```
############ 1 ################################
#Exponential Distribution
set.seed(401)
true_lambda <- 0.5
true_mean <- 1 / true_lambda
sample_size <- 50
num_simulations <- 10000
sample_means <- replicate(num_simulations, {
  data_exp <- rexp(n = sample_size, rate = true_lambda)
  mean(data_exp)
})
mean_of_estimators_exp <- mean(sample_means)
median_of_estimators_exp <- median(sample_means)
get_mode <- function(x) {
  d <- density(x)
  d$x[which.max(d$y)]
}
mode_of_estimators_exp <- get_mode(sample_means)
print("--- Exponential Distribution Results (Target Parameter: Mean=2) ---")
cat("True Parameter Value (Mean):", true_mean, "\n")
cat("Mean of Sample Means (Check Unbiasedness):", mean_of_estimators_exp, "\n")
cat("Median
of
Sample
Means
(Check
```

```
Median-Unbiasedness):",
    median_of_estimators_exp, "\n")
cat("Mode of Sample Means (Check Modal-Unbiasedness):", mode_of_estimators_exp,
    "\n")
hist(sample_means, breaks=30, main="Sampling Distribution of the Sample Mean (Exp.)",
     xlab="Sample Mean (Estimator)", col="skyblue", border="white")
abline(v = true_mean, col = "red", lwd = 2, lty = 2) # True Mean
legend("topright", legend = "True Mean (2.0)", col = "red", lty = 2, lwd = 2)
#Uniform Distribution
set.seed(402)
true_alpha <- 0
true_beta <- 10
sample_size <- 50
num_simulations <- 10000
scaled_sample_maxima <- replicate(num_simulations, {
  data_unif <- runif(n = sample_size, min = true_alpha, max = true_beta)
  max(data_unif) * (sample_size + 1) / sample_size
})
mean_of_estimators_unif <- mean(scaled_sample_maxima)
median_of_estimators_unif <- median(scaled_sample_maxima)
mode_of_estimators_unif <- get_mode(scaled_sample_maxima)
print("--- Uniform Distribution Results (Target Parameter: Upper Limit Beta=10) ---")
cat("True Parameter Value (Beta):", true_beta, "\n")
cat("Mean of Scaled Sample Maxima (Check Unbiasedness):", mean_of_estimators_unif,
    "\n")
cat("Median
```

```
of
Scaled
Sample Maxima (Check Median-Unbiasedness):",
   median_of_estimators_unif, "\n")
cat("Mode
of
Scaled
Sample
Maxima (Check Modal-Unbiasedness):",
   mode_of_estimators_unif, "\n")
hist(scaled_sample_maxima, breaks=30, main="Sampling Distribution of the UMVUE for
Beta (Unif.)",
    xlab="Scaled Sample Maximum (Estimator)", col="lightcoral", border="white")
abline(v = true_beta, col = "blue", lwd = 2, lty = 2) # True Beta
legend("topright", legend = "True Beta (10.0)", col = "blue", lty = 2, lwd = 2)
hist(sample_means, breaks=30, main="Sampling Distribution of the Sample Mean (Exp.)",
    xlab="Sample Mean (Estimator)", col="skyblue", border="white")
abline(v = true_mean, col = "red", lwd = 2, lty = 2) # True Mean
legend("topright", legend = "True Mean (2.0)", col = "red", lty = 2, lwd = 2)
```

```
################ 2 ########################################

# Load dataset

data(iris)

head(iris)


# Select only numeric variables

X <- iris[, 1:4]


# 1. Simultaneous Estimation of Parameters

# Estimate mean vector and covariance matrix

mean_vector <- colMeans(X)

cov_matrix <- cov(X)


print("Mean Vector:")

print(mean_vector)


print("Covariance Matrix:")

print(cov_matrix)


