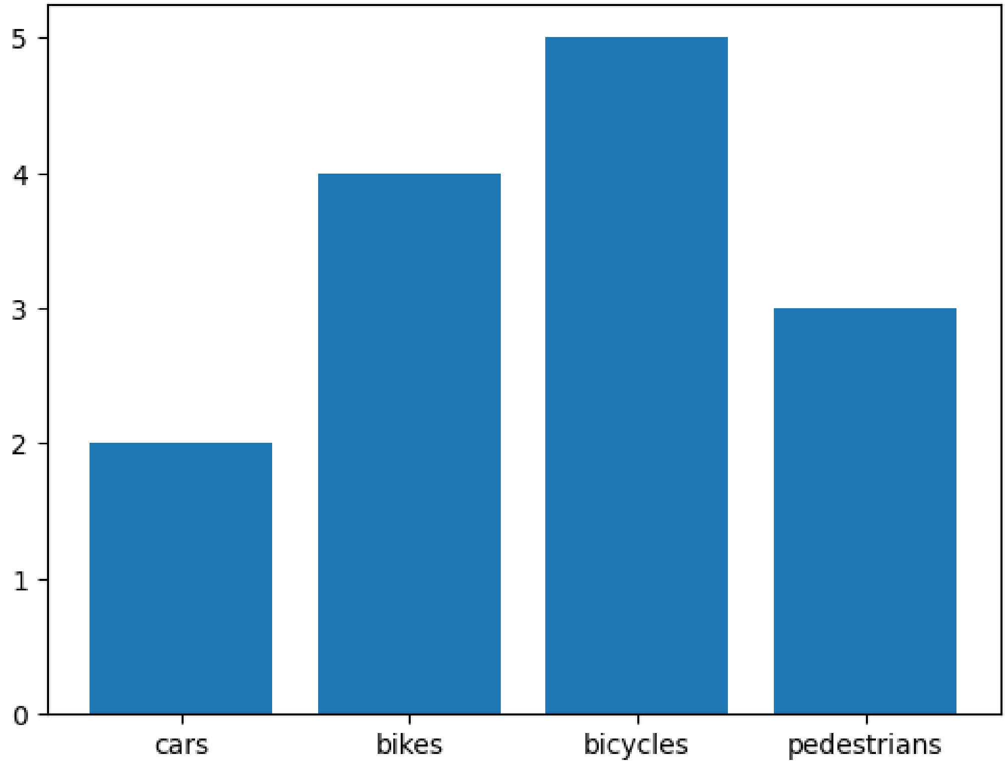
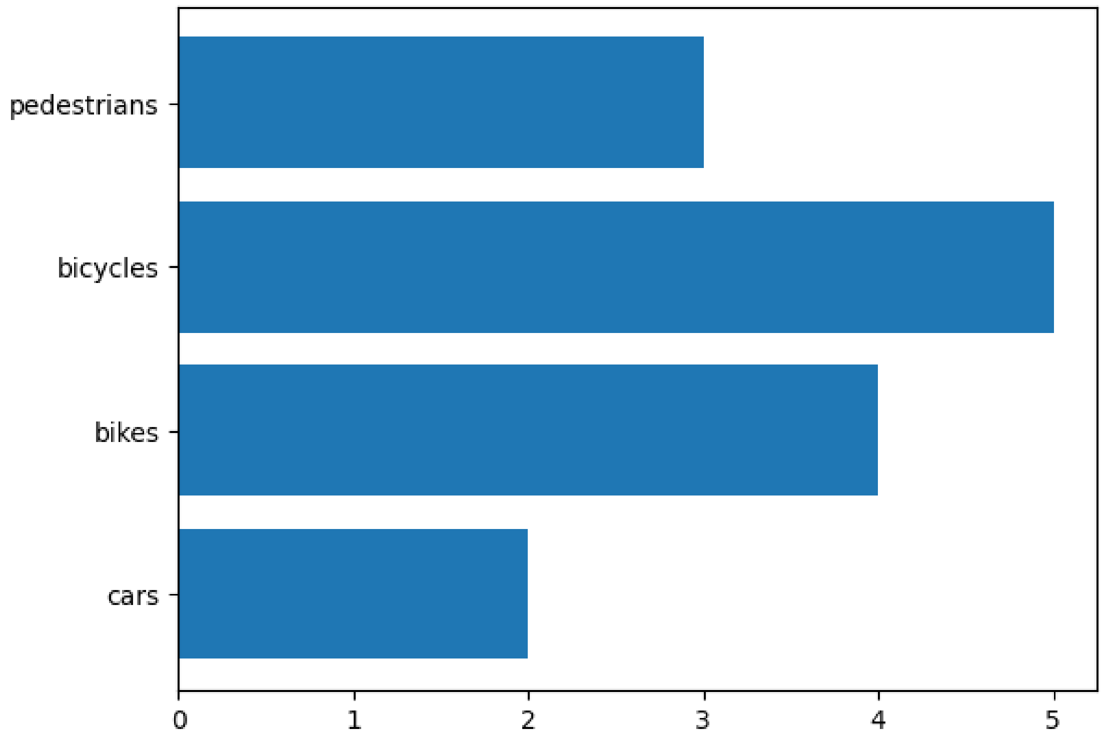
barchart-1

November 1, 2024

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

#plt.bar(x, height, width, bottom, align)


#Vertical Bar Graph
x = [1, 2, 3, 4]
height = [2, 4, 5 , 3]
labels = ['cars', 'bikes', 'bicycles', 'pedestrians']
y = np.arange(0.2,100)
plt.bar(x, height, align='center')
plt.xticks(x, labels)    #optional to set the class names for the bars
#plt.yticks(x, y)     #optional to set the values of y axis
plt.show()
```



[2]:
```
#Horizontal Bar Graph
x = [1, 2, 3, 4]
height = [2, 4, 5 , 3]
labels = ['cars', 'bikes', 'bicycles', 'pedestrians']
y = np.arange(0.2, 100)
plt.barh(x, height, align='center')
plt.yticks(x, labels)    #optional to set the class names for the bars
#plt.xticks(x, y)     #optional to set the values of y axis
plt.show()
```

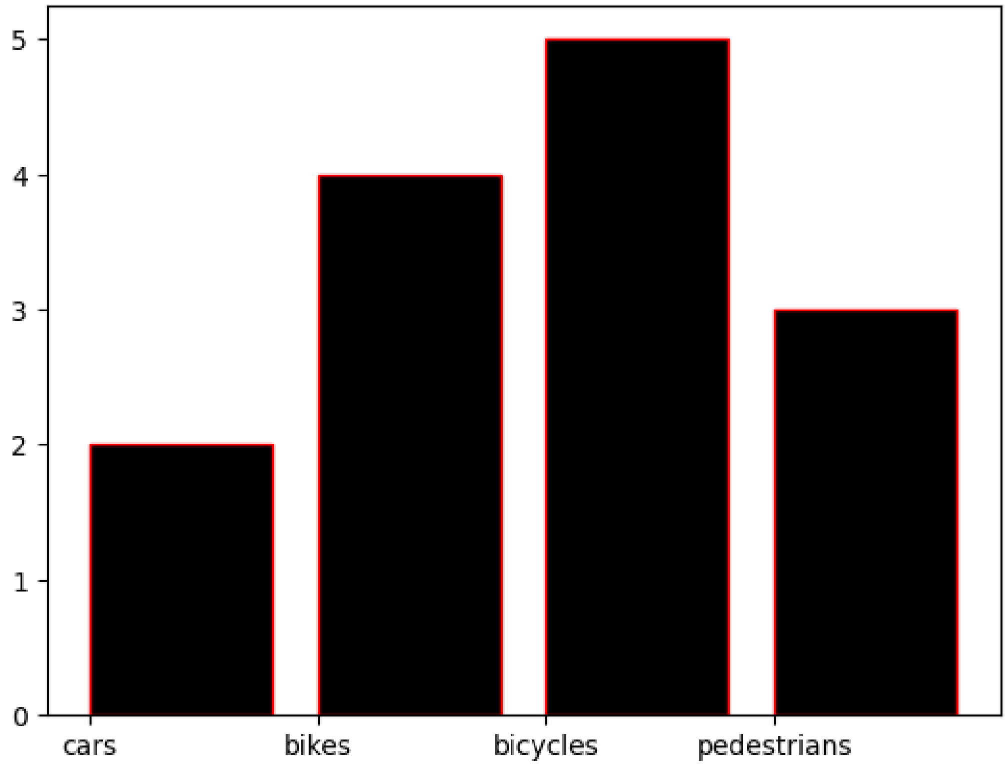


[3]:
```
y
```

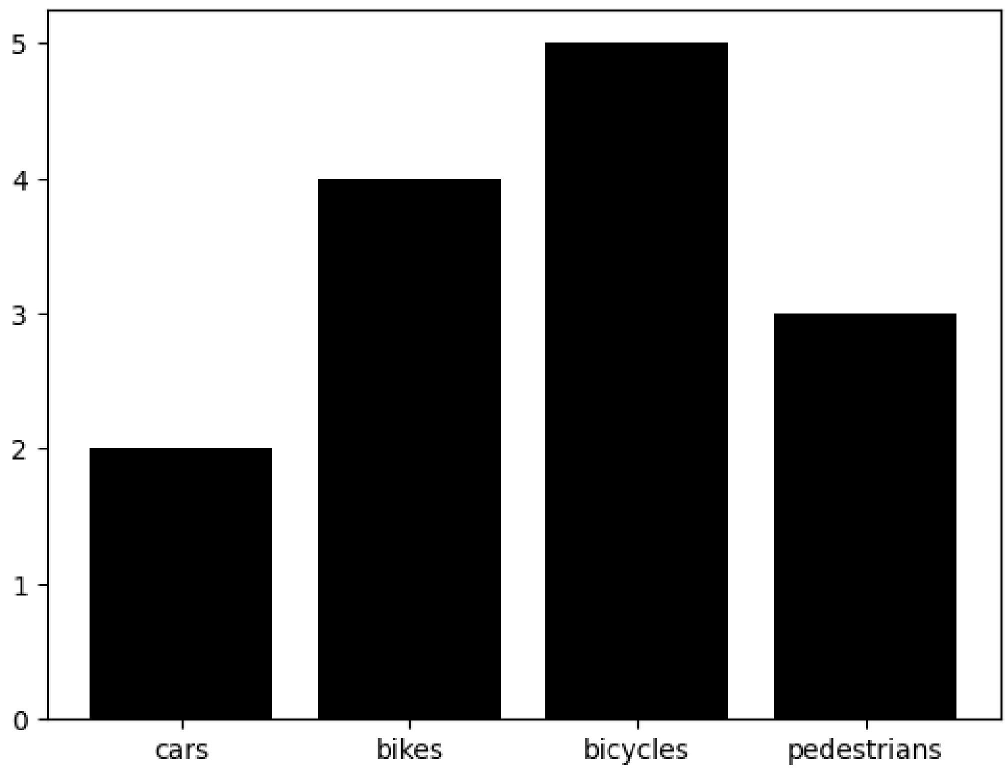
[3]:
```
array([ 0.2,  1.2,  2.2,  3.2,  4.2,  5.2,  6.2,  7.2,  8.2,  9.2, 10.2,
       11.2, 12.2, 13.2, 14.2, 15.2, 16.2, 17.2, 18.2, 19.2, 20.2, 21.2,
       22.2, 23.2, 24.2, 25.2, 26.2, 27.2, 28.2, 29.2, 30.2, 31.2, 32.2,
       33.2, 34.2, 35.2, 36.2, 37.2, 38.2, 39.2, 40.2, 41.2, 42.2, 43.2,
       44.2, 45.2, 46.2, 47.2, 48.2, 49.2, 50.2, 51.2, 52.2, 53.2, 54.2,
       55.2, 56.2, 57.2, 58.2, 59.2, 60.2, 61.2, 62.2, 63.2, 64.2, 65.2,
       66.2, 67.2, 68.2, 69.2, 70.2, 71.2, 72.2, 73.2, 74.2, 75.2, 76.2,
       77.2, 78.2, 79.2, 80.2, 81.2, 82.2, 83.2, 84.2, 85.2, 86.2, 87.2,
```

```
88.2, 89.2, 90.2, 91.2, 92.2, 93.2, 94.2, 95.2, 96.2, 97.2, 98.2,
99.2])
```

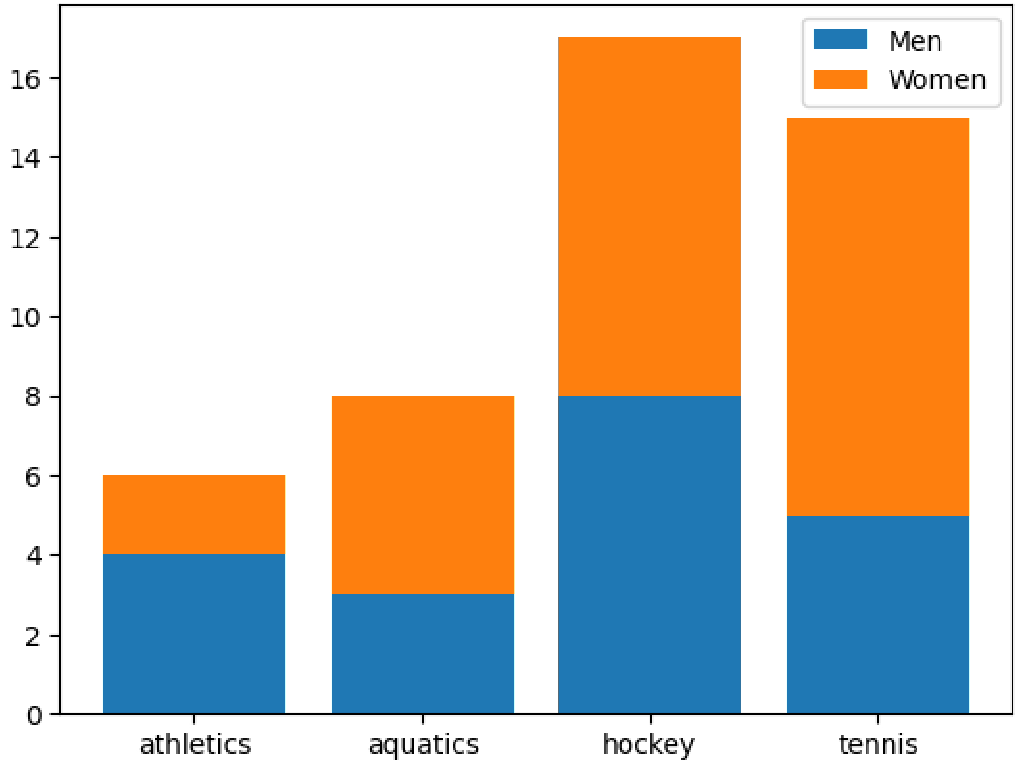
[4]:
```
#edge aligned bar charts
plt.bar(x, height, align='edge',ec='red',color='black')
plt.xticks(x, labels)
plt.show()
```



[5]:
```
#setting the colours of the bars
plt.bar(x, height, color='black')
plt.xticks(x, labels)
plt.show()
```



[6]:
```
#stacked bar chart
x = [1, 2, 3, 4]
men = [4, 3, 8, 5]
women = [2, 5, 9, 10]
labels = ['athletics', 'aquatics', 'hockey', 'tennis']
p1 = plt.bar(x, men)
p2 = plt.bar(x, women, bottom=men)
plt.xticks(x, labels)
plt.legend((p1[0], p2[0]), ('Men', 'Women'))
plt.show()
```

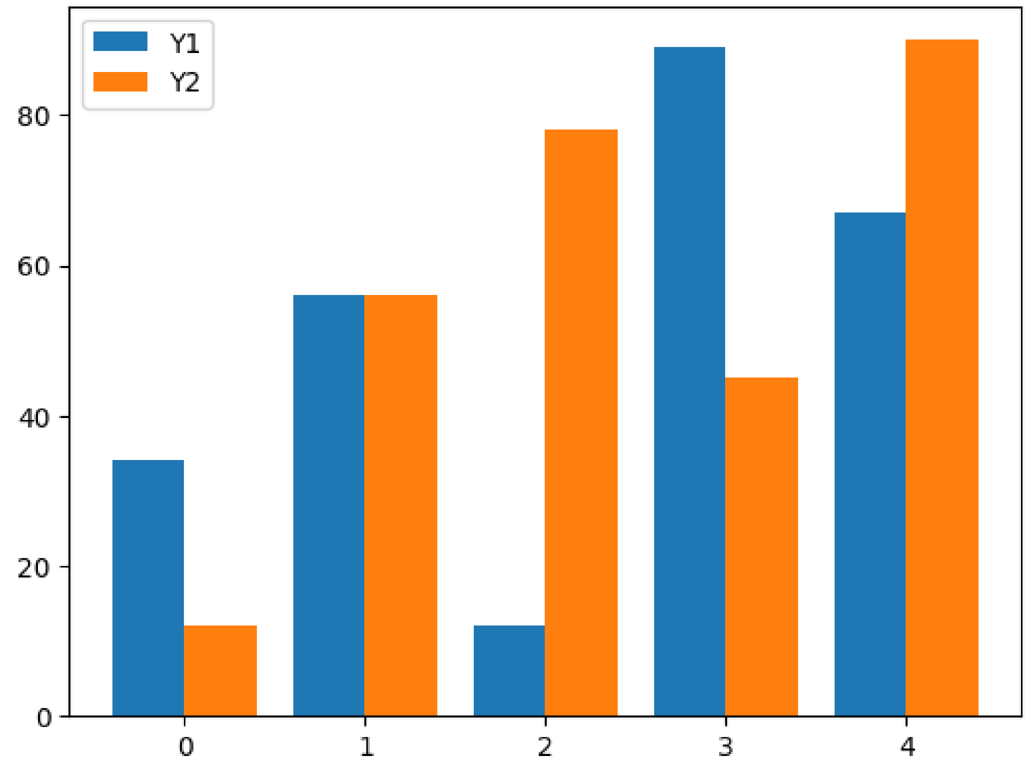


[7]:
```
np.arange(0.2,100)
```

[7]:
```
array([ 0.2,  1.2,  2.2,  3.2,  4.2,  5.2,  6.2,  7.2,  8.2,  9.2, 10.2,
       11.2, 12.2, 13.2, 14.2, 15.2, 16.2, 17.2, 18.2, 19.2, 20.2, 21.2,
       22.2, 23.2, 24.2, 25.2, 26.2, 27.2, 28.2, 29.2, 30.2, 31.2, 32.2,
       33.2, 34.2, 35.2, 36.2, 37.2, 38.2, 39.2, 40.2, 41.2, 42.2, 43.2,
       44.2, 45.2, 46.2, 47.2, 48.2, 49.2, 50.2, 51.2, 52.2, 53.2, 54.2,
       55.2, 56.2, 57.2, 58.2, 59.2, 60.2, 61.2, 62.2, 63.2, 64.2, 65.2,
       66.2, 67.2, 68.2, 69.2, 70.2, 71.2, 72.2, 73.2, 74.2, 75.2, 76.2,
       77.2, 78.2, 79.2, 80.2, 81.2, 82.2, 83.2, 84.2, 85.2, 86.2, 87.2,
       88.2, 89.2, 90.2, 91.2, 92.2, 93.2, 94.2, 95.2, 96.2, 97.2, 98.2,
       99.2])
```

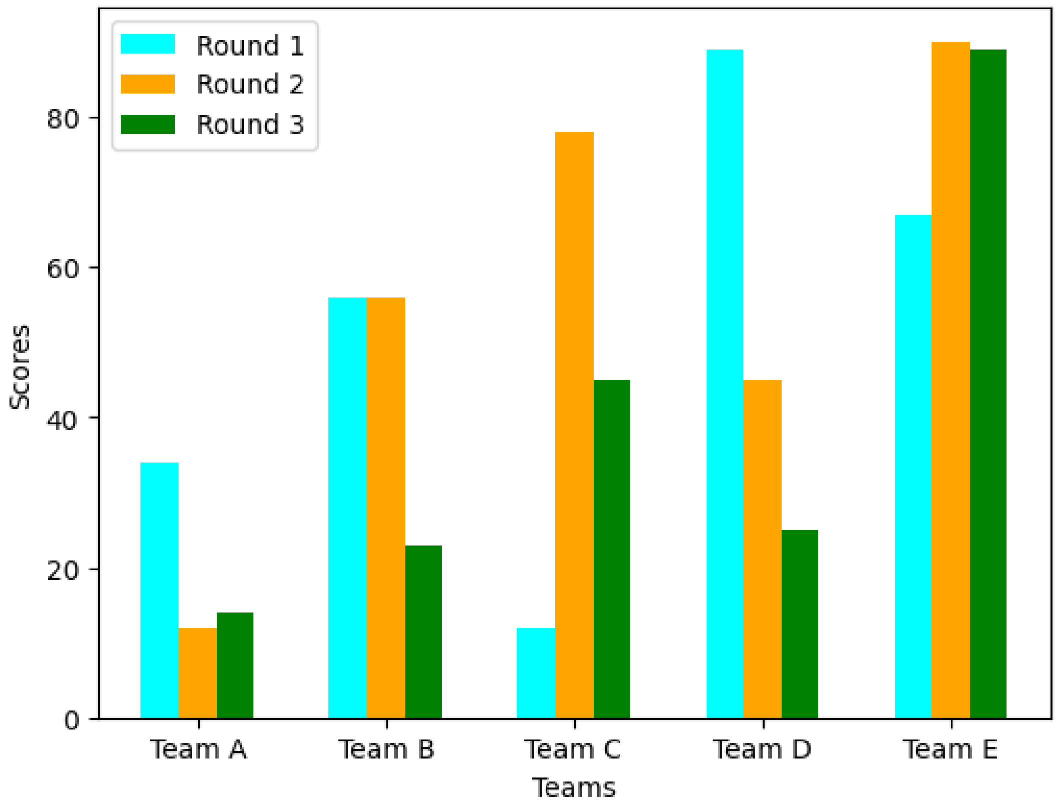
[8]:
```
# importing package
import matplotlib.pyplot as plt
import numpy as np

# create data
x = np.arange(5)    #0,1,2,3,4
y1 = [34, 56, 12, 89, 67]
y2 = [12, 56, 78, 45, 90]
width = 0.40
```

```
# plot data in grouped manner of bar type
p1=plt.bar(x-0.2, y1, width)
p2=plt.bar(x+0.2, y2, width)
plt.legend((p1[0], p2[0]), ('Y1', 'Y2'))
plt.show()
```



[9]:
```
# importing package
import matplotlib.pyplot as plt
import numpy as np

