TITLE HERE

A Thesis Proposal Presented to

The Faculty of the SS

univ

In Partial Fulfillment

of the Requirements for the Degree of

degree

1st Semester A.Y. 2025-2026

by

Nmae

```
knitr::opts_chunk$set(
  echo    = FALSE,
  message = FALSE,
  warning = FALSE
)
```

Abstract

Contents

1 Methodology

Let $\theta_1, \theta_2, \ldots, \theta_K$ be the true parameter values and $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_K$ be the estimates obtained.

1.1 Parametric bootstrap

Let $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_K$ be independent but not identically distributed estimates. For this study, it is assumed that $\hat{\theta}_k \sim N\left(\theta_k, \sigma_k^2\right)$, $k = 1, 2, \ldots, K$, where σ_k^2 is known. Denote the corresponding ordered values by $\hat{\theta}_{(1)}, \hat{\theta}_{(2)}, \ldots, \hat{\theta}_{(K)}$.

Algorithm 1 Computation of Joint Confidence Region via Parametric Bootstrap

```
1: for $b = 1, 2, \ldots, B$ do
2:     Generate $\hat{\theta}^*_{bk} \sim N\left(\hat{\theta}_k, \sigma_k^2\right)$, $i = 1, 2, \ldots, K$ and let $\hat{\theta}_{b(1)}, \hat{\theta}_{b(2)}, \ldots, \hat{\theta}_{b(K)}$ be the
       corresponding ordered values
```

	$k=1$	$k=2$	$\ldots$	$k=K$
$b=1$	$\hat{\theta}^*_{1(1)}$	$\hat{\theta}^*_{1(2)}$	$\ldots$	$\hat{\theta}^*_{1(K)}$
$b=2$	$\hat{\theta}^*_{2(1)}$	$\hat{\theta}^*_{2(2)}$	$\ldots$	$\hat{\theta}^*_{2(K)}$
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$
$b=B$	$\hat{\theta}^*_{B(1)}$	$\hat{\theta}^*_{B(2)}$	$\ldots$	$\hat{\theta}^*_{B(K)}$

```
3:     Compute
           $\hat{\sigma}^*_{b(k)} = \sqrt{\text{kth ordered value among } \left\{\hat{\theta}^{*2}_{b1} + \sigma_1^2, \hat{\theta}^{*2}_{b2} + \sigma_2^2, \ldots, \hat{\theta}^{*2}_{bK} + \sigma_K^2\right\} - \hat{\theta}^{*2}_{(k)}}$
4:     Compute $t^*_b = \max_{1 \le k \le K} \left| \frac{\hat{\theta}^*_{b(k)} - \hat{\theta}^*_k}{\sigma^*_{b(k)}} \right|$
5: end for
6: Compute the $(1-\alpha)$-sample quantile of $t^*_1, t^*_2, \ldots, t^*_B$, call this $\hat{t}$.
7: The joint confidence region of $\theta_{(1)}, \theta_{(2)}, \ldots, \theta_{(K)}$ is given by
       $\mathfrak{R} = \left[\hat{\theta}_{(1)} \pm \hat{t} \times \hat{\sigma}_{(1)}\right] \times \left[\hat{\theta}_{(2)} \pm \hat{t} \times \hat{\sigma}_{(2)}\right] \times \cdots \times \left[\hat{\theta}_{(K)} \pm \hat{t} \times \hat{\sigma}_{(K)}\right]$
   where $\hat{\sigma}_{(k)}$ is computed as
       $\hat{\sigma}_{(k)} = \sqrt{\text{kth ordered value among } \left\{\hat{\theta}_1^2 + \sigma_1^2, \hat{\theta}_2^2 + \sigma_2^2, \ldots, \hat{\theta}_K^2 + \sigma_K^2\right\} - \hat{\theta}^2_{(k)}}$
```

