('Failed to reject NULL hypothesis')
#variation with different parameters can be shown here
```

```
H0 : μ = 558
H1 : μ != 558
alpha value is : 0.05

actual z value : 1.959963984540054
hypothesis z value : 1.9424460431654675

Failed to reject NULL hypothesis
```

In [2]:
```python
#one sided hypothesis test(for smaller than in NULL hypothesis)
def one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    actual_z = abs(norm.ppf(alpha))
    hypo_z = (sample_mean - pop_mean) / (std_dev/sqrt(sample_size))
    print('actual z value :', actual_z)
    print('hypothesis z value :', hypo_z, '\n')
    if hypo_z >= actual_z:
        return True
    else:
        return False

alpha = 0.05
sample_mean = 108
pop_mean = 100
sample_size =  36
std_dev = 15

print('H0 : μ <=', pop_mean)
```

```python
print('H1 : μ >', pop_mean)
print('alpha value is :', alpha, '\n')

reject = one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject:
    print('Reject NULL hypothesis')
else:
    print('Failed to reject NULL hypothesis')
#variation with different parameters can be shown here
```

```
H0 : μ <= 100
H1 : μ > 100
alpha value is : 0.05

actual z value : 1.6448536269514729
hypothesis z value : 3.2

Reject NULL hypothesis
```

In [2]:
```
import csv
import pandas as pd
from random import sample

df = pd.read_csv('train.csv')
```

In [3]:
```
#simple random sampling
no_of_elements = 10
random_index = sample(range(df.shape[0]), no_of_elements)
print(random_index)
print(df.iloc[random_index])
```

```
[247, 73, 86, 105, 480, 510, 193, 51, 307, 548]
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
247  LP001819    Male     Yes          1  Not Graduate            No
73   LP001250    Male     Yes         3+  Not Graduate            No
86   LP001280    Male     Yes          2  Not Graduate            No
105  LP001367    Male     Yes          1      Graduate            No
480  LP002534  Female      No          0  Not Graduate            No
510  LP002637    Male      No          0  Not Graduate            No
193  LP001658    Male      No          0      Graduate            No
51   LP001157  Female      No          0      Graduate            No
307  LP001994  Female      No          0      Graduate            No
548  LP002776  Female      No          0      Graduate            No

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
247             6608                0.0       137.0             180.0
73              4755                0.0        95.0               NaN
86              3333             2000.0        99.0             360.0
105             3052             1030.0       100.0             360.0
480             4350                0.0       154.0             360.0
510             3598             1287.0       100.0             360.0
193             3858                0.0        76.0             360.0
51              3086                0.0       120.0             360.0
307             2400             1863.0       104.0             360.0
548             5000                0.0       103.0             360.0

     Credit_History Property_Area Loan_Status
247             1.0         Urban           Y
73              0.0     Semiurban           N
86              NaN     Semiurban           Y
105             1.0         Urban           Y
480             1.0         Rural           Y
510             1.0         Rural           N
193             1.0     Semiurban           Y
51              1.0     Semiurban           Y
307             0.0         Urban           N
548             0.0     Semiurban           N
```

In [4]:
```
#systematic sampling
Kth = 100
index = [i for i in range(df.shape[0]) if i%Kth==0]
print(df.iloc[index])
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
100  LP001345    Male     Yes          2  Not Graduate            No
200  LP001674    Male     Yes          1  Not Graduate            No
300  LP001964    Male     Yes          0  Not Graduate            No
400  LP002288    Male     Yes          2  Not Graduate            No
500  LP002603  Female      No          0      Graduate            No
600  LP002949  Female      No         3+      Graduate           NaN

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
100             4288             3263.0       133.0             180.0
200             2600             2500.0        90.0             360.0
300             1800             2934.0        93.0             360.0
400             2889                0.0        45.0             180.0
500              645             3683.0       113.0             480.0
600              416            41667.0       350.0             180.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
100             1.0         Urban           Y
200             1.0     Semiurban           Y
300             0.0         Urban           N
400             0.0         Urban           N
500             1.0         Rural           Y
600             NaN         Urban           N
```

In [5]:

```
#stratified sampling
#stratas formed based on Education
no_of_elements = 4        #number of elements in each strata
unique = list(set(df['Education']))
print('stratas :', unique, '\n')
index_set = [sample(list(df.index[df['Education']==i]), no_of_elements) for i in
index = [j for i in index_set for j in i]
print(df.iloc[index])
```

```
stratas : ['Not Graduate', 'Graduate']

      Loan_ID  Gender Married Dependents     Education Self_Employed  \
401  LP002296    Male      No          0  Not Graduate            No
279  LP001908  Female     Yes          0  Not Graduate            No
66   LP001228    Male      No          0  Not Graduate            No
200  LP001674    Male     Yes          1  Not Graduate            No
533  LP002729    Male      No          1      Graduate            No
132  LP001478    Male      No          0      Graduate            No
383  LP002234    Male      No          0      Graduate           Yes
437  LP002401    Male     Yes          0      Graduate            No

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
401             2755                0.0        65.0             300.0
279             4100                0.0       124.0             360.0
66              3200             2254.0       126.0             180.0
200             2600             2500.0        90.0             360.0
533            11250                0.0       196.0             360.0
132             2718                0.0        70.0             360.0
383             7167                0.0       128.0             360.0
437             2213             1125.0         NaN             360.0

