Applied Stochastic Assignment 3

Manyara Baraka - mbaraka

October 27, 2024

Question 1: Random Vectors and Principal Component Analysis

Part 1: Understanding the Covariance Matrix of Random Vectors

1. Interpretation of the Covariance Matrix:

a. Diagonal Elements
Elements represent the variance of the corresponding individual features. For instance: For a random vector $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & X_3 & X_4 \end{bmatrix}^T$, the diagonal elements of the covariance matrix are:

$$\begin{aligned} \text{Var}(X_1) &= 5 \\ \text{Var}(X_2) &= 0.5 \\ \text{Var}(X_3) &= 3 \\ \text{Var}(X_4) &= 2 \end{aligned}$$

b. Off-diagonal Elements
Represents the covariance of two different features and how much they vary from each other and change together. For example:

$$\begin{aligned} \text{Cov}(X_1, X_2) &= 1.2 \\ \text{Cov}(X_1, X_3) &= 0.8 \\ \text{Cov}(X_1, X_4) &= 0.6 \end{aligned}$$

Positive values indicate that as one feature increases, the other tends to increase as well, while negative values indicate an inverse relationship.

2. Random Vector and Variance:

a. The total variance of Random vector X

$$\begin{aligned} X &= 5 + 4 + 3 + 2 \\ &= 14 \end{aligned}$$

b. Compute Variance of Single feature:
For a single feature, you use the diagonal elements:

e.g X_1 = 5, X_2 = 4, X_3 = 3 ....

3. Eigenvalues and eigenvectors

a. Calculate Eigenvalues:

The covariance matrix Σ_X is given by:

$$\Sigma_X = \begin{bmatrix} 5 & 1.2 & 0.8 & 0.6 \\ 1.2 & 4 & 0.5 & 0.3 \\ 0.8 & 0.5 & 3 & 0.2 \\ 0.6 & 0.3 & 0.2 & 2 \end{bmatrix}$$

To find the eigenvalues, we solve the characteristic equation:

$$\det(\Sigma_X - \lambda I) = 0$$

Where $\Sigma_X - \lambda I$ is:

$$\begin{bmatrix} 5-\lambda & 1.2 & 0.8 & 0.6 \\ 1.2 & 4-\lambda & 0.5 & 0.3 \\ 0.8 & 0.5 & 3-\lambda & 0.2 \\ 0.6 & 0.3 & 0.2 & 2-\lambda \end{bmatrix}$$

Expand this determinant by cofactor expansion along the first row:

$$\det(\Sigma_X - \lambda I) = (5-\lambda)\det\begin{vmatrix} 4-\lambda & 0.5 & 0.3 \\ 0.5 & 3-\lambda & 0.2 \\ 0.3 & 0.2 & 2-\lambda \end{vmatrix} - 1.2\det\begin{vmatrix} 1.2 & 0.5 & 0.3 \\ 0.8 & 3-\lambda & 0.2 \\ 0.6 & 0.2 & 2-\lambda \end{vmatrix} +$$

$$0.8\det\begin{vmatrix} 1.2 & 4-\lambda & 0.3 \\ 0.8 & 0.5 & 0.2 \\ 0.6 & 0.3 & 2-\lambda \end{vmatrix} - 0.6\det\begin{vmatrix} 1.2 & 4-\lambda & 0.5 \\ 0.8 & 0.5 & 3-\lambda \\ 0.6 & 0.3 & 0.2 \end{vmatrix} = 0$$

Now, compute each 3x3 determinant:

First Term:

$$(5-\lambda)\det\begin{vmatrix} 4-\lambda & 0.5 & 0.3 \\ 0.5 & 3-\lambda & 0.2 \\ 0.3 & 0.2 & 2-\lambda \end{vmatrix}$$

Expanding:

$$= (5-\lambda)(5.96 - 5\lambda + \lambda^2 - 0.04)$$

Simplifying and expanding:

$$\begin{aligned}(5-\lambda)(5.92 - 5\lambda + \lambda^2) &= 5(5.92 - 5\lambda + \lambda^2) - \lambda(5.92 - 5\lambda + \lambda^2) \\ &= 5 \cdot 5.92 - 5 \cdot 5\lambda + 5 \cdot \lambda^2 - \lambda \cdot 5.92 + \lambda \cdot 5\lambda - \lambda \cdot \lambda^2 \\ &= 29.6 - 25\lambda + 5\lambda^2 - 5.92\lambda + 5\lambda^2 - \lambda^3 \\ &= 29.6 - 30.92\lambda + 10\lambda^2 - \lambda^3\end{aligned}$$

Second Term:

$$-1.2\det\begin{vmatrix} 1.2 & 0.5 & 0.3 \\ 0.8 & 3-\lambda & 0.2 \\ 0.6 & 0.2 & 2-\lambda \end{vmatrix}$$

Expand:

$$1.2(5.96 - 5\lambda + \lambda^2) - 0.5(1.48 - 0.8\lambda) + 0.3(-1.64 + 0.6\lambda)$$

Expanding:

$$= 1.2(5.96 - 5\lambda + \lambda^2) - 0.74 + 0.4\lambda - 0.492 + 0.18\lambda$$

$$= 1.2(5.96 - 5\lambda + \lambda^2) - 0.74 + 0.4\lambda - 0.492 + 0.18\lambda$$

$$= 7.152 - 6\lambda + 1.2\lambda^2 - 0.74 + 0.4\lambda - 0.492 + 0.18\lambda$$

$$= 7.152 - 6.252\lambda + 1.2\lambda^2$$

Multiplying by -1.2:

$$= -1.2(7.152 - 6.252\lambda + 1.2\lambda^2)$$

$$= -8.5824 + 7.5024\lambda - 1.44\lambda^2$$

$$= -1.2(7.152 - 6.252\lambda + 1.2\lambda^2)$$

$$= -8.5824 + 7.5024\lambda - 1.44\lambda^2$$

Third Term:

$$+0.8 \det \begin{vmatrix} 1.2 & 4-\lambda & 0.3 \\ 0.8 & 0.5 & 0.2 \\ 0.6 & 0.3 & 2-\lambda \end{vmatrix}$$

Expand:

$$-4.81 + 4.6\lambda - \lambda^2$$

Multiplying by 0.8:

$$0.8(-4.81 + 4.6\lambda - \lambda^2)$$

$$= -3.848 + 3.68\lambda - 0.8\lambda^2$$

$$= 0.8(-4.81 + 4.6\lambda - \lambda^2)$$
$$= -3.848 + 3.68\lambda - 0.8\lambda^2$$

Fourth Term:

$$-0.6 \det \begin{vmatrix} 1.2 & 4-\lambda & 0.5 \\ 0.8 & 0.5 & 3-\lambda \\ 0.6 & 0.3 & 0.2 \end{vmatrix}$$

Expand:

$$= 1.2(-0.8 + 0.3\lambda) - (4 - \lambda)(-1.64 + 0.6\lambda) + 0.5(-0.06)$$

$$-0.96 + 0.36\lambda - 6.56 + 4.04\lambda - 0.6\lambda^2 - 0.03$$
$$= -7.55 + 4.4\lambda - 0.6\lambda^2$$

Multiply by -0.6:

$$-0.6(-7.55 + 4.4\lambda - 0.6\lambda^2)$$
$$= 4.53 - 2.64\lambda + 0.36\lambda^2$$

Combine all equations:

$$-\lambda^3 + 7.76\lambda^2 - 19.7376\lambda + 17.1696 = 0$$

Eigenvalues are:

$$\lambda_1 \approx 6.203, \quad \lambda_2 \approx 1.88, \quad \lambda_3 \approx 3.206, \quad \lambda_4 \approx 2.710$$

Eigenvectors:

$$\begin{bmatrix} 0.78907948 & 0.16466637 & 0.52644338 & 0.27036259 \\ 0.51741855 & 0.03622962 & -0.84693231 & 0.11692356 \\ 0.28804636 & 0.04206535 & 0.04585165 & -0.95559271 \\ 0.16328172 & -0.98478572 & 0.05882726 & 0.00869061 \end{bmatrix}$$

b. List eigenvalues and what they represent

$$\lambda_1 \approx 6.203, \quad \lambda_3 \approx 3.206, \quad \lambda_4 \approx 2.710, \quad \lambda_2 \approx 1.88,$$

These eigenvalues represent the variance explained by each principal component.