Algorithm 2 Computation of Coverage Probability for Parametric Bootstrap

For given values of $\theta_1, \theta_2, \ldots, \theta_K$ and thus $\theta_{(1)}, \theta_{(2)}, \ldots, \theta_{(K)}$

```
1: for replications = $1, 2, \ldots, 5000$ do
2:     Generate $\hat{\theta}_k \sim N(\theta_k, \sigma_k^2)$, for $k = 1, 2, \ldots, K$
3:     Compute the rectangular confidence region $\mathfrak{R}$ using Algorithm 1.
4:     Check if $\left(\theta_{(1)}, \theta_{(2)}, \ldots, \theta_{(K)}\right) \in \mathfrak{R}$ and compute
           $T_1 = \frac{1}{K}\sum_{k=1}^{K} \left|\Lambda_{Ok}\right|$
           $T_2 = \prod_{k=1}^{K} \left|\Lambda_{Ok}\right|$
           $T_3 = 1 - \frac{K + \sum_{k=1}^{K} \left|\Lambda_{Ok}\right|}{K^2}$
5: end for
6: Compute the proportion of times that the condition in step 4 is satisfied and the
   average of $T_1, T_2$, and $T_3$.
```

1.2 Nonrank-based method

The nonrank-based method assumes that $\hat{\boldsymbol{\theta}} = \left(\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_K\right) \sim N\left(\boldsymbol{\theta}, \boldsymbol{\Sigma}\right)$. It accounts for potential correlation among items being ranked. For this case, an exchangeable correlation, $\boldsymbol{\rho}$ (See Equation 1.1.), is assumed and used in the calculation of the variance covariance matrix (See Equation 1.2.).

$$\boldsymbol{\rho} = (1-\rho)\,\mathbf{I}_K + \rho \mathbf{1}_K \mathbf{1}_K' \tag{1.1}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Delta}^{1/2} \boldsymbol{\rho} \boldsymbol{\Delta}^{1/2} \tag{1.2}$$

where $\boldsymbol{\Delta} = \text{diag}\left\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_K^2\right\}$, with known σ_k's and ρ is studied for $0.1, 0.5, 0.9$.

Algorithm 3 Computation of Joint Confidence Region via Nonrank-based Method

Let the data consist of $\hat{\theta}_1, \ldots, \hat{\theta}_K$ and suppose $\mathbf{\Sigma}$ is known

```
1: for $b = 1, 2, \ldots, B$ do
2:     Generate $\hat{\boldsymbol{\theta}}_b^* \sim N_K\left(\hat{\boldsymbol{\theta}}, \mathbf{\Sigma}\right)$ and write $\hat{\boldsymbol{\theta}}_b^* = \left(\hat{\theta}_{b1}^*, \hat{\theta}_{b2}^*, \ldots, \hat{\theta}_{bK}^*\right)'$
3:     Compute $t_b^* = \max_{1 \le j \le K} \left| \frac{\hat{\theta}_{bj}^* - \hat{\theta}_j^*}{\sigma_j} \right|$
4: end for
5: Compute the $(1-\alpha)$-sample quantile of $t_1^*, t_2^*, \ldots, t_B^*$, call this $\hat{t}$.
6: The joint confidence region of $\theta_1, \theta_2, \ldots, \theta_K$ is given by
       $\mathfrak{R} = \left[\hat{\theta}_1 \pm \hat{t} \times \sigma_1\right] \times \left[\hat{\theta}_2 \pm \hat{t} \times \sigma_2\right] \times \cdots \times \left[\hat{\theta}_K \pm \hat{t} \times \sigma_K\right]$
```

Algorithm 4 Computation of Coverage Probability for Nonrank-based Method

For given values of $\theta_1, \theta_2, \ldots, \theta_K$ and $\mathbf{\Sigma}$

```
1: for replications $= 1, 2, \ldots, 5000$ do
2:     Generate $\hat{\boldsymbol{\theta}} \sim N_K(\boldsymbol{\theta}, \mathbf{\Sigma})$
3:     Compute the rectangular confidence region $\mathfrak{R}$ using Algorithm 3.
4:     Check if $(\theta_1, \theta_2, \ldots, \theta_K) \in \mathfrak{R}$ and compute $T_1, T_2$, and $T_3$.
5: end for
6: Compute the proportion of times that the condition in step 4 is satisfied and the
   average of $T_1, T_2$, and $T_3$.
```

1.3 Results

For the simulation studies, α is fixed at 0.1, while the true standard deviations are varied ($sd = 2.0, 3.6, 6.0$) along with the number of items to be ranked ($K = 5, 10, 20, 30, 40, 51$). Table 1 shows that when the correlation is zero, the nonrank-based methodology, independent, and Bonferroni CI exhibit similar coverage values regardless of K and sd. In contrast, the parametric approach generally yields lower coverage for smaller sd while showing comparable variability across different values of K.

