Stat 142 Machine Problem 1

First Semester, A.Y. 2025-26

I. Monte Carlo Experiment and Confidence Intervals

In this item, you will use **Monte Carlo experiment** to study the performance (coverage probability and length) of three CI methods (Normal, t-type, and percentile bootstrap CI) for the population mean under different settings.

A. Bootstrap Implementation

Confidence Intervals

Write separate functions to compute each of the following 95% confidence intervals for the population mean (include documentations):

a. **Normal-theory (z-based) CI**:

$$\bar{x} \;\pm\; z_{1-\alpha/2}\frac{s}{\sqrt{n}}$$

Bind your function to the name `normal_ci`.

b. **t-type (Studentized) Bootstrap CI and Nonparametric Bootstrap Percentile CI**:

t-type: For each bootstrap sample $b = 1, 2, ..., B$, compute

$$t_b^* = \frac{\bar{x}_b^* - \bar{x}}{SE(\bar{x}_b^*)}$$

where $\bar{x}_b^*$ is the bootstrap mean and $SE(\bar{x}_b^*) = s_b^*/\sqrt{n}$ is its standard error within the bootstrap sample. Use the empirical quantiles of the t_b^* distribution to form the $(1-\alpha)$ confidence interval:

$$\left(\bar{x} \;-\; t^*_{1-\alpha/2}SE(\bar{x}),\; \bar{x} \;-\; t^*_{(\alpha/2)}SE(\bar{x})\right)$$

where $SE(\bar{x}) = s/\sqrt{n}$ is the standard error from the original sample. This function should also return a percentile bootstrap CI (i.e., this function should return two CI types, so that you can re-use bootstrap resamples). Use replicate when performing the bootstrapping part.

Bind your function to the name `bootstrap_ci`.

* The functions should accept the following arguments:

- `x`: original sample
- `alpha`: default = 0.05
- `seed` (for item(b))
- `B`: default = 500 (for item(b))

and should return the CI/s.

```r
# a. Normal-theory (z-based) CI
normal_ci <- function(x, alpha = 0.05) {
  # Description
  # Computes a (1 - alpha) normal-theory (z-based) confidence interval for the population
  # mean using the sample mean and sample standard deviation.

   # Parameters
  # x -- numeric vector of sample data
  # alpha -- level of significance (default = 0.05 for a 95% CI)

  # Value
  # A numeric vector containing the lower and upper confidence limits of the normal-theory
  # confidence interval for the population mean.

  xbar <- mean(x)
  s <- sd(x)
  n <- length(x)
  lcl <- xbar - qnorm(1 - alpha/2)*(s/sqrt(n))
  ucl <- xbar + qnorm(1 - alpha/2)*(s/sqrt(n))
  c(lcl, ucl)
}
```

```r
# b. t-type (Studentized) Bootstrap CI and Nonparametric Bootstrap Percentile CI
bootstrap_ci <- function(x, alpha = 0.05, seed, B = 500){
  # Description
  # Computes two types of (1 - alpha)100% bootstrap confidence intervals for the population
  # mean: (1) the t-type (Studentized) bootstrap CI, and (2) the nonparametric
  # bootstrap percentile CI

  # Parameters
  # x -- numeric vector of sample data
  # alpha -- level of significance (default = 0.05 for a 95% CI)
  # seed -- an arbitrary number used to set the random seed for reproducibility

  # Value
  # A list containing:
  # t_CI -- the t-type bootstrap CI for the mean
  # perc_CI -- the percentile CI for the mean

  n <- length(x)
  xbar <- mean(x)
  s <- sd(x)
  se <- s/sqrt(n)

  set.seed(seed)

  bootstrap_estimates <- replicate(B, {
    sampling <- sample(x, size = n, replace=TRUE)
    xbar_b <- mean(sampling)
    s_b <- sd(sampling)
    se_b <- s_b/sqrt(n)
    t_b <- (xbar_b - xbar) / se_b
    c(mean=xbar_b, t=t_b)
  })


xbar_star <- bootstrap_estimates["mean", ]
t_star <- bootstrap_estimates["t", ]

  # t-type studentized CI
  t_quant <- quantile(t_star, c(alpha/2, 1-alpha/2))

  t_CI <- c(xbar - t_quant[2]*se, xbar - t_quant[1]*se)

  # Nonpar Percentile CI
  perc_CI <- quantile(xbar_star, c(alpha/2, 1-(alpha/2)))

 list(t_CI = t_CI,
      perc_CI = perc_CI)
}
```

B. Monte Carlo Experiment

You will now evaluate how well each CI method works using Monte Carlo simulation.