Part 2: Principal Component Analysis (PCA)

1. Principal Component Directions:

(a) Description of Principal Component Directions

Eigenvectors give the direction of each principle component and points in the data space, the associated eigenvalues give the amount of the intensity of variance captured by each principle component.
A higher eigenvalue suggests a greater spread of data showing a more informative feature for the data space. Example from the above-worked solution:

- Eigenvector 1 corresponds to $\lambda_1 \approx 6.203$ which captures the most variance.
- Eigenvector 2 corresponds to $\lambda_3 \approx 3.206$.
- Eigenvector 3 corresponds to $\lambda_4 \approx 2.710$.
- Eigenvector 4 corresponds to $\lambda_2 \approx 1.88$ and captures the least variance.

(b) Concept of Orthogonality in PCA Orthogonality in PCA indicates that the principle components that is the eigenvectors are perpendicular to each other. This concept is important because:

- **Independence**: This shows that the vectors are uncorrelated hence reducing overlapping and enhances capturing unique information/
- **Interprating**: Since the orthogonal components are uncorrelated it is easier to interpret the results of each component as they can be analyzed independently.

2. Transformation of Random Vector:

a. Covariance Matrix of Y:

$$\Sigma_{\mathbf{Y}} = \mathbf{P}^T \Sigma_{\mathbf{X}} \mathbf{P}$$

Since **P** is orthogonal, the covariance matrix with eigenvalues on its diagonal. Therefore:

$$\text{Cov}(Y) = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix} = \begin{pmatrix} 6.203 & 0 & 0 & 0 \\ 0 & 3.206 & 0 & 0 \\ 0 & 0 & 2.710 & 0 \\ 0 & 0 & 0 & 1.88 \end{pmatrix} \tag{1}$$

b. Effect on Correlation Between Transformed Features

The transformation leads to the features becoming uncorrelated since the covariance matrix of Y is diagonal. The transformed features capture the distinct portion of variance without any correlation with other components this results in the components becoming or being considered independent.

Part 3: Performing PCA by Hand on a Simple Dataset

$$Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}^T$$

$$Y = \begin{pmatrix} 1.2 & 2.8 \\ 0.8 & 2.4 \\ 1.6 & 3.2 \\ 1.4 & 2.9 \end{pmatrix}$$

Step 1: Mean Centering:

a. Mean for Y_1 and Y_2

$$\text{Mean of } Y_1 = \frac{1.2 + 0.8 + 1.6 + 1.4}{4} = \frac{5.0}{4} = 1.25$$

$$\text{Mean of } Y_2 = \frac{2.8 + 2.4 + 3.2 + 2.9}{4} = \frac{11.3}{4} = 2.825$$

b. Center data

$$Y_{\text{centered}} = Y - \begin{bmatrix} 1.25 \\ 2.825 \end{bmatrix} = \begin{bmatrix} -0.05 & -0.025 \\ -0.45 & -0.425 \\ 0.35 & 0.375 \\ 0.15 & 0.075 \end{bmatrix}$$

Step 2: Covariance Matrix:

The covariance matrix:

$$\text{Cov}(\mathbf{Y}_{\text{centered}}) = \frac{1}{n-1}\mathbf{Y}_{\text{centered}}^T\mathbf{Y}_{\text{centered}}$$

$n = 4$, the scaling factor is $\frac{1}{3}$ and

$$Y_1 = [-0.05, -0.45, 0.35, 0.15]$$
$$Y_2 = [-0.025, -0.425, 0.375, 0.075]$$

The covariance matrix is structure:

$$\text{Cov}(\mathbf{Y}_{\text{centered}}) = \begin{bmatrix} \text{Var}(Y_1) & \text{Cov}(Y_1, Y_2) \\ \text{Cov}(Y_1, Y_2) & \text{Var}(Y_2) \end{bmatrix}$$

Therefore:

$$\text{Var}(Y_1) = \frac{1}{n-1}\sum_{i=1}^{n}(Y_{1,i} - \bar{Y}_1)^2$$

substituting

$$\text{Var}(Y_1) = \frac{0.35}{3} = 0.1167$$

$$\text{Var}(Y_2) = \frac{1}{n-1}\sum_{i=1}^{n}(Y_{2,i} - \bar{Y}_2)^2$$

substituting

$$\text{Var}(Y_2) = \frac{0.3275}{3} = 0.1092$$

Calculate Cov:

$$\text{Cov}(Y_1, Y_2) = \frac{1}{n-1}\sum_{i=1}^{n}(Y_{1,i} - \bar{Y}_1)(Y_{2,i} - \bar{Y}_2)$$

substituting

$$\text{Cov}(Y_1, Y_2) = \frac{0.335}{3} = 0.1117$$

Covariance matrix is:

$$\text{Cov}(\mathbf{Y}_{\text{centered}}) = \begin{bmatrix} 0.1167 & 0.1117 \\ 0.1117 & 0.1092 \end{bmatrix}$$

Step 3: Eigenvalue Decomposition:

a. Eigen values and vectors We calculate eigen values by:

$$\det\begin{bmatrix} 0.1167-\lambda & 0.1117 \\ 0.1117 & 0.1092-\lambda \end{bmatrix} = 0$$

Expanding:

$$(0.1167-\lambda)(0.1092-\lambda) - (0.1117)^2 = 0$$

$$\lambda^2 - (0.1167 + 0.1092)\lambda + (0.1167 \times 0.1092 - 0.1117^2) = 0$$

$$\lambda^2 - 0.2259\lambda + 0.002705 = 0$$

Solving we find Eigenvalues:

$$\lambda_1 = 0.22465$$

$$\lambda_2 = 0.00119$$

Find **Eigenvectors for each eigenvalue by solving:**

$$(\text{Cov} - \lambda I)\vec{v} = 0$$

Eigenvector for $\lambda_1 = 0.22465$:
Substitute:

$$\begin{bmatrix} 0.11667 - 0.22465 & 0.11167 \\ 0.11167 & 0.10917 - 0.22465 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} -0.10798 & 0.11167 \\ 0.11167 & -0.11548 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\vec{v}_1 = \begin{bmatrix} 0.71888 \\ 0.69514 \end{bmatrix}$$

Eigenvector for $\lambda_2 = 0.00119$:
Substitute:

$$\begin{bmatrix} 0.11667 - 0.00119 & 0.11167 \\ 0.11167 & 0.10917 - 0.00119 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} 0.11548 & 0.11167 \\ 0.11167 & 0.10798 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\vec{v}_2 = \begin{bmatrix} -0.69514 \\ 0.71888 \end{bmatrix}$$

b. Identify Principal component
The principal component corresponding to the largest eigenvalue (0.22465) is:

$$\begin{bmatrix} 0.71888 \\ 0.69514 \end{bmatrix}$$

Step 4: Project the Data:

Project the centered data onto the principal component axis:

$$Y_{\text{projected}} = Y_{\text{centered}} \cdot \begin{bmatrix} 0.71888 \\ 0.69514 \end{bmatrix}$$

$$Y_{\text{projected}} = \begin{bmatrix} -0.05 & -0.45 & 0.35 & 0.15 \\ -0.025 & -0.425 & 0.375 & 0.075 \end{bmatrix} \cdot \begin{bmatrix} 0.71888 \\ 0.69514 \end{bmatrix}$$
$$= \begin{bmatrix} -0.05332, -0.61893, 0.51228, 0.15997 \end{bmatrix}$$

Part 4: PCA in Practice with a Larger Dataset

Step 1: Load the Dataset

a & b: Load data and Cov:

```
# Import libraries
import pandas as pd

data = pd.read_csv("user_activity_data.csv")
data.cov()
```

	Usage Time (minutes)	Interactions	Activity Type 1	Activity Type 2
Usage Time (minutes)	385.143174	2.892147	-3.147290	1.703028
Interactions	2.892147	33.422922	-1.638074	0.555387
Activity Type 1	-3.147290	-1.638074	24.566131	-1.052789
Activity Type 2	1.703028	0.555387	-1.052789	8.352480

Step 2: Perform PCA

Perform PCA, eigen values and eigen vector

```
from sklearn.decomposition import PCA
import numpy as np

# Perform PCA
pca = PCA()
pca.fit(data)

#Calculate eigen velues and eigenvectors
eigenvalues = pca.explained_variance_
eigenvectors = pca.components_

print("Eigenvalues:", eigenvalues)
print("Eigenvectors:\n", eigenvectors)
```

Eigenvalues:

$$\begin{bmatrix} 385.20247624 \\ 33.70345427 \\ 24.30801924 \\ 8.27075746 \end{bmatrix}$$