Table 1: Simulation results for coverage probabilities.

		Coverage			
K	sd	Parametric	Non-rankbased	Independent	Bonferroni
	2.0	0.9198	0.8970	0.9000	0.9042
5	3.6	0.8534	0.8970	0.9000	0.9042
	6.0	0.8034	0.8970	0.9000	0.9042
	2.0	0.8900	0.8962	0.8984	0.9034
10	3.6	0.8830	0.8962	0.8984	0.9034
	6.0	0.8212	0.8962	0.8984	0.9034
	2.0	0.8842	0.9070	0.9096	0.9134
20	3.6	0.8952	0.9070	0.9096	0.9134
	6.0	0.8264	0.9070	0.9096	0.9134
	2.0	0.8786	0.9026	0.9048	0.9088
30	3.6	0.8594	0.9026	0.9048	0.9088
	6.0	0.8458	0.9026	0.9048	0.9088
	2.0	0.8820	0.8958	0.8972	0.9020
40	3.6	0.8804	0.8958	0.8972	0.9020
	6.0	0.8474	0.8958	0.8972	0.9020
	2.0	0.9442	0.9010	0.9008	0.9054
51	3.6	0.9128	0.9010	0.9008	0.9054
	6.0	0.9124	0.9010	0.9008	0.9054

The case is different in terms of T_1 as it increases with decreasing sd and increasing K. The CIs are wider for the parametric approach compared to the remaining approaches whose T_1 only vary by a small margin, with nonrank-based method having the smallest T_1 and Bonferroni, the largest one.

Table 2: Simulation results for average T_1.

		T1			
K	sd	Parametric	Non-rankbased	Independent	Bonferroni
5	2.0	2.212960	2.127360	2.128560	2.130800
	3.6	1.983280	1.834720	1.835600	1.842240
	6.0	1.622000	1.462560	1.462560	1.465520
10	2.0	4.002280	3.243800	3.246320	3.259040
	3.6	2.710680	2.307480	2.308680	2.317680
	6.0	1.924080	1.731200	1.733320	1.736840
20	2.0	7.022060	5.333380	5.336680	5.361700
	3.6	4.207100	3.448120	3.451680	3.465240
	6.0	2.787040	2.453000	2.454920	2.462440
30	2.0	11.830987	9.538733	9.547507	9.591387
	3.6	7.653613	5.802093	5.806080	5.833933
	6.0	4.764387	3.778320	3.781573	3.797987
40	2.0	15.153140	12.100160	12.108990	12.160730
	3.6	10.392030	7.201310	7.205910	7.241320
	6.0	6.508180	4.483520	4.485920	4.506070
51	2.0	20.446533	15.607796	15.614745	15.685914
	3.6	13.206918	9.098996	9.103490	9.143977
	6.0	8.598722	5.786298	5.789153	5.814400

2 Introduction

2.1 Background of the Study

2.2 Statement of the Problem

2.3 Objective of the Study

2.4 Study Hypothesis

2.5 Significance of the Study

2.6 Scope and Limitation

2.7 Definition of Terms

3 Background

THIS IS Rizzo (2008) and Klein et al. (2020)

References

Klein, M., Wright, T., & Wieczorek, J. (2020). *A joint confidence region for an overall ranking of populations.*

