Aging\_Proteomics

Dustyn Lewis

2025-04-23

Table of Contents

## SECTION 0: Load Required Libraries

*Purpose*

* Define all of the R packages needed for my analysis, install any that are missing (distinguishing between CRAN and Bioconductor packages), and then loads them into the session.

# ---- 0. Load Required Libraries ---- # Section header  
# Define and install required libraries  
required\_packages <- c(  
 "readxl", # reading Excel files  
 "dplyr", # data manipulation  
 "stringr", # string operations  
 "tidyr", # data tidying  
 "httr", # HTTP requests  
 "jsonlite", # JSON parsing  
 "purrr", # functional programming tools  
 "tibble", # modern data frame structure  
 "broom", # tidying statistical model output  
 "ReactomePA", # pathway analysis (Bioconductor)  
 "clusterProfiler",# enrichment analysis (Bioconductor)  
 "org.Hs.eg.db", # human gene annotation database (Bioconductor)  
 "enrichplot", # plotting enrichment results (Bioconductor)  
 "ggplot2", # data visualization  
 "forcats", # factor level manipulation  
 "fmsb", # utilities for radar charts and more  
 "BiocManager" # managing Bioconductor installations  
)  
  
# Install missing libraries  
for (pkg in required\_packages) {  
 if (!requireNamespace(pkg, quietly = TRUE)) {  
 if (pkg %in% c("ReactomePA", "clusterProfiler", "org.Hs.eg.db", "enrichplot")) {  
 BiocManager::install(pkg, update = FALSE)  
 } else {  
 install.packages(pkg)  
 }  
 }  
}

##

##

# Load libraries  
invisible(lapply(required\_packages, library, character.only = TRUE))

## Warning: package 'readxl' was built under R version 4.4.3

## Warning: package 'dplyr' was built under R version 4.4.3

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

## Warning: package 'jsonlite' was built under R version 4.4.3

##   
## Attaching package: 'purrr'

## The following object is masked from 'package:jsonlite':  
##   
## flatten

## Warning: package 'broom' was built under R version 4.4.3

## ReactomePA v1.50.0 Learn more at https://yulab-smu.top/contribution-knowledge-mining/  
##   
## Please cite:  
##   
## Guangchuang Yu, Qing-Yu He. ReactomePA: an R/Bioconductor package for  
## reactome pathway analysis and visualization. Molecular BioSystems.  
## 2016, 12(2):477-479

## clusterProfiler v4.14.6 Learn more at https://yulab-smu.top/contribution-knowledge-mining/  
##   
## Please cite:  
##   
## S Xu, E Hu, Y Cai, Z Xie, X Luo, L Zhan, W Tang, Q Wang, B Liu, R Wang,  
## W Xie, T Wu, L Xie, G Yu. Using clusterProfiler to characterize  
## multiomics data. Nature Protocols. 2024, 19(11):3292-3320

##   
## Attaching package: 'clusterProfiler'

## The following object is masked from 'package:purrr':  
##   
## simplify

## The following object is masked from 'package:stats':  
##   
## filter

## Loading required package: AnnotationDbi

## Loading required package: stats4

## Loading required package: BiocGenerics

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:dplyr':  
##   
## combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, aperm, append, as.data.frame, basename, cbind,  
## colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,  
## get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,  
## match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,  
## Position, rank, rbind, Reduce, rownames, sapply, saveRDS, setdiff,  
## table, tapply, union, unique, unsplit, which.max, which.min

## Loading required package: Biobase

## Welcome to Bioconductor  
##   
## Vignettes contain introductory material; view with  
## 'browseVignettes()'. To cite Bioconductor, see  
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

##   
## Attaching package: 'Biobase'

## The following object is masked from 'package:httr':  
##   
## content

## Loading required package: IRanges

## Loading required package: S4Vectors

##   
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:clusterProfiler':  
##   
## rename

## The following object is masked from 'package:tidyr':  
##   
## expand

## The following objects are masked from 'package:dplyr':  
##   
## first, rename

## The following object is masked from 'package:utils':  
##   
## findMatches

## The following objects are masked from 'package:base':  
##   
## expand.grid, I, unname

##   
## Attaching package: 'IRanges'

## The following object is masked from 'package:clusterProfiler':  
##   
## slice

## The following object is masked from 'package:purrr':  
##   
## reduce

## The following objects are masked from 'package:dplyr':  
##   
## collapse, desc, slice

## The following object is masked from 'package:grDevices':  
##   
## windows

##   
## Attaching package: 'AnnotationDbi'

## The following object is masked from 'package:clusterProfiler':  
##   
## select

## The following object is masked from 'package:dplyr':  
##   
## select

## enrichplot v1.26.6 Learn more at https://yulab-smu.top/contribution-knowledge-mining/  
##   
## Please cite:  
##   
## S Xu, E Hu, Y Cai, Z Xie, X Luo, L Zhan, W Tang, Q Wang, B Liu, R Wang,  
## W Xie, T Wu, L Xie, G Yu. Using clusterProfiler to characterize  
## multiomics data. Nature Protocols. 2024, 19(11):3292-3320

## Warning: package 'ggplot2' was built under R version 4.4.3

## Warning: package 'fmsb' was built under R version 4.4.3

## Warning: package 'BiocManager' was built under R version 4.4.3

## Bioconductor version '3.20' is out-of-date; the current release version '3.21'  
## is available with R version '4.5'; see https://bioconductor.org/install

*Step-by-step Breakdown*

required\_packages <- c(...)

Creates a character vector required\_packages containing the names of all packages your script will use.

for (pkg in required\_packages) { … }

Starts a loop that goes through each package name stored in required\_packages.

if (!requireNamespace(pkg, quietly = TRUE)) { … }

Checks whether the package pkg is already installed (without loading it). If not installed, proceeds to the next step.

if (pkg %in% c("ReactomePA", "clusterProfiler", "org.Hs.eg.db", "enrichplot")) { BiocManager::install(pkg, update = FALSE) }

Installs Bioconductor packages via BiocManager::install() without updating other packages.

else { install.packages(pkg) }

Installs CRAN packages with install.packages() if they’re not among the specified Bioconductor set.

invisible(lapply(required\_packages, library, character.only = TRUE))

Loads each package into the R session using library(). Wrapping in invisible() suppresses the usual startup messages.

## SECTION 1: Data Loading & Cleaning Pipeline

*Purpose*

* Read in the proteomics data from Excel, remove any rows or columns that are entirely NA, and then parse out the gene symbol and protein accession from a identifier column. Then, compute a simple “detection score” by summing up presence/abundance values across all sample columns.

# ---- 1. Load & Clean Data ----  
# Load data from Excel, remove empty rows/columns, and extract gene/protein details  
sarco <- read\_excel(  
 "13\_MASTER\_SEER proteome analysis (4-4-23).xlsx",  
 sheet = "MASTER\_SARCO\_data"  
) %>%  
 # Keep only columns where at least one value is non-NA  
 dplyr::select(where(~ any(!is.na(.)))) %>%  
 # Remove any rows that are entirely NA  
 filter(if\_any(everything(), ~ !is.na(.))) %>%  
 # Parse out Gene\_Symbol and Accession from the Gene\_protein column  
 mutate(  
 Gene\_Symbol = str\_remove(  
 str\_extract(Gene\_protein, "^[^\_;]+"),  
 ";?GN=+$"  
 ),  
 Accession = str\_extract(  
 Gene\_protein,  
 "\\b[A-Z][0-9][A-Z0-9]{3}[0-9]\\b"  
 )  
 )  
  
# Identify which columns correspond to sample measurements  
sample\_cols <- grep("^(EAA|PRE|POST|PPS)", names(sarco), value = TRUE)  
  
# Compute a detection score by summing across all sample columns  
sarco <- sarco %>%  
 mutate(  
 detection\_score = rowSums(  
 across(all\_of(sample\_cols)),  
 na.rm = TRUE  
 )  
 )

*Step-by-Step Breakdown*

sarco <- read\_excel(..., sheet = "MASTER\_SARCO\_data")

Uses readxl’s read\_excel() to import the specified worksheet into a data frame called sarco.

dplyr::select(where(~ any(!is.na(.))))

Selects only those columns in which at least one entry is not NA, effectively dropping empty columns.

filter(if\_any(everything(), ~ !is.na(.)))

Keeps only rows where any column has a non-NA value, removing fully empty rows.

1. mutate( Gene\_Symbol = ..., Accession = ... )

* str\_extract(Gene\_protein, "^[^\_;]+") pulls the first chunk before any semicolon or underscore from Gene\_protein.
* str\_remove(..., ";?GN=+$") then strips any trailing “;GN=” patterns to clean the gene symbol.
* str\_extract(..., "\\b[A-Z][0-9][A-Z0-9]{3}[0-9]\\b") captures the UniProt-style accession (e.g., “P12345”).

1. sample\_cols <- grep("^(EAA|PRE|POST|PPS)", names(sarco), value = TRUE)

* Uses base R’s grep() to find all column names starting with “EAA”, “PRE”, “POST”, or “PPS”, storing them for later.

mutate(detection\_score = rowSums(across(all\_of(sample\_cols)), na.rm = TRUE))

With dplyr’s across(), selects the sample columns, then applies base R’s rowSums() (ignoring NA) to give each row a total count or abundance score.

*Key Packages*

readxl: Importing Excel (XLSX) data directly into R without Java dependencies.

dplyr: Core tools for data frame manipulation—selecting, filtering, mutating, and piping (%>%).

stringr: Convenient functions for regex-based string extraction and removal (str\_extract(), str\_remove()).

## SECTION 2: Query UniProt for Missing Gene Symbols

*Purpose*

* Define a function to retrieve missing gene symbols by querying the UniProt REST API for a list of protein accession IDs, and then apply this function to those accessions in my dataset that lack an assigned Gene\_Symbol.

# ---- 2. Query UniProt for Missing Gene Symbols ----  
  
# Define UniProt query function  
query\_uniprot <- function(accessions) {  
 url <- "https://rest.uniprot.org/uniprotkb/search"  
 # Construct OR-separated query string from the vector of accession IDs  
 q <- paste(accessions, collapse = " OR ")  
 # Perform GET request: ask for accession and primary gene name, JSON format, up to 500 entries  
 r <- GET(  
 url,  
 query = list(  
 query = q,  
 fields = "accession,gene\_primary",  
 format = "json",  
 size = 500  
 )  
 )  
 # If the request is successful, parse JSON and return a tibble of results  
 if (status\_code(r) == 200) {  
 j <- httr::content(r, as = "parsed", type = "application/json")  
 tibble(  
 Accession = sapply(j$results, function(x) x$primaryAccession),  
 Gene\_Symbol\_API = sapply(  
 j$results,  
 function(x)  
 if (!is.null(x$genes[[1]]$geneName$value))  
 x$genes[[1]]$geneName$value  
 else  
 NA\_character\_  
 )  
 )  
 } else {  
 stop("UniProt query failed")  
 }  
}  
  
# Identify accessions requiring a UniProt query:  
# 1. For each (Accession, existing Gene\_Symbol) pair, keep the row with highest detection\_score  
# 2. Filter those with missing or empty Gene\_Symbol  
# 3. Extract unique accession IDs  
accessions\_to\_query <- sarco %>%  
 group\_by(Accession, Gene\_Symbol) %>%  
 slice\_max(detection\_score, n = 1, with\_ties = FALSE) %>%  
 ungroup() %>%  
 filter(is.na(Gene\_Symbol) | Gene\_Symbol == "") %>%  
 pull(Accession) %>%  
 unique() %>%  
 na.omit()  
  
# Break the list into chunks of 100 accessions, query UniProt for each chunk,  
# then row-bind all results and remove duplicate accessions  
uniprot\_results <- accessions\_to\_query %>%  
 split(ceiling(seq\_along(.) / 100)) %>%  
 map\_dfr(query\_uniprot) %>%  
 distinct(Accession, .keep\_all = TRUE)

*Step-by-Step Breakdown*

1. query\_uniprot <- function(accessions) { … }

Defines a reusable function taking a character vector of UniProt accession IDs.

q <- paste(accessions, collapse = " OR ")

Joins IDs into a single search string where each ID is separated by OR, matching the UniProt query syntax.

r <- GET(url, query = list(...))

Uses httr’s GET() to call the UniProt REST endpoint, requesting only accession and gene\_primary fields in JSON form, up to 500 records.

if (status\_code(r) == 200) { … } else stop("UniProt query failed")

Checks for a successful HTTP 200 response; otherwise stops execution with an error.

j <- httr::content(r, as = "parsed", type = "application/json")

Parses the JSON content into an R list for further processing.

tibble(Accession = sapply(j$results, …), Gene\_Symbol\_API = sapply(j$results, …))

Iterates through jvalue (if present) into a two‐column tibble.