1. **Simulation settings**
   - True mean $\mu = 0$ and variance $\sigma^2 = 1$
   - Generate samples under two settings:
     - Case 1: Normal(0,1)
     - Case 2: (Exponential(1) - 1) (centered to have mean 0)
   - Consider sample sizes `n = 20` and `n = 100`
   - For the bootstrap seed numbers:
     1. Normal, n = 20: seed = 1000
     2. Normal, n = 100: seed = 2000
     3. Exponential, n = 20: seed = 3000
     4. Exponential, n = 100: seed = 4000
2. **Simulation proper**
   - For each case and sample size:
     - Repeat the following for `M = 500` iterations (using a for loop and setting the random seed to i for the i^{th} iteration):
       - Generate a random sample of size n (no need to use SRS, just generate directly from the distributions).
       - Compute the 95% CI for each method (Normal, t-type, percentile).
       - Record whether the CI contains the true mean.
       - Record the CI length.
   - Estimate:
     - Coverage probability (proportion of times true mean is covered)
     - Average CI length
   - **Simulation Sub-item: Monte Carlo Tail Probability (Transformed Distribution)**

- Estimate the probability using $M = 500$.

$$P(Y > 2), \quad Y = \text{Exponential}(1) - 1$$

  Note: Use the inverse transform method to generate samples from an Exponential(1) distribution, starting from random numbers obtained using a modified linear congruential generator (LCG).
  - seed number = 142
  - a = 1664525123
  - b = 1013904223
  - m = 2^32-1

- Via plotting (ggplot), show how the estimate changes if you increase or decrease M (from 1 to 500), relative to the true value. Note that the exact value of this probability is e^{-3}. Do this via `for loop` with seed number `i*10` per i^{th} iteration). Does the plot show an expected behavior? Still implement LCG for this item.

3. **Output**
   - Summarize the CI results in a **data frame** with the following variables (and print this data frame afterwards):
     - Distribution (Normal vs Exponential)
     - Sample size (n = 20, 100)
     - CI type (Normal-based, t-type, Percentile)
     - Coverage probability
     - Mean length

Write items (2) and (3) (excluding the Simulation Sub-Item) as a reusable Monte Carlo experiment function based on the settings in item (1).

