import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import stemgraphic

from scipy.special import comb # For combinatorial calculations (see line 209)

from scipy.stats import hypergeom # For dhyper() equivalent (see line 225)

from scipy.stats import binom # For dbinom() equivalent (see line 264)

from scipy.stats import poisson # For dpois() equivalent (see line 304)

from scipy.stats import nbinom # For dnbinom() equivalent (see line 324)

from scipy.stats import geom # For dgeom() equivalent (see line 338)

from scipy.stats import norm # For pnorm() equivalent (see line 349)

from scipy.stats import probplot # For qqnorm() equivalent (see line 367)

from scipy.stats import lognorm # For plnorm() equivalent (see line 429)

from scipy.stats import expon # For pexp() equivalent (see line 438)

from scipy.stats import gamma # For pgamma() equivalent (see line 453)

from scipy.stats import weibull\_min # For pweibull() equivalent (see line 462)

#Problem 1.A

temp = [127, 125, 131, 124, 129, 121, 142, 151, 160, 125, 124, 123,

120, 119, 128, 133, 137, 124, 142, 123, 121, 136, 140, 137,

125, 124, 128, 129, 130, 122, 118, 131, 125, 133, 141, 125, 140, 131, 129, 126]

temp.sort()

np.median(temp)

#128

#Problem 1.B

#Any amount. It will not change the median

#Problem 1.C

np.mean(temp)

#Mean: 129.975

np.std(temp)

#Std Dev:8.8019

#Problem 1.D

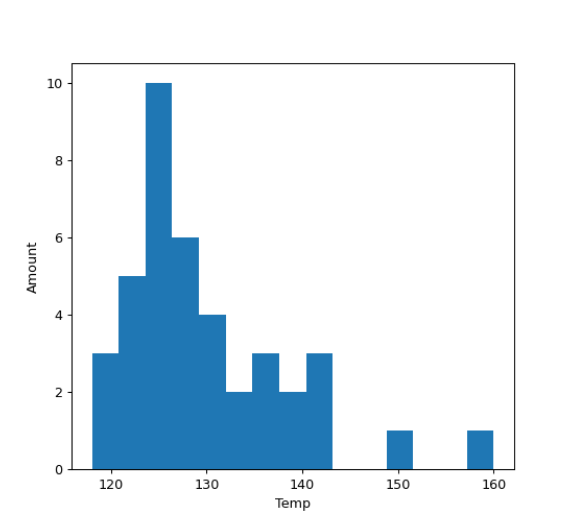
plt.close()

plt.hist(temp, bins=15)

plt.xlabel('Temp')

plt.ylabel('Amount')

plt.show()

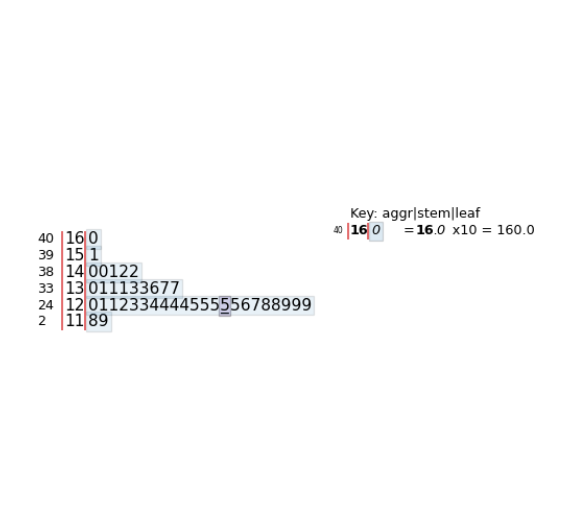


#Problem 1.E

plt.close()

stemgraphic.stem\_graphic(temp, scale=10, leaf\_order=True)

plt.show()



#Problem 1.E

plt.close()

stemgraphic.stem\_graphic(temp, scale=10, leaf\_order=True)

plt.show()

#Problem 1.F

tempQuarts = np.quantile(temp, [.25, .75])

print(tempQuarts)

#124. 133.75

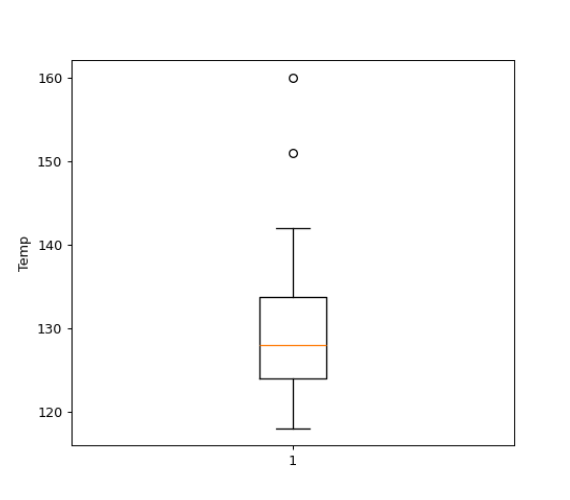
#Problem 1.G

plt.close()

plt.boxplot(temp)

plt.ylabel('Temp')

plt.show()



#Problem 1.H

#Right Leaning. Has 2 outliers.

