Final Project Code

Zebang Zhang (zz3309)

2025-12-16

Code repository (GitHub): https://github.com/ZebangZHANGstat/P8124-Final-Project.git

```r
library(tidyverse)
library(janitor)
library(glasso)
library(Matrix)
library(glmnet)
```

Step 0: Build a subject-level index table and sanity-check the inputs

```r
# Paths
data_dir    <- "data"
pheno_path  <- "phenotypic_CMU.csv"
labels_path <- "dos160_labels.csv"

# ROI files
roi_pattern <- "rois_dosenbach160\\.1D$"
roi_files <- list.files(data_dir, pattern = roi_pattern, full.names = TRUE)
stopifnot(length(roi_files) > 0)

# Read phenotype and ROI labels
pheno <- readr::read_csv(pheno_path, show_col_types = FALSE) %>%
  janitor::clean_names()

labels <- readr::read_csv(labels_path, show_col_types = FALSE) %>%
  janitor::clean_names()

stopifnot(all(c("sub_id", "dx_group") %in% names(pheno)))

# summarize a .1D file
peek_roi_1d <- function(path) {
  header <- readLines(path, n = 1, warn = FALSE) %>% stringr::str_trim()
  cols <- stringr::str_split(header, "\\s+", simplify = TRUE)
  n_cols <- ncol(cols)

  # Count total lines; first line is header
  n_lines <- length(readLines(path, warn = FALSE))
  n_rows <- max(0, n_lines - 1)
  extra_col_name <- if (n_cols > 160) cols[1, n_cols] else NA_character_

  tibble(
    t_points = n_rows,
    n_cols = n_cols,
    extra_col_name = extra_col_name
  )
}

# Build file index
extract_sub_id <- function(file_name) {
  m <- stringr::str_match(file_name, "CMU_[A-Za-z]_([0-9]+)_rois")
  as.integer(m[, 2])  # converts "0050642" -> 50642, "50642" -> 50642
}

roi_index <- tibble(
  file_path = roi_files,
  file_name = basename(roi_files),
  sub_id = purrr::map_int(basename(roi_files), extract_sub_id)
) %>%
  arrange(sub_id)

bad_files <- roi_index %>% filter(is.na(sub_id))
if (nrow(bad_files) > 0) {
  print(bad_files %>% select(file_name))
  stop("Failed to parse SUB_ID from some ROI filenames. See printed file_name(s) above.")
}

# Attach file-level metadata
roi_meta <- purrr::map_dfr(roi_index$file_path, peek_roi_1d) %>%
  bind_cols(roi_index) %>%
  mutate(
    has_extra_col = (n_cols != 160),
    extra_col_is_223 = (!is.na(extra_col_name) & extra_col_name == "#223")
  )

# Join phenotype info
subject_index <- roi_meta %>%
  left_join(pheno, by = "sub_id") %>%
  mutate(
    dx_group = factor(dx_group, levels = c(1, 2), labels = c("ASD", "NT")),
    sex = if ("sex" %in% names(.)) factor(sex, levels = c(1, 2), labels = c("Male", "Female")) else NUL
  )

# Sanity checks
missing_pheno <- subject_index %>% filter(is.na(dx_group))
if (nrow(missing_pheno) > 0) {
  warning("Some ROI files could not be matched to phenotype SUB_ID. Check filenames vs phenotypic_CMU.c
}

pheno_without_file <- pheno %>%
  anti_join(roi_meta %>% distinct(sub_id), by = "sub_id")
```

```r
# summary
cat("Number of ROI files found:", nrow(roi_meta), "\n")
```

```
## Number of ROI files found: 27
```

```r
cat("Subjects in phenotype file:", nrow(pheno), "\n")
```

```
## Subjects in phenotype file: 27
```

```r
cat("Phenotype rows without ROI file:", nrow(pheno_without_file), "\n")
```

```
## Phenotype rows without ROI file: 0
```

```r
cat("ROI files with extra columns (expected bug):", sum(subject_index$has_extra_col), "\n")
```

```
## ROI files with extra columns (expected bug): 27
```

```r
cat("ROI files where extra column is '#223':", sum(subject_index$extra_col_is_223, na.rm = TRUE), "\n\n
```

```
## ROI files where extra column is '#223': 27
```

```r
# Group counts and timepoints
group_summary <- subject_index %>%
  filter(!is.na(dx_group)) %>%
  group_by(dx_group) %>%
  summarise(
    n = n(),
    t_min = min(t_points),
    t_median = median(t_points),
    t_max = max(t_points),
    .groups = "drop"
  )

print(group_summary)
```

```
## # A tibble: 2 x 5
##   dx_group     n t_min t_median t_max
##   <fct>    <int> <dbl>    <dbl> <dbl>
## 1 ASD         14   102     216.   315
## 2 NT          13   107     220    320
```

```r
# A compact table
subject_index_compact <- subject_index %>%
  select(sub_id, dx_group, sex, age_at_scan, t_points, n_cols, extra_col_name, file_name) %>%
  arrange(dx_group, sub_id)

subject_index_compact
```

```
## # A tibble: 27 x 8
##    sub_id dx_group sex    age_at_scan t_points n_cols extra_col_name file_name
##     <dbl> <fct>    <fct>        <dbl>    <dbl>  <int> <chr>          <chr>
##  1  50642 ASD      Male            33      179    161 #223           CMU_a_0050~
##  2  50643 ASD      Male            21      250    161 #223           CMU_b_0050~
##  3  50644 ASD      Female          19      291    161 #223           CMU_b_0050~
##  4  50645 ASD      Male            20      102    161 #223           CMU_b_0050~
##  5  50646 ASD      Male            21      105    161 #223           CMU_a_0050~
##  6  50647 ASD      Male            27      190    161 #223           CMU_a_0050~
##  7  50648 ASD      Female          31      244    161 #223           CMU_b_0050~
##  8  50649 ASD      Male            22      240    161 #223           CMU_a_0050~
##  9  50650 ASD      Female          31      108    161 #223           CMU_b_0050~
## 10  50651 ASD      Male            39      304    161 #223           CMU_b_0050~
## # i 17 more rows
```

Step 1: Read ROI time series, drop the extra '#223' column, and z-score per subject (column-wise).

Outputs are saved under a dedicated cache directory.

```r
# Cache directory
cache_dir <- "cache"
if (!dir.exists(cache_dir)) {
  dir.create(cache_dir, recursive = TRUE, showWarnings = FALSE)
}

stopifnot(exists("subject_index"))
stopifnot(all(c("sub_id", "file_path", "t_points", "n_cols") %in% names(subject_index)))

# safe z-score
zscore_matrix <- function(X) {
  mu <- colMeans(X, na.rm = TRUE)
  sdv <- apply(X, 2, sd, na.rm = TRUE)
  sdv[sdv == 0 | is.na(sdv)] <- 1
  sweep(sweep(X, 2, mu, "-"), 2, sdv, "/")
}

# Read one .1D file into a numeric matrix
read_roi_1d_matrix <- function(path) {
  # IMPORTANT: disable comment parsing because column names start with '#'
  df <- readr::read_table(
    file = path,
    col_names = TRUE,
    comment = "",
    show_col_types = FALSE,
    progress = FALSE
  )

  # Drop the known extra column (161st)
  dropped_extra <- FALSE
  if (ncol(df) == 161) {
    df <- df[, 1:160]
    dropped_extra <- TRUE
  } else if (ncol(df) != 160) {
    stop("Unexpected number of columns in file: ", basename(path), " (ncol=", ncol(df), ")")
  }

  X <- as.matrix(df)
  storage.mode(X) <- "double"

  list(X = X, dropped_extra = dropped_extra)
}

# Build standardized matrices for all subjects
res_list <- purrr::pmap(
  list(subject_index$file_path, subject_index$sub_id),
  function(fp, sid) {
    out <- read_roi_1d_matrix(fp)
    X_raw <- out$X
    X_z <- zscore_matrix(X_raw)

    # Use clean, consistent ROI column names
    colnames(X_z) <- sprintf("ROI%03d", 1:ncol(X_z))

    list(
      sub_id = sid,
      X_z = X_z,
      dropped_extra = out$dropped_extra
    )
  }
)

# Create a named list of standardized matrices
X_list <- purrr::map(res_list, "X_z")
names(X_list) <- purrr::map_chr(res_list, ~ as.character(.x$sub_id))

# QC table
qc_step1 <- tibble(
  sub_id = purrr::map_dbl(res_list, "sub_id"),
  T_points = purrr::map_int(res_list, ~ nrow(.x$X_z)),
  P = purrr::map_int(res_list, ~ ncol(.x$X_z)),
  dropped_extra = purrr::map_lgl(res_list, "dropped_extra"),
  any_na = purrr::map_lgl(res_list, ~ any(is.na(.x$X_z))),
  min_col_sd = purrr::map_dbl(res_list, ~ min(apply(.x$X_z, 2, sd))),
  max_col_sd = purrr::map_dbl(res_list, ~ max(apply(.x$X_z, 2, sd)))
) %>%
  arrange(sub_id)

print(qc_step1)
```

```
## # A tibble: 27 x 7
##    sub_id T_points     P dropped_extra any_na min_col_sd max_col_sd
##     <dbl>    <int> <int> <lgl>         <lgl>       <dbl>      <dbl>
##  1  50642      179   160 TRUE          FALSE           1          1
##  2  50643      250   160 TRUE          FALSE           1          1
##   3  50644      291   160 TRUE          FALSE           1           1
##   4  50645      102   160 TRUE          FALSE           1           1
##   5  50646      105   160 TRUE          FALSE           1           1
##   6  50647      190   160 TRUE          FALSE           1           1
##   7  50648      244   160 TRUE          FALSE           1           1
##   8  50649      240   160 TRUE          FALSE           1           1
##   9  50650      108   160 TRUE          FALSE           1           1
##  10  50651      304   160 TRUE          FALSE           1           1
## # i 17 more rows
```

```r
# Sanity checks
stopifnot(all(qc_step1$P == 160))
stopifnot(all(qc_step1$dropped_extra))
stopifnot(!any(qc_step1$any_na))

# Save for later steps
saveRDS(X_list, file = file.path(cache_dir, "step1_X_list.rds"))
saveRDS(qc_step1, file = file.path(cache_dir, "step1_qc.rds"))

# Preview one subject
example_id <- names(X_list)[1]
cat("Example subject:", example_id, "\n")
```

```
## Example subject: 50642
```

```r
cat("Dim:", paste(dim(X_list[[example_id]]), collapse = " x "), "\n")
```

```
## Dim: 179 x 160
```

```r
summary(as.vector(X_list[[example_id]]))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.50846 -0.66395  0.01201  0.00000  0.67025  5.95809
```