Your function, named `MC_CI`, should at least accept the following arguments: distribution type, sample size n, number of iterations M, and seed. It should return an output consistent with item (3), which you will later combine after running all simulation settings. No need to include documentation.

```r
MC_CI <- function(dist, n, M = 500, seed) {

  if (!dist %in% c("Normal", "Exponential")) {
    stop("dist must be either 'Normal' or 'Exponential'")
  }

  covers_normal <- numeric(M)
  covers_t      <- numeric(M)
  covers_perc   <- numeric(M)

  lengths_normal <- numeric(M)
  lengths_t      <- numeric(M)
  lengths_perc   <- numeric(M)

  true_mu <- 0

  for (i in 1:M) {
    set.seed(i)

    # Generating sample according to the specified distribution
    if (dist == "Normal") {
      x <- rnorm(n, 0, 1)
    } else if (dist == "Exponential") {
      # Centering Exponential(1): Exp(1) has mean 1, so subtract 1 to get mean 0
      x <- rexp(n, 1) - 1
    }

    # Normal-theory (z-based) CI
    ci_norm <- normal_ci(x)
    covers_normal[i] <- ci_norm[1] <= true_mu & true_mu <= ci_norm[2]
    lengths_normal[i] <- ci_norm[2] - ci_norm[1]

    # Bootstrap CIs
    ci_boot <- bootstrap_ci(x, seed = seed)

    # t-type (Studentized) bootstrap CI
    ci_t <- ci_boot$t_CI
    covers_t[i] <- ci_t[1] <= true_mu & true_mu <= ci_t[2]
    lengths_t[i] <- ci_t[2] - ci_t[1]

    # Percentile bootstrap CI
    ci_perc <- ci_boot$perc_CI
    covers_perc[i] <- ci_perc[1] <= true_mu & true_mu <= ci_perc[2]
    lengths_perc[i] <- ci_perc[2] - ci_perc[1]
  }
  # Summarizing the results for all three CI methods
  summary_df <- data.frame(
    Distribution = dist,
    Sample_size = n,
    CI_type = c("Normal-based", "t-type", "Percentile"),
    Coverage_probability = c(mean(covers_normal),
      mean(covers_t),
      mean(covers_perc)),
    Mean_length = c(mean(lengths_normal),
      mean(lengths_t),
      mean(lengths_perc))
  )
  return(summary_df)
}
```

```r
results <- rbind(
  MC_CI("Normal",      n = 20,  M = 500, seed = 1000),
  MC_CI("Normal",      n = 100, M = 500, seed = 2000),
  MC_CI("Exponential", n = 20,  M = 500, seed = 3000),
  MC_CI("Exponential", n = 100, M = 500, seed = 4000)
)
print(results, digits = 4)
```

```
##    Distribution Sample_size      CI_type Coverage_probability Mean_length
## 1        Normal          20 Normal-based                0.932      0.8515
## 2        Normal          20       t-type                0.946      0.9137
## 3        Normal          20   Percentile                0.922      0.8205
## 4        Normal         100 Normal-based                0.952      0.3899
## 5        Normal         100       t-type                0.946      0.3896
## 6        Normal         100   Percentile                0.948      0.3826
## 7   Exponential          20 Normal-based                0.904      0.8209
## 8   Exponential          20       t-type                0.940      1.0126
## 9   Exponential          20   Percentile                0.896      0.7866
## 10  Exponential         100 Normal-based                0.956      0.3876
## 11  Exponential         100       t-type                0.958      0.4023
## 12  Exponential         100   Percentile                0.950      0.3817
```

```
# Simulation Sub-item: Monte Carlo Tail Probability

lcg <- function(seed, a, b, m, n){
  x <- numeric(n+1)
  x[1] <- seed
  for (i in 2:(n+1)){
    x[i] <- (a*x[i-1] + b) %% m
  }
  u <- x[-1]/m
  return(u)
}

# Parameters for the modified LCG
seed <- 142
a <- 1664525123
b <- 1013904223
m <- 2^32 - 1

# Estimating P(Y > 2) where Y = Exponential(1) - 1 using the inverse transform method
# and our specified LCG
U <- lcg(142,1664525123,1013904223,2^32-1,500)
Y <- -log(1-U)-1

# Then P(Y>2) is
prob <- mean(Y>2)
prob
```

