HW6: CI estimate using BootStrap Sampling and analytical methods.

Total : 50 pts This is an individual assignment.

You can use python to solve some numerical calculations. but make sure to show your steps.

Hint: Most of the problems require CI estimator from data. You can write a general function and call it with different sample set. This is optional.

In [2]:
```
import numpy as np
import numpy.random as npr
import random
import itertools

from plotvec import plotvec, plotvecR

import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
# plt.style.use('ggplot')

import scipy.stats as stats
```

Description

Provide analytical solutions to the problems below and upload your answers as a PDF. (There is no need to type your answers -- just scan your handwritten solutions with an app like CamScanner (https://www.camscanner.com/user/download) or Scannable (https://apps.apple.com/us/app/evernote-scannable/id883338188).)

Problem 1

(total 10 points, a) and b) 2 points each, c) 3 points, d)3 points)

Consider the following samples

$$x = [12.49, 9.85, 9.74, 12.71, 9.38, 7.97]$$

determine the unbiased estimates of

a) the true mean μ_X;

b) the true variance σ_X^2;

c) Find the 90% and 95% confidence intervals of the true mean μ_X using analytical methods.

d) Find the 90% and 95% confidence intervals of the true mean μ_X using Bootstrap sampling methods.

f) What do you observe from the intervals obtained in c) and d)? This is an open ended question. No standard answer.

Solution:

In [3]:
```
x = np.array([12.49, 9.85, 9.74, 12.71, 9.38, 7.97])

# a)
sample_mean = np.mean(x)
print(f"a) true mean: {sample_mean:.4f}")

# b)
sample_var = np.var(x, ddof=1)
print(f"b) true variance: {sample_var:.4f}")

# c)
n = len(x)
std_error = np.sqrt(sample_var / n)
t_90 = stats.t.ppf(0.95, df=n-1)
t_95 = stats.t.ppf(0.975, df=n-1)

ci_90_lower = sample_mean - t_90 * std_error
ci_90_upper = sample_mean + t_90 * std_error
ci_95_lower = sample_mean - t_95 * std_error
ci_95_upper = sample_mean + t_95 * std_error

print(f"c) 90% Confidence interval (analytical): ({ci_90_lower:.4f}, {ci_90_upper:.4f})")
print(f"   95% Confidence interval (analytical): ({ci_95_lower:.4f}, {ci_95_upper:.4f})")

# d)
np.random.seed(42)
num_bootstrap = 10000
bootstrap_means = []

for i in range(num_bootstrap):
    bootstrap_sample = npr.choice(x, size=n, replace=True)
    bootstrap_means.append(np.mean(bootstrap_sample))

ci_90_bootstrap = np.percentile(bootstrap_means, [5, 95])
ci_95_bootstrap = np.percentile(bootstrap_means, [2.5, 97.5])

print(f"d) 90% Confidence interval (bootstrap): ({ci_90_bootstrap[0]:.4f}, {ci_90_bootstrap[1]:.4f})")
print(f"   95% Confidence interval (bootstrap): ({ci_95_bootstrap[0]:.4f}, {ci_95_bootstrap[1]:.4f})")

# Compare analytical and bootstrap CIs
print("\nComparison between analytical and bootstrap CIs:")
print(f"90% CI: Analytical width = {ci_90_upper - ci_90_lower:.4f}, Bootstrap width = {ci_90_bootstrap[1] - (
```

```
print(f"95% CI: Analytical width = {ci_95_upper - ci_95_lower:.4f}, Bootstrap width = {ci_95_bootstrap[1] - c
```

```
a) true mean: 10.3567
b) true variance: 3.4753
c) 90% Confidence interval (analytical): (8.8231, 11.8902)
   95% Confidence interval (analytical): (8.4003, 12.3130)
d) 90% Confidence interval (bootstrap): (9.2050, 11.5867)
   95% Confidence interval (bootstrap): (9.0483, 11.6785)

Comparison between analytical and bootstrap CIs:
90% CI: Analytical width = 3.0671, Bootstrap width = 2.3817
95% CI: Analytical width = 3.9127, Bootstrap width = 2.6301
```

f)

- The bootstrap CIs are slightly narrower than the analytical CIs.
- The analytical method might be more conservative, providing slightly wider intervals.
- The bootstrap method doesn't rely on distributional assumptions, which can be beneficial when the underlying distribution is unknown.