Step 2A: Estimate subject-level Gaussian graphical models via Graphical Lasso (glasso) and select the tuning parameter rho using EBIC.

```r
fit_path <- file.path(cache_dir, "step2A_glasso_fits.rds")
sum_path <- file.path(cache_dir, "step2A_summary.rds")
set_path <- file.path(cache_dir, "step2A_settings.rds")

# Settings
rho_grid <- exp(seq(log(0.01), log(1.00), length.out = 30))
ebic_gamma <- 0.5

X_list <- readRDS(file.path(cache_dir, "step1_X_list.rds"))
qc_step1 <- readRDS(file.path(cache_dir, "step1_qc.rds"))

# Ensure ordering matches
```

```r
subject_index2 <- subject_index %>%
  mutate(sub_id_chr = as.character(sub_id)) %>%
  filter(sub_id_chr %in% names(X_list)) %>%
  arrange(sub_id)

# If cache exists AND settings match, load from cache
cache_ok <- FALSE
if (file.exists(fit_path) && file.exists(sum_path) && file.exists(set_path)) {
  old_set <- readRDS(set_path)
  same_grid <- isTRUE(all.equal(old_set$rho_grid, rho_grid))
  same_gamma <- isTRUE(all.equal(old_set$ebic_gamma, ebic_gamma))

  if (same_grid && same_gamma) {
    glasso_fits <- readRDS(fit_path)
    step2A_summary <- readRDS(sum_path)
    cache_ok <- TRUE
  }
}

if (!cache_ok) {
  # ---- EBIC helper for Gaussian graphical models ----
  # EBIC(Theta) = -2*logLik(Theta) + |E|*log(n) + 4*gamma*|E|*log(p)
  # where logLik(Theta) = (n/2)*(logdet(Theta) - trace(S %*% Theta)) up to an additive constant.
  ebic_ggm <- function(Theta, S, n, gamma = 0.5) {
    p <- nrow(Theta)

    ld <- as.numeric(determinant(Theta, logarithm = TRUE)$modulus)
    tr <- sum(S * Theta)  # trace(S %*% Theta)

    loglik <- (n / 2) * (ld - tr)

    off <- Theta
    diag(off) <- 0
    n_edges <- sum(off != 0) / 2

    ebic <- -2 * loglik + n_edges * log(n) + 4 * gamma * n_edges * log(p)

    list(ebic = ebic, n_edges = n_edges, loglik = loglik)
  }

  # Convert precision matrix to partial correlation matrix
  prec_to_pcorr <- function(Theta) {
    d <- sqrt(diag(Theta))
    P <- -Theta / (d %o% d)
    diag(P) <- 1
    P
  }

  # Fit glasso across a rho grid and select by EBIC
  fit_glasso_subject <- function(X, rho_grid, gamma = 0.5) {
    n <- nrow(X)
    S <- cov(X)
    ebic_vals <- numeric(length(rho_grid))
    edge_vals <- numeric(length(rho_grid))
    thetas <- vector("list", length(rho_grid))

    for (k in seq_along(rho_grid)) {
      rho <- rho_grid[k]
      fit <- glasso::glasso(s = S, rho = rho, penalize.diagonal = FALSE)
      Theta <- fit$wi
      thetas[[k]] <- Theta

      e <- ebic_ggm(Theta, S = S, n = n, gamma = gamma)
      ebic_vals[k] <- e$ebic
      edge_vals[k] <- e$n_edges
    }

    k_best <- which.min(ebic_vals)
    Theta_best <- thetas[[k_best]]
    P_best <- prec_to_pcorr(Theta_best)

    list(
      Theta = Theta_best,
      Pcorr = P_best,
      rho_best = rho_grid[k_best],
      ebic_best = ebic_vals[k_best],
      n_edges = edge_vals[k_best],
      rho_grid = rho_grid,
      ebic_grid = ebic_vals,
      edges_grid = edge_vals
    )
  }

  # Fit all subjects
  glasso_fits <- vector("list", nrow(subject_index2))
  names(glasso_fits) <- as.character(subject_index2$sub_id)

  for (i in seq_len(nrow(subject_index2))) {
    sid <- as.character(subject_index2$sub_id[i])
    X <- X_list[[sid]]
    glasso_fits[[sid]] <- fit_glasso_subject(X, rho_grid = rho_grid, gamma = ebic_gamma)
  }

  # summary table
  step2A_summary <- subject_index2 %>%
    transmute(
      sub_id = sub_id,
      dx_group = dx_group,
      t_points = t_points
    ) %>%
    mutate(
      rho_best = purrr::map_dbl(as.character(sub_id), ~ glasso_fits[[.x]]$rho_best),
      ebic_best = purrr::map_dbl(as.character(sub_id), ~ glasso_fits[[.x]]$ebic_best),
      n_edges_glasso = purrr::map_dbl(as.character(sub_id), ~ glasso_fits[[.x]]$n_edges)
    )
```

```r
  # Save results and settings
  saveRDS(glasso_fits, fit_path)
  saveRDS(step2A_summary, sum_path)
  saveRDS(list(rho_grid = rho_grid, ebic_gamma = ebic_gamma), set_path)
}

# Print group-level summary
print(step2A_summary %>%
        group_by(dx_group) %>%
        summarise(
          n = n(),
          rho_median = median(rho_best),
          edges_median = median(n_edges_glasso),
          edges_min = min(n_edges_glasso),
          edges_max = max(n_edges_glasso),
          .groups = "drop"
        ))
```

```
## # A tibble: 2 x 6
##   dx_group     n rho_median edges_median edges_min edges_max
##   <fct>    <int>      <dbl>        <dbl>     <dbl>     <dbl>
## 1 ASD         14      0.452          427         0      1022
## 2 NT          13      0.386          387         0      1091
```

Step 2B (Nodewise Lasso): Estimate subject-level graphs via neighborhood selection.

```r
fit_path <- file.path(cache_dir, "step2B_nodewise_fits.rds")
sum_path <- file.path(cache_dir, "step2B_summary.rds")
set_path <- file.path(cache_dir, "step2B_settings.rds")

# Settings
nodewise_nfolds <- 5
nodewise_alpha <- 1
nodewise_nlambda <- 50
nodewise_lambda_rule <- "lambda.1se"  # lambda.1se"

X_list <- readRDS(file.path(cache_dir, "step1_X_list.rds"))

subject_index2 <- subject_index %>%
  mutate(sub_id_chr = as.character(sub_id)) %>%
  filter(sub_id_chr %in% names(X_list)) %>%
  arrange(sub_id)

# If cache exists AND settings match, load from cache
cache_ok <- FALSE
if (file.exists(fit_path) && file.exists(sum_path) && file.exists(set_path)) {
  old_set <- readRDS(set_path)

  same_nfolds <- isTRUE(all.equal(old_set$nodewise_nfolds, nodewise_nfolds))
  same_alpha <- isTRUE(all.equal(old_set$nodewise_alpha, nodewise_alpha))
  same_nlambda <- isTRUE(all.equal(old_set$nodewise_nlambda, nodewise_nlambda))
  same_rule <- isTRUE(all.equal(old_set$nodewise_lambda_rule, nodewise_lambda_rule))

  if (same_nfolds && same_alpha && same_nlambda && same_rule) {
    nodewise_fits <- readRDS(fit_path)
    step2B_summary <- readRDS(sum_path)
    cache_ok <- TRUE
    cat("Step 2B: Loaded cached results from", cache_dir, "\n")
  }
}
```

```
## Step 2B: Loaded cached results from cache
```

```r
if (!cache_ok) {
  cat("Step 2B: No valid cache found. Recomputing...\n")

  # Blocked fold assignment to reduce leakage in time series CV
  make_blocked_folds <- function(n, K = 5) {
    block_size <- ceiling(n / K)
    foldid <- rep(seq_len(K), each = block_size)[seq_len(n)]
    foldid
  }

  # Fit nodewise lasso for one subject and return an adjacency matrix
  fit_nodewise_subject <- function(X, nfolds = 5, alpha = 1, nlambda = 50, lambda_rule = "lambda.1se")
    n <- nrow(X)
    p <- ncol(X)

    foldid <- make_blocked_folds(n, K = nfolds)

    adj <- matrix(0L, nrow = p, ncol = p)

    for (j in seq_len(p)) {
      y <- X[, j]
      X_minus <- X[, -j, drop = FALSE]

      cvfit <- glmnet::cv.glmnet(
        x = X_minus,
        y = y,
        alpha = alpha,
        nfolds = nfolds,
        foldid = foldid,
        standardize = FALSE,
        intercept = TRUE,
        type.measure = "mse",
        nlambda = nlambda
      )

      beta <- as.vector(coef(cvfit, s = lambda_rule))[-1]  # drop intercept
      neighbors <- which(beta != 0)

      if (length(neighbors) > 0) {
        idx_map <- setdiff(seq_len(p), j)
        adj[j, idx_map[neighbors]] <- 1L
      }
    }

    # Symmetrize with OR rule
    adj_sym <- (adj + t(adj)) > 0
    diag(adj_sym) <- FALSE
    storage.mode(adj_sym) <- "integer"

    n_edges <- sum(adj_sym) / 2

    list(
      Adj = adj_sym,
      n_edges = n_edges
    )
  }

  # Fit all subjects
  nodewise_fits <- vector("list", nrow(subject_index2))
  names(nodewise_fits) <- as.character(subject_index2$sub_id)

  for (i in seq_len(nrow(subject_index2))) {
    sid <- as.character(subject_index2$sub_id[i])
    X <- X_list[[sid]]
    nodewise_fits[[sid]] <- fit_nodewise_subject(
      X,
      nfolds = nodewise_nfolds,
      alpha = nodewise_alpha,
      nlambda = nodewise_nlambda,
      lambda_rule = nodewise_lambda_rule
    )
  }

  # Summary table
  step2B_summary <- subject_index2 %>%
    transmute(
      sub_id = sub_id,
      dx_group = dx_group,
      t_points = t_points
    ) %>%
    mutate(
      n_edges_nodewise = purrr::map_dbl(as.character(sub_id), ~ nodewise_fits[[.x]]$n_edges)
    )

  # Save results and settings
  saveRDS(nodewise_fits, fit_path)
  saveRDS(step2B_summary, sum_path)
  saveRDS(
    list(
      nodewise_nfolds = nodewise_nfolds,
      nodewise_alpha = nodewise_alpha,
      nodewise_nlambda = nodewise_nlambda,
      nodewise_lambda_rule = nodewise_lambda_rule
    ),
    set_path
  )

  cat("Step 2B: Saved new results to", cache_dir, "\n")
}

# Print group-level summary
print(step2B_summary %>%
        group_by(dx_group) %>%
        summarise(
          n = n(),
          edges_median = median(n_edges_nodewise),
          edges_min = min(n_edges_nodewise),
          edges_max = max(n_edges_nodewise),
          .groups = "drop"
        ))
```

```
## # A tibble: 2 x 5
##   dx_group     n edges_median edges_min edges_max
##   <fct>    <int>        <dbl>     <dbl>     <dbl>
## 1 ASD         14        1832.      1529      2412
## 2 NT          13        1928       1592      2452
```

Step 3A: Compare ASD vs NT connectivity networks using glasso-based partial correlations.