# 2. Wilk’s Generalized Variance

# Determinant of covariance matrix

wilk_generalized_variance <- det(cov_matrix)

print(paste("Wilk’s Generalized Variance =", wilk_generalized_variance))
```

```
# 3.Ellipsoid of Concentration (for first two variables)
library(car)
dataEllipse(X$Sepal.Length, X$Sepal.Width,
        levels = 0.95,
        center.pch = 19,
        col = "blue",
        xlab = "Sepal Length",
        ylab = "Sepal Width",
        main = "Ellipsoid of Concentration (95% Confidence Region)")
```

```
############################  3 ########################################
# Load dataset
data(iris)

# Select univariate data
x <- iris$Sepal.Length
```

```
### 1. Bootstrap Method ###
library(boot)

# Define statistic function (mean)
boot_mean <- function(data, indices) {
  d <- data[indices]
  return(mean(d))
}

# Apply bootstrap with 1000 resamples
set.seed(123)
boot_result <- boot(data = x, statistic = boot_mean, R = 1000)

# Bootstrap estimates
boot_mean_est <- mean(boot_result$t)
boot_var_est <- var(boot_result$t)
boot_ci <- boot.ci(boot_result, type = "perc")

# Display results
cat("Bootstrap Mean Estimate =", boot_mean_est, "\n")
cat("Bootstrap Variance Estimate =", boot_var_est, "\n")
print(boot_ci)

### 2. Jackknife Method (Manual Implementation) ###

n <- length(x)
```

```
# Jackknife leave-one-out estimates
jack_values <- sapply(1:n, function(i) mean(x[-i]))

# Jackknife mean estimate
jack_mean_est <- mean(jack_values)

# Jackknife variance
jack_var_est <- (n - 1) / n * sum((jack_values - jack_mean_est)^2)

# Jackknife standard error
jack_se <- sqrt(jack_var_est)

# Jackknife 95% CI (normal approx.)
jack_ci <- mean(x) + c(-1.96, 1.96) * jack_se

cat("Jackknife Mean Estimate =", jack_mean_est, "\n")
cat("Jackknife Variance Estimate =", jack_var_est, "\n")
cat("Jackknife 95% CI =", jack_ci, "\n")

### 3. Comparison ###
cat("\n===== COMPARISON =====\n")
cat("Bootstrap Mean:", boot_mean_est, "\n")
cat("Jackknife Mean:", jack_mean_est, "\n")
cat("Bootstrap Variance:", boot_var_est, "\n")
cat("Jackknife Variance:", jack_var_est, "\n")
```

```
cat("Bootstrap CI (95%):", boot_ci$percent[4:5], "\n")
cat("Jackknife CI (95%):", jack_ci, "\n")
```

```
########################## 4 ################################
rm(list = ls())
data_vec <- mtcars$mpg
data_vec
n <- length(data_vec)
theta_hat <- mean(data_vec)
s2 <- var(data_vec)
sigma_hat <- sqrt(s2)
sigma_hat
#Empirical variance of sample mean
```

```
emp_var_mean <- s2 / n
emp_var_mean
#Chapman-Robbins-Kieffer lower bound
DKL <- function(h, n, sigma2) {
  n * h^2 / (2 * sigma2)
}
crk_fn <- function(h, n, sigma2) {
  h^2 / (2 * DKL(h, n, sigma2))
}
h_seq <- seq(1e-6, 4 * sigma_hat, length.out = 10000)
head(h_seq)
CRK_values <- crk_fn(h_seq, n, s2)
head(CRK_values)
Chapman_Robbins_Kieffer <- max(CRK_values)
Chapman_Robbins_Kieffer
h_opt_CRK <- h_seq[which.max(CRK_values)]
h_opt_CRK
#Cramer-Rao lower bound
score <- function(x, mu, sigma2) {
  (x - mu) / sigma2
}
I1 <- mean(score(data_vec, theta_hat, s2)^2)
In <- n * I1
CRLB <- 1 / In
CRLB
# Bhattacharryea lower bound
```

```
bhattacharyya_lb <- function(loglik, theta, data, h = 1e-6) {
