Exp 1:

**🔹 What is Fisher’s Lemma?**

**Fisher’s Lemma (or Fisher’s Theorem)** says that:

For a *normal distribution*, the **sample mean** and the **sample variance** are *independent* of each other.

In simple words:

* If we take many samples from a **normal distribution**,  
  the **average value (mean)** and **spread (variance)** of each sample **do not affect each other**.
* That means, if the mean changes, the variance doesn’t necessarily change, and vice versa.

We can check (verify) this using **simulation** — by generating many random samples from a normal distribution.

**🔹 What You Have to Do in This Lab**

You have to:

1. **Generate random samples** from a normal distribution (using rnorm()).
2. **Calculate** the sample **mean** and **variance** for each sample.
3. **Repeat** this process many times (e.g., 1000 times).
4. **Check** if the mean and variance are **independent** — by finding their **correlation**.
5. **Visualize** the results with histograms, scatter plot, and Q-Q plot.

If the correlation between mean and variance is close to **0**,  
then they are **independent**, which **verifies Fisher’s Lemma** ✅

**🔹 Step-by-Step Explanation of the R Code**

set.seed(123)

mu <- 5; sigma <- 2; n <- 30; N\_sim <- 1000

* set.seed(123) makes random results repeatable.
* mu = population mean (5)
* sigma = population standard deviation (2)
* n = number of observations in one sample (30)
* N\_sim = number of samples (1000 times)

res <- replicate(N\_sim, {

x <- rnorm(n, mu, sigma)

c(mean(x), var(x))

})

* rnorm(n, mu, sigma) generates **30 random values** from a normal distribution.
* For each sample, it calculates:
  + mean(x) → sample mean
  + var(x) → sample variance
* replicate() repeats this **1000 times** and stores the results.

sample\_means <- res[1, ]

sample\_vars <- res[2, ]

This separates the results:

* sample\_means → list of all 1000 means
* sample\_vars → list of all 1000 variances

cat("Correlation between sample mean and variance:", cor(sample\_means, sample\_vars), "\n")

* Finds the **correlation** between the 1000 sample means and variances.
* If the correlation ≈ **0**, they are independent → **Fisher’s Lemma verified** ✅

**Now the Plots**

par(mfrow = c(2,2))

* This divides the plot window into **4 parts** (2 rows × 2 columns).

**1️⃣ Histogram of Sample Means**

hist(sample\_means, col="lightblue", main="Sample Means", xlab="Mean", border="white")

curve(dnorm(x, mu, sigma/sqrt(n)), add=TRUE, col="red", lwd=2)

* Shows how the **sample means** are distributed.
* The red curve shows the **theoretical normal curve** for comparison.

**2️⃣ Histogram of Sample Variances**

hist(sample\_vars, col="lightgreen", main="Sample Variances", xlab="Variance", border="white")

* Shows the distribution of **sample variances**.

**3️⃣ Scatter Plot: Mean vs Variance**

plot(sample\_means, sample\_vars, pch=19, col=rgb(0,0,1,0.5),

main="Mean vs Variance", xlab="Mean", ylab="Variance")

abline(h=sigma^2, col="red", lwd=2)

* Each point represents one sample.
* If there’s **no visible pattern or trend**, it means mean and variance are **independent**.

**4️⃣ Q-Q Plot of Means**

qqnorm(sample\_means, col="blue", main="Q-Q Plot (Means)")

qqline(sample\_means, col="red", lwd=2)

* Checks if the **sample means** follow a **normal distribution**.
* Points close to the red line → distribution is normal.

**✅ What You’ll Conclude in the Lab Report**

After running the code:

* You’ll get a **correlation value** (e.g., around 0 or very small).
* This shows that **sample mean and variance are independent**.
* Therefore, **Fisher’s Lemma is verified** using simulation.

Exp 2

**What is Chi-Square (χ²) Distribution?**

The **Chi-Square distribution** is a type of probability distribution that is **used in statistics** — especially in **hypothesis testing** and **variance estimation**.

It is obtained when you **sum the squares of k independent standard normal random variables** (where k = degrees of freedom).

👉 For example,  
if Z1,Z2,...,ZkZ\_1, Z\_2, ..., Z\_kZ1​,Z2​,...,Zk​ are independent standard normal variables,  
then

X=Z12+Z22+...+Zk2X = Z\_1^2 + Z\_2^2 + ... + Z\_k^2X=Z12​+Z22​+...+Zk2​

follows a **Chi-Square distribution** with **k degrees of freedom**.

**🔹 What You Have to Do in This Lab**

You have to:

1. **Generate data** that follows a Chi-Square distribution.
2. **Calculate** its mean and variance.
3. **Compare** them with the **theoretical values**.
4. **Plot** and visually analyze the distribution using histogram, density plot, and Q-Q plot.

This helps you **understand the shape and behavior** of Chi-Square data.

**🔹 Step-by-Step Explanation of the R Code**

# Parameters

k <- 5

N\_sim <- 1000

* k → degrees of freedom (5 in this case).
* N\_sim → number of simulated data points (1000).

# Generate chi-squared data

set.seed(123)

chi2\_data <- rchisq(N\_sim, df = k)

* rchisq() generates **random numbers** that follow a Chi-Square distribution.
* df = k → means 5 degrees of freedom.
* set.seed(123) ensures the same random values are generated each time for reproducibility.

So now, chi2\_data contains 1000 values sampled from a χ²(5) distribution.

# Compute mean and variance

mean\_chi2 <- mean(chi2\_data)

var\_chi2 <- var(chi2\_data)

* Finds the **sample mean** and **sample variance** of the simulated data.

Theoretical formulas for χ² distribution:

Mean=k\text{Mean} = kMean=k Variance=2k\text{Variance} = 2kVariance=2k

So, for k = 5 →  
Expected Mean = 5  
Expected Variance = 10

You can check how close your simulated mean and variance are to these theoretical values.