I vectorize the upper triangle of each subject's Pcorr matrix and compute pairwise distances.

Then I compare between-group distances (ASD-NT) vs within-group distances (ASD-ASD & NT-NT) using a permutation test.

```
fit_path <- file.path(cache_dir, "step2A_glasso_fits.rds")
sum_path <- file.path(cache_dir, "step2A_summary.rds")

stopifnot(file.exists(fit_path), file.exists(sum_path))

glasso_fits <- readRDS(fit_path)
step2A_summary <- readRDS(sum_path)

# vectorize upper triangle
vec_upper <- function(M) {
  M[upper.tri(M, diag = FALSE)]
}

# Subject list and group labels
subjects <- step2A_summary %>%
  transmute(sub_id = as.character(sub_id), dx_group = as.character(dx_group)) %>%
  filter(sub_id %in% names(glasso_fits)) %>%
  arrange(sub_id)
```

```r
sid <- subjects$sub_id
grp <- subjects$dx_group
names(grp) <- sid

stopifnot(all(grp %in% c("ASD", "NT")))

# Build Pcorr vectors
pcorr_vecs <- purrr::map(sid, ~ vec_upper(glasso_fits[[.x]]$Pcorr))
names(pcorr_vecs) <- sid

L <- length(pcorr_vecs[[1]])
stopifnot(all(purrr::map_int(pcorr_vecs, length) == L))

# Pair indices and explicit sub_i/sub_j vectors (FIX)
pair_idx <- utils::combn(seq_along(sid), 2)
i_idx <- pair_idx[1, ]
j_idx <- pair_idx[2, ]

sub_i <- sid[i_idx]
sub_j <- sid[j_idx]

# Compute all pairwise distances (Euclidean on Pcorr vectors)
dist_vec <- purrr::map2_dbl(sub_i, sub_j, function(a, b) {
  v1 <- pcorr_vecs[[a]]
  v2 <- pcorr_vecs[[b]]
  sqrt(sum((v1 - v2)^2))
})

pairs_df <- tibble(
  sub_i = sub_i,
  sub_j = sub_j,
  grp_i = grp[sub_i],
  grp_j = grp[sub_j],
  pair_type = case_when(
    grp_i == "ASD" & grp_j == "ASD" ~ "ASD-ASD",
    grp_i == "NT"  & grp_j == "NT"  ~ "NT-NT",
    TRUE                            ~ "ASD-NT"
  ),
  dist = dist_vec
)

# Summaries
summary_by_type <- pairs_df %>%
  group_by(pair_type) %>%
  summarise(
    n_pairs = n(),
    dist_mean = mean(dist),
    dist_median = median(dist),
    dist_sd = sd(dist),
    .groups = "drop"
  )

print(summary_by_type)
```

```
## # A tibble: 3 x 5
##   pair_type n_pairs dist_mean dist_median dist_sd
##   <chr>       <int>     <dbl>       <dbl>   <dbl>
## 1 ASD-ASD        91      2.16        2.39   0.870
## 2 ASD-NT        182      2.16        2.39   0.873
## 3 NT-NT          78      2.24        2.43   0.781
```

```
within_mean <- mean(pairs_df$dist[pairs_df$pair_type %in% c("ASD-ASD", "NT-NT")])
between_mean <- mean(pairs_df$dist[pairs_df$pair_type == "ASD-NT"])
delta_obs <- between_mean - within_mean

cat("\nObserved means:\n")
```

```
##
## Observed means:
```

```
cat("  between_mean (ASD-NT) =", between_mean, "\n")
```

```
##   between_mean (ASD-NT) = 2.161448
```

```
cat("  within_mean  (ASD-ASD & NT-NT pooled) =", within_mean, "\n")
```

```
##   within_mean  (ASD-ASD & NT-NT pooled) = 2.194538
```

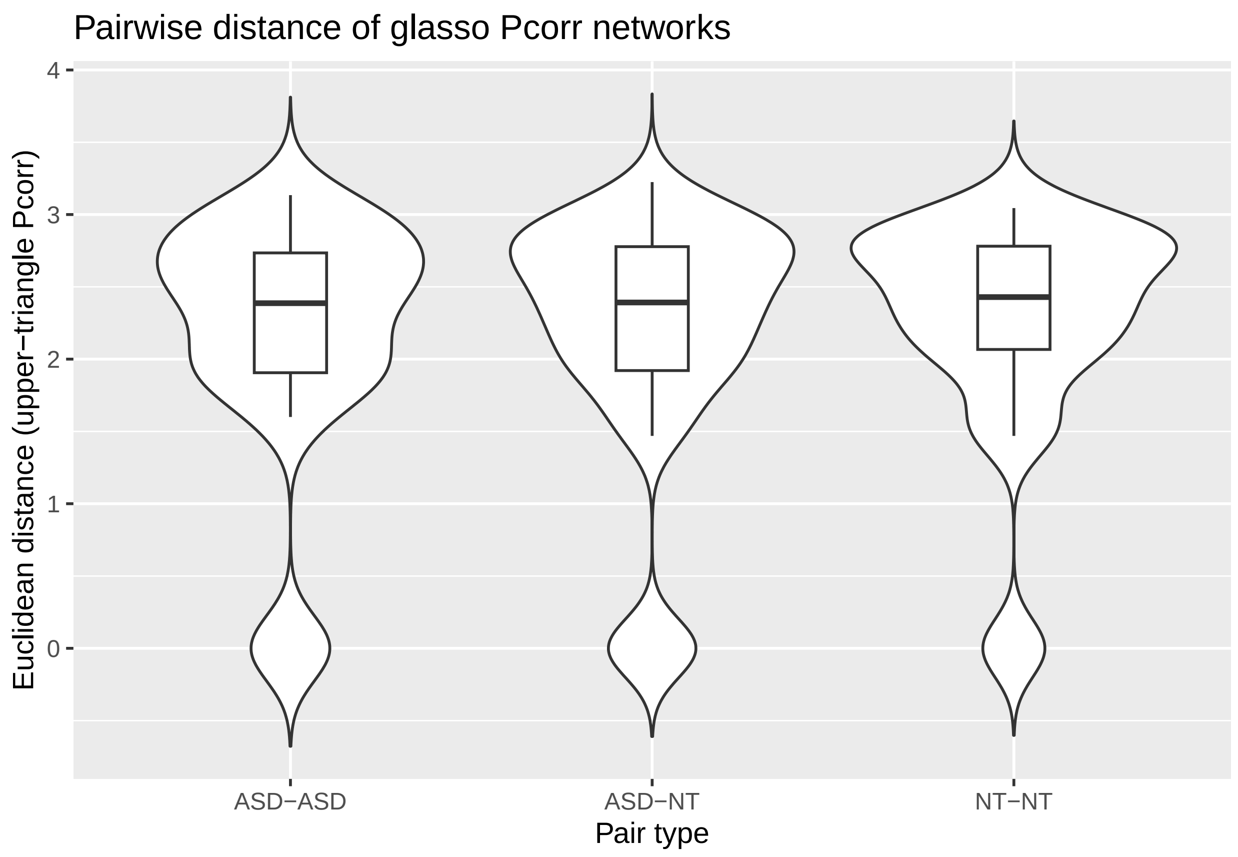
```
cat("  delta_obs (between - within) =", delta_obs, "\n\n")
```

```
##   delta_obs (between - within) = -0.03308994
```

```
# Permutation test (shuffle group labels)
set.seed(1)
B <- 2000
delta_perm <- numeric(B)

for (b in seq_len(B)) {
  grp_perm <- sample(grp, replace = FALSE)
  names(grp_perm) <- names(grp)

  grp_i_b <- grp_perm[sub_i]
  grp_j_b <- grp_perm[sub_j]

  is_between <- (grp_i_b != grp_j_b)
  between_b <- mean(dist_vec[is_between])
  within_b  <- mean(dist_vec[!is_between])

  delta_perm[b] <- between_b - within_b
}

p_two_sided <- (1 + sum(abs(delta_perm) >= abs(delta_obs))) / (B + 1)
cat("Permutation test (two-sided) p-value:", p_two_sided, "\n")
```

```
## Permutation test (two-sided) p-value: 0.4847576
```

```
# Visualization
ggplot(pairs_df, aes(x = pair_type, y = dist)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.2, outlier.shape = NA) +
  labs(
    title = "Pairwise distance of glasso Pcorr networks",
    x = "Pair type",
    y = "Euclidean distance (upper-triangle Pcorr)"
  )
```



Step 3B: Robustness check using nodewise-Lasso adjacency matrices.

I compare graphs using Jaccard distance on edge sets (upper triangle of adjacency).

Then I run the same between-vs-within comparison and permutation test.

```
fit_path <- file.path(cache_dir, "step2B_nodewise_fits.rds")
sum_path <- file.path(cache_dir, "step2B_summary.rds")

stopifnot(file.exists(fit_path), file.exists(sum_path))

nodewise_fits <- readRDS(fit_path)
step2B_summary <- readRDS(sum_path)
```

```r
# vectorize upper triangle (logical)
vec_upper_logical <- function(M) {
  as.logical(M[upper.tri(M, diag = FALSE)])
}

# Jaccard distance for binary edge indicators
jaccard_dist <- function(a, b) {
  inter <- sum(a & b)
  uni <- sum(a | b)
  if (uni == 0) return(0)
  1 - inter / uni
}

subjects <- step2B_summary %>%
  transmute(sub_id = as.character(sub_id), dx_group = as.character(dx_group)) %>%
  filter(sub_id %in% names(nodewise_fits)) %>%
  arrange(sub_id)

sid <- subjects$sub_id
grp <- subjects$dx_group
names(grp) <- sid

stopifnot(all(grp %in% c("ASD", "NT")))

adj_vecs <- purrr::map(sid, ~ vec_upper_logical(nodewise_fits[[.x]]$Adj))
names(adj_vecs) <- sid

L <- length(adj_vecs[[1]])
stopifnot(all(purrr::map_int(adj_vecs, length) == L))

# Pair indices and explicit sub_i/sub_j vectors (FIX)
pair_idx <- utils::combn(seq_along(sid), 2)
i_idx <- pair_idx[1, ]
j_idx <- pair_idx[2, ]

sub_i <- sid[i_idx]
sub_j <- sid[j_idx]

dist_vec <- purrr::map2_dbl(sub_i, sub_j, function(a, b) {
  jaccard_dist(adj_vecs[[a]], adj_vecs[[b]])
})

pairs_df <- tibble(
  sub_i = sub_i,
  sub_j = sub_j,
  grp_i = grp[sub_i],
  grp_j = grp[sub_j],
  pair_type = case_when(
    grp_i == "ASD" & grp_j == "ASD" ~ "ASD-ASD",
    grp_i == "NT"  & grp_j == "NT"  ~ "NT-NT",
    TRUE                            ~ "ASD-NT"
  ),
  dist = dist_vec
)

summary_by_type <- pairs_df %>%
  group_by(pair_type) %>%
  summarise(
    n_pairs = n(),
    dist_mean = mean(dist),
    dist_median = median(dist),
    dist_sd = sd(dist),
    .groups = "drop"
  )

print(summary_by_type)
```

```
## # A tibble: 3 x 5
##   pair_type n_pairs dist_mean dist_median dist_sd
##   <chr>       <int>     <dbl>       <dbl>   <dbl>
## 1 ASD-ASD        91     0.880       0.879 0.0117
## 2 ASD-NT        182     0.876       0.876 0.0108
## 3 NT-NT          78     0.873       0.873 0.00794
```

```r
within_mean <- mean(pairs_df$dist[pairs_df$pair_type %in% c("ASD-ASD", "NT-NT")])
between_mean <- mean(pairs_df$dist[pairs_df$pair_type == "ASD-NT"])
delta_obs <- between_mean - within_mean

cat("\nObserved means:\n")
```

```
##
## Observed means:
```

```r
cat("  between_mean (ASD-NT) =", between_mean, "\n")
```

```
##   between_mean (ASD-NT) = 0.8764295
```

```r
cat("  within_mean  (ASD-ASD & NT-NT pooled) =", within_mean, "\n")
```

```
##   within_mean  (ASD-ASD & NT-NT pooled) = 0.8766081
```