  score <- sapply(data, function(x) {
    (loglik(x, theta + h) - loglik(x, theta - h)) / (2 * h)
  })
  second <- sapply(data, function(x) {
    (loglik(x, theta + h) - 2 * loglik(x, theta) + loglik(x, theta - h)) / (h^2)
  })
  I1 <- mean(score^2)
  I2 <- mean(second^2)
  BLB <- 1 / (I1 + 0.5 * I2)
  return(list(BLB = BLB, I1 = I1, I2 = I2))
}
loglik_normal <- function(x, mu) { -0.5 * log(2 * pi * s2) - (x - mu)^2 / (2 * s2)
}
#Compute Bhattacharyya Lower Bound
result <- bhattacharyya_lb(loglik = loglik_normal,
                           theta = theta_hat,
                           data = data_vec)
result
BLB_mean <- result$BLB / n
BLB_mean
#MVB
MVB <- s2 / n
MVB
bounds <- c(CRLB = CRLB,
            Bhattacharyya = BLB_mean,
```

```
        CRK = Chapman_Robbins_Kieffer,
        MVB = MVB)
cat("All bounds:\n")
print(bounds)
# Determine the tightest bound (largest value)
best_bound <- names(bounds)[which.max(bounds)]
best_bound
```

```
############################## 5 ###################################
library(MASS)
library(glmnet)

# Define MSE manually (since Metrics package doesn't exist)
mse <- function(actual, predicted) {
  mean((actual - predicted)^2)
}
```

```
# Load Boston dataset
data("Boston")

# Features and target
X <- Boston[, -14]      # remove medv column
y <- Boston$medv

# Train-test split (80–20)
set.seed(42)
train_indices <- sample(1:nrow(Boston), nrow(Boston) * 0.8)

X_train <- X[train_indices, ]
y_train <- y[train_indices]

X_test <- X[-train_indices, ]
y_test <- y[-train_indices]

### 1. OLS (Linear Regression)
ols_model <- lm(y_train ~ ., data = X_train)
y_pred_ols <- predict(ols_model, X_test)
mse_ols <- mse(y_test, y_pred_ols)

### 2. Ridge Regression (alpha = 0)
ridge_model <- glmnet(as.matrix(X_train), y_train, alpha = 0)
y_pred_ridge <- predict(ridge_model, as.matrix(X_test), s = 0.1)
mse_ridge <- mse(y_test, y_pred_ridge)
```

```
### 3. Lasso Regression (alpha = 1)
lasso_model <- glmnet(as.matrix(X_train), y_train, alpha = 1)
y_pred_lasso <- predict(lasso_model, as.matrix(X_test), s = 0.1)
mse_lasso <- mse(y_test, y_pred_lasso)

### 4. MLE (same as OLS in linear model with Gaussian errors)
mse_mle <- mse_ols

### Print results
cat("MSE of OLS: ", mse_ols, "\n")
cat("MSE of Ridge: ", mse_ridge, "\n")
cat("MSE of Lasso: ", mse_lasso, "\n")
cat("MSE of MLE (OLS equivalent): ", mse_mle, "\n")
```

################## 6 ######################################

```
set.seed(123); n <- 100
exp_d <- rexp(n, 2); unif_d <- runif(n, 2, 7)
pois_d <- rpois(n, 3); gamma_d <- rgamma(n, 2, 1/3)
pit_loc <- \(x) mean(x)
# Pitman estimator for location
pit_scale <- \(x) if(all(x>0)) mean(x) else sd(x)  # for scale
cat("Exponential scale:", pit_scale(exp_d),
    "\nUniform loc:", pit_loc(unif_d),
    "\nPoisson loc:", pit_loc(pois_d),
    "\nGamma scale:", pit_scale(gamma_d), "\n")
cat("\nUMVUE Check (Exponential):\n")
cat("Pitman =", pit_scale(exp_d),
    " UMVUE =", mean(exp_d),
    " Equal? ", abs(pit_scale(exp_d)-mean(exp_d))<1e-10, "\n")
```

```
############################## 7 ##########################################
set.seed(89)
x <- c(rnorm(95,50,10),150,200,180,160,190)