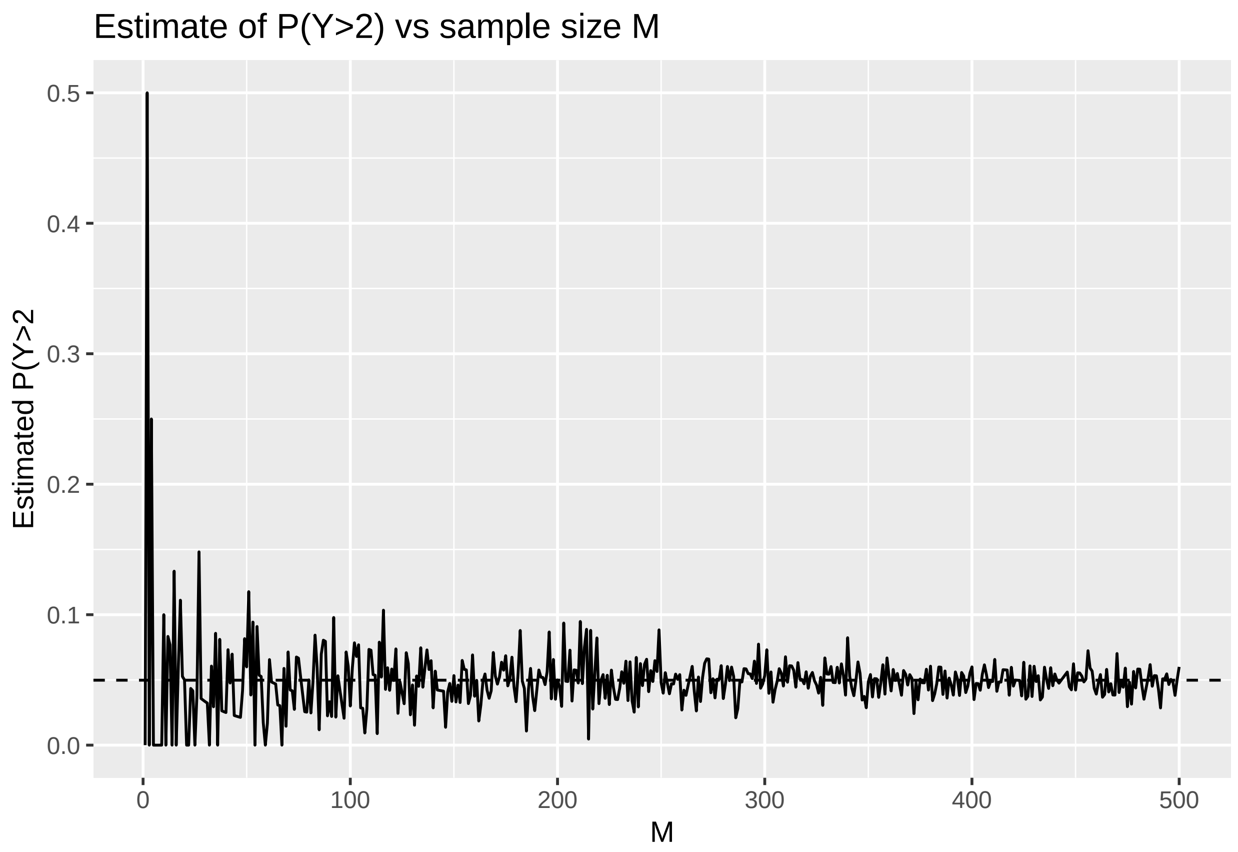
```
## [1] 0.042
```

```
# True probability
true_p <- exp(-3)
true_p
```

```
## [1] 0.04978707
```

```
# Showing how the MC estimate of P(Y > 2) changes/converges as M goes from 1 to 500
M_max <- 500
estimates <- numeric(M_max)

for (i in 1:M_max){
  seed_i <- i*10
  U <- lcg(seed_i, 1664525123, 1013904223, 2^32-1,i)
  Y <- -log(1-U)-1
  estimates[i] <- mean(Y>2)
```

```r
}

tail <- data.frame(M=1:M_max, p_hat=estimates)

true_p <- exp(-3)

ggplot(tail, aes(x=M, y=p_hat))+
  geom_line()+
  geom_hline(yintercept=true_p, linetype="dashed")+
  labs(title = "Estimate of P(Y>2) vs sample size M",
       x = "M",
       y = "Estimated P(Y>2")
```



From the plot above it can be seen that small values of M have greater fluctuations. As M grows larger, the fluctuations start to stabilize, indicating that they are converging to the true value.

C. Discussion

Answer briefly:

1. Which method achieves coverage closest to 95% under each setting?

   - For the Normal distribution, the t-type bootstrap CI performed best, achieving 94.6% coverage, for n = 20. For n = 100, the percentile bootstrap (94.8%) and normal-theory (95.2%) CIs are both equally close to achieving 95% coverage.

- For the centered Exponential distribution, the t-type bootstrap CI again is the closest to 95% coverage for n = 20. For n = 100, the percentile bootstrap CI is the closest, since it appears to have achieved a coverage of exactly 95%.

2. How does the performance change with sample size? Describe in terms of coverage probability and length.

   Regarding coverage probability, all three methods generally improve and approach 95% coverage as the sample size increases (except the t-type CI for normal distribution, which did not increase in coverage but was already close to 95% at n = 20 anyway).

   With respect to interval length, larger sample sizes consistently produce significantly shorter confidence intervals as well, meaning more reliable estimates.

3. Between symmetric and skewed data (Normal vs Exponential), which interval type is more affected (negatively) by skewness in terms of coverage?

   For the symmetric (Normal) case, all three CI methods have similar coverage. When we move to the skewed (Exponential) case, the **Normal-based and Percentile CIs lose more coverage**, especially at n = 20, while the **t-type CI changes very little**. Thus, the Normal-based and Percentile intervals are more negatively affected by skewness, and the t-type CI is the most robust.