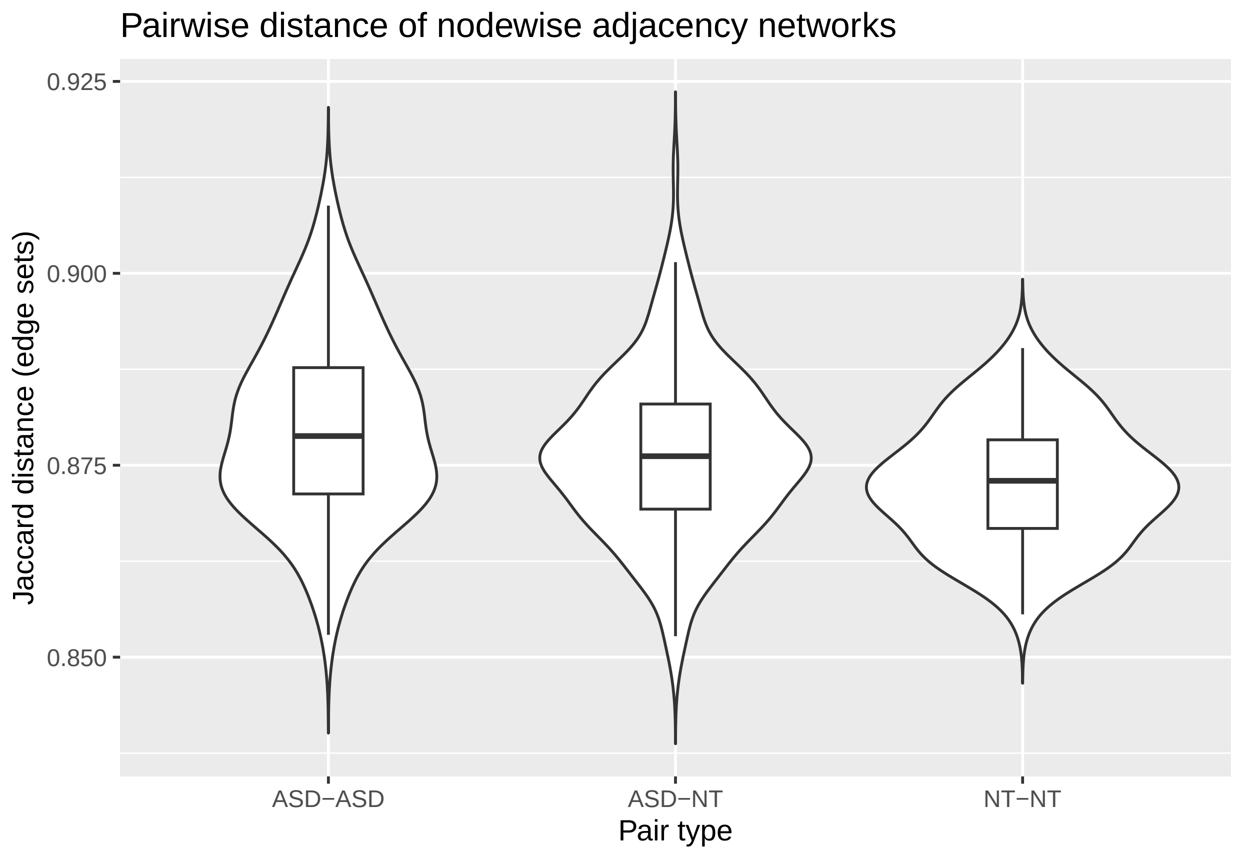
```r
cat("  delta_obs (between - within) =", delta_obs, "\n\n")
```

```
##   delta_obs (between - within) = -0.0001785877
```

```r
set.seed(1)
B <- 2000
delta_perm <- numeric(B)

for (b in seq_len(B)) {
  grp_perm <- sample(grp, replace = FALSE)
  names(grp_perm) <- names(grp)
  grp_i_b <- grp_perm[sub_i]
  grp_j_b <- grp_perm[sub_j]

  is_between <- (grp_i_b != grp_j_b)
  between_b <- mean(dist_vec[is_between])
  within_b  <- mean(dist_vec[!is_between])

  delta_perm[b] <- between_b - within_b
}

p_two_sided <- (1 + sum(abs(delta_perm) >= abs(delta_obs))) / (B + 1)
cat("Permutation test (two-sided) p-value:", p_two_sided, "\n")
```

```
## Permutation test (two-sided) p-value: 0.7951024
```

```
ggplot(pairs_df, aes(x = pair_type, y = dist)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.2, outlier.shape = NA) +
  labs(
    title = "Pairwise distance of nodewise adjacency networks",
    x = "Pair type",
    y = "Jaccard distance (edge sets)"
  )
```



Step 4A: Group-level comparison of network sparsity.

I compare ASD vs NT on: (1) glasso edge counts and density, (2) nodewise edge counts and density

I use Wilcoxon rank-sum tests (small n, robust) and report descriptive summaries (mean/median).

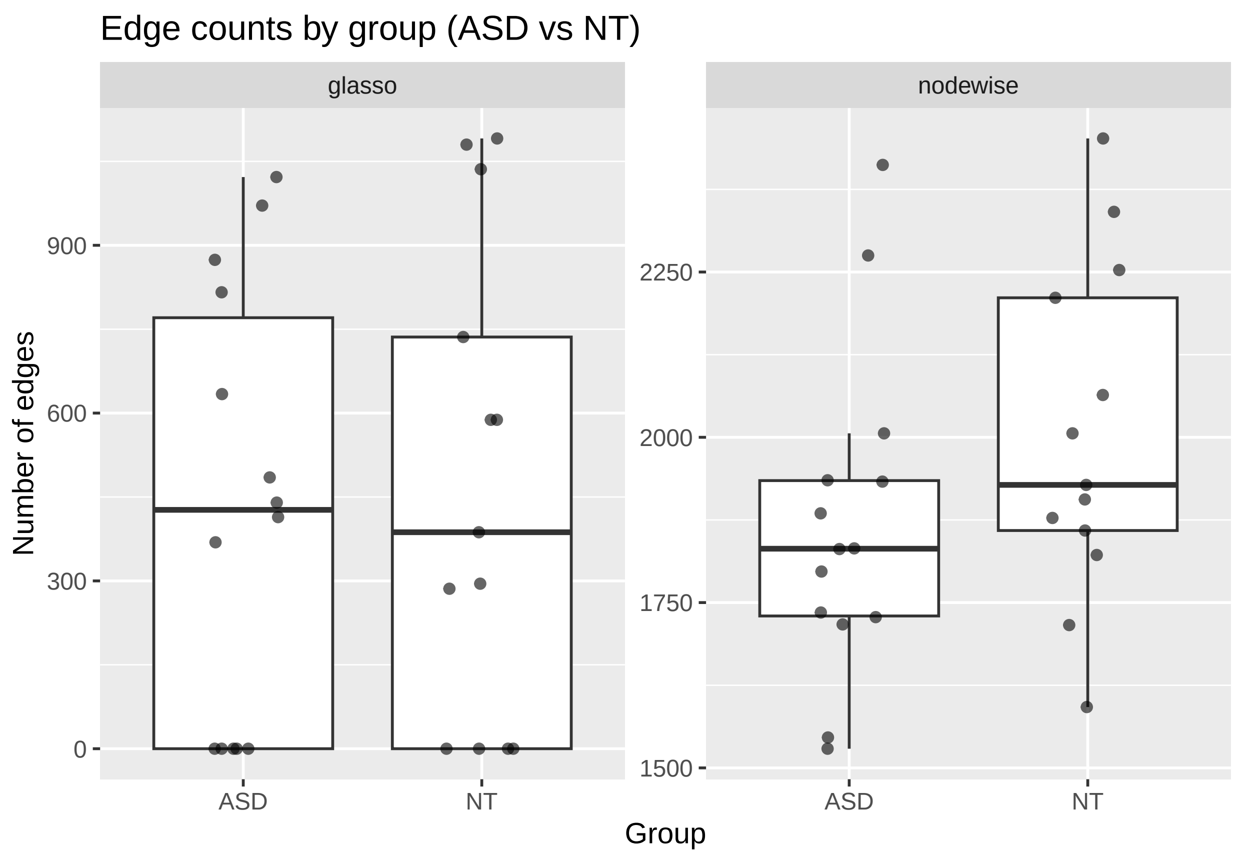
```r
# Load summaries
step2A_sum_path <- file.path(cache_dir, "step2A_summary.rds")
step2B_sum_path <- file.path(cache_dir, "step2B_summary.rds")

stopifnot(file.exists(step2A_sum_path), file.exists(step2B_sum_path))

step2A_summary <- readRDS(step2A_sum_path)
step2B_summary <- readRDS(step2B_sum_path)

# compute density from edges for p nodes
edge_density <- function(n_edges, p) {
  m <- p * (p - 1) / 2
  n_edges / m
}

p_nodes <- 160

# Build analysis tables
dfA <- step2A_summary %>%
  transmute(
    sub_id = as.character(sub_id),
    dx_group = as.character(dx_group),
    method = "glasso",
    n_edges = as.numeric(n_edges_glasso),
    density = edge_density(n_edges, p_nodes),
    rho_best = as.numeric(rho_best),
    t_points = as.numeric(t_points)
  )

dfB <- step2B_summary %>%
  transmute(
    sub_id = as.character(sub_id),
    dx_group = as.character(dx_group),
    method = "nodewise",
    n_edges = as.numeric(n_edges_nodewise),
    density = edge_density(n_edges, p_nodes),
    t_points = as.numeric(t_points)
  )

df_edges <- bind_rows(dfA, dfB) %>%
  filter(dx_group %in% c("ASD", "NT")) %>%
  mutate(dx_group = factor(dx_group, levels = c("ASD", "NT")))

# Quick descriptive summaries
desc_tbl <- df_edges %>%
  group_by(method, dx_group) %>%
  summarise(
    n = n(),
    edges_median = median(n_edges),
    edges_mean = mean(n_edges),
    dens_median = median(density),
    dens_mean = mean(density),
    .groups = "drop"
  )

print(desc_tbl)
```

```
## # A tibble: 4 x 7
##   method   dx_group     n edges_median edges_mean dens_median dens_mean
##   <chr>    <fct>    <int>        <dbl>      <dbl>       <dbl>     <dbl>
## 1 glasso   ASD         14          427        430.      0.0336    0.0338
## 2 glasso   NT          13          387        468.      0.0304    0.0368
## 3 nodewise ASD         14         1832.      1869.      0.144     0.147
## 4 nodewise NT          13         1928       2002.      0.152     0.157
```