     Credit_History Property_Area Loan_Status
401             1.0         Rural           N
279             NaN         Rural           Y
66              0.0         Urban           N
200             1.0     Semiurban           Y
533             NaN     Semiurban           N
132             1.0     Semiurban           Y
383             1.0         Urban           Y
437             1.0         Urban           Y
```

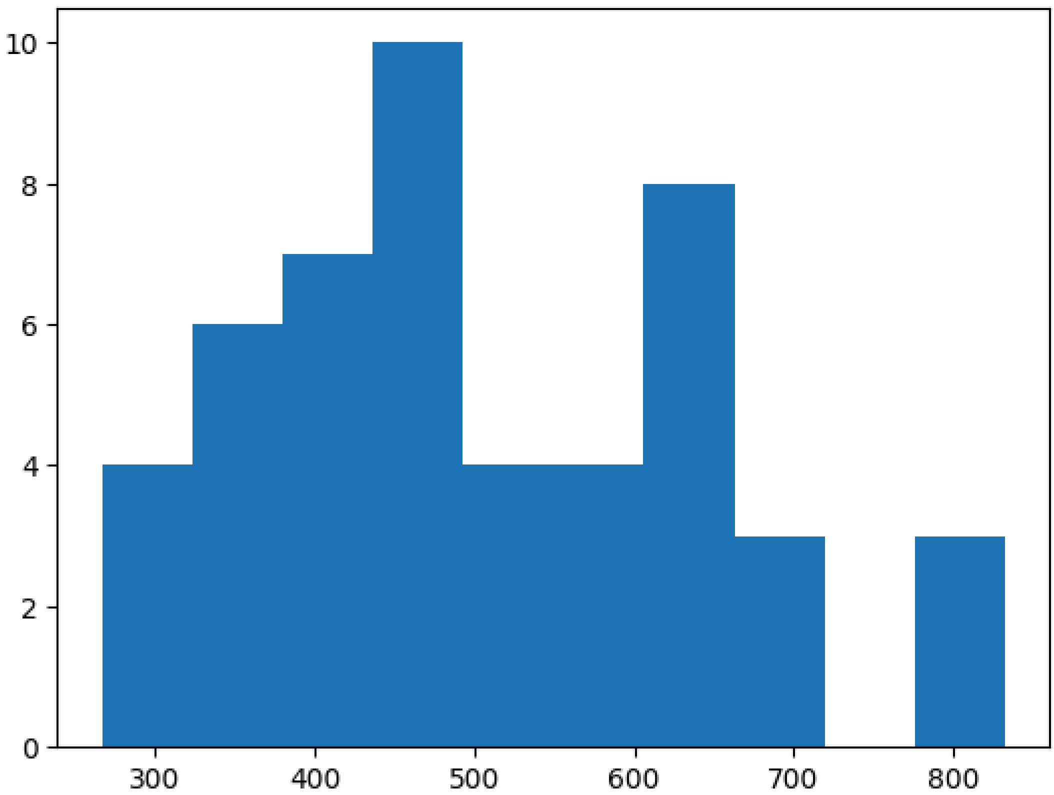
In [6]:

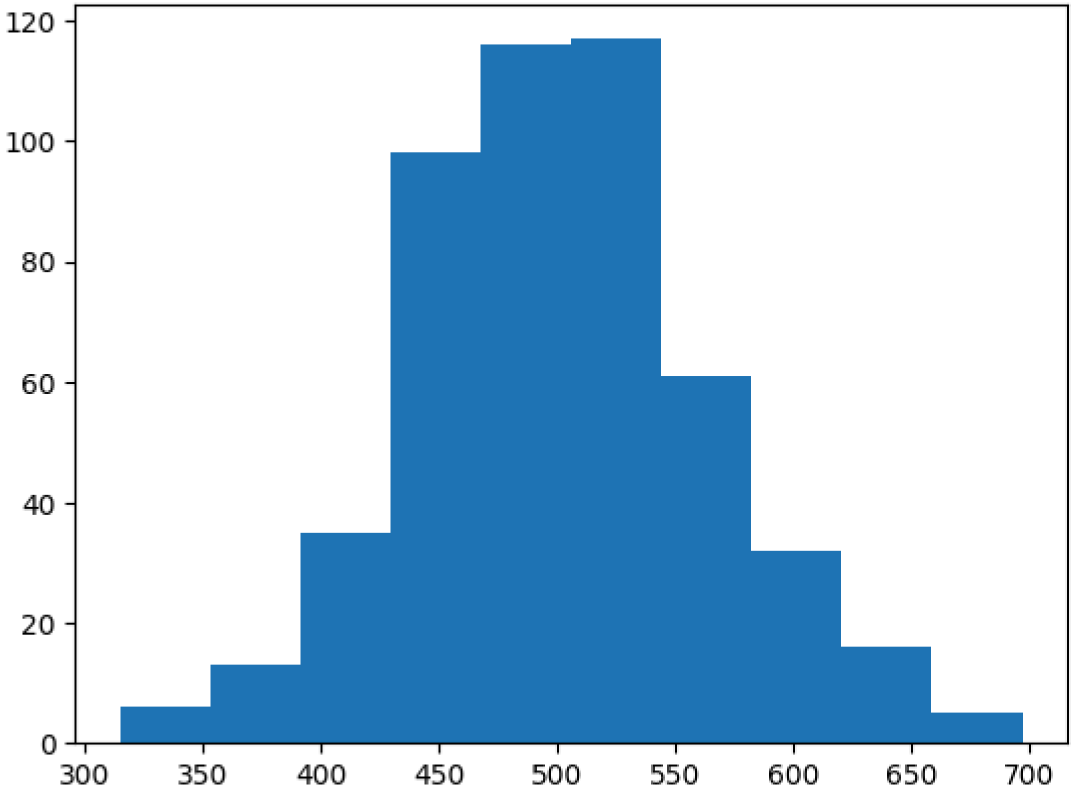
```
#cluster sampling
#clusters formed based on number of Dependents
no_of_clusters = 5
unique = list(set(df['Dependents']))
smp = sample(unique, no_of_clusters)
print("clusters :", smp, "selected out of :", unique, '\n')
index_set = [list(df.index[df['Dependents']==i]) for i in smp]
index = [j for i in index_set for j in i]
print(df.iloc[index])
```

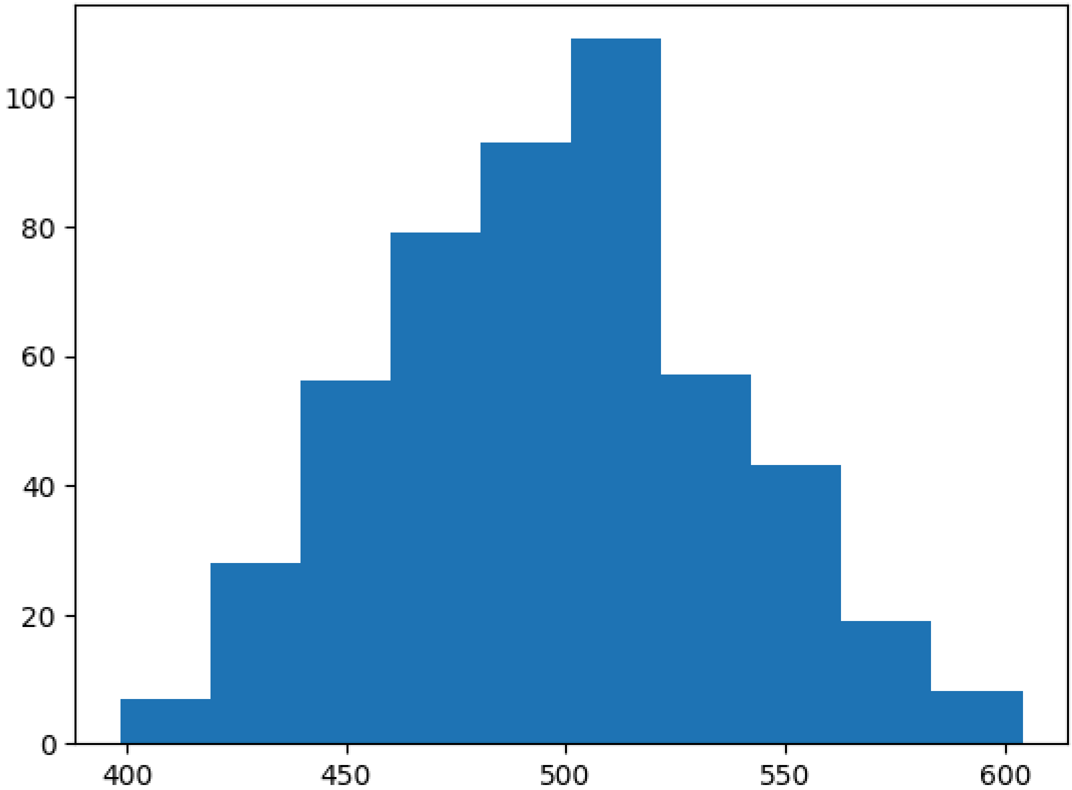
```
clusters : [nan, '0', '1', '3+', '2'] selected out of : ['1', nan, '2', '0', '3
+']

      Loan_ID Gender Married Dependents     Education Self_Employed  \
0    LP001002   Male      No          0      Graduate            No
2    LP001005   Male     Yes          0      Graduate           Yes
3    LP001006   Male     Yes          0  Not Graduate            No
4    LP001008   Male      No          0      Graduate            No
6    LP001013   Male     Yes          0  Not Graduate            No
..        ...    ...     ...        ...           ...           ...
591  LP002931   Male     Yes          2      Graduate           Yes
596  LP002941   Male     Yes          2  Not Graduate           Yes
599  LP002948   Male     Yes          2      Graduate            No
607  LP002964   Male     Yes          2  Not Graduate            No
612  LP002984   Male     Yes          2      Graduate            No

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
6               2333             1516.0        95.0             360.0
..               ...                ...         ...               ...
591             6000                0.0       205.0             240.0
596             6383             1000.0       187.0             360.0
599             5780                0.0       192.0             360.0
607             3987             1411.0       157.0             360.0
612             7583                0.0       187.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
6               1.0         Urban           Y
..              ...           ...         ...
591             1.0     Semiurban           N
596             1.0         Rural           N
599             1.0         Urban           Y
607             1.0         Rural           Y
612             1.0         Urban           Y

[599 rows x 13 columns]
```

In [ ]:

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from random import sample
from statistics import mean

def plot(arr, N, n_samples):
    x = []
    for i in range(1, n_samples):
        #to find N samples from the arr
        smp = sample(arr, N)
        mu = mean(smp)
        x.append(mu)
    plt.hist(x)
    plt.show()

#example data(population)
arr = [i for i in range(1000)]

#variations
plot(arr, 5, 50)
plot(arr, 20, 500)
plot(arr, 50, 500)
#so as number of samples(n_sample) increases the distribution becomes normal
#so as sample size increases the flatness of the distribution decreses
```

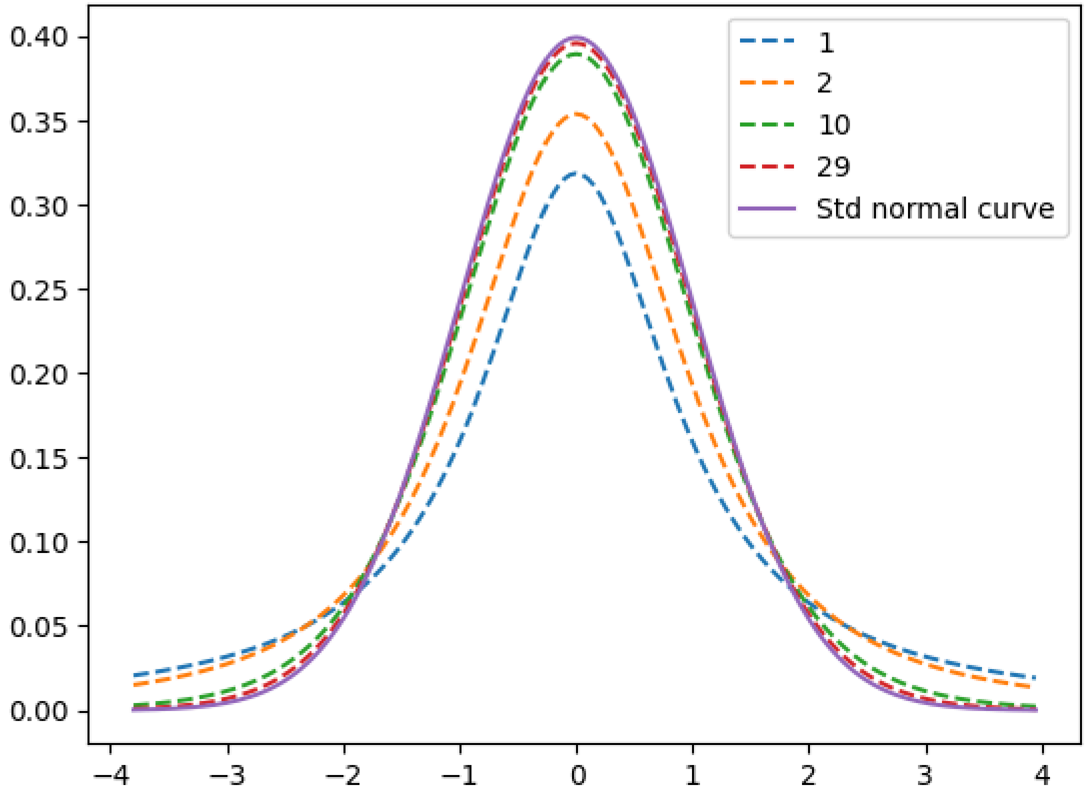



120
100
80
60
40
20
0
300
350
400
450
500
550
600
650
700


100
80
60
40
20
0
400
450
500
550
600

In [ ]:

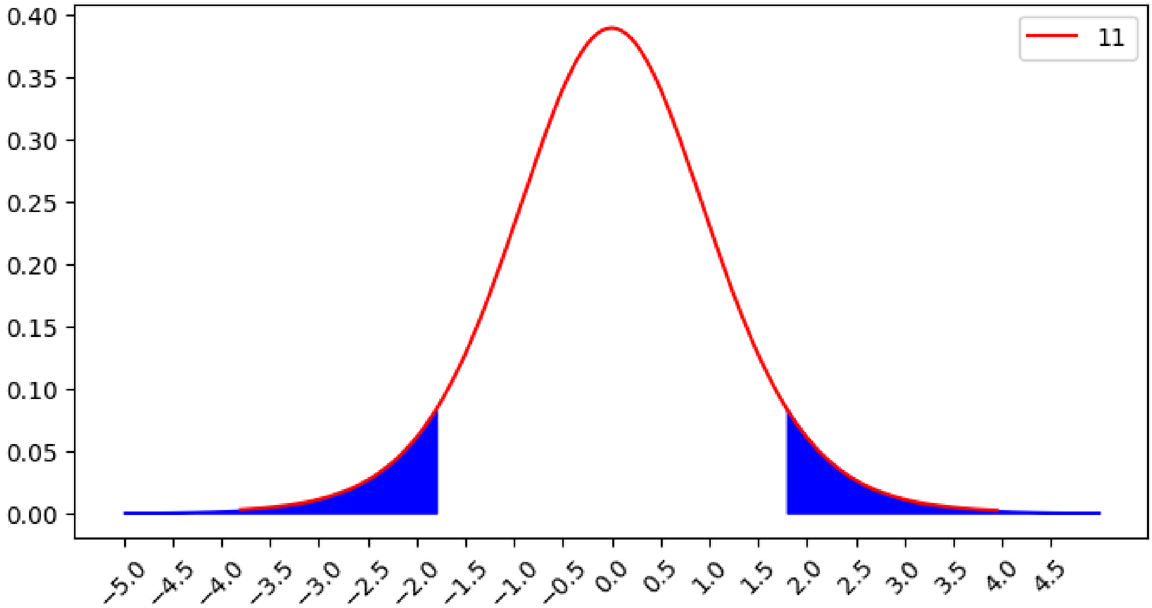
In [7]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.stats import t, norm
import numpy as np
import pandas as pd

x = np.arange(-3.8,4,1/20)    #a random population

for i in [1, 2,10, 29]:
    #plotting all the t-dist curves(pdf gives prob desnity func)
    plt.plot(x, t.pdf(x, i),'--',label=i)

#plotting a regular normal curve
plt.plot(x, norm.pdf(x), label='Std normal curve')
plt.legend(loc = 'upper right')
plt.show()

print("1-cdf gives :", 1-t.cdf(1.59, 2))
print('same as     :', t.sf(1.59, 2))
print(1-norm.cdf(2), norm.sf(2))
```




```
1-cdf gives : 0.12639805893063705
same as     : 0.12639805893063707
0.02275013194817921 0.0227501319481792
```

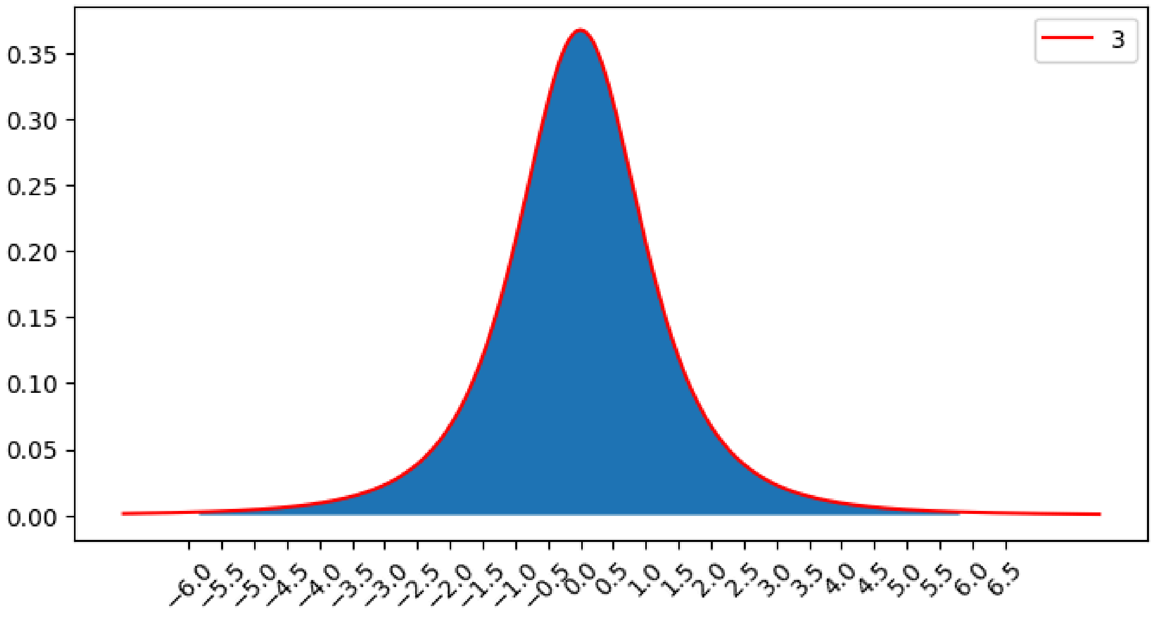
In [8]:
```
def t_table(n, alpha):
    s = t.ppf(alpha/2, n -1 )
    plt.figure(figsize=(8,4))
    plt.plot(x, t.pdf(x, n - 1), color= 'red',label= n - 1)
    #calculating the area under the graps to  be filled
    section1 = np.arange(-5, s, 1/20.)
    section2 = np.arange(-s, 5, 1/20.)
    #fill those above selected areas
    plt.fill_between(section1, t.pdf(section1, n - 1), color='blue')
```

```
    plt.fill_between(section2, t.pdf(section2, n - 1), color='blue')
    plt.xticks(np.arange(-5,5,0.5), rotation = 45)
    plt.legend(loc = 'upper right')
    plt.show()

#t_table(sample_size, alpha)
t_table(12, 0.1)
```



In [9]:
```
x = np.arange(-7, 8, 1/20)
def ci(t_score, n):
    plt.figure(figsize=(8,4))
    #gives the whole area under the graph
    area = t.cdf(t_score, n - 1) - t.cdf(-t_score, n - 1)
    print('Confidence Level', area * 100)
    plt.plot(x, t.pdf(x, n - 1), color= 'red',label= n - 1)
    #to fill from -t end to +t end
    section = np.arange(-t_score, t_score, 1/20.)
    plt.fill_between(section, t.pdf(section, n - 1))
    plt.xticks(np.arange(-6,7,0.5), rotation = 45)
    plt.legend(loc = 'upper right')
    plt.show()

ci(5.841, 4)
```

```
Confidence Level 99.00004355246759
```


3
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−6.0 −5.5 −5.0 −4.5 −4.0 −3.5 −3.0 −2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5

In [1]:
```
from scipy.stats import chi2_contingency # defining the table
data = [[207, 282, 241], [234, 242, 232]]
stat, p, dof, expected = chi2_contingency(data) # interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
p value is 0.10319714047309392
Independent (H0 holds true)
```

In [2]:
```
import numpy as np
from scipy.stats import chi2

# Observed frequencies
observed = np.array([115, 47, 41, 101, 200, 96])

# Expected frequencies (assuming a fair die)
expected = np.array([100, 100, 100, 100, 100, 100])

# Calculate chi-square statistic
chi2_stat = np.sum((observed - expected)**2 / expected)

# Degrees of freedom (number of categories - 1)
df = len(observed) - 1

# Critical value for 10% significance level
critical_value = chi2.ppf(0.90, df)

# p-value
p_value = 1 - chi2.cdf(chi2_stat, df)

# Output results
print(f"Chi-squared Statistic: {chi2_stat}")
print(f"Critical Value at 10% significance level: {critical_value}")
print(f"p-value: {p_value}")