# create data
x = np.arange(5)
y1 = [34, 56, 12, 89, 67]
y2 = [12, 56, 78, 45, 90]
y3 = [14, 23, 45, 25, 89]
width = 0.2

# plot data in grouped manner of bar type
plt.bar(x-0.2, y1, width, color='cyan')
plt.bar(x, y2, width, color='orange')
plt.bar(x+0.2, y3, width, color='green')
```

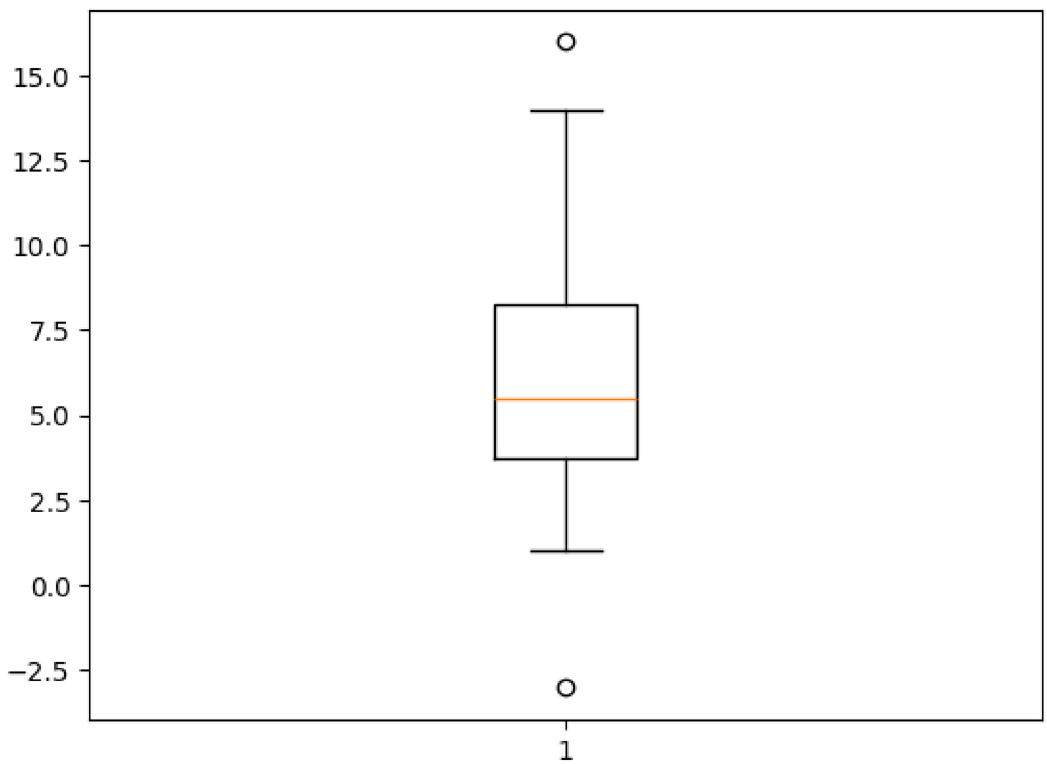
```
plt.xticks(x, ['Team A', 'Team B', 'Team C', 'Team D', 'Team E'])
plt.xlabel("Teams")
plt.ylabel("Scores")
plt.legend(["Round 1", "Round 2", "Round 3"])
plt.show()
```



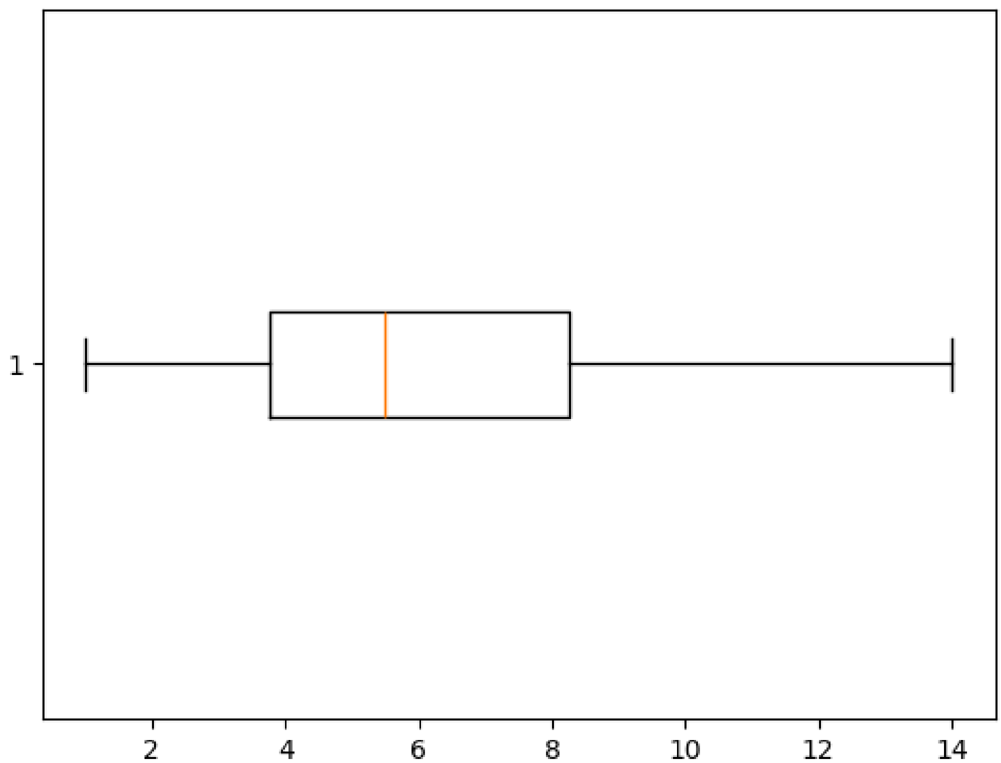
boxplot

November 1, 2024

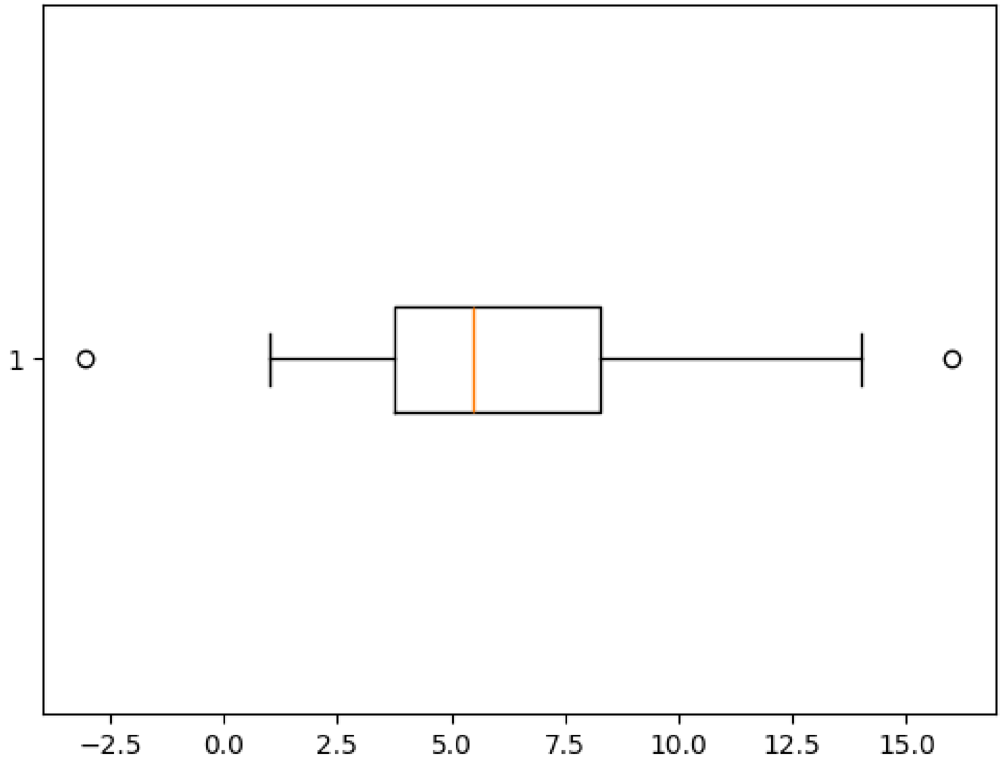
[1]:
```
import matplotlib.pyplot as plt
from matplotlib.pyplot import boxplot, show   #libraries req for boxplot
values = [2, 3, 4, 1, -3.04, 5, 4, 6, 7, 2, 4, 6, 8, 6 , 9, 12, 14, 11, 5, 16] ␣
 ↪ #datapoints need not be ordered
plt.boxplot(values, vert=True,showfliers=True)     #simple way to create a␣
 ↪boxplot
plt.show()
#aka whisker plot
```



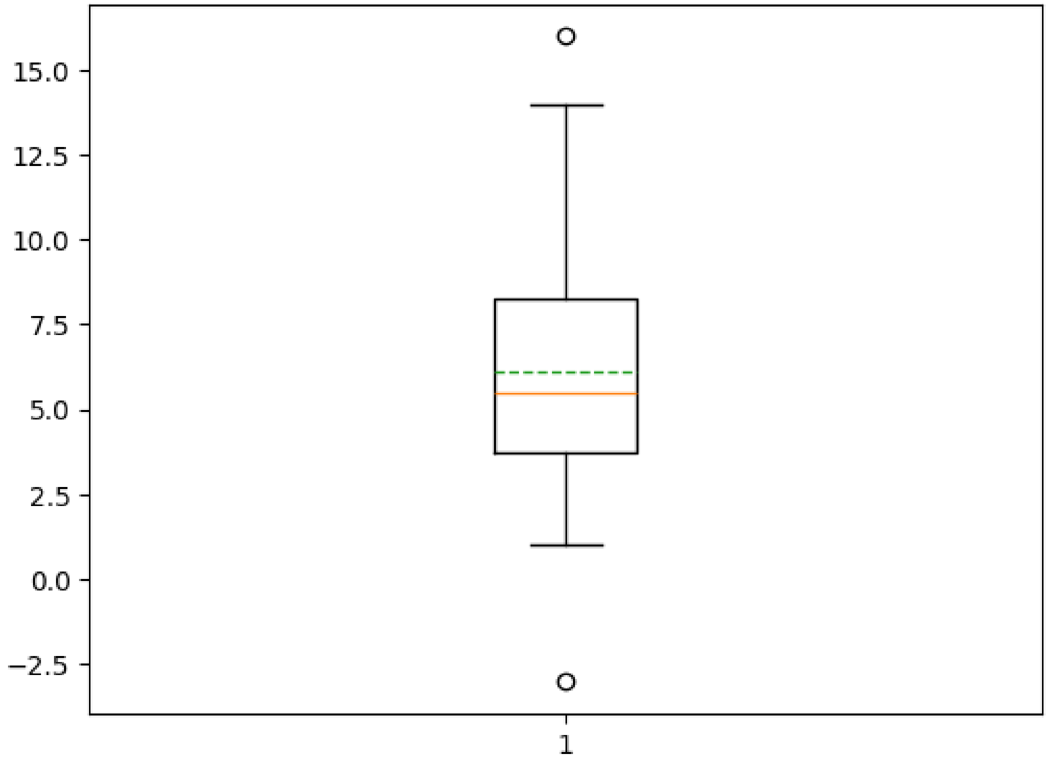
[2]:
```
plt.boxplot(values, showfliers=False, vert=False)   #to remove all the outliers
plt.show()
```



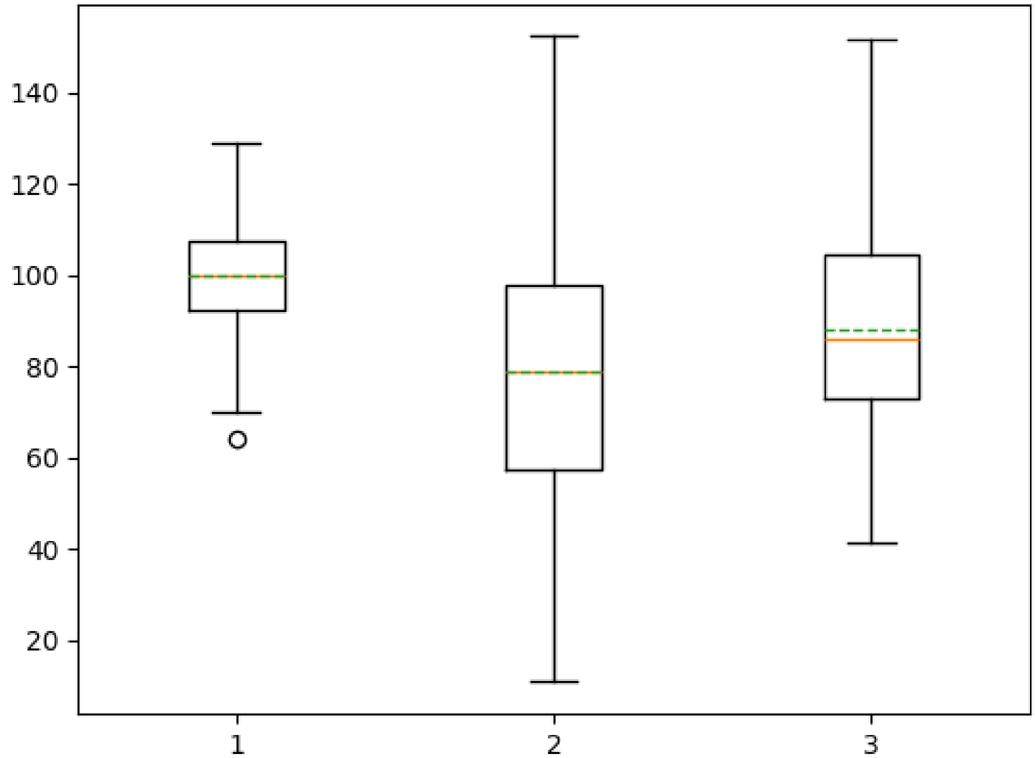
[5]:
```
plt.boxplot(values,vert=False)    #to consider all the outliers under the range
plt.show()
```




```
[6]: plt.boxplot(values, meanline=True, showmeans=True, vert=True)    #to show the␣
     ↪mean of the datapoints
     plt.show()
```



[7]:
```
#to plot multiple boxplots in one plane
import numpy as np
collectn_1 = np.random.normal(100, 10, 200)   #random generation of datapoints
collectn_2 = np.random.normal(80, 30, 200)
collectn_3 = np.random.normal(90, 20, 200)
values = [collectn_1, collectn_2, collectn_3]    #list of lists of datapoints
plt.boxplot(values,showmeans=True,meanline=True)
plt.show()
```

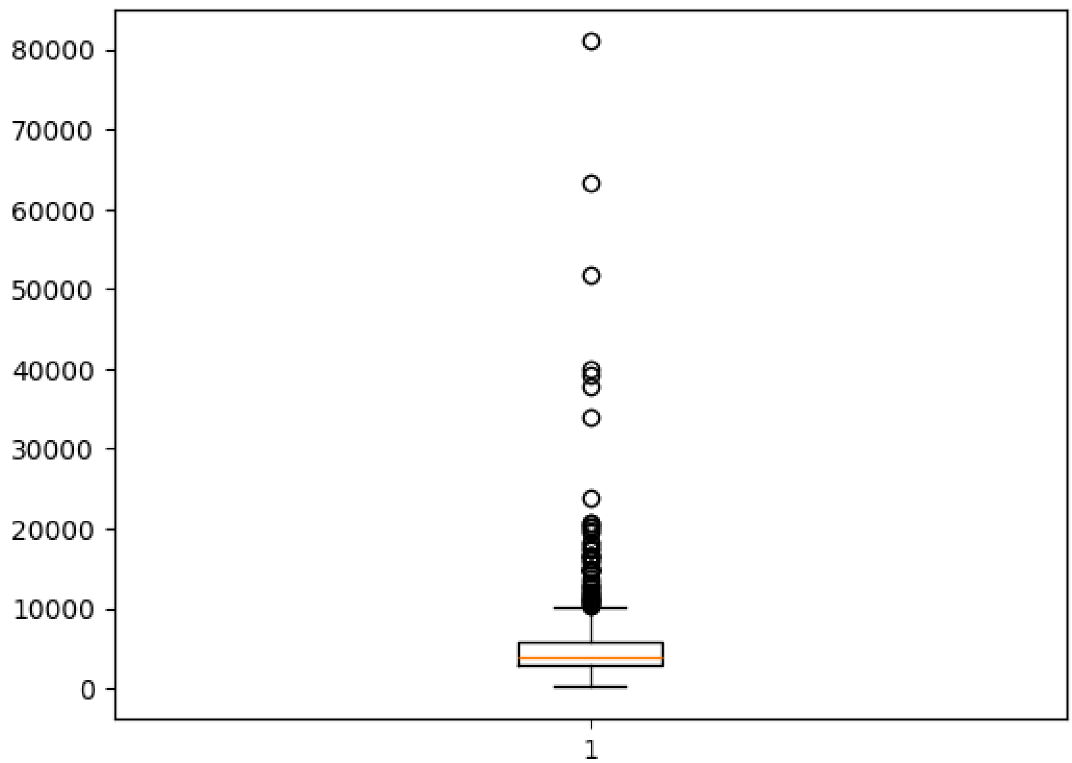


[8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import statistics as st
df = pd.read_csv("train.csv")
df
```

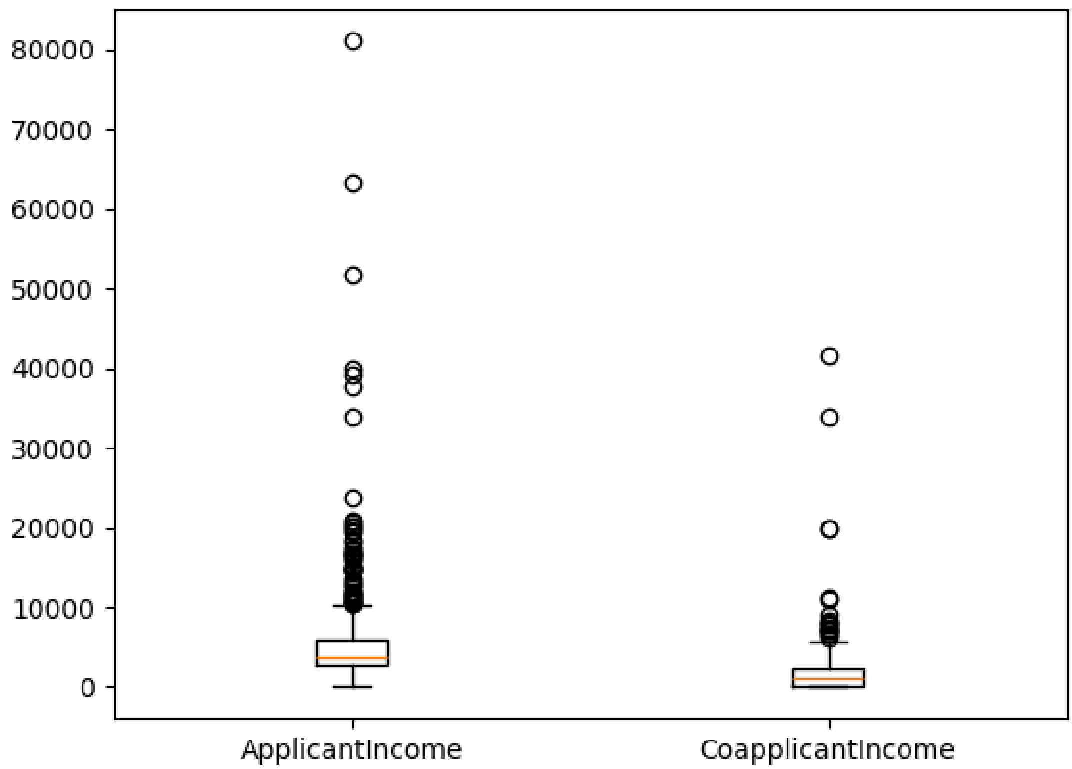
[8]:
```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
```

```
0              5849          0.0        NaN            360.0
1              4583       1508.0      128.0            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
..               …             …          …              …
609            2900          0.0       71.0            360.0
610            4106          0.0       40.0            180.0
611            8072        240.0      253.0            360.0
612            7583          0.0      187.0            360.0
613            4583          0.0      133.0            360.0

     Credit_History Property_Area Loan_Status
0              1.0         Urban           Y
1              1.0         Rural           N
2              1.0         Urban           Y
3              1.0         Urban           Y
4              1.0         Urban           Y
..             …             …         …
609            1.0         Rural           Y
610            1.0         Rural           Y
611            1.0         Urban           Y
612            1.0         Urban           Y
613            0.0     Semiurban           N

[614 rows x 13 columns]
```

[9]:
```
values=df['ApplicantIncome']
plt.boxplot(values, vert=True)    #simple way to create a boxplot
plt.show()
```



[10]:
```
v1=df['ApplicantIncome']
v2=df['CoapplicantIncome']

values=[v1,v2]
plt.boxplot(values, vert=True,labels=['ApplicantIncome','CoapplicantIncome'])
plt.show()
```


80000
70000
60000
50000
40000
30000
20000
10000
0
ApplicantIncome
CoapplicantIncome

chi-square

November 1, 2024

[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 0.103197140473O9392
Independent (H0 holds true)
```

[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}")
```

```python
# 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.
```

[3]:
```python
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
    [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
```

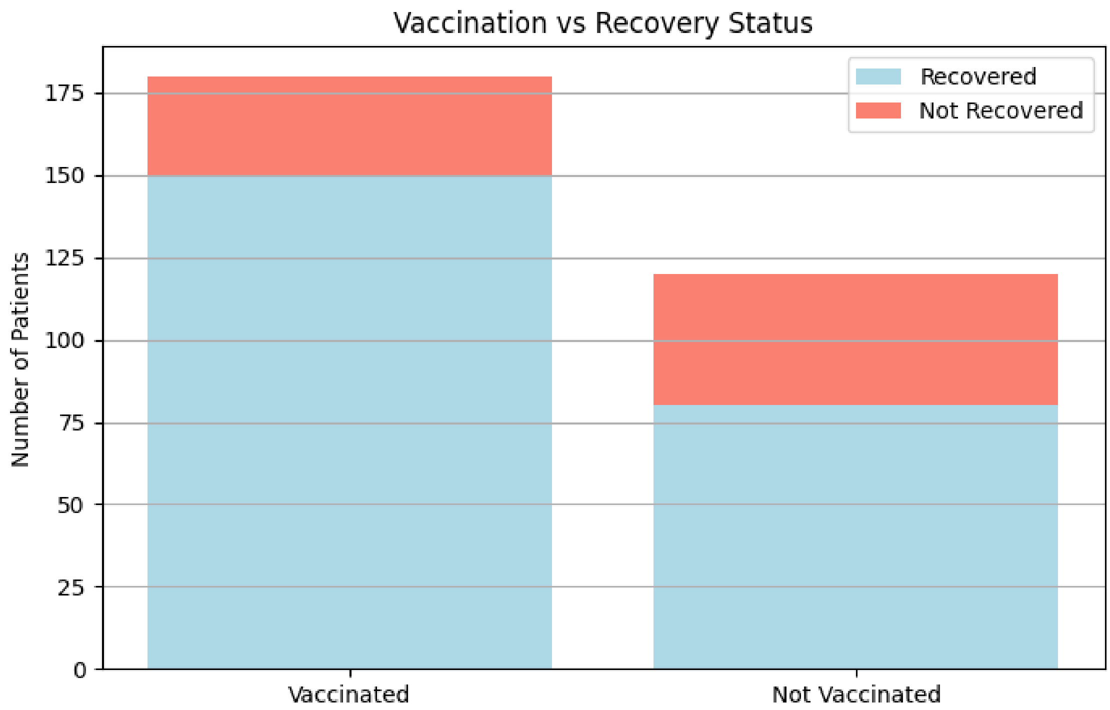
```
Machine 1        10  102          8
Machine 2        34  161          5
Machine 3        12   79          9
Machine 4        10   60         10

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.
```

[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)

# 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␣
↪between vaccination and recovery.")
else:
    print("Fail to reject the null hypothesis: No significant association␣
↪between vaccination and recovery.")
```

```
# 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='lightblue')
plt.bar(['Vaccinated', 'Not Vaccinated'], [30, 40], label='Not Recovered',␣
 ↪color='salmon', bottom=[150, 80])
plt.ylabel('Number of Patients')
plt.legend()
plt.grid(axis='y')
plt.show()
```

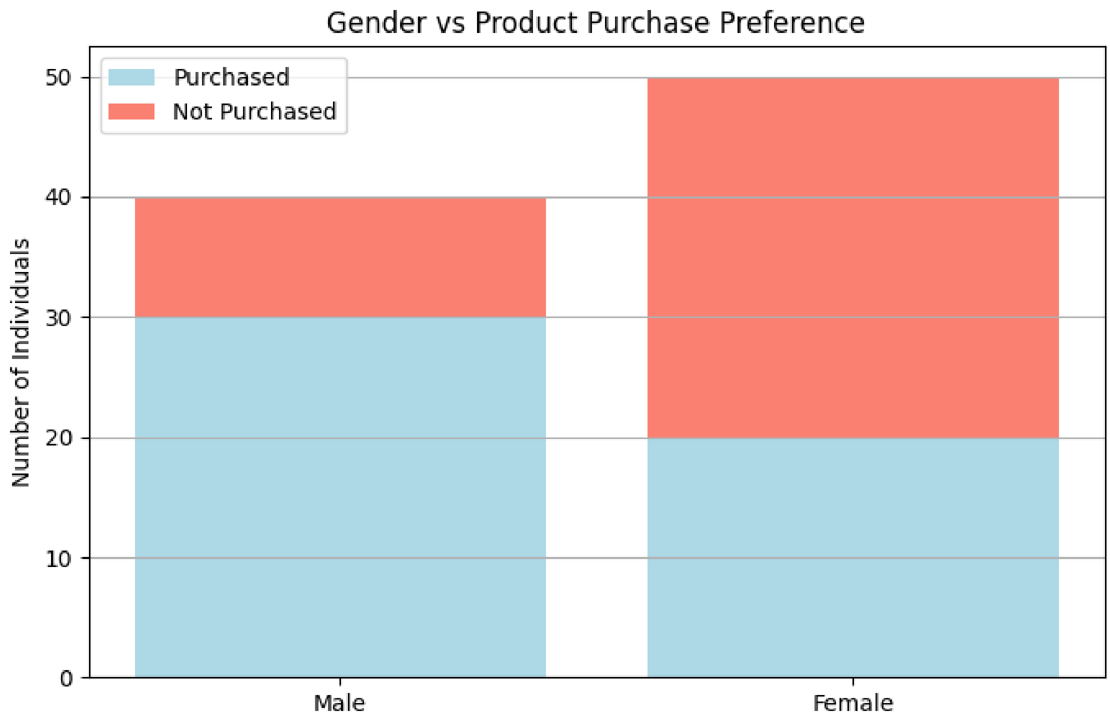
```
Contingency Table:
                 Recovered  Not Recovered
Vaccinated             150             30
Not Vaccinated          80             40

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
vaccination and recovery.
```



[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␣
 ↪between gender and product preference.")
else:
    print("Fail to reject the null hypothesis: No significant association␣
 ↪between gender and product preference.")

# 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',␣
 ↪bottom=[30, 20])
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.
```



clt

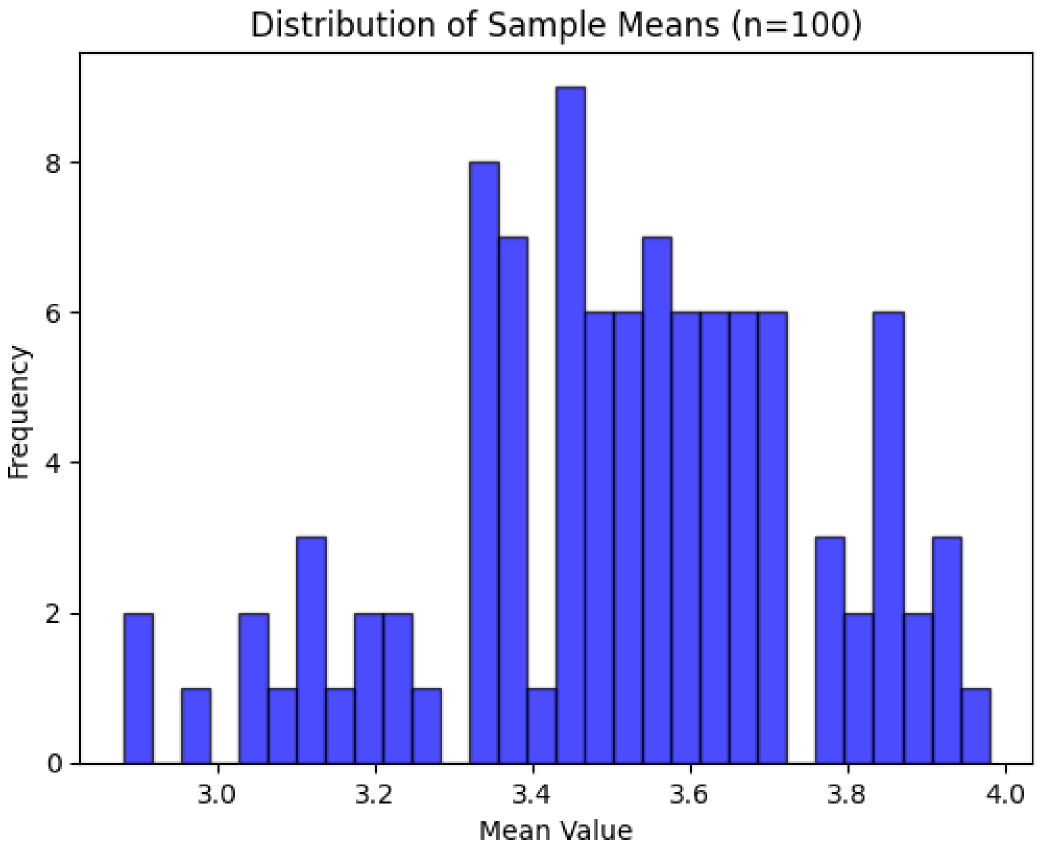
November 1, 2024

[3]:
```
# Import necessary functions and libraries
from numpy.random import seed # For setting a random seed
from numpy.random import randint # For generating random integers
from numpy import mean # For calculating the mean of an array
from matplotlib import pyplot # For plotting graphs
```