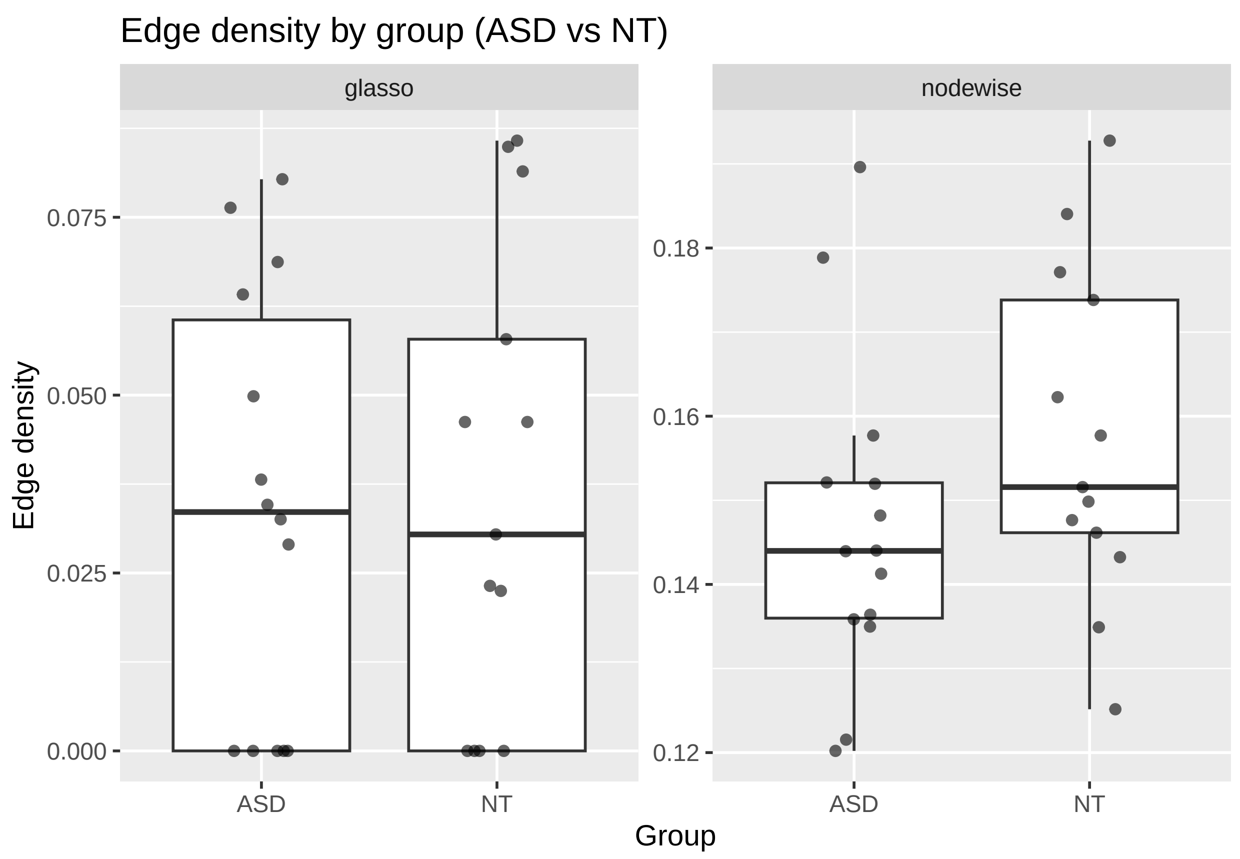
```r
# Wilcoxon tests
wilcox_by_method <- function(dat, y) {
  dat2 <- dat %>% filter(!is.na(.data[[y]]))
  res <- wilcox.test(dat2[[y]] ~ dat2$dx_group, exact = FALSE)
  tibble(
    outcome = y,
    W = unname(res$statistic),
    p_value = res$p.value
  )
}

tests <- df_edges %>%
  group_by(method) %>%
  group_modify(~ bind_rows(
    wilcox_by_method(.x, "n_edges"),
    wilcox_by_method(.x, "density")
  )) %>%
  ungroup()

print(tests)
```

```
## # A tibble: 4 x 4
##   method   outcome      W p_value
##   <chr>    <chr>    <dbl>   <dbl>
## 1 glasso   n_edges   86     0.824
## 2 glasso   density   86     0.824
## 3 nodewise n_edges   63.5   0.190
## 4 nodewise density   63.5   0.190
```

```r
# Plots: edges and density by group
ggplot(df_edges, aes(x = dx_group, y = n_edges)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0.15, height = 0, alpha = 0.6) +
  facet_wrap(~ method, scales = "free_y") +
  labs(
    title = "Edge counts by group (ASD vs NT)",
    x = "Group",
    y = "Number of edges"
  )
```



```r
ggplot(df_edges, aes(x = dx_group, y = density)) +
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(width = 0.15, height = 0, alpha = 0.6) +
  facet_wrap(~ method, scales = "free_y") +
  labs(
    title = "Edge density by group (ASD vs NT)",
    x = "Group",
    y = "Edge density"
  )
```



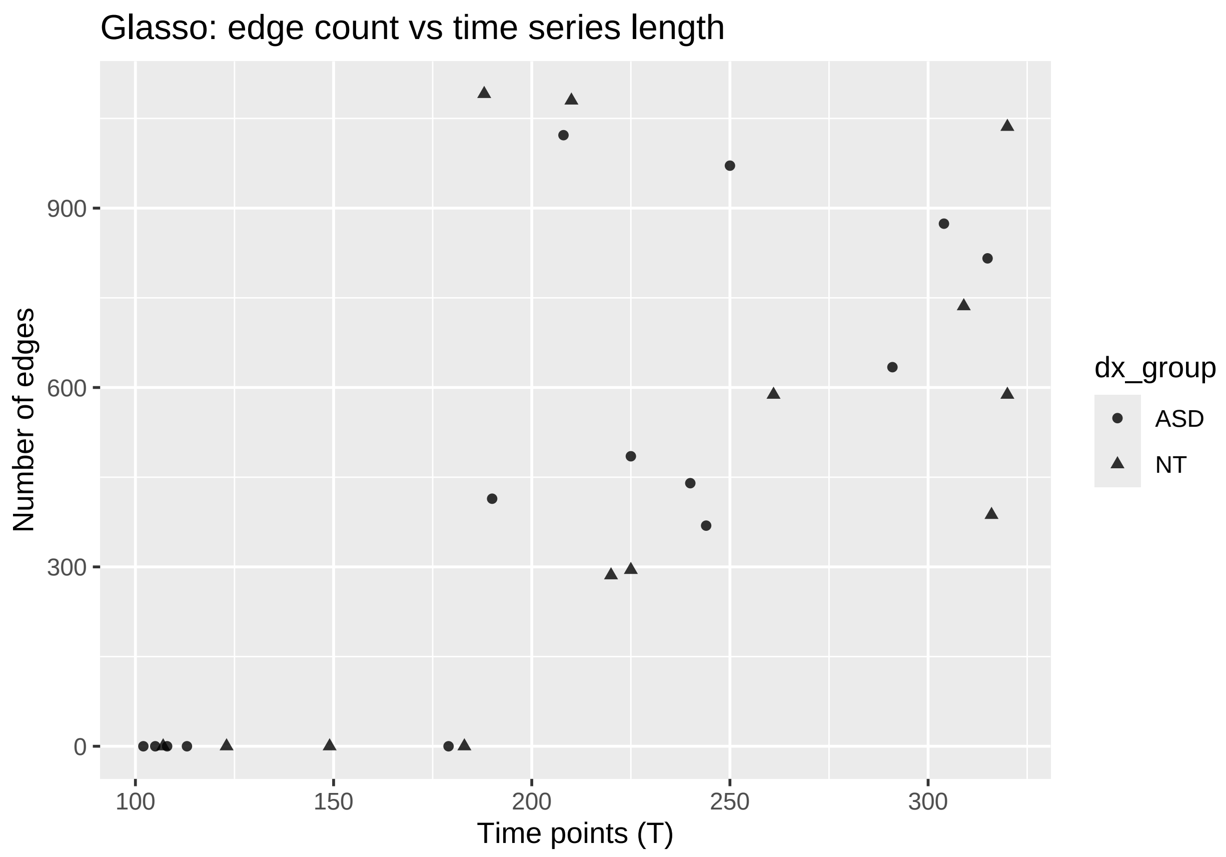
Step 4B: Check whether network summaries are associated with potential confounders: time series length (t_points), age, and sex.

I use Spearman correlations (robust) and simple visualizations.