Rizzo, M. (2008). *Statistical computing with r.*

Appendices

Codes for algorithm 1

```
get_independent_ci <- function(theta_hat,
                               S,
                               alpha){
```

```
  K <- length(theta_hat)
  gamma = 1-(1-alpha)^(1/K)
  z = qnorm(1-gamma/2)
  ci_lower <- theta_hat - z*S
  ci_upper <- theta_hat + z*S
  return(list(
    ci_lower = ci_lower,
    ci_upper = ci_upper
  ))
}

get_bonferroni_ci <- function(theta_hat,
                              S,
                              alpha){
  K <- length(theta_hat)
  z = qnorm(1-(alpha/K)/2)
  ci_lower <- theta_hat - z*S
  ci_upper <- theta_hat + z*S
  return(list(
    ci_lower = ci_lower,
    ci_upper = ci_upper
  ))
}

get_parametric_ci <- function(B,
                              theta_hat,
                              S,
                              alpha) {
  K <- length(theta_hat)
```

```r
# step 1a ====================================
thetahat_star <- sapply(seq_len(K), function(i) {
  rnorm(B, mean = theta_hat[i], sd = S[i])
})
colnames(thetahat_star) <- paste0("thetahat_star",
                                  sprintf("%02d", 1:K))
sorted_thetahat_star <- t(apply(thetahat_star, 1, sort))
colnames(sorted_thetahat_star) <- paste0("sorted_thetahat_star",
                                         sprintf("%02d", 1:K))
# step 1b ====================================
variance_vector <- S^2
minuend <- thetahat_star^2 + rep(
  variance_vector, each = nrow(thetahat_star))
sigma_hat_star <- sqrt(
  t(apply(minuend, 1, sort)) - sorted_thetahat_star^2)
# step 1c ====================================
sorted_theta_hat <- sort(theta_hat)
t_star <- apply(
  abs(
    (
      sorted_thetahat_star - rep(
        sorted_theta_hat,
        each = nrow(sorted_thetahat_star)
        )
      )/sigma_hat_star
    ),
  1,
  max)
# step 2 =====================================
  t_hat <- quantile(t_star, probs = 1 - alpha)
  # step 3 ========================================
  sigma_hat <- sqrt(
    sort(theta_hat^2 + variance_vector) - sorted_theta_hat^2)
  # step 6 ========================================
  ci_lower <- sorted_theta_hat - t_hat*sigma_hat
  ci_upper <- sorted_theta_hat + t_hat*sigma_hat
  return(list(
    ci_lower = ci_lower,
    ci_upper = ci_upper
  ))
}

get_nonrankbased_ci <- function(B,
                                theta_hat,
                                alpha,
                                varcovar_matrix) {
  K <- length(theta_hat)
  # step 1a =======================================
  generate_data <- function(){MASS::mvrnorm(n = 1,
                                    mu = theta_hat,
                                    Sigma = varcovar_matrix)}
  thetahat_star <- t(replicate(B, generate_data()))
  # step 1b =======================================
  t_star <- apply(thetahat_star,
                  1,
                  function(x) max(abs((x - theta_hat) / sqrt(
                    diag(varcovar_matrix)))))
  # step 2 ========================================
  t_hat <- quantile(t_star, probs = 1 - alpha)
  # step 3 ========================================
  ci_lower <- theta_hat - t_hat*sqrt(diag(varcovar_matrix))
  ci_upper <- theta_hat + t_hat*sqrt(diag(varcovar_matrix))
  return(list(
    ci_lower = ci_lower,
    ci_upper = ci_upper
  ))
}
```