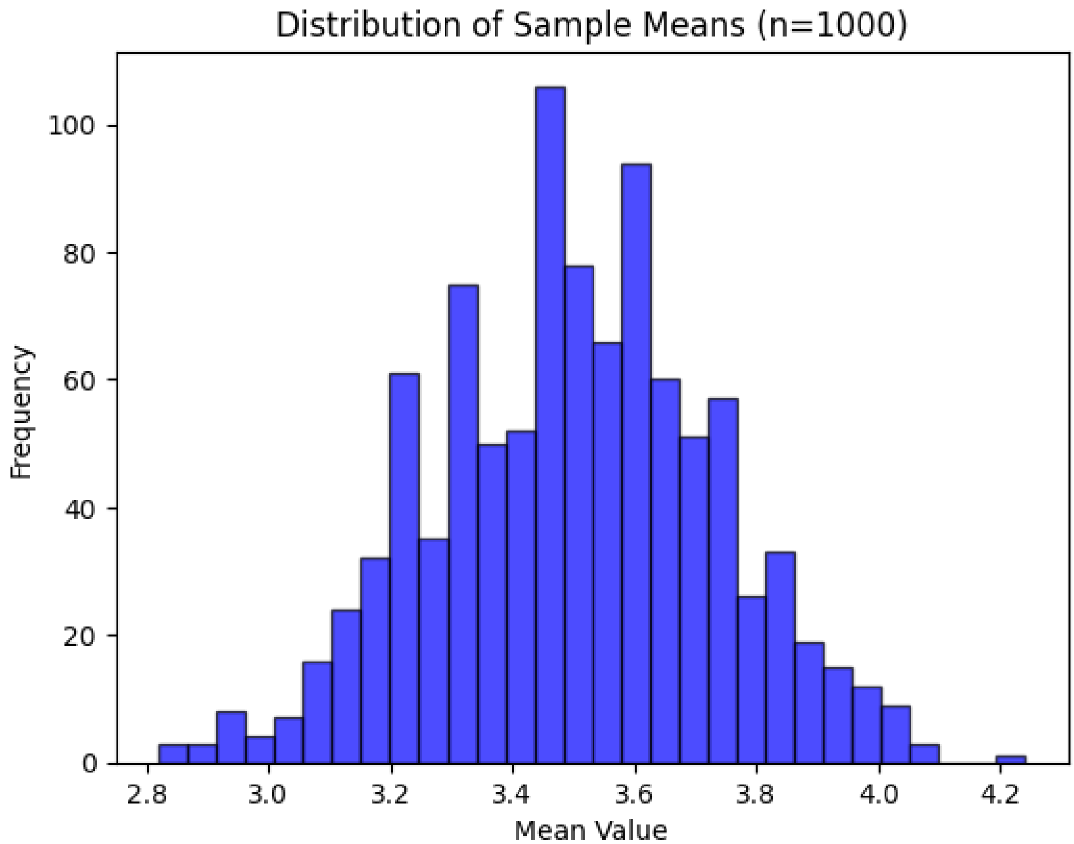
[4]:
```
# Seed the random number generator for reproducibility
seed(1)
```

[5]:
```
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 as a histogram
    pyplot.hist(means, bins=30, alpha=0.7, color='blue', edgecolor='black') #␣
↪Added bins, alpha, color, and edgecolor for better visibility
    pyplot.title(f'Distribution of Sample Means (n={n})') # Title indicating␣
↪the number of samples
    pyplot.xlabel('Mean Value') # Label for the x-axis
    pyplot.ylabel('Frequency') # Label for the y-axis
    pyplot.show() # Display the plot
```

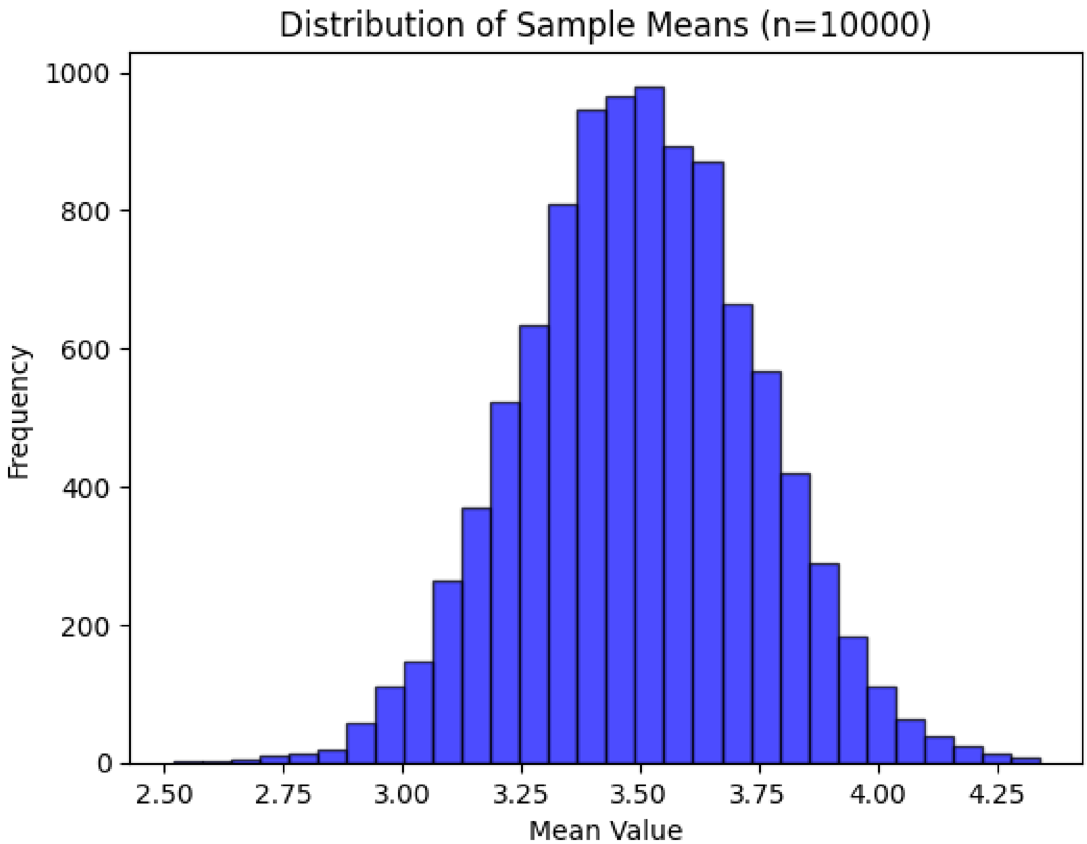
[6]:
```
# Call the function to plot the distribution of sample means for 100 samples
plot_clt(100)
```



[7]:
```
# Call the function to plot the distribution of sample means for 1000 samples
plot_clt(1000)
```



[9]:
```
# Call the function to plot the distribution of sample means for 10000 samples
plot_clt(10000)
```


Distribution of Sample Means (n=10000)
Frequency
Mean Value
1000
800
600
400
200
0
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25

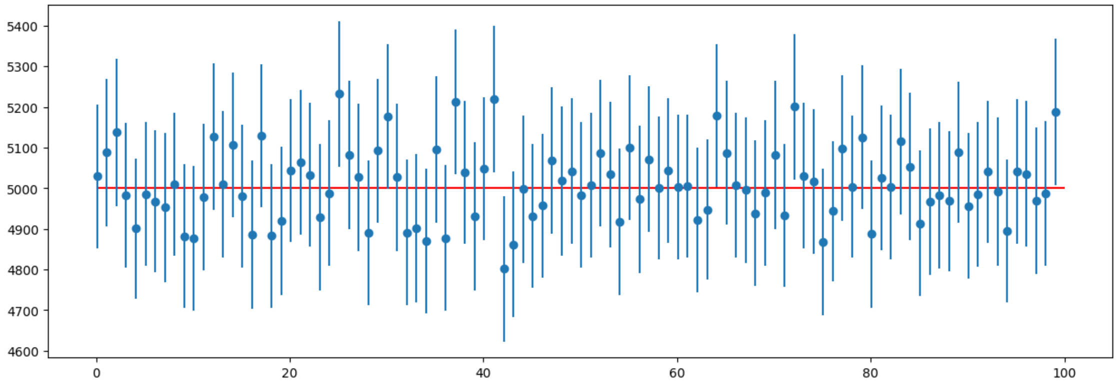
confidence-intervals

November 1, 2024

[2]:
```
%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␣
↪- low)/2 for low, upp in intervals])
    plt.show()
```

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

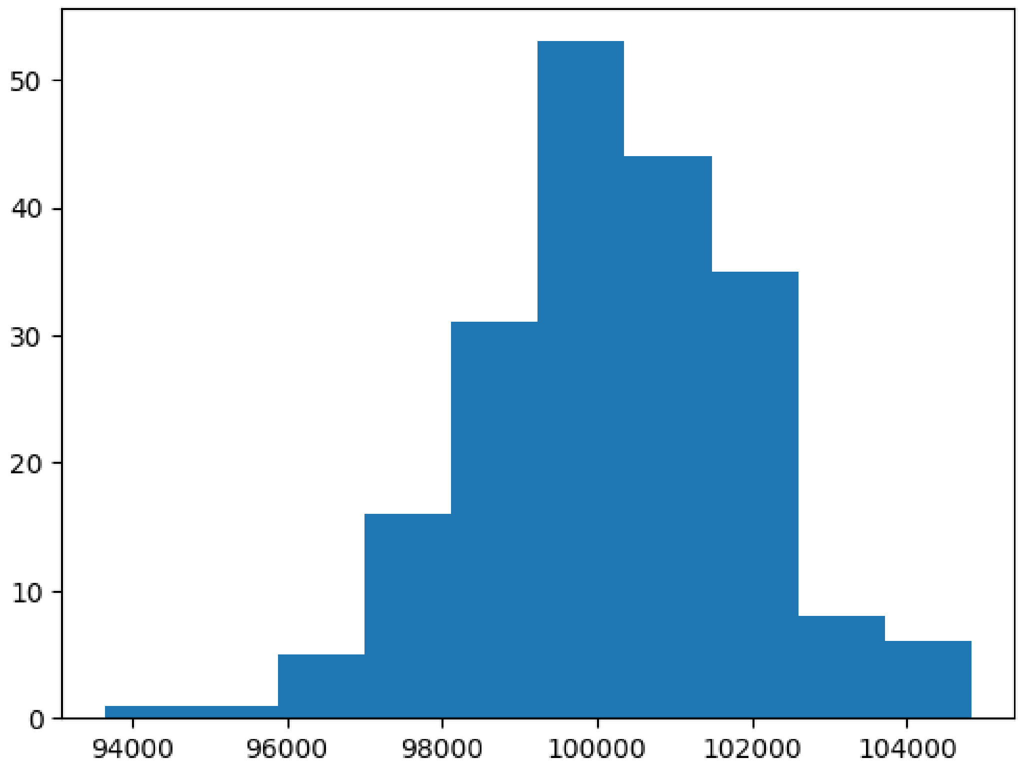
```
Proportion of CIs covering Pop mean 0.93
```



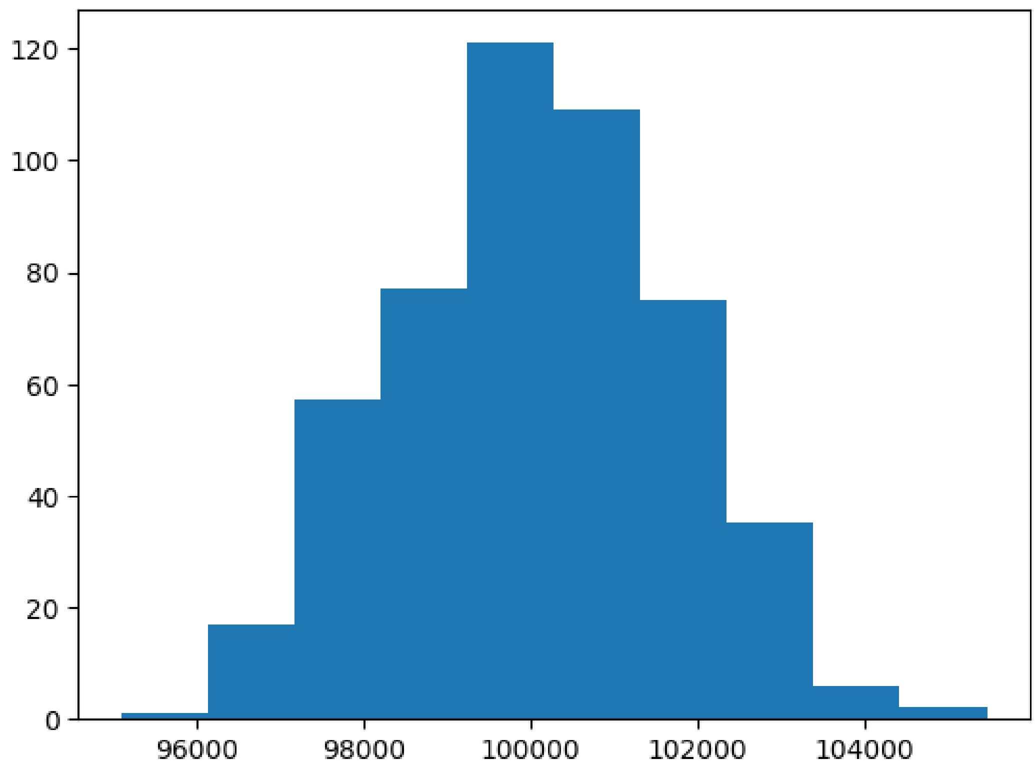
[3]:
```
#CI for population where 85% of the people say YES to a certain question
import numpy as np
import matplotlib.pyplot as plt
from random import sample
import scipy.stats as st
import math

#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
```

[4]:
```
#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␣
↪better approx of actual mean)
```

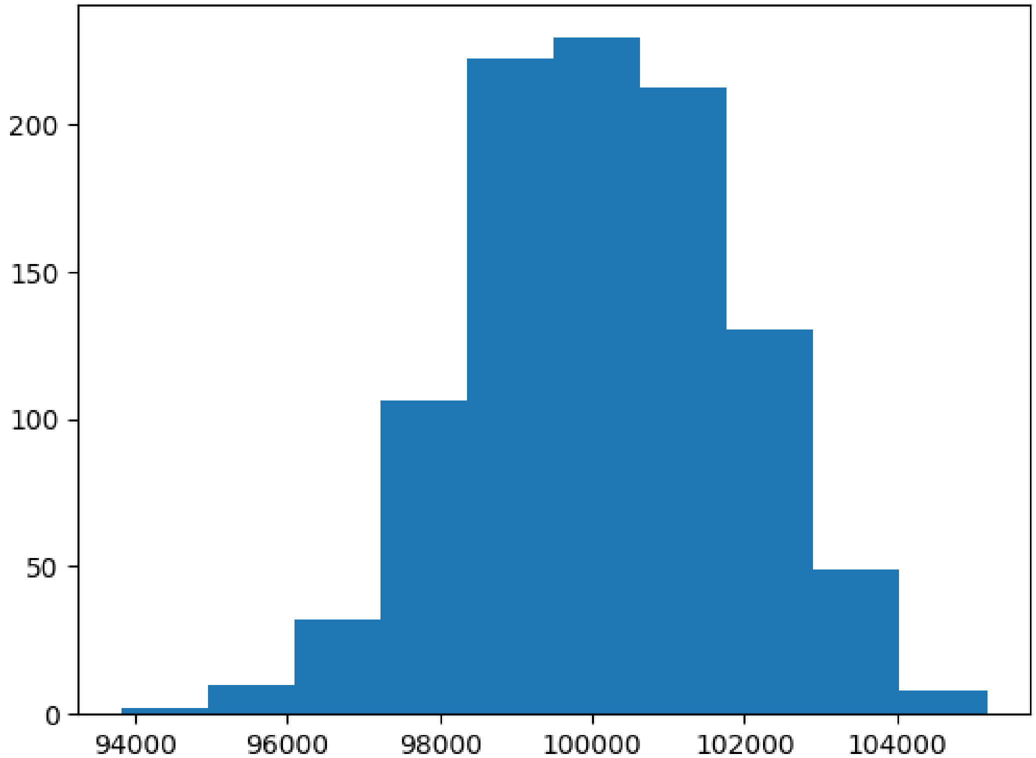
```
for ci of 0.85 sample_size 1000
actual mean : 100086.7646
mean of samples : 100198.62897500001
confidence interval : 100086.7646 +- 58.340833701766975
```




```
for ci of 0.85 sample_size 1000
actual mean : 100086.7646
mean of samples : 100089.081846
confidence interval : 100086.7646 +- 54.26313169132391
```

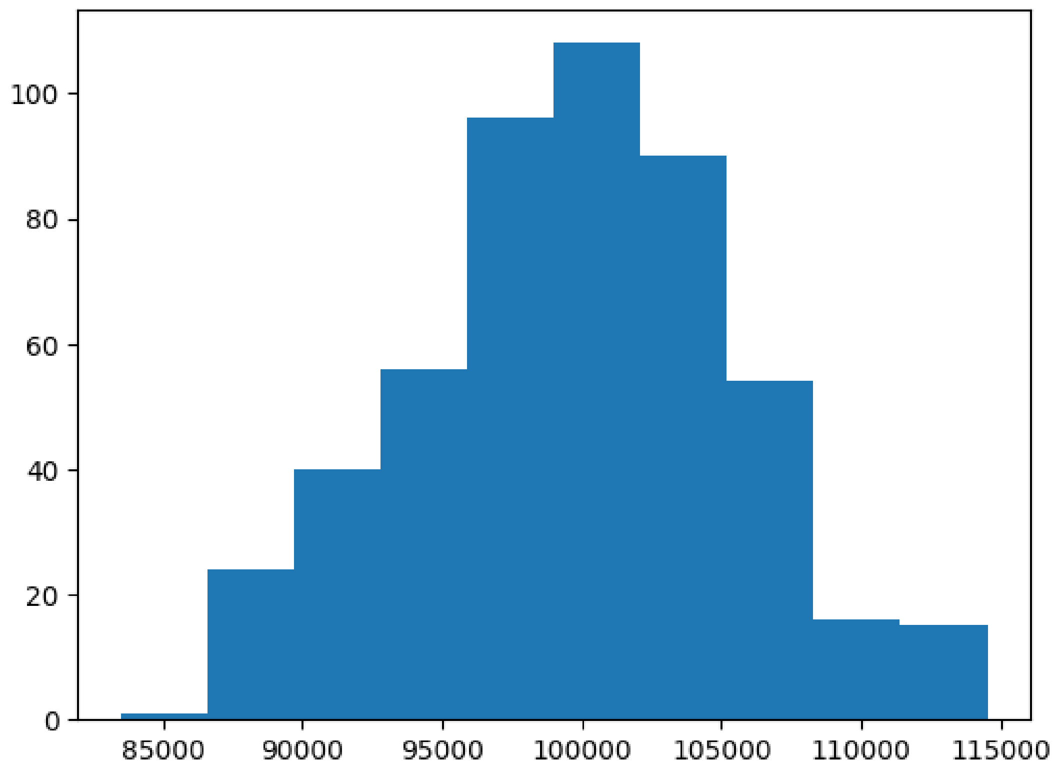



```
for ci of 0.85 sample_size 1000
actual mean : 100086.7646
mean of samples : 100140.380726
confidence interval : 100086.7646 +- 57.08417155868274
```

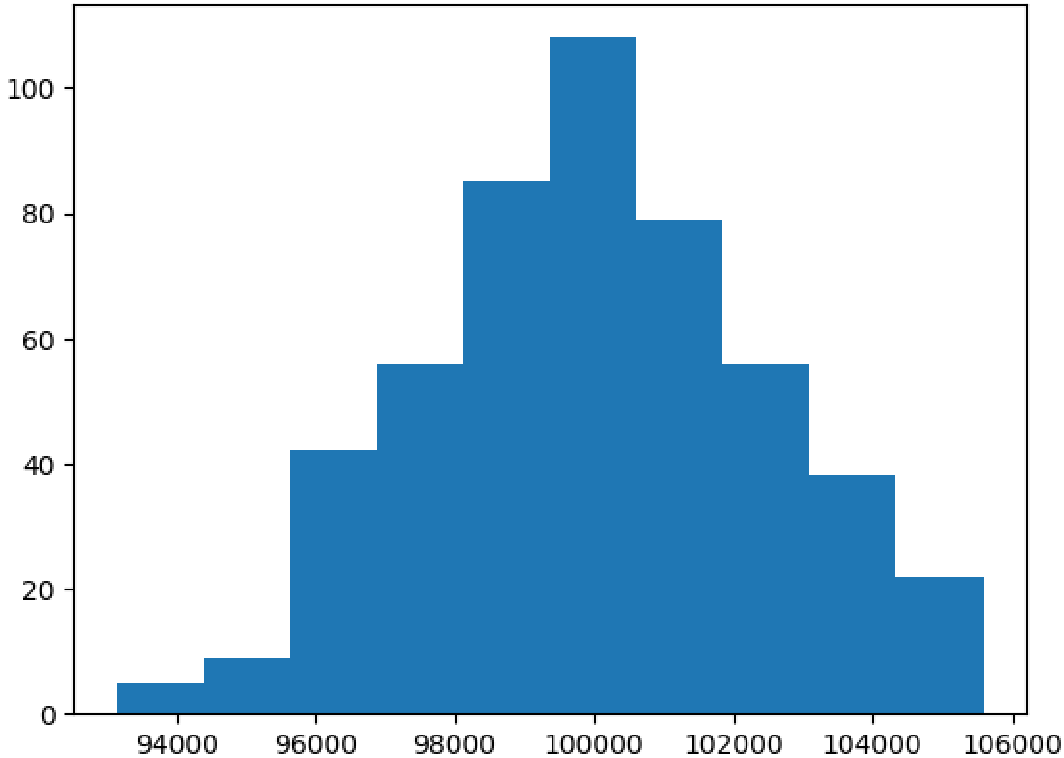


[5]:
```
#varying sample size
CI(pop, 0.85, 100, 500)
CI(pop, 0.85, 500, 500)
CI(pop, 0.85, 1000, 500)
#reduction in the size of interval as sample_size increases(better approx of␣
↪population)
```

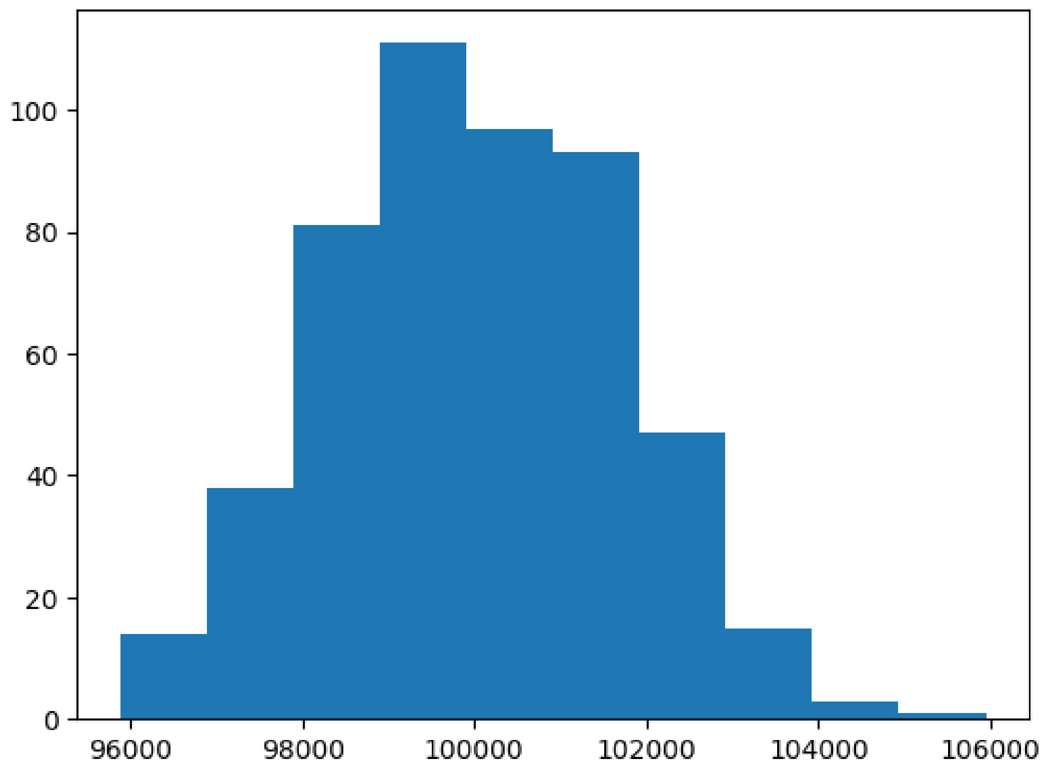
```
for ci of 0.85 sample_size 100
actual mean : 100086.7646
mean of samples : 99745.10796000001
confidence interval : 100086.7646 +- 604.1682474005653
```




```
for ci of 0.85 sample_size 500
actual mean : 100086.7646
mean of samples : 100001.622948
confidence interval : 100086.7646 +- 113.51762131217545
```




```
for ci of 0.85 sample_size 1000
actual mean : 100086.7646
mean of samples : 100021.621678
confidence interval : 100086.7646 +- 55.25179384960677
```