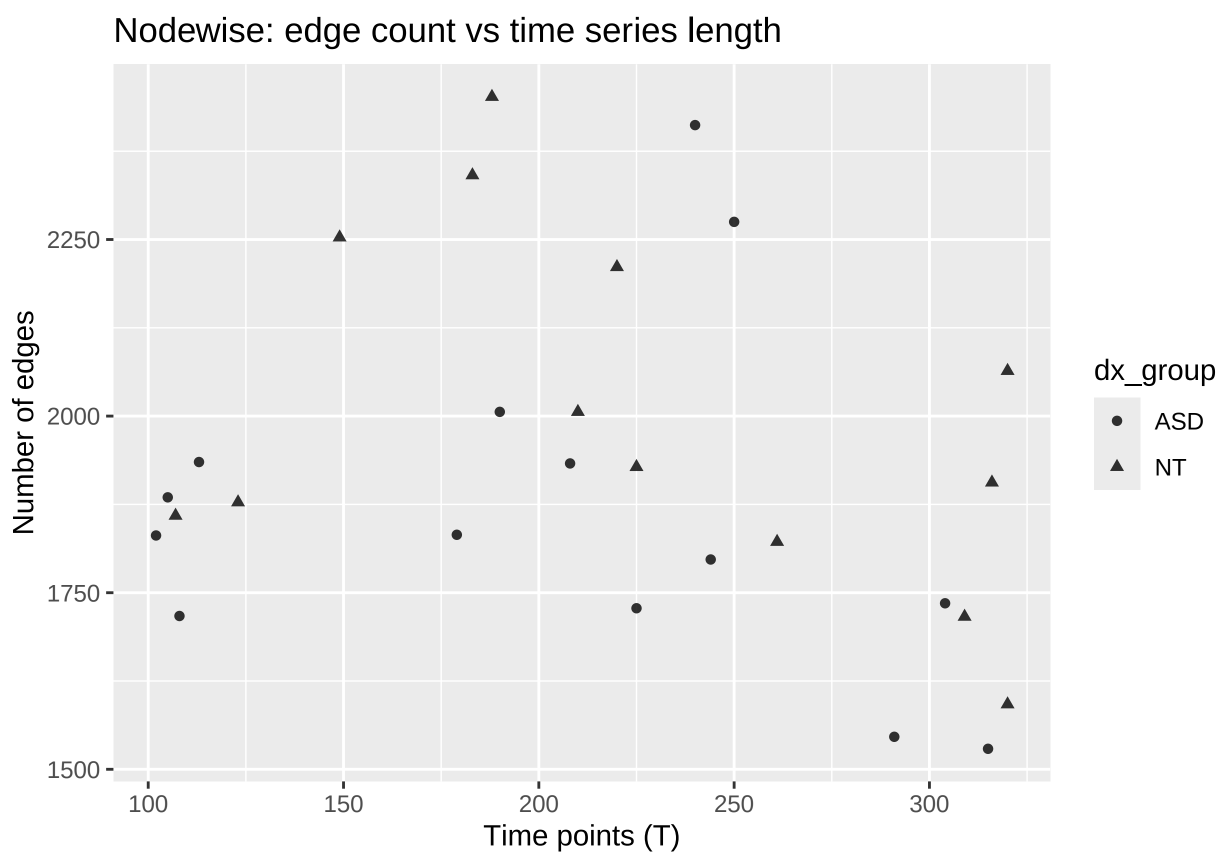
```r
# Load summaries
step2A_summary <- readRDS(file.path(cache_dir, "step2A_summary.rds"))
step2B_summary <- readRDS(file.path(cache_dir, "step2B_summary.rds"))

p_nodes <- 160
edge_density <- function(n_edges, p) n_edges / (p * (p - 1) / 2)

# Merge to a single subject-level table
df_all <- subject_index %>%
  transmute(
    sub_id = as.character(sub_id),
    dx_group = as.character(dx_group),
    sex = if ("sex" %in% names(subject_index)) as.character(sex) else NA_character_,
    age_at_scan = if ("age_at_scan" %in% names(subject_index)) as.numeric(age_at_scan) else NA_real_,
    t_points = as.numeric(t_points)
  ) %>%
  left_join(
    step2A_summary %>%
      transmute(
        sub_id = as.character(sub_id),
        edges_glasso = as.numeric(n_edges_glasso),
        dens_glasso = edge_density(edges_glasso, p_nodes),
        rho_best = as.numeric(rho_best)
      ),
    by = "sub_id"
  ) %>%
  left_join(
    step2B_summary %>%
      transmute(
        sub_id = as.character(sub_id),
        edges_nodewise = as.numeric(n_edges_nodewise),
        dens_nodewise = edge_density(edges_nodewise, p_nodes)
      ),
    by = "sub_id"
  ) %>%
  filter(dx_group %in% c("ASD", "NT"))

# Spearman correlation
spearman_tbl <- function(x, y, x_name, y_name) {
  ok <- is.finite(x) & is.finite(y)
  if (sum(ok) < 5) {
    return(tibble(x = x_name, y = y_name, rho = NA_real_, p_value = NA_real_, n = sum(ok)))
  }
  ct <- suppressWarnings(cor.test(x[ok], y[ok], method = "spearman", exact = FALSE))
  tibble(x = x_name, y = y_name, rho = unname(ct$estimate), p_value = ct$p.value, n = sum(ok))
}

# Run correlations: t_points vs edges/density and rho_best
corrs <- bind_rows(
  spearman_tbl(df_all$t_points, df_all$edges_glasso,   "t_points", "edges_glasso"),
  spearman_tbl(df_all$t_points, df_all$dens_glasso,    "t_points", "dens_glasso"),
  spearman_tbl(df_all$t_points, df_all$rho_best,       "t_points", "rho_best"),
  spearman_tbl(df_all$t_points, df_all$edges_nodewise, "t_points", "edges_nodewise"),
  spearman_tbl(df_all$t_points, df_all$dens_nodewise,  "t_points", "dens_nodewise"),
  spearman_tbl(df_all$age_at_scan, df_all$edges_glasso,   "age_at_scan", "edges_glasso"),
  spearman_tbl(df_all$age_at_scan, df_all$edges_nodewise, "age_at_scan", "edges_nodewise")
)

print(corrs)
```

```
## # A tibble: 7 x 5
##   x           y                  rho    p_value     n
##   <chr>       <chr>            <dbl>      <dbl> <int>
## 1 t_points    edges_glasso    0.683  0.0000861     27
## 2 t_points    dens_glasso     0.683  0.0000861     27
## 3 t_points    rho_best       -0.756  0.00000517    27
## 4 t_points    edges_nodewise -0.257  0.196         27
## 5 t_points    dens_nodewise  -0.257  0.196         27
## 6 age_at_scan edges_glasso   -0.0585 0.772         27
## 7 age_at_scan edges_nodewise -0.181  0.366         27
```

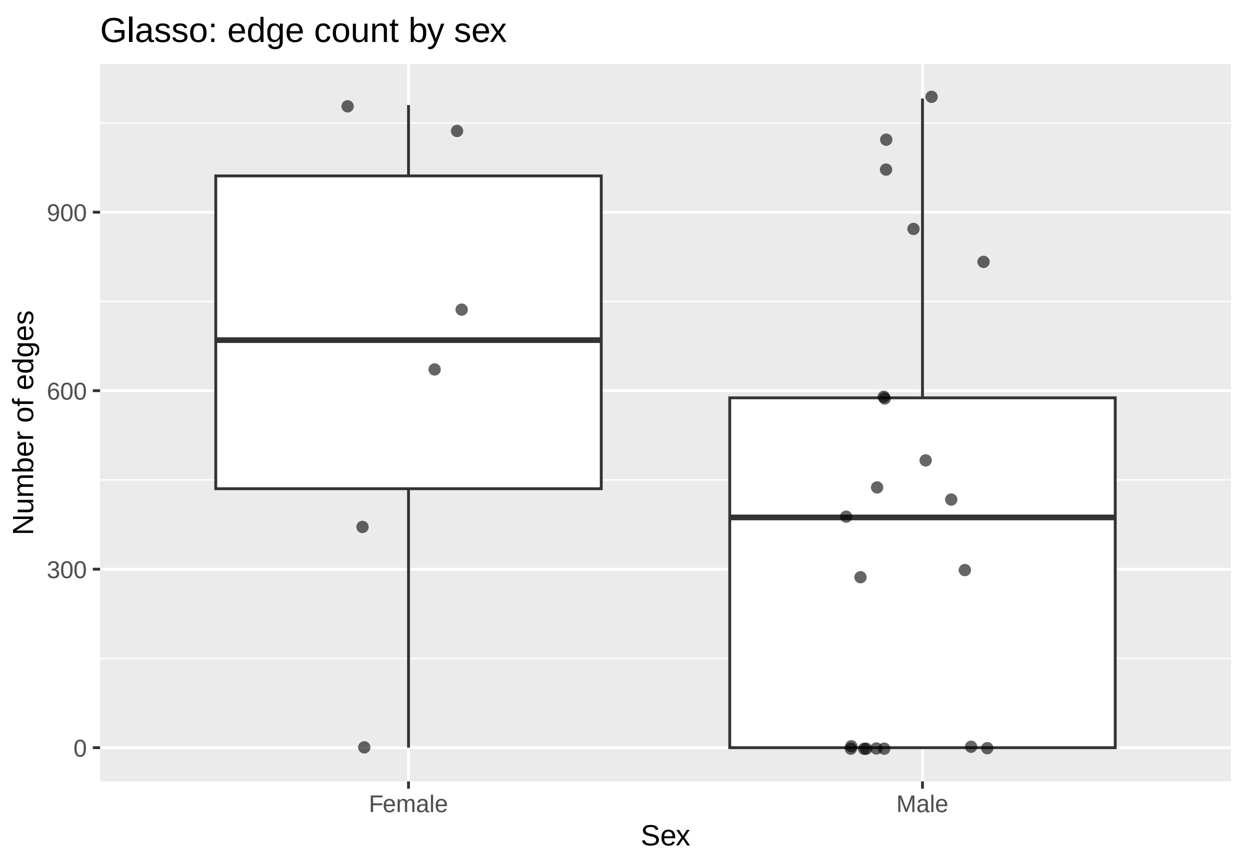
```r
# Visualizations
# t_points vs edges (glasso)
ggplot(df_all, aes(x = t_points, y = edges_glasso, shape = dx_group)) +
  geom_point(alpha = 0.8) +
  labs(
    title = "Glasso: edge count vs time series length",
    x = "Time points (T)",
    y = "Number of edges"
  )
```



```r
# t_points vs edges (nodewise)
ggplot(df_all, aes(x = t_points, y = edges_nodewise, shape = dx_group)) +
  geom_point(alpha = 0.8) +
  labs(
    title = "Nodewise: edge count vs time series length",
    x = "Time points (T)",
    y = "Number of edges"
  )
```



```r
# Sex (exploratory): visualize whether sparsity differs by sex
if (!all(is.na(df_all$sex))) {
  ggplot(df_all, aes(x = sex, y = edges_glasso)) +
    geom_boxplot(outlier.shape = NA) +
    geom_jitter(width = 0.15, alpha = 0.6) +
    labs(
      title = "Glasso: edge count by sex",
      x = "Sex",
      y = "Number of edges"
    )
}
```



Step 5: Address alternative explanations for ASD vs NT differences (Question ii).

Key concern: time series length (t_points) strongly affects glasso sparsity via EBIC-selected rho.