**🔹 Graphical Analysis**

par(mfrow = c(3, 1))

* Divides the plot window into **3 rows** for 3 plots.

**1️⃣ Histogram**

hist(chi2\_data, breaks = 30, col = "lightblue", probability = TRUE,

main = "Chi-Squared Distribution", xlab = "Value", border = "white")

curve(dchisq(x, df = k), add = TRUE, col = "red", lwd = 2)

* Draws a **histogram** of the generated χ² data.
* probability = TRUE makes it a density histogram.
* The **red curve** (dchisq) shows the **theoretical Chi-Square probability density function**.
* If the histogram shape matches the red curve, the simulation is correct ✅

**2️⃣ Density Plot**

plot(density(chi2\_data), col = "blue", lwd = 2,

main = "Density Plot", xlab = "Value")

curve(dchisq(x, df = k), add = TRUE, col = "red", lwd = 2)

* This shows a **smooth estimate** of the probability density (blue line).
* The **red line** is again the theoretical curve.
* Both lines should be close to each other if your simulation is good.

**3️⃣ Q-Q Plot**

qqplot(qchisq(ppoints(N\_sim), df = k), chi2\_data,

main = "Q-Q Plot for Chi-Squared Data", col = "blue",

xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")

abline(0, 1, col = "red", lwd = 2)

* Q-Q (Quantile-Quantile) plot checks whether your data actually follows a χ² distribution.
* Points close to the red line → your simulated data **fits the theoretical distribution well** ✅

**🔹 What You’ll Conclude in the Lab**

* The **mean** and **variance** from the simulation are **close** to the theoretical values (mean ≈ k, variance ≈ 2k).
* The **histogram** and **density plot** match the theoretical χ² curve.
* The **Q-Q plot** shows points close to the straight line.

✅ Therefore, your generated data **follows the Chi-Square distribution** correctly.

Exp 3

**What Is the t-Distribution?**

The **t-distribution (Student’s t-distribution)** is very similar to the **normal distribution**,  
but it is used when:

* The **sample size is small** (n < 30), and
* The **population variance is unknown**.

It has **heavier tails** (wider spread) than the normal distribution.  
That means, with a small sample, there’s **more chance of extreme values**.

As the sample size increases, the t-distribution **approaches the normal distribution**.

**🔹 What You Have to Do in This Lab**

In this experiment, you have to:

1. Generate data following a **t-distribution** (small sample size, n = 10).
2. Generate data following a **normal distribution**.
3. Compare both distributions **graphically**:
   * Using histograms
   * Using density plots
   * Using Q-Q plot
4. Observe the difference in their **shape** and **spread**.

**🔹 Step-by-Step Explanation of the Code**

n <- 10

N\_sim <- 1000

* n = 10 → sample size (small sample).
* N\_sim = 1000 → number of random values to simulate.

set.seed(123)

t\_data <- rt(N\_sim, df = n-1)

normal\_data <- rnorm(N\_sim)

* set.seed(123) → makes random results reproducible.
* rt(N\_sim, df = n-1) → generates 1000 random values from a **t-distribution** with degrees of freedom = n − 1 = 9.
* rnorm(N\_sim) → generates 1000 random values from a **standard normal distribution** (mean = 0, sd = 1).

So now we have:

* t\_data → simulated t-distribution values
* normal\_data → simulated normal distribution values

par(mfrow = c(2, 2))

* Splits the plot window into 2 rows × 2 columns (4 plots).

**1️⃣ Histogram of t-distribution**

hist(t\_data, breaks = 30, col = "lightblue", probability = TRUE, main = "t-Distribution", xlab = "Value", border = "white")

curve(dt(x, df = n-1), add = TRUE, col = "red", lwd = 2)

* Shows the **shape of the t-distribution**.
* The **red curve** overlays the theoretical t-distribution curve.
* You will see it has **wider tails** (more spread out) than normal.

**2️⃣ Histogram of normal distribution**

hist(normal\_data, breaks = 30, col = "lightgreen", probability = TRUE, main = "Normal Distribution", xlab = "Value", border = "white")

curve(dnorm(x), add = TRUE, col = "blue", lwd = 2)

* Shows the **shape of a standard normal distribution**.
* The **blue curve** is the theoretical normal probability curve.
* You will see it looks **narrower** than the t-distribution.

**3️⃣ Density Plot Comparison**

plot(density(t\_data), col = "red", lwd = 2, main = "Density Comparison", xlab = "Value", ylim = c(0, 0.4))

lines(density(normal\_data), col = "blue", lwd = 2)

legend("topright", legend = c("t-Distribution", "Normal Distribution"), col = c("red", "blue"), lwd = 2)

* This compares both distributions in the **same graph**.
* The **red line** = t-distribution  
  The **blue line** = normal distribution
* You will notice:
  + Both are centered around 0 (same mean),
  + But **t-distribution** is **wider and shorter**, showing heavier tails.

**4️⃣ Q-Q Plot for t-distribution**

qqplot(qt(ppoints(N\_sim), df = n-1), t\_data, main = "Q-Q Plot for t-Distribution", col = "red", xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")

abline(0, 1, col = "blue", lwd = 2)

* Q-Q plot checks if the simulated t-data fits the theoretical t-distribution.
* Points close to the blue line → data follows t-distribution properly.

**🔹 What You’ll Observe and Conclude**

* For **small sample size (n = 10)**:
  + The **t-distribution** is **wider** and has **thicker tails** than the **normal distribution**.
  + It gives **more probability** to extreme values.
* As n increases, the t-distribution **becomes more similar** to the normal one.
* The plots clearly show this difference.

✅ **Conclusion:**  
For small samples, the **t-distribution differs from the normal** — it is more spread out and accounts for uncertainty in estimating the population variance.  
When sample size is large, **both distributions become almost identical**.