1. accessions\_to\_query <- sarco %>% group\_by(...) %>% slice\_max(...) %>% filter(...) %>% pull(...) %>% unique() %>% na.omit()

* group\_by(Accession, Gene\_Symbol) then slice\_max(detection\_score, n = 1) ensures one representative row per accession/symbol.
* filter(is.na(Gene\_Symbol) | Gene\_Symbol == "") selects only those with missing symbols.
* pull(Accession) extracts the vector of IDs, then unique() and na.omit() clean it.

1. split(ceiling(seq\_along(.) / 100))

Splits the accession vector into sublists of up to 100 IDs each, to respect potential API size limits.

map\_dfr(query\_uniprot)

Applies query\_uniprot() to each sublist and row‐binds (\_dfr) all returned tibbles.

distinct(Accession, .keep\_all = TRUE)

Removes any duplicate rows by accession, retaining the first occurrence of each.

*Key Packages*

httr: Facilitates HTTP requests (here, GET()) and response handling for REST APIs.

jsonlite: Underlies JSON parsing; though here httr::content() is used directly, jsonlite is a dependency for parsing.

tibble: Creates tidy, modern data frames for the API results.

purrr: Provides map\_dfr() to iterate over list elements and combine results efficiently.

dplyr: Used for data wrangling—grouping, filtering, pulling columns, and chaining operations via the pipe.

## SECTION 3: Merge and Clean Gene Symbols

*Purpose*

* Join the UniProt‐derived gene symbols back into my main sarco (sarcoplasm) table, uses them to fill any missing or empty original symbols, strip out any “isoform\_” prefixes, and then drop the temporary API column.

# ---- 3. Merge and Clean Gene Symbols ----  
# Merge UniProt results and clean up gene symbols  
sarco <- sarco %>%  
 # Add Gene\_Symbol\_API from the UniProt query by matching on Accession  
 left\_join(uniprot\_results, by = "Accession") %>%  
 # Replace missing/empty Gene\_Symbol with the API‐provided value,  
 # then remove any leading "isoform\_" text  
 mutate(  
 Gene\_Symbol = if\_else(  
 is.na(Gene\_Symbol) | Gene\_Symbol == "",  
 Gene\_Symbol\_API,  
 Gene\_Symbol  
 ),  
 Gene\_Symbol = str\_remove(Gene\_Symbol, "^isoform\_")  
 ) %>%  
 # Drop the now‐redundant API column  
 dplyr::select(-Gene\_Symbol\_API)

*Step-by-Step Breakdown*

left\_join(uniprot\_results, by = "Accession")

Performs a left join of sarco with uniprot\_results on the Accession column, adding a new column Gene\_Symbol\_API containing the API query results for each accession.

mutate(Gene\_Symbol = if\_else(...))

Uses dplyr’s if\_else() to update Gene\_Symbol: if the existing symbol is NA or an empty string, it replaces it with Gene\_Symbol\_API; otherwise, it leaves the original symbol intact.

mutate(Gene\_Symbol = str\_remove(...))

Applies stringr’s str\_remove() with the regex ^isoform\_ to strip off any “isoform\_” prefix from the updated gene symbols.

dplyr::select(-Gene\_Symbol\_API)

Drops the helper column Gene\_Symbol\_API, since its values have been merged into Gene\_Symbol and are no longer needed.

*Key Packages*

dplyr: For data manipulation verbs (left\_join(), mutate(), select()) and piping.

stringr: For regex‐based string cleaning with str\_remove().

## SECTION 4: Remove Duplicates by Detection Score

*Purpose*

* De-duplicate the sarco dataset by first retaining the entry with the highest detection score for each unique Accession–Gene combination, and then ensuring that each Gene\_Symbol appears only once by keeping its highest‐scoring record.

# ---- 4. Remove Duplicates by Detection Score ----  
sarco\_clean <- sarco %>%  
 # For each (Accession, Gene\_Symbol) pair, keep only the row with the highest detection\_score  
 group\_by(Accession, Gene\_Symbol) %>%  
 slice\_max(detection\_score, n = 1, with\_ties = FALSE) %>%  
 ungroup() %>%  
 # Then, for each Gene\_Symbol (across any accessions), keep only the row with the highest detection\_score  
 group\_by(Gene\_Symbol) %>%  
 slice\_max(detection\_score, n = 1, with\_ties = FALSE) %>%  
 ungroup()

*Step-by-Step Breakdown*

group\_by(Accession, Gene\_Symbol)

Uses dplyr to create groups defined by each unique combination of Accession and Gene\_Symbol.

1. slice\_max(detection\_score, n = 1, with\_ties = FALSE)

* Within each group, selects the single row with the maximum detection\_score.
* n = 1 ensures only one row is kept; with\_ties = FALSE breaks ties by arbitrarily dropping extras.

1. ungroup()

Removes the current grouping structure so subsequent operations are applied to the whole data frame.

group\_by(Gene\_Symbol)

Regroups the pruned dataset by Gene\_Symbol alone, so that each gene symbol’s entries (potentially from multiple accessions) form a group.

slice\_max(detection\_score, n = 1, with\_ties = FALSE)

Again, selects the top detection\_score row within each Gene\_Symbol group, ensuring one representative per gene.

ungroup()

Finalizes the operation by removing grouping, yielding the cleaned sarco\_clean data frame.

*Key Packages*

dplyr: Provides the chaining (%>%), grouping (group\_by()), slice-based selection (slice\_max()), and ungrouping (ungroup()) verbs used to filter and de-duplicate the data.

## SECTION 5: Convert to Long Format

*Purpose*

* Reshape the de-duplicated, wide‐format proteomics data into a long‐format table suitable for downstream analysis, add metadata columns for timepoint and experimental group based on sample names, and then save both the long and wide cleaned datasets.

# ---- 5. Convert to Long Format ----  
  
long\_df <- sarco\_clean %>%  
 # Drop any leftover statistical summary or helper columns  
 dplyr::select(-matches("^T-test\_|^DELTA\_|^CORREL\_|^Young\_|^detection\_")) %>%  
 # Pivot the sample measurement columns into two columns: Sample (name) and Abundance (value)  
 pivot\_longer(  
 cols = all\_of(sample\_cols),  
 names\_to = "Sample",  
 values\_to = "Abundance"  
 ) %>%  
 # Exclude any rows where Sample refers to averages or standard deviations  
 filter(!str\_detect(Sample, "\_Avg|\_StdDev")) %>%  
 # Derive new metadata columns from the Sample names  
 mutate(  
 Timepoint = case\_when(  
 str\_detect(Sample, "PRE") ~ "Pre", # Samples containing "PRE"  
 str\_detect(Sample, "POST") ~ "Post", # Samples containing "POST"  
 TRUE ~ NA\_character\_  
 ),  
 Group = case\_when(  
 str\_detect(Sample, "EAA|PPS") ~ "Young", # Young group prefixes  
 TRUE ~ "MA" # Otherwise, Middle‐Aged  
 )  
 )  
  
# Save the reshaped long and the cleaned wide datasets  
write.csv(long\_df, "Clean\_Long2.csv", row.names = FALSE)  
write.csv(sarco\_clean, "Clean\_Wide2.csv", row.names = FALSE)

*Step-by-Step Breakdown*

dplyr::select(-matches("^T-test\_|^DELTA\_|^CORREL\_|^Young\_|^detection\_"))

Removes any columns whose names begin with summary prefixes (e.g. “T-test\_”, “DELTA\_”) or the original detection score, keeping only raw sample measurements and annotation fields.

pivot\_longer(cols = all\_of(sample\_cols), names\_to = "Sample", values\_to = "Abundance")

Uses tidyr to transform the wide set of sample columns into a long format, creating one row per sample measurement with the column name in Sample and its value in Abundance.

filter(!str\_detect(Sample, "\_Avg|\_StdDev"))

Excludes rows where the Sample name indicates an average or standard deviation column (e.g., “\_Avg” or “\_StdDev”) via stringr’s str\_detect().

1. mutate(Timepoint = case\_when(...), Group = case\_when(...))

* Adds a Timepoint column labeling each row as “Pre” or “Post” based on whether Sample contains “PRE” or “POST”.
* Adds a Group column labeling samples as “Young” if the name contains “EAA” or “PPS”, otherwise “MA” for middle‐aged.

1. write.csv()

Exports long\_df and the cleaned wide data frame sarco\_clean to CSV files for future use.

*Key Packages*

dplyr: Chaining operations (%>%), selecting/deselecting columns, filtering rows, and creating new columns (mutate).

tidyr: Reshaping data with pivot\_longer().

stringr: String pattern matching with str\_detect() to identify and exclude unwanted rows and to derive metadata.

utils (base R): write.csv() for writing data frames to CSV files.

## SECTION 6: Summary Table of Mean Abundances

*Purpose*

* Compute the average protein abundance for each gene across the defined experimental groups (“Young”, “Pre\_MA”, “Post\_MA”), reshape the results into a wide summary table, and save it as a CSV for later (could use for something else).

# ---- 6. Summary Table: Mean Abundances ----  
  
mean\_abundance\_summary <- long\_df %>%  
 # Define a new grouping variable based on Timepoint and Group  
 mutate(  
 Grouping = case\_when(  
 Group == "Young" ~ "Young", # Young subjects  
 Group == "MA" & Timepoint == "Pre" ~ "Pre\_MA", # Middle-aged Pre-training  
 Group == "MA" & Timepoint == "Post" ~ "Post\_MA", # Middle-aged Post-training  
 TRUE ~ NA\_character\_  
 )  
 ) %>%  
 # Exclude any rows where we couldn't assign a valid grouping  
 filter(!is.na(Grouping)) %>%  
 # Calculate the mean abundance for each gene in each grouping  
 group\_by(Gene\_Symbol, Grouping) %>%  
 summarise(  
 Mean\_Abundance = mean(Abundance, na.rm = TRUE),  
 .groups = "drop"  
 ) %>%  
 # Pivot the summary so each grouping becomes its own column  
 pivot\_wider(  
 names\_from = Grouping,  
 values\_from = Mean\_Abundance  
 )  
  
# Write the summary table to disk  
write.csv(  
 mean\_abundance\_summary,  
 "Group\_Abundance.csv",  
 row.names = FALSE  
)

*Step-by-Step Breakdown*

1. mutate(Grouping = case\_when(...)) Creates a new Grouping column:

* “Young” for samples labeled as young subjects.
* “Pre\_MA” for middle-aged pre-training samples.
* “Post\_MA” for middle-aged post-training samples.
* NA otherwise.

filter(!is.na(Grouping)) Drops any rows where Grouping is NA, ensuring only valid groups remain.

group\_by(Gene\_Symbol, Grouping) Organizes the data into groups defined by gene symbol and the new grouping variable.

summarise(Mean\_Abundance = mean(Abundance, na.rm = TRUE), .groups = "drop") Calculates the average Abundance within each gene–group pair, removing grouping afterwards (.groups = “drop”).

pivot\_wider(names\_from = Grouping, values\_from = Mean\_Abundance) Converts the long summary into a wide format, with one column per grouping (e.g., Young, Pre\_MA, Post\_MA).

write.csv(mean\_abundance\_summary, "Group\_Abundance.csv", row.names = FALSE) Exports the resulting summary table to a CSV file without row names.

*Key Packages*

dplyr: For data transformation verbs (mutate(), filter(), group\_by(), summarise()) and piping.

tidyr: For reshaping data with pivot\_wider().

utils (base R): Provides write.csv() to export data frames to CSV files.

## SECTION 7: Compute log2FC and Per-Gene Stats

*Purpose*

* Calculate log2 fold-changes between experimental groups for each protein and then perform statistical tests (t-tests) comparing Pre\_MA vs Young, Post\_MA vs Young, and Post\_MA vs Pre\_MA. Then, adjust p-values for multiple testing and merges fold-change and statistics into one table.