#Problem 1.I

plt.close()

fig, ax = plt.subplots()

res = probplot(temp, dist="norm")

ax.scatter(res[0][0], res[0][1], label="Data Points") # Scatter plot for octane data

ax.plot(res[0][0], res[1][1] + res[1][0] \* res[0][0], color="red", label="Q-Q Line") # Reference line

ax.set\_title("Q-Q Plot with Data on X-axis")

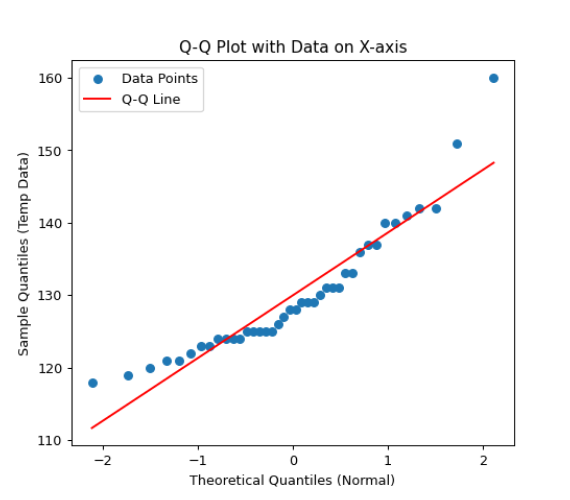
ax.set\_ylabel("Sample Quantiles (Temp Data)")

ax.set\_xlabel("Theoretical Quantiles (Normal)")

ax.legend()

plt.show()

#Yes. The data that isn't an outlier lay on the line for the most part.



#Problem 2.A

n = 3

p = .15

prob = geom.pmf(k=n, p=p)

print(prob)

#.108375

#Problem 2.B

#1/p or 6.6667 patients

#Problem 2.C

n = 50

k = 10

p = .15

prob = binom.pmf(k,n,p)

print(prob)

#.088989

#Problem 3.A

N = 25

n = 5

D = 2

prob = hypergeom.pmf(0, N, D, n)

print(prob)

#.6333333

#Problem 3.B

p = D/N

prob = binom.pmf(0, n, p)

print(prob)

#.65908. Yes it's probably fine, relatively close to the other one

#Problem 3.C

N=150

p=D/N

prob = hypergeom.pmf(0, N, D, n)

print(prob)

prob = binom.pmf(0, n, p)

print(prob)

#Yes! they are .934 and .935 respectively. it's even better.

#Problem 3.D

N = 25

p = .95

D = 5

x = range(1, N+1)

for n in x:

prob = hypergeom.pmf(0,N,D,n)

if 1-prob >= p:

print(n)

break

#should sample 11 items.

#Problem 4

n = 1000

p = .1

x = range(0, n+1)

prob = [poisson.pmf(k, p) for k in x]

print(1-prob[0])

#.09516

#Problem 5

u = 5000

std = 50

p = .005

x = norm.ppf(p, loc = u, scale = std)

print(x)

#4871.2

#Problem 6.A

rate = .1

scale = 1/rate

x = 48

prob = expon.cdf(x, scale=scale)

print(prob)

#.99177

#Problem 6.B

x = 5

prob = 1 - expon.cdf(x, scale=scale)

print(prob)

#.60653

#Problem 7.A

u = 100

std = 2

ll = 97

ul = 102

proportion = norm.cdf(ul, loc=u, scale=std) - norm.cdf(ll, loc=u, scale=std)

print(proportion)

#77.45%

#Problem 7.B

x = np.arange(0, 201, .01)

llCost = [norm.cdf(ll, loc=k, scale=std) for k in x]

ulCost = [(1 - norm.cdf(ul, loc=k, scale=std)) \* 5 for k in x]

table = pd.DataFrame({'llCost': llCost, 'ulCost': ulCost})

sumTable = table['ulCost'] + table['llCost']

minMean = sumTable.idxmin()/100

print(minMean)

#98.21 should minimize

#Problem 8

#using Normal Dist cause n is "large"

p = 12/38

n = 50

u = n\*p

k = 12

std = np.sqrt(u\*(1-p))

prob = 1 - norm.cdf(k, loc=u, scale=std)

print(prob)

#.87553