100
80
60
40
20
0
96000
98000
100000
102000
104000
106000

```
[ ]:
```

datacleaning

November 1, 2024

[1]:
```
# import the pandas library
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
'h'],columns=['one', 'two', 'three'])
print( df)
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

print( df)
print (df['one'].median())
print (df['one'].isnull())
#Total missing value for each attribute
print (df.isnull().sum())
#any missing values?
print (df['one'].isnull().values.any())
#Total no. of missing values
print (df.isnull().sum().sum())
```

```
        one       two     three
a  0.848768 -0.128940  0.578229
c -2.804692  1.306723  0.656576
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
h  2.104977 -0.764836  0.975284
        one       two     three
a  0.848768 -0.128940  0.578229
b       NaN       NaN       NaN
c -2.804692  1.306723  0.656576
d       NaN       NaN       NaN
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g       NaN       NaN       NaN
h  2.104977 -0.764836  0.975284
0.8487681538621735
a    False
b     True
c    False
```

```
d     True
e    False
f    False
g     True
h    False
Name: one, dtype: bool
one      3
two      3
three    3
dtype: int64
True
9
```

[2]:
```
print ("NaN replaced with '0':")
print( df.fillna(0))
```

```
NaN replaced with '0':
        one       two     three
a  0.848768 -0.128940  0.578229
b  0.000000  0.000000  0.000000
c -2.804692  1.306723  0.656576
d  0.000000  0.000000  0.000000
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g  0.000000  0.000000  0.000000
h  2.104977 -0.764836  0.975284
```

[3]:
```
print(df)
print( df.fillna(method='pad'))
```

```
        one       two     three
a  0.848768 -0.128940  0.578229
b       NaN       NaN       NaN
c -2.804692  1.306723  0.656576
d       NaN       NaN       NaN
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g       NaN       NaN       NaN
h  2.104977 -0.764836  0.975284
        one       two     three
a  0.848768 -0.128940  0.578229
b  0.848768 -0.128940  0.578229
c -2.804692  1.306723  0.656576
d -2.804692  1.306723  0.656576
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g -0.329088 -1.381245  1.210031
h  2.104977 -0.764836  0.975284
```

```
C:\Users\Prateek\AppData\Local\Temp\ipykernel_15920\1346297352.py:2:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  print( df.fillna(method='pad'))
```

[4]:
```
print(df)
print( df.fillna(method='bfill'))
```

```
        one       two     three
a  0.848768 -0.128940  0.578229
b       NaN       NaN       NaN
c -2.804692  1.306723  0.656576
d       NaN       NaN       NaN
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g       NaN       NaN       NaN
h  2.104977 -0.764836  0.975284
        one       two     three
a  0.848768 -0.128940  0.578229
b -2.804692  1.306723  0.656576
c -2.804692  1.306723  0.656576
d  1.042466 -0.982625  0.023920
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g  2.104977 -0.764836  0.975284
h  2.104977 -0.764836  0.975284
```

```
C:\Users\Prateek\AppData\Local\Temp\ipykernel_15920\190117098.py:2:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  print( df.fillna(method='bfill'))
```

[5]:
```
print(df)
print( df.dropna())
```

```
        one       two     three
a  0.848768 -0.128940  0.578229
b       NaN       NaN       NaN
c -2.804692  1.306723  0.656576
d       NaN       NaN       NaN
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g       NaN       NaN       NaN
h  2.104977 -0.764836  0.975284
        one       two     three
a  0.848768 -0.128940  0.578229
c -2.804692  1.306723  0.656576
e  1.042466 -0.982625  0.023920
```

```
f -0.329088 -1.381245  1.210031
h  2.104977 -0.764836  0.975284
```

[6]:

```
#Interpolation of immediate data before and after it (average is taken)

print(df.interpolate())
```

```
        one       two     three
a  0.848768 -0.128940  0.578229
b -0.977962  0.588891  0.617403
c -2.804692  1.306723  0.656576
d -0.881113  0.162049  0.340248
e  1.042466 -0.982625  0.023920
f -0.329088 -1.381245  1.210031
g  0.887945 -1.073041  1.092658
h  2.104977 -0.764836  0.975284
```

[7]:

```
import pandas as pd

df = pd.read_csv("loan_data_set.csv")      #paste entire file path
df.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[7], line 3
      1 import pandas as pd
----> 3 df = pd.read_csv("loan_data_set.csv")      #paste entire file path
      4 df.head()

File c:
 ↪\Users\Prateek\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\io\parsers\
 ↪py:1026, in read_csv(filepath_or_buffer, sep, delimiter, header, names,␣
 ↪index_col, usecols, dtype, engine, converters, true_values, false_values,␣
 ↪skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,␣
 ↪na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format,␣
 ↪keep_date_col, date_parser, date_format, dayfirst, cache_dates, iterator,␣
 ↪chunksize, compression, thousands, decimal, lineterminator, quotechar,␣
 ↪quoting, doublequote, escapechar, comment, encoding, encoding_errors, dialect␣
 ↪on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision,␣
 ↪storage_options, dtype_backend)
   1013 kwds_defaults = _refine_defaults_read(
   1014     dialect,
   1015     delimiter,
   (…)
   1022     dtype_backend=dtype_backend,
   1023 )
   1024 kwds.update(kwds_defaults)
-> 1026 return _read(filepath_or_buffer, kwds)
```

```
File c:
↪\Users\Prateek\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\io\parsers\
↪py:620, in _read(filepath_or_buffer, kwds)
    617 _validate_names(kwds.get("names", None))
    619 # Create the parser.
--> 620 parser = TextFileReader(filepath_or_buffer, **kwds)
    622 if chunksize or iterator:
    623     return parser

File c:
↪\Users\Prateek\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\io\parsers\
↪py:1620, in TextFileReader.__init__(self, f, engine, **kwds)
   1617     self.options["has_index_names"] = kwds["has_index_names"]
   1619 self.handles: IOHandles | None = None
-> 1620 self._engine = self._make_engine(f, self.engine)

File c:
↪\Users\Prateek\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\io\parsers\
↪py:1880, in TextFileReader._make_engine(self, f, engine)
   1878     if "b" not in mode:
   1879         mode += "b"
-> 1880 self.handles = get_handle(
   1881     f,
   1882     mode,
   1883     encoding=self.options.get("encoding", None),
   1884     compression=self.options.get("compression", None),
   1885     memory_map=self.options.get("memory_map", False),
   1886     is_text=is_text,
   1887     errors=self.options.get("encoding_errors", "strict"),
   1888     storage_options=self.options.get("storage_options", None),
   1889 )
   1890 assert self.handles is not None
   1891 f = self.handles.handle

File c:
↪\Users\Prateek\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\io\common.
↪py:873, in get_handle(path_or_buf, mode, encoding, compression, memory_map,␣
↪is_text, errors, storage_options)
    868 elif isinstance(handle, str):
    869     # Check whether the filename is to be opened in binary mode.
    870     # Binary mode does not support 'encoding' and 'newline'.
    871     if ioargs.encoding and "b" not in ioargs.mode:
    872         # Encoding
--> 873         handle = open(
    874             handle,
    875             ioargs.mode,
    876             encoding=ioargs.encoding,
    877             errors=errors,
    878             newline="",
```

```
            )
    else:
        # Binary mode
        handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'loan_data_set.csv'
```

[ ]:
```python
to_drop = ['Gender','Married']
#df.drop(columns=to_drop, inplace=True)
df.drop(to_drop, inplace=True, axis=1)
```

[ ]:
```python
df.head()
```

[ ]:
```
    Loan_ID Dependents     Education Self_Employed  ApplicantIncome  \
0  LP001002          0      Graduate            No             5849
1  LP001003          1      Graduate            No             4583
2  LP001005          0      Graduate           Yes             3000
3  LP001006          0  Not Graduate            No             2583
4  LP001008          0      Graduate            No             6000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                0.0         NaN             360.0             1.0
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0

  Property_Area Loan_Status
0         Urban           Y
1         Rural           N
2         Urban           Y
3         Urban           Y
4         Urban           Y
```

[ ]:
```python
df = pd.DataFrame({
    'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
    'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
    'rating': [4, 4, 3.5, 15, 5]
})
df
```

[ ]:
```
     brand  rating style
0  Yum Yum     4.0   cup
1  Yum Yum     4.0   cup
2  Indomie     3.5   cup
3  Indomie    15.0  pack
4  Indomie     5.0  pack
```

[ ]:
```
df.drop_duplicates()
```

[ ]:

	brand	rating	style
0	Yum Yum	4.0	cup
2	Indomie	3.5	cup
3	Indomie	15.0	pack
4	Indomie	5.0	pack

[ ]:
```
#To remove duplicates on specific column(s), use subset.
df.drop_duplicates(subset=['brand'])
```

[ ]:

	brand	rating	style
0	Yum Yum	4.0	cup
2	Indomie	3.5	cup

[ ]:
```
#To remove duplicates on specific column(s), use subset.
#to remove duplicates and keep last occurrences, use keep.
df.drop_duplicates(subset=['brand', 'style'], keep='last')
```

[ ]:

	brand	rating	style
1	Yum Yum	4.0	cup
2	Indomie	3.5	cup
4	Indomie	5.0	pack

[ ]:
```
#https://pandas.pydata.org/docs/reference/frame.html
```

datacleaning-file-1

November 1, 2024

[1]:
```python
import pandas as pd

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

[2]:
```python
print(df)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        …       …       …         …            …             …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             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
..               …                 …          …               …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
```

```
..             …          …        …
609          1.0      Rural           Y
610          1.0      Rural           Y
611          1.0      Urban           Y
612          1.0      Urban           Y
613          0.0  Semiurban           N

[614 rows x 13 columns]
```

[3]:
```
df.drop(['Dependents'], axis=1) #drop the column
```

[3]:
```
      Loan_ID  Gender Married     Education Self_Employed  ApplicantIncome  \
0    LP001002    Male      No      Graduate            No             5849
1    LP001003    Male     Yes      Graduate            No             4583
2    LP001005    Male     Yes      Graduate           Yes             3000
3    LP001006    Male     Yes  Not Graduate            No             2583
4    LP001008    Male      No      Graduate            No             6000
..        …       …     …           …          …              …
609  LP002978  Female      No      Graduate            No             2900
610  LP002979    Male     Yes      Graduate            No             4106
611  LP002983    Male     Yes      Graduate            No             8072
612  LP002984    Male     Yes      Graduate            No             7583
613  LP002990  Female      No      Graduate           Yes             4583

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                  0.0         NaN             360.0             1.0
1               1508.0       128.0             360.0             1.0
2                  0.0        66.0             360.0             1.0
3               2358.0       120.0             360.0             1.0
4                  0.0       141.0             360.0             1.0
..                 …         …               …             …
609                0.0        71.0             360.0             1.0
610                0.0        40.0             180.0             1.0
611              240.0       253.0             360.0             1.0
612                0.0       187.0             360.0             1.0
613                0.0       133.0             360.0             0.0

    Property_Area Loan_Status
0           Urban           Y
1           Rural           N
2           Urban           Y
3           Urban           Y
4           Urban           Y
..            …         …
609         Rural           Y
610         Rural           Y
611         Urban           Y
```

```
612         Urban           Y
613     Semiurban           N

[614 rows x 12 columns]
```

[4]:
```
df.drop([0, 1])   #drop the rows
```

[4]:
```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
5    LP001011    Male     Yes          2      Graduate           Yes
6    LP001013    Male     Yes          0  Not Graduate            No
..        …       …       …          …           …             …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
5               5417             4196.0       267.0             360.0
6               2333             1516.0        95.0             360.0
..               …                 …          …               …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
5               1.0         Urban           Y
6               1.0         Urban           Y
..              …           …         …
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[612 rows x 13 columns]
```

[5]:
```
df.columns[0]  #displays 1st column name
```

[5]: 'Loan_ID'

[6]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("train.csv")
print(df.replace(np.NaN,0))
#df['DataFrame Column'] = df['DataFrame Column'].replace(np.nan, 0)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        …     …     …        …          …           …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         0.0             360.0
1               4583             1508.0       128.0             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
..               …               …        …               …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              …         …        …
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
```

```
613             0.0     Semiurban           N

[614 rows x 13 columns]
```

[7]:
```
print ("NaN replaced with '0':")
print( df.fillna(method='pad'))
```

```
NaN replaced with '0':
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        …       …       …          …           …             …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             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
..               …                 …         …               …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              …           …         …
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]
```

```
C:\Users\Prateek\AppData\Local\Temp\ipykernel_17908\2002809902.py:2:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  print( df.fillna(method='pad'))
```

[8]:
```
print (df['Loan_ID'].isnull())
```

```
0      False
1      False
2      False
3      False
4      False
       …
609    False
610    False
611    False
612    False
613    False
Name: Loan_ID, Length: 614, dtype: bool
```

[9]:
```
print (df['Dependents'].notnull())
```

```
0      True
1      True
2      True
3      True
4      True
       …
609    True
610    True
611    True
612    True
613    True
Name: Dependents, Length: 614, dtype: bool
```

[10]:
```
print( df['Self_Employed'].isnull())
```

```
0      False
1      False
2      False
3      False
4      False
       …
609    False
610    False
611    False
612    False
613    False
```

```
Name: Self_Employed, Length: 614, dtype: bool
```

[11]:
```
print(df)
print ("NaN replaced with '0':")
print( df.fillna(0))
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             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
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              ...           ...         ...
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]
NaN replaced with '0':
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
```

```
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        …        …       …         …           …             …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         0.0             360.0
1               4583             1508.0       128.0             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
..               …                 …          …               …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              …           …         …
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]
```

[12]:
```
df = df.dropna() #drops rows with null values
```

[13]:
```
print(df)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
```

```
4    LP001008    Male      No          0      Graduate            No
5    LP001011    Male     Yes          2      Graduate           Yes
..        …        …       …        …          …          …
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
1               4583             1508.0       128.0             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
5               5417             4196.0       267.0             360.0
..                …                  …          …                 …
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
5               1.0         Urban           Y
..               …            …          …
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[480 rows x 13 columns]
```

[14]:
```python
import pandas as pd
import numpy as np
df = pd.read_csv("train.csv")
df.head()
```

[14]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
```

```
4  LP001008    Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             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

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[15]:
```
to_drop = ['Gender','Married']
#df.drop(columns=to_drop, inplace=True)
df.drop(to_drop, inplace=True, axis=1)
```

[16]:
```
df.head()
```

[16]:
```
    Loan_ID Dependents     Education Self_Employed  ApplicantIncome  \
0  LP001002          0      Graduate            No             5849
1  LP001003          1      Graduate            No             4583
2  LP001005          0      Graduate           Yes             3000
3  LP001006          0  Not Graduate            No             2583
4  LP001008          0      Graduate            No             6000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                0.0         NaN             360.0             1.0
1             1508.0       128.0             360.0             1.0
2                0.0        66.0             360.0             1.0
3             2358.0       120.0             360.0             1.0
4                0.0       141.0             360.0             1.0

  Property_Area Loan_Status
0         Urban           Y
1         Rural           N
2         Urban           Y
3         Urban           Y
4         Urban           Y
```

[17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
```

```
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Dependents         599 non-null    object 
 2   Education          614 non-null    object 
 3   Self_Employed      582 non-null    object 
 4   ApplicantIncome    614 non-null    int64  
 5   CoapplicantIncome  614 non-null    float64
 6   LoanAmount         592 non-null    float64
 7   Loan_Amount_Term   600 non-null    float64
 8   Credit_History     564 non-null    float64
 9   Property_Area      614 non-null    object 
 10  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(6)
memory usage: 52.9+ KB
```

[18]:
```
df.shape
```

[18]:
```
(614, 11)
```

[19]:
```
df.count()
```

[19]:
```
Loan_ID              614
Dependents           599
Education            614
Self_Employed        582
ApplicantIncome      614
CoapplicantIncome    614
LoanAmount           592
Loan_Amount_Term     600
Credit_History       564
Property_Area        614
Loan_Status          614
dtype: int64
```

[20]:
```
df.isnull()
```

[20]:
```
     Loan_ID  Dependents  Education  Self_Employed  ApplicantIncome  \
0      False       False      False          False            False   
1      False       False      False          False            False   
2      False       False      False          False            False   
3      False       False      False          False            False   
4      False       False      False          False            False   
..       …           …          …              …                …   
609    False       False      False          False            False   
610    False       False      False          False            False   
```

```
611    False      False      False          False              False
612    False      False      False          False              False
613    False      False      False          False              False

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
0                False        True             False           False
1                False       False             False           False
2                False       False             False           False
3                False       False             False           False
4                False       False             False           False
..                 …           …               …             …
609              False       False             False           False
610              False       False             False           False
611              False       False             False           False
612              False       False             False           False
613              False       False             False           False

     Property_Area  Loan_Status
0            False        False
1            False        False
2            False        False
3            False        False
4            False        False
..             …           …
609          False        False
610          False        False
611          False        False
612          False        False
613          False        False

[614 rows x 11 columns]
```

[21]:
```
missing_values=df.isnull()
```

[22]:
```
missing_values.dtypes
```

[22]:
```
Loan_ID              bool
Dependents           bool
Education            bool
Self_Employed        bool
ApplicantIncome      bool
CoapplicantIncome    bool
LoanAmount           bool
Loan_Amount_Term     bool
Credit_History       bool
Property_Area        bool
Loan_Status          bool
```

```
dtype: object
```

[23]:
```
no_missing_values=missing_values.sum()
```

[24]:
```
missing_values.sum()
```

[24]:
```
Loan_ID               0
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

[25]:
```
len(df)
```

[25]:
```
614
```

[26]:
```
no_missing_values/len(df)
```

[26]:
```
Loan_ID              0.000000
Dependents           0.024430
Education            0.000000
Self_Employed        0.052117
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.035831
Loan_Amount_Term     0.022801
Credit_History       0.081433
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

[27]:
```
no_missing_values/len(df)*100
```

[27]:
```
Loan_ID              0.000000
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
```

```
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

[28]:
```
df.isnull().mean().round(4) * 100
```

[28]:
```
Loan_ID              0.00
Dependents           2.44
Education            0.00
Self_Employed        5.21
ApplicantIncome      0.00
CoapplicantIncome    0.00
LoanAmount           3.58
Loan_Amount_Term     2.28
Credit_History       8.14
Property_Area        0.00
Loan_Status          0.00
dtype: float64
```

[29]:
```
#https://towardsdatascience.com/
↪data-cleaning-in-python-the-ultimate-guide-2020-c63b88bf0a0d
```

[30]:
```
#https://medium.com/dunder-data/
↪finding-the-percentage-of-missing-values-in-a-pandas-dataframe-a04fa00f84ab
```

heatmap

November 1, 2024

[1]:
```python
#https://towardsdatascience.com/heatmap-basics-with-pythons-seaborn-fb92ea280a6c
#The idea is straightforward, replace numbers with colors.
#Now, this visualization style came a long way from simple color-coded
#tables, it became widely used with geospatial data,
#and its commonly applied for describing density or intensity of variables,
#visualize patterns, variance, and even anomalies.
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
```

[2]:
```python
# read file
df = pd.read_csv('Foreign_Exchange_Rates.csv')
#print(df)
df = pd.read_csv('Foreign_Exchange_Rates.csv',
                 usecols=[1,7], names=['DATE', 'CAD_USD'],
                 skiprows=1, index_col=0, parse_dates=[0])
df
```

[2]:
```
           CAD_USD
DATE
2000-01-03  1.4465
2000-01-04  1.4518
2000-01-05  1.4518
2000-01-06  1.4571
2000-01-07  1.4505
…             …
2019-12-25      ND
2019-12-26  1.3124
2019-12-27  1.3073
2019-12-30  1.3058
2019-12-31  1.2962

[5217 rows x 1 columns]
```

[3]:
```python
df['CAD_USD'] = pd.to_numeric(df.CAD_USD, errors='coerce')
df.dropna(inplace=True)
```

```
print(df)
```

```
            CAD_USD
DATE
2000-01-03   1.4465
2000-01-04   1.4518
2000-01-05   1.4518
2000-01-06   1.4571
2000-01-07   1.4505
…             …
2019-12-24   1.3160
2019-12-26   1.3124
2019-12-27   1.3073
2019-12-30   1.3058
2019-12-31   1.2962

[5019 rows x 1 columns]
```

[4]:
```
# create a copy of the dataframe, and add columns for month and year
df_m = df.copy()
df_m['month'] = [i.month for i in df_m.index]
df_m['year'] = [i.year for i in df_m.index]
# group by month and year, get the average
df_m = df_m.groupby(['month', 'year']).mean()
print(df_m)
```

```
             CAD_USD
month year
1     2000  1.448600
      2001  1.503200
      2002  1.599714
      2003  1.541448
      2004  1.295755
…               …
12    2015  1.371255
      2016  1.333919
      2017  1.276870
      2018  1.343611
      2019  1.316895

[240 rows x 1 columns]
```

[5]:
```
df_m = df_m.unstack(level=0)
print(df_m)
```

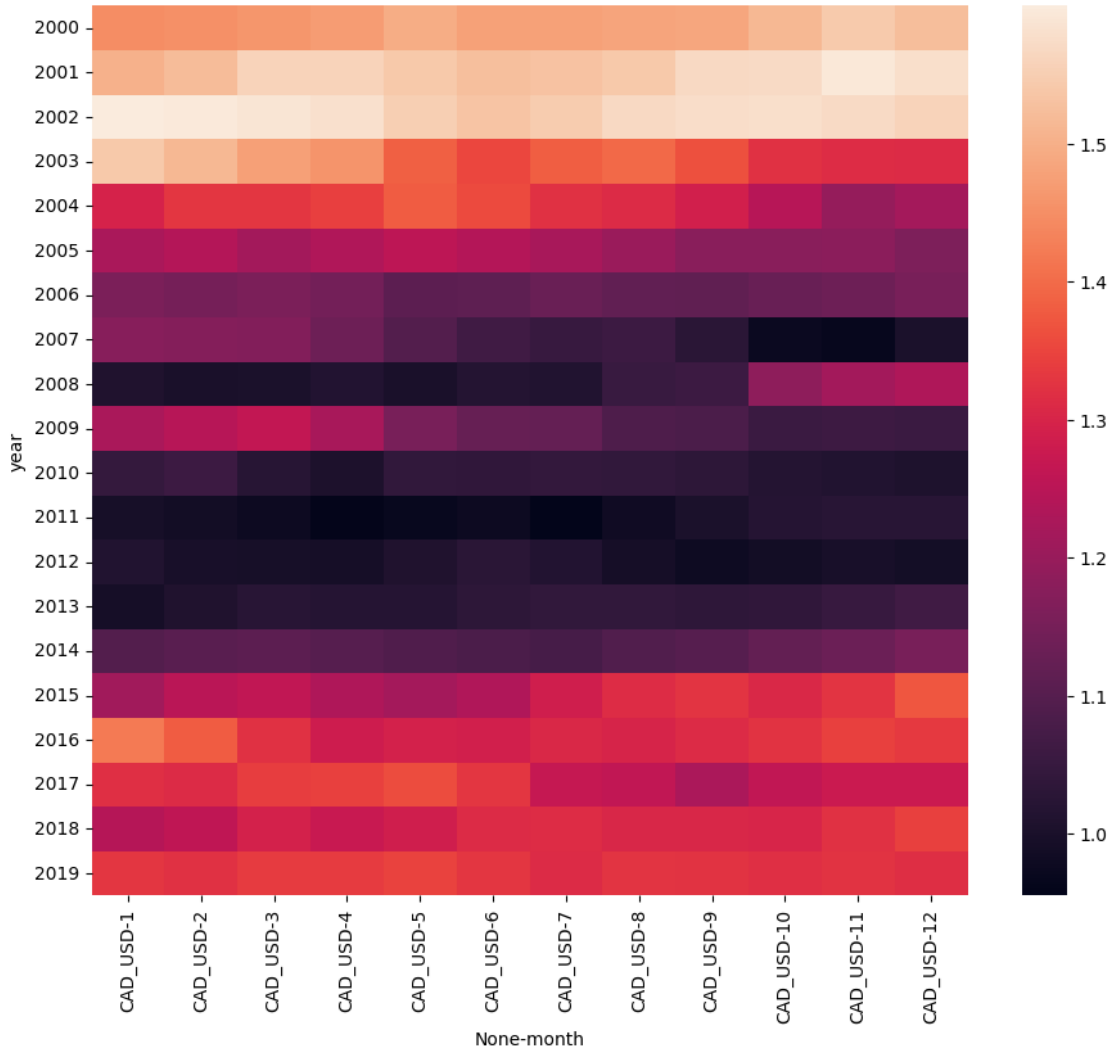
```
        CAD_USD                                                         \
month         1         2         3         4         5         6         7
year
```

2000	1.448600	1.451210	1.460774	1.468875	1.495736	1.477045	1.477785
2001	1.503200	1.521563	1.558741	1.557767	1.541050	1.524538	1.530790
2002	1.599714	1.596400	1.587743	1.581486	1.550155	1.531840	1.545550
2003	1.541448	1.512147	1.476081	1.458205	1.383957	1.352510	1.382091
2004	1.295755	1.329895	1.328578	1.341973	1.378860	1.357841	1.322505
2005	1.224835	1.240053	1.216026	1.235900	1.255529	1.240168	1.222855
2006	1.157165	1.148895	1.157309	1.144105	1.109991	1.113727	1.129445
2007	1.176262	1.170989	1.168159	1.134986	1.095086	1.065105	1.050186
2008	1.009943	0.998555	1.002943	1.013718	0.999305	1.016624	1.012964
2009	1.224820	1.245200	1.264518	1.224182	1.152785	1.126355	1.122861
2010	1.043811	1.057211	1.022900	1.005209	1.040280	1.037623	1.042229
2011	0.993945	0.987637	0.976561	0.957952	0.968043	0.976645	0.955315
2012	1.012985	0.996745	0.993773	0.992824	1.009732	1.028000	1.014200
2013	0.992057	1.009784	1.024424	1.018673	1.019559	1.031400	1.040214
2014	1.094010	1.105442	1.110681	1.099209	1.089386	1.083038	1.073918
2015	1.212190	1.249905	1.261832	1.233682	1.217640	1.236495	1.286314
2016	1.420811	1.379690	1.322639	1.281814	1.294529	1.289405	1.305235
2017	1.318305	1.310916	1.338700	1.343705	1.360573	1.329486	1.269040
2018	1.242905	1.258821	1.293255	1.273162	1.286627	1.312452	1.313343
2019	1.330045	1.320872	1.337052	1.337814	1.345977	1.328870	1.310523