# ---- 7. Compute log2FC and Per-Gene Stats ----  
  
 # 1. Calculate log2 fold-change from mean abundances  
fc\_df <- mean\_abundance\_summary %>%  
 mutate(  
 log2FC\_Pre\_vs\_Young = log2(Pre\_MA / Young), # Pre-training MA vs Young  
 log2FC\_Post\_vs\_Young = log2(Post\_MA / Young), # Post-training MA vs Young  
 log2FC\_Post\_vs\_Pre\_MA = log2(Post\_MA / Pre\_MA) # Post vs Pre within MA  
 )  
  
 # 2. For each gene, run three t-tests and collect p-values  
stat\_df <- long\_df %>%  
 group\_by(Gene\_Symbol) %>% # operate within each gene  
 do({  
 df <- . # capture the per-gene subset  
  
 # Pre\_MA vs Young comparison  
 df1 <- df %>%   
 filter((Group == "MA" & Timepoint == "Pre") | Group == "Young")  
 p1 <- if (n\_distinct(df1$Group) == 2)   
 t.test(Abundance ~ Group, data = df1)$p.value   
 else NA\_real\_  
  
 # Post\_MA vs Young comparison  
 df2 <- df %>%   
 filter((Group == "MA" & Timepoint == "Post") | Group == "Young")  
 p2 <- if (n\_distinct(df2$Group) == 2)   
 t.test(Abundance ~ Group, data = df2)$p.value   
 else NA\_real\_  
  
 # Post\_MA vs Pre\_MA within MA  
 df3 <- df %>%   
 filter(Group == "MA", Timepoint %in% c("Pre", "Post"))  
 p3 <- if (n\_distinct(df3$Timepoint) == 2)   
 t.test(Abundance ~ Timepoint, data = df3)$p.value   
 else NA\_real\_  
  
 # Return a one-row tibble of p-values  
 tibble(  
 p\_Pre\_vs\_Young = p1,  
 p\_Post\_vs\_Young = p2,  
 p\_Post\_vs\_Pre\_MA = p3  
 )  
 }) %>%  
 ungroup() %>% # remove grouping  
 mutate(  
 # 3. Adjust all three sets of p-values using the FDR method  
 adj\_Pre\_vs\_Young = p.adjust(p\_Pre\_vs\_Young, method = "fdr"),  
 adj\_Post\_vs\_Young = p.adjust(p\_Post\_vs\_Young, method = "fdr"),  
 adj\_Post\_vs\_Pre\_MA = p.adjust(p\_Post\_vs\_Pre\_MA, method = "fdr")  
 )  
  
 # 4. Merge fold-change and statistics into a single data frame  
gene\_stats <- fc\_df %>%  
 left\_join(stat\_df, by = "Gene\_Symbol")

*Step-by-Step Breakdown*

mutate(log2FC\_... = log2(...))

Computes the log₂ fold-change for each comparison by taking the ratio of mean abundances.

group\_by(Gene\_Symbol)

Splits the long-format data into subsets, one per gene.

1. do({ … })

* Executes custom code for each gene subset:
* filter() selects the two groups to compare for each test.
* n\_distinct(...)=2 checks that both groups are present.
* t.test(Abundance ~ Group or Timepoint) performs the t-test.
* Results are packed into a one-row tibble of p-values.

1. p.adjust(..., method = "fdr")

Applies the Benjamini–Hochberg procedure to control the false discovery rate across all genes for each comparison.

left\_join(stat\_df, by = "Gene\_Symbol")

Combines the fold-change data (fc\_df) with the statistics (stat\_df) into gene\_stats.

*Key Packages*

dplyr: For data manipulation verbs (mutate(), group\_by(), filter(), ungroup(), left\_join()), piping, and slice\_max().

stats (base R): Provides t.test() for performing two-sample t-tests and p.adjust() for multiple testing correction.

tibble: Creates the one-row tibbles of p-values in the do() call.

stringr: (Indirectly) if any string operations are used elsewhere; not directly called in this section but part of previous cleaning steps.

## SECTION 8: Pathway Enrichment Analysis (Aging in Muscle)

*Purpose*

* Convert gene symbols to Entrez IDs and then conduct pathway enrichment analyses (KEGG, Reactome, and GO Biological Process) for the “Pre\_MA vs Young” comparison to identify biological pathways associated with aging in muscle.

# ---- 8. Pathway Enrichment Analysis (Aging in Muscle) ----  
  
# 1. Map gene symbols to Entrez IDs using the org.Hs.eg.db annotation database  
entrez\_map <- bitr(  
 gene\_stats$Gene\_Symbol, # Input vector of gene symbols  
 fromType = "SYMBOL", # Convert from SYMBOL  
 toType = "ENTREZID", # Convert to ENTREZID  
 OrgDb = org.Hs.eg.db # Human annotation package  
)

## 'select()' returned 1:many mapping between keys and columns

## Warning in bitr(gene\_stats$Gene\_Symbol, fromType = "SYMBOL", toType =  
## "ENTREZID", : 0.79% of input gene IDs are fail to map...

# 2. Prepare a named numeric vector of log2 fold-changes for enrichment  
gene\_list\_Pre\_vs\_Young <- gene\_stats %>%  
 inner\_join(entrez\_map, by = c("Gene\_Symbol" = "SYMBOL")) %>% # Keep only genes with Entrez IDs  
 arrange(desc(log2FC\_Pre\_vs\_Young)) %>% # Sort descending by effect size  
 { set\_names(.$log2FC\_Pre\_vs\_Young, .$ENTREZID) } # Create named vector: names = ENTREZID, values = log2FC  
  
# 3. Run KEGG pathway enrichment for aging (Pre\_MA vs Young)  
kegg\_Pre\_vs\_Young <- enrichKEGG(  
 gene = names(gene\_list\_Pre\_vs\_Young), # Vector of Entrez IDs  
 organism = "hsa", # Human KEGG code  
 pvalueCutoff = 0.10 # Include terms with p ≤ 0.10  
)

## Reading KEGG annotation online: "https://rest.kegg.jp/link/hsa/pathway"...

## Reading KEGG annotation online: "https://rest.kegg.jp/list/pathway/hsa"...

# 4. Run Reactome pathway enrichment  
reactome\_Pre\_vs\_Young <- enrichPathway(  
 gene = names(gene\_list\_Pre\_vs\_Young), # Same Entrez IDs  
 organism = "human", # Reactome expects "human"  
 pvalueCutoff = 0.10  
)  
  
# 5. Run Gene Ontology (Biological Process) enrichment  
go\_Pre\_vs\_Young <- enrichGO(  
 gene = names(gene\_list\_Pre\_vs\_Young), # Entrez IDs  
 OrgDb = org.Hs.eg.db, # Annotation database  
 ont = "BP", # Ontology: Biological Process  
 pvalueCutoff = 0.10  
)  
  
# 6. Collect all results into a single list for easy access  
enrichment\_results <- list(  
 KEGG\_Pre\_vs\_Young = kegg\_Pre\_vs\_Young,  
 Reactome\_Pre\_vs\_Young = reactome\_Pre\_vs\_Young,  
 GO\_Pre\_vs\_Young = go\_Pre\_vs\_Young  
)

*Step-by-Step Breakdown*

bitr(..., OrgDb = org.Hs.eg.db)

Uses clusterProfiler’s bitr() to map human gene symbols (SYMBOL) to Entrez IDs (ENTREZID) via the org.Hs.eg.db Bioconductor annotation package.

inner\_join(entrez\_map, by = c("Gene\_Symbol" = "SYMBOL"))

Joins gene\_stats with entrez\_map to retain only genes that successfully mapped to an Entrez ID.

arrange(desc(log2FC\_Pre\_vs\_Young))

Orders the joined data frame by descending log₂ fold-change, so the most upregulated aging-associated genes come first.

{ set\_names(.$log2FC\_Pre\_vs\_Young, .$ENTREZID) }

Constructs a named numeric vector where each element is the log₂ fold-change and its name is the corresponding Entrez ID, as required by enrichment functions like enrichKEGG().

enrichKEGG(gene = ..., organism = "hsa", pvalueCutoff = 0.10)

Runs KEGG pathway enrichment using clusterProfiler, specifying human (“hsa”) and including pathways with p-values ≤ 0.10.

enrichPathway(..., organism = "human")

Performs Reactome enrichment via ReactomePA’s enrichPathway(), again with a 0.10 p-value threshold.

enrichGO(..., ont = "BP")

Executes Gene Ontology Biological Process enrichment using clusterProfiler’s enrichGO(), referencing the org.Hs.eg.db database.

list(...)

Aggregates the three enrichment result objects into a named list (enrichment\_results) for downstream plotting or inspection.

*Key Packages*

clusterProfiler: Provides functions for enrichment analyses, including bitr(), enrichKEGG(), and enrichGO().

ReactomePA: Supplies enrichPathway() for Reactome database enrichment.

org.Hs.eg.db: Bioconductor annotation package mapping human gene symbols to Entrez IDs.

dplyr: Used for data manipulation (inner\_join(), arrange() and pipelining).

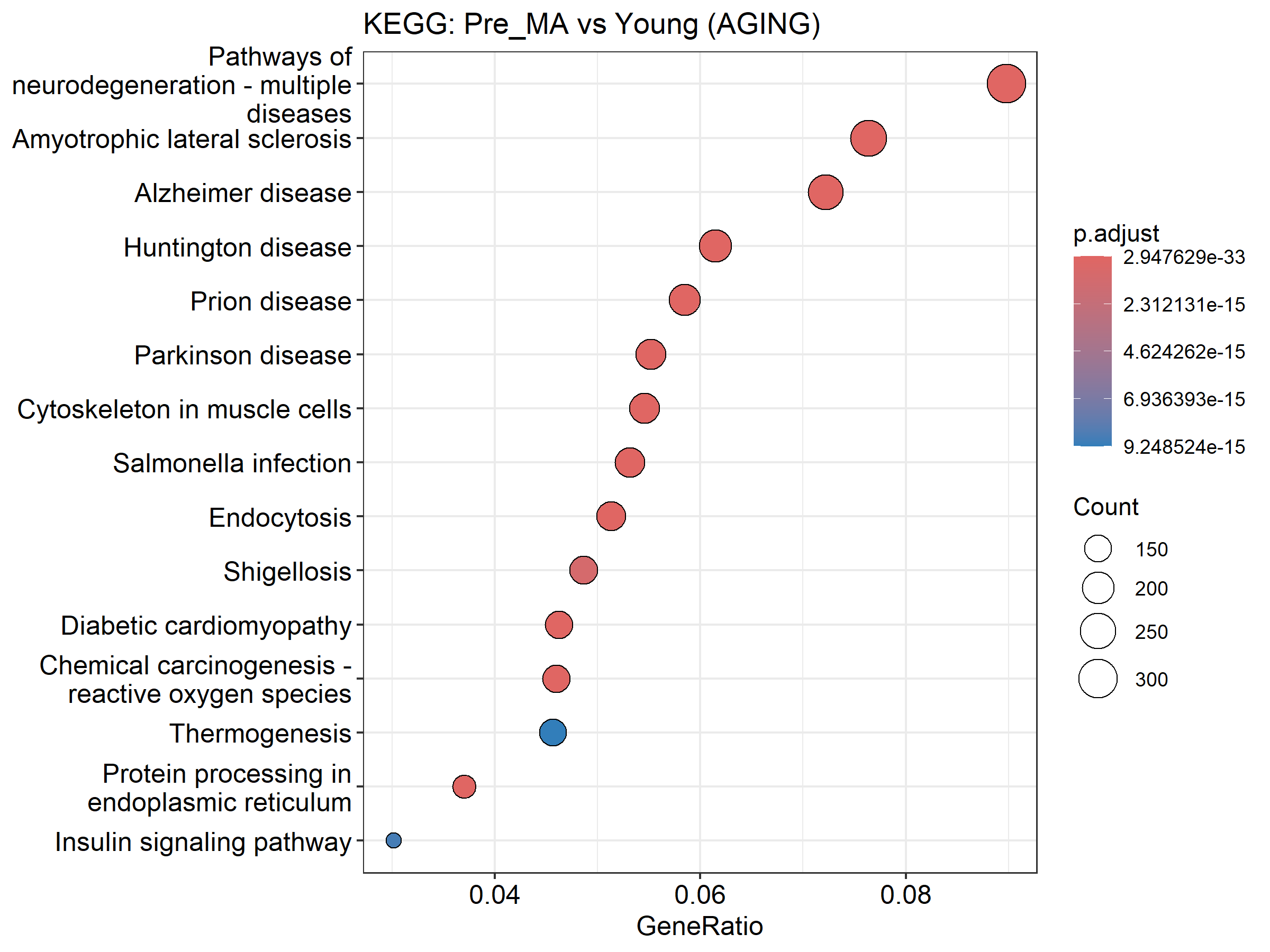
magrittr: For the pipe (%>%) and the compound expression ({}) to set names on the vector.

