ML Predictor of IBD Status from Microbiome Profiles

Paul B. Popa

(mailto:#)ppopa93@gmail.com (mailto:ppopa93@gmail.com)

2025-07-13

Packages

```
library(caret)
library(curatedMetagenomicData)
library(dplyr)
library(DT)
library(furrr)
library(ggplot2)
library(mia)
library(microbiome)
library(phyloseq)
library(plotly)
library(pROC)
library(PRROC)
library(purrr)
library(randomForest)
library(SummarizedExperiment)
library(table1)
library(tidymodels)
library(tidyverse)
library(TreeSummarizedExperiment)

# setup
options(DT.fillContainer = FALSE,
        DT.options = list(scrollY = "500px", scrollX = TRUE, scrollCollapse = FALSE, pagi
)
```

Retrieval of datasets from IBD-associated studies

```
data("sampleMetadata")

# what diseases do we have available?
availablediseases <- pull(sampleMetadata, study_condition) %>%
  table() %>%
  sort(decreasing = TRUE)
availablediseases
#> .
#>                   control                       IBD                       T2D
#>                     15121                      2088                       882
#>                       IGT             premature_born                  melanoma
#>                       563                       448                       315
#>                      ACVD                   adenoma                       FMT
#>                       214                       209                       149
#>                       STH                    otitis             schizophrenia
#>                       108                       107                       106
#>                        AS                        HF                       T1D
#>                        97                        95                        89
#>          pre-hypertension                       CDI                    ME/CFS
#>                        56                        53                        50
#>                      STEC               fatty_liver                psoriasis carcinoma
#>                        42                        41                        41
#>                        AD             cephalosporins                        PD
#>                        38                        36                        31
#>             periodontitis                       SRP          peri-implantitis
#>                        24                        24                        23
#>                 mucositis                bronchitis   TKI_dependent_diarrhoea
#>                        20                        18                        16
#>        metabolic_syndrome                     donor            pyelonephritis infectiou
#>                        10                         9                         6
#>                     MDRB                     fever                 pneumonia
#>                         5                         3                         3
#>                    cough              pyelonefritis                   skininf
#>                         2                         2                         2
#>                 cystitis              salmonellosis                    sepsis
#>                         1                         1                         1

# get list of IBD studies
list_ibd_studies <- filter(sampleMetadata, study_condition %in% "IBD") %>% pull(study_name
list_ibd_studies
#> [1] "HallAB_2017"     "HMP_2019_ibdmdb" "IaniroG_2022"    "IjazUZ_2017"     "LiJ_2014"
#> [7] "VilaAV_2018"

# create TSE object -- note that 'counts' is FALSE by default
se <- suppressMessages(
  sampleMetadata %>%
    filter(study_name %in% list_ibd_studies & study_condition %in% c("IBD", "control")) %>
    returnSamples(dataType = "relative_abundance", counts = TRUE)
)
```

Exploration

Cohort characteristics

```
table1::table1( ~ disease + disease_subtype + age + gender + study_name,
                data = colData(se))
```

	Overall (N=2887)
disease	
healthy	799 (27.7%)
IBD	2051 (71.0%)
IBD;perianal_fistula	37 (1.3%)
disease_subtype	
CD	1205 (41.7%)
UC	860 (29.8%)
undetermined_colitis	20 (0.7%)
Missing	802 (27.8%)
age	
Mean (SD)	31.5 (18.6)
Median [Min, Max]	29.0 [6.00, 76.0]
Missing	657 (22.8%)
gender	
female	1362 (47.2%)
male	1169 (40.5%)
Missing	356 (12.3%)
study_name	
HallAB_2017	259 (9.0%)
HMP_2019_ibdmdb	1627 (56.4%)
IaniroG_2022	6 (0.2%)
IjazUZ_2017	94 (3.3%)
LiJ_2014	150 (5.2%)
NielsenHB_2014	396 (13.7%)
VilaAV_2018	355 (12.3%)

Convert TSE to phyloseq object

```
# still a bug when working with counts...change name to 'counts' from 'relative_abundance'
assayNames(se) <- "counts"
```

```
phy <- mia::convertToPhyloseq(se, assay.type = "counts")
#> Warning: Tips of rowTree are renamed to match rownames.
phy
#> phyloseq-class experiment-level object
#> otu_table()   OTU Table:         [ 940 taxa and 2887 samples ]
#> sample_data() Sample Data:       [ 2887 samples by 140 sample variables ]
#> tax_table()   Taxonomy Table:    [ 940 taxa by 7 taxonomic ranks ]
#> phy_tree()    Phylogenetic Tree: [ 940 tips and 939 internal nodes ]
```

Note that we have multiple samples per subject ID, i.e., our there is not a 1:1 mapping between sample and subject. This violates the assumption that each observation is independent. Additionally, subjects with more time points contribute disproportionately to alpha- and beta-diversity calculations, skewing measures of community richness and dissimilarity.

Investigate pre-filtered data

```
n_species <- tax_table(phy) %>%
  as("matrix") %>%                                # force a plain character matrix
  as_tibble() %>%                                 # now a tibble
  filter(!is.na(species), species != "") %>%      # drop NA or blank
  summarise(n = n()) %>%                          # count species
  pull(n)

n_species
#> [1] 935
```

So we have 940 TaxonID-level features, but 935 species-level features. Do we drop them?

```r
# turn tax_table from s4 obj to tibble
tax_df <- tax_table(phy) %>%
  as("matrix") %>%
  as_tibble(rownames = "TaxonID")

# turn sample_data from s4 obj to tibble
samp_df <- sample_data(phy) %>%
  as("matrix") %>%
  as_tibble(rownames = "SampleID")

# turn otu_table from s4 obj to tibble, with cols: TaxonID, Sample1, Sample2, …, SampleN
otu_df <- otu_table(phy) %>%
  as("matrix") %>%
  {if (!taxa_are_rows(phy)) t(.) else .} %>%
  as_tibble(rownames = "TaxonID")

# make otu_long have one row for every (TaxonID, SampleID) combo where count > 0
otu_long <- otu_df %>%
  pivot_longer(
    cols = -TaxonID,
    names_to  = "SampleID",
    values_to = "Count"
  )

# join otu_long to samp_df so that each row also carries its “study”
otu_with_study <- otu_long %>%
  dplyr::left_join(samp_df, by = "SampleID")

# collect unique set of studies where each TaxonID appears
tax_study <- otu_with_study %>%
  group_by(TaxonID) %>%
  summarize(
    # collapse unique study names into a comma-separated string (or keep as list)
    Studies = paste0(study_name, collapse = "; "),
    # check if Count = 0 for all values for each TaxonID
    Counts_in_study = if (all(Count == 0)) {
      "All 0"
    } else {
      paste0(Count, collapse = "; ")
    }
  ) %>%
  ungroup()

na_species_with_study <- tax_table(phy) %>%
  as("matrix") %>%
  as_tibble(rownames = "TaxonID") %>%
  filter(is.na(species) | species == "") %>%
  dplyr::left_join(tax_study, by = "TaxonID")

na_species_with_study
#> # A tibble: 5 × 10
#>   TaxonID                                superkingdom phylum class order family genu
```

```
#>   <chr>                                      <chr>       <chr>  <chr> <chr> <chr>  <chr
#> 1 k__Bacteria|p__Proteobacteria|c__Gammapr… Bacteria    Prote… Gamm… Ente… Enter… Ente
#> 2 k__Bacteria|p__Proteobacteria|c__Gammapr… Bacteria    Prote… Gamm… Pseu… Pseud… Pseu
#> 3 k__Bacteria|p__Firmicutes|c__Bacilli|o__… Bacteria    Firmi… Baci… Lact… Strep… Stre
#> 4 k__Bacteria|p__Actinobacteria|c__Corioba… <NA>        <NA>   <NA>  <NA>  <NA>   <NA>
#> 5 k__Bacteria|p__Firmicutes|c__Bacilli|o__… Bacteria    Firmi… Baci… Baci… Bacil… Baci
```

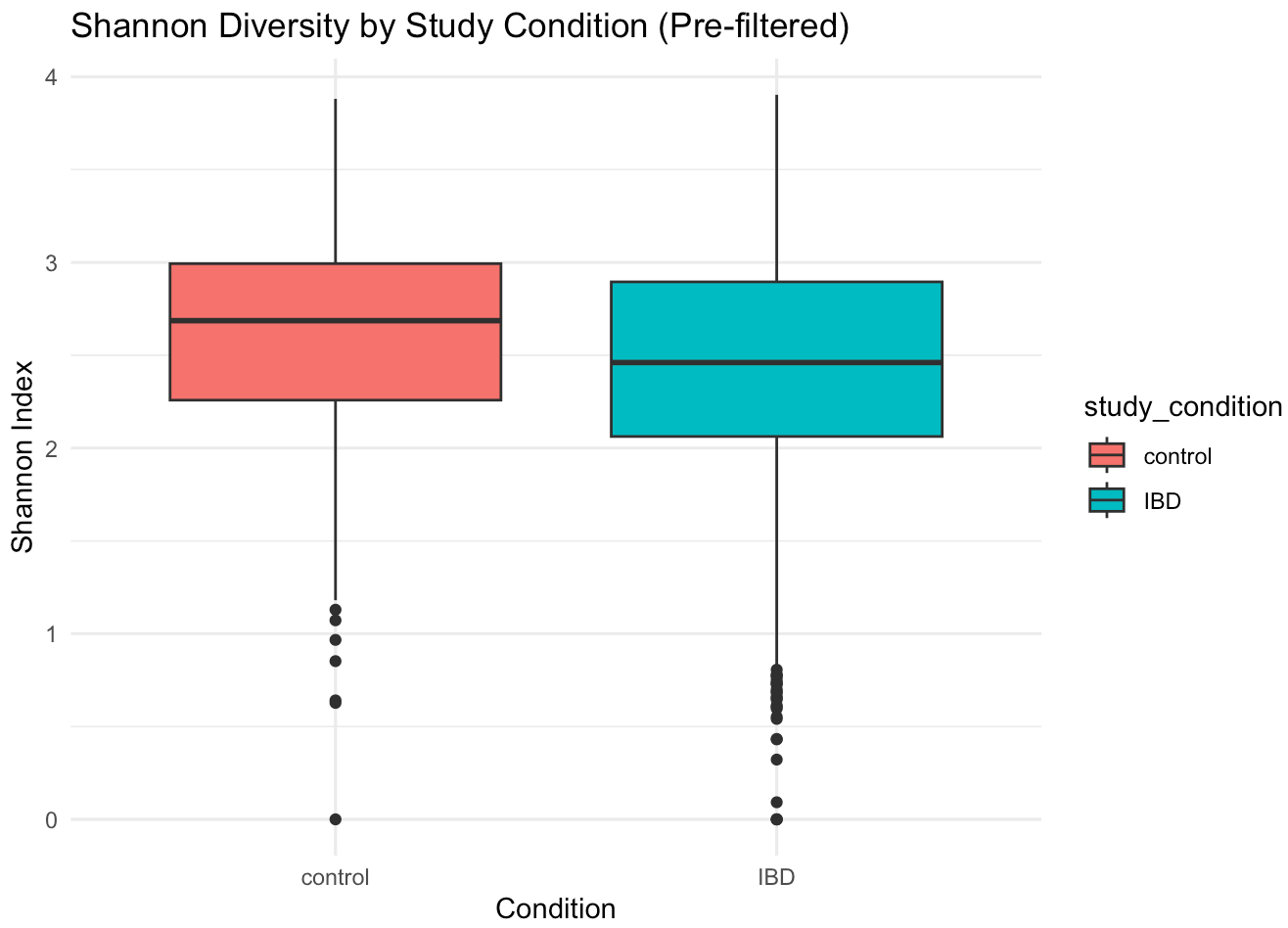
Even though the species is NA, we have counts for the TaxonID, so we should focus on the TaxonID.