df <- data.frame(
  Estimator = c("Mean", "Median", "Trimmed", "Winsorized"),
  Estimate = c(mean(x), median(x),
               mean(x, trim = 0.1),
               mean(pmin(pmax(x, quantile(x, 0.1)), quantile(x, 0.9)))),
  Robustness = c("Low", "High", "Medium", "Medium")
)
print(df)
# Define the general and robust means
general_mean <- mean(x)
robust_mean  <- median(x)   # You can choose trimmed or winsorized if you prefer
boxplot(list(Clean = rnorm(95, 50, 10), Contaminated = x), main = "Effect of Outliers")
abline(h = c(general_mean, robust_mean), col = c("red", "blue"), lty = 2)
legend("topright", legend = c("Mean", "Median"), col = c("red", "blue"), lty = 2)
cat("Use robust estimators when outliers/heavy tails exist.\n\n")
print(df)
```

```
######################## 8  ########################################
set.seed(89)
n <- 30
sigma <- 1
mu_values <- seq(0, 1, by = 0.2)  # true means from 0 to 1
alpha <- 0.05
reps <- 1000  # number of simulations per mean
power_results <- data.frame(mu = mu_values,
                            MP = NA,
                            UMP = NA,
                            Unbiased = NA,
                            LUMPU = NA)
for (i in 1:length(mu_values)) {
  mu <- mu_values[i]
  reject_MP <- reject_UMP <- reject_Unbiased <- reject_LUMPU <- 0
  for (j in 1:reps) {
    x <- rnorm(n, mean = mu, sd = sigma)
    # (a) Most Powerful (LR test)
    L0 <- sum(dnorm(x, mean = 0, sd = sigma, log = TRUE))
```

```
    L1 <- sum(dnorm(x, mean = mu, sd = sigma, log = TRUE))
    LR <- 2 * (L1 - L0)
    if (LR > qchisq(0.95, 1)) reject_MP <- reject_MP + 1
    # (b) Uniformly Most Powerful (one-sided z-test)
    z <- (mean(x) - 0) / (sigma / sqrt(n))
    if (z > qnorm(0.95)) reject_UMP <- reject_UMP + 1
    # (c) Unbiased (two-sided t-test)
    t_res <- t.test(x, mu = 0, alternative = "two.sided")
    if (t_res$p.value < 0.05) reject_Unbiased <- reject_Unbiased + 1
    # (d) Locally Uniformly Most Powerful Unbiased (score test)
    z_lumpu <- (mean(x) - 0) / (sd(x) / sqrt(n))
    if (z_lumpu > qnorm(0.95)) reject_LUMPU <- reject_LUMPU + 1
  }
  power_results$MP[i] <- reject_MP / reps
  power_results$UMP[i] <- reject_UMP / reps
  power_results$Unbiased[i] <- reject_Unbiased / reps
  power_results$LUMPU[i] <- reject_LUMPU / reps
}
print(power_results)
power_results
# Reshape for ggplot
library(reshape2)
data_long <- melt(power_results, id.vars = "mu", variable.name = "Test", value.name =
                    "Power")
# Plot
library(ggplot2)
```

```
ggplot(data_long, aes(x = mu, y = Power, color = Test, group = Test)) +
  geom_line(linewidth = 0.7) +
  geom_point(size = 0.7) +
  labs(
    title = "Comparison of Test Powers",
    x = expression(mu),
    y = "Power"
  ) +
  theme_minimal(base_size = 14)
print(power_results)
power_results
```

```
############################## 9 ######################################
set.seed(123)
n <- 50
lambda <- 0.5
alpha=0.05
data <- rexp(n, rate = lambda)
data
```

```
S <- sum(data)
cdf <- function(s) pgamma(s, shape = n, rate = lambda)
c0 <- qgamma(alpha, shape = n, rate = lambda)
P_value <- cdf(c0)