## SECTION 9: Visualize Enrichment Results (Aging in Muscle)

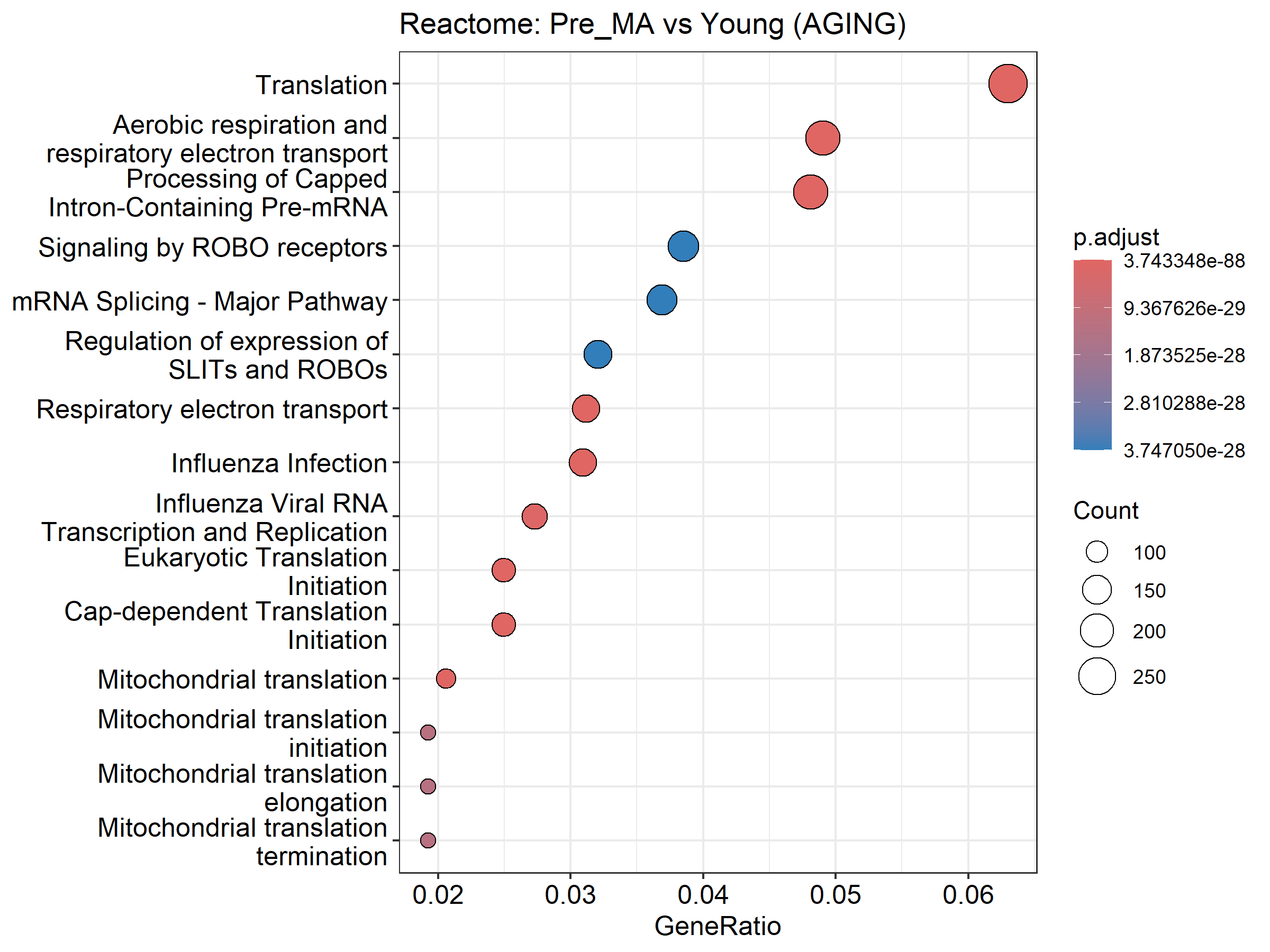
*Purpose*

* Generate and display dot plots of the top 15 enriched pathways from the KEGG, Reactome, and GO analyses for the Pre\_MA vs Young comparison, labeling each plot with a clear title.

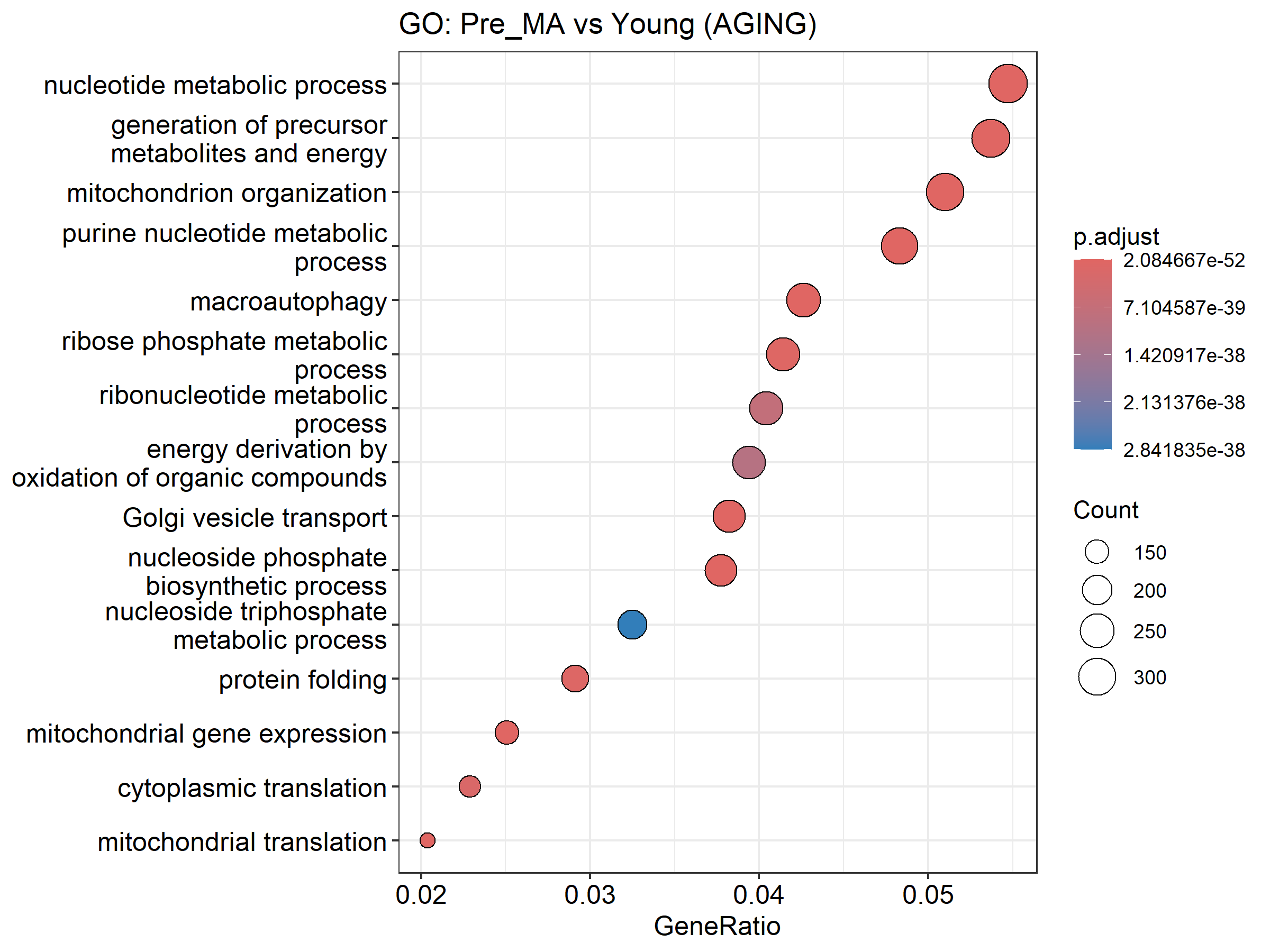
# ---- 9. Visualize Enrichment Results (Aging in Muscle) ----  
  
# Plot KEGG enrichment results, showing the top 15 pathways  
print(  
 dotplot(  
 kegg\_Pre\_vs\_Young,  
 showCategory = 15 # Number of categories (pathways) to display  
 ) +  
 ggtitle("KEGG: Pre\_MA vs Young (AGING)") # Add a descriptive title  
)



# Plot Reactome enrichment results  
print(  
 dotplot(  
 reactome\_Pre\_vs\_Young,  
 showCategory = 15  
 ) +  
 ggtitle("Reactome: Pre\_MA vs Young (AGING)")  
)



# Plot GO Biological Process enrichment results  
print(  
 dotplot(  
 go\_Pre\_vs\_Young,  
 showCategory = 15  
 ) +  
 ggtitle("GO: Pre\_MA vs Young (AGING)")  
)



*Step-by-Step Breakdown*

dotplot(kegg\_Pre\_vs\_Young, showCategory = 15)

Uses enrichplot’s dotplot() to create a bubble/dot plot for the KEGG enrichment object, displaying the top 15 enriched pathways by default sorted by gene ratio or p-value.

+ ggtitle("KEGG: Pre\_MA vs Young (AGING)")

Adds a title to the ggplot object via ggplot2’s ggtitle(), clarifying the comparison and context.

1. print(...)

Ensures the plot is rendered in non-interactive R sessions or scripts; explicitly prints the combined plot object to the current graphics device.

1. Repeat for reactome\_Pre\_vs\_Young and go\_Pre\_vs\_Young

Generates analogous plots for Reactome and GO Biological Process results, each showing the top 15 terms and bearing its own title.

*Key Packages*

enrichplot: Provides the dotplot() function tailored for enrichment result objects from clusterProfiler and ReactomePA.

ggplot2: Underpins the plotting system, used here for adding titles (ggtitle()) and handling plot objects.

grid (implicitly via base R): Manages the display of multiple plots when using print().

## SECTION 10: Facetted “Zoom-In” Pathway Plots (p ≤ 0.10 only)

*Purpose*

* Define concise sets of biologically relevant pathways for KEGG, Reactome, and GO so that subsequent plots can focus (“zoom in”) on these specific targets, filtered by p ≤ 0.10.

# ---- 10. Facetted “Zoom-In” Pathway Plots (p ≤ 0.10 only) ----  
  
# Define pathway targets for aging and training  
kegg\_targets <- c(  
 Glutathione = "Glutathione metabolism",  
 Proteasome = "Proteasome",  
 ER\_Processing = "Protein processing in endoplasmic reticulum",  
 Ubiquitin = "Ubiquitin mediated proteolysis"  
)  
  
reactome\_targets <- c(  
 DetoxROS = "Detoxification of Reactive Oxygen Species",  
 ERAD = "ER-Associated Degradation (ERAD)",  
 Folding = "Protein folding",  
 Macroautophagy = "Macroautophagy"  
)  
  
go\_targets <- c(  
 OxidativeStress = "response to oxidative stress",  
 ProteinFolding = "protein folding",  
 Proteolysis = "proteasome-mediated ubiquitin-dependent protein catabolic process",  
 UPR = "endoplasmic reticulum unfolded protein response"  
)

*Step-by-Step Breakdown*

kegg\_targets <- c(...)

Constructs a named character vector where each name (e.g., “Glutathione”) is a short label used for faceting or annotation, and each value is the exact KEGG pathway string to match in your enrichment results.

reactome\_targets <- c(...)

Similarly creates a named vector of key Reactome pathways of interest, mapping concise labels (“DetoxROS”, “ERAD”, etc.) to full pathway names for subsetting.

go\_targets <- c(...)

Defines a named vector of GO Biological Process terms, pairing simple labels with their corresponding GO descriptions for focused plotting.

*Key Packages*

Vectors will be used downstream with packages like dplyr and ggplot2 to filter enrichment result objects and generate facetted visualizations.)

## SECTION 11: Function to Create Individual Zoom-In Plots with Expanded Thresholds

*Purpose*

* Define function to extract genes from a specific enriched pathway, filter them by an adjusted-p threshold, assign significance stars, and generate a horizontal bar plot of their log2 fold-changes with clear annotations.