II. Jackknife vs. Bootstrap for a Complex Statistic

In this item, you will compare the **jackknife** and **bootstrap** methods for conducting inference on a **non-linear statistic**. Use the dataset mtcars for this item.

$$Statistic : T = \frac{median(mpg)}{mean(mpg)}$$

A. Implement jackknife for T:

Write the jackknife implementation as a recursive function that returns the jackknife "resample" estimates, and perform the following (no need for documentations):

```r
set.seed(2025)
jack_recur <- function(x, i = 1, theta_jack = numeric(0)) {
  n <- length(x)

 # Base Case
  if (i > n) {
    return(theta_jack)
  }

  current_theta <- median(x[-i]) / mean(x[-i])
  new_theta_jack <- c(theta_jack, current_theta)

  # Recursive Call
  return(jack_recur(x, i = i + 1, theta_jack = new_theta_jack))
}
```

1. Compute the jackknife estimate of T.

```r
jack_result <- jack_recur(mtcars$mpg)
theta_bar <- mean(jack_result)
theta_bar
```

```
## [1] 0.9557569
```

2. Compute the jackknife standard error.

```r
n <- 32
var_jack <- ((n - 1) / n) * sum((jack_result - theta_bar)^2)
se_jack <- sqrt(var_jack)
se_jack
```

```
## [1] 0.05100542
```

3. Construct a 95% percentile confidence interval from the jackknife "resample" estimates.

```r
quantile(jack_result, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.9410277 0.9754789
```

B. Implement parametric bootstrap for T:

Assuming that the mpg data came from a normal distribution:

1. Compute the bootstrap estimate (aggregated, i.e., mean of the bootstrap estimates) of T using $B = 500$.

```r
set.seed(2025)
boot_recur <- function(x, B = 500, i = 1, boot_estimates = numeric(B)) {
  n <- length(x)
  x_bar <- mean(x)
  s <- sd(x)

  # Base case
  if (i > B) {
    return(boot_estimates)
  }

  temp_samp <- rnorm(n, mean = x_bar, sd = s)
  boot_estimates[i] <- median(temp_samp) / mean(temp_samp)

  # Recursive call
  boot_recur(x, B, i + 1, boot_estimates)
}

result <- boot_recur(mtcars$mpg)
mean(result)
```

```
## [1] 0.9998657
```

2. Compute the bootstrap standard error.

```r
sd(result)
```

```
## [1] 0.03782994
```

3. Compute the 95% percentile confidence interval.

```r
quantile(result, c(0.025, 0.975))
```

```
##      2.5%     97.5% 
## 0.9317288 1.0711211 
```

4. Compute the bootstrap p-value for $H_a : T \neq 1$ and interpret the result for $\alpha = 0.05$.

```r
T0 <- 1
T_obs <- median(mtcars$mpg) / mean(mtcars$mpg)
T_centered <- result - mean(result) + T0
p_value <- mean(abs(T_centered - T0) >= abs(T_obs - T0))
p_value
```

```
## [1] 0.252
```

Since the p-value $= 0.252$, at $\alpha = 0.05$, we do NOT reject the null hypothesis. That is, we do not have sufficient evidence to say that the ratio of median mpg and mean mpg is not equal to 1.

C. Comparison and Interpretation

1. Compare the standard error estimates from jackknife and bootstrap.

   The bootstrap estimate has a lower standard error estimate (≈ 0.0378) than the jackknife estimate (≈ 0.0510).

2. Compare the confidence intervals. Are they similar in length and center?

   The **Bootstrap CI is significantly longer** (length $= 0.139$) than the **Jackknife CI** (length $= 0.034$), indicating the Bootstrap estimated much higher variability, while the Bootstrap's center (1.001) is also higher than the Jackknife's center (0.958).

3. Compare the bootstrap p-value with the jackknife CI. Do they lead to the same conclusion about H_a?

   The bootstrap p-value (0.252) leads to the non-rejection of the null hypothesis that $T = 1$, while the jackknife CI leads to the rejection of the null hypothesis that $T - 1$. Hence, the two do not lead to the same conclusion about H_a.

III. Model Comparisons

Similar to item II, use `mtcars` data for the next items. You will be comparing 4 models (3 of which are polynomial models) in terms of their predictive performance via k-Fold cross-validation.