Therefore, we re-test group differences after adjusting for t_points (and optionally age/sex), using (A) regression models and (B) residualization + Wilcoxon as a robust nonparametric check.

Nodewise results are included as a robustness check because they are less sensitive to t_points.

```
step2A_summary <- readRDS(file.path(cache_dir, "step2A_summary.rds"))
step2B_summary <- readRDS(file.path(cache_dir, "step2B_summary.rds"))

stopifnot(exists("subject_index"))

# Build a unified analysis table
df <- subject_index %>%
  transmute(
    sub_id = as.character(sub_id),
    dx_group = dx_group,
    sex = sex,
```

```r
    age_at_scan = age_at_scan,
    t_points = t_points
  ) %>%
  left_join(
    step2A_summary %>%
      transmute(
        sub_id = as.character(sub_id),
        edges_glasso = n_edges_glasso,
        rho_best = rho_best
      ),
    by = "sub_id"
  ) %>%
  left_join(
    step2B_summary %>%
      transmute(
        sub_id = as.character(sub_id),
        edges_nodewise = n_edges_nodewise
      ),
    by = "sub_id"
  )

# Basic sanity checks
stopifnot(nrow(df) == 27)
stopifnot(all(!is.na(df$dx_group)))
stopifnot(all(!is.na(df$t_points)))
stopifnot(all(!is.na(df$edges_glasso)))
stopifnot(all(!is.na(df$edges_nodewise)))

# Residual-based group test:
# Fit y ~ covariates, extract residuals, then compare residuals by group using Wilcoxon.
residual_wilcox <- function(formula, data, group_var = "dx_group") {
  fit <- lm(formula, data = data)
  res <- resid(fit)
  g <- data[[group_var]]
  wt <- wilcox.test(res ~ g, exact = FALSE)

  tibble(
    formula = deparse(formula),
    n = nrow(data),
    W = unname(wt$statistic),
    p_value = wt$p.value
  )
}

# (A) Linear models controlling for t_points (and optionally age/sex)
# log1p stabilizes variance and reduces the influence of near-zero edge counts.
m_glasso_1 <- lm(log1p(edges_glasso) ~ dx_group + t_points, data = df)
m_node_1   <- lm(log1p(edges_nodewise) ~ dx_group + t_points, data = df)

# Extended models (complete cases only)
df2 <- df %>% filter(!is.na(sex), !is.na(age_at_scan))
m_glasso_2 <- lm(log1p(edges_glasso) ~ dx_group + t_points + age_at_scan + sex, data = df2)
m_node_2   <- lm(log1p(edges_nodewise) ~ dx_group + t_points + age_at_scan + sex, data = df2)
```

```r
cat("=== Model (glasso): log1p(edges) ~ dx_group + t_points ===\n")
```

```
## === Model (glasso): log1p(edges) ~ dx_group + t_points ===
```

```r
print(summary(m_glasso_1))
```

```
##
## Call:
## lm(formula = log1p(edges_glasso) ~ dx_group + t_points, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2280 -0.9066 -0.6604  1.2319  3.8651
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.982872   1.102671  -2.705   0.0124 *
## dx_groupNT  -0.409632   0.706699  -0.580   0.5676
## t_points     0.034697   0.004822   7.195 1.95e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.817 on 24 degrees of freedom
## Multiple R-squared:  0.684,  Adjusted R-squared:  0.6576
## F-statistic: 25.97 on 2 and 24 DF,  p-value: 9.922e-07
```

```r
cat("\n[Key coefficient] dx_group effect (glasso main):\n")
```

```
##
## [Key coefficient] dx_group effect (glasso main):
```

```r
print(coef(summary(m_glasso_1))["dx_groupNT", , drop = FALSE])
```

```
##              Estimate Std. Error    t value  Pr(>|t|)
## dx_groupNT -0.4096323  0.7066992 -0.5796417 0.5675582
```

```r
cat("\n=== Model (nodewise): log1p(edges) ~ dx_group + t_points ===\n")
```

```
##
## === Model (nodewise): log1p(edges) ~ dx_group + t_points ===
```

```r
print(summary(m_node_1))
```

```
##
## Call:
## lm(formula = log1p(edges_nodewise) ~ dx_group + t_points, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.17319 -0.08630 -0.01578  0.07823  0.28056
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.6316181  0.0741051 102.984   <2e-16 ***
## dx_groupNT   0.0796858  0.0474938   1.678    0.106
## t_points    -0.0005148  0.0003241  -1.589    0.125
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1221 on 24 degrees of freedom
## Multiple R-squared:  0.1636, Adjusted R-squared:  0.09387
## F-statistic: 2.347 on 2 and 24 DF,  p-value: 0.1173
```

```
cat("\n[Key coefficient] dx_group effect (nodewise main):\n")
```

```
##
## [Key coefficient] dx_group effect (nodewise main):
```

```
print(coef(summary(m_node_1))["dx_groupNT", , drop = FALSE])
```

```
##              Estimate Std. Error  t value  Pr(>|t|)
## dx_groupNT 0.07968578 0.04749383 1.677813 0.1063606
```

```
cat("\n=== Extended model (glasso): + age + sex (complete cases only) ===\n")
```

```
##
## === Extended model (glasso): + age + sex (complete cases only) ===
```

```
print(summary(m_glasso_2))
```

```
##
## Call:
## lm(formula = log1p(edges_glasso) ~ dx_group + t_points + age_at_scan +
##     sex, data = df2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1496 -0.9130 -0.6444  1.2605  3.8754
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.766552   2.045075  -1.353    0.190
## dx_groupNT  -0.404714   0.738087  -0.548    0.589
## t_points     0.034539   0.005179   6.669 1.05e-06 ***
## age_at_scan -0.008071   0.065530  -0.123    0.903
## sexFemale    0.134842   0.903113   0.149    0.883
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.896 on 22 degrees of freedom
## Multiple R-squared:  0.6845, Adjusted R-squared:  0.6272
## F-statistic: 11.93 on 4 and 22 DF,  p-value: 2.627e-05
```

```r
cat("\n[Key coefficient] dx_group effect (glasso extended):\n")
```

```
##
## [Key coefficient] dx_group effect (glasso extended):
```

```r
print(coef(summary(m_glasso_2))["dx_groupNT", , drop = FALSE])
```

```
##               Estimate Std. Error    t value  Pr(>|t|)
## dx_groupNT -0.4047144  0.7380871 -0.5483288 0.5889877
```

```r
cat("\n=== Extended model (nodewise): + age + sex (complete cases only) ===\n")
```

```
##
## === Extended model (nodewise): + age + sex (complete cases only) ===
```

```r
print(summary(m_node_2))
```

```
##
## Call:
## lm(formula = log1p(edges_nodewise) ~ dx_group + t_points + age_at_scan +
##     sex, data = df2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.19686 -0.07239 -0.01115  0.06072  0.22284
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.7673285  0.1179061  65.877   <2e-16 ***
## dx_groupNT   0.0807724  0.0425534   1.898   0.0709 .
## t_points    -0.0003306  0.0002986  -1.107   0.2801
## age_at_scan -0.0055542  0.0037780  -1.470   0.1557
## sexFemale   -0.1266143  0.0520678  -2.432   0.0236 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1093 on 22 degrees of freedom
## Multiple R-squared:  0.3855, Adjusted R-squared:  0.2738
## F-statistic:  3.45 on 4 and 22 DF,  p-value: 0.02472
```

```r
cat("\n[Key coefficient] dx_group effect (nodewise extended):\n")
```

```
##
## [Key coefficient] dx_group effect (nodewise extended):
```

```r
print(coef(summary(m_node_2))["dx_groupNT", , drop = FALSE])
```

```
##               Estimate Std. Error  t value   Pr(>|t|)
## dx_groupNT 0.08077245 0.04255344 1.898141 0.07087539
```

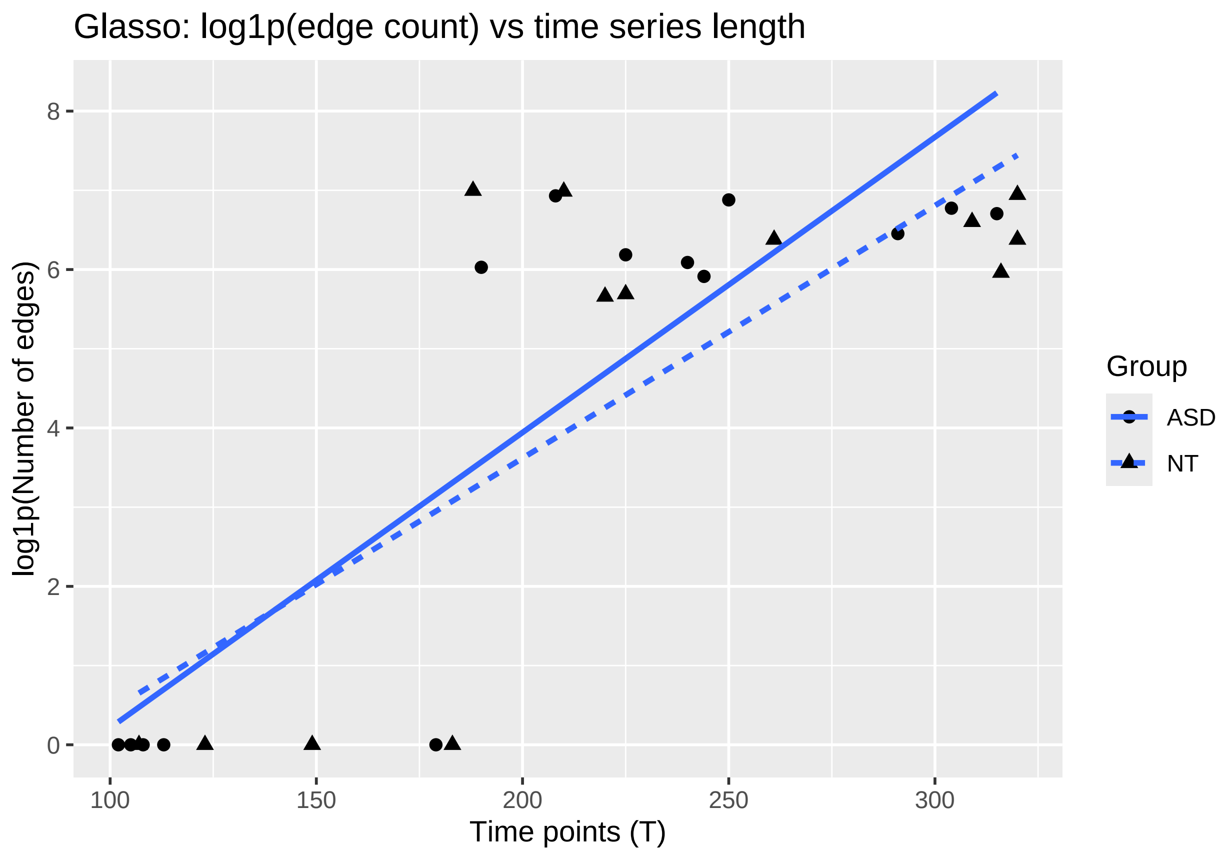
```r
# (B) Residualization + Wilcoxon test (nonparametric)
# Core adjustment (t_points only)
rw_glasso <- residual_wilcox(log1p(edges_glasso) ~ t_points, data = df)
rw_node   <- residual_wilcox(log1p(edges_nodewise) ~ t_points, data = df)

# Extended adjustment (t_points + age + sex) on complete cases
rw_glasso_ext <- residual_wilcox(log1p(edges_glasso) ~ t_points + age_at_scan + sex, data = df2)
rw_node_ext   <- residual_wilcox(log1p(edges_nodewise) ~ t_points + age_at_scan + sex, data = df2)

res_tests <- bind_rows(
  rw_glasso %>% mutate(method = "glasso", adjust = "t_points"),
  rw_node   %>% mutate(method = "nodewise", adjust = "t_points"),
  rw_glasso_ext %>% mutate(method = "glasso", adjust = "t_points + age + sex"),
  rw_node_ext   %>% mutate(method = "nodewise", adjust = "t_points + age + sex")
) %>%
  select(method, adjust, n, W, p_value, formula)

cat("\n=== Residual-based Wilcoxon tests (group effect after adjustment) ===\n")
```

```
##
## === Residual-based Wilcoxon tests (group effect after adjustment) ===
```