α-diversity

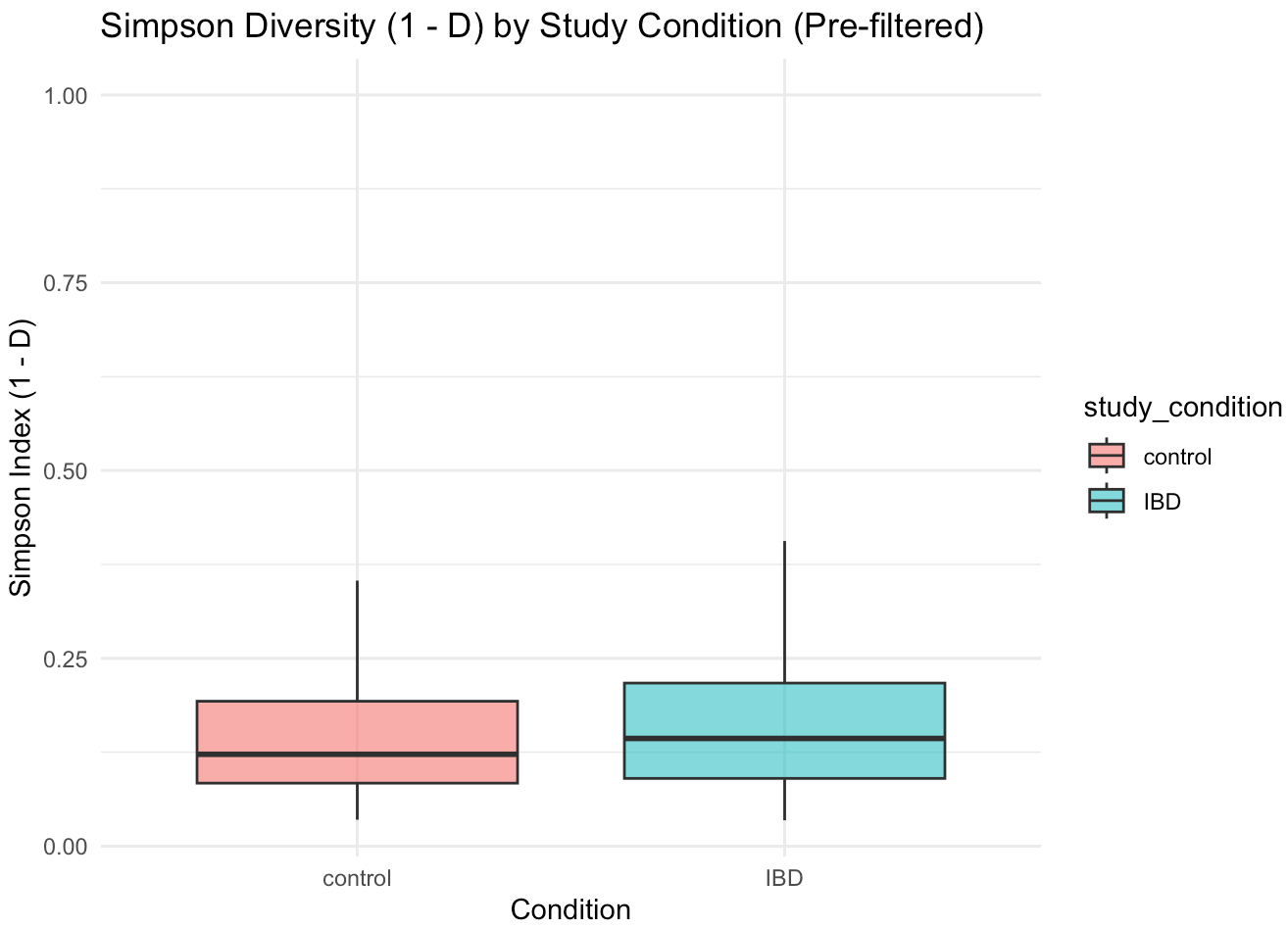
Next, we want to better understand the control vs IBD data. We'll look at α-diversity and use two indices: Shannon and Simpson (see here for further info on Shannon and Simpson), to better understand how "diverse" the community is by combining info about richness (how many taxa) and evenness (how evenly their abundances are distributed).

Note that, since we used `counts = TRUE`, the counts are rounded, which has led to no singletons. This means we cannot estimate richness metrics such as Observed or Chao1. We have left "Observed" as a measure to display the warning that we in fact do not have singletons.

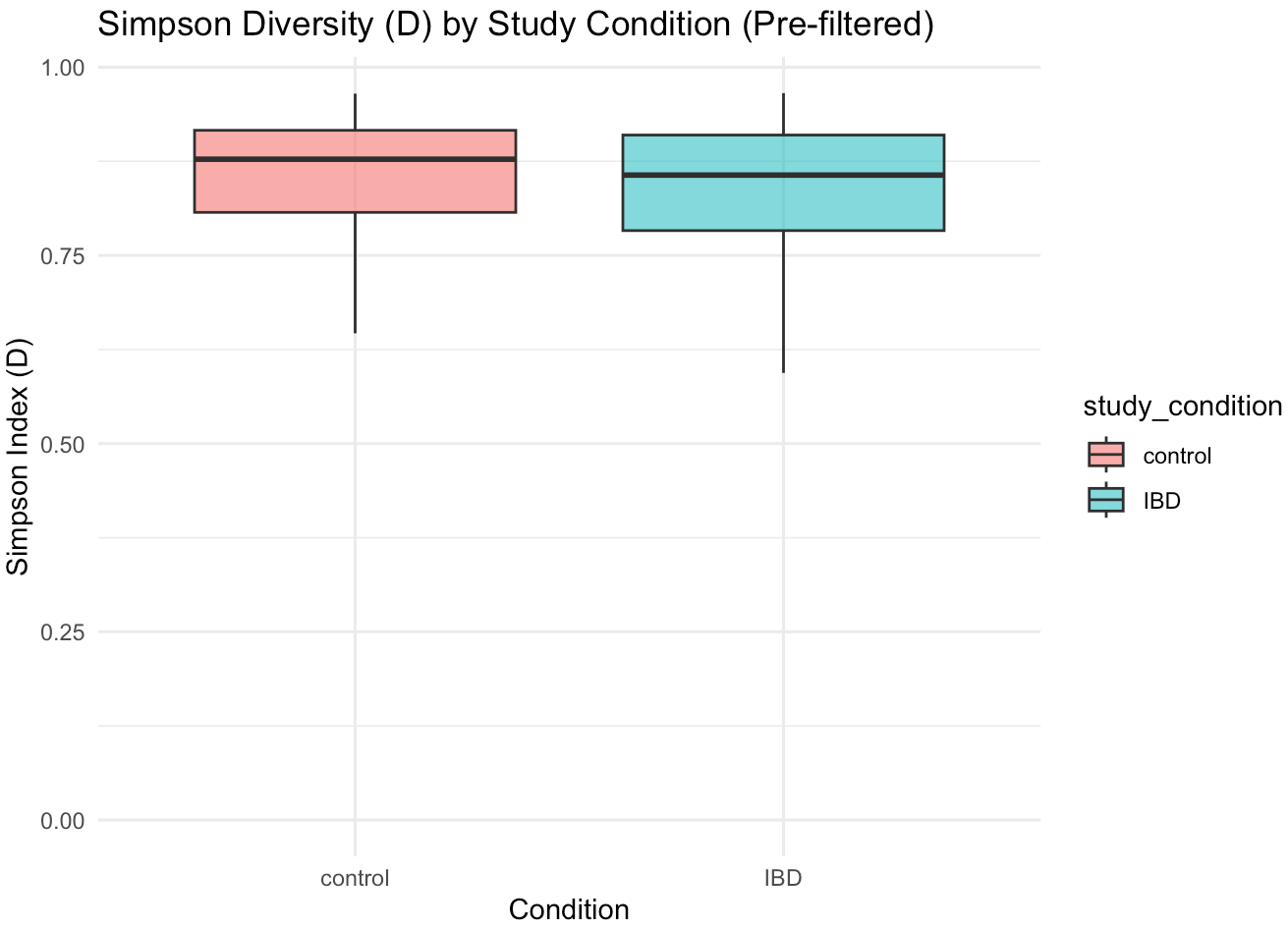
```r
# Extract richness from counts
#    a) remove sequencing noise and artifacts
#    b) reduces overestimation of diversity
#    c) improve comparability across samples
#    d) improve statistical validity (e.g., avoid unneccesarily inflating Type I error)

# estimate_richness provides per-sample diversity indices
alpha_raw <- estimate_richness(physeq =
  phy,
  measures = c("Shannon", "Simpson", "Observed")
)
#> Warning in estimate_richness(physeq = phy, measures = c("Shannon", "Simpson", : The dat
#> any singletons. This is highly suspicious. Results of richness
#> estimates (for example) are probably unreliable, or wrong, if you have already
#> trimmed low-abundance taxa from the data.
#>
#> We recommended that you find the un-trimmed data and retry.

# create df that combines metadata with richness estimates
div_df <- data.frame(
  sample = sample_names(phy),
  study_condition = sample_data(phy)$study_condition,
  Shannon = alpha_raw$Shannon,
  Simpson_D = alpha_raw$Simpson,
  InvSimpson = 1 - alpha_raw$Simpson      # (1 – D)
)

# boxplot Shannon by group
ggplot(div_df, aes(x = study_condition, y = Shannon, fill = study_condition)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Shannon Diversity by Study Condition (Pre-filtered)",
       x = "Condition", y = "Shannon Index")
```




```
# boxplot Simpson Diversity (1-D) by group
ggplot(div_df, aes(x = study_condition, y = InvSimpson, fill = study_condition)) +
  geom_boxplot(outlier.shape = NA, alpha = 0.6) +
  theme_minimal() +
  labs(title = "Simpson Diversity (1 - D) by Study Condition (Pre-filtered)",
       x = "Condition", y = "Simpson Index (1 - D)")
```