1. **k-Fold CV**

   - Write a k-fold CV function, that returns both the CV MSE and CV MAE. Evaluate the models below using your function (Note that all of these can be fitted via`lm()`).
     - Model A: `mpg ~ wt + hp`

    – Model B to D: `mpg ~ poly(wt, d, raw=TRUE) + hp` for d = 2, 3, 4

- Perform both 5-fold (use starting seed = 1500) and 10-fold (use starting seed = 1901) cross-validation. Compare and interpret the resulting CV MSEs and MAEs. What can you conclude about the differences between the two settings?
- Note: For polynomial terms, make sure to double check the coefficient name generated by lm() for later use (e.g., poly(wt, 2, raw = TRUE) corresponds to the wtnot the quadratic term wt^2.

```r
# CV function that returns the CV MSE and CV MAE
kfold <- function(formula, data, k) {

  # Description:
  # Returns the cross-validated Mean Squared Error (MSE) and Mean Absolute Error
# (MAE) for a linear model using k-fold cross-validation.

  # Parameters:
  # formula -- formula of the model to be fitted
  # data -- data set containing the model variables
  # k -- number of folds to use for cross-validation

  # Value:
  # A list containing the following:
  # CV MSE -- cross-validated mean squared error averaged across all folds
  # CV MAE -- cross-validated mean absolute error averaged across all folds

  n <- nrow(data)
  # Assigning folds
  folds <- sample(rep(1:k, length.out = n))

  MSE <- numeric(k)
  MAE <- numeric(k)
  for (i in 1:k) {
    # Splitting the dataset
    train <- data[folds!=i,]
    test <- data[folds==i,]

    # Modelling
    mod <- lm(formula, data = train)
    pred <- predict(mod, test)
    y  <- all.vars(formula)[1]
    error <- test[[y]] - pred
    MSE[i] <- mean(error^2)
    MAE[i] <- mean(abs(error))
  }
  return(list(CV_MSE = mean(MSE),
      CV_MAE = mean(MAE)))
}
```

```r
# 5-fold CV

set.seed(1500)

models <- list(
  ModelA = mpg ~ wt + hp,
  ModelB = mpg ~ poly(wt, 2, raw = TRUE) + hp,
  ModelC = mpg ~ poly(wt, 3, raw = TRUE) + hp,
  ModelD = mpg ~ poly(wt, 4, raw = TRUE) + hp
)

for (m in names(models)) {
  fivef_cv  <- kfold(models[[m]], mtcars, k = 5)

  cat("\n==============", m, "==============\n",
      "5-Fold  CV MSE:", fivef_cv$CV_MSE,  "\n",
      "        CV MAE:", fivef_cv$CV_MAE,  "\n")
}
```

```
##
## ============== ModelA ==============
##  5-Fold  CV MSE: 7.383884
##          CV MAE: 2.106932
##
## ============== ModelB ==============
##  5-Fold  CV MSE: 6.245041
##          CV MAE: 2.104016
##
## ============== ModelC ==============
##  5-Fold  CV MSE: 5.994254
##          CV MAE: 2.009083
##
## ============== ModelD ==============
##  5-Fold  CV MSE: 6.710549
##          CV MAE: 2.110516
```

```r
# 10-fold CV

set.seed(1901)
models <- list(
  ModelA = mpg ~ wt + hp,
  ModelB = mpg ~ poly(wt, 2, raw = TRUE) + hp,
  ModelC = mpg ~ poly(wt, 3, raw = TRUE) + hp,
  ModelD = mpg ~ poly(wt, 4, raw = TRUE) + hp
)

for (m in names(models)) {
  tenf_cv <- kfold(models[[m]], mtcars, k = 10)

  cat("\n==============", m, "==============\n",
      "10-Fold CV MSE:", tenf_cv$CV_MSE, "\n",
      "        CV MAE:", tenf_cv$CV_MAE, "\n")
}
```

```
##
## ============== ModelA ==============
##  10-Fold CV MSE: 7.234148
##          CV MAE: 2.08931
##
```

```
## ============== ModelB ==============
##  10-Fold CV MSE: 5.779092
##          CV MAE: 1.953107
##
## ============== ModelC ==============
##  10-Fold CV MSE: 6.754576
##          CV MAE: 2.117272
##
## ============== ModelD ==============
##  10-Fold CV MSE: 6.582251
##          CV MAE: 2.066924
```

For Model A, the 10-fold CV produces slightly lower CV MSE and CV MAE than the 5-fold CV. This indicates that using more folds, and therefore having more data in each training split, improves the predictive accuracy and reduces bias in the error estimates. This is expected because larger training sets generally lead to more stable and reliable model performance.

For Model B, the difference is even more noticeable since both the CV MSE and CV MAE are clearly lower under 10-fold CV than under 5-fold CV. Like Model A, Model B benefits from the increased training data provided by using more folds suggesting that this model becomes more accurate and stable when the training portion is maximized.