P_eq <- dgamma(c0, shape = n, rate = lambda) * 0.0001
P_eq
gamma <- (alpha - P_value) / P_eq
gamma <- ifelse(gamma < 0, 0, gamma)
gamma
if (S < c0) {
  decision <- "Reject H0"
} else if (abs(S - c0) < 1e-6) {
  u <- runif(1)
  decision <- ifelse(u < gamma, "Reject H0 (randomized)", "Do not reject H0
(randomized)")
} else {
  decision <- "Do not reject H0"
}
cat("Observed S =", round(S,4), "\n")
cat("Critical value c0 =", round(c0,4), "\n")
cat("Decision:", decision, "\n")
cat("Randomization probability gamma =", round(gamma,4), "\n")
S <- sum(data)
S
# Find critical value under H0
c <- qgamma(alpha, shape = n, rate = lambda)
```

```
# Decision rule
if (S <= c) {
  cat("Reject H0\n")
} else {
  cat("Do not reject H0\n")
}
# Print results
cat("Observed S =", round(S, 4), "\n")
cat("Critical value =", round(c, 4), "\n")
#Define a range of true lambda values
lambda_true <- seq(0.2, 1.0, by = 0.05)
#Compute probabilities (Power = P(Reject H0))
#For exponential data, S ~ Gamma(shape=n, rate=lambda)
power <- pgamma(c0, shape = n, rate = lambda_true)
#OC curve = 1 - Power (probability of accepting H0)
OC <- 1 - power
#Plot OC Curve
plot(lambda_true, OC, type = "l", lwd = 2, col = "blue",
     main = "Operating Characteristic (OC) Curve",
     xlab = expression(True~lambda),
     ylab = "Probability of Accepting H0")
grid()
#Plot Power Curve
plot(lambda_true, power, type = "l", lwd = 2, col = "red",
     main = "Power Curve for Non-randomized LRT",
     xlab = expression(True~lambda),
```

```
    ylab = "Power (Probability of Rejecting H0)")
abline(h = alpha, lty = 2, col = "darkgray")
grid()
```

```
############### 10 ##########################
#################################################################
# PART 1 — OLS, Ridge, Lasso, MLE on the Boston Dataset
#################################################################

library(MASS)
library(glmnet)

# Define MSE function (since Metrics package does not exist)
mse <- function(actual, predicted) {
  mean((actual - predicted)^2)
```

```
}

# Load Boston dataset
data("Boston")

# Features and target
X <- Boston[, -14]     # remove medv column
y <- Boston$medv

# Train-test split 80/20
set.seed(42)
train_idx <- sample(1:nrow(Boston), nrow(Boston)*0.8)

X_train <- X[train_idx, ]
y_train <- y[train_idx]

X_test <- X[-train_idx, ]
y_test <- y[-train_idx]

### 1. OLS Regression (MLE under Gaussian assumptions)
ols_model <- lm(y_train ~ ., data = X_train)
y_pred_ols <- predict(ols_model, X_test)
mse_ols <- mse(y_test, y_pred_ols)

### 2. Ridge Regression
ridge_model <- glmnet(as.matrix(X_train), y_train, alpha = 0)
```

```
y_pred_ridge <- predict(ridge_model, as.matrix(X_test), s = 0.1)
mse_ridge <- mse(y_test, y_pred_ridge)

### 3. Lasso Regression
lasso_model <- glmnet(as.matrix(X_train), y_train, alpha = 1)
y_pred_lasso <- predict(lasso_model, as.matrix(X_test), s = 0.1)
mse_lasso <- mse(y_test, y_pred_lasso)

### 4. MLE (same as OLS)
mse_mle <- mse_ols

### Print results
cat("MSE of OLS: ", mse_ols, "\n")
cat("MSE of Ridge: ", mse_ridge, "\n")
cat("MSE of Lasso: ", mse_lasso, "\n")
cat("MSE of MLE (OLS equivalent): ", mse_mle, "\n")