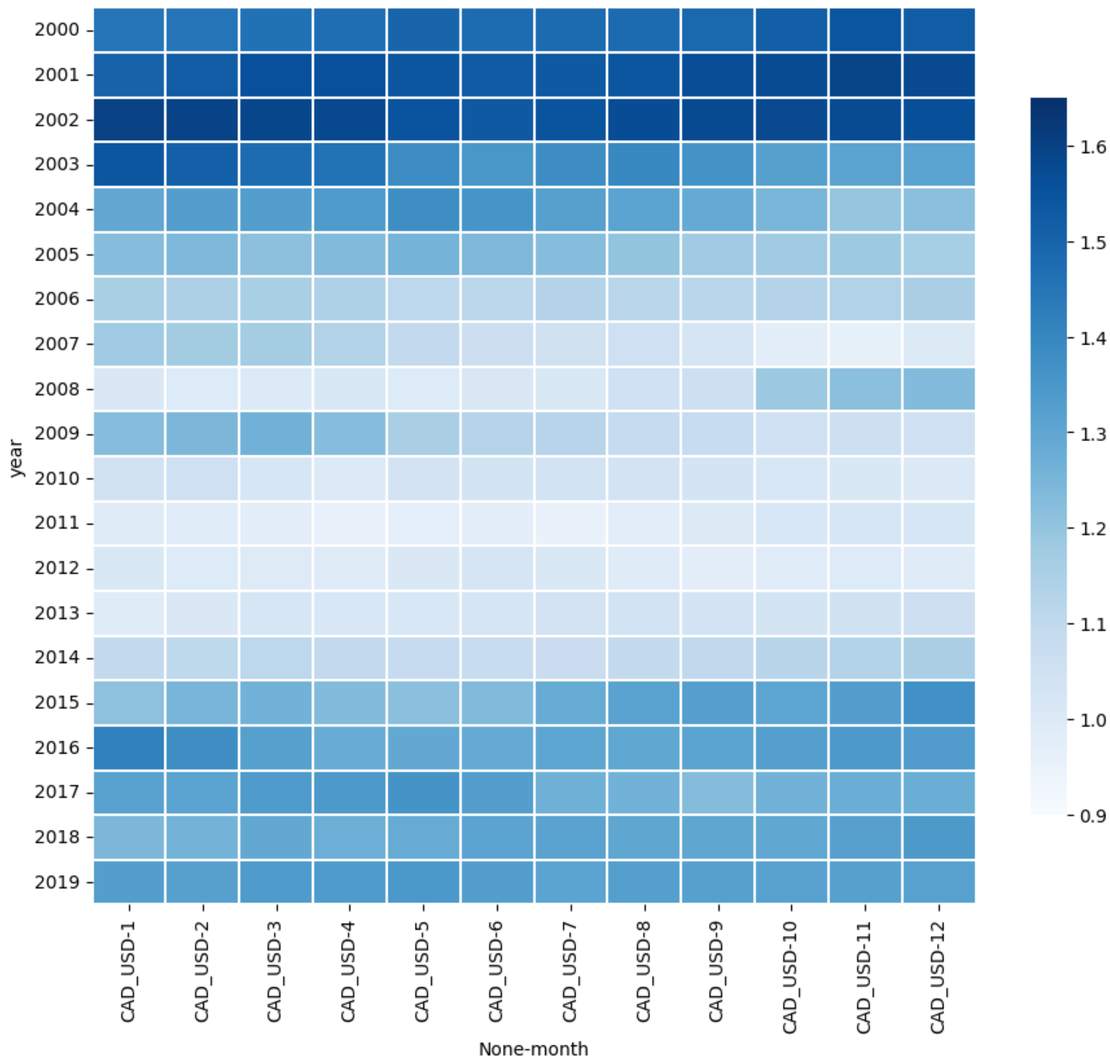
month	8	9	10	11	12
year					
2000	1.482813	1.486430	1.512476	1.542638	1.521875
2001	1.539857	1.567939	1.571677	1.592245	1.578755
2002	1.569418	1.576135	1.578009	1.571453	1.559219
2003	1.396271	1.363371	1.322095	1.313044	1.312755
2004	1.312677	1.288095	1.246935	1.196770	1.218883
2005	1.204283	1.177681	1.177415	1.181545	1.161481
2006	1.118213	1.116120	1.128538	1.135881	1.153235
2007	1.057852	1.026745	0.975413	0.967238	1.002070
2008	1.053457	1.058205	1.184695	1.217094	1.233695
2009	1.087238	1.081638	1.054676	1.059300	1.053691
2010	1.040395	1.032957	1.017900	1.012900	1.008062
2011	0.981709	1.002500	1.019800	1.024755	1.023524
2012	0.992383	0.978300	0.987155	0.996970	0.989820
2013	1.040718	1.034235	1.036282	1.048642	1.063919
2014	1.092633	1.101052	1.121155	1.132539	1.153162
2015	1.314724	1.326581	1.307224	1.327853	1.371255
2016	1.299783	1.310776	1.325095	1.343415	1.333919
2017	1.260770	1.227875	1.260690	1.277335	1.276870
2018	1.304248	1.303400	1.300441	1.320480	1.343611
2019	1.327314	1.324050	1.318923	1.323658	1.316895

[6]:
```
fig, ax = plt.subplots(figsize=(11, 9))
sb.heatmap(df_m)
```

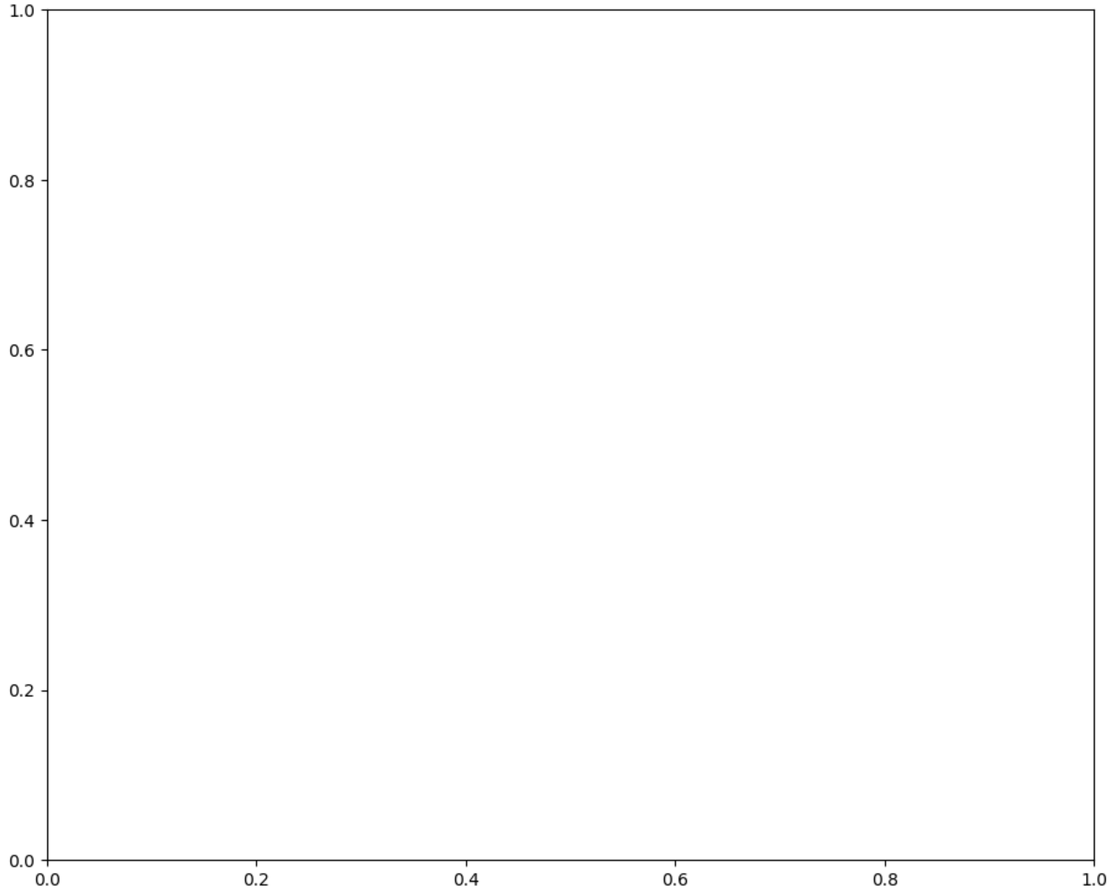
```
plt.show()
```



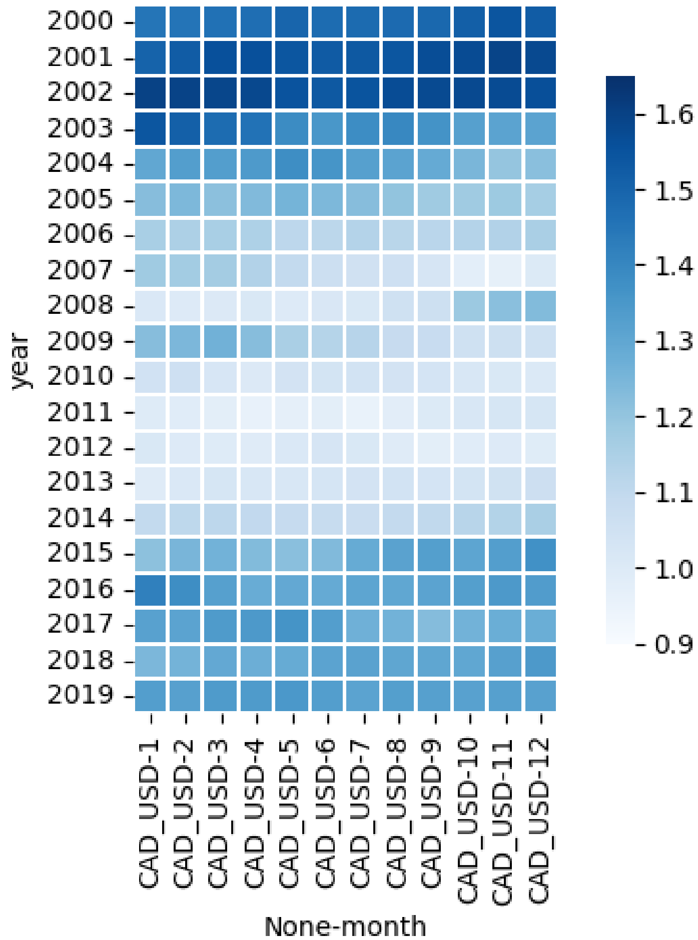
[7]:
```
fig, ax = plt.subplots(figsize=(11, 9))
# plot heatmap
sb.heatmap(df_m, cmap="Blues", vmin= 0.9, vmax=1.65,
           linewidth=0.3, cbar_kws={"shrink": .8})
plt.show()
```



[8]:
```
# figure
fig, ax = plt.subplots(figsize=(11, 9))
plt.show()
```

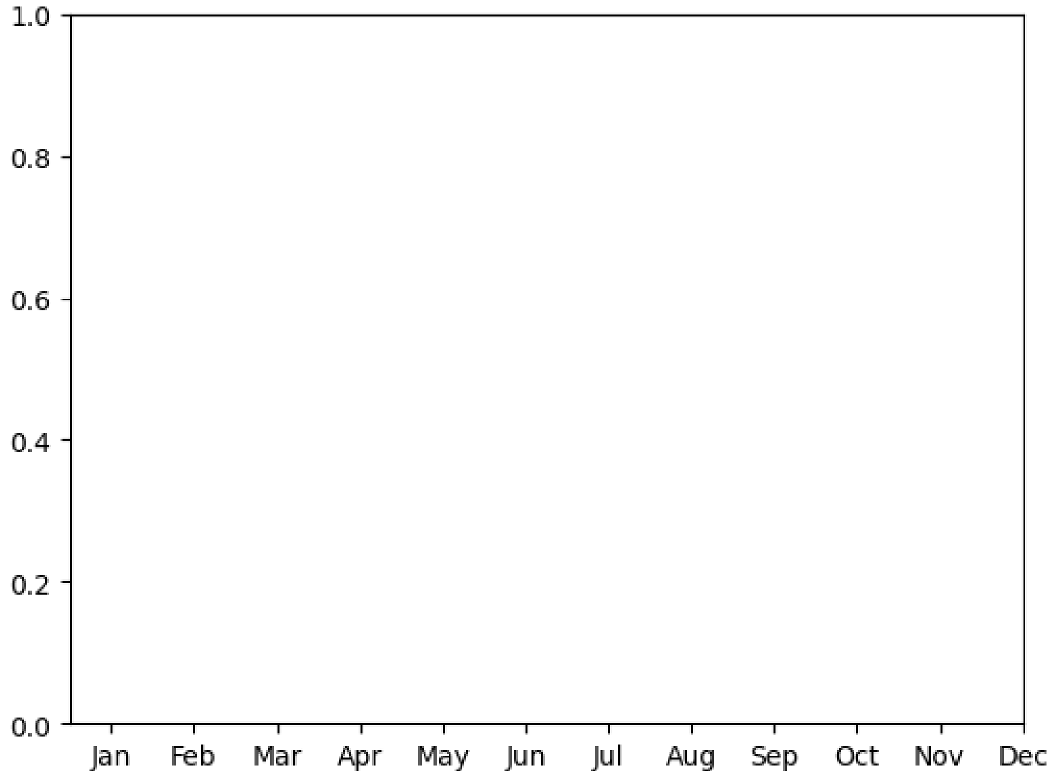


[9]:
```
# plot heatmap
sb.heatmap(df_m, cmap="Blues", vmin= 0.9, vmax=1.65, square=True,
           linewidth=0.3, cbar_kws={"shrink": .8})
plt.show()
```

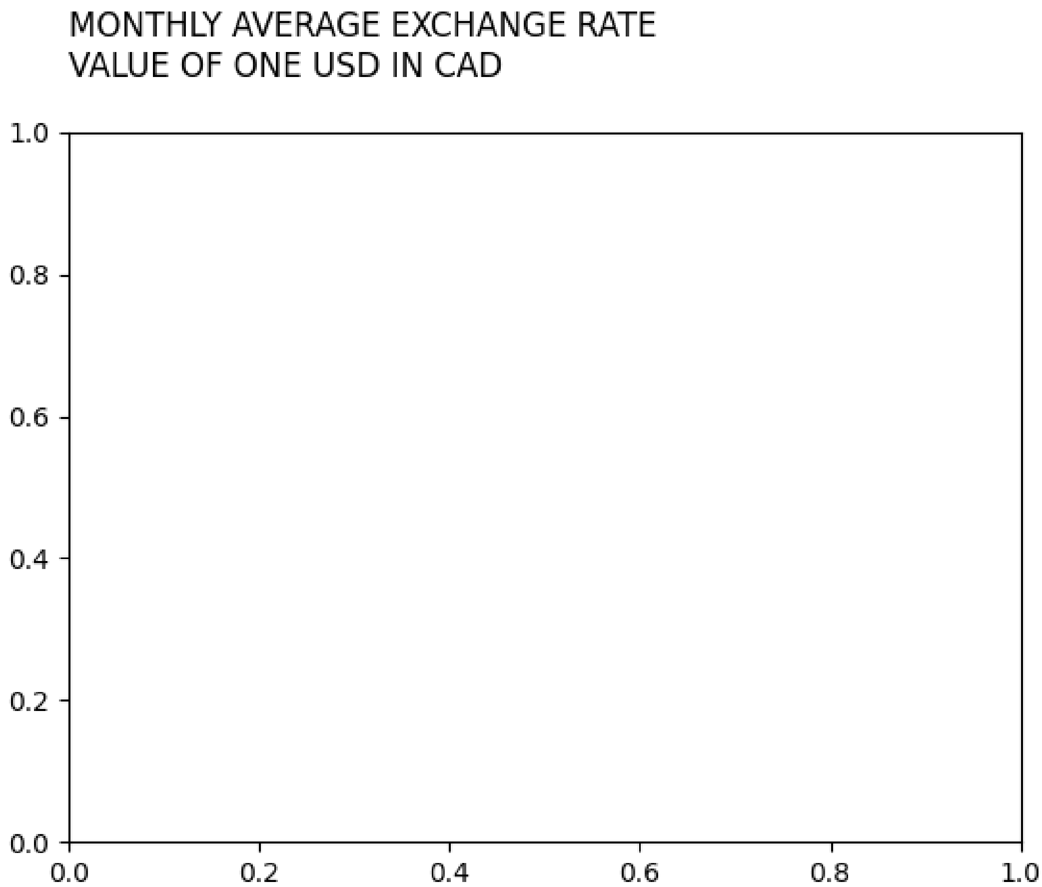


[ ]:

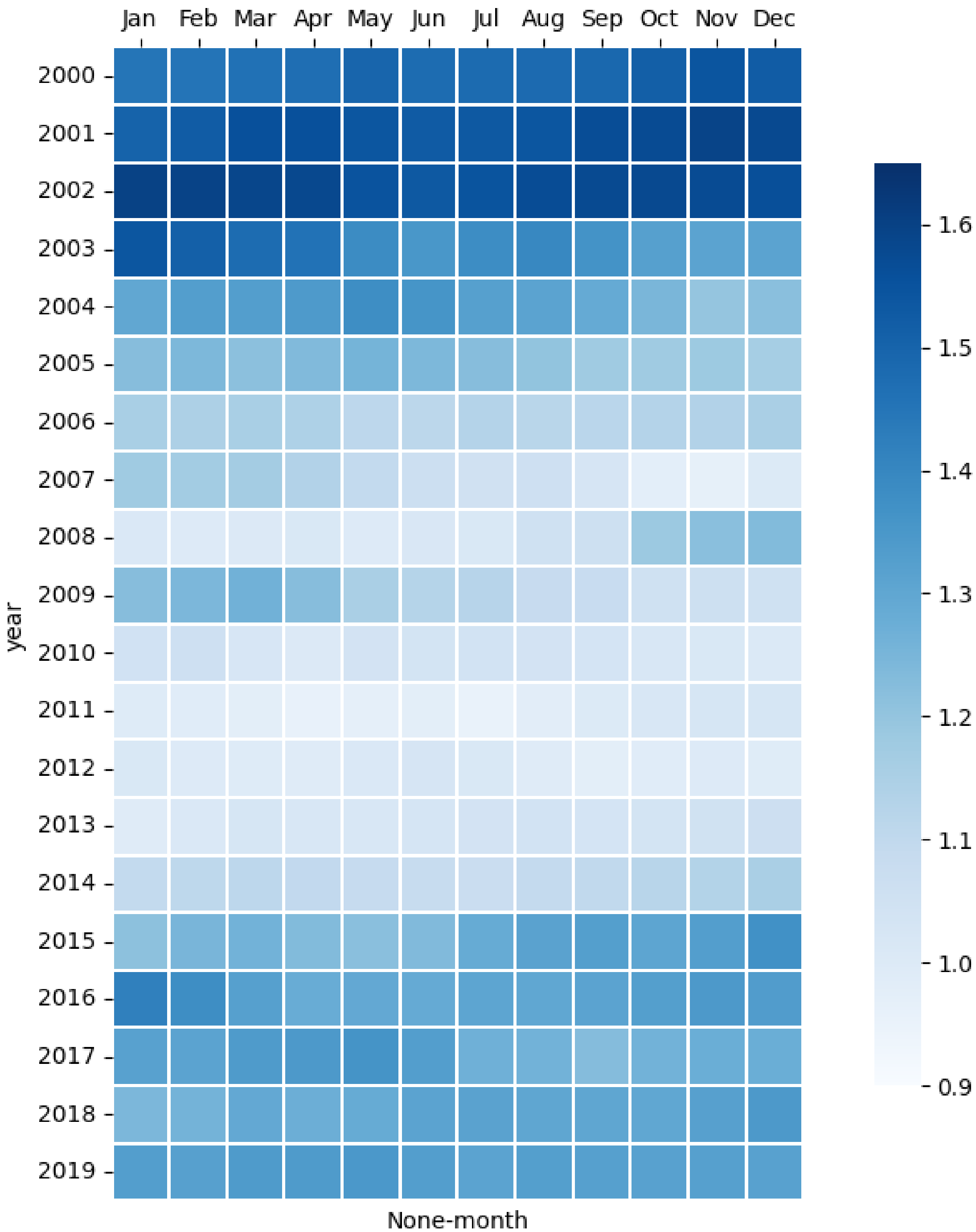
[10]:
```
ax.xaxis.tick_top()
labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
y = np.arange(0.5, 12)
plt.xticks(y, labels)     #optional to set the class names for the bars

plt.show()
```



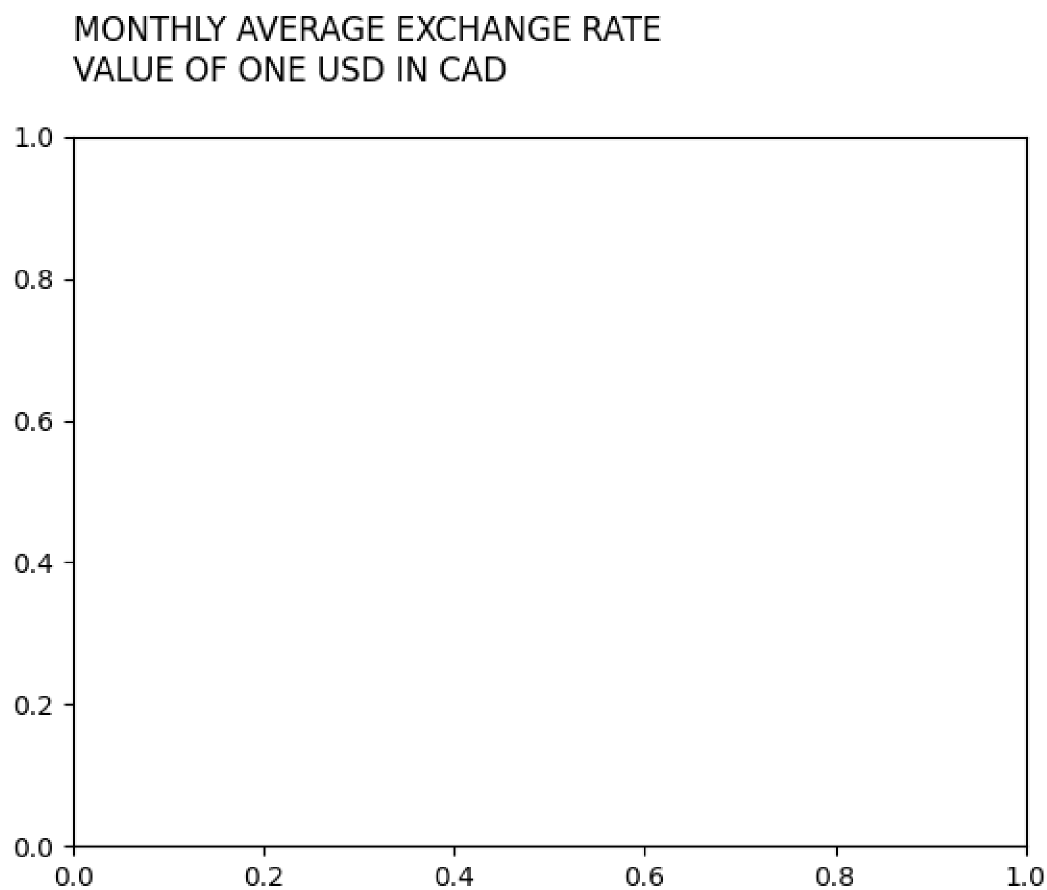
[11]:
```
# axis labels
plt.xlabel('')
plt.ylabel('')
# title
title = 'monthly Average exchange rate\nValue of one USD in CAD\n'.upper()
plt.title(title, loc='left')
plt.show()
```



[12]:
```
# figure
fig, ax = plt.subplots(figsize=(11, 9))
# plot heatmap
sb.heatmap(df_m, cmap="Blues", vmin= 0.9, vmax=1.65, square=True,
           linewidth=0.3, cbar_kws={"shrink": .8})
# xticks
ax.xaxis.tick_top()
labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
y = np.arange(0.5, 12)
plt.xticks(y, labels)    #optional to set the class names for the bars

plt.show()
# axis labels
plt.xlabel('')
plt.ylabel('')
# title
title = 'monthly Average exchange rate\nValue of one USD in CAD\n'.upper()
plt.title(title, loc='left')
```



[12]: Text(0.0, 1.0, 'MONTHLY AVERAGE EXCHANGE RATE\nVALUE OF ONE USD IN CAD\n')