In contrast, for Model C, the error estimates are lower under 5-fold CV than under 10-fold CV. This means Model C performs slightly better when each fold has a larger test set and a smaller training set. Model C becomes less stable when trained on the smaller training sets of 10-fold CV, possibly because of its complexity or sensitivity to the fold partitions.

For Model D, the CV MSE decreases slightly when using 10-fold CV compared to 5-fold CV, while the CV MAE also decreases. This indicates that Model D performs slightly better under 10-fold CV, but still the results are fairly consistent across different fold splits.

Overall, the effect of increasing the number of folds is not the same across all models. Simpler or moderately complex models (A and B) clearly benefit from using more folds, while more flexible models (C and D) show less consistent behavior; for instance, Model C performs worse under 10-fold CV, while Model D improves only slightly.

Under 5-fold CV, the model with the lowest CV MSE is Model C, so it will be used for the next item.

2. **Coefficient estimation via bootstrap**

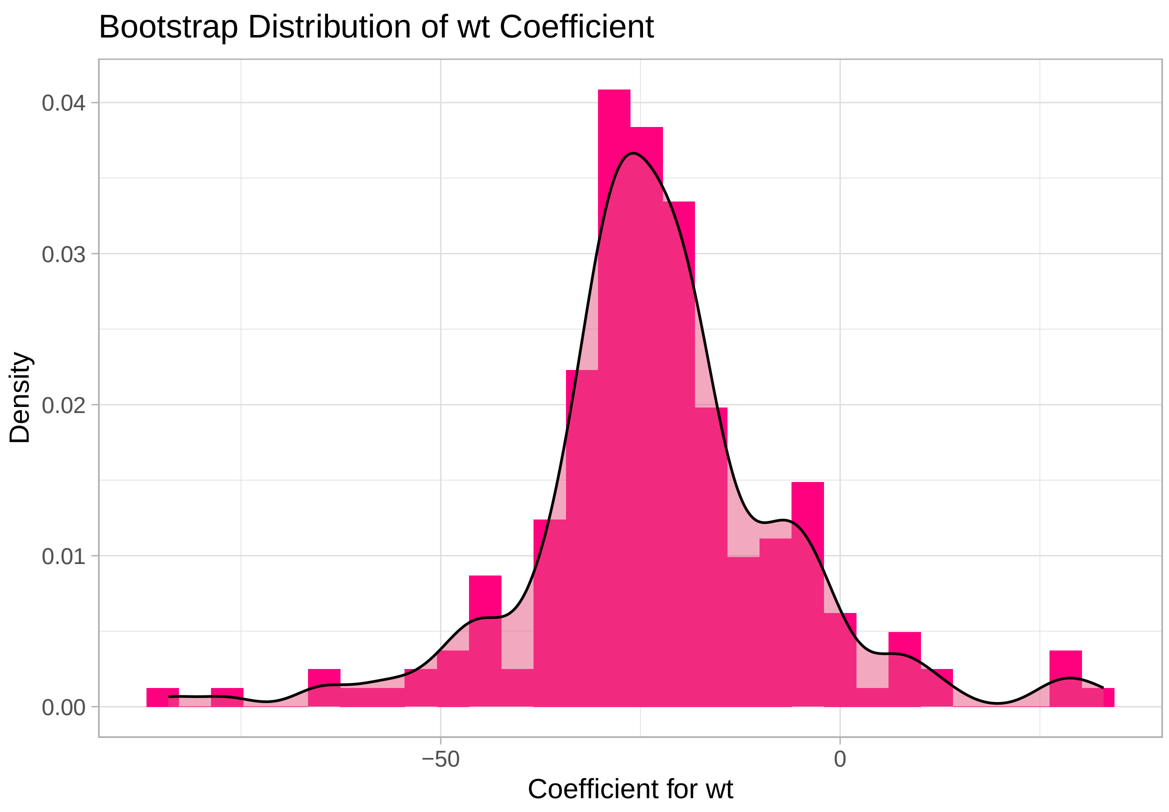
   - Refit the CV-selected model from item (1) on $B = 200$ bootstrap samples (resample rows with replacement from the training set) and use a starting seed of 2026. For each fitted model, record the coefficient estimate of `wt`. Produce a density plot (using ggplot) of the bootstrap coefficient estimates and compute mean (overall bootstrap estimate), SD, and 95% percentile and empirical CI. Compare the lengths of the percentile and empirical CI, and describe the shape and spread of the bootstrap distribution of `wt` based on the summaries and plot you obtained.
   - Fit the CV-selected model using the entire dataset and compare the coefficient estimate from this fitted model with the overall bootstrap estimate.
   - Finally, conduct a bootstrap hypothesis test for $H_a : \beta_{wt} \neq 0$ for $\alpha = 0.05$ and interpret the result.

```r
# Refitting the CV-selected model on B = 200 bootstrap samples and starting
# seed = 2026

set.seed(2026)
n <- nrow(mtcars)
B <- 200
```

```r
boot_coefs <- numeric(B)

for (b in 1:B) {
  idx <- sample(1:n, size = n, replace = TRUE)
  df_b <- mtcars[idx,]
  fit_b <- lm(mpg ~ poly(wt, 3, raw = TRUE) + hp, data = df_b)
  boot_coefs[b] <- coef(fit_b)["poly(wt, 3, raw = TRUE)1"]
}
```

```r
# Density plot of the bootstrap coefficient estimates

ggplot(data = data.frame(wt = boot_coefs), aes(wt)) +
  geom_histogram(aes(y = after_stat(density)), fill = "#FF007F") +
  geom_density(alpha = 0.5, fill = "#E75480") +
  theme_light() +
    labs(title = "Bootstrap Distribution of wt Coefficient",
       x = "Coefficient for wt",
       y = "Density")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
# Mean
boot_mean <- mean(boot_coefs)
boot_mean
```

```
## [1] -22.67506
```

```
# SD
boot_sd <- sd(boot_coefs)
boot_sd
```

```
## [1] 16.31446
```

```
# Fitting the CV-selected model using the entire dataset
fit_orig <- lm(mpg ~ poly(wt, 3, raw = TRUE) + hp, data = mtcars)

# Obtain the coefficient estimate from this fitted model (fit_orig)
coef_orig <- coef(fit_orig)["poly(wt, 3, raw = TRUE)1"]
```

```
# Percentile CI
percentile_CI <- quantile(boot_coefs, probs = c(0.025, 0.975))

# Empirical CI
empirical_CI <- c(
  2*coef_orig - quantile(boot_coefs, 0.975),
  2*coef_orig - quantile(boot_coefs, 0.025))

# Display the results
ci_summary <- data.frame(
  CI_Type = c("Percentile CI", "Empirical CI"),
  Lower = c(percentile_CI[1], empirical_CI[1]),
  Upper = c(percentile_CI[2], empirical_CI[2]),
  Length = c(diff(percentile_CI), diff(empirical_CI))
)

print(ci_summary)
```

```
##                                CI_Type     Lower    Upper   Length
## 2.5%                     Percentile CI -57.02295 11.54910 68.57205
## poly(wt, 3, raw = TRUE)1  Empirical CI -54.44294 14.12912 68.57205
```