# Conclusion
if chi2_stat < critical_value:
    print("Fail to reject the null hypothesis: The die is unbiased.")
else:
    print("Reject the null hypothesis: The die is biased.")
```

```
Chi-squared Statistic: 165.32000000000002
Critical Value at 10% significance level: 9.236356899781123
p-value: 0.0
Reject the null hypothesis: The die is biased.
```

In [3]:
```
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

# Define the observed data
data = np.array([
    [10, 102, 8],    # Machine 1
    [34, 161, 5],    # Machine 2
    [12, 79, 9],     # Machine 3
```

```python
    [10, 60, 10]    # Machine 4
])

# Create a DataFrame for better visualization (optional)
df = pd.DataFrame(data, columns=['Too Thin', 'OK', 'Too Thick'],
                  index=['Machine 1', 'Machine 2', 'Machine 3', 'Machine 4'])

print("Observed Data:\n", df)

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(data)

# Display results
print("\nChi-Square Statistic:", chi2_stat)
print("P-Value:", p_value)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:\n", expected)

# Determine if the result is significant
alpha = 0.05
if chi2_stat > chi2.ppf(1 - alpha, dof):
    print("Reject the null hypothesis: There is a significant difference.")
else:
    print("Fail to reject the null hypothesis: No significant difference.")
```

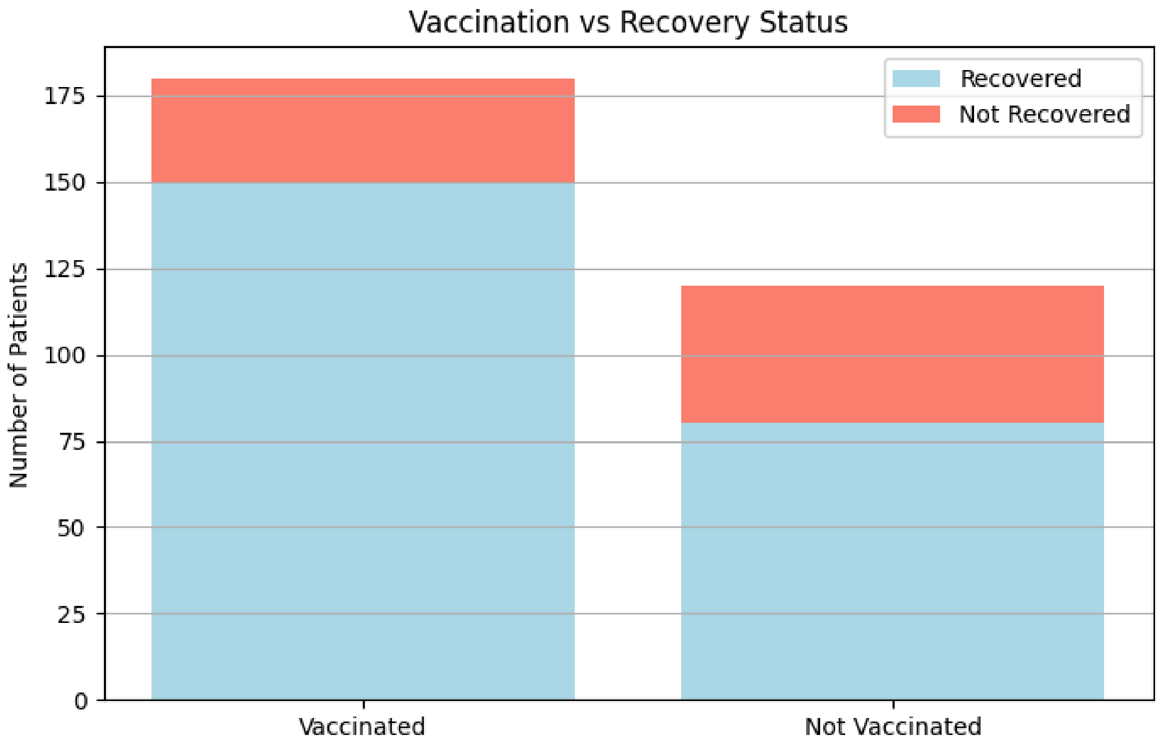
```
Observed Data:
            Too Thin   OK  Too Thick
Machine 1        10  102          8
Machine 2        34  161          5
Machine 3        12   79          9
Machine 4        10   60         10

Chi-Square Statistic: 15.584353328056686
P-Value: 0.01616760116149423
Degrees of Freedom: 6
Expected Frequencies:
 [[ 15.84  96.48   7.68]
 [ 26.4  160.8   12.8 ]
 [ 13.2   80.4    6.4 ]
 [ 10.56  64.32   5.12]]
Reject the null hypothesis: There is a significant difference.
```

In [4]:

```python
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt

# Create a contingency table
data = np.array([[150, 30],     # Vaccinated
                 [80, 40]])   # Not Vaccinated

# Display the contingency table as a DataFrame for clarity
contingency_table = pd.DataFrame(data,
                                  columns=['Recovered', 'Not Recovered'],
                                  index=['Vaccinated', 'Not Vaccinated'])
print("Contingency Table:\n", contingency_table)

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(data)
```

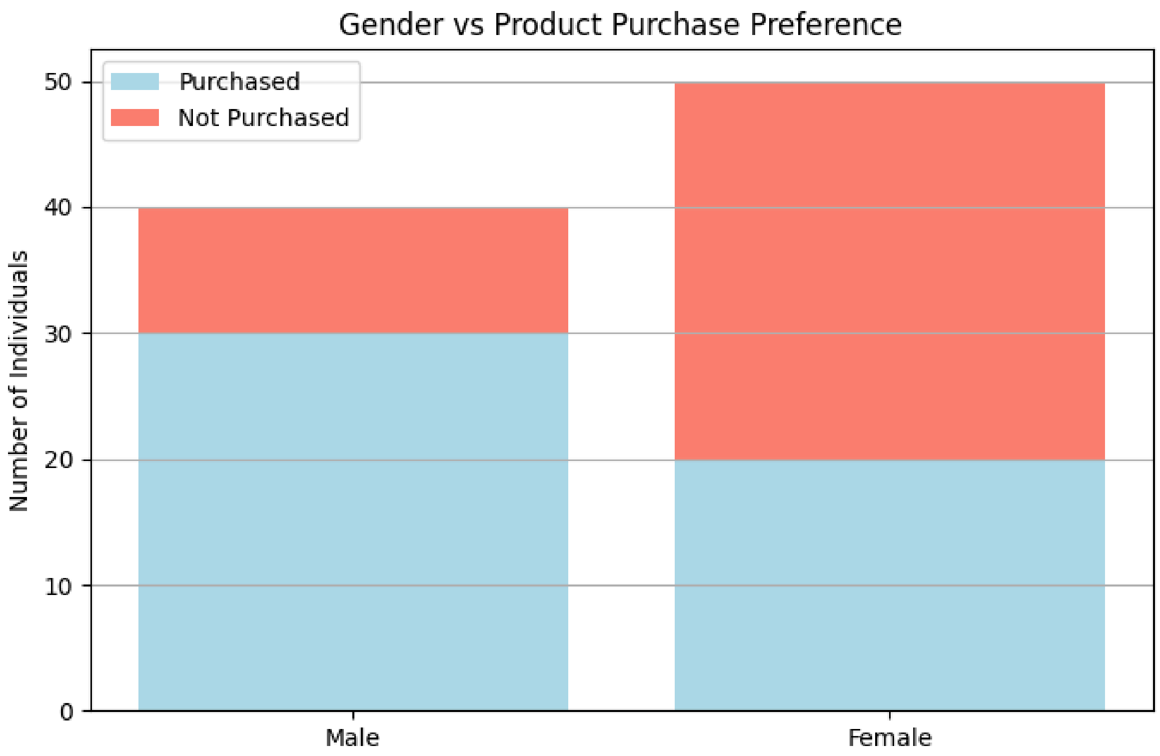
```python
# Display results
print("\nChi-Square Statistic:", chi2_stat)
print("P-Value:", p_value)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:\n", expected)

# Determine significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant association betwee
else:
    print("Fail to reject the null hypothesis: No significant association betwee

# Optional: Plotting the contingency table
plt.figure(figsize=(8, 5))
plt.title("Vaccination vs Recovery Status")
plt.bar(['Vaccinated', 'Not Vaccinated'], [150, 80], label='Recovered', color='l
plt.bar(['Vaccinated', 'Not Vaccinated'], [30, 40], label='Not Recovered', color
plt.ylabel('Number of Patients')
plt.legend()
plt.grid(axis='y')
plt.show()
```

```
Contingency Table:
                 Recovered  Not Recovered
Vaccinated             150             30
Not Vaccinated          80             40