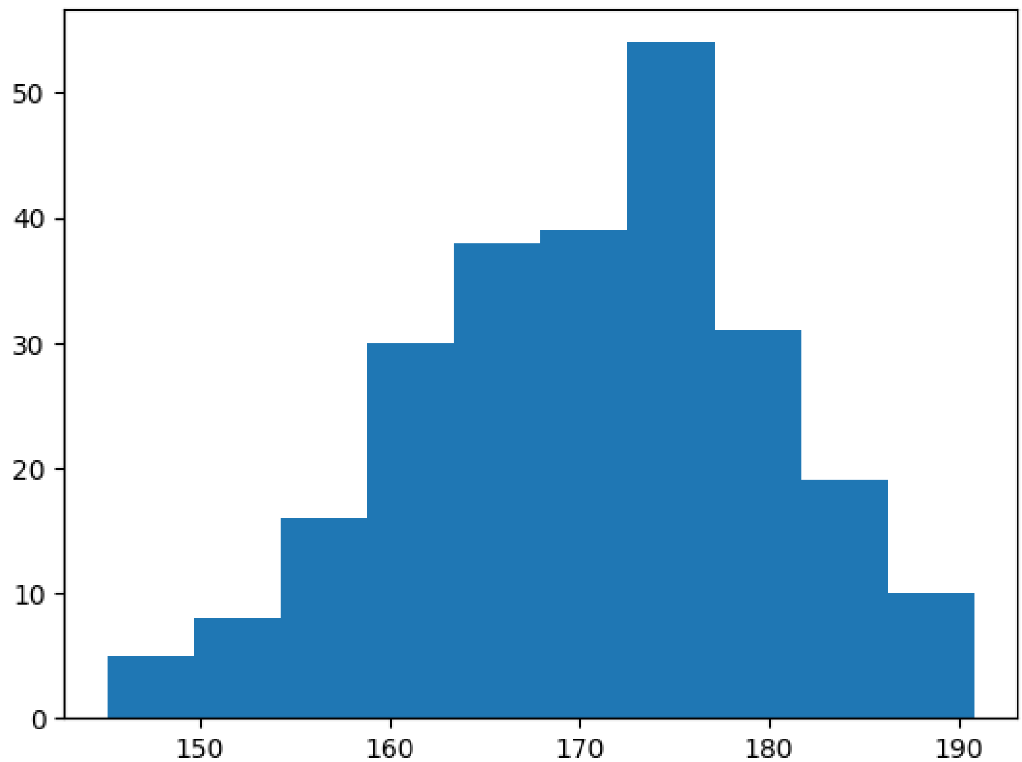

MONTHLY AVERAGE EXCHANGE RATE
VALUE OF ONE USD IN CAD
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
[ ]:
```

histogram

November 1, 2024

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```



[2]:
```
'''Syntax: matplotlib.pyplot.hist(x, bins=None, range=None, density=False,
                                weights=None, cumulative=False, bottom=None,␣
↪histtype='bar', align='mid',
                                orientation='vertical', rwidth=None, log=False,␣
↪color=None, label=None,
```

```
                                stacked=False, \*, data=None, \*\*)

Parameters: This method accept the following parameters that are described␣
 ↪below:

x : This parameter are the sequence of data.
bins : This parameter is an optional parameter and
    it contains the integer or sequence or string.

range : This parameter is an optional parameter and
    it the lower and upper range of the bins.

density : This parameter is an optional parameter and
    it contains the boolean values.

weights : This parameter is an optional parameter and
    it is an array of weights, of the same shape as x.

bottom : This parameter is the location of the bottom baseline
    of each bin.
histtype : This parameter is an optional parameter and
    it is used to draw type of histogram.
    {‘bar’, ‘barstacked’, ‘step’, ‘stepfilled’}

align : This parameter is an optional parameter and
    it controls how the histogram is plotted.
    {‘left’, ‘mid’, ‘right’}

rwidth : This parameter is an optional parameter and
    it is a relative width of the bars
    as a fraction of the bin width

log : This parameter is an optional parameter and
    it is used to set histogram axis to a log scale

color : This parameter is an optional parameter and
    it is a color spec or sequence of color specs,
    one per dataset.

label : This parameter is an optional parameter and
    it is a string, or sequence of strings
    to match multiple datasets.

normed : This parameter is an optional parameter and
    it contains the boolean values.
    It uses the density keyword argument instead.
```

```
    Returns:
        n :This returns the values of the histogram bins.
bins :This returns the edges of the bins.
patches :This returns the list of individual patches used to
    create the histogram.'''
```

[2]: "Syntax: matplotlib.pyplot.hist(x, bins=None, range=None, density=False,\n weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', \n orientation='vertical', rwidth=None, log=False, color=None, label=None,\n stacked=False, *, data=None, **)\n\nParameters: This method accept the following parameters that are described below:\n\nx : This parameter are the sequence of data.\nbins : This parameter is an optional parameter and \n   it contains the integer or sequence or string.\n   \nrange : This parameter is an optional parameter and\n   it the lower and upper range of the bins.\n \ndensity : This parameter is an optional parameter and\n   it contains the boolean values.\n   \nweights : This parameter is an optional parameter and \n it is an array of weights, of the same shape as x.\n   \nbottom : This parameter is the location of the bottom baseline\n   of each bin.\nhisttype : This parameter is an optional parameter and\n   it is used to draw type of histogram.\n   {'bar', 'barstacked', 'step', 'stepfilled'}\n   \nalign : This parameter is an optional parameter and\n   it controls how the histogram is plotted.\n   {'left', 'mid', 'right'}\n   \nrwidth : This parameter is an optional parameter and \n   it is a relative width of the bars \n   as a fraction of the bin width\n   \nlog : This parameter is an optional parameter and \n   it is used to set histogram axis to a log scale\n   \ncolor : This parameter is an optional parameter and \n   it is a color spec or sequence of color specs, \n   one per dataset.\n   \nlabel : This parameter is an optional parameter and \n   it is a string, or sequence of strings \n   to match multiple datasets.\n   \nnormed : This parameter is an optional parameter and \n   it contains the boolean values.\n   It uses the density keyword argument instead.\n   \n   Returns:\n   n :This returns the values of the histogram bins.\nbins :This returns the edges of the bins.\npatches :This returns the list of individual patches used to \n   create the histogram."

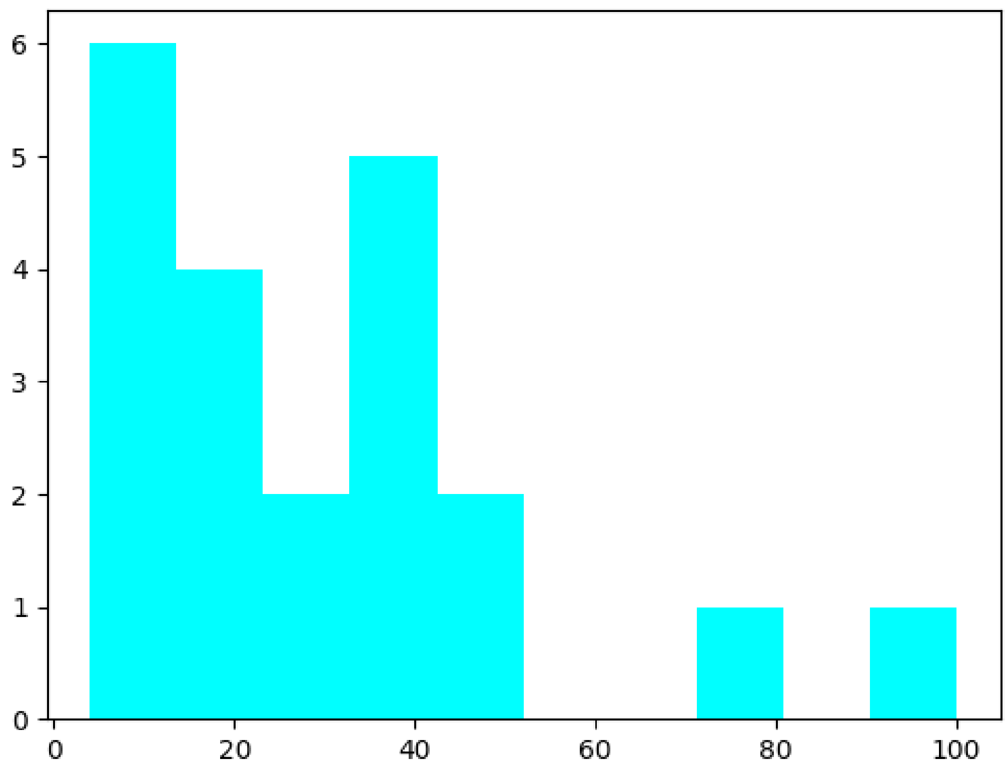
[3]:
```
x
```

[3]:
```
array([180.1891559 , 181.82130153, 151.0668477 , 164.74761425,
       156.2688702 , 163.6915134 , 160.66162479, 173.42438473,
       169.38124832, 171.94088107, 158.14620939, 151.00103365,
       174.88031851, 175.09007443, 173.59376329, 189.6826205 ,
       173.3506868 , 168.63890476, 169.11992538, 170.72441942,
       183.4699989 , 176.66731067, 177.77064196, 163.47498432,
       173.32572649, 177.93999991, 172.14267539, 161.93396696,
       179.69351339, 170.91869829, 157.46563784, 178.21022054,
       187.45887605, 164.98677176, 162.07027002, 149.47145897,
       161.13846585, 176.66130575, 151.14384026, 166.15445415,
       170.35016316, 155.43680541, 171.86872469, 161.09826464,
```

```
162.14347399, 168.42889575, 184.01099535, 165.95270288,
158.22502074, 178.77659096, 175.58163288, 165.5931718 ,
182.83269122, 168.753855  , 170.13104198, 174.49564919,
181.17215491, 155.99992172, 156.45939994, 178.00824447,
161.80007414, 158.13358705, 177.17690621, 162.25869367,
182.07035293, 154.01765828, 153.11680382, 166.41730348,
181.63473271, 179.03854895, 158.06988626, 173.39452932,
159.07434101, 174.21462882, 152.01916398, 176.23598915,
165.54675575, 161.44268824, 164.37923421, 171.9356047 ,
184.17090877, 176.30459248, 182.62864513, 169.37111736,
176.90419913, 161.02017308, 182.78360832, 172.98717014,
171.77450139, 146.65165918, 177.90293963, 169.43326009,
175.84744201, 188.88152738, 167.20759594, 167.10356235,
172.24937591, 172.71098658, 168.15534057, 174.10411343,
172.60458931, 174.85063902, 186.43178479, 160.70577265,
173.70747588, 163.08938047, 176.37712032, 161.65851484,
179.49879928, 172.3821393 , 164.20456516, 160.78842538,
183.95665191, 172.77981707, 163.64172854, 176.1890116 ,
162.3232011 , 163.78928812, 174.22252627, 167.50461407,
166.6792524 , 171.97320553, 177.00088154, 182.71599223,
170.25969077, 155.50786061, 187.31405951, 170.17182328,
172.59460215, 160.73071251, 149.62250015, 167.15264382,
164.45721781, 171.14116328, 183.00837686, 174.85516674,
190.83714678, 167.44552831, 185.82337777, 172.83572763,
172.94753023, 182.71833908, 179.59932041, 174.21053205,
172.63577566, 187.59324866, 183.79744561, 175.95671517,
172.00800751, 156.2733159 , 178.92254929, 165.34046579,
187.2098573 , 176.59351614, 154.97658562, 181.14221905,
182.52125413, 163.62244694, 172.61099678, 182.18928159,
162.33101453, 163.10074031, 152.53478013, 155.61285449,
169.36077576, 176.04330126, 181.20616   , 172.2412444 ,
170.33324403, 164.68528635, 168.96827276, 160.52692431,
162.63623663, 169.4550788 , 176.25737146, 179.00431882,
175.88978438, 161.89308904, 167.54408039, 175.94683532,
167.61426588, 162.1327988 , 183.51838304, 177.555114  ,
167.54018547, 169.92044994, 171.46557142, 169.09062221,
177.91512621, 164.59787423, 178.32761306, 171.88130167,
166.84220133, 159.56706329, 159.94401771, 170.60770969,
178.02760858, 154.13383453, 180.27145769, 168.07217441,
177.21222836, 173.57034618, 174.54837206, 178.09082136,
163.93240385, 175.34037568, 155.69816066, 174.55757414,
159.76557563, 159.73965996, 176.45165302, 165.62634865,
172.37069318, 164.59499487, 172.33941233, 172.70385329,
146.99029766, 162.52164597, 145.13208403, 187.33156942,
180.35543922, 178.11564027, 166.44322925, 166.31082958,
179.34398995, 175.10394809, 160.57386495, 182.13447835,
173.45220398, 167.09081408, 165.19759455, 155.78112776,
```

```
157.49635031, 172.89709373, 182.9511535 , 178.74028621,
168.08279516, 167.66981233, 174.9560128 , 177.20456812,
170.3215117 , 162.38248385, 165.81939794, 176.63727045,
181.042489  , 188.11357173, 174.84603481, 175.54192573,
166.32833033, 173.74250277])
```

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]
n_bins = 10     #no of bins
#patches is the specifics of histogram diagram measurements
bin_heights, bins, patches = plt.hist(x,bins=10, facecolor='cyan')
plt.show()
```



[5]:
```
bin_heights
```

[5]:
```
array([6., 4., 2., 5., 2., 0., 0., 1., 0., 1.])
```

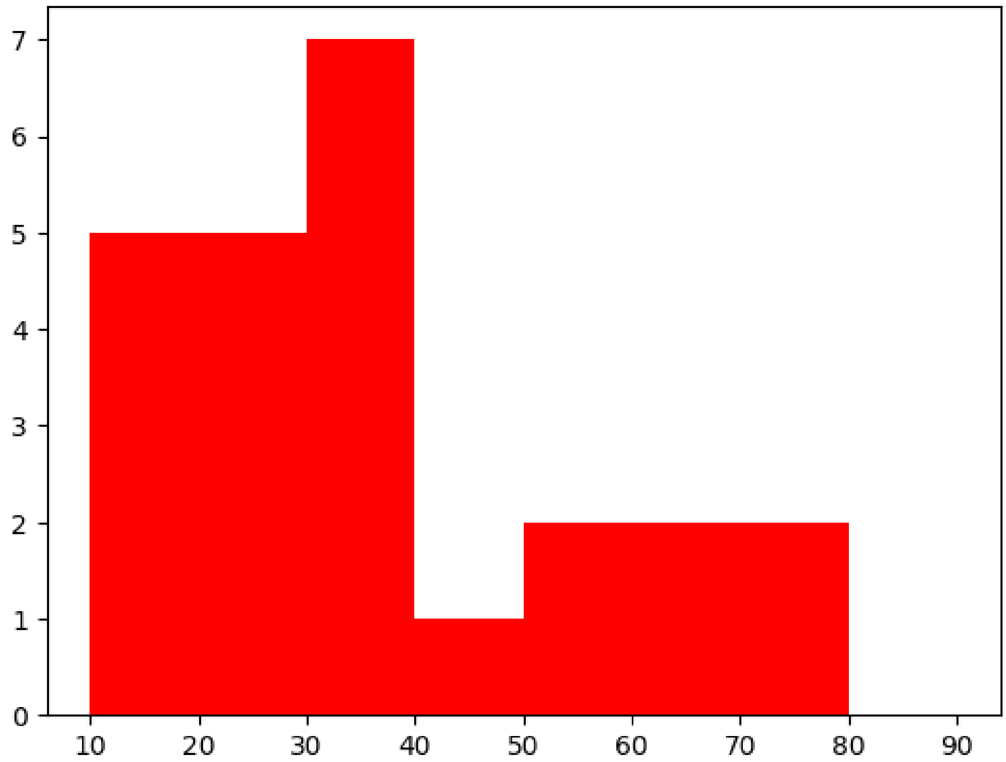
[6]:
```
bins
```

[6]:
```
array([  4. ,  13.6,  23.2,  32.8,  42.4,  52. ,  61.6,  71.2,  80.8,
        90.4, 100. ])
```

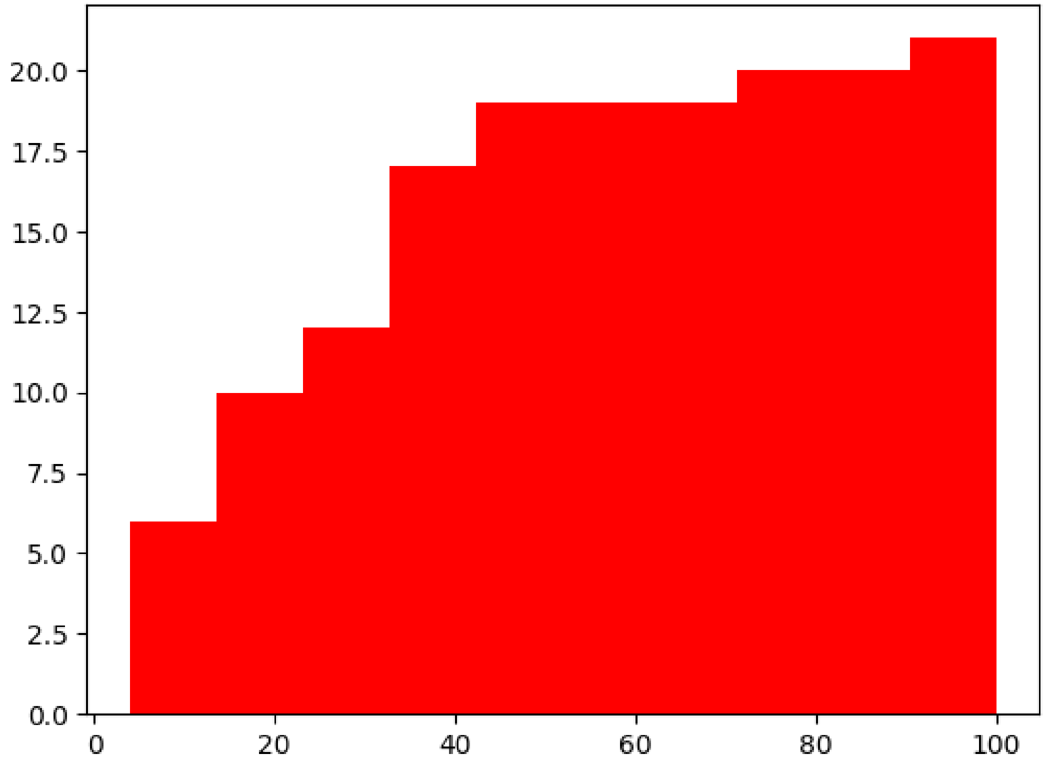
[7]:
```
print(patches[0])
```

```
Rectangle(xy=(4, 0), width=9.6, height=6, angle=0)
```

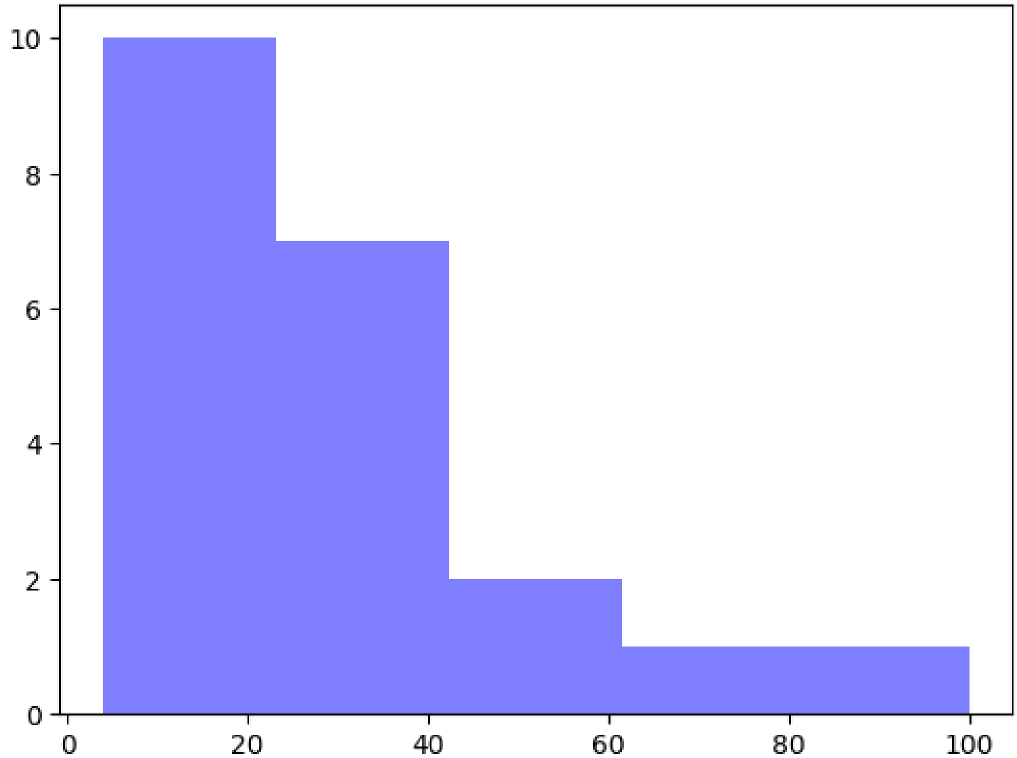
[8]:
```
#to create unequally sized bins

n_bins = [10, 30, 40, 50,80,90]      #bin values
bin_heights, bins, patches = plt.hist(x, n_bins, facecolor='red')
plt.show()
```



[9]:
```
#cumulative histogram -> sets the bin height as, plotted_bin(n) = actual_bin(n)
↪+ plotted_bin(n-1) for all bins
plt.hist(x, cumulative=True, facecolor='red')
plt.show()
```

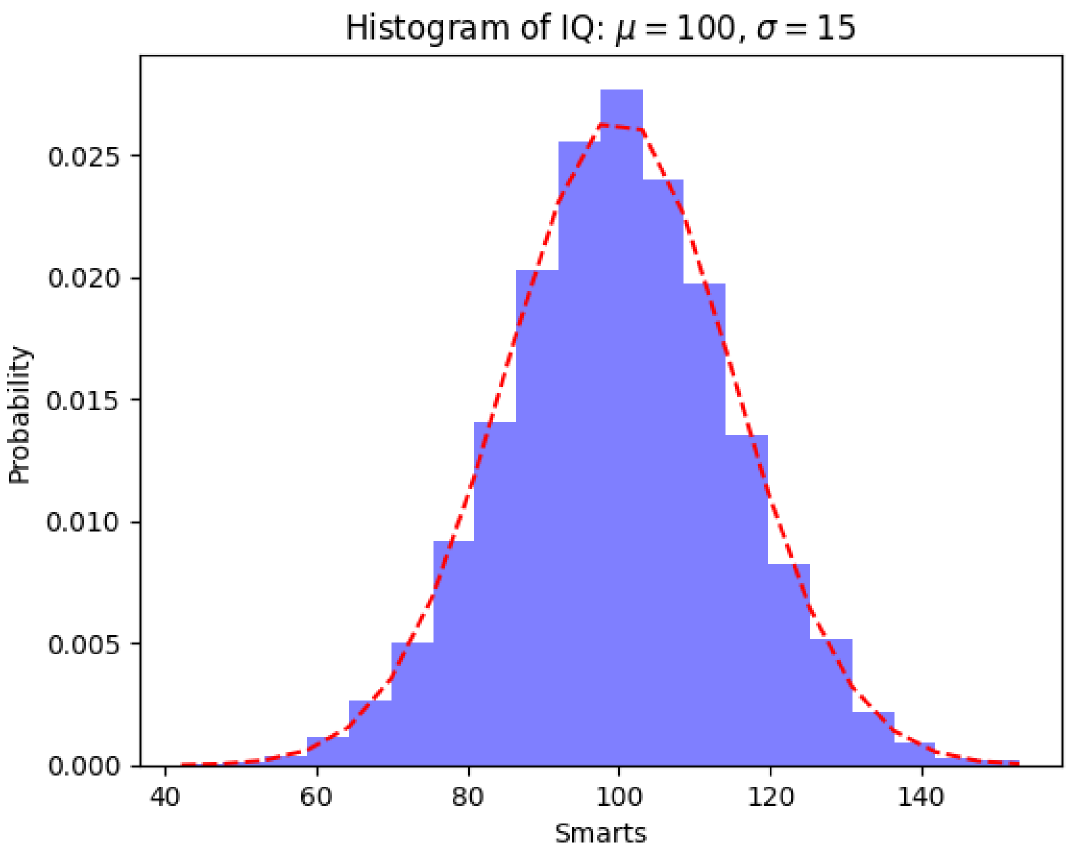


[10]:
```
import numpy as np

import matplotlib.pyplot as plt

x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,35,36,37,18,49,50,100]
num_bins = 5
#n, bins, patches = plt.hist(x, num_bins, facecolor='blue', alpha=0.5)
plt.hist(x, num_bins, facecolor='blue', alpha=0.5)
plt.show()
```



[11]:
```
#!/usr/bin/env python

import numpy as np
#import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
from scipy.stats import norm
# example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(10000)

num_bins = 20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, density=1, facecolor='blue', alpha=0.5)

# add a 'best fit' line
y = norm.pdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')
```

```
# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```



[12]:
```
n
```

[12]:
```
array([1.80419699e-05, 1.44335759e-04, 3.60839398e-04, 1.13664410e-03,
       2.63412761e-03, 4.97958370e-03, 9.20140466e-03, 1.40907785e-02,
       2.02611322e-02, 2.55293874e-02, 2.77124658e-02, 2.39958200e-02,
       1.97018311e-02, 1.35495194e-02, 8.28126419e-03, 5.19608733e-03,
       2.20112033e-03, 9.56224405e-04, 2.88671519e-04, 1.80419699e-04])
```

[13]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame(
    np.random.randint(1, 7, 6000),
    columns = ['one'])
```