**Lab 4**  
**Simulation of F-Distributed Data and Its Relationship with Chi-Square (χ²) Distributions.”**

The **F-distribution** is a probability distribution that arises when we compare **two sample variances**.  
It is mainly used in:

* **ANOVA (Analysis of Variance)**
* **Regression analysis**
* **Variance ratio tests**

Mathematically, if:

and both are independent,  
then the random variable

F=(X1/df1)(X2/df2)F = \frac{(X\_1 / df\_1)}{(X\_2 / df\_2)}F=(X2​/df2​)(X1​/df1​)​

follows an **F-distribution** with **df₁** and **df₂** degrees of freedom.

**🔹 What You Have to Do in This Lab**

You must:

1. **Generate two chi-square distributed datasets**, each with different degrees of freedom.
2. Use these two χ² datasets to form an **F-distributed dataset**.
3. **Analyze** its shape and verify it behaves like a true F-distribution by:
   * Plotting histogram
   * Plotting density
   * Creating Q-Q plot
   * Showing boxplot

This shows the **relationship between F-distribution and chi-square distributions.**

**🔹 Step-by-Step Explanation of the Code**

df1 <- 5

df2 <- 10

N\_sim <- 1000

* df1 = 5 → degrees of freedom for the first χ² distribution.
* df2 = 10 → degrees of freedom for the second χ² distribution.
* N\_sim = 1000 → number of simulated random samples.

set.seed(123)

chi2\_1 <- rchisq(N\_sim, df = df1)

chi2\_2 <- rchisq(N\_sim, df = df2)

* rchisq() generates random numbers from a **chi-square distribution**.
* Here:
  + chi2\_1 → χ² data with df = 5
  + chi2\_2 → χ² data with df = 10
* set.seed(123) ensures the same random values every time you run it.

f\_data <- (chi2\_1 / df1) / (chi2\_2 / df2)

💡 This is the key step.  
It uses the formula:

F=(X1/df1)(X2/df2)F = \frac{(X\_1 / df\_1)}{(X\_2 / df\_2)}F=(X2​/df2​)(X1​/df1​)​

Thus, f\_data is your **F-distributed dataset**, built using two chi-square variables.

par(mfrow = c(2, 2))

* Splits the plotting window into 4 parts (2 rows × 2 columns).

**1️⃣ Histogram of F-distribution**

hist(f\_data, breaks = 30, col = "lightblue", probability = TRUE, main = "F-Distribution", xlab = "Value", border = "white")

curve(df(x, df1 = df1, df2 = df2), add = TRUE, col = "red", lwd = 2)

* Draws the **histogram** of your simulated F-data.
* The **red curve** adds the **theoretical F-distribution** for comparison.  
  ✅ If the histogram matches the curve shape, simulation is correct.

**2️⃣ Density Plot**

plot(density(f\_data), col = "blue", lwd = 2, main = "Density Plot", xlab = "Value")

curve(df(x, df1 = df1, df2 = df2), add = TRUE, col = "red", lwd = 2)

* The **blue line** = simulated F-data density.
* The **red line** = theoretical F-distribution density.  
  ✅ Both lines should overlap closely.

**3️⃣ Q-Q Plot**

qqplot(qf(ppoints(N\_sim), df1 = df1, df2 = df2), f\_data,

main = "Q-Q Plot for F-Distribution", col = "blue",

xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")

abline(0, 1, col = "red", lwd = 2)

* Compares your simulated data (f\_data) to the **theoretical F-distribution**.  
  ✅ If points follow the red line → data fits F-distribution correctly.

**4️⃣ Boxplot**

boxplot(f\_data, col = "lightgreen", main = "Boxplot of F-Distributed Data", ylab = "Value")

* Shows **data spread, median, and outliers**.
* Since F-distribution is right-skewed (not symmetric),  
  you will see a longer tail on the higher side.

**🔹 What You’ll Observe and Conclude**

* The **histogram and density plots** match the theoretical F curve.
* The **Q-Q plot** points lie close to the 45° line → confirms correct F-distribution.
* The **boxplot** shows the right-skewed nature of F-distribution.
* Thus, you successfully generated F-distributed data using **two χ²-distributions**.

✅ **Conclusion:**  
The experiment verifies that an **F-distribution** can be formed as the **ratio of two independent chi-square distributions divided by their degrees of freedom.**  
This simulation confirms the theoretical relationship between **F** and **χ²** distributions.

**Lab 5**

**Distribution of Medians and Ranges from Sampled Populations.”**  
Let’s go through it **step-by-step in very simple language**, so you can easily understand the meaning and what exactly this R code does.

**🔹 What the Lab Is About**

This lab shows how **sample statistics** like **median** and **range** behave when you take **many random samples** from the same population.

You’ll learn:

* What the **distribution of sample medians** looks like.
* What the **distribution of sample ranges** looks like.
* How these sample statistics vary when repeatedly sampling from a **normal population**.

**🔹 Key Ideas Before the Code**

1. **Population**: Here, we assume a **normal population** with
   * Mean (μ) = 0
   * Standard deviation (σ) = 1
2. **Sample size**: n = 20 → each sample contains 20 observations.
3. **Number of simulations**: N\_sim = 1000 → we will take 1000 such samples.
4. For each sample, we will:
   * Compute its **median** (middle value).
   * Compute its **range** = (maximum − minimum).

This helps us see how **medians** and **ranges** vary across repeated samples.

**🔹 Step-by-Step Explanation of the Code**

mu <- 0

sigma <- 1

n <- 20

N\_sim <- 1000

* mu = population mean (0)
* sigma = population standard deviation (1)
* n = sample size (20 observations per sample)
* N\_sim = total number of samples (1000)

medians <- numeric(N\_sim)

ranges <- numeric(N\_sim)

* Creates empty vectors to store **1000 median values** and **1000 range values**.