# ---- 11. Function to create individual zoom-in plots with expanded thresholds ----  
create\_zoom\_in\_plot <- function(enrichment, pathway\_name, target\_description, contrast, db\_name, max\_genes = 10) {  
 # 1. Locate the exact pathway description; fall back to grep if needed  
 idx <- which(enrichment$Description == target\_description)  
 if (length(idx) == 0) idx <- grep(target\_description, enrichment$Description, ignore.case = TRUE)  
 if (length(idx) == 0) return(NULL) # No matching pathway  
   
 # 2. Split the geneID field (Entrez IDs separated by "/") into a vector  
 genes <- str\_split(enrichment$geneID[idx[1]], "/")[[1]]  
 # 3. Map Entrez IDs back to gene symbols via the entrez\_map lookup  
 symbols <- entrez\_map$SYMBOL[match(genes, entrez\_map$ENTREZID)]  
   
 # 4. Build a data frame of gene stats, filtering by adjusted p ≤ 0.20  
 plot\_data <- gene\_stats %>%  
 filter(  
 Gene\_Symbol %in% symbols,  
 .data[[paste0("adj\_", contrast)]] <= 0.20  
 ) %>%  
 transmute(  
 Gene\_Symbol, # Keep the gene symbol  
 value = .data[[paste0("log2FC\_", contrast)]], # Corresponding log2FC  
 star = case\_when( # Significance stars  
 .data[[paste0("adj\_", contrast)]] < 0.001 ~ "\*\*\*",  
 .data[[paste0("adj\_", contrast)]] < 0.01 ~ "\*\*",  
 .data[[paste0("adj\_", contrast)]] < 0.05 ~ "\*",  
 .data[[paste0("adj\_", contrast)]] < 0.10 ~ ".",  
 TRUE ~ ""  
 ),  
 significance = case\_when( # Categorical significance labels  
 .data[[paste0("adj\_", contrast)]] < 0.05 ~ "Significant",  
 .data[[paste0("adj\_", contrast)]] < 0.10 ~ "Near Significant",  
 .data[[paste0("adj\_", contrast)]] < 0.20 ~ "Marginally Significant",  
 TRUE ~ "Not Significant"  
 )  
 ) %>%  
 arrange(desc(value)) %>% # Sort by descending log2FC  
 head(max\_genes) # Keep only top N genes  
   
 if (nrow(plot\_data) == 0) return(NULL) # Nothing to plot  
   
 # 5. Generate and print the ggplot  
 print(  
 ggplot(plot\_data, aes(x = reorder(Gene\_Symbol, value), y = value, fill = significance)) +  
 geom\_col(show.legend = TRUE) + # Bar plot of log2FC  
 geom\_text(  
 aes(label = star),  
 hjust = -0.3, size = 5, fontface = "bold"  
 ) + # Add stars just outside the bars  
 coord\_flip() + # Horizontal bars  
 scale\_fill\_manual(values = c( # Custom fill colors  
 "Significant" = "#D55E00",  
 "Near Significant" = "#F0E442",  
 "Marginally Significant" = "#009E73",  
 "Not Significant" = "#C0C0C0"  
 )) +  
 labs(  
 title = paste(db\_name, "Pathway:", pathway\_name, "(", contrast, ")"),  
 subtitle = paste(  
 "Pathway:", target\_description,  