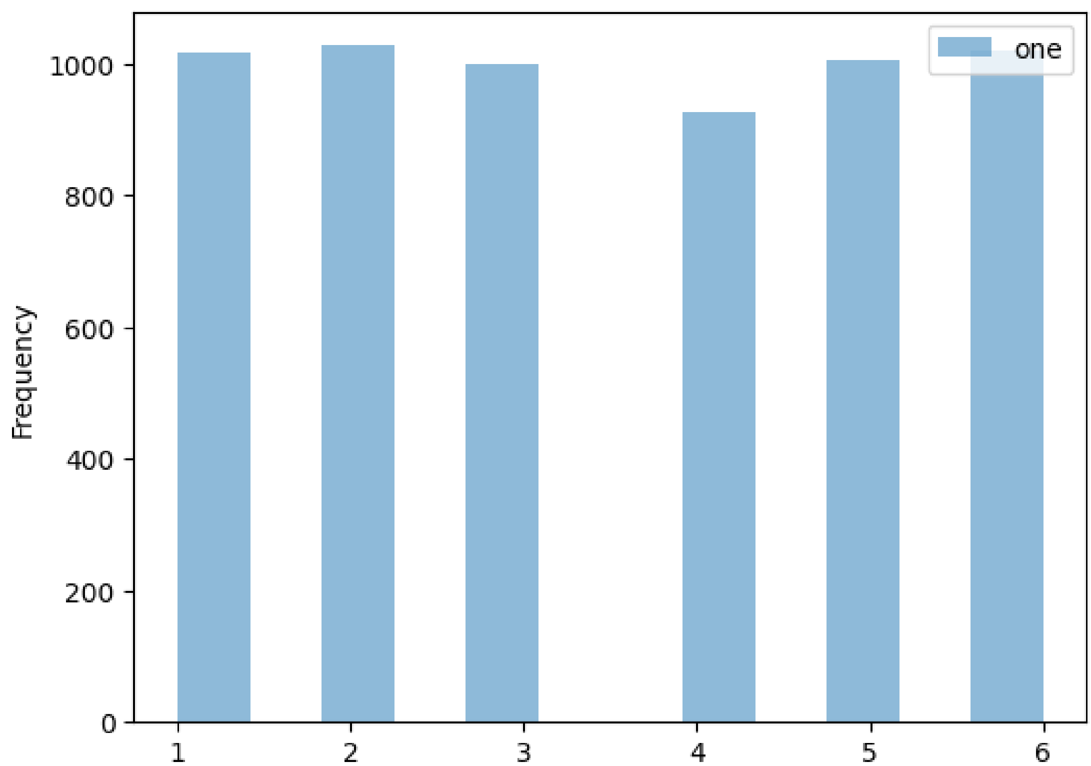
[14]:
```
df
```

[14]:

	one
0	1
1	6
2	2
3	1
4	6
…	…
5995	6
5996	2
5997	1
5998	6
5999	2

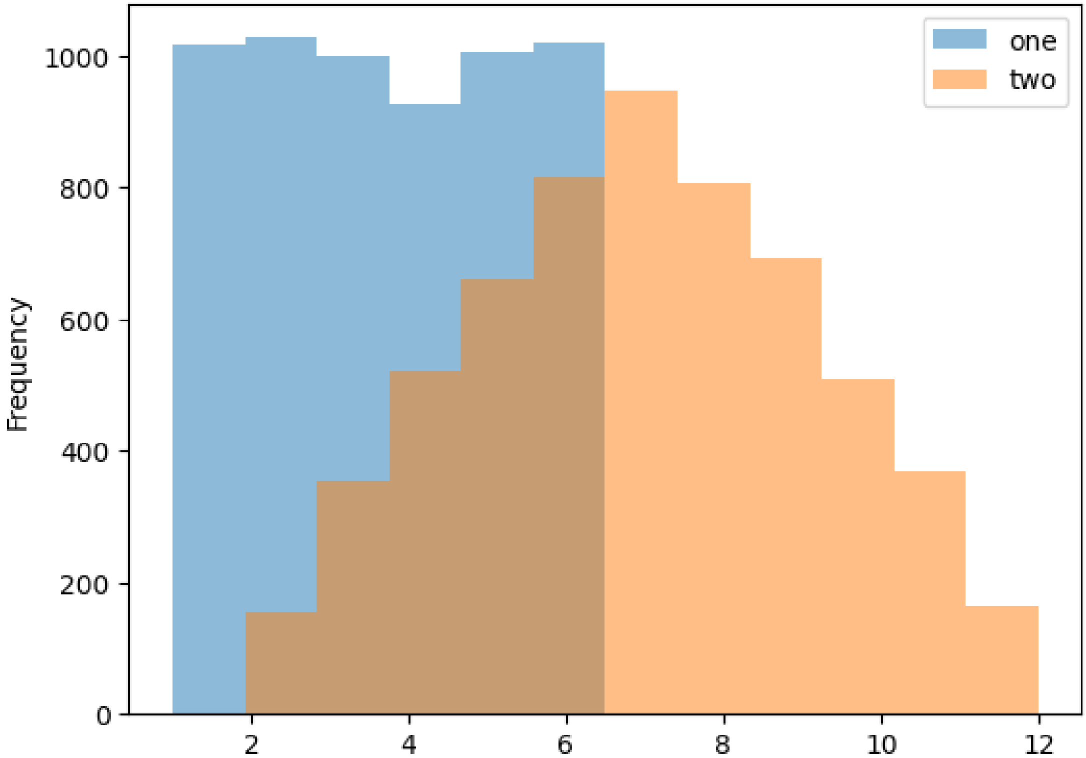
[6000 rows x 1 columns]

[15]:
```
%matplotlib inline
ax = df.plot.hist(bins=12, alpha=0.5)
```



[16]:
```
df['two'] = df['one'] + np.random.randint(1, 7, 6000)
ax = df.plot.hist(bins=12, alpha=0.5)
9
```

[16]: 9



[ ]:

hypothesis-test

November 1, 2024

[1]:
```
from scipy.stats import norm # Import the normal distribution functions from␣
↪SciPy
from math import sqrt # Import the square root function from the math module
def two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    # Calculate the critical z-value for a two-tailed test
    actual_z = abs(norm.ppf(alpha / 2)) # Get the z-value corresponding to␣
↪alpha/2
    # Calculate the z-value for the hypothesis test
    hypo_z = (sample_mean - pop_mean) / (std_dev / sqrt(sample_size)) # Z-test␣
↪statistic
    print('actual z value :', actual_z) # Print the actual critical z-value
    print('hypothesis z value :', hypo_z, '\n') # Print the calculated z-value␣
↪for the hypothesis
    # Check if the calculated z-value falls into the rejection region
    if hypo_z >= actual_z or hypo_z <= -actual_z:
        return True # Reject the null hypothesis
    else:
        return False # Fail to reject the null hypothesis
# Define parameters for the hypothesis test
alpha = 0.05 # Significance level
sample_mean = 585 # Mean of the sample
pop_mean = 558 # Population mean under the null hypothesis
sample_size = 100 # Sample size
std_dev = 139 # Standard deviation of the population
# Print hypotheses
print('H0 :  =', pop_mean) # Null hypothesis: population mean is equal to␣
↪pop_mean
print('H1 :  !=', pop_mean) # Alternative hypothesis: population mean is not␣
↪equal to pop_mean
print('alpha value is :', alpha, '\n') # Print the significance level
# Perform the two-sided hypothesis test
reject = two_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject:
    print('Reject NULL hypothesis') # If the test result indicates rejection
else:
    print('Failed to reject NULL hypothesis') # If the test result does not␣
↪indicate rejection
```

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

actual z value : 1.959963984540054
hypothesis z value : 1.9424460431654675

Failed to reject NULL hypothesis
```

[2]:

```python
# One-sided hypothesis test (for greater than in the null hypothesis)
def one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha):
    # Calculate the critical z-value for a one-tailed test
    actual_z = abs(norm.ppf(alpha)) # Get the z-value corresponding to alpha
    # Calculate the z-value for the hypothesis test
    hypo_z = (sample_mean - pop_mean) / (std_dev / sqrt(sample_size)) # Z-test␣
↪statistic
    print('actual z value :', actual_z) # Print the actual critical z-value
    print('hypothesis z value :', hypo_z, '\n') # Print the calculated z-value␣
↪for the hypothesis
    # Check if the calculated z-value falls into the rejection region
    if hypo_z >= actual_z:
        return True # Reject the null hypothesis
    else:
        return False # Fail to reject the null hypothesis
# Define parameters for the one-sided hypothesis test
alpha = 0.05 # Significance level
sample_mean = 108 # Mean of the sample
pop_mean = 100 # Population mean under the null hypothesis
sample_size = 36 # Sample size
std_dev = 15 # Standard deviation of the population
# Print hypotheses
print('H0 :   <=', pop_mean) # Null hypothesis: population mean is less than or␣
↪equal to pop_mean
print('H1 :   >', pop_mean) # Alternative hypothesis: population mean is␣
↪greater than pop_mean
print('alpha value is :', alpha, '\n') # Print the significance level
# Perform the one-sided hypothesis test
reject = one_sided_hypo(sample_mean, pop_mean, std_dev, sample_size, alpha)
if reject:
    print('Reject NULL hypothesis') # If the test result indicates rejection
else:
    print('Failed to reject NULL hypothesis') # If the test result does not␣
↪indicate rejection
```

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

```
actual z value : 1.6448536269514729
hypothesis z value : 3.2

Reject NULL hypothesis
```

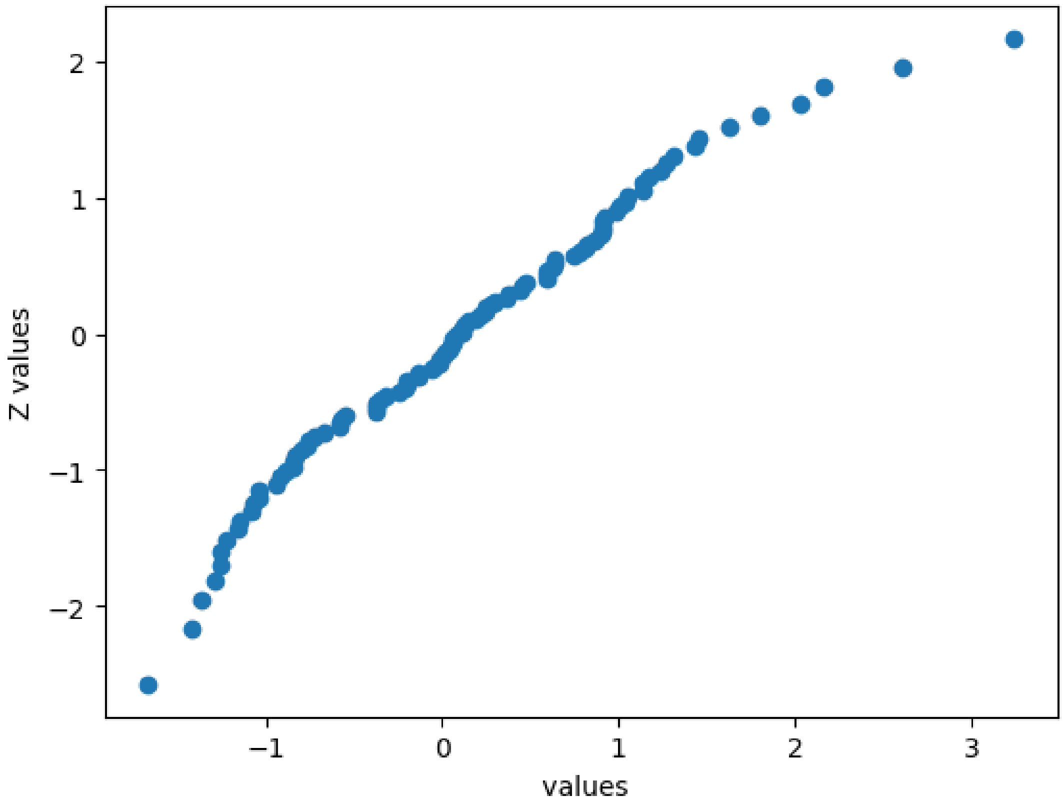
Prateek P PES1UG23AM211

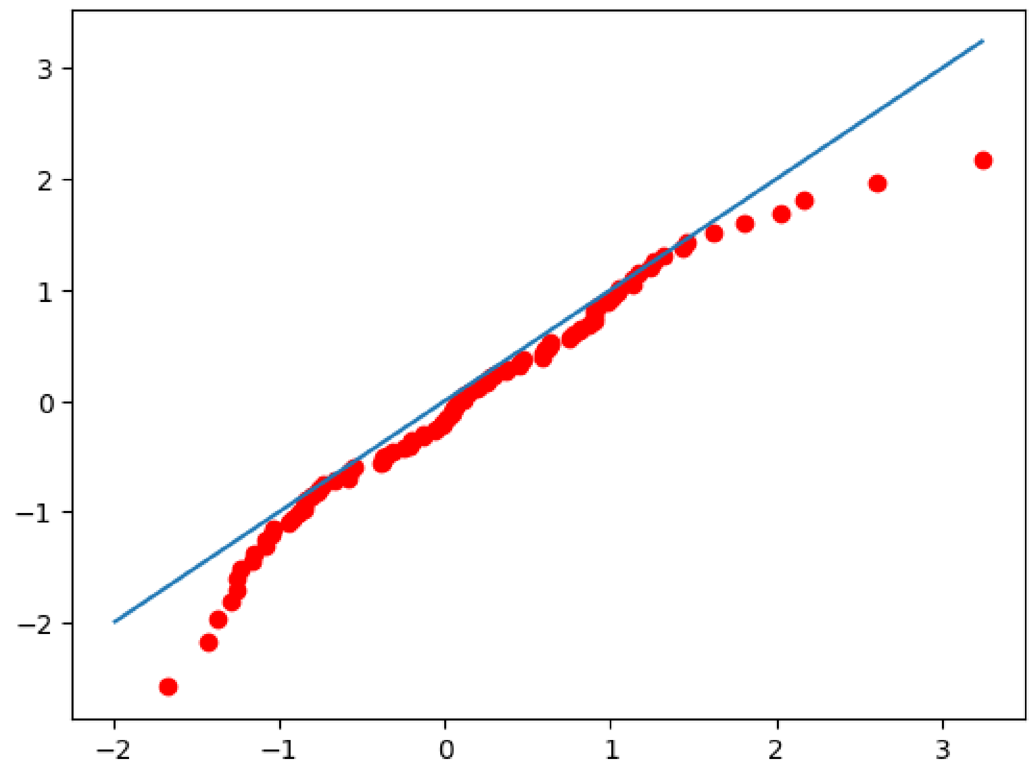
normal-prob-plot

November 1, 2024

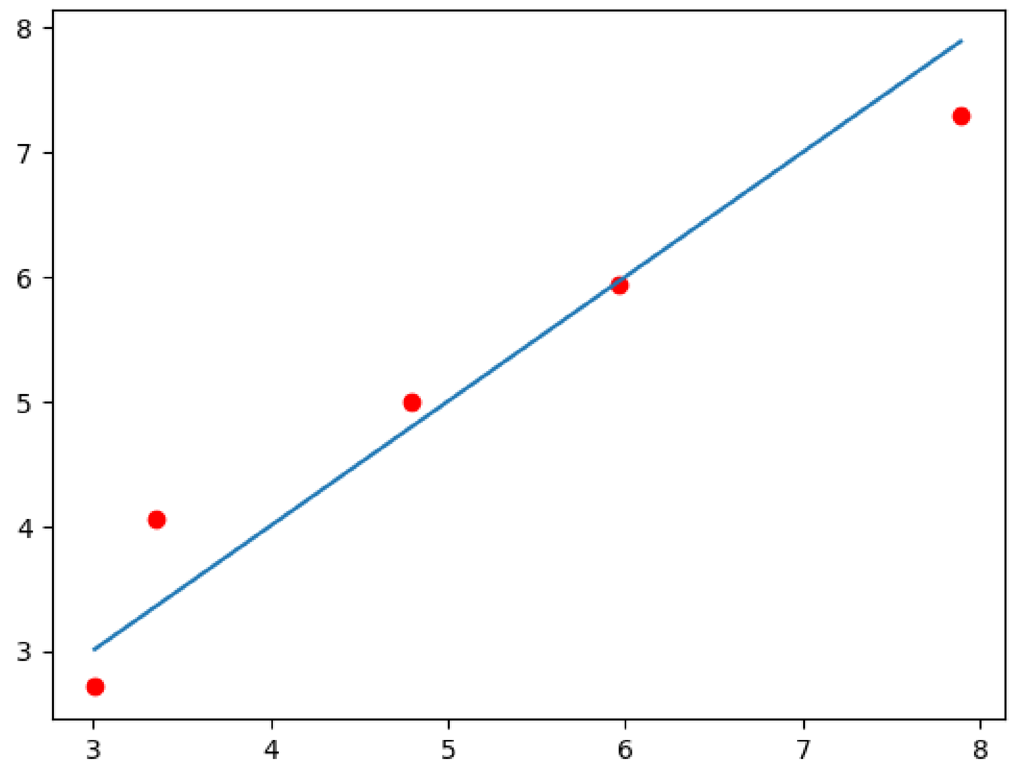
[1]:

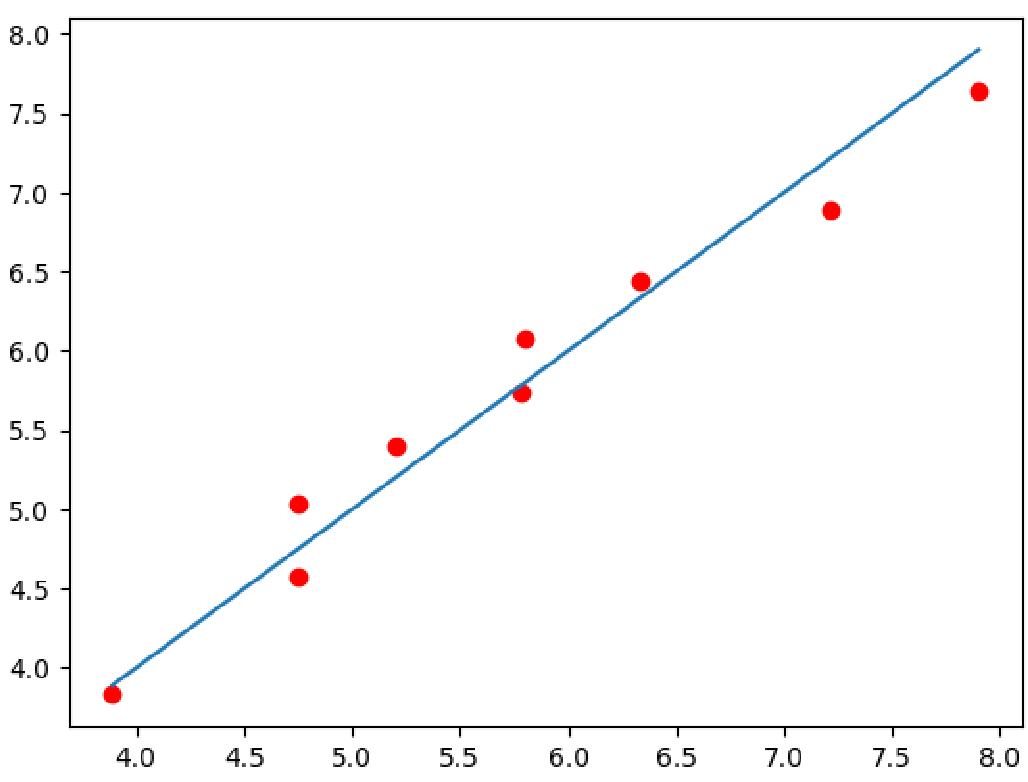
```
from scipy.stats import zscore
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
#its just an example npp of original values v/s theortical values(z scores)
def npp(data):
    data = sorted(data)
    p = [(data.index(i)-0.5)/len(data) for i in data]
    z = zscore(p)

    # t = [norm.ppf(i, np.mean(data), np.std(data)) for i in p]
    #xi=[np.std(data)*zi+np.mean(data) for zi in z]

    xi=norm.ppf(p)

    plt.scatter(data, xi)
    plt.ylabel('Z values')
    plt.xlabel(' values')
    plt.show()
    plt.plot(data,  xi,'ro',data,  data)
    plt.show()

#n datapoints
n = 100
data = np.random.randn(n)
npp(data)
```

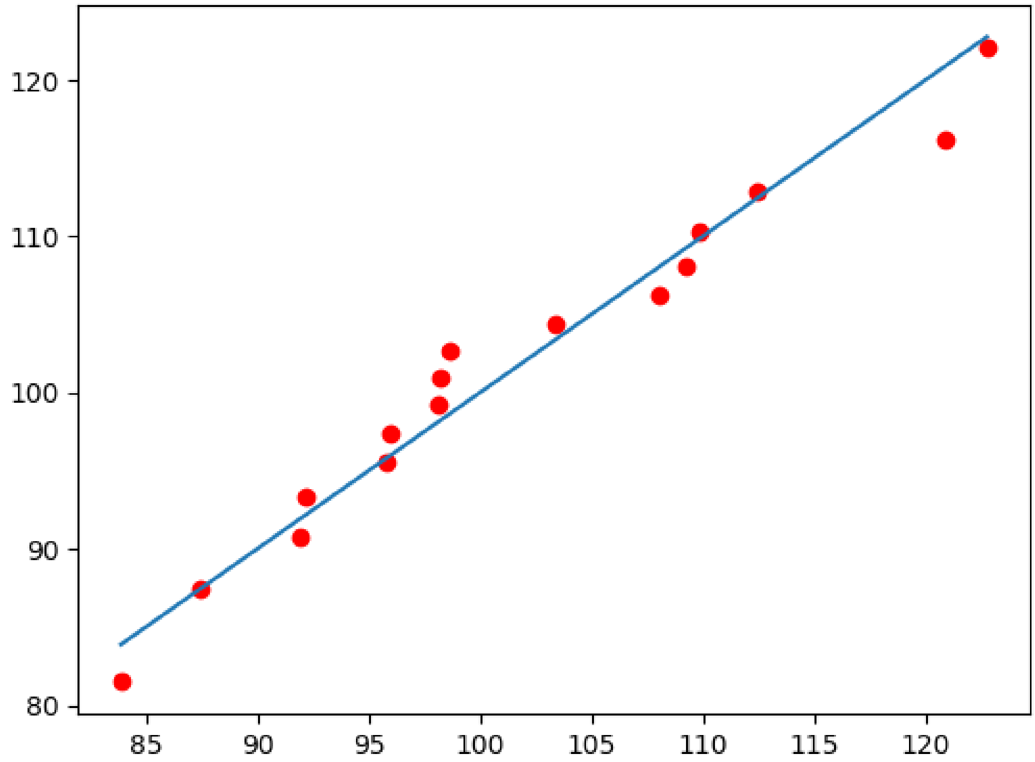

Z values
values
2
1
0
−1
−2
−1
0
1
2
3



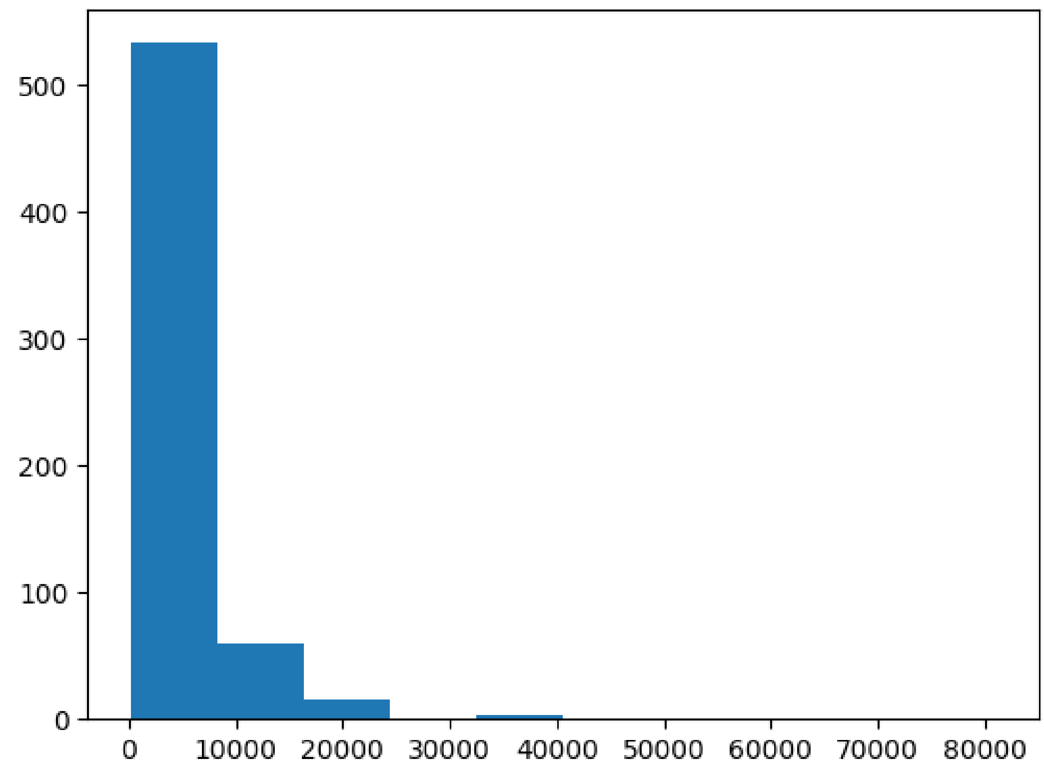
[2]:
```
X1 = [3.01, 3.35, 4.79, 5.96, 7.89]

X2 = [3.89, 4.75, 4.75, 5.20, 5.78, 5.80, 6.33, 7.21, 7.90]