Both the percentile and empirical (basic) bootstrap confidence intervals have almost the same length, differing only slightly in their endpoints. This happens when the bootstrap distribution is not perfectly symmetric, but not highly skewed either. In this case, the empirical CI is shifted slightly upward compared to the percentile CI, reflecting a mild left skew in the distribution.

The density plot supports this. The distribution is generally bell-shaped but shows a longer left tail. Because the skewness is small, both CI methods produce intervals of nearly identical width, differing mostly by a slight shift rather than a change in length.

```
# Display the results
coef_table <- data.frame(
  Description = c("Coefficient from entire dataset", "Bootstrap mean coefficient"),
  Value = c(coef_orig, boot_mean),
  row.names = NULL
)
coef_table
```

```
##                       Description     Value
## 1 Coefficient from entire dataset -21.44692
## 2      Bootstrap mean coefficient -22.67506
```

The two estimates are very close, differing by only -1.22814, indicating that the coefficient for wt is highly stable across bootstrap samples. This small difference represents a minimal estimated bias, indicating that the original coefficient is near the center of the bootstrap distribution. Combined with its stability, this indicates that the coefficient is reliable.

Through the bootstrap confidence intervals, we can see that 0 is included in both the percentile and the empirical CI. Therefore, we do not reject Ho. At alpha = 0.05, we do not have sufficient evidence to conclude that the coefficient of wt differs from 0. Therefore, wt does not have a significant effect on fuel efficiency.