Problem 2

(20 pt)

An Electrical Engineering professor wants to estimate the average time it takes for students to complete a complex circuit design experiment. A sample of 30 students records their completion times (in minutes).

array([44., 65., 67., 47., 53., 69., 74., 56., 76., 57., 59., 68., 57., 61., 58., 65., 63., 63., 48., 65., 57., 59., 60., 60., 60., 66., 55., 44., 76., 49.])

1. Compute a 95% confidence interval for the true average completion time of all students using bootstrap sampling.
2. Assume the true mean is 62, What is the probability that the Bootstrap 95% confidence interval does not contains the true mean?
3. Compute a 95% confidence interval for the true average completion time of all students using analytical methods.
4. Assume the true variance of the student's completion time is 5, Compute a 95% confidence interval for the true average completion time of all students using analytical methods.
5. If the sample size were increased to 100 students, how would the confidence interval change?

In [4]:
```
Xtimes = np.array([44., 65., 67., 47., 53., 69., 74., 56., 76., 57., 59., 68., 57.,
       61., 58., 65., 63., 63., 48., 65., 57., 59., 60., 60., 60., 66.,
       55., 44., 76., 49.])
```

In [5]:
```
# 1.
np.random.seed(42)
n = len(Xtimes)
sample_mean = np.mean(Xtimes)
sample_var = np.var(Xtimes, ddof=1)

print(f"Sample mean: {sample_mean:.4f}")
print(f"Sample variance: {sample_var:.4f}")
print(f"Sample size: {n}")

num_bootstrap = 10000
bootstrap_means = []

for i in range(num_bootstrap):
    bootstrap_sample = npr.choice(Xtimes, size=n, replace=True)
    bootstrap_means.append(np.mean(bootstrap_sample))

ci_95_bootstrap = np.percentile(bootstrap_means, [2.5, 97.5])

print(f"1. 95% Confidence interval (bootstrap): ({ci_95_bootstrap[0]:.4f}, {ci_95_bootstrap[1]:.4f})")

# 2.
true_mean = 62
bootstrap_cis_contain_true_mean = 0
trials = 1000

for i in range(trials):
    bootstrap_sample = npr.choice(Xtimes, size=n, replace=True)
    bootstrap_means_temp = [np.mean(npr.choice(bootstrap_sample, size=n, replace=True)) for _ in range(trials
    ci = np.percentile(bootstrap_means_temp, [2.5, 97.5])

    if ci[0] <= true_mean <= ci[1]:
        bootstrap_cis_contain_true_mean += 1

probability_ci_not_contains_true = 1 - (bootstrap_cis_contain_true_mean / trials)
print(f"2. Probability that 95% CI doesn't contain true mean: {probability_ci_not_contains_true:.4f}")

# 3.
std_error = np.sqrt(sample_var / n)
t_95 = stats.t.ppf(0.975, df=n-1)
ci_95_analytical = [sample_mean - t_95 * std_error, sample_mean + t_95 * std_error]

print(f"3. 95% Confidence interval (analytical): ({ci_95_analytical[0]:.4f}, {ci_95_analytical[1]:.4f})")
```

```
# 4.
known_variance = 5
std_error_known = np.sqrt(known_variance / n)
z_95 = stats.norm.ppf(0.975)
ci_95_known_var = [sample_mean - z_95 * std_error_known, sample_mean + z_95 * std_error_known]

print(f"4. 95% CI with known variance=5: ({ci_95_known_var[0]:.4f}, {ci_95_known_var[1]:.4f})")

# 5.
n_new = 100
std_error_new = np.sqrt(sample_var / n_new)
t_95_new = stats.t.ppf(0.975, df=n_new-1)
ci_95_new = [sample_mean - t_95_new * std_error_new, sample_mean + t_95_new * std_error_new]

print(f"5. 95% CI with n=100 (analytical): ({ci_95_new[0]:.4f}, {ci_95_new[1]:.4f})")
print(f"   Width of CI with n=30: {ci_95_analytical[1] - ci_95_analytical[0]:.4f}")
print(f"   Width of CI with n=100: {ci_95_new[1] - ci_95_new[0]:.4f}")
```

```
Sample mean: 60.0333
Sample variance: 72.4471
Sample size: 30
1. 95% Confidence interval (bootstrap): (57.0658, 62.9000)
2. Probability that 95% CI doesn't contain true mean: 0.2930
3. 95% Confidence interval (analytical): (56.8551, 63.2116)
4. 95% CI with known variance=5: (59.2332, 60.8335)
5. 95% CI with n=100 (analytical): (58.3444, 61.7222)
   Width of CI with n=30: 6.3566
   Width of CI with n=100: 3.3778
```