Eigenvectors:

$$\begin{bmatrix} 0.99991691 & 0.00826886 & -0.00877718 & 0.00455545 \\ -0.00981203 & 0.98389093 & -0.17625618 & 0.0282155 \\ 0.00741691 & 0.17771336 & 0.98235299 & -0.05784073 \\ -0.00385725 & -0.01755614 & 0.06196225 & 0.99791662 \end{bmatrix}$$

b. Components needed for 90%

Number of Components for **90% Variance: 2**

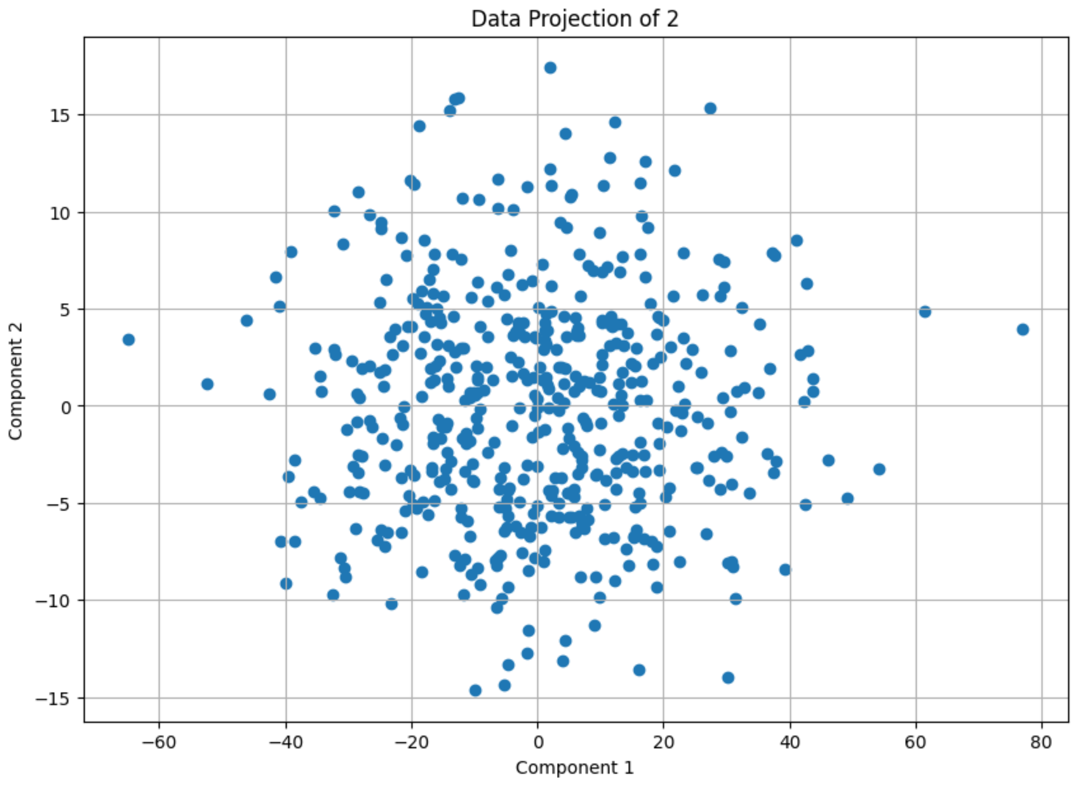
```
# Number of components to retain 90% of the total variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)
num_components = np.argmax(cumulative_variance >= 0.90) + 1

# print("Cumulative Variance:", cumulative_variance)
print("Number of Components for 90% Variance:", num_components)
```

Step 3: Project the Data:

a. Project the first two components

```
import matplotlib.pyplot as plt

# Project the data onto the first two principal components
pca_2d = PCA(n_components=2)
data_2d = pca_2d.fit_transform(data)

# 2D Scatter plot
plt.figure(figsize=(10, 7))
plt.scatter(data_2d[:, 0], data_2d[:, 1])
plt.title("Data Projection of 2")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.grid(True)
plt.show()
```

b. Scatter plot



Step 4: Interpretation of Results

a. Whether well separated.
The data points seem to be highly concentrated around the center this doesn't clearly indicate or show a clear separation between the two components. This also might show that the variance explained by the two components might be limited in representing the significant patterns or differences among the data points

b. How well capture
Due to the central concentration, this might imply that the 2 components may not capture a reasonable proportion of the total dataset variance. Hence the original structure of the data may not be well represented with just the first two components.

Part 5: Interpretation and Business Insights

1. Feature Interpretation in PCA:

(a) Contribution to Principal Components

The component with the highest eigenvalue is the first component and the second highest in the second:

First Principal Component: Usage Time (minutes) with Eigenvalue: 385.20247624
This captures the most variance and this is seen by its **eigen vectors** of [0.9999,0.0098,0.0074,0.0039]. Therefore, Usage Time contributes the most to PC1 (with a coefficient of 0.9999), meaning it is the primary source of variation captured by this component..

Second Principal Component: Interactions with Eigenvalue: 33.70345427. with the **eigen vector** of [0.0083 , 0.9839 , 0.1777 , - 0.0176] This shows that Interactions is the most significant feature in PC2, with a coefficient of 0.9839.

b. Explain feature reduction

The two principal components (PC1 and PC2) provide a general view of user behavior patterns:

- PC1Usage time: Shows how long the user stays active or how much time the user stays on the system. Higher values indicate the user spending time whereas low values indicate a shorter time the user is active on the system.
- PC2Interactions: This shows the user engagement and how focused the user interacts with the system.

2. Using PCA for Future Decision-Making:

a. Faster Processing and Improved Decision-Making
Reducing the dimensionality will allow:

- **Focused and targeted insights:** By using the reduced dimensions will enable decision-makers to focus on the most impactful and meaningful behaviors of the users within the system without being overwhelmed by less significant data points.
- **Fast Data processing:** reduced dimensionality allo efficient and fast data procession since the computational resources will be used effectively speeding up the analysis.

(b) Potential Risks and Mitigation
Potential risks:

- Loss of information, during performing dimension reduction important nuances might be lost hence losing some information
- Complex interactions might not be captured hence this might lead to over-implications

Mitigation:

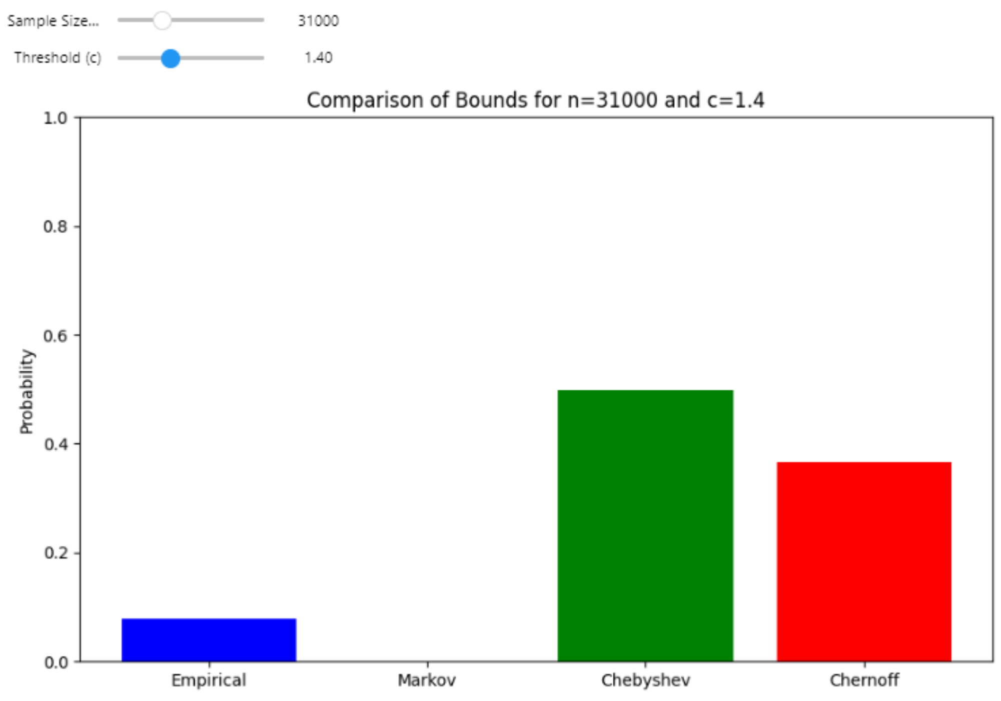
- Constant review - this is a continuous evaluation of the impact of the decisions and outcomes of the reductions and adjust where necessary.
- Ensure during dimension reduction that enough components are retained to capture every critical variance.