**🔹 Simulation Loop**

set.seed(123)

for (i in 1:N\_sim) {

data <- rnorm(n, mean = mu, sd = sigma)

medians[i] <- median(data)

ranges[i] <- max(data) - min(data)

}

Let’s explain each part clearly:

* rnorm(n, mean = mu, sd = sigma) generates one random sample of 20 values from a **normal distribution**.
* median(data) → finds the median of that sample.
* max(data) - min(data) → finds the range of that sample.
* This repeats **1000 times**, storing all medians and ranges in two vectors.

So, at the end:

* medians contains 1000 median values.
* ranges contains 1000 range values.

par(mfrow = c(2, 2))

* Divides the plotting area into **4 parts** (2 rows × 2 columns).

**1️⃣ Histogram of Medians**

hist(medians, breaks = 30, col = "lightblue", probability = TRUE, main = "Distribution of Medians", xlab = "Median", border = "white")

curve(dnorm(x, mean = mu, sd = sigma/sqrt(n)), add = TRUE, col = "red", lwd = 2)

* Shows how the **sample medians** are distributed.
* The **red curve** is the theoretical normal distribution for the sample mean (μ, σ/√n).  
  → This comparison shows that **sample medians are roughly normal** and centered near μ = 0.

**2️⃣ Histogram of Ranges**

hist(ranges, breaks = 30, col = "lightgreen", probability = TRUE, main = "Distribution of Ranges", xlab = "Range", border = "white")

* Shows how **sample ranges** are distributed.
* You’ll notice it’s **not symmetric** — it’s usually **right-skewed**, since range values can’t be negative.

**3️⃣ Density Plot of Medians**

plot(density(medians), col = "blue", lwd = 2, main = "Density Plot of Medians", xlab = "Median")

curve(dnorm(x, mean = mu, sd = sigma/sqrt(n)), add = TRUE, col = "red", lwd = 2)

* The **blue line** shows the smoothed shape of sample medians.
* The **red line** again shows the theoretical normal curve.  
  ✅ Both lines should match closely, meaning medians from normal data are **approximately normally distributed**.

**4️⃣ Boxplot of Medians and Ranges**

boxplot(list(Medians = medians, Ranges = ranges), col = c("lightblue", "lightgreen"), main = "Boxplot")

* Displays **summary and spread** of medians and ranges.
* The **median box** will be centered around 0.
* The **range box** will show higher positive values and some variability.

**🔹 What You’ll Observe and Conclude**

* The **distribution of medians** looks **approximately normal**, centered near the population mean (μ = 0).
* The **distribution of ranges** is **right-skewed** (not normal) because the range cannot be negative.
* The **boxplot** shows that medians have smaller variation than ranges.

✅ **Conclusion:**  
When random samples are drawn from a normal population:

* The **sample medians** follow an approximately **normal distribution**.
* The **sample ranges** show a **skewed distribution**.  
  This experiment demonstrates how **different sample statistics (median and range)** behave across repeated sampling.

**Lab 6**

**Lab 6: Estimate Population Parameters (Mean, Variance) from Sample Data**.  
Let’s break it down simply so you can **understand each part easily** and also know what this experiment shows.

**🔹 Objective**

To **estimate the population mean and variance** from a sample of data and construct **confidence intervals (CIs)** for both parameters.

**🔹 Concept (Theory in Simple Words)**

When we collect a **sample** from a population, we don’t know the true values of:

* **Population mean (μ)**
* **Population variance (σ²)**

So, we use **sample statistics** to **estimate** them:

* **Sample mean (x̄)** → estimates μ
* **Sample variance (s²)** → estimates σ²

We can also calculate **confidence intervals (CI)** to show a range within which the true parameter likely lies

**What You Have to Do in This Lab**

You must:

1. **Generate a random sample dataset** from a normal distribution with a given population mean (μ) and standard deviation (σ).
2. **Calculate the sample mean and sample variance** — these act as **point estimates** of the population parameters.
3. **Construct 95% confidence intervals (CIs)** for both:
   * Population mean (using the **t-distribution**)
   * Population variance (using the **chi-square distribution**)
4. **Display the numerical results**, showing:
   * Estimated sample mean and variance
   * Lower and upper confidence limits for both parameters
5. **Visualize the results graphically** by:
   * Plotting a **histogram** of the sample data with vertical lines showing mean and CI limits
   * Drawing a **boxplot** to observe the spread and variation of the sample

**🔹 Step-by-Step Explanation of the R Code**

**1️⃣ Define Population Parameters**

mu <- 5

sigma <- 2

n <- 30

N\_sim <- 1000

* True population mean = 5
* True standard deviation = 2
* Sample size = 30
* 1000 = (for simulation purpose, though only one sample used here)

**2️⃣ Generate Sample Data**

set.seed(123)

sample\_data <- rnorm(n, mean = mu, sd = sigma)

* Generates **30 random values** from a **normal distribution** with mean = 5 and sd = 2.
* set.seed(123) ensures **reproducibility** (you get the same data every time).

**3️⃣ Calculate Point Estimates**

sample\_mean <- mean(sample\_data)

sample\_var <- var(sample\_data)

* sample\_mean → estimate of population mean.
* sample\_var → estimate of population variance.

**4️⃣ Calculate 95% Confidence Intervals**

**(a) For Mean**

conf\_int\_mean <- t.test(sample\_data)$conf.int

* Uses **t-distribution** (since population σ unknown)
* Returns the **lower and upper limits** of the 95% confidence interval for the mean.

**(b) For Variance**

conf\_int\_var <- c((n-1)\*sample\_var / qchisq(0.975, df = n-1),

(n-1)\*sample\_var / qchisq(0.025, df = n-1))

* Uses **chi-square distribution** to estimate CI for variance.