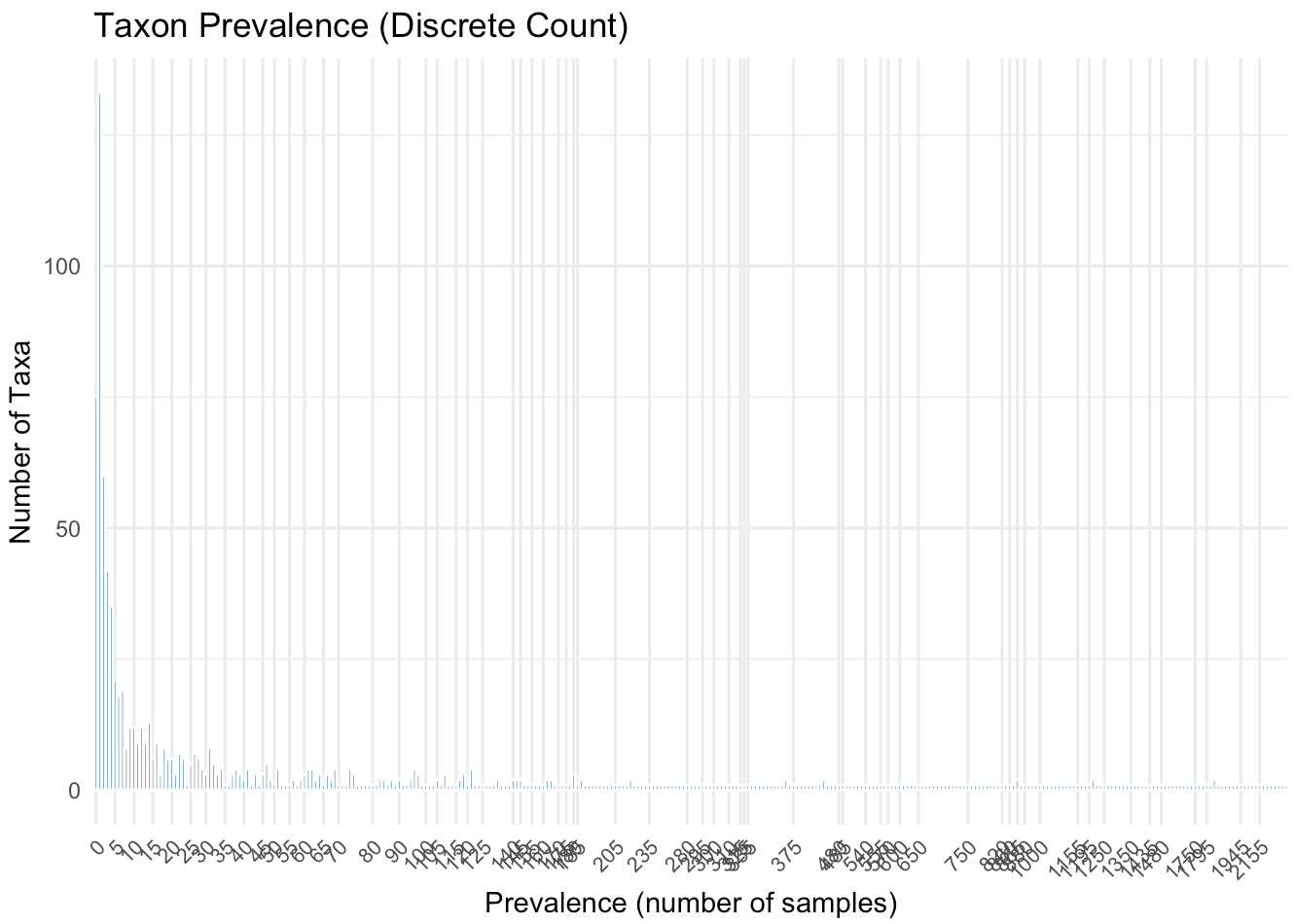



```
# boxplot Simpson Diversity (D) by group
ggplot(div_df, aes(x = study_condition, y = Simpson_D, fill = study_condition)) +
  geom_boxplot(outlier.shape = NA, alpha = 0.6) +
  theme_minimal() +
  labs(title = "Simpson Diversity (D) by Study Condition (Pre-filtered)",
       x = "Condition", y = "Simpson Index (D)")
```


Simpson Diversity (D) by Study Condition (Pre-filtered)
1.00
0.75
0.50
0.25
0.00
Simpson Index (D)
control
IBD
Condition
study_condition
control
IBD

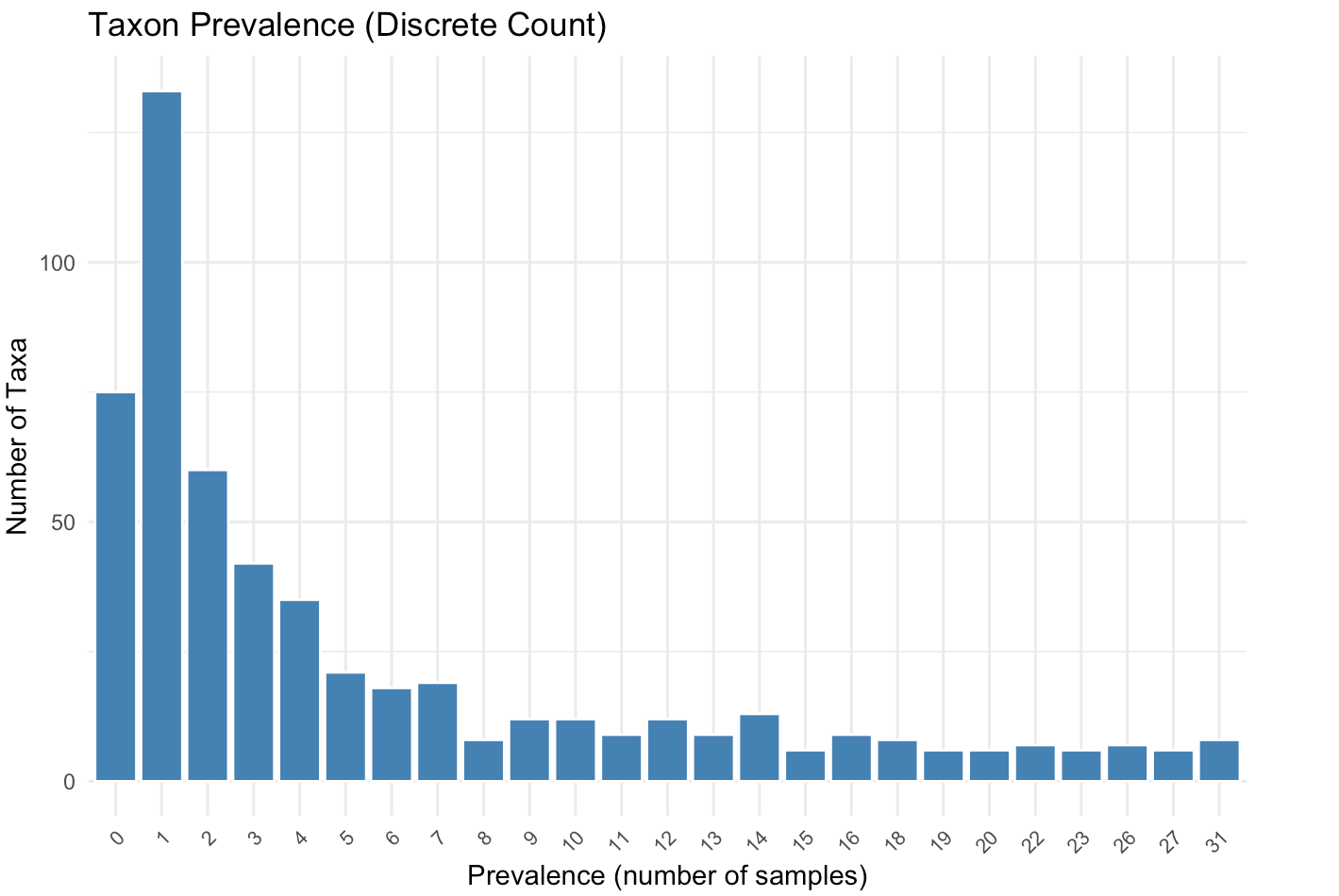
Plot prevalence histogram

```
# extract OTU table as a matrix
otu_mat <- as(otu_table(phy), "matrix")

# make sure rows = taxa, columns = samples. If phy has taxa as columns, transpose it
if (!taxa_are_rows(phy)) {
  otu_mat <- t(otu_mat)
}

# calculate prevalence: number of samples in which each taxon has count > 0
prevalence_df <- data.frame(
  Taxon      = rownames(otu_mat),
  prevalence = rowSums(otu_mat > 0)
) %>%
  mutate(prevalence_prop = prevalence / nsamples(phy)) %>%
  arrange(-prevalence)


# bar plot of prevalence
ggplot(prevalence_df, aes(x = factor(prevalence))) +
  geom_bar(fill = "steelblue", color = "white") +
  theme_minimal() +
  labs(
    title = "Taxon Prevalence (Discrete Count)",
    x     = "Prevalence (number of samples)",
    y     = "Number of Taxa"
  ) +
  scale_x_discrete(
    breaks = as.character(seq(0, max(prevalence_df$prevalence), by = 5))
  ) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 8)
  )
```



View subset

```
# plot subset that have > 5 prevalence
ggplot(prevalence_df %>%
         group_by(prevalence) %>%
         filter(n() > 5) %>%
         ungroup(),
       aes(x = factor(prevalence))) +
  geom_bar(fill = "steelblue", color = "white") +
  theme_minimal() +
  labs(
    title = "Taxon Prevalence (Discrete Count)",
    x     = "Prevalence (number of samples)",
    y     = "Number of Taxa"
  ) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 8)
  )
```




```
# table view
DT::datatable(
  prevalence_df %>%
    filter(prevalence > 5) %>%
    arrange(-prevalence)
  )
```

Search:

k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Ruminococcaceae|g__Faecalibacterium|s__Faecaliba
k__Bacteria|p__Bacteroidetes|c__Bacteroidia|o__Bacteroidales|f__Bacteroidaceae|g__Bacteroides|s__Bacteroide
k__Bacteria|p__Bacteroidetes|c__Bacteroidia|o__Bacteroidales|f__Bacteroidaceae|g__Bacteroides|s__Bacteroide
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Ruminococcaceae|g__Flavonifractor|s__Flavonifracto
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Anaerostipes|s__Anaerostipes_h
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Lachnospiraceae_unclassified|s

k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Fusicatenibacter|s__Fusicatenib
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Blautia|s__Blautia_wexlerae
k__Bacteria|p__Bacteroidetes|c__Bacteroidia|o__Bacteroidales|f__Tannerellaceae|g__Parabacteroides|s__Paraba
k__Bacteria|p__Bacteroidetes|c__Bacteroidia|o__Bacteroidales|f__Bacteroidaceae|g__Bacteroides|s__Bacteroide
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Dorea|s__Dorea_longicatena
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Lachnospiraceae|g__Roseburia|s__Roseburia_inuliniv
k__Bacteria|p__Firmicutes|c__Clostridia|o__Clostridiales|f__Ruminococcaceae|g__Ruthenibacterium|s__Ruthenib