Problem 3

(20pt)

In this problem you will be working with the [Breast Cancer Data Set (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic))](https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)).

This data set contains 569 samples of digitized images of a fine needle aspirate (FNA) of a breast mass. Each sample describes the mass using 30 features, which include the average radius of the cell present in the FNA image. Each sample is labeled as benign (class = 1) or malignant (class = 0).

In [6]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer(return_X_y = False)
# print(data.DESCR) # uncomment this to learn more about this dataset

df = pd.DataFrame(data = np.hstack((data.target[:,np.newaxis], data.data)),
                        columns = np.concatenate((['Class'],data.feature_names)))
df
```

Out[6]:

	Class	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	...	worst radius	worst texture	wors perimete
0	0.0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380	17.33	184.6
1	0.0	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990	23.41	158.8
2	0.0	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570	25.53	152.5
3	0.0	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910	26.50	98.8
4	0.0	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540	16.67	152.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	0.0	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450	26.40	166.1
565	0.0	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690	38.25	155.0
566	0.0	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980	34.12	126.7
567	0.0	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740	39.42	184.6
568	1.0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456	30.37	59.1

569 rows × 31 columns

** For each class (benign and malignant), consider the mean radius column. Answer the following questions:**

1. (5pt) Compute 95% confidence intervals for the mean radius for both benign (class = 1) and malignant (class = 0) tumors.
2. (5pt) Compute 98% confidence intervals for the mean radius for both benign (class = 1) and malignant (class = 0) tumors.
3. (10pt) Confidence Interval for Classification

We compute separate confidence intervals for the mean radius of benign and malignant tumors.

- If the confidence interval of a new sample's feature (e.g., mean radius) falls within the benign range, it suggests the tumor is more likely benign.
- If the sample's feature falls within the malignant range, it suggests the tumor is more likely malignant.
- If there is overlap between the confidence intervals, additional features should be considered.

Given a newly sampled mean_radius = 15.5, does this patient fall into

A. Likely Benign,

B. Likely Malignant,

C. Uncertain, needs more data.

Answer this question using 95% CI and 98% CI respectively. You can implement a decision function to do it. Or just answer it by observing the CI computed.

In [7]:
```
# Create a function to compute CI for each class
def compute_ci(series, confidence=0.95):
    n = len(series)
    mean = np.mean(series)
    std = np.std(series, ddof=1)
    ste_error = std / np.sqrt(n)
    t_value = stats.t.ppf((1 + confidence) / 2, df=n-1)

    ci_lower = mean - t_value * std_error
    ci_upper = mean + t_value * std_error

    return ci_lower, ci_upper

benign = df[df['Class'] == 1]
malignant = df[df['Class'] == 0]
```

In [8]:
```
# 1.
ci_95_benign = compute_ci(benign['mean radius'], confidence=0.95)
ci_95_malignant = compute_ci(malignant['mean radius'], confidence=0.95)

print("1. 95% Confidence Intervals for Mean Radius:")
print(f"   Malignant (CI = ({ci_95_benign[0]:.4f}, {ci_95_benign[1]:.4f})")
print(f"   Benign (CI = ({ci_95_malignant[0]:.4f}, {ci_95_malignant[1]:.4f})")