 "\nSignificance: \*\*\* p<0.001, \*\* p<0.01, \* p<0.05, . p<0.10, marginal p<0.20"  
 ),  
 x = "Gene Symbol",  
 y = paste0("log₂ Fold Change (", contrast, ")"),  
 caption = "Bars = log2FC; color = adjusted-p significance."  
 ) +  
 theme\_minimal(base\_size = 14) + # Clean theme  
 theme(  
 plot.title = element\_text(face = "bold", size = 16),  
 plot.subtitle = element\_text(size = 14, margin = margin(b = 10)),  
 axis.text.y = element\_text(size = 12),  
 axis.text.x = element\_text(size = 12),  
 axis.title = element\_text(size = 14, face = "bold"),  
 plot.caption = element\_text(size = 10, face = "italic"),  
 legend.title = element\_text(size = 12),  
 legend.text = element\_text(size = 12)  
 )  
 )  
}

*Step-by-Step Breakdown*

1. Locate pathway index

* Uses which() for exact match on Description; if none found, falls back to grep() ignoring case; returns `NULL`` if still not found.

1. Extract gene list

* Splits the /-separated geneID string into individual Entrez IDs using stringr’s str\_split(), then maps them back to Gene\_Symbol via the entrez\_map lookup table.

1. Filter and annotate

* Filters gene\_stats to genes in the pathway whose adjusted p-value for the given contrast is ≤ 0.20.
* Uses transmute() to:
* Pull out Gene\_Symbol and the appropriate log2FC\_\* value.
* Create star, a string of significance asterisks based on adjusted-p thresholds.
* Create significance, a categorical label (“Significant,” “Near Significant,” etc.).

1. Sort and limit

Orders by descending value (log2FC) and retains only the top max\_genes.

1. Plotting

* Constructs a horizontal bar plot (geom\_col() + coord\_flip()).
* Overlays significance stars with geom\_text(), just outside each bar.
* Applies scale\_fill\_manual() to assign distinct colors to each significance category.
* Adds descriptive titles, subtitles, axis labels, and a caption via labs().
* Applies a minimalist theme and customizes text sizes and styles with theme().

*Key Packages*

dplyr: Data manipulation (filter(), transmute(), arrange(), head()), piping, and use of the .data pronoun.

stringr: String splitting (str\_split()) for processing the geneID field.

ggplot2: Plot construction and customization (ggplot(), geom\_col(), geom\_text(), coord\_flip(), scale\_fill\_manual(), labs(), theme\_minimal(), theme()).

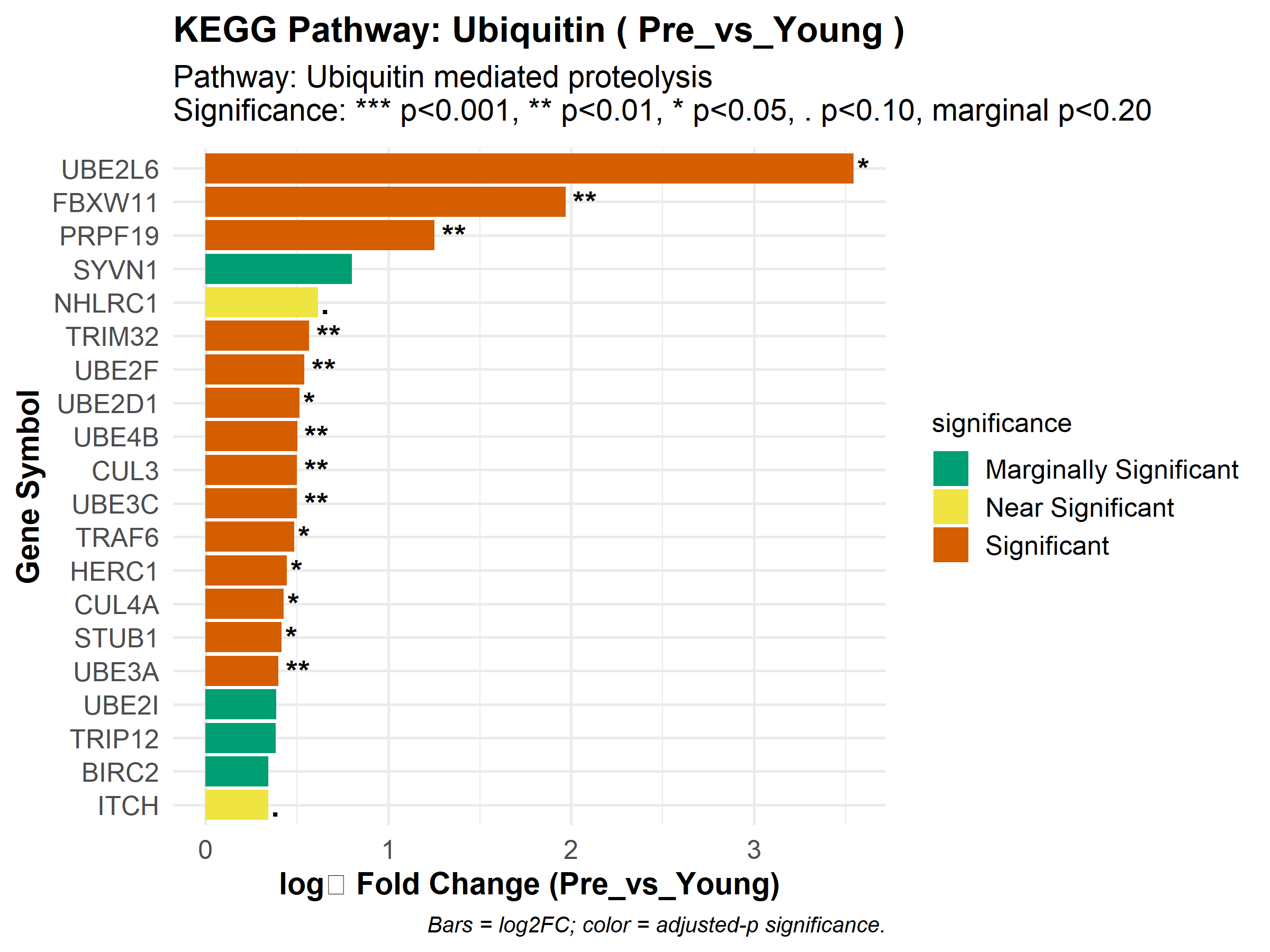
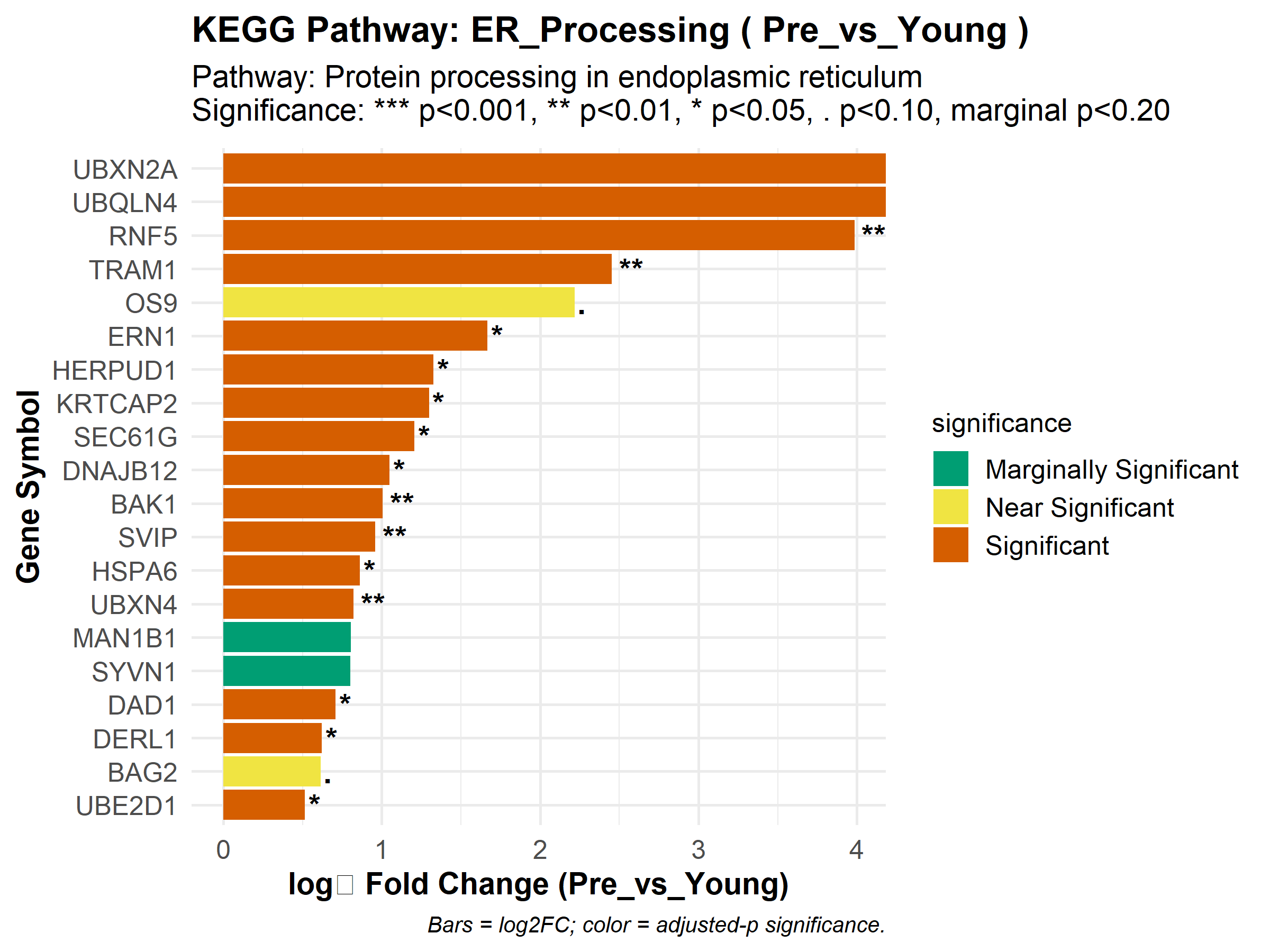
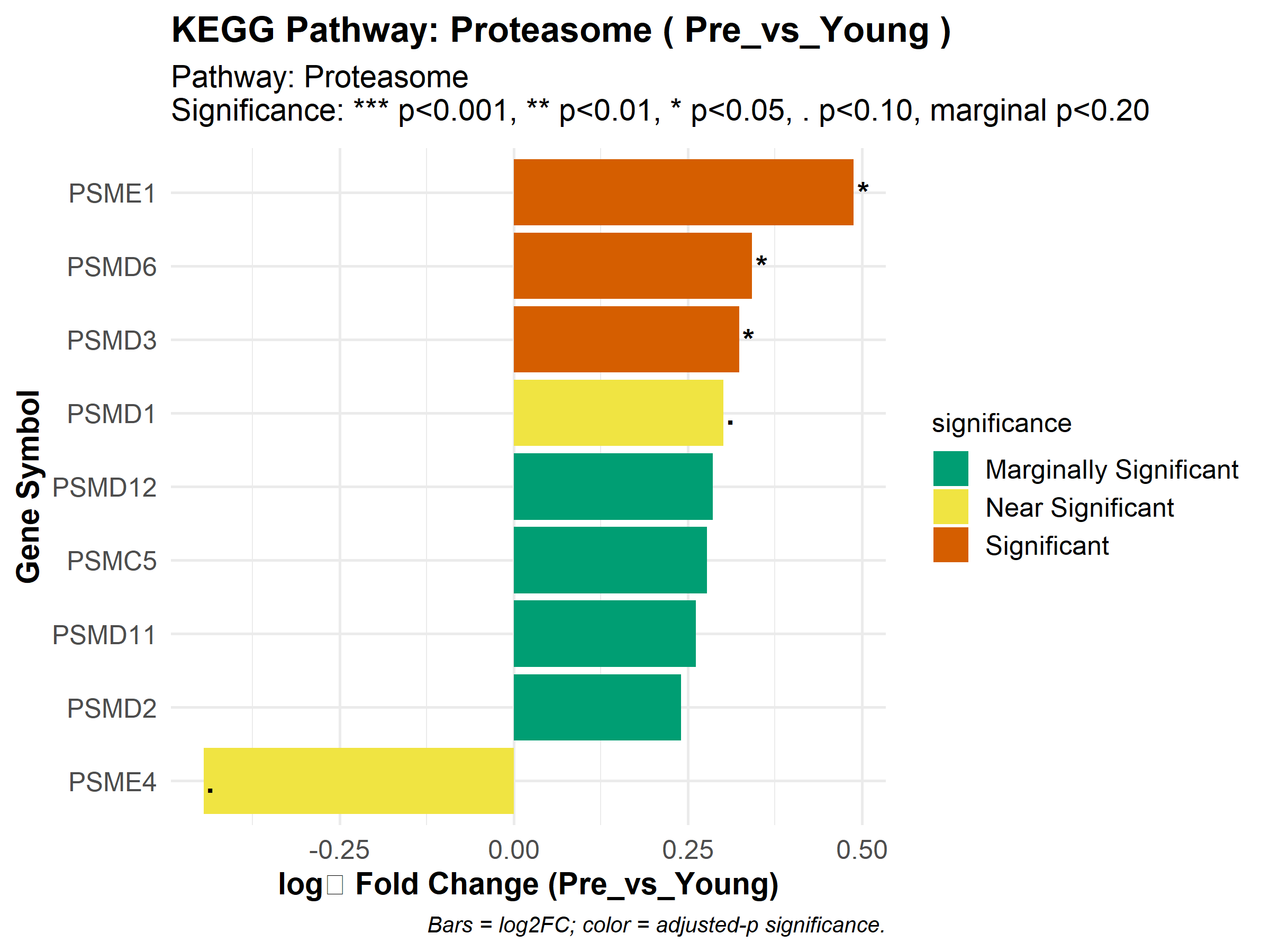
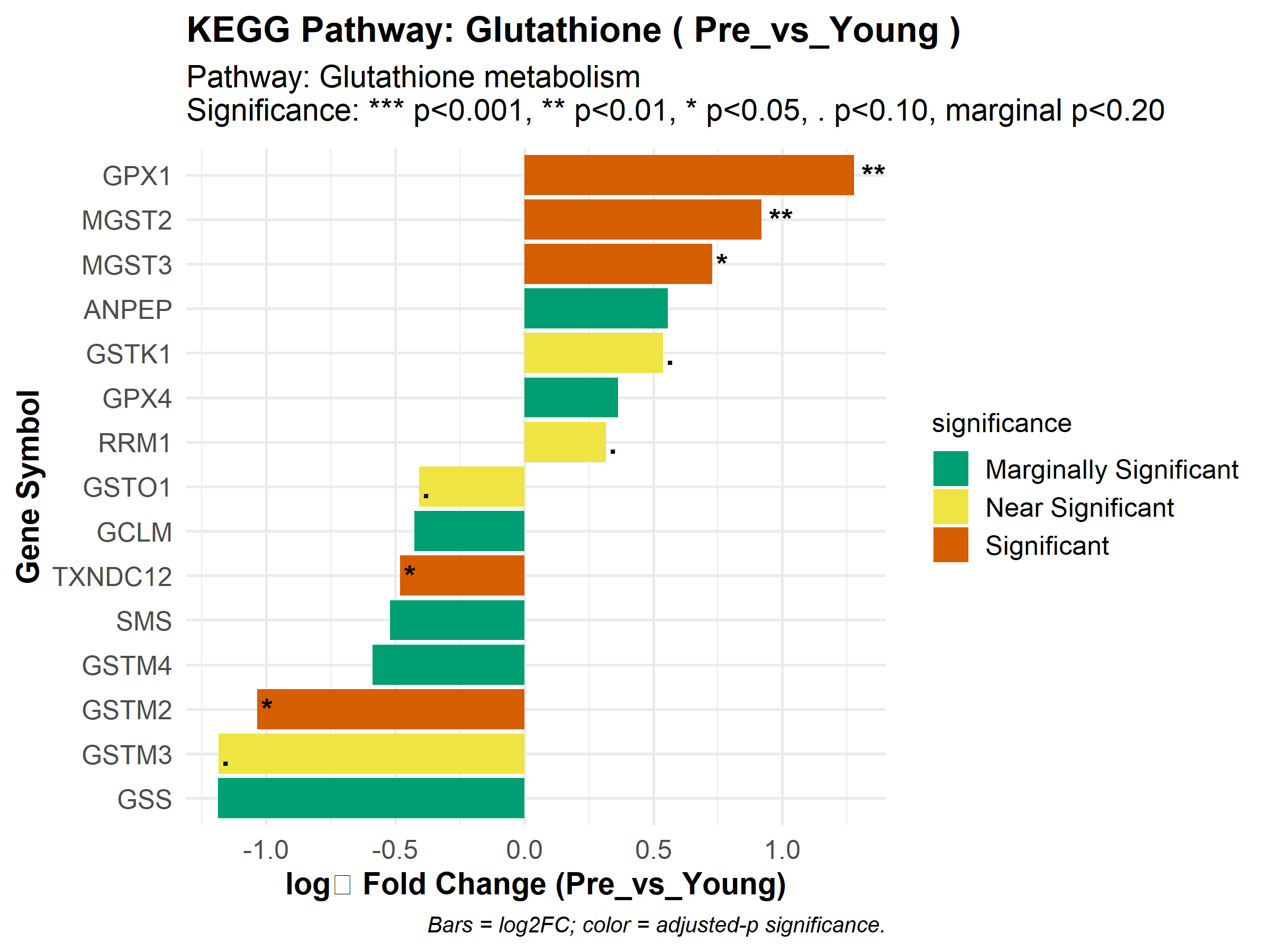
rlang: Enables the use of .data[[…]] for programmatic column access within dplyr pipelines.