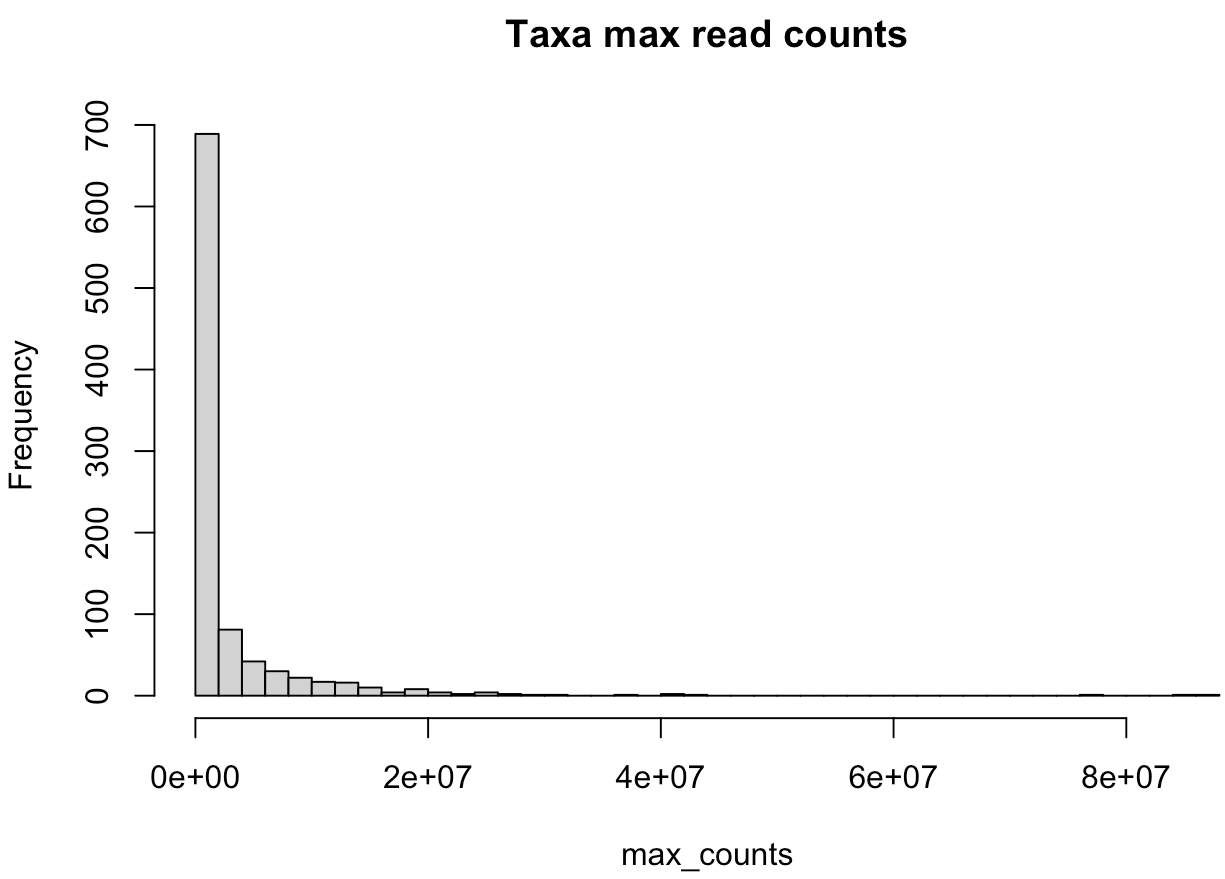
Showing 1 to 574 of 574 entries

Filter low-abundance taxa

Ideally, we capture every taxon. However,

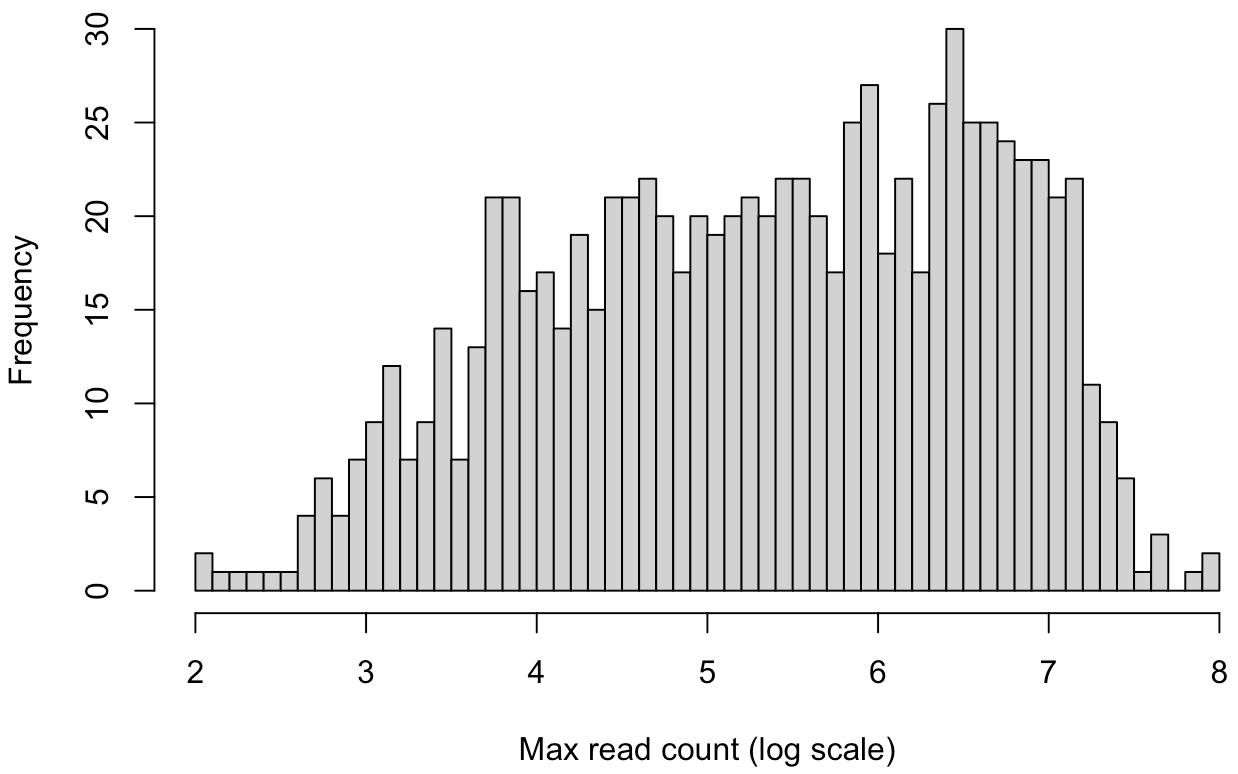
```
# define grid of parameters
params <- expand.grid(
  min_reads            = c(1, 2, 3, 4, 5, 10, 50, 100, 200, 500, 1000, 2000),
  min_prevalence_prop  = c(0.001, 0.01, 0.05, 0.1, 0.2, 0.5), # number of samples
  stringsAsFactors     = FALSE
)

# for each row in params, filter taxa and record how many remain
keep_counts <- params %>%
  pmap_dfr(function(min_reads, min_prevalence_prop) {
    # keep taxa observed in >= min_reads in > min_prevalence_count
    phy_filt <- filter_taxa(
      phy,
      function(x) sum(x >= min_reads) > (min_prevalence_prop * length(x)),
      prune = TRUE
    )

    # create a one-row tibble of results
    tibble(
      min_reads           = min_reads,
      min_prevalence_prop = min_prevalence_prop,
      n_taxa_kept         = ntaxa(phy_filt)
    )
  })

keep_counts %>% arrange(-n_taxa_kept)
#> # A tibble: 72 × 3
#>    min_reads min_prevalence_prop n_taxa_kept
#>        <dbl>               <dbl>       <int>
#>  1         1               0.001         672
#>  2         2               0.001         672
#>  3         3               0.001         672
#>  4         4               0.001         672
#>  5         5               0.001         672
#>  6        10               0.001         672
#>  7        50               0.001         672
#>  8       100               0.001         671
#>  9       200               0.001         669
#> 10       500               0.001         660
#> # ℹ 62 more rows
```

```
max_counts <- apply(otu_table(phy), 1, max)
hist(max_counts, breaks = 50, main="Taxa max read counts")
```



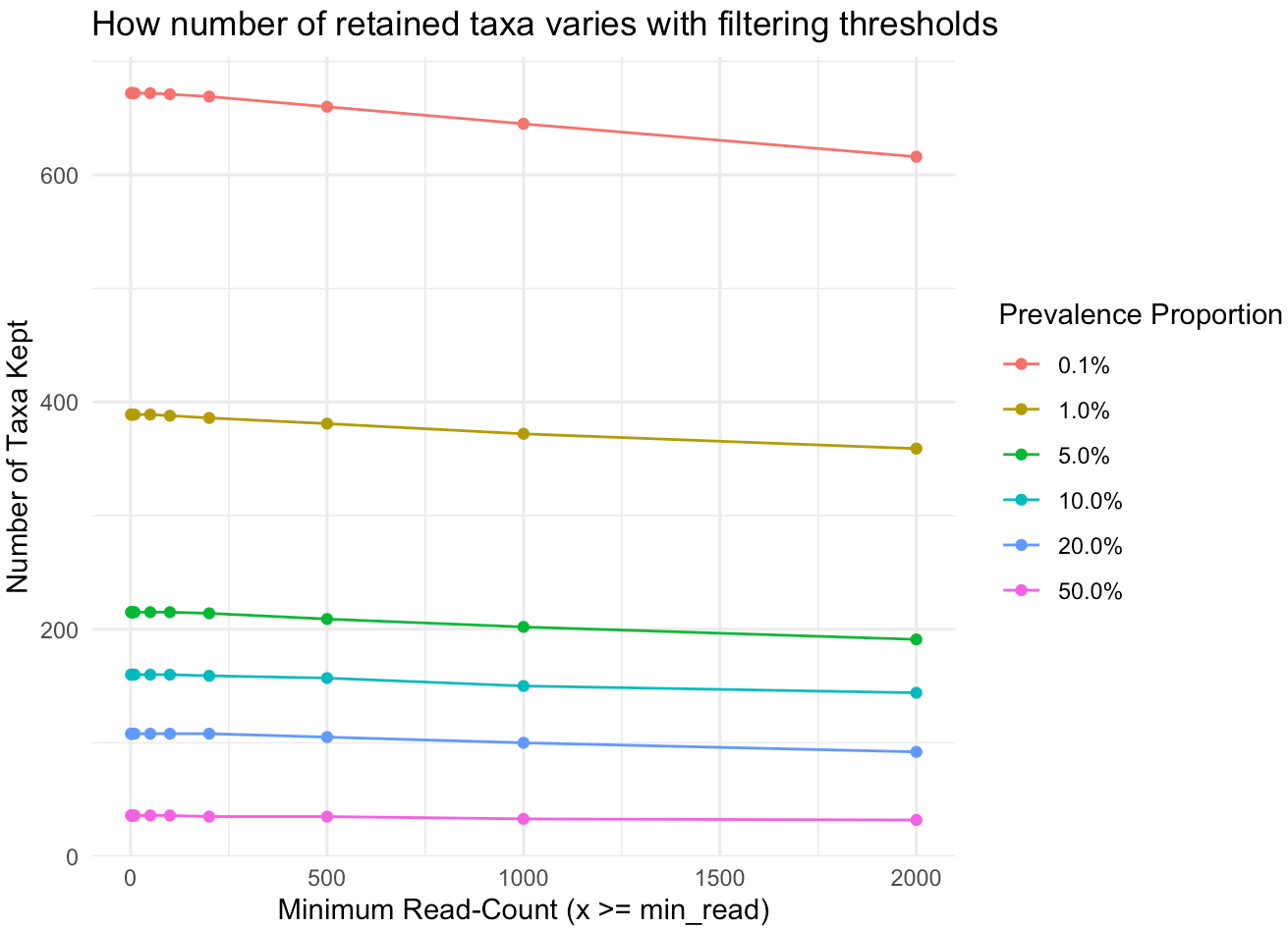
This is hard to read — let's "spread" the data, i.e., log-transform our data.

```
# histogram plot of > 0 log-transformed counts
mc <- max_counts[max_counts > 0]
hist(log10(mc),
     breaks = 50,
     main   = "Log-scaled Taxa Max Read Counts",
     xlab   = "Max read count (log scale)"
     )
```