################################################################
# PART 2 — Logistic Regression Tests using mtcars dataset
################################################################

library(lmtest)

# Load data
data(mtcars)
mtcars$am <- as.integer(mtcars$am)  # convert to 0/1
```

```
# Fit null (intercept-only) and full (with wt)
glm_null <- glm(am ~ 1, family = binomial(link="logit"), data=mtcars)
glm_full <- glm(am ~ wt, family = binomial(link="logit"), data=mtcars)

cat("\nSummary of full model (am ~ wt):\n")
print(summary(glm_full))

################################################################
# (i) Likelihood Ratio Test (nested models)
################################################################
lr_res <- lrtest(glm_full, glm_null)

cat("\nLikelihood Ratio Test (full vs null):\n")
print(lr_res)

# Generalized LR (same as deviance difference)
dev_diff <- as.numeric(2 * (logLik(glm_full) - logLik(glm_null)))
df_diff  <- attr(logLik(glm_full), "df") - attr(logLik(glm_null), "df")
p_glr <- pchisq(dev_diff, df=df_diff, lower.tail=FALSE)

cat("\nGeneralized Likelihood Ratio Test:\n")
cat("Deviance difference =", round(dev_diff,4),
    ", df =", df_diff,
    ", p-value =", signif(p_glr,4), "\n")
```

```
#################################################################
# (ii) Wald Test
#################################################################
wald_res <- waldtest(glm_full, glm_null)

cat("\nWald Test (full vs null):\n")
print(wald_res)

# Individual Wald test for coefficient of wt
coef_wt <- summary(glm_full)$coefficients["wt", "Estimate"]
se_wt   <- summary(glm_full)$coefficients["wt", "Std. Error"]
wald_stat_wt <- (coef_wt / se_wt)^2
p_wt <- pchisq(wald_stat_wt, df=1, lower.tail=FALSE)

cat("\nIndividual Wald Test for 'wt':\n")
cat("Wald statistic =", round(wald_stat_wt,4),
    ", p-value =", signif(p_wt,4), "\n")

#################################################################
# (iii) Score Test (LM Test) — Custom Implementation
#################################################################

score_test <- function(y, X) {
  X_full <- model.matrix(glm_full)
  fit0 <- glm(am ~ 1, family=binomial(link="logit"), data=mtcars)
  mu0 <- fitted(fit0)
```

```
  W <- mu0 * (1 - mu0)
  U <- colSums(X_full * (y - mu0))     # score vector
  I <- t(X_full) %*% (X_full * W)      # information matrix
  LM <- t(U) %*% solve(I) %*% U
  p <- pchisq(LM, df=ncol(X_full)-1, lower.tail=FALSE)
  list(statistic=LM, p.value=p)
}

lm_res_custom <- score_test(mtcars$am, mtcars$wt)

cat("\nScore (LM) Test (custom):\n")
cat("LM Statistic =", round(lm_res_custom$statistic,4),
    ", df =", ncol(model.matrix(glm_full))-1,
    ", p-value =", signif(lm_res_custom$p.value,4), "\n")

#################################################################
# (iv) SPRT — Explanation Only
#################################################################

cat("\nSequential Probability Ratio Test (SPRT):\n")
cat("SPRT requires sequential data collection; it is not applicable to a fixed dataset like mtcars.\n")
cat("Decision is based on log(Likelihood_H1 / Likelihood_H0) crossing upper/lower thresholds.\n")
```