# 2.
ci_98_benign = compute_ci(benign['mean radius'], confidence=0.98)
ci_98_malignant = compute_ci(malignant['mean radius'], confidence=0.98)

print("2. 98% Confidence Intervals for Mean Radius:")
print(f"   Malignant (CI = ({ci_98_benign[0]:.4f}, {ci_98_benign[1]:.4f})")
print(f"   Benign (CI = ({ci_98_malignant[0]:.4f}, {ci_98_malignant[1]:.4f})")


# 3.
def classify_radius(r, ci_benign, ci_malignant):
    in_benign = ci_benign[0] <= r <= ci_benign[1]
    in_malignant = ci_malignant[0] <= r <= ci_malignant[1]

    if in_benign and not in_malignant:
        return "A. Likely Benign"
    elif in_malignant and not in_benign:
        return "B. Likely Malignant"
    elif in_benign and in_malignant:
        return "C. Uncertain, needs more data"
    else:
        return "C. Uncertain, needs more data"

# Apply for both CIs
new_radius = 15.5
decision_95 = classify_radius(new_radius, ci_95_benign, ci_95_malignant)
decision_98 = classify_radius(new_radius, ci_98_benign, ci_98_malignant)

print(f"3. Classification for new sample with mean_radius = {new_radius}:")
print(f"   Using 95% CI: {decision_95}")
print(f"   Using 98% CI: {decision_98}")
```

```
1. 95% Confidence Intervals for Mean Radius:
   Malignant (CI = (9.0904, 15.2027)
   Benign (CI = (14.3995, 20.5262)
2. 98% Confidence Intervals for Mean Radius:
   Malignant (CI = (8.5150, 15.7780)
   Benign (CI = (13.8200, 21.1056)
3. Classification for new sample with mean_radius = 15.5:
   Using 95% CI: B. Likely Malignant
   Using 98% CI: C. Uncertain, needs more data
```

Additional exercise---Not Graded

An ECE student measures the output voltage of an amplifier circuit in a lab. Due to variability in components, the output voltage varies slightly. A sample of 25 voltage readings is collected.

[4.8, 5.1, 4.9, 5.0, 4.7, 5.2, 5.1, 4.9, 5.0, 4.8, 5.3, 4.9, 5.0, 4.6, 5.1, 4.7, 4.9, 5.2, 5.1, 4.8, 5.0, 5.1, 4.9, 4.7, 5.2]

1. Compute a 90% confidence interval for the true mean output voltage using analytical approach.
2. The circuit is designed to output exactly 5.0V. Does the confidence interval suggest that the circuit is well-calibrated?
3. If we wanted a more precise confidence interval, what do you recommend the student to do?

In [9]:
```
Volts= np.array([4.8, 5.1, 4.9, 5.0, 4.7, 5.2, 5.1, 4.9, 5.0, 4.8, 5.3, 4.9, 5.0, 4.6, 5.1, 4.7, 4.9, 5.2, 5
```

In [10]:
```
n = len(Volts)
sample_mean = np.mean(Volts)
sample_std = np.std(Volts, ddof=1)  # Using Bessel's correction for unbiased estimate
std_error = sample_std / np.sqrt(n)

# 1.
confidence_level = 0.90
alpha = 1 - confidence_level
t_critical = stats.t.ppf(1 - alpha/2, df=n-1)

ci_lower = sample_mean - t_critical * std_error
ci_upper = sample_mean + t_critical * std_error

print(f"1. 90% CI: ({ci_lower:.4f}, {ci_upper:.4f})")

# 2.
print("2. Target value (5.0V) is within the 90% confidence interval. This suggests that the circuit is well-c

# 3.
print("3. Increase the sample size")
```

```
1. 90% CI: (4.8968, 5.0232)
2. Target value (5.0V) is within the 90% confidence interval. This suggests that the circuit is well-calibr
ated.
3. Increase the sample size
```