Note that the left tail (at around 2-3) implies “rare” taxa, while the right tail (at around > 6) implies highly dominant taxa.

```
# plot impact of filtering
ggplot(keep_counts, aes(x = min_reads, y = n_taxa_kept, color = factor(min_prevalence_prop
  geom_line() +
  geom_point() +
  scale_color_discrete(
    name   = "Prevalence Proportion",         # legend title
    labels = function(x) percent(as.numeric(x), accuracy = 0.1)     # 0.05 to 5%
  ) +
  theme_minimal() +
  labs(
    title = "How number of retained taxa varies with filtering thresholds",
    x = "Minimum Read-Count (x >= min_read)",
    y = "Number of Taxa Kept"
  )
```



We need to balance noise removal and biological signal. Let's investigate the phylum breakout to make sure we don't lose phylums with > 1 prevalence.

```
# Check phylum-level breakdown of the M taxa kept to make sure don't filter out entire Phy
# pre-filter
tax_table_df <- as.data.frame(table(tax_table(phy)[, "phylum"])) %>%
  arrange(-Freq) %>%
  rename(
    phylum     = Var1,
    freq_orig  = Freq
  ) %>%
  mutate(
    Percent_phylum_orig = round((freq_orig / sum(freq_orig)) * 100, 1)
  )

str(tax_table_df)
#> 'data.frame':    20 obs. of  3 variables:
#>  $ phylum            : Factor w/ 20 levels "Actinobacteria",..: 12 16 3 1 13 10 18 2 1
#>  $ freq_orig         : int  482 149 140 119 18 5 5 4 3 2 ...
#>  $ Percent_phylum_orig: num  51.4 15.9 14.9 12.7 1.9 0.5 0.5 0.4 0.3 0.2 ...
tax_table_df
#>                         phylum freq_orig Percent_phylum_orig
#> 1                   Firmicutes       482                51.4
#> 2                Proteobacteria       149                15.9
#> 3                  Bacteroidota       140                14.9
#> 4                Actinobacteria       119                12.7
#> 5                  Fusobacteria        18                 1.9
#> 6                 Euryarchaeota         5                 0.5
#> 7                 Synergistetes         5                 0.5
#> 8                    Ascomycota         4                 0.4
#> 9                  Spirochaetes         3                 0.3
#> 10                       Evosea         2                 0.2
#> 11                 Tenericutes         2                 0.2
#> 12                Basidiomycota         1                 0.1
#> 13  Candidatus Melainabacteria         1                 0.1
#> 14 Candidatus Thermoplasmatota         1                 0.1
#> 15                   Chlamydiae         1                 0.1
#> 16                  Chloroflexi         1                 0.1
#> 17                Cyanobacteria         1                 0.1
#> 18                Lentisphaerae         1                 0.1
#> 19              Planctomycetota         1                 0.1
#> 20              Verrucomicrobia         1                 0.1
```

```
# keep taxa with reads >= 100 in at least 1% of samples
phy_filtered <- filter_taxa(phy, function(x) sum(x >= 100) > (0.01 * length(x)), TRUE)
message("Kept ", ntaxa(phy_filtered), " out of ", ntaxa(phy), " taxa.\n")
#> Kept 388 out of 940 taxa.
```

```
# check phylum-level breakdown of the M taxa kept to make sure don't filter out entire Phy
# post-filter
tax_table_filtered_df <- as.data.frame(table(tax_table(phy_filtered)[, "phylum"])) %>%
  arrange(-Freq) %>%
  rename(
    phylum    = Var1,
    freq_sub  = Freq
  )

str(tax_table_filtered_df)
#> 'data.frame':    12 obs. of  2 variables:
#>  $ phylum  : Factor w/ 12 levels "Actinobacteria",..: 7 3 1 10 8 6 11 2 4 5 ...
#>  $ freq_sub: int  234 72 42 27 3 2 2 1 1 1 ...
tax_table_filtered_df
#>                          phylum freq_sub
#> 1                    Firmicutes      234
#> 2                   Bacteroidota       72
#> 3                Actinobacteria       42
#> 4                Proteobacteria       27
#> 5                  Fusobacteria        3
#> 6                 Euryarchaeota        2
#> 7                 Synergistetes        2
#> 8                    Ascomycota        1
#> 9   Candidatus Melainabacteria        1
#> 10 Candidatus Thermoplasmatota        1
#> 11               Lentisphaerae        1
#> 12             Verrucomicrobia        1
```

```
merged_tax_table_df <- merge(
  tax_table_df,
  tax_table_filtered_df,
  by    = "phylum",
  all.x = TRUE
  ) %>%
  mutate(
    # Compute percent of Freq_sub of original
    Percent_phylum_sub_of_orig = round((freq_sub / sum(freq_orig)) * 100, 1),
    Percent_freq_sub_of_orig   = round((freq_sub / freq_orig) * 100, 1)
  )


DT::datatable(merged_tax_table_df)
```

Search:

	phylum	freq_orig	Percent_phylum_orig	freq_sub	Percent_phylum_sub_of_
1	Actinobacteria	119	12.7	42	
2	Ascomycota	4	0.4	1	

3	Bacteroidota	140	14.9	72
4	Basidiomycota	1	0.1	
5	Candidatus Melainabacteria	1	0.1	1
6	Candidatus Thermoplasmatota	1	0.1	1
7	Chlamydiae	1	0.1	
8	Chloroflexi	1	0.1	
9	Cyanobacteria	1	0.1	
10	Euryarchaeota	5	0.5	2
11	Evosea	2	0.2	
12	Firmicutes	482	51.4	234

Showing 1 to 20 of 20 entries

So we end up losing 8 taxa, but these are rare. Recall our task is RF classification. Filtering rare taxa (even up to moderate prevalence thresholds) reduces α-diversity differences between batches and alleviates technical variability, while maintaining β-diversity structure. Additionally, in disease studies, filtering retains significant taxa and preserves RF AUC for classification tasks [1].

We will choose 1% as our threshold, which seems like a good balance of signal vs noise while still retaining a broad range of taxa.