## SECTION 12:

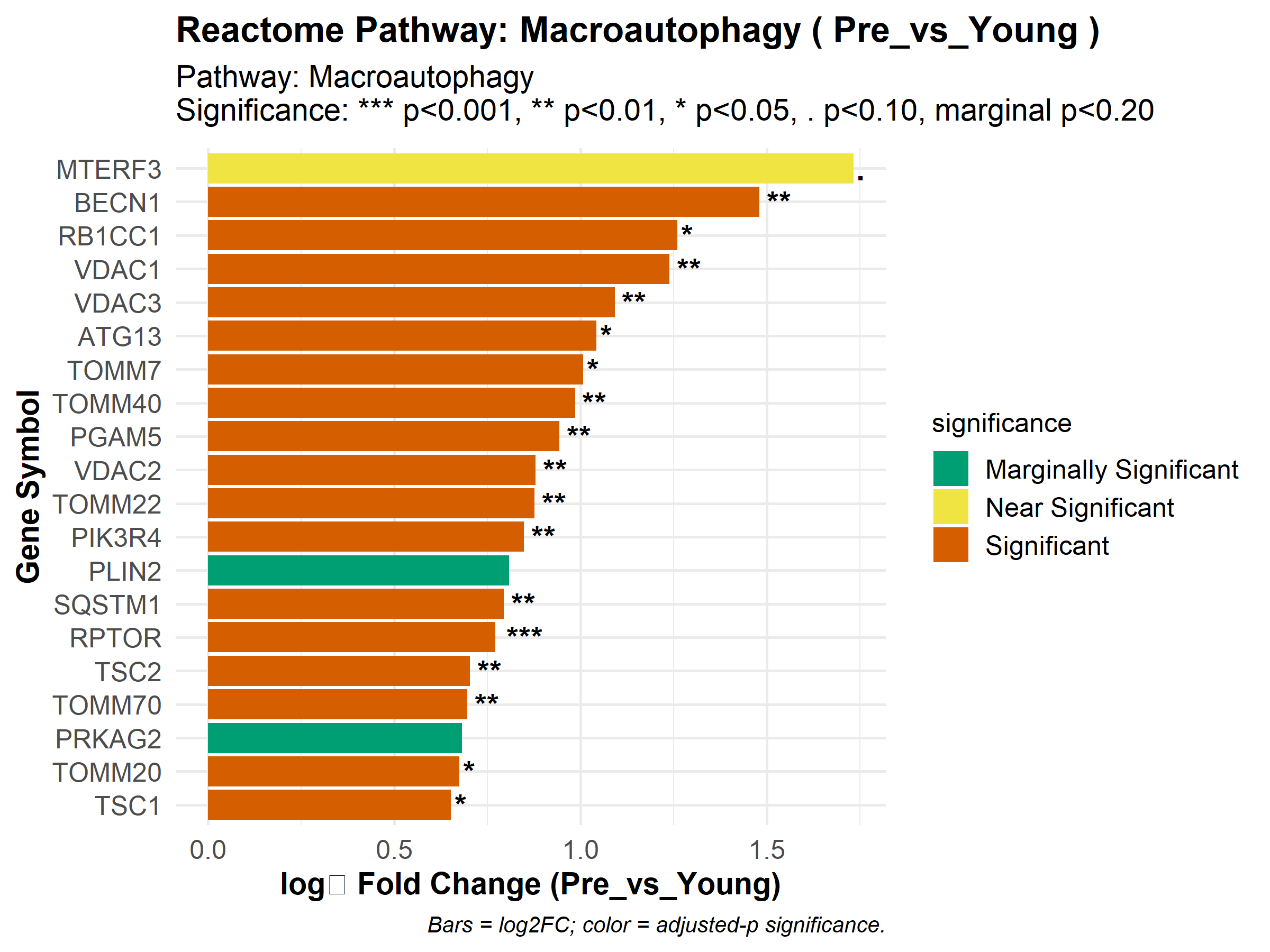
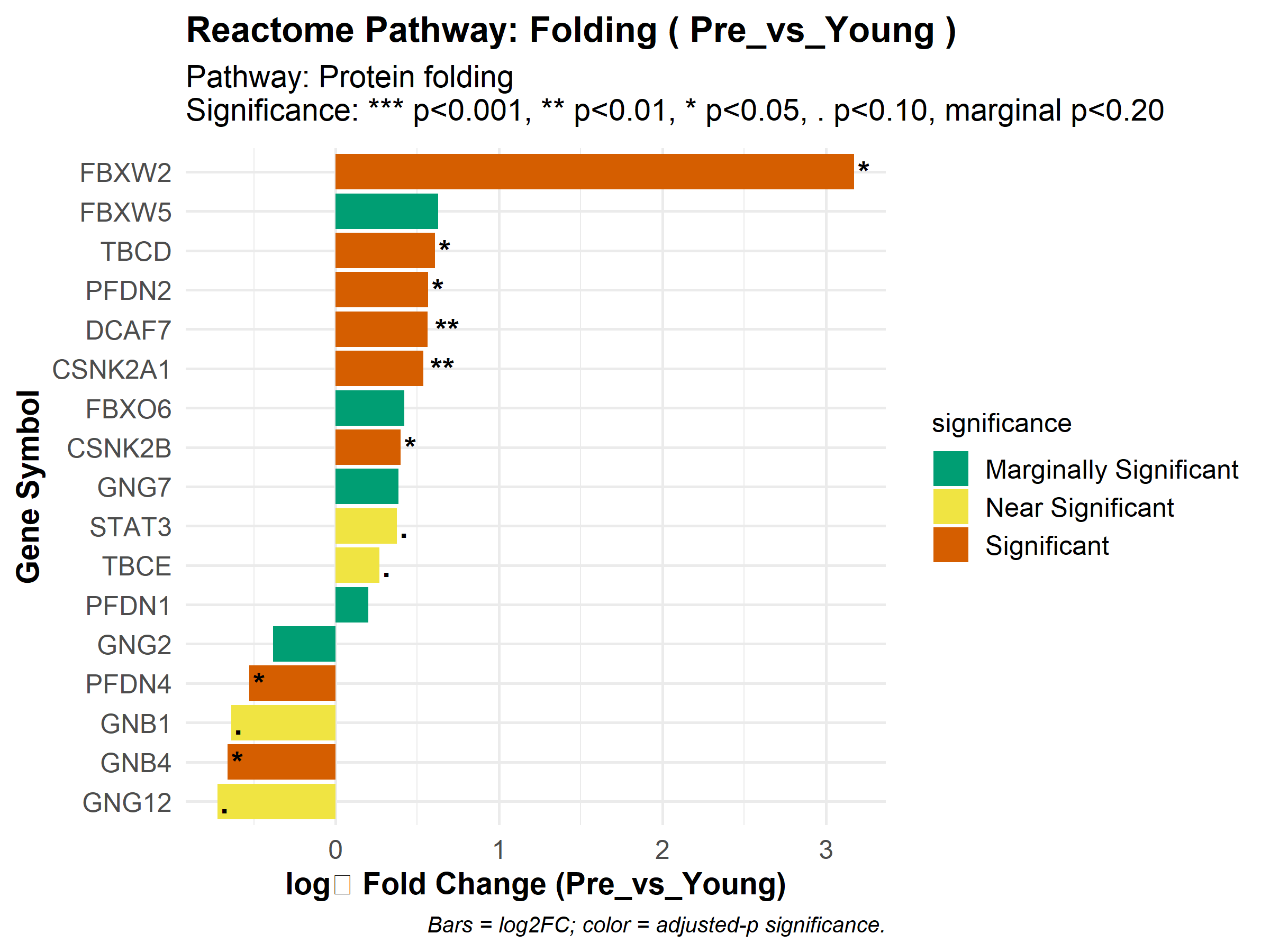
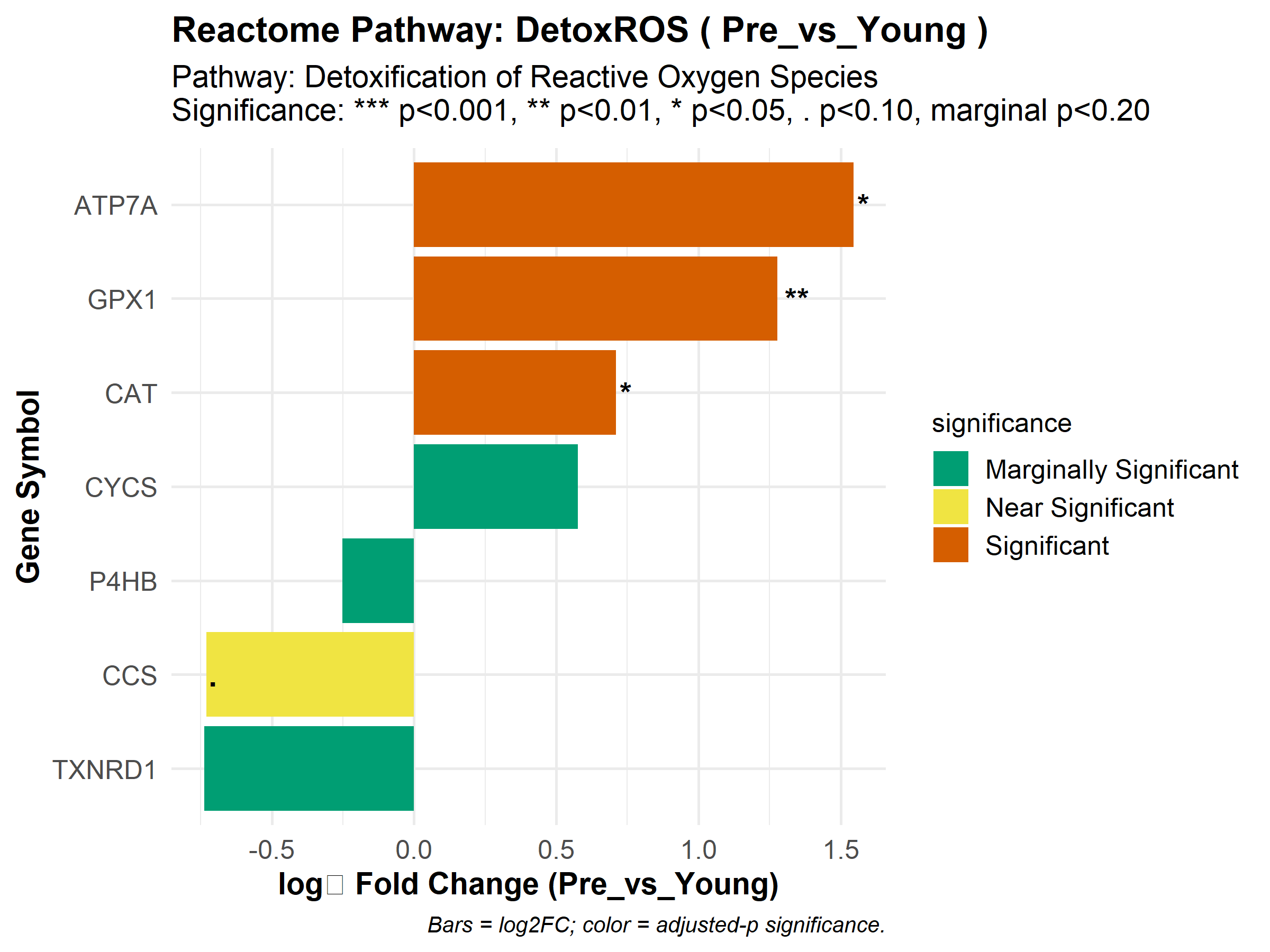
*Purpose*

* Loop over defined contrasts (aging) and key pathway sets (KEGG, Reactome, GO), retrieve the corresponding enrichment results, and invokes the create\_zoom\_in\_plot() function to produce focused bar plots for each pathway.