X3 = [108.047, 109.249, 103.385, 112.454, 95.780,  122.734, 109.842 , 120.858,
      98.604, 98.122,  95.971, 98.173,   87.437 , 91.884, 92.193,  83.882]
def npp1(data):
    p =[]
    t =[]
    data = np.sort(np.array(data))
    p = [(i - 0.5)/len(data) for i in range(1, len(data)+1)]
    t = [norm.ppf(i, np.mean(data), np.std(data)) for i in p]
    plt.plot(data, t, 'ro', data, data)
    plt.show()

npp1(X1)
npp1(X2)
npp1(X3)
```


8
7
6
5
4
3
3
4
5
6
7
8

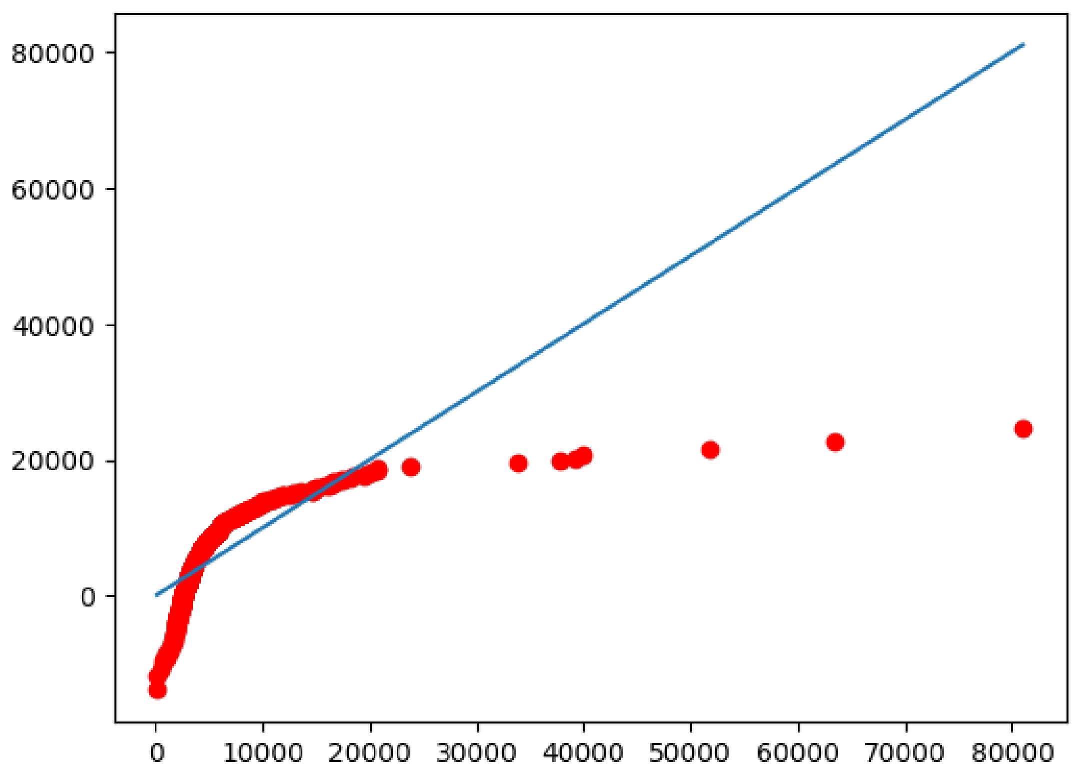

8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
4.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



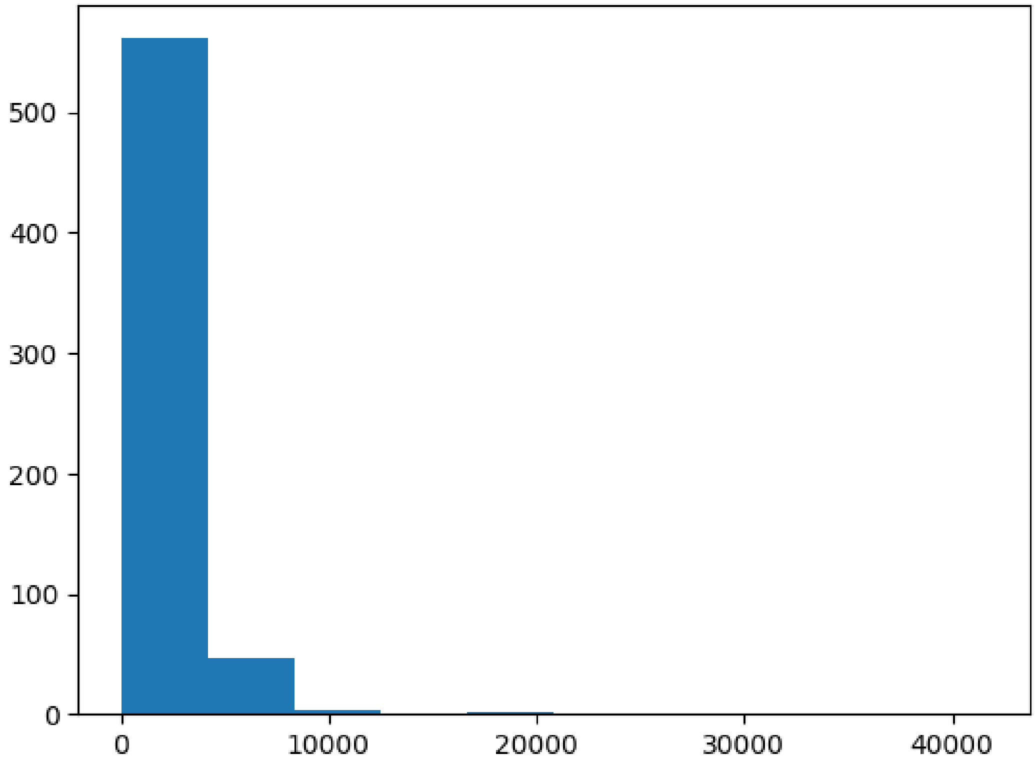
[3]:
```
import pandas as pd

df = pd.read_csv('train.csv')
plt.hist(df.ApplicantIncome)
plt.show()
npp1(df.ApplicantIncome)
df1 = pd.read_csv('train.csv')
plt.hist(df1.CoapplicantIncome)
plt.show()
npp1(df1.CoapplicantIncome)
```



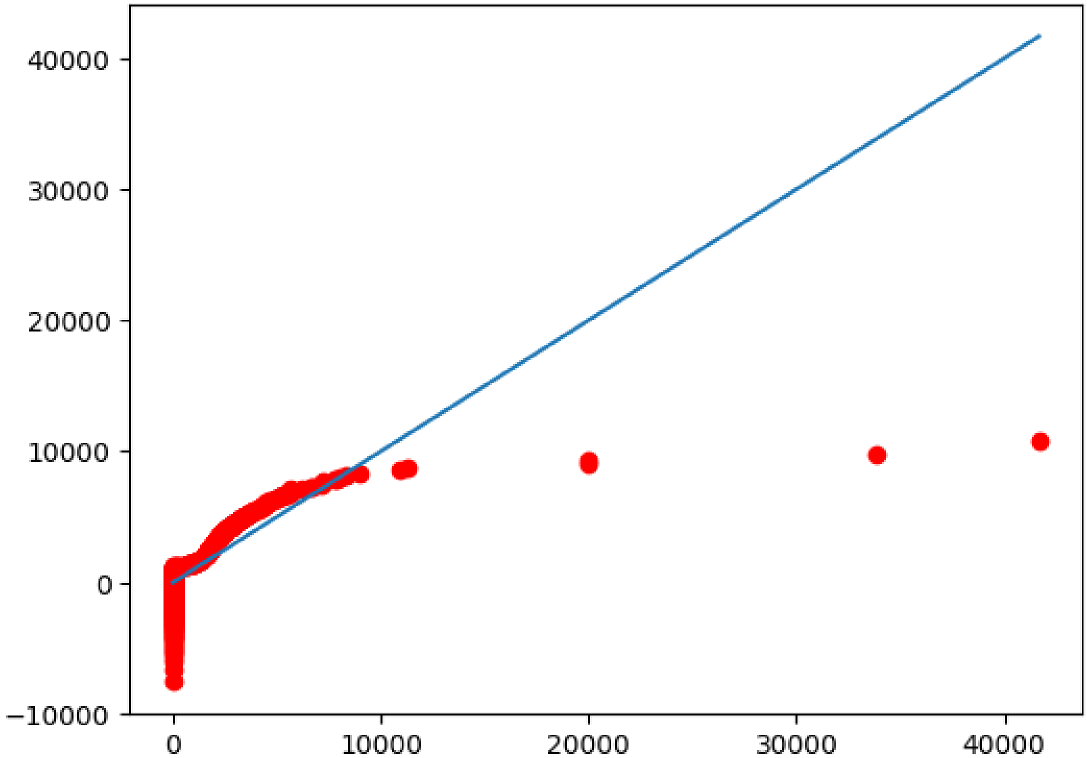
0
100
200
300
400
500
0
10000
20000
30000
40000
50000
60000
70000
80000



0
20000
40000
60000
80000
0
10000
20000
30000
40000
50000
60000
70000
80000



500
400
300
200
100
0
0
10000
20000
30000
40000

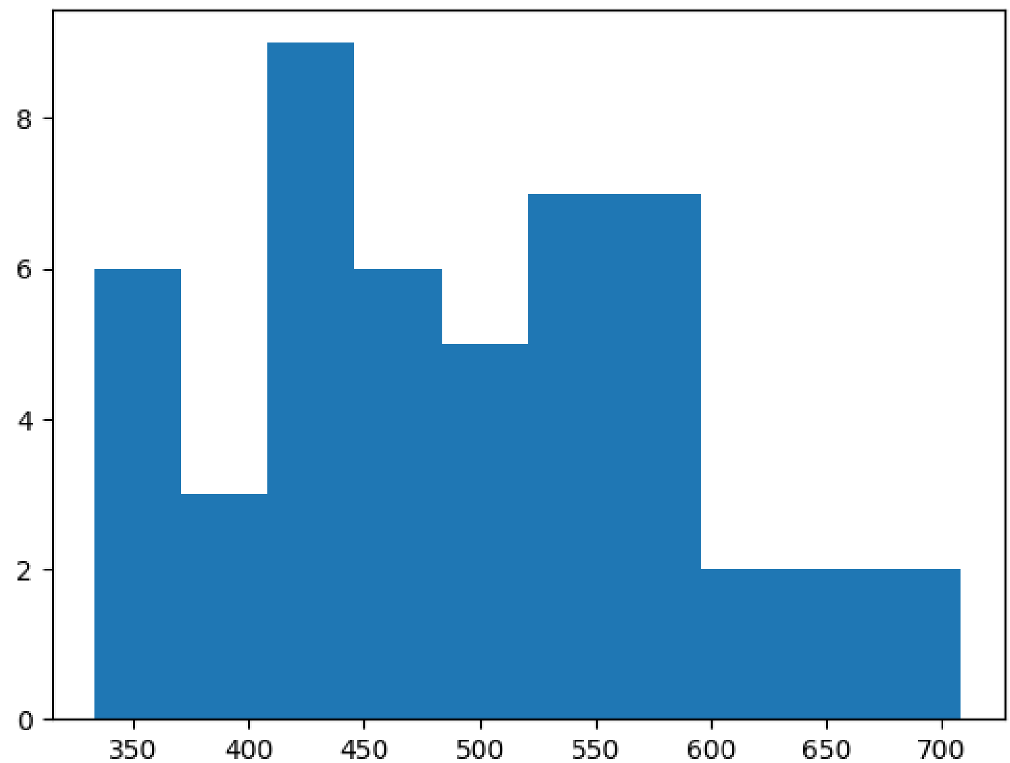

40000
30000
20000
10000
0
−10000
0
10000
20000
30000
40000

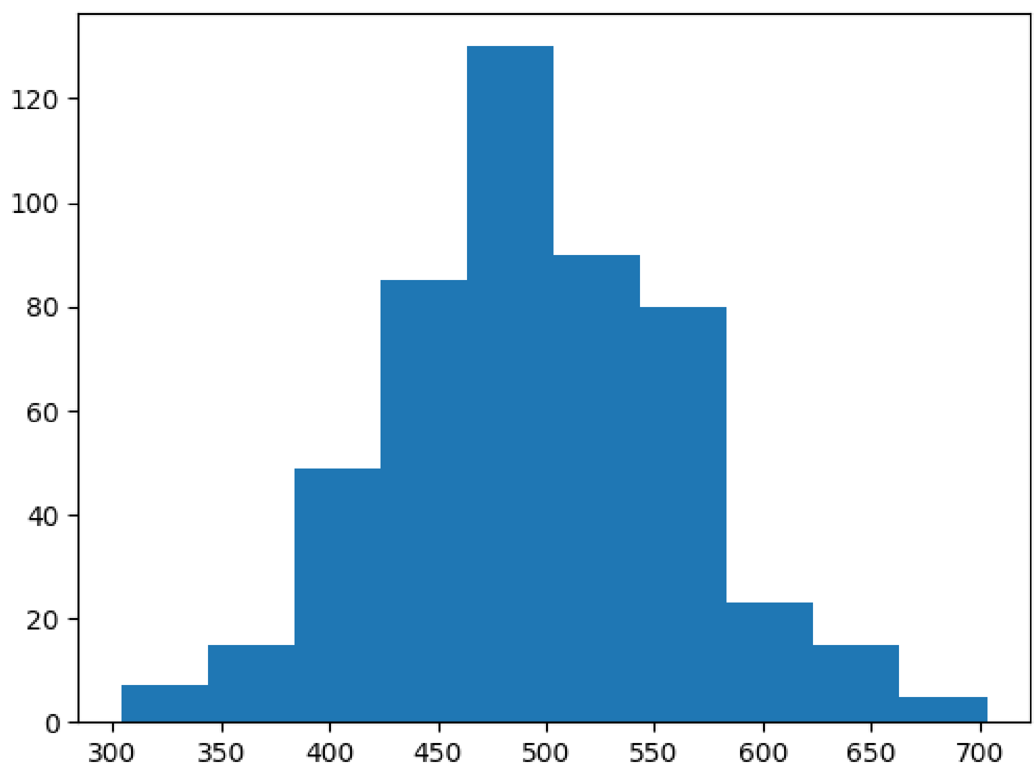
```
[ ]:
```

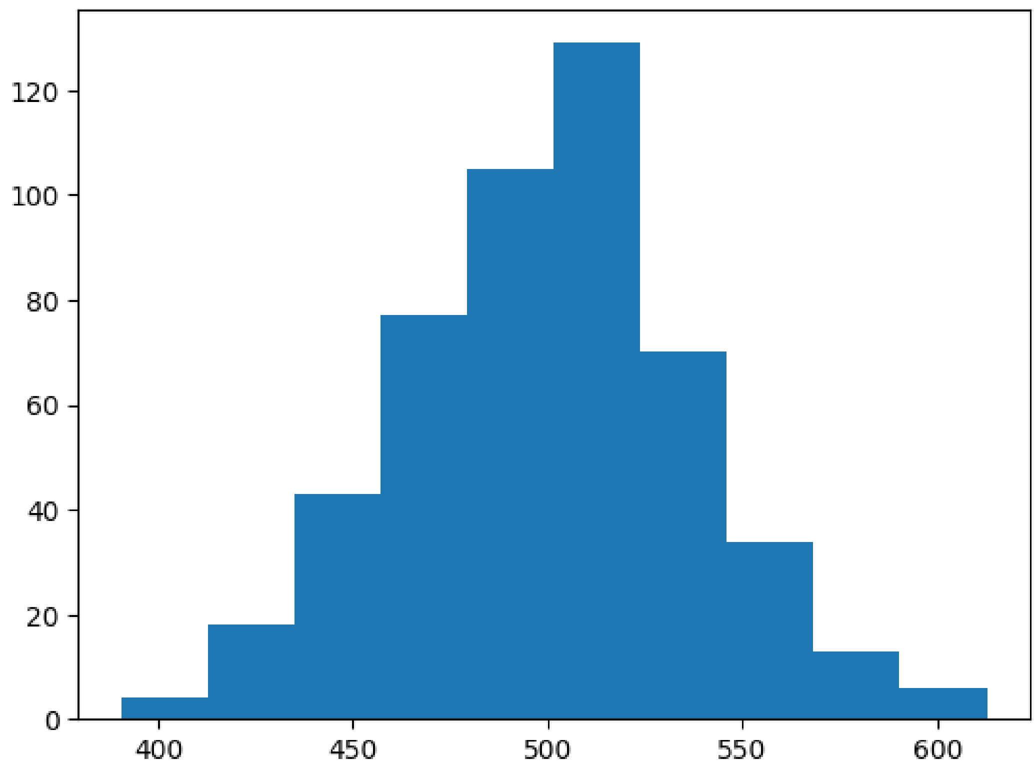
sampling-dist

November 1, 2024

[1]:
```python
# Import necessary libraries
import numpy as np # For numerical operations (not used in this snippet)
import matplotlib.pyplot as plt # For creating plots
from random import sample # To sample elements from a list randomly
from statistics import mean # To calculate the mean of a list of numbers
# Define a function to plot the distribution of sample means
def plot(arr, N, n_samples):
    x = [] # Initialize an empty list to store sample means
    # Loop to take samples and calculate means
    for i in range(1, n_samples): # Iterate n_samples times (excluding the
 ↪first index)
        # To find N samples from the arr
        smp = sample(arr, N) # Randomly sample N elements from the array 'arr'
        mu = mean(smp) # Calculate the mean of the sampled elements
        x.append(mu) # Append the calculated mean to the list x
    plt.hist(x) # Create a histogram of the sample means
    plt.show() # Display the histogram
    # Example data (population)
arr = [i for i in range(1000)] # Create a list of integers from 0 to 999
# Variations of sampling and plotting
plot(arr, 5, 50) # Plot sample means for 50 samples of size 5
plot(arr, 20, 500) # Plot sample means for 500 samples of size 20
plot(arr, 50, 500) # Plot sample means for 500 samples of size 50
# Explanation of the observed results:
# As the number of samples (n_samples) increases, the distribution of the means
 ↪tends to become normal
# As the sample size (N) increases, the spread (flatness) of the distribution
 ↪decreases, indicating more precision in the estimates
```


8
6
4
2
0
350
400
450
500
550
600
650
700

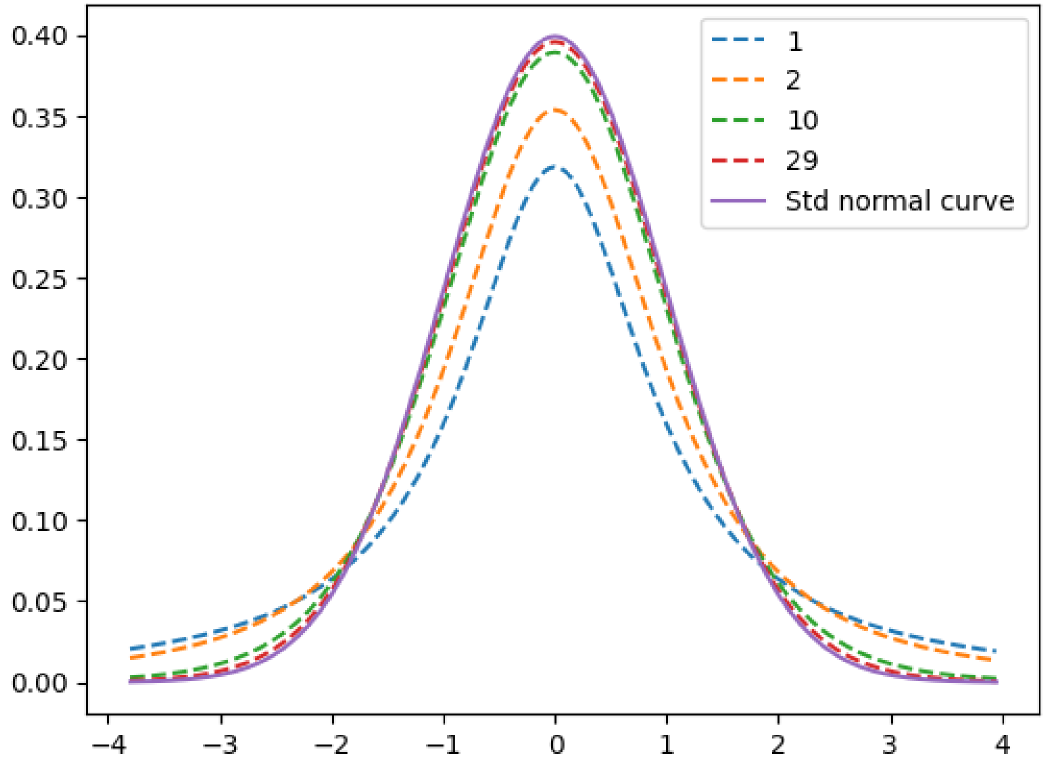

120
100
80
60
40
20
0
300
350
400
450
500
550
600
650
700


120
100
80
60
40
20
0
400
450
500
550
600

studentstdist

November 1, 2024

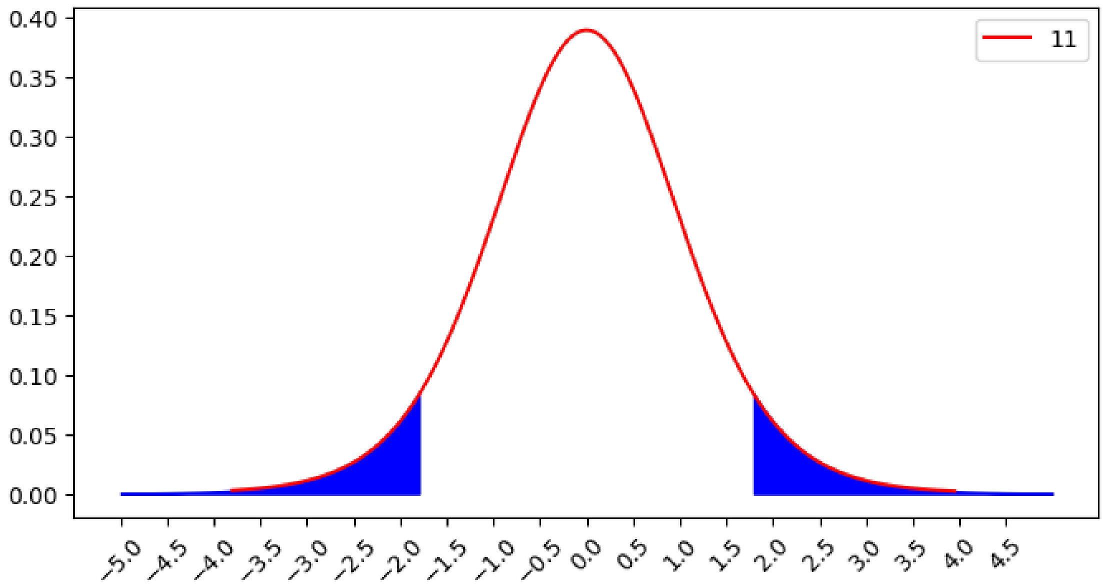
[1]:
```
# Enable inline plotting for Jupyter notebooks
%matplotlib inline
# Import necessary libraries
import matplotlib.pyplot as plt # For creating plots
from scipy.stats import t, norm # For statistical functions related to␣
↪t-distribution and normal distribution
import numpy as np # For numerical operations
import pandas as pd # For data manipulation (not used in this snippet)
# Create an array of values from -3.8 to 4, with increments of 1/20
x = np.arange(-3.8, 4, 1/20) # This serves as the x-axis values for the plots
# Loop through different degrees of freedom for the t-distribution
for i in [1, 2, 10, 29]: # List of degrees of freedom to plot
    # Plotting the t-distribution curves (PDF gives probability density␣
↪function)
    plt.plot(x, t.pdf(x, i), '--', label=i) # Dashed lines for different␣
↪t-distribution curves
# Plotting the standard normal curve
plt.plot(x, norm.pdf(x), label='Std normal curve') # Solid line for the␣
↪standard normal distribution
# Add legend to the upper right of the plot
plt.legend(loc='upper right') # Show the legend with labels for each curve
plt.show() # Display the plot
# Calculate and print the complement of the cumulative distribution function␣
↪(CDF) for the t-distribution
print("1 - cdf gives :", 1 - t.cdf(1.59, 2)) # Tail probability for␣
↪t-distribution with 2 degrees of freedom
print('same as :', t.sf(1.59, 2)) # Calculate the survival function (SF), which␣
↪is equivalent to 1 - CDF
# Calculate and print the tail probabilities for the standard normal␣
↪distribution
print(1 - norm.cdf(2), norm.sf(2)) # Tail probability for the standard normal␣
↪distribution at z = 2
```




```
1 - cdf gives : 0.12639805893063705
same as : 0.12639805893063707
0.022750131948179210 0.0227501319481792
```

[2]:
```python
# Define a function to plot the t-distribution with shaded areas for the␣
↪critical region
def t_table(n, alpha):
    # Calculate the critical value (t-score) for the given alpha level
    s = t.ppf(alpha / 2, n - 1) # PPF is the percent point function (inverse of␣
↪CDF)
    # Set up the figure size for the plot
    plt.figure(figsize=(8, 4))
    # Plot the t-distribution curve for n-1 degrees of freedom
    plt.plot(x, t.pdf(x, n - 1), color='red', label=n - 1) # Red curve for␣
↪t-distribution
    # Calculate the ranges for the areas to be shaded
    section1 = np.arange(-5, s, 1/20.) # Range from -5 to the critical value s
    section2 = np.arange(-s, 5, 1/20.) # Range from -s to 5
    # Fill the areas under the t-distribution curve
    plt.fill_between(section1, t.pdf(section1, n - 1), color='blue') # Fill␣
↪left critical region
```

```
    plt.fill_between(section2, t.pdf(section2, n - 1), color='blue') # Fill␣
↪right critical region
    # Set x-ticks for better readability
    plt.xticks(np.arange(-5, 5, 0.5), rotation=45) # Set ticks and rotate for␣
↪clarity
    plt.legend(loc='upper right') # Show the legend for the plot
    plt.show() # Display the plot
# Call the t_table function with sample size and significance level
t_table(12, 0.1) # Sample size of 12 and alpha level of 0.1
```



[3]:
```
# Create an array of values from -7 to 8, with increments of 1/20
x = np.arange(-7, 8, 1/20)
# Define a function to plot the confidence interval
def ci(t_score, n):
    # Set up the figure size for the plot
    plt.figure(figsize=(8, 4))
    # Calculate the area under the t-distribution curve for the confidence␣
↪interval
    area = t.cdf(t_score, n - 1) - t.cdf(-t_score, n - 1) # Area between␣
↪-t_score and +t_score
    print('Confidence Level', area * 100) # Print the confidence level as a␣
↪percentage
    # Plot the t-distribution curve for n-1 degrees of freedom
    plt.plot(x, t.pdf(x, n - 1), color='red', label=n - 1) # Red curve for␣
↪t-distribution
    # Define the range for the shaded area (confidence interval)
```

```
    section = np.arange(-t_score, t_score, 1/20.) # Range from -t_score to␣
↪+t_score
    # Fill the area between -t_score and +t_score
    plt.fill_between(section, t.pdf(section, n - 1)) # Fill the confidence␣
↪interval area
    # Set x-ticks for better readability
    plt.xticks(np.arange(-6, 7, 0.5), rotation=45) # Set ticks and rotate for␣
↪clarity
    plt.legend(loc='upper right') # Show the legend for the plot
    plt.show() # Display the plot
# Call the ci function with a specific t-score and sample size
ci(5.841, 4) # t-score of 5.841 and sample size of 4
```

```
Confidence Level 99.00004355246759
```