Investigate filtered data

How does “diversity” differ between IBD vs. control

α-diversity often differs in disease vs. healthy (e.g., IBD typically has lower Shannon). We first visualize and then perform a formal test (i.e., Wilcoxon rank-sum test) to qualitatively and quantitatively compare the groups before jumping to our ML analysis.

```
# Extract richness from the raw counts
#    a) remove sequencing noise and artifacts
#    b) reduces overestimation of diversity
#    c) improve comparability across samples
#    d) improve statistical validity (e.g., avoid unneccesarily inflating Type I error)
alpha_raw <- estimate_richness(
  phy_filtered,
  measures = c("Shannon", "Simpson")
)
#> Warning in estimate_richness(phy_filtered, measures = c("Shannon", "Simpson")): The dat
#> any singletons. This is highly suspicious. Results of richness
#> estimates (for example) are probably unreliable, or wrong, if you have already
#> trimmed low-abundance taxa from the data.
#>
#> We recommended that you find the un-trimmed data and retry.

alpha_raw
#>                       Shannon   Simpson
#> SKST006_6_G102964  1.2522940 0.6372417
#> SKST006_7_G102965  1.4010656 0.5467980
#> SKST006_4_G102962  1.7122378 0.7658535
#> SKST006_5_G102963  1.2635459 0.6865648
#> SKST006_2_G102960  0.9780697 0.5907783
#> SKST006_3_G102961  0.6130828 0.4220964
#> SKST006_10_G102994 0.0000000 0.0000000
#> SKST006_1_G102959  1.3779269 0.6894939
#> SKST006_9_G103014  1.4712727 0.6384448
#> SKST027_3_G102945  2.6546096 0.8849364
#> SKST027_4_G102958  2.7693670 0.9025294
#> SKST027_6_G102955  1.0200554 0.6072276
#> SKST027_9_G102947  2.5510789 0.8891747
#> SKST027_11_G102996 2.0612132 0.7799172
#> SKST027_12_G103011 2.2657963 0.8509940
#> SKST027_1_G102984  1.1866775 0.5446189
#> SKST027_2_G102985  2.1089348 0.8399448
#> SKST032_1_G102943  2.3864312 0.8444605
#> SKST032_2_G102946  2.4135732 0.8751413
#> SKST014_3_G102978  1.7020262 0.7173127
#> SKST014_2_G102977  1.7731820 0.6793101
#> SKST014_5_G102980  2.5844154 0.8805813
#> SKST014_4_G102979  2.0348834 0.8489550
#> SKST014_7_G102940  2.4629366 0.8500670
#> SKST014_6_G102951  2.4753211 0.8777362
#> SKST023_2_G102987  1.8026734 0.7462762
#> SKST023_1_G102981  2.0283657 0.7817109
#> SKST023_4_G102937  2.8169363 0.9088080
#> SKST023_3_G102941  2.7045113 0.8781404
#> SKST011_3_G102973  2.8240780 0.9060256
#> SKST011_2_G102972  2.7897078 0.9035860
#> SKST011_1_G102971  2.8115253 0.9067192
#> SKST010_8_G102998  2.6287806 0.8766470
```

```
#> SKST010_7_G103003  2.7272082 0.8715787
#> SKST010_6_G103004  2.3480810 0.8592615
#> SKST010_5_G102990  2.7711954 0.8921098
#> SKST010_4_G102956  2.8546624 0.8897277
#> SKST011_5_G102993  2.7648181 0.8958376
#> SKST011_4_G102952  2.7560408 0.8990970
#> SKST023_7_G103015  3.0339162 0.9251253
#> SKST024_1_G102982  2.2056892 0.8348499
#> SKST023_5_G103007  1.6690423 0.7235394
#> SKST023_6_G103009  2.8821410 0.9141447
#> SKST024_4_G102938  2.8904509 0.8926771
#> SKST024_5_G103001  1.9978854 0.8059146
#> SKST024_2_G102953  2.1638165 0.7435241
#> SKST024_3_G102944  2.8871684 0.9210467
#> SKST024_6_G103012  2.6716317 0.8487322
#> SKST024_7_G103022  2.3202246 0.8072082
#> SKST011_6_G103002  2.8433942 0.9045948
#> SKST011_7_G103017  2.6457497 0.8868671
#> SKST011_8_G103000  2.5949466 0.8581854
#> SKST012_1_G102974  3.2063818 0.9353146
#> SKST012_2_G102975  3.2757311 0.9332595
#> SKST012_3_G102950  2.9107180 0.9018265
#> SKST012_4_G102954  3.2504997 0.9313359
#> SKST012_5_G102992  2.7658572 0.8999757
#> SKST012_6_G102997  3.0290341 0.9162627
#> SKST014_1_G102976  0.6842360 0.4911153
#> SKST036_3_G103025  2.7133186 0.8846836
#> SKST036_2_G103020  2.4866486 0.8723806
#> SKST037_2_G103023  2.1749561 0.7924307
#> SKST032_4_G103008  2.3228094 0.8444105
#> SKST032_3_G102989  2.7601873 0.9161065
#> SKST036_1_G103013  2.5246610 0.8584318
#> SKST032_5_G103024  2.3957859 0.8529960
#> SKST041_1_G103005  2.8809232 0.9108480
#> SKST037_3_G103026  1.2527133 0.6787014
#> SKST041_2_G103027  2.0987718 0.8033809
#> SKST041_3_G103028  2.2962906 0.8474506
#> SKST010_2_G102969  2.5482905 0.8826525
#> SKST010_3_G102970  3.3414326 0.9396790
#> SKST007_8_G102999  2.6317371 0.8958339
#> SKST010_1_G102968  3.4896605 0.9544864
#> SKST007_6_G102995  2.6634173 0.8338101
#> SKST007_7_G103016  2.8152047 0.9110826
#> SKST007_3_G102949  2.9240942 0.9226028
#> SKST007_4_G102948  2.6810693 0.8931690
#> SKST007_1_G102966  2.7810739 0.9093627
#> SKST007_2_G102967  2.5204322 0.8741401
#> SKST027_10_G102936 1.8598491 0.8024193
#> SKST025_9_G103021  1.5556967 0.6776121
#> SKST025_2_G102986  2.6696480 0.8958129
#> SKST025_1_G102983  2.7781552 0.8919120
#> SKST024_8_G103031  2.3069551 0.8079908
```

```
#> SKST025_6_G103010  1.5780891 0.7437443
#> SKST025_5_G103006  2.0191108 0.7401280
#> SKST025_4_G102939  1.8767355 0.7931885
#> SKST025_3_G102942  2.4100870 0.8393832
#> p8883_mo6          2.7980886 0.9148663
#> p9223_mo4          2.1881319 0.6885799
#> p9223_mo3          2.7076171 0.8123768
#> p9223_mo2          2.7358612 0.8100018
#> p9223_mo1          2.0431248 0.6344728
#> p9220_mo7          2.9987451 0.9184255
#> p9220_mo6          2.7864065 0.8938321
#> p9220_mo5          3.2924001 0.9469014
#> p9220_mo4          3.1979251 0.9360076
#> p9220_mo3          3.2498214 0.9433278
#> p9220_mo2          3.1767250 0.9310763
#> p8646_mo8          3.2712163 0.9467655
#> p8646_mo9          3.1069294 0.9333765
#> p8646_mo6          2.8863611 0.8884233
#> p8646_mo7          2.9868318 0.9208884
#> p8649_mo11         2.1949472 0.7365005
#> p8649_mo12         2.7410957 0.8342998
#> p8649_mo1          3.3293681 0.9484852
#> p8649_mo10         3.2217033 0.9339763
#> p8649_mo2          3.1776585 0.9329343
#> p8649_mo3          2.9045244 0.8851852
#> p8582_mo5          3.2352955 0.9424675
#> p8582_mo7          3.2676648 0.9412434
#> p8582_mo6          3.2571216 0.9458514
#> p9281_mo5          3.1516610 0.9288301
#> p9281_mo6          1.9488341 0.7903650
#> p9223_mo7          2.7323455 0.8154577
#> p9223_mo8          2.8639164 0.8480329
#> p9223_mo5          3.1969248 0.9219603
#> p9223_mo6          2.7585885 0.8478928
#> p9281_mo3          3.2784516 0.9429546
#> p9281_mo4          3.0690481 0.9239806
#> p9281_mo1          3.3366552 0.9415510
#> p9281_mo2          3.1349278 0.9249857
#> p8646_mo3          2.8212167 0.8770671
#> p8646_mo2          2.6068442 0.8508834
#> p8646_mo12         3.1970799 0.9379695
#> p8646_mo11         3.1906148 0.9385366
#> p8646_mo10         3.1904190 0.9328649
#> p8646_mo1          2.4563344 0.8190123
#> p8600_mo9          2.7306945 0.8812102
#> p8600_mo8          2.5054908 0.8435909
#> p8646_mo5          3.3252302 0.9431623
#> p8646_mo4          3.2608190 0.9272229
#> p8808_mo7          2.6817387 0.8918650
#> p8808_mo6          2.8267354 0.9175881
#> p8808_mo9          2.6598647 0.8968506
#> p8808_mo8          2.7542698 0.9090572
```

```
#> p8808_mo12         2.1685183 0.8268748
#> p8808_mo11         2.0175372 0.7927332
#> p8883_mo2          2.9641448 0.9164362
#> p8808_mo4          2.4652800 0.8516828
#> p8816_mo10         2.8854457 0.9153551
#> p8816_mo1          3.2642581 0.9372518
#> p8883_mo3          2.8320962 0.8885733
#> p8600_mo1          2.3235612 0.8419463
#> p8600_mo10         2.5144408 0.8568160
#> p8600_mo11         2.4295687 0.8235414
#> p8600_mo12         2.5358425 0.8374550
#> p8600_mo2          2.7809702 0.9032680
#> p8600_mo3          2.8533362 0.9127022
#> p8600_mo4          2.5195770 0.8483133
#> p8600_mo5          2.9446139 0.9161902
#> p8600_mo6          3.1102496 0.9345201
#> p8600_mo7          2.3905778 0.8205733
#> p8775_mo7          2.9502307 0.9082173
#> p8775_mo8          3.2512044 0.9410491
#> p8775_mo5          3.0296708 0.9141398
#> p8775_mo6          3.2244045 0.9381442
#> p8775_mo3          3.0645091 0.9189255
#> p8775_mo4          3.0852510 0.9217384
#> p8775_mo12         2.9004152 0.9077146
#> p8775_mo2          2.7901970 0.8848275
#> p8775_mo9          2.6037354 0.8563312
#> p8808_mo10         2.7467400 0.9204208
#> p8816_mo8          2.8811862 0.9125761
#> p8816_mo9          2.9347897 0.9134674
#> p8816_mo4          3.0869504 0.9222314
#> p8816_mo5          2.9759955 0.9165706
#> p8816_mo6          2.9136666 0.9110562
#> p8816_mo7          2.9174809 0.9160573
#> p8816_mo11         3.0211472 0.9228923
#> p8816_mo12         2.9358613 0.9036371
#> p8816_mo2          2.7700241 0.8799520
#> p8816_mo3          3.2272053 0.9394574
#> p8585_mo9          3.0697299 0.9066927
#> p8585_mo8          2.5014031 0.7884440
#> p8585_mo4          3.1355768 0.9298240
#> p8585_mo3          2.7160162 0.9054014
#> p8585_mo7          2.7432172 0.8466796
#> p8585_mo6          2.8369090 0.8679917
#> p8582_mo9          3.2366453 0.9410905
#> p8582_mo8          3.3240788 0.9418381
#> p8585_mo2          2.9459631 0.9122657
#> p8585_mo1          2.6820190 0.9014596
#> p8883_mo12         2.5724065 0.8754515
#> p8883_mo11         2.8374424 0.9141775
#> p8855_mo3          2.9524064 0.9244398
#> p8855_mo12         2.6117483 0.8739644
#> p8855_mo11         2.5363042 0.8779739
```

```
#> p8855_mo10         2.0196478 0.7770400
#> p8883_mo10         2.9630591 0.9199234
#> p8855_mo9          2.4258523 0.8606390
#> p8855_mo8          2.6717697 0.8905552
#> p8855_mo4          3.0695902 0.9328922
#> p8808_mo5          2.7017634 0.9061237
#> p8883_mo9          2.6951496 0.8936411
#> p9061_mo1          1.9915813 0.7012513
#> p8883_mo7          2.8513694 0.9146848
#> p8883_mo8          2.7596145 0.9069852
#> p8883_mo4          2.8667172 0.9232376
#> p8582_mo11         3.1501048 0.9298924
#> p8582_mo10         3.2368306 0.9348806
#> p8582_mo1          3.2097885 0.9376378
#> p9061_mo2          1.6375674 0.6029582
#> p9061_mo3          1.5964755 0.5363953
#> p8775_mo11         2.9130216 0.9003362
#> p8775_mo10         3.1864834 0.9229290
#> p8748_mo6          3.0807144 0.9266031
#> p8748_mo5          3.1427895 0.9314801
#> p8748_mo4          2.7276586 0.8711416
#> p8748_mo12         3.1616456 0.9320979
#> p8775_mo1          3.0019386 0.9131593
#> p8748_mo9          3.1620400 0.9328056
#> p8748_mo8          2.9495557 0.8995987
#> p8748_mo7          2.9105501 0.9105712
#> p9193_mo1          3.1057316 0.9187077
#> p9061_mo8          1.2805019 0.4355977
#> p9193_mo4          2.6846294 0.8412674
#> p9193_mo2          2.5539648 0.8535735
#> p9061_mo5          1.6246152 0.5817584
#> p9061_mo4          1.6147715 0.5975075
#> p9061_mo7          1.8977298 0.6684717
#> p9061_mo6          1.6851856 0.5794057
#> p9193_mo6          3.1315381 0.9104450
#> p9193_mo5          3.1930885 0.9291899
#> p8712_mo6          2.7492731 0.8738769
#> p8712_mo7          2.8948333 0.9115400
#> p8712_mo8          3.1432878 0.9335759
#> p8712_mo9          2.8835082 0.9130206
#> p8712_mo2          2.9265593 0.9094713
#> p8712_mo3          2.7109820 0.8989880
#> p8712_mo4          3.1953152 0.9270473
#> p8712_mo5          2.6224049 0.8899828
#> p8748_mo10         3.1398351 0.9250578
#> p8748_mo11         3.0593933 0.9268043
#> p9193_mo7          2.8919771 0.9012290