**5️⃣ Display Output**

print(paste("Sample Mean:", sample\_mean))

print(paste("Sample Variance:", sample\_var))

print(paste("95% CI for Mean:", conf\_int\_mean))

print(paste("95% CI for Variance:", conf\_int\_var))

* Shows:
  + Estimated mean and variance.
  + Confidence interval for both.

**6️⃣ Graphical Output**

**(a) Histogram with Mean and CI Lines**

hist(sample\_data, breaks = 30, col = "lightblue", main = "Sample Data with Mean", xlab = "Value", border = "white")

abline(v = sample\_mean, col = "red", lwd = 2)

abline(v = conf\_int\_mean, col = "blue", lty = 2)

* **Red line** → sample mean
* **Blue dashed lines** → 95% CI for mean  
  This helps you visually see where the mean lies within the data distribution.

**(b) Boxplot**

boxplot(sample\_data, col = "lightgreen", main = "Boxplot of Sample Data", ylab = "Value")

* Shows spread, median, and possible outliers of the sample data.

**🔹 Expected Output (Example)**

Sample Mean: 5.097

Sample Variance: 3.45

95% CI for Mean: (4.35, 5.84)

95% CI for Variance: (2.1, 6.1)

*(Values will vary slightly due to randomness)*

**🔹 Conclusion**

* The **sample mean and variance** are **good estimators** of population parameters.
* The **confidence interval** gives a **range of plausible values** for the true mean and variance.
* Most of the time, the true values (μ = 5, σ² = 4) will lie within the 95% confidence intervals.

**LAB 7**

**Objective**

To show that as the **sample size increases**, the **sample mean and its variance** get **closer to the true population mean and variance** — demonstrating the **consistency** of the estimators.

**Theory (Simple Explanation)**

* **Consistency** means an estimator (like sample mean) **approaches the true population parameter** as the sample size increases.
* With **small samples**, the sample mean can be far from the true mean.
* With **larger samples**, the sample mean **converges** to the true mean (μ), and the variance of the sample mean decreases.
* This experiment **simulates repeated sampling** for different sample sizes and visualizes convergence.

**Step-by-Step Explanation of the Code**

**1️⃣ Set Parameters**

mu <- 5; sigma <- 2; ns <- c(10, 30, 100, 500, 1000); N <- 1000

* mu = population mean = 5
* sigma = population standard deviation = 2
* ns = list of sample sizes to test (10, 30, 100, 500, 1000)
* N = number of simulations per sample size = 1000

**2️⃣ Run the Simulation Using sapply**

res <- sapply(ns, function(n) {

sm <- replicate(N, mean(rnorm(n, mu, sigma)))

c(mean(sm), var(sm))

})

* **Loop over sample sizes (ns)** using sapply.
* For each n:
  1. replicate(N, mean(rnorm(n, mu, sigma))) → generate **N random samples of size n** and calculate the **mean** of each sample.
  2. mean(sm) → average of these sample means (should approach μ).
  3. var(sm) → variance of these sample means (should decrease as n increases).
* The result res is a **2 × 5 matrix**:
  1. **Row 1:** mean of sample means for each n
  2. **Row 2:** variance of sample means for each n

**3️⃣ Graphical Output**

par(mfrow = c(1, 2))

* Divides the plotting area into 1 row × 2 columns.

**(a) Plot Sample Means vs Sample Size**

plot(ns, res[1, ], type = "b", col = "blue", main = "Convergence of Sample Mean", xlab = "Sample Size", ylab = "Sample Mean")

abline(h = mu, col = "red", lwd = 2)

* **Blue line:** mean of simulated sample means for each sample size.
* **Red line:** true population mean (μ = 5).
* Observation: As sample size increases, the blue line **gets closer to the red line**, showing **convergence of the sample mean**.

**(b) Plot Sample Variances vs Sample Size**

plot(ns, res[2, ], type = "b", col = "green", main = "Convergence of Sample Variance", xlab = "Sample Size", ylab = "Sample Variance")

abline(h = sigma^2, col = "red", lwd = 2)

* **Green line:** variance of simulated sample means for each n.
* **Red line:** true population variance (σ² = 4).
* Observation: As sample size increases, the green line **decreases** and **approaches the true variance**, showing **reduction in variability** of the sample mean.

**Steps Performed in Lab 7 (Code 2)**

1. **Set the population parameters**: defined the true mean (mu = 5) and standard deviation (sigma = 2).
2. **Defined different sample sizes** (ns = 10, 30, 100, 500, 1000) to test the effect of increasing sample size.
3. **Set the number of simulations** (N = 1000) for repeated sampling to observe variability.
4. **Generated repeated random samples** for each sample size using replicate() and calculated the **sample mean** for each repetition.
5. **Computed the mean of the sample means** for each sample size — this shows how the sample mean estimates the true population mean.
6. **Computed the variance of the sample means** for each sample size — this shows how variability decreases as sample size increases.
7. **Stored results in a matrix (res)**:
   * Row 1 → mean of sample means
   * Row 2 → variance of sample means
8. **Plotted the results**:
   * **Sample mean vs sample size**: shows convergence of the sample mean to the true population mean.
   * **Sample variance vs sample size**: shows reduction in variability and convergence toward true variance.
9. **Added reference lines** in the plots:
   * Red line for the **true population mean** (mu = 5)
   * Red line for the **true population variance** (sigma² = 4)
10. **Observed consistency**:
    * Sample mean **gets closer to true mean** as sample size increases.
    * Variance of sample mean **decreases** with larger sample sizes.