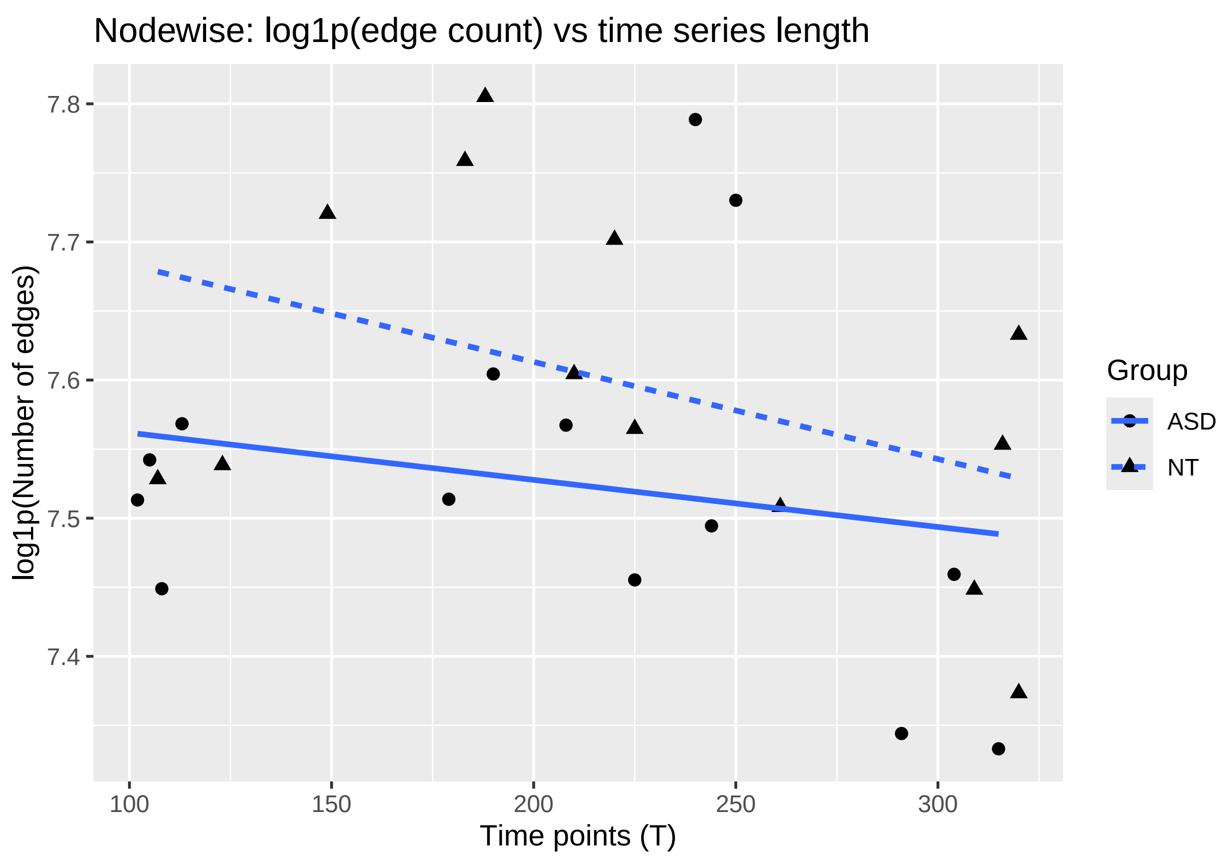
```r
print(res_tests)
```

```
## # A tibble: 4 x 6
##   method   adjust                  n     W p_value formula
##   <chr>    <chr>               <int> <dbl>   <dbl> <chr>
## 1 glasso   t_points               27   115  0.254  log1p(edges_glasso) ~ t_poi~
## 2 nodewise t_points               27    56  0.0941 log1p(edges_nodewise) ~ t_p~
## 3 glasso   t_points + age + sex    27   114  0.275  log1p(edges_glasso) ~ t_poi~
## 4 nodewise t_points + age + sex    27    51  0.0553 log1p(edges_nodewise) ~ t_p~
```

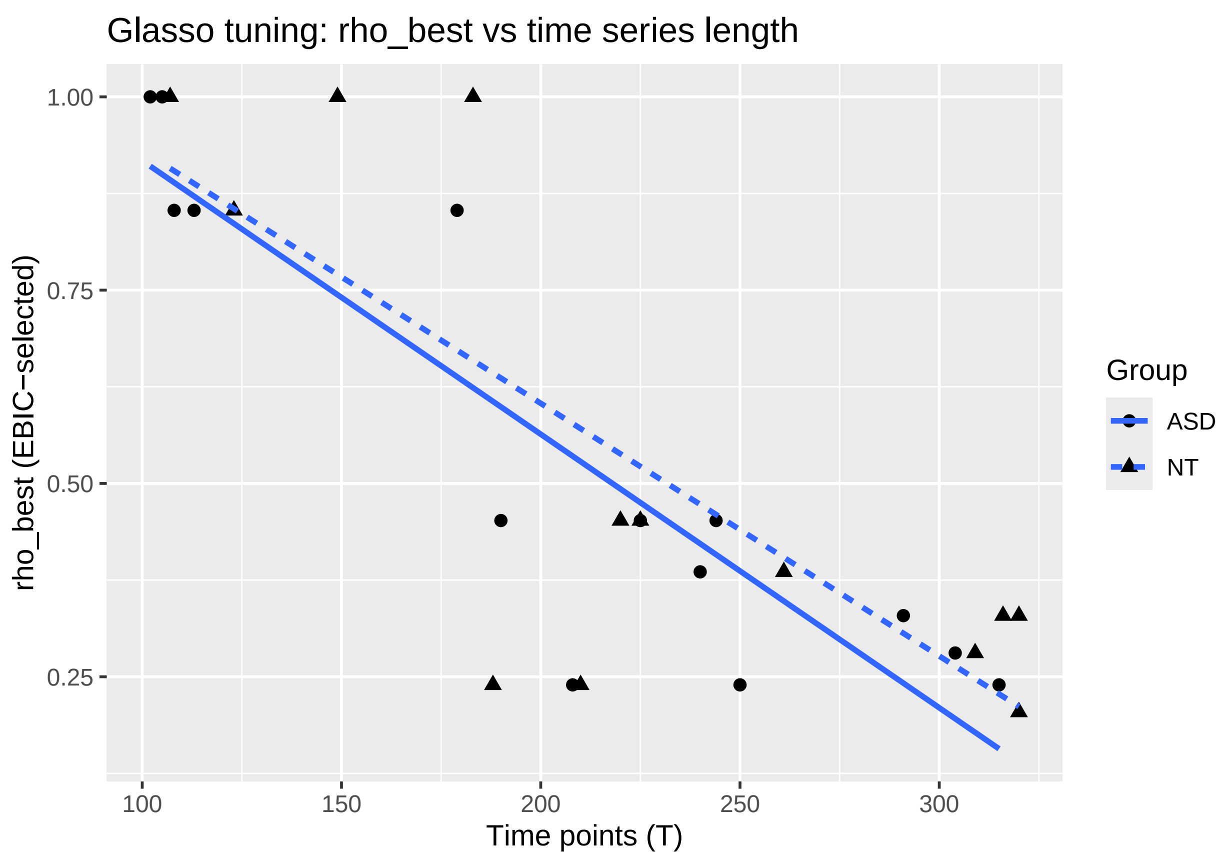
```r
# Visualization: log1p(edge counts) vs t_points with fitted lines per group
p1 <- ggplot(df, aes(x = t_points, y = log1p(edges_glasso), shape = dx_group)) +
  geom_point(size = 2) +
  geom_smooth(aes(linetype = dx_group), method = "lm", se = FALSE) +
  labs(
    title = "Glasso: log1p(edge count) vs time series length",
    x = "Time points (T)",
    y = "log1p(Number of edges)",
    shape = "Group",
    linetype = "Group"
  )

p2 <- ggplot(df, aes(x = t_points, y = log1p(edges_nodewise), shape = dx_group)) +
  geom_point(size = 2) +
  geom_smooth(aes(linetype = dx_group), method = "lm", se = FALSE) +
  labs(
    title = "Nodewise: log1p(edge count) vs time series length",
    x = "Time points (T)",
    y = "log1p(Number of edges)",
    shape = "Group",
    linetype = "Group"
  )

print(p1)
```



```
print(p2)
```



```r
# Diagnostic for glasso selection: rho_best ~ t_points by group
p3 <- ggplot(df, aes(x = t_points, y = rho_best, shape = dx_group)) +
  geom_point(size = 2) +
  geom_smooth(aes(linetype = dx_group), method = "lm", se = FALSE) +
  labs(
    title = "Glasso tuning: rho_best vs time series length",
    x = "Time points (T)",
    y = "rho_best (EBIC-selected)",
    shape = "Group",
    linetype = "Group"
  )

print(p3)
```



```r
# Save a compact results object for the report
step5_results <- list(
  df = df,
  model_glasso_main = summary(m_glasso_1),
  model_nodewise_main = summary(m_node_1),
  model_glasso_ext = summary(m_glasso_2),
  model_nodewise_ext = summary(m_node_2),
  residual_tests = res_tests
)

saveRDS(step5_results, file.path(cache_dir, "step5_adjustment_results.rds"))
```