#> p9216_mo1          3.2428005 0.9381028
#> p9216_mo2          0.6453250 0.4529451
#> p9216_mo3          3.3974279 0.9498241
#> p9216_mo4          3.1197677 0.9348338
#> p9216_mo5          3.0389745 0.9222445
```

```
#> p9216_mo6          2.9881114 0.9199452
#> p9216_mo7          2.8526506 0.8921778
#> p9216_mo8          2.8344539 0.8838441
#> p9220_mo1          2.9962468 0.9098635
#> p8649_mo5          1.5583795 0.6939676
#> p8649_mo4          3.0511013 0.9240628
#> p8649_mo7          2.6524014 0.8808904
#> p8649_mo6          2.9850130 0.9112910
#> p8649_mo9          3.2421201 0.9397685
#> p8649_mo8          3.1185726 0.9334617
#> p8712_mo10         3.1243752 0.9317430
#> p8712_mo1          3.2938875 0.9274425
#> p8712_mo12         3.0672867 0.9265143
#> p8712_mo11         2.9264568 0.9248038
#> p8582_mo4          2.8042167 0.8951669
#> p8582_mo3          3.2326613 0.9420385
#> p8582_mo2          3.2542472 0.9448576
#> p8582_mo12         2.9923019 0.9143940
#> CSM5FZ3N_P         1.6520594 0.7114941
#> CSM5FZ3R_P         1.4575519 0.6394882
#> CSM5YRY7_P         1.7427193 0.7787577
#> CSM5FZ3V_P         1.6696654 0.7271918
#> CSM5FZ4C_P         1.8650021 0.7995602
#> CSM5MCVD_P         1.4817786 0.5847641
#> CSM5MCVF_P         1.5700839 0.6936967
#> CSM5MCVV_P         2.0499632 0.7855607
#> CSM5MCWI_P         1.5890611 0.7133541
#> CSM5MCXD           1.8335070 0.6998957
#> CSM5MCYS           2.3497039 0.8392791
#> CSM67U9J           1.6994994 0.6963556
#> CSM67UA2           1.8170014 0.7582638
#> CSM67UGC           2.2371059 0.8500937
#> CSM79HG5           1.4434056 0.6974857
#> CSM79HGP           1.8406152 0.7472978
#> CSM5FZ3T_P         1.0715167 0.4989353
#> CSM5FZ3X_P         1.0820570 0.4799621
#> CSM5FZ3Z_P         1.4495247 0.6820875
#> CSM5FZ42_P         1.1987048 0.4665043
#> CSM5FZ44_P         1.2666094 0.5628854
#> CSM5FZ46_P         1.1325944 0.5775475
#> CSM5MCVJ_P         1.0825625 0.5584290
#> CSM5MCVL           1.4703354 0.6291225
#> CSM5MCVN           1.3819303 0.6077765
#> CSM67UBF           1.3302051 0.6089956
#> CSM67UBH           1.6581611 0.7099598
#> CSM67UBN           1.2096162 0.5201594
#> CSM67UBR           1.3971923 0.6368677
#> CSM79HJW           1.2396614 0.5559902
#> CSM79HJY           1.5662045 0.6754470
#> CSM5FZ4E_P         2.0671811 0.7866476
#> CSM5FZ4G_P         2.2676452 0.8479068
#> CSM5FZ4K_P         2.2956714 0.8365199
```

```
#> CSM5FZ4M          1.8789970 0.7682081
#> CSM5MCWM_P        1.6700215 0.6949797
#> CSM5MCWQ          2.5182501 0.8687125
#> CSM67UBX          2.2446969 0.8161451
#> CSM67UBZ          2.2782451 0.8363078
#> CSM67UC6          2.2371169 0.7958937
#> CSM79HLM          2.2323573 0.8163183
#> CSM5FZ4A_P        1.7745244 0.7636662
#> CSM5MCU8_P        2.7507206 0.8943257
#> CSM5MCUA_P        1.9482560 0.7901734
#> CSM5MCUC_P        1.9492407 0.7612829
#> CSM5MCUE_P        2.2063365 0.8290263
#> CSM5MCXH          1.8688807 0.7496933
#> CSM5MCXJ          2.0342706 0.7927273
#> CSM5MCXL          2.2213913 0.8274654
#> CSM5MCXN          2.5865278 0.8871789
#> CSM5MCXP          2.3700899 0.8488629
#> CSM5MCXR          2.2646426 0.8396055
#> CSM67UDF          2.1672023 0.8199337
#> CSM67UDJ          2.1026285 0.8108594
#> CSM67UDN          2.1456938 0.8210810
#> CSM67UDR          2.2462528 0.8413014
#> CSM67UDR_TR       2.4136372 0.8556242
#> CSM67UDY          2.1115567 0.8096098
#> CSM79HLA          3.0016989 0.9226866
#> CSM79HLA_TR       2.7229365 0.8944191
#> CSM79HLG          2.5316464 0.8740511
#> CSM79HLE          2.0075122 0.7808684
#> CSM79HLC          2.0679304 0.7955003
#> CSM79HLI          1.9597261 0.7719270
#> CSM79HLK          2.2393932 0.8426094
#> CSM5MCUQ_P        2.6783770 0.8637182
#> CSM5MCUS_P        2.6921982 0.9067930
#> CSM5MCUW_P        2.4626218 0.8583972
#> CSM5MCUY_P        2.3973860 0.8519193
#> CSM5MCY4          2.4670861 0.8593528
#> CSM5MCY8          2.4660379 0.8562897
#> CSM67UE3          2.5326477 0.8773569
#> CSM67UE7          2.4140401 0.8555483
#> CSM67UEA          2.4856349 0.8555281
#> CSM67UEM          2.2832589 0.8613496
#> CSM67UEI          2.7682715 0.9099509
#> CSM79HO1          2.3205384 0.8749319
#> CSM5MCTZ_P        2.6005596 0.8765023
#> CSM5MCUG_P        2.3055449 0.7895238
#> CSM5MCUK_P        2.5135075 0.8455467
#> CSM5MCUO          2.6080384 0.8605206
#> CSM5MCX3          2.7068830 0.8533647
#> CSM67UFV          2.6571884 0.8766414
#> CSM67UFZ          2.3020577 0.8272115
#> CSM67UG8          2.6087727 0.8675150
#> CSM79HMN          2.5662174 0.8616955
```

```
#> CSM79HMP          2.7397032 0.8683731
#> CSM79HMT          2.4884927 0.8462184
#> CSM5MCVB_P        0.7260058 0.2897553
#> CSM5MCV1_P        1.3888298 0.6346785
#> CSM5MCV5_P        2.3639086 0.8507675
#> CSM5MCU4_P        1.2578061 0.4464664
#> CSM5MCVZ_P        0.7328460 0.2477585
#> CSM5MCW4_P        0.6529225 0.2301479
#> CSM5MCW6          1.0731881 0.3700803
#> CSM5MCYW          1.2099423 0.4223702
#> CSM5MCZ3          2.0581724 0.7055605
#> CSM5MCZ5          1.6620127 0.6904695
#> CSM5MCZ7          1.3028416 0.4870008
#> CSM79HKT          1.0890029 0.3941205
#> CSM79HKV          1.0304231 0.3742944
#> CSM79HQ9          0.5412542 0.1816627
#> CSM79HQB          3.2261264 0.9350468
#> CSM79HQF          0.8414003 0.2826816
#> CSM5MCWA_P        2.4202420 0.8492143
#> CSM5MCWC          1.4565385 0.5962797
#> CSM5MCWE          1.9258630 0.7983450
#> CSM5MCWG          0.9201713 0.4039083
#> CSM67UAA          2.2373656 0.8362436
#> CSM67UAG          2.4895809 0.8919591
#> CSM79HI3          2.1803656 0.8120504
#> CSM79HI7          2.3412222 0.8647213
#> CSM79HIB          1.8860286 0.7353994
#> CSM7KOJE          2.2914175 0.8586280
#> CSM7KOJG          2.3790257 0.8656365
#> CSM7KOJO          1.9526797 0.7721331
#> CSM5MCWK_P        2.4267912 0.8408240
#> CSM5MCXT          2.5034042 0.8629637
#> CSM5MCXV          2.4149949 0.8427963
#> CSM5MCXX_P        2.4782326 0.8426309
#> CSM5MCXZ_P        2.3393009 0.8639190
#> CSM5MCY2          2.4352973 0.8317373
#> CSM67UCK          2.5340254 0.8952975
#> CSM79HK9          2.4231148 0.8229437
#> CSM79HKB          2.2074003 0.7743956
#> CSM79HOF          2.5418416 0.8765570
#> CSM79HOH          2.2236296 0.7612016
#> CSM7KOL4          2.3670605 0.8311296
#> CSM7KOLA          2.7662853 0.8967304
#> CSM7KOLE          2.3005350 0.7919191
#> CSM5MCXB_P        2.7322093 0.8721046
#> CSM5MCYI_P        2.6332278 0.8557106
#> CSM5MCYM_P        2.8302765 0.9107432
#> CSM5MCYO_P        3.0007114 0.9142543
#> CSM5MCYQ_P        2.7967775 0.9066532
#> CSM67UEP_P        2.7550914 0.8948342
#> CSM67UET_P        2.3697001 0.8474677
#> CSM67UEW_TR       2.5824682 0.8373300
```

```
#> CSM67UEW_P         2.5318625 0.8140992
#> CSM67UEW           2.6356918 0.8546149
#> CSM67UF1_P         2.8194173 0.9076976
#> CSM67UF1           2.8531779 0.9169461
#> CSM67UF5           2.8628972 0.9188387
#> CSM79HM1           2.7156725 0.9005944
#> CSM79HQT_P         2.8251836 0.9098684
#> CSM79HM5_P         2.3679320 0.8465712
#> CSM79HM7           2.7485613 0.9105959
#> CSM79HM9_P         2.8005309 0.8832614
#> CSM7KOMP           2.8358852 0.8953962
#> CSM7KOMR_P         2.7017806 0.8848653
#> CSM7KOMT           2.9099990 0.9149722
#> CSM7KOMV_P         2.9196537 0.9220534
#> CSM5MCXF_P         2.5083123 0.8641082
#> CSM5MCZB           2.2194506 0.8146034
#> CSM5MCZD           2.7450952 0.9040306
#> CSM5MCZF           2.6247389 0.8813408
#> CSM67U9B           2.7143194 0.9020766
#> CSM67U9D           2.7771167 0.9069387
#> CSM67UGO           2.5587765 0.8694826
#> CSM79HOJ           2.1863259 0.7821766
#> CSM79HOL           2.6010945 0.8844704
#> CSM79HOT           2.5970640 0.8900031
#> CSM7KOMX           2.4647684 0.8697834
#> CSM7KOMZ           2.5156831 0.8763176
#> CSM7KON2           2.6834510 0.8982009
#> CSM7KON8           2.6645752 0.8962852
#> CSM5MCYU_P         1.6419736 0.6856381
#> CSM67U9H_P         1.0814877 0.4259980
#> CSM67U9H           1.1086691 0.4454290
#> CSM67U9N           2.6171500 0.9074400
#> CSM67U9P_P         2.3110086 0.8468533
#> CSM67U9R_P         2.1193957 0.8115351
#> CSM79HGD_P         2.7411134 0.9177279
#> CSM79HGF_P         2.8224090 0.9149204
#> CSM79HGF           2.8160284 0.9138864
#> CSM79HGH_P         2.5633546 0.8883050
#> CSM79HGJ_P         1.8994297 0.7640444
#> CSM79HGL_P         2.6310666 0.9020998
#> CSM79HGN_P         1.0104367 0.3468809
#> CSM79HPK           0.9758634 0.3272030
#> CSM79HPM_P         2.1351373 0.7567209
#> CSM79HPS           1.4860028 0.5256804
#> CSM79HPQ_P         1.1539386 0.4091165
#> CSM79HPO           1.4297854 0.5699975
#> CSM79HPU           1.1379207 0.6090767
#> CSM7KONS_P         0.8148724 0.2822093
#> CSM7KONU           2.1304211 0.7932978
#> CSM7KONW_P         2.3828223 0.8349342
#> CSM67U9T_P         2.0464678 0.8216547
#> CSM67UAK           2.1528099 0.8106580
```

```
#> CSM67UAM          1.9515439 0.7983612
#> CSM67UAO          1.9364527 0.7971388
#> CSM67UAQ          1.7762350 0.7753406
#> CSM67UAS          2.1299781 0.7853237
#> CSM79HID          1.9685105 0.7990658
#> CSM79HIF          1.7683507 0.7706143
#> CSM79HIH          1.9796532 0.8123354
#> CSM79HIJ          1.8338575 0.7338011
#> CSM79HIL          1.6184739 0.6440276
#> CSM79HIN          2.1831085 0.8183828
#> CSM7KOJQ          1.8788489 0.7585125
#> CSM7KOJS          1.6858282 0.6963451
#> CSM7KOJU          1.5010562 0.6118940
#> CSM7KOJW          2.1501010 0.8374249
#> CSM7KOJY          1.6974945 0.6822248
#> CSM7KOK1          1.6782081 0.6933858
#> CSM7KOPG          1.7543722 0.7495756
#> CSM7KOPI          1.9365718 0.7715212
#> CSM7KOPK          1.8292069 0.7474934
#> CSM7KOPM          1.8125224 0.7236093
#> CSM7KOPO          2.1260201 0.7881507
#> CSM67U9V_P        2.2850727 0.8458366
#> CSM67UAU          2.4508029 0.8511657
#> CSM67UAW          2.3430000 0.8259059
#> CSM67UAY          2.4384515 0.8415933
#> CSM67UB1          2.3451076 0.8193004
#> CSM67UB3          2.3129413 0.7989170
#> CSM79HIR          2.3050103 0.8071099
#> CSM79HIT          2.7919294 0.8975031
#> CSM79HIV          2.4568749 0.8371586
#> CSM79HIX          2.3011099 0.8545825
#> CSM79HIZ          2.3809022 0.8508787
#> CSM7KOK3          2.3392459 0.8485317
#> CSM7KOK5          2.4744151 0.8883133
#> CSM7KOK7          2.5436889 0.8838503
#> CSM7KOKB          2.2479679 0.8653775
#> CSM7KOKD          2.4038986 0.8890229
#> CSM7KOPS          2.6354585 0.8994741
#> CSM7KOPU          2.2475654 0.8539026
#> CSM7KOPW          2.1918940 0.8393966
#> CSM7KOQ1          2.3885893 0.8693312
#> CSM67U9X_P        1.6924441 0.5937814
#> CSM67UB5_P        2.3604698 0.8442756
#> CSM67UB7_P        2.4756172 0.8415878
#> CSM67UB9_P        1.2949859 0.6056482
#> CSM67UB9          1.2348934 0.5880483
#> CSM67UBB          1.7436522 0.7094986
#> CSM79HJ2_P        1.7789146 0.6674772
#> CSM79HJ4_P        1.9121700 0.7122009
#> CSM79HJ6_P        2.5426781 0.8747581
#> CSM79HJ8_P        2.6754786 0.8799124
#>  [ reached 'max' / getOption("max.print") -- omitted 2387 rows ]
```