**LAB 8**

**Steps Performed in the Code**

1. **Set population parameters**: defined true mean (mu = 5) and standard deviation (sigma = 2).
2. **Defined sample size (n = 30)** and number of simulations (N = 1000).
3. **Generated simulated samples** of size 30 from a normal distribution using rnorm() and replicate().
4. **Computed unbiased sample variance** for each sample using var() (divides by n-1).
5. **Computed biased sample variance** for each sample using the formula dividing by n.
6. **Plotted histograms** of unbiased and biased variances with the true population variance (sigma^2 = 4) marked as a red line.
7. **Generated two independent groups** (g1 and g2) with different means and standard deviations.
8. **Calculated Z statistic manually** to compare the means of the two groups.
9. **Computed the two-sided p-value** for the Z-test using the standard normal distribution.
10. **Printed the Z-test result** (Z value and p-value).
11. **Plotted histograms of the two groups** to visualize the distributions and compare their means.
12. **Observed key concepts**:
    * Biased variance underestimates population variance.
    * Unbiased variance is correct on average.
    * Z-test shows difference between group means.

**Lab 9**

**Lab 9: Calculate Efficiency of Estimators (Sample Mean vs Sample Median)**

**Objective**

* To compare the **efficiency** of two estimators (sample mean and sample median) for estimating the population mean.
* Efficiency is measured as the **ratio of variances**:

Efficiency=Var(median)Var(mean)\text{Efficiency} = \frac{\text{Var(median)}}{\text{Var(mean)}}Efficiency=Var(mean)Var(median)​

* Lower variance → more efficient estimator.

**Step-by-Step Explanation**

1. **Set population parameters**:
   * Mean mu = 5
   * Standard deviation sigma = 2
   * Sample size n = 30
   * Number of simulations N\_sim = 1000
2. **Initialized vectors** to store simulated sample means and medians.
3. **Simulated 1000 samples** of size 30 from a normal distribution using rnorm().
4. **Calculated sample mean and sample median** for each simulated sample and stored them in vectors.
5. **Calculated the variance** of the sample means and sample medians.
6. **Calculated efficiency** as:
7. efficiency = var(medians) / var(means)
   * Efficiency < 1 → mean is more efficient than median.
   * Efficiency ≈ 1 → similar performance.
8. **Plotted histograms** of the sample means (light blue) and sample medians (light green) to visualize their distributions.

**Key Observations**

* Sample mean typically has **smaller variance** than sample median for normal data → **more efficient**.
* Histogram shows mean distribution is **tighter**, median is slightly wider.
* Efficiency value quantifies this difference numerically.

**Steps Performed (Short Version for Lab Report)**

1. Set population parameters: mean (mu), standard deviation (sigma), sample size (n), and simulations (N\_sim).
2. Simulated N\_sim samples of size n from a normal distribution.
3. Calculated sample mean and sample median for each sample.
4. Computed variances of sample means and medians.
5. Calculated efficiency as Var(median) / Var(mean).
6. Plotted histograms of sample means and medians for comparison.

**Lab 10**

**Derive MLEs for Binomial, Poisson, and Normal Distributions**

**Theory:**

* **MLE (Maximum Likelihood Estimation):**
  + It is a method used to **estimate the unknown parameters** of a probability distribution from sample data.
  + The basic idea: choose the parameter values that **make the observed data most likely**.
  + For example: if you toss a coin many times, the MLE of the probability of heads is just the **proportion of heads** in your sample.
* **For Different Distributions:**
  + **Binomial Distribution:**
    - Parameter: p (probability of success).
    - MLE formula: p\_mle = (sum of successes) / (number of trials)
  + **Poisson Distribution:**
    - Parameter: λ (average number of occurrences).
    - MLE formula: lambda\_mle = mean of the data
  + **Normal Distribution:**
    - Parameters: μ (mean) and σ (standard deviation).
    - MLE formulas:
      * mu\_mle = mean of the data
      * sigma\_mle = sqrt(mean of squared deviations from mean)

**Objective / Target of the Experiment:**

* To **derive and calculate MLEs** of the parameters of Binomial, Poisson, and Normal distributions from given or simulated sample data.
* To understand how **MLE works** and how it estimates parameters that best fit the observed data.

1. **Binomial Distribution:**
   * Set number of trials n\_binom = 20 and true probability p\_true = 0.6.
   * Generate 100 random Binomial data points.
   * Calculate MLE of p using formula: p\_mle = mean(data) / n.
2. **Poisson Distribution:**
   * Set true λ lambda\_true = 3.
   * Generate 100 random Poisson data points.
   * Calculate MLE of λ as the mean of the data: lambda\_mle = mean(data).
3. **Normal Distribution:**
   * Set true mean mu\_true = 5 and std deviation sigma\_true = 2.
   * Generate 100 random Normal data points.
   * Calculate MLE of μ as mean of data: mu\_mle = mean(data).
   * Calculate MLE of σ as sqrt of mean squared deviation: sigma\_mle = sqrt(mean((data - mu\_mle)^2)).
4. **Output:**
   * Print the MLE values of each distribution parameter.

**Lab 11**

**Objective:**

The objective of this experiment is to simulate the process of **statistical decision-making** using **hypothesis testing**. The aim is to determine whether there is enough evidence from a sample to reject a given null hypothesis (H₀) about the population mean in favor of the alternative hypothesis (H₁).

In simple words, we test whether the true mean of a population differs significantly from a hypothesized value.

**Theory:**

**Hypothesis testing** is a statistical method used to make decisions or inferences about population parameters using sample data.

* **Null Hypothesis (H₀):** Represents the default assumption. For example, H₀: μ = μ₀ (mean is equal to hypothesized mean).
* **Alternative Hypothesis (H₁):** Represents what we want to test. For example, H₁: μ > μ₀ (mean is greater than hypothesized mean).
* **Significance Level (α):** The probability of rejecting H₀ when it is actually true. Commonly α = 0.05.
* **Test Statistic:** A value computed from the sample data (e.g., t-value) used to decide whether to reject H₀.
* **P-value:** The probability of getting a test statistic at least as extreme as the one observed, assuming H₀ is true.  
  If **p-value < α**, we **reject H₀** (evidence supports H₁). Otherwise, we **fail to reject H₀**.