Part 5: Comparison of Markov, Chebyshev, and Chernoff Inequalities

1. Simulation

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display

# Step 1: Generate Samples
def generate_samples(n):
    return np.random.normal(0, 1, n)

# Step 2: Compute Empirical Probability
def compute_empirical_probability(samples, c):
    return np.mean(samples >= c)

# Step 3: Calculate Bounds
def calculate_bounds(samples, c):
    mean_X = np.mean(samples)
    var_X = np.var(samples)

    markov_bound = mean_X / c
    chebyshev_bound = var_X / (c - mean_X) ** 2
    chernoff_bound = np.exp(-((c - mean_X) ** 2) / (2 * var_X))

    return markov_bound, chebyshev_bound, chernoff_bound

# Step 4: Plot Results
def plot_results(n, c):
    samples = generate_samples(n)
    empirical_prob = compute_empirical_probability(samples, c)
    markov_bound, chebyshev_bound, chernoff_bound = calculate_bounds(samples, c)

    plt.figure(figsize=(10, 6))
    plt.bar(['Empirical', 'Markov', 'Chebyshev', 'Chernoff'],
            [empirical_prob, markov_bound, chebyshev_bound, chernoff_bound],
            color=['blue', 'orange', 'green', 'red'])
    plt.ylim(0, 1)
    plt.title(f'Comparison of Bounds for n={n} and c={c}')
    plt.ylabel('Probability')
    plt.show()

# Step 5: Dynamic Visualization
n_slider = widgets.IntSlider(value=1000, min=1000, max=100000, step=1000, description
c_slider = widgets.FloatSlider(value=0.5, min=0.5, max=3.0, step=0.1, description='Th
widgets.interactive(plot_results, n=n_slider, c=c_slider)
```

2. Visualization



3. Inferential Questions

a.Inequalities provide the tightest bound for different values of c
Chernoff bound tends to provide the tightest bound, Chebyshev tends to provide a looser bound than Chernoff and Markov is the loosest bound. This is because Chernoff is designed to handle the tail probability for normally distributed data whereas Markov uses only the mean which is quite less informative.

b. Performance of the bounds change as c increases
As increases, empirical probabilities generally decrease, for the bounds:

- Markov's bound as c increases it becomes very loose as it only leverages the mean and since It decays as $1/c$, which is the slowest of the three.
- Chebyshev's bound improves with a larger c but remains less effective than Chernoff. Since its decay rate is proportional to $1/c^2$, which is slower than Chernoff's.
- Chernoff continues to provide the tightest bounds as it decreases exponentially with increasing c.

c. Advantages and disadvantages of each inequality

1. **Markov's Inequality:**
   Advantage: Simple to be applied to any non-negative random variable regardless of the distribution.
   Disadvantage: It gives a very loose bound since it ignores distribution properties.

2. **Chebyshev's Inequality:**
   Advatage: Can be used for any distribution giving more meaningful bound compared to Markov.
   Disadvantage: It's fairly loose for Gausian distribution since it doesn't capture the decay rate for the tails.

3. **Chernoff's Inequality:**
   Advantage: Gives the tightest bound for Gaussian distributions since it leverages exponational decay.
   Disadvantage: Requires the assumption of a Gaussian or sub-Gaussian distribution, so it's not universally applicable.

c. Impact of Sample Size n

Empirical Probability Accuracy: As n (sample size) increases, the empirical probability estimate of $P(X \geq c)$ becomes more accurate. This is due to the Law of Large Numbers, which states that as the sample size grows, the sample mean (or any proportion) converges to the true probability.

Bounds: The bounds (Markov, Chebyshev, and Chernoff) are theoretical and do not depend on n. However, with larger sample sizes, the empirical probability provides a closer approximation to the true probability, allowing us to more accurately assess how tight or loose each bound is.

Question 3: Estimation and Hypothesis Testing

Part 1: Point Estimation; Estimating the Average Battery Life

1. The sample mean and sample varience

Sample Mean ($\bar{X}$)

The sample mean $\bar{X}$:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

where $n = 20$: Sum the data:

$$8.2 + 8.5 + 8.9 + 9.0 + 7.8 + 8.6 + 8.4 + 8.1 + 9.1 + 8.7 + 9.2$$

$$+8.8 + 8.3 + 9.3 + 8.0 + 8.9 + 8.4 + 8.6 + 8.7 + 8.2 = 170.9$$

$$\bar{X} = \frac{170.9}{20} = 8.545$$

Sample Variance (s^2)
The sample variance s^2:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Substitute:

$$(8.2 - 8.545)^2 + (8.5 - 8.545)^2 + \cdots + (8.2 - 8.545)^2 = 3.245$$

Now, calculate the sample variance:

$$s^2 = \frac{3.245}{20-1} = \frac{3.041}{19} = 0.1708$$

Thus:

$$\text{Sample Mean} = 8.585$$

$$\text{Sample Variance} = 0.1708$$

2. Sample mean estimate and is it unbiased.

The sample mean estimates the population mean.
The sample mean is unbiased because the expected value of the sample mean is equal to the population mean:

$$\text{E[X]} = \bar{X} = 8.585$$

3. Mean Square error.

$$\text{MSE}(\bar{X}) = \text{Var}(\bar{X}) + \text{Bias}(\bar{X})^2$$

Since the sample mean is unbiased, the bias is zero.

$$\text{Bias}(\bar{X})^2 = 0$$

Therefore,

$$\text{MSE}(\bar{X}) = \text{Var}(\bar{X})$$

and:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n}$$

Sample variance as an estimate of σ^2:

$$\text{Var}(\bar{X}) = \frac{0.1708}{20} = 0.0085$$

Thus:

$$\text{MSE} = 0.008005$$

4. Effect of sample size change

Error of the Mean: An increase in the sample mean n will lead to a decrease of the error mean and vice versa if it decreases this is because the error is calculated by:

$$\text{SE}(\bar{X}) = \frac{\sigma}{\sqrt{n}}$$

Sample Variance: An increase in sample size will lead to a decrease in the variance and this will lead to the the variance tending to stabilize and become more accurate however a decrease in the sample size will lead to the variance increases and fluctuate more.

5. Effect of outliers on sample mean and varience

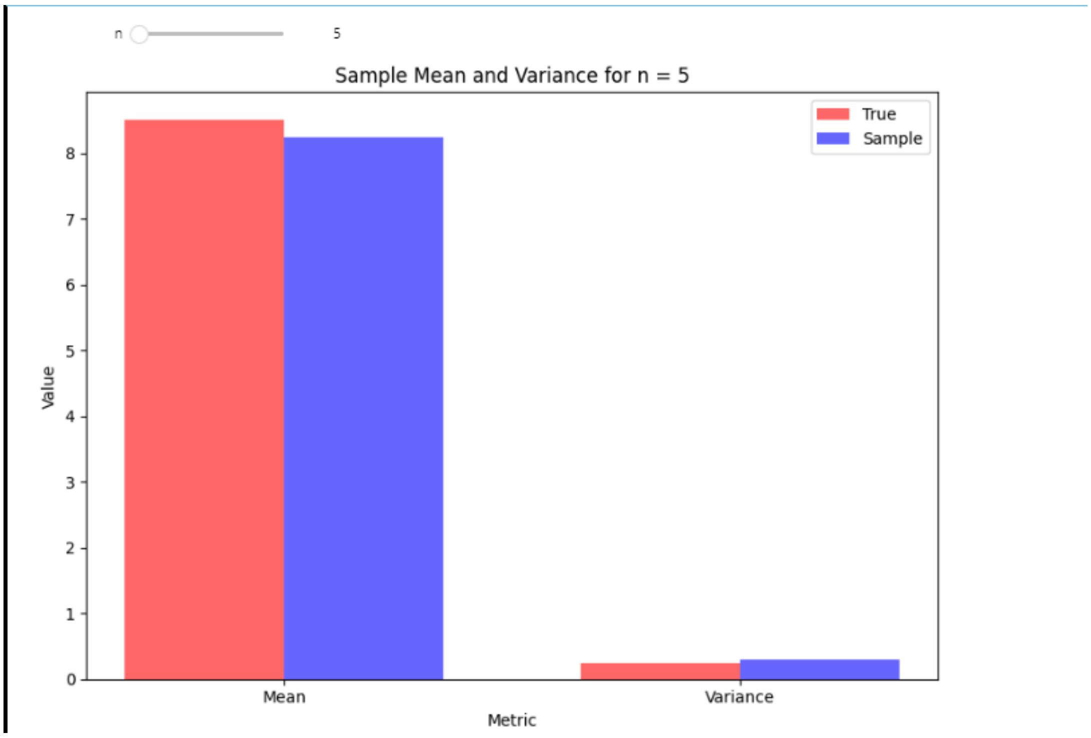
Effect on Mean: This will lead to the mean being skewed since it is more sensitive to outliers this will lead to the mean not representing the centrality of the data.

Effect on Variance: Outliers will increase the variance since the outliers will cause a larger deviation from the mean.

6. Simulation