Codes for algorithm 2

```r
 source("../../R/compute_ci.R")
library("doRNG")

get_ranks <- function(k, tuple_list){
  Lambda_lk <- which(
    tuple_list[,2]<=tuple_list[k,1])
  Lambda_lk <- Lambda_lk[Lambda_lk != k]
  Lambda_Ok <- which(
    tuple_list[,2]>tuple_list[k,1] & tuple_list[k,2] > tuple_list[,1])
  Lambda_Ok <- Lambda_Ok[Lambda_Ok != k]
  ranks <- seq(
    length(unique(Lambda_lk)) + 1,
    length(unique(Lambda_lk)) + length(unique(Lambda_Ok)) + 1,
    1
  )
  return(list(
    ranks = ranks,
    Lambda_Ok = Lambda_Ok
  ))
}

get_t1 <- function(v) mean(v)

get_t2 <- function(v) prod(v)^(1/length(v))

get_t3 <- function(v) {
  1 - ((length(v)+sum(v))/(length(v)^2))
}

get_coverage <- function(ci_lower,
                         ci_upper,
                         true_theta) {
  return(all(ci_lower<=true_theta) & all(true_theta<=ci_upper))
}

algo2_nonrankbased <- function(
    true_theta,
    K,
    reps = 5, # step 4
    B=100,
    alpha= 0.10,
    varcovar_matrix){
  foreach(iter = 1:reps,
          .combine = rbind,
          .packages = c("foreach", "arrow", "MASS"),
          .export = c("get_nonrankbased_ci", "get_independent_ci",
                     "get_bonferroni_ci", "get_ranks", "get_coverage",
                     "get_t1", "get_t2", "get_t3")
  ) %dorng% {

    # step 1 =======
    theta_hat <- mvrnorm(n = 1,
                         mu = true_theta,
                         Sigma = varcovar_matrix)

    # step 2 =======
    S <- sqrt(diag(varcovar_matrix))

    ci_methods <- list(
      nonrankbased = function() get_nonrankbased_ci(B, theta_hat, alpha,
                                                    varcovar_matrix),
      independent  = function() get_independent_ci(theta_hat, S, alpha),
      bonferroni   = function() get_bonferroni_ci(theta_hat, S, alpha)
    )

    ci_results <- lapply(ci_methods, function(f) f())

    coverages <- lapply(ci_results, function(res) {
      get_coverage(
        ci_lower   = res$ci_lower,
        ci_upper   = res$ci_upper,
        true_theta = true_theta
      )
    })
```

```
process_ci_result <- function(result, K) {
  tuple_list <- t(apply(
    data.frame(
      ci_lower = result$ci_lower,
      ci_upper = result$ci_upper
    ),
    1,
    function(row) as.numeric(row)
  ))

  rank_range_length <- sapply(1:K, function(x)
    length(get_ranks(x, tuple_list)$ranks)
  )

  list(
    t1 = get_t1(rank_range_length),
    t2 = get_t2(rank_range_length),
    t3 = get_t3(rank_range_length)
  )
}

processed <- lapply(ci_results, process_ci_result, K = K)

data.frame(
  t1_nonrankbased = processed$nonrankbased$t1,
  t2_nonrankbased = processed$nonrankbased$t2,
  t3_nonrankbased = processed$nonrankbased$t3,
  coverage_nonrankbased = coverages$nonrankbased,
  t1_independent = processed$independent$t1,
      t2_independent = processed$independent$t2,
      t3_independent = processed$independent$t3,
      coverage_independent = coverages$independent,
      t1_bonferroni = processed$bonferroni$t1,
      t2_bonferroni = processed$bonferroni$t2,
      t3_bonferroni = processed$bonferroni$t3,
      coverage_bonferroni = coverages$bonferroni
      )
  }
}


algo2_parametric <- function(
    true_theta,
    K,
    reps = 5, # step 4
    B=100,
    alpha= 0.10,
    S){
  foreach(iter = 1:reps,
          .combine = rbind,
          .packages = c("foreach", "arrow", "MASS"),
          .export = c("get_parametric_ci","get_ranks", "get_coverage",
                      "get_t1", "get_t2", "get_t3")
  ) %dorng% {

    # step 1 =======
    theta_hat <- rnorm(
      n    = K,
```

```r
  mean = true_theta,
  sd   = S
)

# step 2 =======
result <- get_parametric_ci(B,
                            theta_hat,
                            S,
                            alpha)

# step 3 =======
sorted_true_theta <- sort(true_theta)
coverage <- get_coverage(ci_lower = result$ci_lower,
                         ci_upper = result$ci_upper,
                         true_theta = sorted_true_theta)

tuple_list <- t(apply(
  data.frame(ci_lower = result$ci_lower,
             ci_upper = result$ci_upper), 1, function(row) as.numeric(row)))
rank_range_length <- sapply(1:K, function(x) length(
  get_ranks(x, tuple_list)$ranks))
t1 <- get_t1(rank_range_length)
t2 <- get_t2(rank_range_length)
t3 <- get_t3(rank_range_length)

data.frame(
  t1_parametric = t1,
  t2_parametric = t2,
  t3_parametric = t3,
      coverage_parametric = coverage
    )
  }
}
```