This experiment uses a **one-sample t-test** to compare the sample mean with a known population mean.

**Explanation of the Code:**

1. **Define parameters:**
2. mu0 <- 5 # Hypothesized mean (under H₀)
3. mu1 <- 6 # True mean (actual population mean)
4. sigma <- 2 # Standard deviation of population
5. n <- 30 # Sample size
6. alpha <- 0.05 # Significance level
7. **Generate sample data:**
8. set.seed(123)
9. sample\_data <- rnorm(n, mean = mu1, sd = sigma)

→ Generates random data (30 samples) from a normal distribution with mean = 6 and sd = 2.

1. **Perform t-test:**
2. t\_test\_result <- t.test(sample\_data, mu = mu0, alternative = "greater")

→ Performs a one-sample **right-tailed t-test** to check if the sample mean is significantly greater than 5.

1. **Decision-making:**
2. if (t\_test\_result$p.value < alpha) {
3. decision <- "Reject H0"
4. } else {
5. decision <- "Fail to reject H0"
6. }

→ Compares p-value with significance level α (0.05) to decide whether to reject H₀.

1. **Output results:**
2. print(paste("Test Statistic:", t\_test\_result$statistic))
3. print(paste("P-value:", t\_test\_result$p.value))
4. print(paste("Decision:", decision))

→ Displays calculated t-value, p-value, and the final decision.

1. **Graphical output:**
   * Histogram of sample data with red and blue lines for hypothesized and sample means.
   * Density plot showing the critical region and test distribution visually.

**What the Code Has Done (Stepwise Algorithm-like Points):**

1. Define known parameters (μ₀, μ₁, σ, n, α).
2. Generate a random sample using a normal distribution.
3. Perform a one-sample **t-test** to compare sample mean with hypothesized mean.
4. Calculate **p-value** and **test statistic (t-value)**.
5. Compare p-value with significance level α.
6. Print whether to **reject or fail to reject H₀**.
7. Plot histogram and density plot to visualize sample data and decision boundary.

**Lab 12**

**Derive the Best Critical Region for Simple vs. Composite Hypotheses**

**Objective:**

The objective of this experiment is to **derive and identify the best critical region** for testing **simple versus composite hypotheses** using the **Likelihood Ratio Test (LRT)**. The experiment aims to determine a decision boundary (critical region) that minimizes error probability when deciding between two hypotheses — the null (H₀) and the alternative (H₁).

**Theory:**

In hypothesis testing, a **critical region** is the range of values for which we reject the null hypothesis (H₀).

* **Simple Hypothesis:** Specifies the distribution completely (e.g., H₀: μ = 5).
* **Composite Hypothesis:** Does not specify the distribution completely (e.g., H₁: μ > 5).

The **Likelihood Ratio Test (LRT)** is used to derive the **best critical region**. It compares how likely the observed data are under H₀ versus H₁.  
The **likelihood ratio (λ)** is given by:

λ(x)=L(μ0)L(μ1)λ(x) = \frac{L(μ\_0)}{L(μ\_1)}λ(x)=L(μ1​)L(μ0​)​

where L(μ)L(μ)L(μ) is the likelihood function.

If λ(x)<kλ(x) < kλ(x)<k (a threshold value), we **reject H₀**.  
This ensures the test has the **maximum power** (probability of correctly rejecting H₀ when H₁ is true) for a given significance level (α).

The **critical region** is determined using the theoretical distribution (usually normal), and we use qnorm(1−α)qnorm(1 - α)qnorm(1−α) to find the cutoff value beyond which we reject H₀.

**Explanation of the Code:**

1. **Define parameters:**
2. mu0 <- 5
3. mu1 <- 6
4. sigma <- 2
5. n <- 30
6. alpha <- 0.05

→ Sets up the null and alternative means, standard deviation, sample size, and significance level.

1. **Generate sample data:**
2. set.seed(123)
3. sample\_data\_H0 <- rnorm(n, mean = mu0, sd = sigma)
4. sample\_data\_H1 <- rnorm(n, mean = mu1, sd = sigma)

→ Creates two samples: one assuming H₀ is true (mean = 5), and another assuming H₁ is true (mean = 6).

1. **Define likelihood ratio function:**
2. likelihood\_ratio <- function(data, mu0, mu1, sigma) {
3. exp(sum(dnorm(data, mean = mu1, sd = sigma, log = TRUE)) -
4. sum(dnorm(data, mean = mu0, sd = sigma, log = TRUE)))
5. }

→ Calculates the ratio of likelihoods (probabilities) of the data under H₁ and H₀.  
A higher ratio indicates data favor H₁.

1. **Find critical value:**
2. critical\_value <- qnorm(1 - alpha, mean = mu0, sd = sigma / sqrt(n))

→ Determines the cutoff value (boundary) for the test using the normal distribution.  
Values beyond this point belong to the **rejection region**.

1. **Make decision under H₀ and H₁:**
2. decision\_H0 <- mean(sample\_data\_H0) > critical\_value
3. decision\_H1 <- mean(sample\_data\_H1) > critical\_value

→ Checks if sample means fall inside or outside the critical region for both hypotheses.

1. **Display results:**
2. print(paste("Critical Value:", critical\_value))
3. print(paste("Decision under H0:", decision\_H0))
4. print(paste("Decision under H1:", decision\_H1))

→ Outputs the critical value and whether each sample leads to rejection of H₀.

1. **Graphical representation:**
   * Plots density under H₀ and H₁ distributions.
   * The red vertical line marks the **critical value**, showing the rejection region visually.