```
#Import libraries necessary
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider

# set Parameters
```

```
mu = 8.5
sigma = 0.5
true_variance = sigma**2

# Function to simulate and plot
def simulate_and_plot(n):
    # Generate a sample of size n
    sample = np.random.normal(mu, sigma, n)

    # Calculate sample mean and variance
    sample_mean = np.mean(sample)
    sample_variance = np.var(sample, ddof=1)

    # Bar plot
    labels = ['Mean', 'Variance']
    true_values = [mu, true_variance]
    sample_values = [sample_mean, sample_variance]

    # label locations
    x = np.arange(len(labels))
    # width of the bars
    width = 0.35

    plt.figure(figsize=(8, 6))

    # Bar plot for true and sample values
    plt.bar(x - width/2, true_values, width, label='True', color='r', alpha=0.6)
    plt.bar(x + width/2, sample_values, width, label='Sample', color='b', alpha=0.6)

    # Labeling and layout
    plt.xlabel('Metric')
    plt.ylabel('Value')
    plt.title(f'Sample Mean and Variance for n = {n}')
    plt.xticks(x, labels)
    plt.legend()

    plt.tight_layout()
    plt.show()

# Create a widget with a slider
interact(simulate_and_plot, n=IntSlider(min=5, max=1000, step=5));
```



Part 2: Confidence Intervals; Estimating the True Mean Height

Given:

- Sample mean: $\bar{x} = 176$ cm
- Sample standard deviation: $s = 7$ cm
- Sample size: $n = 30$

1. Confidence Intervals of Unknown standard deviation

The confidence interval:

$$\bar{x} \pm t_{\alpha/2} \cdot \frac{s}{\sqrt{n}}$$

Standard error (SE):

$$SE = \frac{s}{\sqrt{n}} = \frac{7}{\sqrt{30}} = 1.278$$

Using the t-table:
Critical t-value: For a 95% confidence level and $n - 1 = 29$ degrees of freedom, is approximately 2.045.
Confidence Interval:

$$CI = 176 \pm 2.045 \times 1.278$$

$$CI = 176 \pm 2.613$$

Confidence interval $\approx (173.387cm, 178.613cm)$.

2. Known standard deviation

We use the z-distribution instead of the t-distribution. The confidence interval:

$$\bar{x} \pm z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}}$$

Where:

$$z_{\alpha/2}$$

The z-critical value for 95% confidence is 1.96.
standard error is the same ($SE = 1.278$):

$$CI = 176 \pm 1.96 \times 1.278$$

$$CI = 176 \pm 2.505$$

Confidence interval $\approx$ (173.495 cm, 178.505 cm).

3. Meaning of confidence interval

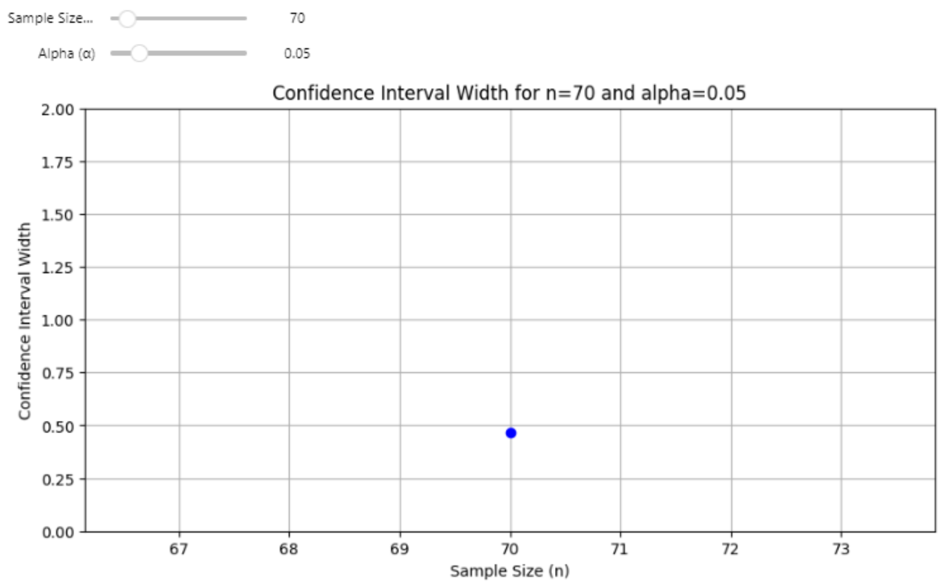
Confidence interval indicates that if more sample intervals are taken, you are certain that the CI of the given percentage e.g. 95% of the intervals will contain the true population mean.
From question 2 we can interpret that we are 95% confident that the true average height of the adult male in the city lies between the upper and lower bound of [173.495 cm, 178.505 cm]

4. Increase in sample size

Increasing the sample size will lead to the reduction of the standard error $\frac{s}{\sqrt{n}}$ and this will lead to making the CI narrower which will give a more precise estimate of the population mean.

5. Simulation

```
# Import libraries
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display

# Function to calculate confidence interval
def confidence_interval(n, alpha):
    # For simplicity use mean = 0 and std = 1
    sample_mean = 0
    sample_std = 1
    z = stats.norm.ppf(1 - alpha / 2)
    margin_of_error = z * (sample_std / np.sqrt(n))
    return sample_mean - margin_of_error, sample_mean + margin_of_error

# Function to update plot
def update_plot(n, alpha):
    ci_lower, ci_upper = confidence_interval(n, alpha)
    ci_width = ci_upper - ci_lower

    plt.figure(figsize=(10, 5))
    plt.plot([n], [ci_width], 'bo')
    plt.title(f'Confidence Interval Width for n={n} and alpha={alpha}')
    plt.xlabel('Sample Size (n)')
    plt.ylabel('Confidence Interval Width')
    plt.ylim(0, 2)
    plt.grid(True)
    plt.show()

# Widgets for sample size and alpha
n_slider = widgets.IntSlider(value=10, min=10, max=500,
step=10, description='Sample Size (n)')
alpha_slider = widgets.FloatSlider(value=0.05, min=0.01,
max=0.2, step=0.01, description='Alpha ()')

# Interactive plot
widgets.interactive(update_plot, n=n_slider, alpha=alpha_slider)
```



Part 3: Hypothesis Testing; Comparing Two Webpage Designs (A/B Testing)

We are given the following data:

- Total visitors: 600
- Clicks on Version A: 240
- Clicks on Version B: 290

1. Proportions of clicks for each version

$$\hat{p}_A = \frac{\text{Clicks on Version A}}{\text{Total visitors}} = \frac{240}{600} = 0.4 = 40\%$$

$$\hat{p}_B = \frac{\text{Clicks on Version B}}{\text{Total visitors}} = \frac{290}{600} = 0.4833 = 48.33\%$$

2. Hypothesis test and Z statistic

Hypotheses

$$H_0 : p_A = p_B$$
$$H_1 : p_A \neq p_B$$

Test Statistic (z-statistic) given by:

$$z = \frac{\hat{p}_A - \hat{p}_B}{\sqrt{p_{pool}(1 - p_{pool})\left(\frac{1}{n_A} + \frac{1}{n_B}\right)}}$$

Where:

$$\hat{p}_A = 0.4 \quad \text{(proportion for Version A)}$$
$$\hat{p}_B = 0.4833 \quad \text{(proportion for Version B)}$$
$$n_A = 600 \quad \text{(sample size for Version A)}$$
$$n_B = 600 \quad \text{(sample size for Version B)}$$

The pooled proportion p_{pool} is:

$$p_{pool} = \frac{\text{Clicks on A} + \text{Clicks on B}}{\text{Total visitors on A} + \text{Total visitors on B}} = \frac{240 + 290}{600 + 600} = \frac{530}{1200} = 0.4417$$

Compute the denominator:

$$\sqrt{0.4417 \times (1 - 0.4417) \times \left(\frac{1}{600} + \frac{1}{600}\right)} = \sqrt{0.4417 \times 0.5583 \times \frac{2}{600}} = \sqrt{0.0008207} = 0.02867$$

Substitute to the formula:

$$z = \frac{0.4 - 0.4833}{0.02867} = \frac{-0.0833}{0.02867} = -2.9055$$

p-value:

$$\text{p-value} = 2(\phi(|-2.91|) = 2(0.0018)$$

$$= 0.0036$$

Compare the z-statistic to the critical value
For a two-tailed test with $\alpha = 0.05$,:

$$\text{Z\_critical} \approx \pm 1.96$$

since:

$$-2.9055 < -1.96$$

or

$$0.0036 < 0.05$$

we reject the null hypothesis. therefore:
There is a statistically significant difference between the CTRs of Version A and Version B at the 5% significance level.

3. Meaning of p-value

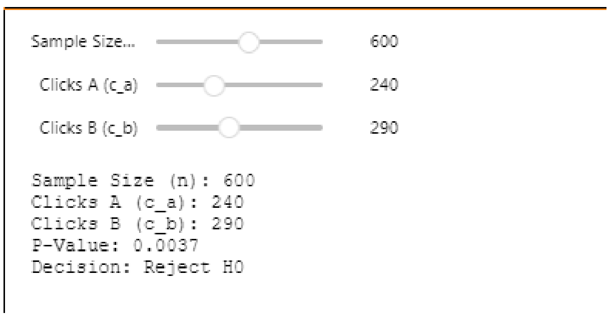
The p-value of 0.0036 indicates a 0.36% probability of observing a difference as extreme as or more extreme than the one observed if the null hypothesis was true.

4. Interpret

Version B of the website has a higher CTR of 48.33% unlike A with 40% this CTR is not due to random chance but they are statistically significant and therefore implementing or using version B of the website will lead to a high CTR.

5. Simulation