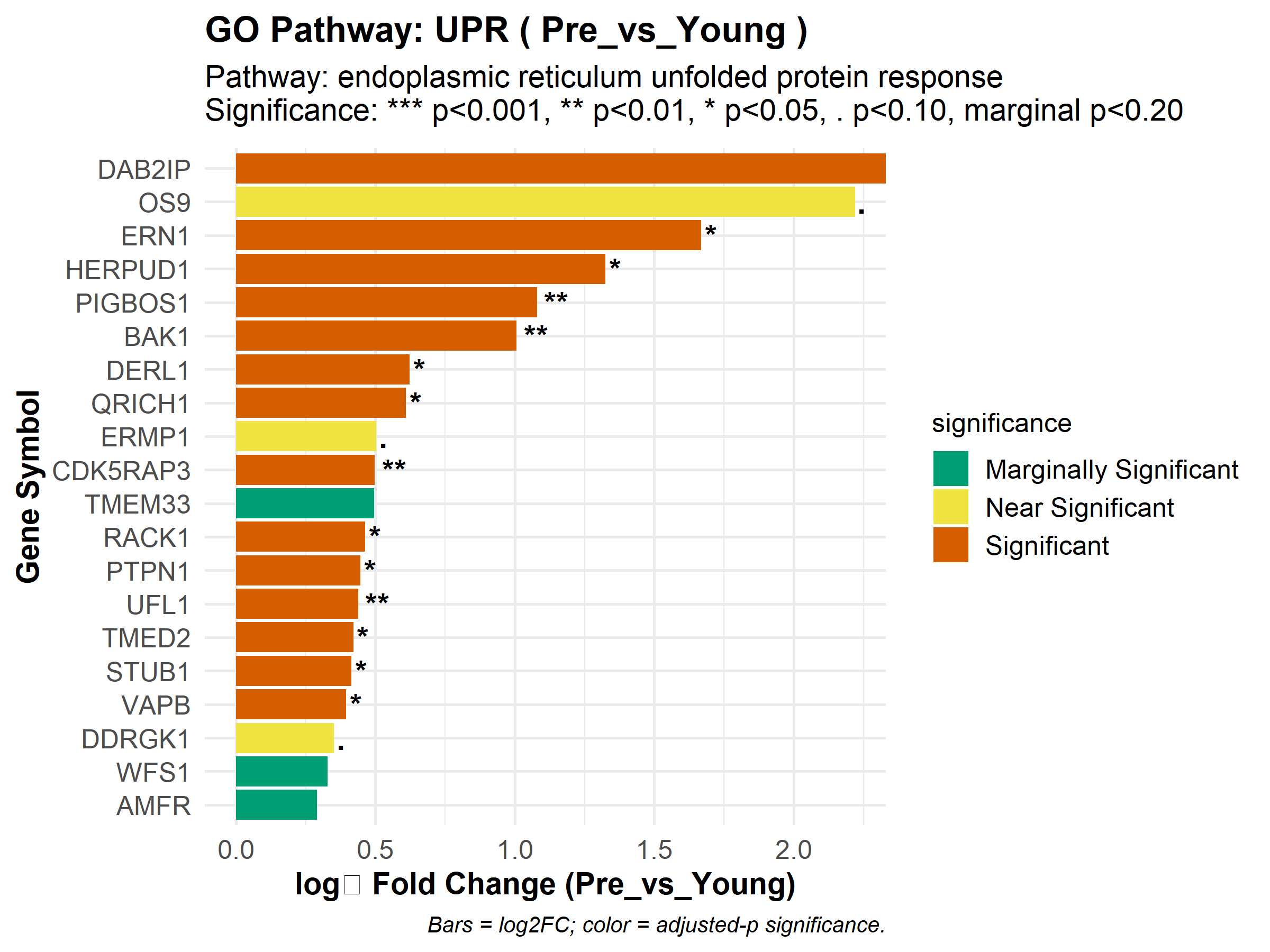
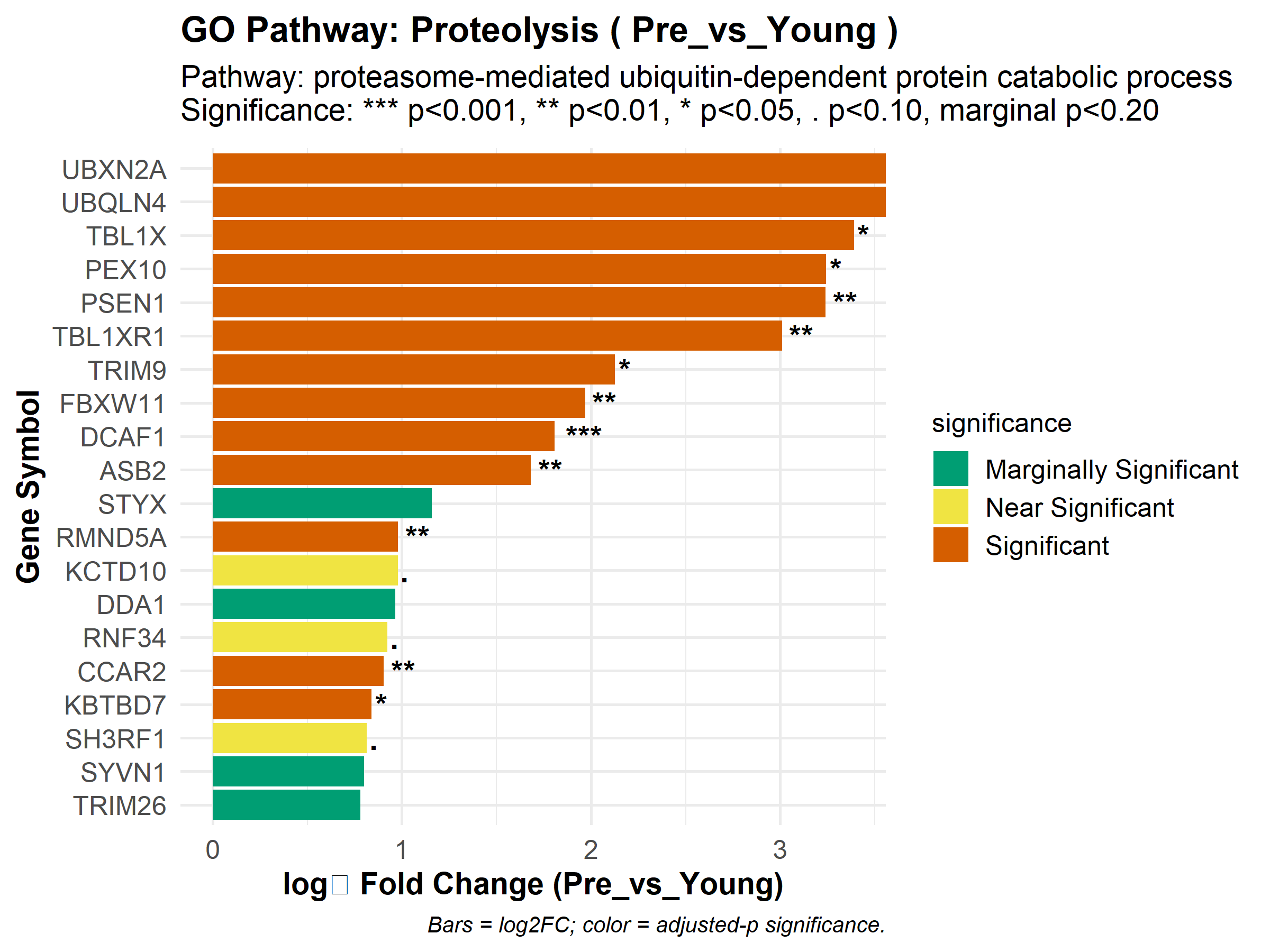
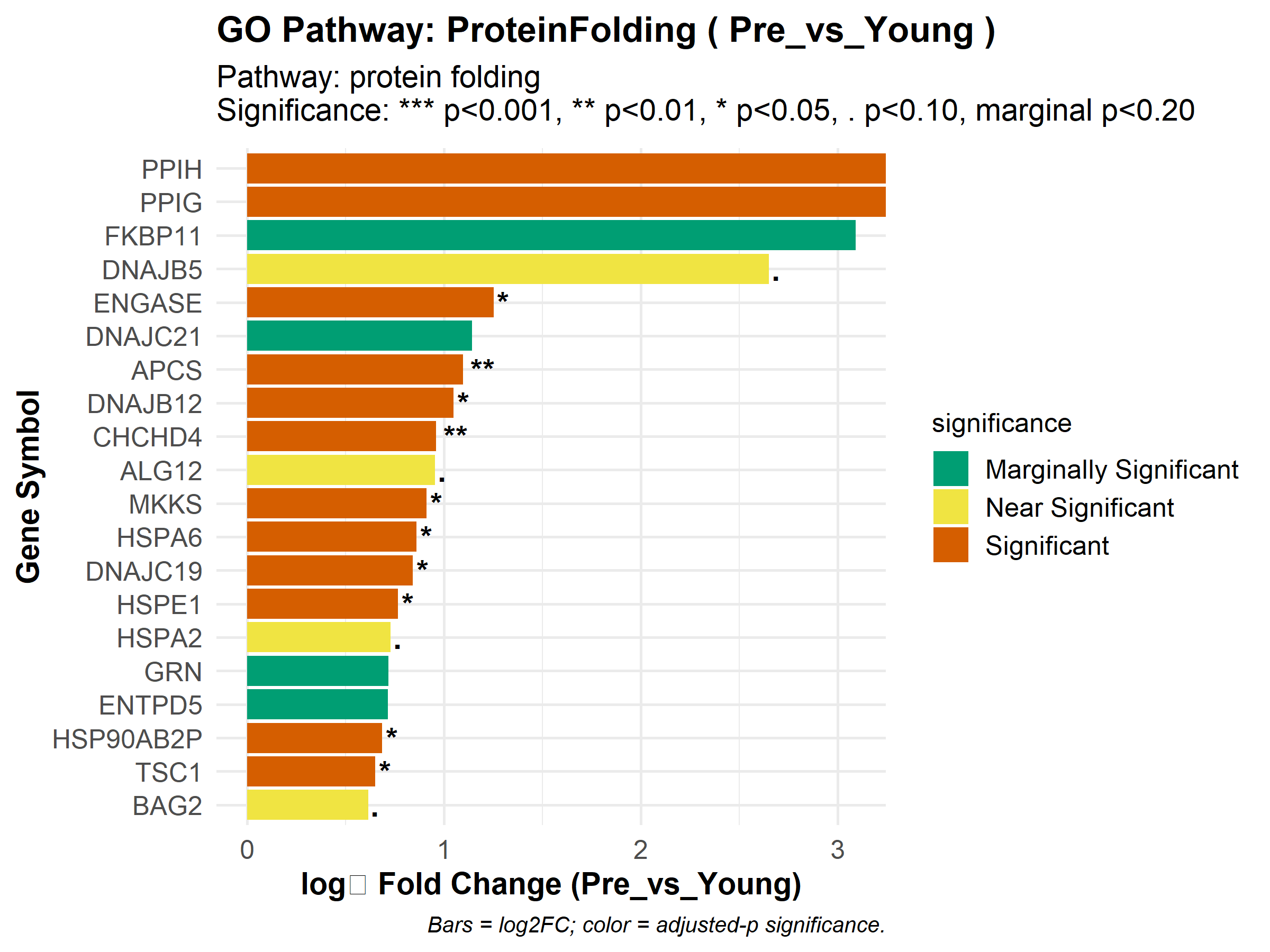
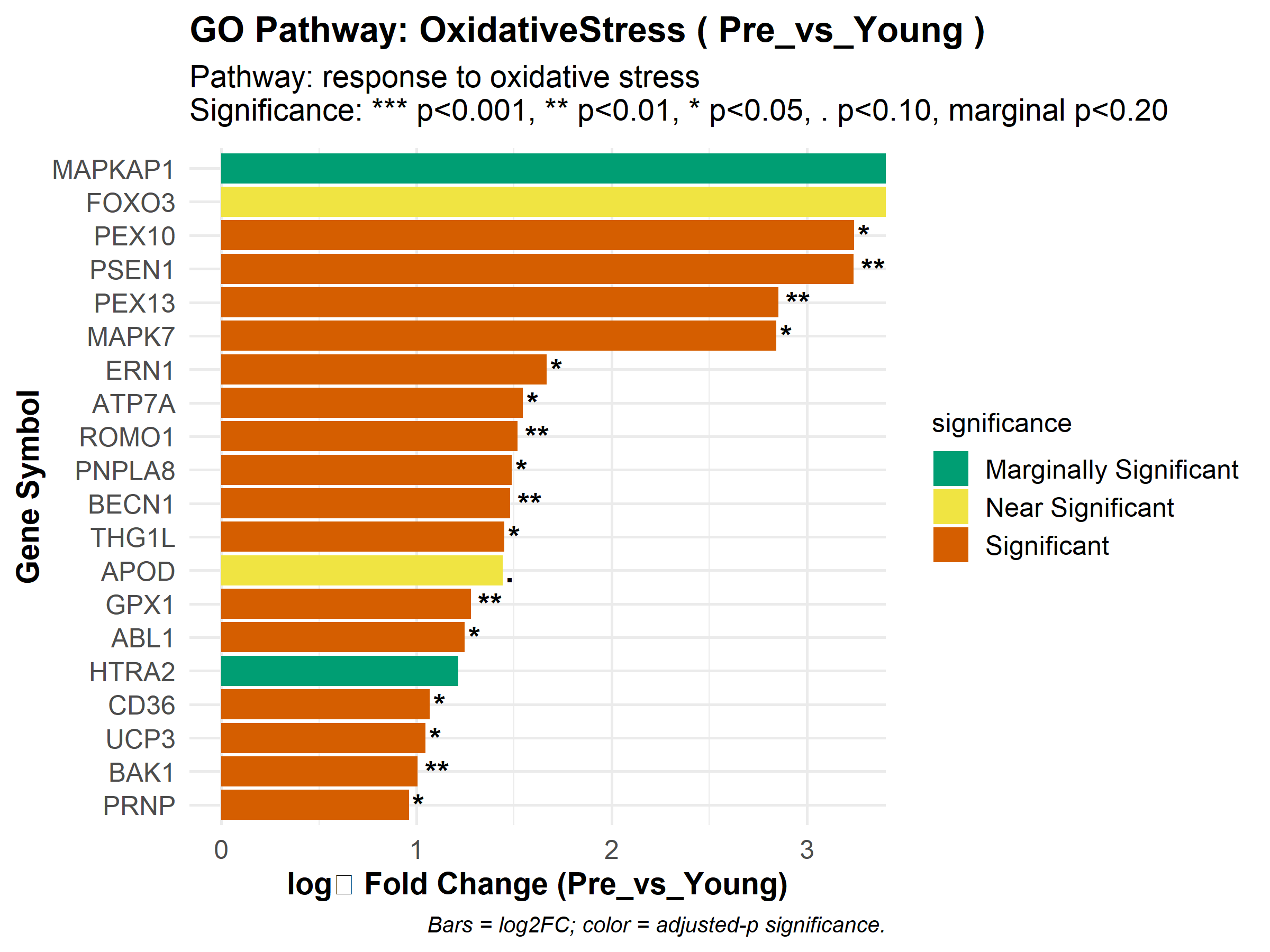
# ---- 12. Generate plots for each database and contrast ----  
  
# Define the contrasts to iterate over: aging (Pre\_vs\_Young), \*could use and alter for other comparisons, but keeping it simple for now\*  
contrasts <- c("Pre\_vs\_Young")  
  
# KEGG pathways: for each contrast and each target pathway, generate a zoom-in plot  
for (contrast in contrasts) {  
 for (pathway\_name in names(kegg\_targets)) {  
 # Retrieve the KEGG enrichment result object for this contrast  
 enrichment <- enrichment\_results[[paste0("KEGG\_", contrast)]]  
 # Produce the bar plot for this pathway  
 create\_zoom\_in\_plot(  
 enrichment,  
 pathway\_name,  
 kegg\_targets[[pathway\_name]],  
 contrast,  
 db\_name = "KEGG",  
 max\_genes = 20  
 )  
 }  
}



# Reactome pathways: same logic as KEGG but using Reactome target list  
for (contrast in contrasts) {  
 for (pathway\_name in names(reactome\_targets)) {  
 enrichment <- enrichment\_results[[paste0("Reactome\_", contrast)]]  
 create\_zoom\_in\_plot(  
 enrichment,  
 pathway\_name,  
 reactome\_targets[[pathway\_name]],  
 contrast,  
 db\_name = "Reactome",  
 max\_genes = 20  
 )  
 }  
}



# GO Biological Process pathways: loop over GO targets  
for (contrast in contrasts) {  
 for (pathway\_name in names(go\_targets)) {  
 enrichment <- enrichment\_results[[paste0("GO\_", contrast)]]  
 create\_zoom\_in\_plot(  
 enrichment,  
 pathway\_name,  
 go\_targets[[pathway\_name]],  
 contrast,  
 db\_name = "GO",  
 max\_genes = 20  
 )  
 }  
}



*Step-by-Step Breakdown*

contrasts <- c("Pre\_vs\_Young", ...)

Defines a character vector of the comparisons to analyze: aging (“Pre\_vs\_Young”) or possibly training response (“Post\_vs\_Pre\_MA”) in the future.

for (contrast in contrasts) { … }

Outer loop: iterates over each contrast string.

for (pathway\_name in names(kegg\_targets)) { … }

Inner loop: iterates over the named elements of the kegg\_targets vector, so you handle each pathway label (“Glutathione”, “Proteasome”, etc.).

enrichment <- enrichment\_results[[paste0("KEGG\_", contrast)]]

Dynamically constructs the list name (e.g., “KEGG\_Pre\_vs\_Young”) and extracts that enrichment result object from enrichment\_results.

1. create\_zoom\_in\_plot(enrichment, pathway\_name, kegg\_targets[[pathway\_name]], contrast, "KEGG", max\_genes = 20)

* Calls the previously defined plotting function, passing:
  + enrichment: the enrichment object,
  + pathway\_name: the short label,
  + kegg\_targets[[pathway\_name]]: the full pathway description,
  + contrast: which log₂FC column and p-values to use,
  + "KEGG": the database name for titles,
  + max\_genes = 20: cap at 20 genes per plot.

1. Repeat for Reactome and GO

Uses the same looping structure and function call, simply swapping in reactome\_targets/GO\_targets and the appropriate prefix in enrichment\_results (e.g., “Reactome\_Post\_vs\_Pre\_MA”).

*Key Packages*

base R: Provides loop constructs (for), vector creation (c()), string concatenation (paste0()), list indexing ([[ ]]), and name retrieval (names()).

ggplot2, dplyr, stringr, rlang: Employed indirectly by create\_zoom\_in\_plot() for data manipulation, filtering, and plotting, enabling the generation of the annotated bar charts.