Chi-Square Statistic: 10.267857142857142
P-Value: 0.0013536793727780064
Degrees of Freedom: 1
Expected Frequencies:
 [[138.  42.]
 [ 92.  28.]]
Reject the null hypothesis: There is a significant association between vaccinatio
n and recovery.
```



In [5]:
```python
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt

# Create a contingency table
data = np.array([[30, 10],  # Male
                 [20, 30]]) # Female

# Display the contingency table as a DataFrame for clarity
contingency_table = pd.DataFrame(data,
                                  columns=['Purchased', 'Not Purchased'],
                                  index=['Male', 'Female'])
print("Contingency Table:\n", contingency_table)

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(data)

# Display results
print("\nChi-Square Statistic:", chi2_stat)
print("P-Value:", p_value)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:\n", expected)

# Determine significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant association betwee
else:
    print("Fail to reject the null hypothesis: No significant association betwee

# Optional: Plotting the contingency table
plt.figure(figsize=(8, 5))
plt.title("Gender vs Product Purchase Preference")
plt.bar(['Male', 'Female'], [30, 20], label='Purchased', color='lightblue')
plt.bar(['Male', 'Female'], [10, 30], label='Not Purchased', color='salmon', bot
plt.ylabel('Number of Individuals')
plt.legend()
plt.grid(axis='y')
plt.show()
```

```
Contingency Table:
         Purchased  Not Purchased
Male           30             10
Female         20             30

Chi-Square Statistic: 9.6530625
P-Value: 0.001890361677058677
Degrees of Freedom: 1
Expected Frequencies:
 [[22.22222222 17.77777778]
 [27.77777778 22.22222222]]
Reject the null hypothesis: There is a significant association between gender and
product preference.
```


Gender vs Product Purchase Preference
Purchased
Not Purchased
Number of Individuals
0
10
20
30
40
50
Male
Female

Question

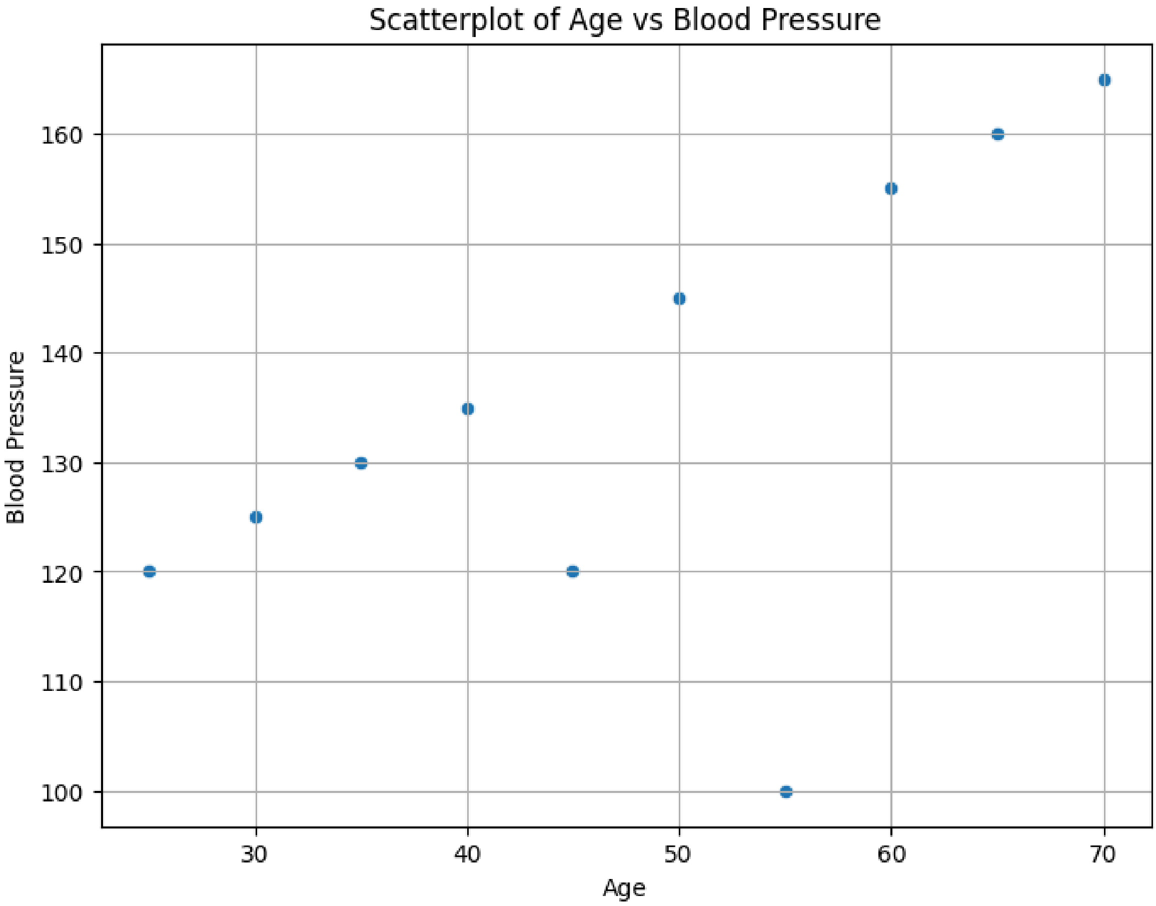
Examine the correlation between patients' age and blood pressure levels. The aim is to determine if there is a significant relationship between increasing age and higher blood pressure. Use Pearson correlation to quantify the strength and direction of the relationship

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr, spearmanr
# Sample data (replace this with actual data)
data = {
    'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],

    # Uncomment one of the following for different types of correlation
    #'BloodPressure': [120, 125, 130, 135, 140, 145, 150, 155, 160, 165], # Stro
    'BloodPressure': [120, 125, 130, 135, 120, 145, 100, 155, 160, 165], # Moder
    #'BloodPressure': [120, 125, 130, 135, 120, 145, 100, 155, 100, 165], # Weak
    #'BloodPressure': [150, 140, 135, 130, 125, 120, 110, 100, 95, 90]  # Strong
}

# Create a DataFrame
df = pd.DataFrame(data)
print("Data:\n", df)
# Plot the data to visualize the relationship
plt.figure(figsize=(8,6))
sns.scatterplot(x='Age', y='BloodPressure', data=df)
plt.title('Scatterplot of Age vs Blood Pressure')
plt.xlabel('Age')
plt.ylabel('Blood Pressure')
plt.grid(True)
plt.show()
pearson_corr, pearson_p = pearsonr(df['Age'], df['BloodPressure'])# Calculate Pe
print(f"Pearson Correlation Coefficient: {pearson_corr:.3f}, p-value: {pearson_p



# Interpretation
if pearson_corr > 0:
    if pearson_corr <= 0.5:
        print("Weak positive correlation.")
    elif 0.5 < pearson_corr < 0.8:
        print("Moderate positive correlation.")
    elif pearson_corr >= 0.8:
        print("Strong positive correlation.")
elif pearson_corr < 0:
    if pearson_corr >= -0.5:
        print("Weak negative correlation.")
    elif -0.8 < pearson_corr < -0.5:
        print("Moderate negative correlation.")
    elif pearson_corr <= -0.8:
        print("Strong negative correlation.")
```

```
else:
    print("No correlation.")
```

```
Data:
   Age  BloodPressure
0   25            120
1   30            125
2   35            130
3   40            135
4   45            120
5   50            145
6   55            100
7   60            155
8   65            160
9   70            165
```




```
Pearson Correlation Coefficient: 0.619, p-value: 0.05647
Moderate positive correlation.
```

In [3]:

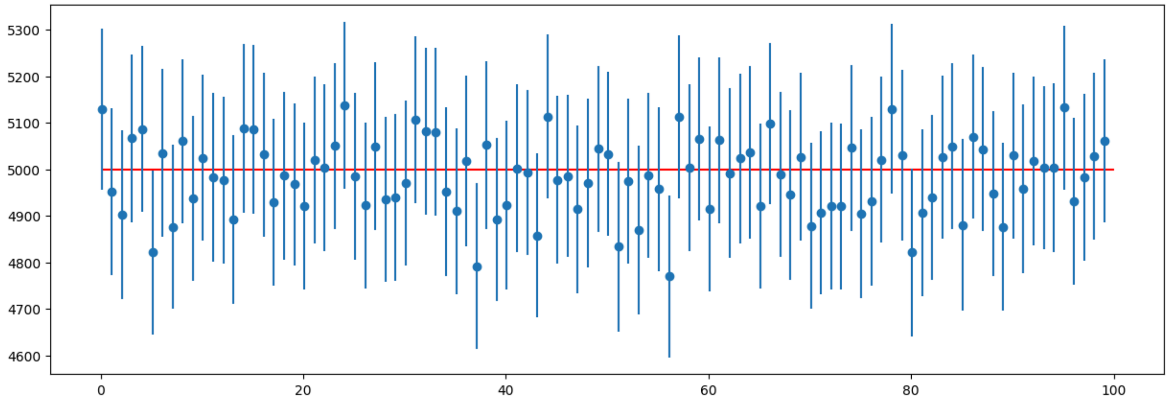
```
# Import necessary libraries
import pandas as pd
import numpy as np

# Sample data for correlation (strong positive correlation)
data = {
    'Age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],
      # Uncomment one of the following for different types of correlation
    #'BloodPressure': [120, 125, 130, 135, 140, 145, 150, 155, 160, 165], # Stro
    'BloodPressure': [120, 125, 130, 135, 120, 145, 100, 155, 160, 165], # Moder
    #'BloodPressure': [120, 125, 130, 135, 120, 145, 100, 155, 100, 165], # Weak
    #'BloodPressure': [150, 140, 135, 130, 125, 120, 110, 100, 95, 90]  # Strong
}

# Create a DataFrame
```

```
df = pd.DataFrame(data)

# Compute means
mean_x = np.mean(df['Age'])
mean_y = np.mean(df['BloodPressure'])

# Pearson correlation computation
numerator = np.sum((df['Age'] - mean_x) * (df['BloodPressure'] - mean_y))
denominator_x = np.sqrt(np.sum((df['Age'] - mean_x) ** 2))
denominator_y = np.sqrt(np.sum((df['BloodPressure'] - mean_y) ** 2))
pearson_corr_manual = numerator / (denominator_x * denominator_y)

print(f"Pearson Correlation Coefficient (Manual Calculation): {pearson_corr_manu

# Interpretation
if pearson_corr > 0:
    if pearson_corr <= 0.5:
        print("Weak positive correlation.")
    elif 0.5 < pearson_corr < 0.8:
        print("Moderate positive correlation.")
    elif pearson_corr >= 0.8:
        print("Strong positive correlation.")
elif pearson_corr < 0:
    if pearson_corr >= -0.5:
        print("Weak negative correlation.")
    elif -0.8 < pearson_corr < -0.5:
        print("Moderate negative correlation.")
    elif pearson_corr <= -0.8:
        print("Strong negative correlation.")
else:
    print("No correlation.")
```

```
Pearson Correlation Coefficient (Manual Calculation): 0.619
Moderate positive correlation.
```

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from math import sqrt
from scipy.stats import norm
import random

population = np.arange(1, 10**4) #random population
pop_mean = np.mean(population)

def sampling(sample_size, no_of_samples):
    sample_means = []
    intervals = []
    count = 0
    for i in range(no_of_samples):
        #a sample of size sample_size will be taken
        sample = random.sample(list(population), sample_size)
        #mean of the samples appended to sample_means
        sample_means.append(np.mean(sample))
        #ci contains lower and upper bound of interval with 0.95 confidence
        ci = norm.interval(0.95, np.mean(sample),
                           np.std(sample, ddof =1)/sqrt(sample_size))
        intervals.append(ci)
        #upcount only if pop_mean lies in confidence interval
        if pop_mean >= ci[0] and pop_mean <= ci[1]:
            count = count + 1

    print('Proportion of CIs covering Pop mean', count/no_of_samples)
    plt.figure(figsize=(15,5))
    #print the horizontal line which is pop_mean
    plt.hlines(y = pop_mean, xmin = 0, xmax = 100, color ='r')
    #print the sample lines with their means indicated as 'o'
    plt.errorbar(np.arange(0.1, 100, 1), sample_means, fmt = 'o', yerr = [(upp -
    plt.show()

#pass sample_size, no_of_samples
sampling(1000, 100)
```

```
Proportion of CIs covering Pop mean 0.97
```



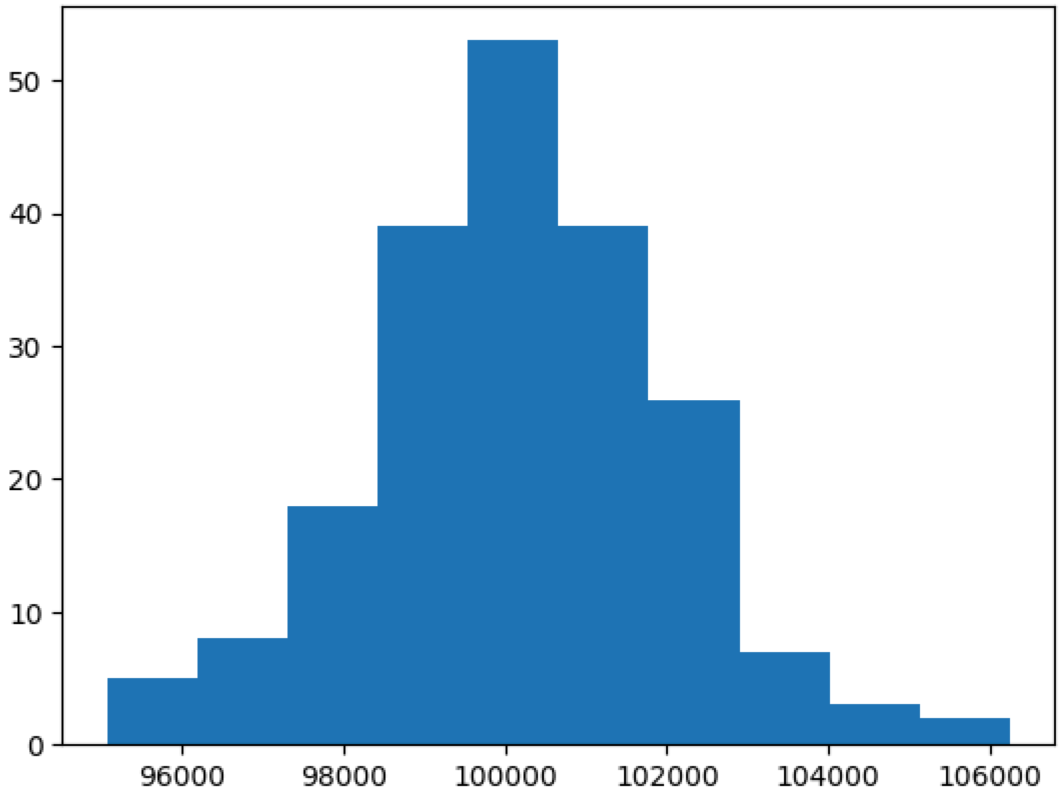
In [2]:
```
#CI for population where 85% of the people say YES to a certain question
import numpy as np
import matplotlib.pyplot as plt
from random import sample
import scipy.stats as st
```

```
import math

#parameters....population, required_CI, sample_size, no_of_samples
def CI(pop, ci, samp_size, no_of_samples):
    print("\nfor ci of", ci, "sample_size", samp_size)
    pop_mean = np.mean(pop)
    print('actual mean :',pop_mean)

    #calculation of same using CI
    samp_means = []      #mean of all the samples
    for i in range(no_of_samples):
        samp_means.append(np.mean(sample(pop, samp_size)))

    #calculation of interval
    print('mean of samples :', np.mean(samp_means))
    pop_stdev = np.std(samp_means) / math.sqrt(samp_size)
    z = st.norm.ppf(ci)
    print("confidence interval :", pop_mean, "+-", z*pop_stdev)
    plt.hist(samp_means)
    plt.show()

pop = sample(range(1, 2*10**5), 10**4)  #random population generation
```

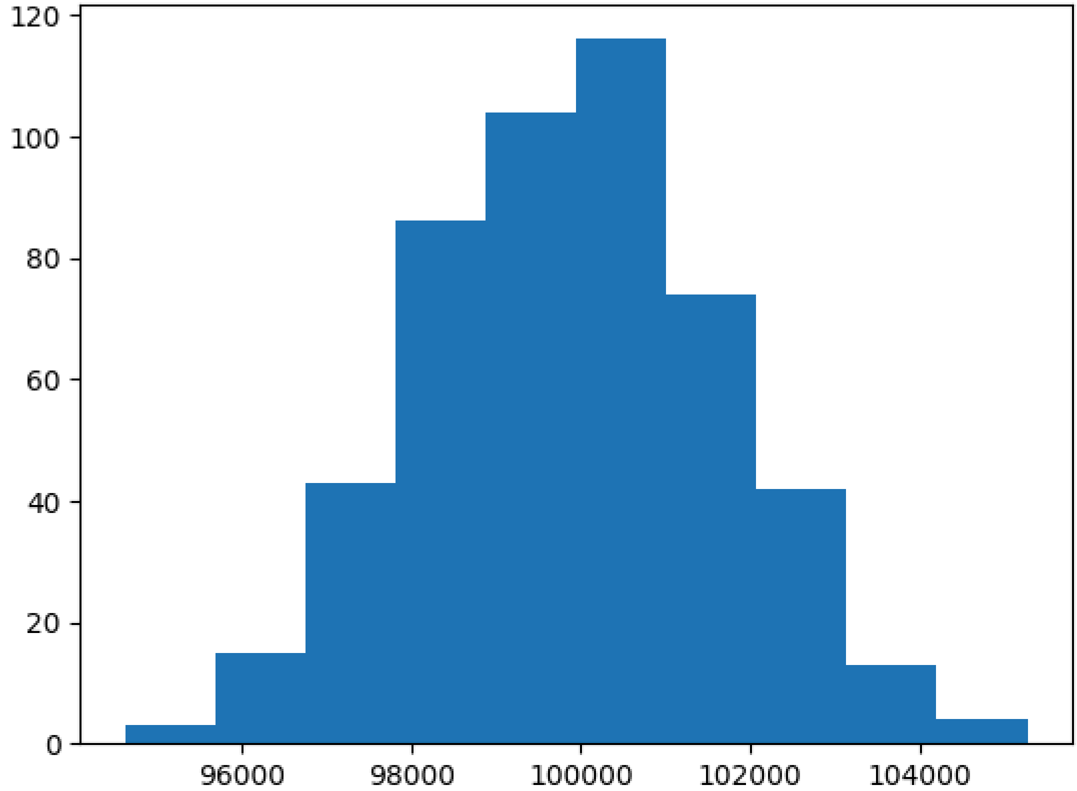
In [3]:

```
#varying no_of_samples
CI(pop, 0.85, 1000, 200)
CI(pop, 0.85, 1000, 500)
CI(pop, 0.85, 1000, 1000)
#shape of the curve becomes normal as the no of samples increases(samp_mean bett
```

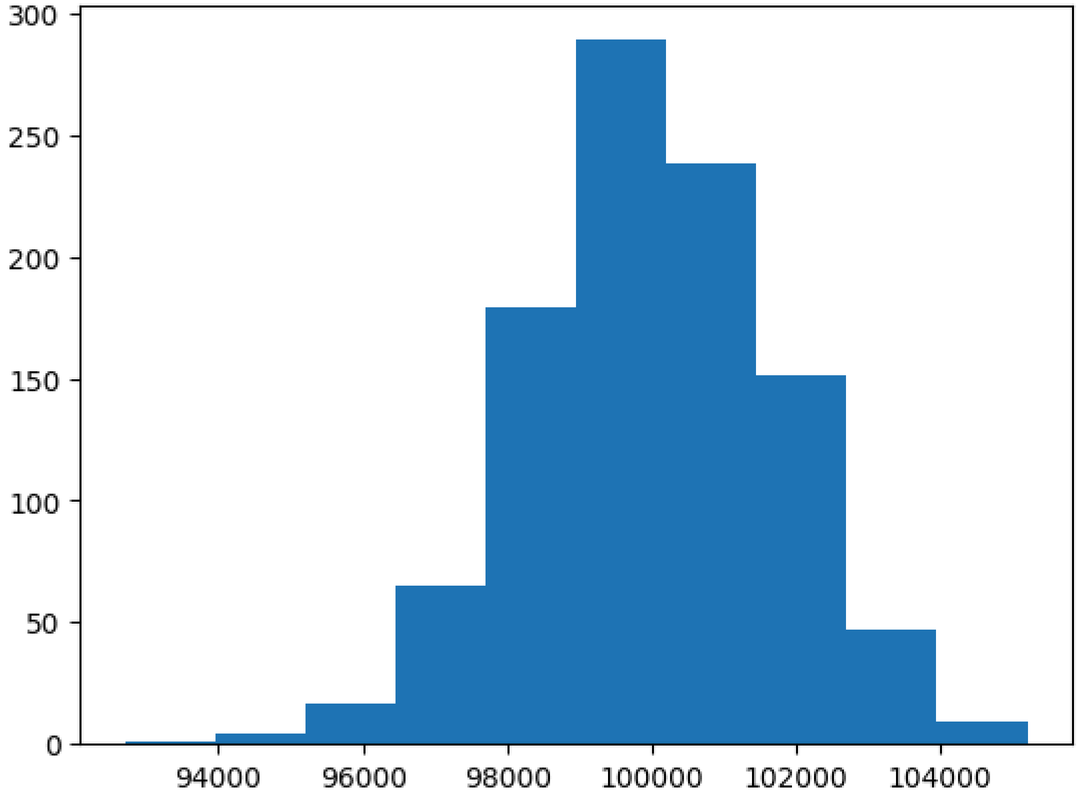
```
for ci of 0.85 sample_size 1000
actual mean : 99976.1885
mean of samples : 100178.85078
confidence interval : 99976.1885 +- 60.56608083307446
```



```
for ci of 0.85 sample_size 1000
actual mean : 99976.1885
mean of samples : 99908.319476
confidence interval : 99976.1885 +- 58.12247581667002
```



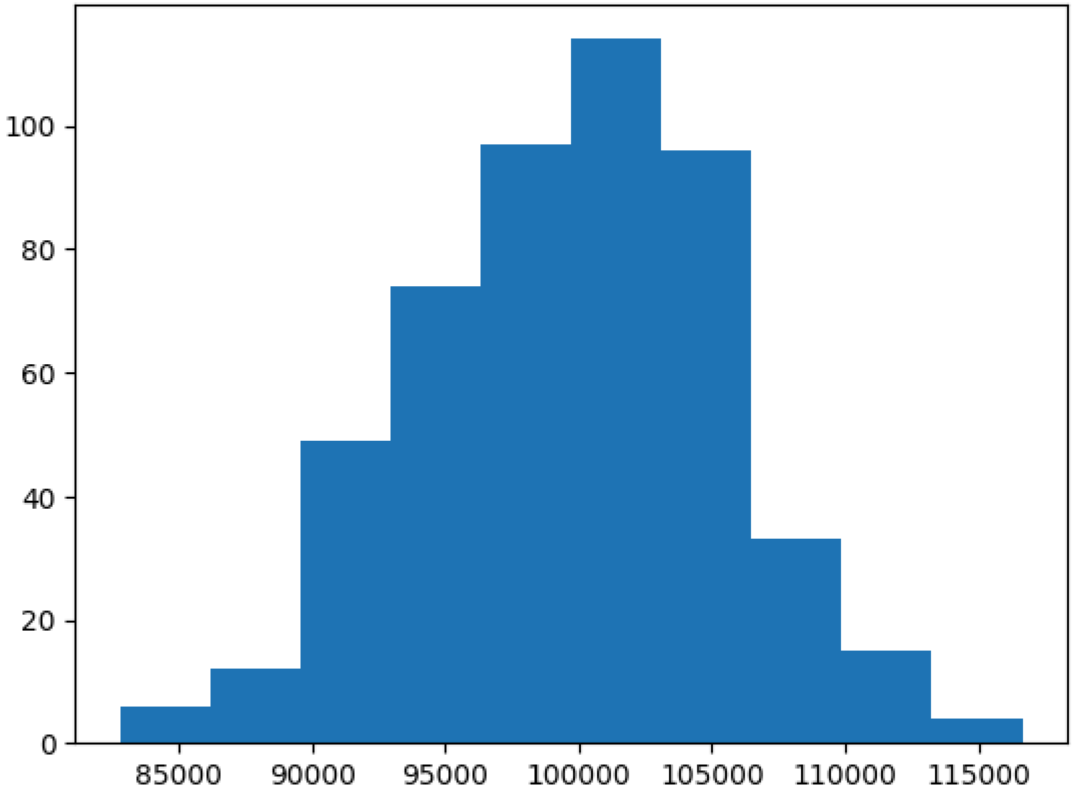

```
for ci of 0.85 sample_size 1000
actual mean : 99976.1885
mean of samples : 99984.57400600001
confidence interval : 99976.1885 +- 56.60941099970361
```



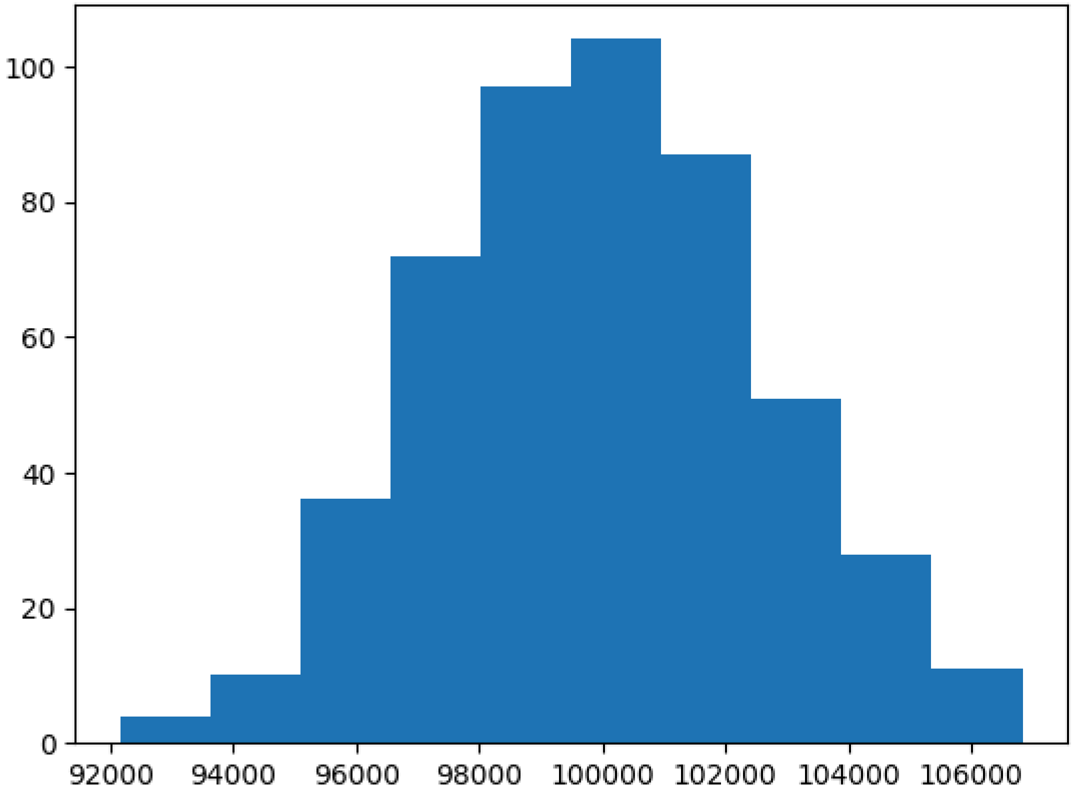
In [4]:
```
#varying sample size
CI(pop, 0.85, 100, 500)
```