```
#Import necessary libraries
import numpy as np
import scipy.stats as stats
import ipywidgets as widgets
from IPython.display import display

# Function to perform hypothesis test
def hypothesis_test(n, c_a, c_b):
    p_a = c_a / n
    p_b = c_b / n
    p_pool = (c_a + c_b) / (2 * n)
    se = np.sqrt(2 * p_pool * (1 - p_pool) / n)
    z = (p_a - p_b) / se
    p_value = 2 * (1 - stats.norm.cdf(abs(z)))
    decision = "Reject H0" if p_value < 0.05 else "Fail to Reject H0"
    return p_value, decision

# Widgets for sample size and click-through rates
n_slider = widgets.IntSlider(value=100, min=100, max=1000, step=50, description='Samp
c_a_slider = widgets.IntSlider(value=50, min=50, max=600, step=10, description='Click
c_b_slider = widgets.IntSlider(value=50, min=50, max=600, step=10, description='Click
# Function to update the output
def update_output(n, c_a, c_b):
    p_value, decision = hypothesis_test(n, c_a, c_b)
    print(f"Sample Size (n): {n}")
    print(f"Clicks A (c_a): {c_a}")
    print(f"Clicks B (c_b): {c_b}")
    print(f"P-Value: {p_value:.4f}")
    print(f"Decision: {decision}")

# Interactive widget
widgets.interactive(update_output, n=n_slider, c_a=c_a_slider, c_b=c_b_slider)
```



```
Sample Size (n): 600
Clicks A (c_a): 240
Clicks B (c_b): 290
P-Value: 0.0037
Decision: Reject H0
```

4. Part 4: True Positives and False Positives in Medical Testing

1. Confusion Matrix

	Actual Positive (Disease)	**Actual Negative (No Disease)**
Predicted Positive	True Positive (TP = 80)	False Positive (FP = 10)
Predicted Negative	False Negative (FN = 10)	True Negative (TN = 900)

2. Sensitivity, specificity, PPV, NPV

Sensitivity (or Recall or True Positive Rate): The ability of the test to correctly identify those with the disease.

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{80}{80 + 10} = \frac{80}{90} = 0.8889$$

Sensitivity = 88.89%

Specificity (or True Negative Rate) The ability of the test to correctly identify those without the disease.

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{900}{900 + 10} = \frac{900}{910} = 0.989$$

Specificity = 98.9%

Positive Predictive Value (PPV): The probability that someone who tests positive actually has the disease.sensitivity

$$\text{PPV} = \frac{TP}{TP + FP} = \frac{80}{80 + 10} = \frac{80}{90} = 0.8889$$

PPV = 88.89%

Negative Predictive Value (NPV): The probability that someone who tests negative actually does not have the disease. Specificity

$$\text{NPV} = \frac{TN}{TN + FN} = \frac{900}{900 + 10} = \frac{900}{910} = 0.989$$

NPV = 98.9%

3. Reliability

High specificity and NPV indicate that the system is reliable in identifying true negatives and ruling out the disease however, for a disease detection system it is better to have a more sensitive system that is specific, the low sensitivity and PPV suggest that the system might miss some positive cases.

4. Increase in population

Increasing the population size with the same sensitivity specificity will lead to an increase in the proportionality of the true and false positives. Example:

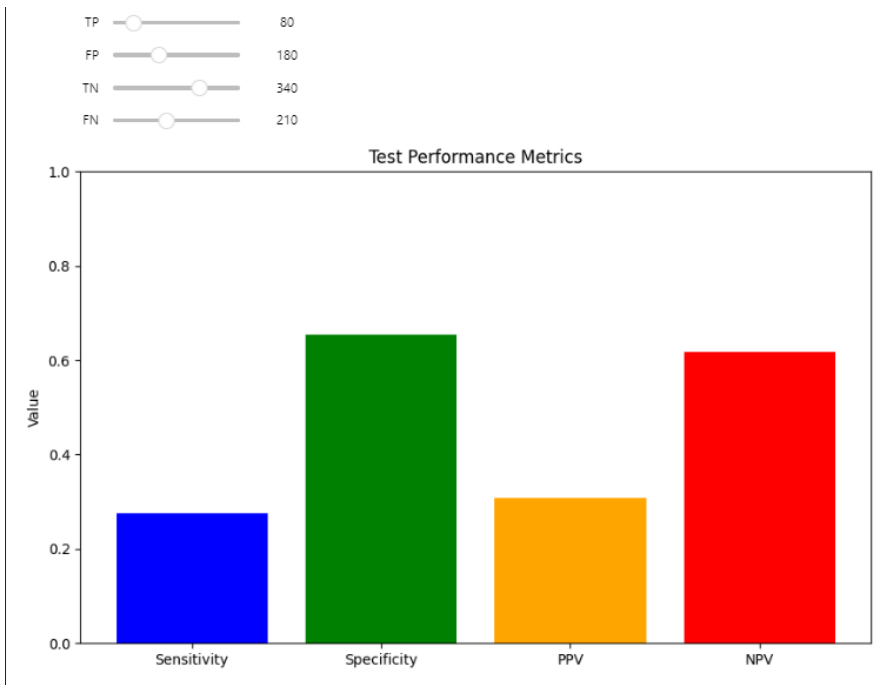
$$\text{True Positive} = 5000 \times \frac{80}{1000} = 400$$

$$\text{False positive} = 5000 \times \frac{10}{1000} = 50$$

5. Simulation

```
#Import libraries
import ipywidgets as widgets
import matplotlib.pyplot as plt
from IPython.display import display
import numpy as np

# Define the sliders for TP, FP, TN, and FN
tp_slider = widgets.IntSlider(value=0, min=0, max=500, step=10, description='TP')
fp_slider = widgets.IntSlider(value=0, min=0, max=500, step=10, description='FP')
tn_slider = widgets.IntSlider(value=0, min=0, max=500, step=10, description='TN')
fn_slider = widgets.IntSlider(value=0, min=0, max=500, step=10, description='FN')

# Function to compute metrics
def compute_metrics(tp, fp, tn, fn):
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
    ppv = tp / (tp + fp) if (tp + fp) > 0 else 0
    npv = tn / (tn + fn) if (tn + fn) > 0 else 0
    return sensitivity, specificity, ppv, npv
```

```
# Function to update the plot
def update_plot(tp, fp, tn, fn):
    sensitivity, specificity, ppv, npv = compute_metrics(tp, fp, tn, fn)

    metrics = [sensitivity, specificity, ppv, npv]
    labels = ['Sensitivity', 'Specificity', 'PPV', 'NPV']

    plt.figure(figsize=(10, 6))
    plt.bar(labels, metrics, color=['blue', 'green', 'orange', 'red'])
    plt.ylim(0, 1)
    plt.ylabel('Value')
    plt.title('Test Performance Metrics')
    plt.show()