**What the Code Has Done (Stepwise Natural Algorithm):**

1. Define mean, standard deviation, sample size, and significance level.
2. Generate two sets of data: one under H₀ and another under H₁.
3. Define a likelihood ratio function to compare both hypotheses.
4. Compute the **critical value** for the chosen α using the normal distribution.
5. Check whether the sample mean from each dataset exceeds the critical value.
6. Print the critical value and decisions under both hypotheses.
7. Plot density graphs showing how data under H₀ and H₁ behave relative to the critical region.

**Lab 13**

**Simulate Type I and Type II Errors in Hypothesis Testing**

**Objective:**

The objective of this experiment is to **simulate and understand Type I and Type II errors** in hypothesis testing using repeated sampling. The experiment also estimates the **power of a statistical test**, which indicates how effectively the test can detect false null hypotheses.

**Theory:**

In hypothesis testing, we make decisions based on sample data, but errors can occur because samples may not perfectly represent the population.

* **Null Hypothesis (H₀):** The population mean equals a hypothesized value (μ = μ₀).
* **Alternative Hypothesis (H₁):** The population mean differs (or is greater/smaller) than μ₀.

There are two types of errors in hypothesis testing:

1. **Type I Error (α):** Rejecting H₀ when it is true.
   * Probability = Significance level (α).
   * Example: Concluding a new drug works when it actually doesn’t.
2. **Type II Error (β):** Failing to reject H₀ when it is false.
   * Probability = β.
   * Example: Concluding a new drug doesn’t work when it actually does.
3. **Power of the Test:**

Power=1−β

It represents the probability of correctly rejecting a false H₀.

**Steps:**

1. Define parameters for mean, standard deviation, sample size, significance level, and simulation count.
2. Initialize counters for Type I and Type II errors.
3. Run 1000 simulations.
4. For each iteration:
   * Generate sample under H₀ and perform t-test → count Type I errors.
   * Generate sample under H₁ and perform t-test → count Type II errors.
5. Compute the proportion of both errors.
6. Calculate test power (1 − Type II error rate).
7. Plot histograms of p-values under H₀ and H₁, showing rejection regions.

**Lab 14**

**Perform Hypothesis Testing Step-by-Step Using Real or Simulated Data**

**Objective:**

The objective of this experiment is to **perform hypothesis testing step-by-step** using simulated data. The goal is to demonstrate the complete procedure of hypothesis testing—from stating hypotheses to making a decision—using a **one-sample t-test**. This helps understand how decisions are made about population parameters using sample data.

**Theory:**

**Hypothesis testing** is a fundamental statistical method used to determine whether there is enough evidence to support a claim about a population parameter.

The general steps in hypothesis testing are:

1. **State the hypotheses:**
   * Null hypothesis (H₀): The sample mean equals the population mean (μ = μ₀).
   * Alternative hypothesis (H₁): The sample mean is greater (or less/different) than μ₀.
2. **Choose the significance level (α):** Usually set at 0.05.
3. **Compute the test statistic:**

t=xˉ−μ0s/nt = \frac{\bar{x} - μ\_0}{s / \sqrt{n}}t=s/n​xˉ−μ0​​

where xˉ\bar{x}xˉ is the sample mean, sss is the sample standard deviation, and nnn is the sample size.

1. **Find the critical value or p-value:** Compare test statistic with critical t-value or p-value.
2. **Decision:**
   * If t>tcriticalt > t\_{critical}t>tcritical​ or p<αp < αp<α: **Reject H₀**.
   * Otherwise: **Fail to reject H₀**.

This process tests whether the observed data provide strong enough evidence against the null hypothesis.

**What the Code Has Done (Stepwise Natural Algorithm):**

1. Define known parameters for hypothesis testing.
2. Generate a random sample assuming the population mean = 6.
3. Clearly state H₀ and H₁ for a right-tailed test.
4. Set the significance level α = 0.05.
5. Calculate the t-test statistic from sample data.
6. Compute critical t-value and corresponding p-value.
7. Compare the test statistic with the critical value to decide whether to reject H₀.
8. Display t-statistic, p-value, critical value, and decision.
9. Plot histogram and density graph to visualize test results and rejection region.

**Lab 15**

**Compare the Power of Different Tests for the Same Hypothesis**

**Objective:**

The objective of this experiment is to **compare the power of different statistical tests**—specifically, the **t-test** and **z-test**—for testing the same hypothesis about a population mean. The goal is to determine which test performs better in detecting a true difference between the sample mean and the hypothesized mean.

**Theory:**

**Power of a test** is the probability of correctly rejecting a false null hypothesis (H₀). It measures how effectively a test identifies real effects or differences.

For the same hypothesis, different tests may have different powers depending on sample size, variance, and test assumptions:

* **Z-test:** Used when the population standard deviation (σ) is known. It relies on the **standard normal distribution**.
* **T-test:** Used when σ is **unknown** and estimated from the sample. It uses the **t-distribution**, which is slightly wider for small samples.

As the **sample size increases**, the t-distribution becomes closer to the normal distribution, and both tests give similar results. The power of both tests also increases with larger sample sizes or greater differences between μ₀ and μ₁.

Mathematically,

Power=1−β\text{Power} = 1 - \betaPower=1−β

where **β** is the probability of a Type II error (failing to reject a false H₀).

**What the Code Has Done (Stepwise Natural Algorithm):**

1. Define parameters for both hypotheses and simulation settings.
2. Initialize counters for the number of successful rejections (power).
3. Run 1000 simulations generating data assuming H₁ is true.
4. Perform both t-test and z-test for each dataset.
5. Count how many times each test rejects H₀.
6. Compute the **power** as the proportion of correct rejections for both tests.
7. Display and compare power results.
8. Plot:
   * A bar chart comparing t-test and z-test power.
   * A line chart showing how t-test power changes with different sample sizes.