Codes for simulation

```
 #3:37PM
source("../../R/implement_algo2.R")

mean <- 23.8
df <- readRDS("../../data/mean_travel_time_ranking_2011.rds")
cl=parallel::makeCluster(15)
registerDoParallel(cl)

sds <- c(2, 3.6, 6)
Ks <- c(51, 40, 30, 20, 10, 5)
corrs <- c(0.1,0.5,0.9)
alphas <- c(0.1)#c(0.05, 0.1, 0.15, 0.2)

for (sd in sds) {
  for (K in Ks) {
    set.seed(123974)
    true_theta <- rnorm(K, mean, sd)
    true_sds <- df$S[1:K]

    for (alpha in alphas) {
```

```
tic("Running parametric...")
coverage_parametric_df <- algo2_parametric(true_theta,
                                          K,
                                          reps = 5000,
                                          B=500,
                                          alpha= alpha,
                                          S=true_sds)
toc()
saveRDS(coverage_parametric_df,  paste0("output/coverage_parametric_",
                                       K,"_", sd, "_", alpha, ".rds"))
for (corr in corrs) {
  corr_matrix <- (1 - corr) * diag(K) + corr * matrix(1, K, K)
  variance_vector <- true_sds^2
  delta <- diag(variance_vector)
  varcovar_matrix <- delta^(1/2) %*% corr_matrix %*% delta^(1/2)

  tic("Running nonrankbased...")
  coverage_output_df <- algo2_nonrankbased(
    true_theta,
    K,
    reps = 5000,
    B = 500,
    alpha=alpha,
    varcovar_matrix = varcovar_matrix)
  toc()

  saveRDS(coverage_output_df,  paste0("output/coverage_probability_",
                                     K,"_", sd, "_", corr, "_",
                                     alpha, ".rds"))
```

```
        }
      }
    }
  }

stopCluster(cl)

param_grid <- expand.grid(K = Ks, sd = sds, corr = corrs, alpha = alphas)

results <- do.call(rbind, lapply(seq_len(nrow(param_grid)), function(i) {
  K <- param_grid$K[i]
  sd <- param_grid$sd[i]
  corr <- param_grid$corr[i]
  alpha <- param_grid$alpha[i]

  a <- readRDS(paste0("output/coverage_probability_",
                      K, "_", sd, "_", corr, "_", alpha, ".rds"))

  data.frame(
    K = K,sd = sd,corr = corr,alpha = alpha,
    Cov_nonrankbased = mean(a$coverage_nonrankbased),
    Cov_independent = mean(a$coverage_independent),
    Cov_bonferroni = mean(a$coverage_bonferroni),
    T1_nonrankbased = mean(a$t1_nonrankbased),
    T1_independent = mean(a$t1_independent),
    T1_bonferroni = mean(a$t1_bonferroni),
    T2_nonrank = mean(a$t2_nonrankbased),
    T2_independent = mean(a$t2_independent),
    T2_bonferroni = mean(a$t2_bonferroni),
```

```r
    T3_independent = mean(a$t3_independent),
    T3_nonrankbased = mean(a$t3_nonrankbased),
    T3_bonferroni = mean(a$t3_bonferroni)
  )
}))

param_grid <- expand.grid(K = Ks, sd = sds, alpha = alphas)

results1 <- do.call(rbind, lapply(seq_len(nrow(param_grid)), function(i) {
  K <- param_grid$K[i]
  sd <- param_grid$sd[i]
  alpha <- param_grid$alpha[i]

  a <- readRDS(paste0("output/coverage_parametric_",
                      K, "_", sd, "_", alpha, ".rds"))

  data.frame(
    K = K,
    sd = sd,
    alpha = alpha,
    Cov_parametric = mean(a$coverage_parametric),
    T1_parametric = mean(a$t1_parametric),
    T2_parametric = mean(a$t2_parametric),
    T3_parametric = mean(a$t3_parametric)
  )
}))

save(results, results1, file = "simulation_results.RData")
```