# Create an interactive widget
interactive_plot = widgets.interactive(update_plot, tp=tp_slider, fp=fp_slider, tn=tn

# Display the widget
display(interactive_plot)
```



TP 80
FP 180
TN 340
FN 210

Part 5: MLE / MAP Estimation

1. Likelihood of given flips

Observation:

$$[1, 0, 1, 1, 1]$$

Counts:

$$\text{Number of heads} = 4$$
$$\text{Number of tails} = 1$$

The probability for each hypothesis H_i is given as p_i.
For each hypothesis H_i, we use the bernouli:

$$L(H_i) = p_i^{\text{number of heads}} \times (1 - p_i)^{\text{number of tails}}$$

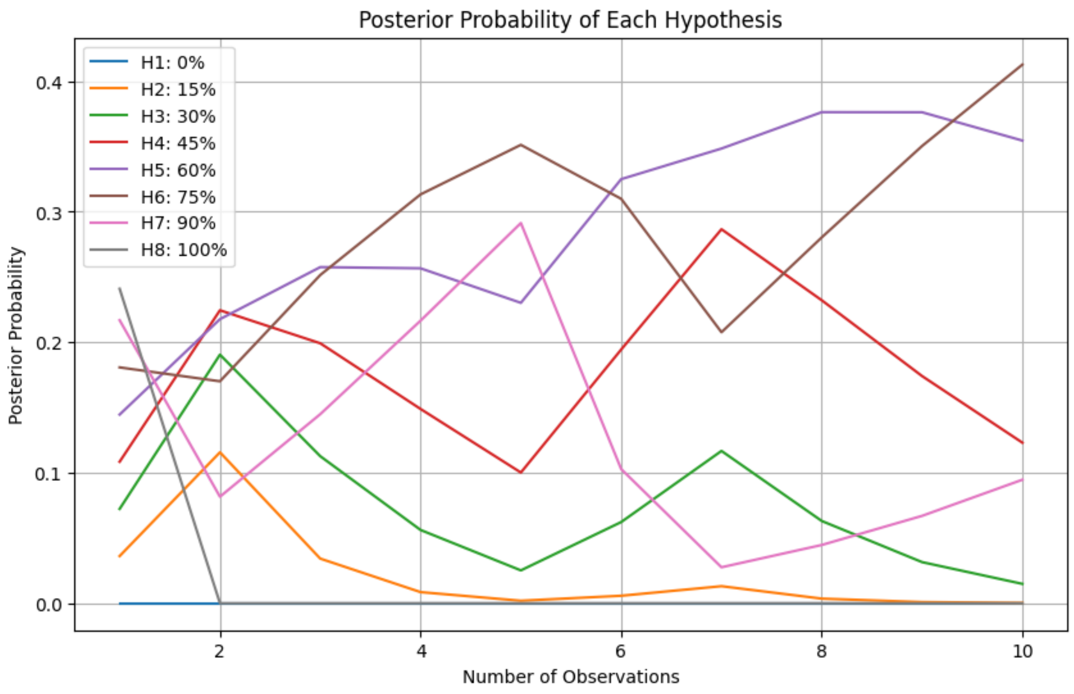
Hypothesis	Probability p_i	Likelihood $L(H_i)$
H_1	0	0
H_2	0.15	0.0004303
H_3	0.30	0.00567
H_4	0.45	0.02255
H_5	0.60	0.05184
H_6	0.75	0.07910
H_7	0.90	0.06561
H_8	1	0

2. MAP and MLE

a. Plot for posterior

```
#Import libraries
import numpy as np
import matplotlib.pyplot as plt

# Coin flip results
data = np.array([1, 0, 1, 1, 1, 0, 0, 1, 1, 1])

# Hypotheses (probabilities of heads)
hypotheses = np.array([0.0, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 1.0])

# Prior probabilities (uniform prior)
priors = np.ones(len(hypotheses)) / len(hypotheses)

# Function to compute likelihood
def likelihood(h, data):
    return np.prod(h**data * (1-h)**(1-data))

# Function to compute posterior
def posterior(hypotheses, priors, data):
    likelihoods = np.array([likelihood(h, data) for h in hypotheses])
    posteriors = likelihoods * priors
    return posteriors / np.sum(posteriors)

# Compute posteriors for each number of observations
```

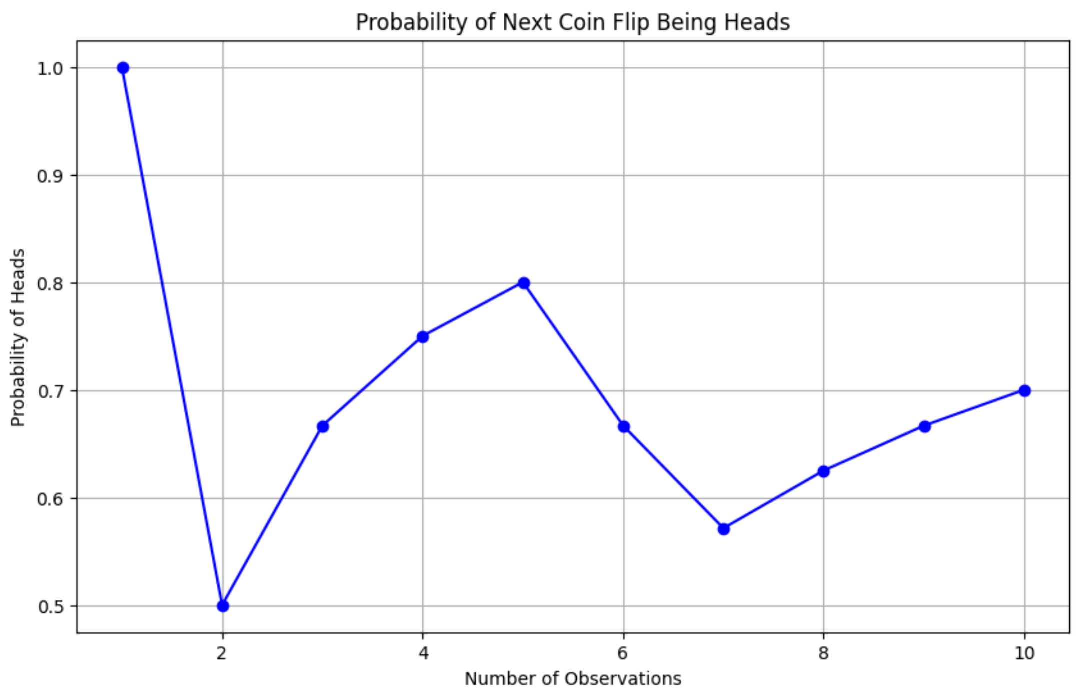
```
posteriors = []
for i in range(1, len(data) + 1):
    posteriors.append(posterior(hypotheses, priors, data[:i]))

# Convert to numpy array for easier plotting
posteriors = np.array(posteriors)

# Plotting
plt.figure(figsize=(10, 6))
for i, h in enumerate(hypotheses):
    plt.plot(range(1, len(data) + 1), posteriors[:, i], label=f'H{i+1}: {int(h*100)}%

plt.xlabel('Number of Observations')
plt.ylabel('Posterior Probability')
plt.title('Posterior Probability of Each Hypothesis')
plt.legend()
plt.grid(True)
plt.show()
```



b. Plot for heads

```
#Import libraries
import numpy as np
import matplotlib.pyplot as plt

# Coin flip results
data = np.array([1, 0, 1, 1, 1, 0, 0, 1, 1, 1])
```

```
# Function to compute the probability of heads
def probability_of_heads(data):
    return np.cumsum(data) / np.arange(1, len(data) + 1)

# Compute probabilities
probabilities = probability_of_heads(data)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(data) + 1), probabilities, marker='o', linestyle='-', color='b'
plt.xlabel('Number of Observations')
plt.ylabel('Probability of Heads')
plt.title('Probability of Next Coin Flip Being Heads')
plt.grid(True)
plt.show()
```



c. Likely hypothesis

The most likely hypothesis is **H6: 75.0%**

```
# Import libraries
import numpy as np

# Coin flip results
data = np.array([1, 0, 1, 1, 1, 0, 0, 1, 1, 1])

# Hypotheses (probabilities of heads)
hypotheses = np.array([0.0, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 1.0])
```

```
# Prior probabilities (uniform prior)
priors = np.ones(len(hypotheses)) / len(hypotheses)

# Function to compute likelihood
def likelihood(h, data):
    return np.prod(h**data * (1-h)**(1-data))

# Function to compute posterior
def posterior(hypotheses, priors, data):
    likelihoods = np.array([likelihood(h, data) for h in hypotheses])
    posteriors = likelihoods * priors
    return posteriors / np.sum(posteriors)