```
CI(pop, 0.85, 500, 500)
CI(pop, 0.85, 1000, 500)
#reduction in the size of interval as sample_size increases(better approx of pop
```

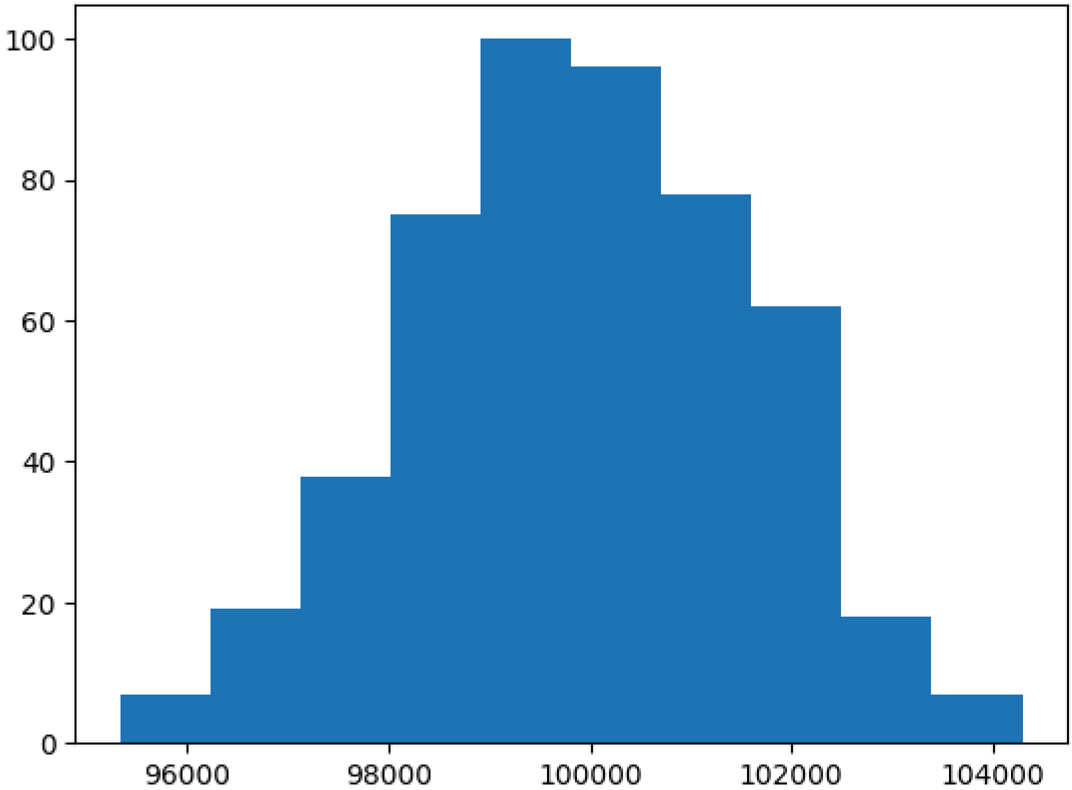
```
for ci of 0.85 sample_size 100
actual mean : 99976.1885
mean of samples : 99723.44803999999
confidence interval : 99976.1885 +- 591.4420803012979
```




```
for ci of 0.85 sample_size 500
actual mean : 99976.1885
mean of samples : 99960.261576
confidence interval : 99976.1885 +- 123.1700022730698
```

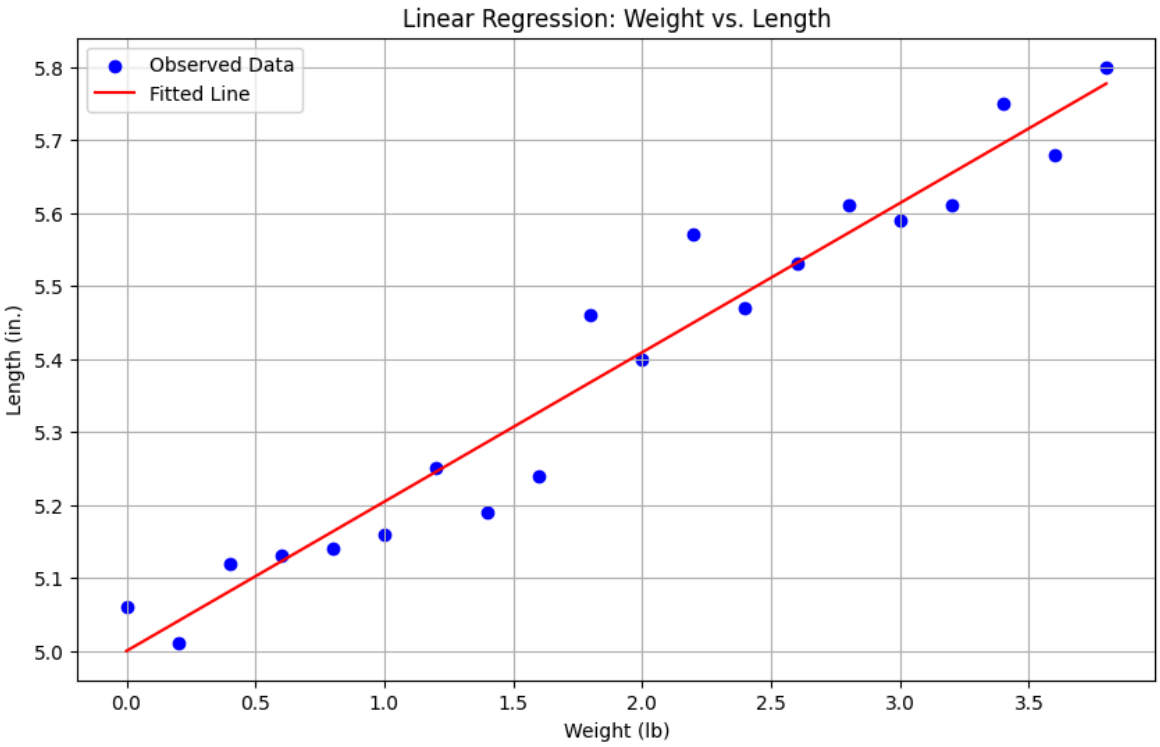



```
for ci of 0.85 sample_size 1000
actual mean : 99976.1885
mean of samples : 99926.154728
confidence interval : 99976.1885 +- 54.621342055094026
```



In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Given data: Weight (lb) (x) and Length (in.) (y)
weight = np.array([
    0.0, 0.2, 0.4, 0.6, 0.8, 1.0,
    1.2, 1.4, 1.6, 1.8,
    2.0, 2.2, 2.4, 2.6, 2.8,
    3.0, 3.2, 3.4, 3.6, 3.8
])

length = np.array([
    5.06, 5.01, 5.12, 5.13, 5.14, 5.16,
    5.25, 5.19, 5.24, 5.46,
    5.40, 5.57, 5.47, 5.53, 5.61,
    5.59, 5.61, 5.75, 5.68, 5.80
])

# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(weight, length)

# Print regression results
print(f"Intercept: {intercept:.4f}")
print(f"Slope: {slope:.4f}")
print(f"R-squared: {r_value**2:.4f}")

# Generate predicted values
length_predicted = intercept + slope * weight

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(weight, length, color='blue', label='Observed Data')
plt.plot(weight, length_predicted, color='red', label='Fitted Line')
plt.title('Linear Regression: Weight vs. Length')
plt.xlabel('Weight (lb)')
plt.ylabel('Length (in.)')
plt.legend()
plt.grid()
plt.show()
```

```
Intercept: 4.9997
Slope: 0.2046
R-squared: 0.9493
```


Linear Regression: Weight vs. Length
Observed Data
Fitted Line
Length (in.)
Weight (lb)
5.0
5.1
5.2
5.3
5.4
5.5
5.6
5.7
5.8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5