# Compute posterior probabilities after all observations
posteriors = posterior(hypotheses, priors, data)

# Find the most likely hypothesis
most_likely_hypothesis_index = np.argmax(posteriors)
most_likely_hypothesis = hypotheses[most_likely_hypothesis_index]

print(f"The most likely hypothesis is H{most_likely_hypothesis_index + 1}: {most_like
```

The most likely hypothesis is **H6: 75.0%**

Part 6: Students' Exam Performance

Given Data

- Group 1 (used cheat sheets):
  - Sample size (n_1) = 45
  - Mean ($\bar{X}_1$) = 88
  - Standard deviation (s_1) = 3
- Group 2 (did not use cheat sheets):
  - Sample size (n_2) = 55
  - Mean ($\bar{X}_2$) = 85
  - Standard deviation (s_2) = 2

1. Pooled standard deviation and Standard error

Pooled Standard Deviation (S_p)

$$S_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}}$$

Substitute the given values:

$$S_p = \sqrt{\frac{(45-1)(3^2)+(55-1)(2^2)}{45+55-2}}$$

$$S_p = \sqrt{\frac{44 \times 9 + 54 \times 4}{98}}$$

$$S_p = \sqrt{\frac{396+216}{98}}$$

$$S_p = \sqrt{\frac{612}{98}} \approx 2.499$$

$S_p \approx 2.499$.
Standard Error (SE)

$$SE = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Substitute:

$$SE = \sqrt{\frac{3^2}{45} + \frac{2^2}{55}} = \sqrt{0.2 + 0.0727} = \sqrt{0.2727} \approx 0.522$$

2. Z-score

$$z = \frac{\bar{x}_1 - \bar{x}_2}{SE}$$
$$= \frac{88-85}{0.522} = \frac{3}{0.522} \approx 5.745$$

z_critical for α 0.05 for one tail is:

$$\text{z\_critical} \approx 1.645$$

3. Null hypothesis and interpretation

Since:

$$5.745 > 1.645$$

- We will reject the null hypothesis.
- This implies that the students who used the cheat sheet performed better than those who didn't, and it's statistically significant that it is not due to randomness.

4. P- value

The z-score of 5.75 is hard to find in the table, so Social science statistics [1] were used.

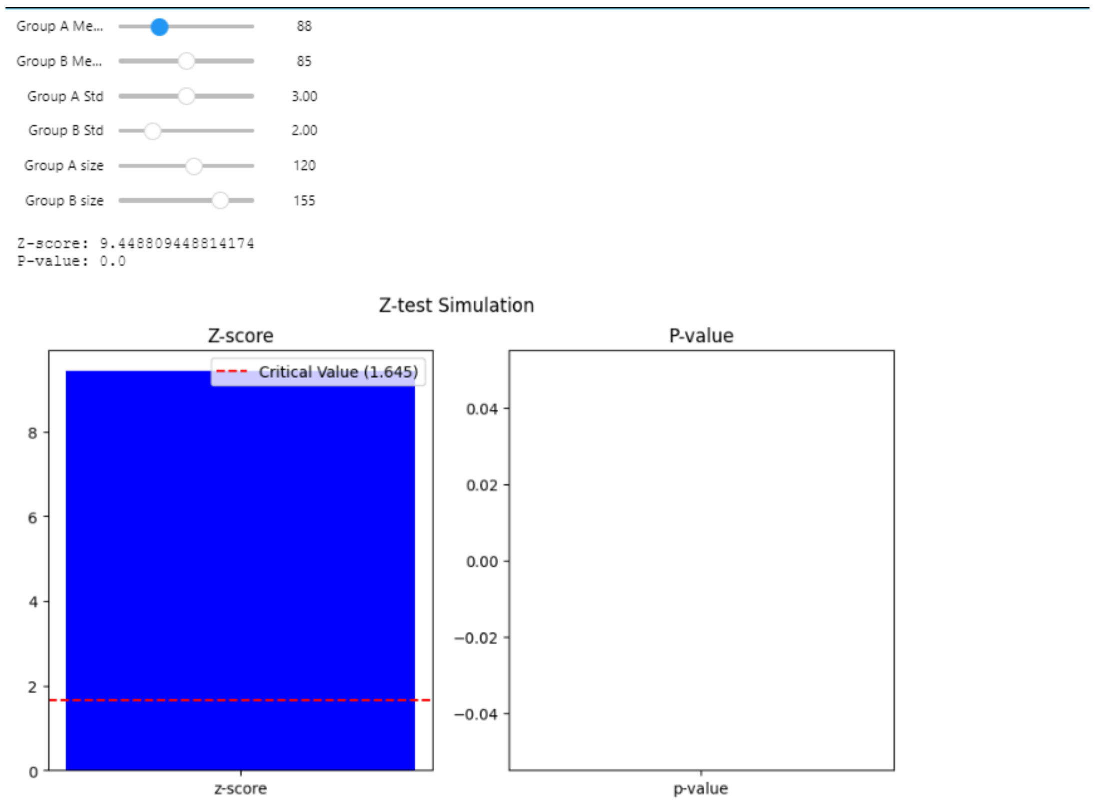
The P-Value is < .00001.

And since the p-value is much less than 0.05 it shows very strong evidence against the null hypothesis this indicates that the difference in the performance of the two groups those that use and don't use the cheat sheet is statistically significant.

5. Simulate

```
# Import libraries
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display

# Define the interactive widgets
muA_slider = widgets.IntSlider(value=90, min=85, max=95, step=1, description='Group A
muB_slider = widgets.IntSlider(value=85, min=80, max=90, step=1, description='Group B
sigmaA_slider = widgets.FloatSlider(value=2.5, min=1, max=5, step=0.1, description='G
sigmaB_slider = widgets.FloatSlider(value=2.5, min=1, max=5, step=0.1, description='G
nA_slider = widgets.IntSlider(value=100, min=20, max=200, step=5, description='Group
nB_slider = widgets.IntSlider(value=100, min=20, max=200, step=5, description='Group

# Perform simulation
def simulate(muA, muB, sigmaA, sigmaB, nA, nB):
    # pooled std
    sp = np.sqrt(((nA - 1) * sigmaA**2 + (nB - 1) * sigmaB**2) / (nA + nB - 2))
    #  standard error
    se = np.sqrt(sigmaA**2/nA + sigmaB**2/nB)
    # z-score
    z = (muA - muB) / se
    # p-value
    p_value = 1 - stats.norm.cdf(z)

    # Print z-score and p-value
    print(f"Z-score: {z}")
    print(f"P-value: {p_value}")

    # Plots
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.bar(['z-score'], [z], color='blue')
    plt.axhline(y=1.645, color='r', linestyle='--', label='Critical Value (1.645)')
    plt.title('Z-score')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.bar(['p-value'], [p_value], color='orange')
    plt.title('P-value')

    plt.suptitle('Z-test Simulation')
    plt.show()

# Create interactive plot
```

```
interactive_plot = widgets.interactive(simulate, muA=muA_slider, muB=muB_slider, sigm
display(interactive_plot)
```



```
Group A Me...   88
Group B Me...   85
Group A Std     3.00
Group B Std     2.00
Group A size    120
Group B size    155

Z-score: 9.448809448814174
P-value: 0.0
```

a. Z- statistics and p-value, their indication

Z-score: 5.744562646538029

P-value: $4.60794358225769e^{-09}$

This indicates that the performance of the student using the cheat sheet is statistically significant and provides strong evidence against the null hypothesis.

b. Increase of the mean of group A and its effect

Increasing the mean to 94:

Z-score: 17.233687939614086

P-value: 0.0

Increasing the mean score for students with cheat sheets increases the z-statistic and decreases the p-value, indicating a more significant difference.

c. Impact of changing standard deviation for Group A

Increasing the standard deviation to 4.30:

Z-score: 4.313907730596443

P-value: $8.019695508121316e - 06$

Increasing the standard deviation for students using cheat sheets increases variability, this leads to a decrease in the z-statistic and an increase in the p-value, which indicates an inconsistency in performance.

d. Standard deviation on both

When changing the standard deviation for both, the group with a higher variability has a more significant impact on the z-test, however, when the standard deviation for both is increased it reduces the z-statistics.

e. Increase of the number and effect on statistics

Increasing the sample size to 120 and 155 respectively we get:

Z-score: 9.448809448814174

P-value: 0.0

Therefore increasing the number of students generally will give a more reliable test and since this will lead to the reduction of the standard error, it leads to the increase of the z-statistics.

f. Impact of unequal number, effect on reliability

Having an unequal number of students will affect the reliability of the output since this will mostly be accounted for when calculating the pooled standard deviations. A larger difference additionally will lead to skewed results.
The test will not be reliable unless more information is shared on the reason for using the unbalanced number.