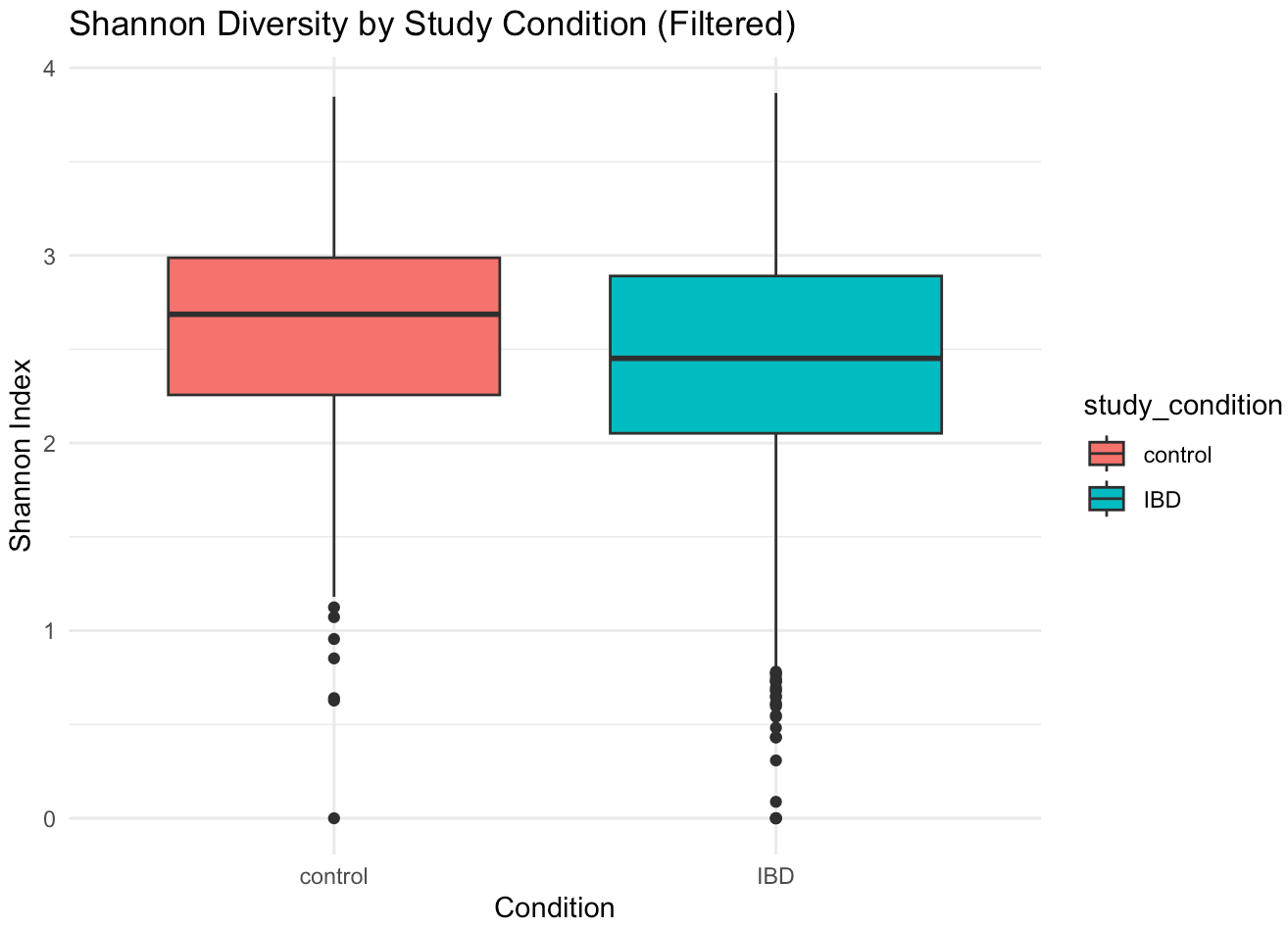
```
# Build a data.frame that combines metadata with these richness estimates.
div_df <- data.frame(
  sample          = rownames(alpha_raw),
  study_condition = sample_data(phy_filtered)$study_condition,
  Shannon         = alpha_raw$Shannon,
  Simpson_D       = alpha_raw$Simpson,       # raw D
  InvSimpson      = 1 - alpha_raw$Simpson    # (1 - D)
)
div_df
#>                 sample study_condition   Shannon Simpson_D InvSimpson
#> 1    SKST006_6_G102964             IBD 1.2522940 0.6372417 0.36275833
#> 2    SKST006_7_G102965             IBD 1.4010656 0.5467980 0.45320196
#> 3    SKST006_4_G102962             IBD 1.7122378 0.7658535 0.23414654
#> 4    SKST006_5_G102963             IBD 1.2635459 0.6865648 0.31343520
#> 5    SKST006_2_G102960             IBD 0.9780697 0.5907783 0.40922169
#> 6    SKST006_3_G102961             IBD 0.6130828 0.4220964 0.57790363
#> 7   SKST006_10_G102994         control 0.0000000 0.0000000 1.00000000
#> 8    SKST006_1_G102959             IBD 1.3779269 0.6894939 0.31050609
#> 9    SKST006_9_G103014             IBD 1.4712727 0.6384448 0.36155524
#> 10   SKST027_3_G102945             IBD 2.6546096 0.8849364 0.11506357
#> 11   SKST027_4_G102958             IBD 2.7693670 0.9025294 0.09747058
#> 12   SKST027_6_G102955             IBD 1.0200554 0.6072276 0.39277236
#> 13   SKST027_9_G102947             IBD 2.5510789 0.8891747 0.11082530
#> 14  SKST027_11_G102996             IBD 2.0612132 0.7799172 0.22008285
#> 15  SKST027_12_G103011             IBD 2.2657963 0.8509940 0.14900603
#> 16   SKST027_1_G102984             IBD 1.1866775 0.5446189 0.45538110
#> 17   SKST027_2_G102985             IBD 2.1089348 0.8399448 0.16005522
#> 18   SKST032_1_G102943             IBD 2.3864312 0.8444605 0.15553946
#> 19   SKST032_2_G102946             IBD 2.4135732 0.8751413 0.12485866
#> 20   SKST014_3_G102978             IBD 1.7020262 0.7173127 0.28268733
#> 21   SKST014_2_G102977             IBD 1.7731820 0.6793101 0.32068985
#> 22   SKST014_5_G102980             IBD 2.5844154 0.8805813 0.11941867
#> 23   SKST014_4_G102979             IBD 2.0348834 0.8489550 0.15104501
#> 24   SKST014_7_G102940             IBD 2.4629366 0.8500670 0.14993296
#> 25   SKST014_6_G102951             IBD 2.4753211 0.8777362 0.12226384
#> 26   SKST023_2_G102987         control 1.8026734 0.7462762 0.25372381
#> 27   SKST023_1_G102981         control 2.0283657 0.7817109 0.21828911
#> 28   SKST023_4_G102937         control 2.8169363 0.9088080 0.09119199
#> 29   SKST023_3_G102941         control 2.7045113 0.8781404 0.12185965
#> 30   SKST011_3_G102973         control 2.8240780 0.9060256 0.09397439
#> 31   SKST011_2_G102972         control 2.7897078 0.9035860 0.09641398
#> 32   SKST011_1_G102971         control 2.8115253 0.9067192 0.09328076
#> 33   SKST010_8_G102998         control 2.6287806 0.8766470 0.12335298
#> 34   SKST010_7_G103003         control 2.7272082 0.8715787 0.12842131
#> 35   SKST010_6_G103004         control 2.3480810 0.8592615 0.14073846
#> 36   SKST010_5_G102990         control 2.7711954 0.8921098 0.10789023
#> 37   SKST010_4_G102956         control 2.8546624 0.8897277 0.11027231
#> 38   SKST011_5_G102993         control 2.7648181 0.8958376 0.10416241
#> 39   SKST011_4_G102952         control 2.7560408 0.8990970 0.10090304
#> 40   SKST023_7_G103015         control 3.0339162 0.9251253 0.07487473
#> 41   SKST024_1_G102982         control 2.2056892 0.8348499 0.16515009
```

```
#> 42   SKST023_5_G103007         control 1.6690423 0.7235394 0.27646059
#> 43   SKST023_6_G103009         control 2.8821410 0.9141447 0.08585531
#> 44   SKST024_4_G102938         control 2.8904509 0.8926771 0.10732292
#> 45   SKST024_5_G103001         control 1.9978854 0.8059146 0.19408543
#> 46   SKST024_2_G102953         control 2.1638165 0.7435241 0.25647593
#> 47   SKST024_3_G102944         control 2.8871684 0.9210467 0.07895329
#> 48   SKST024_6_G103012         control 2.6716317 0.8487322 0.15126780
#> 49   SKST024_7_G103022         control 2.3202246 0.8072082 0.19279180
#> 50   SKST011_6_G103002         control 2.8433942 0.9045948 0.09540524
#> 51   SKST011_7_G103017         control 2.6457497 0.8868671 0.11313289
#> 52   SKST011_8_G103000         control 2.5949466 0.8581854 0.14181457
#> 53   SKST012_1_G102974         control 3.2063818 0.9353146 0.06468544
#> 54   SKST012_2_G102975         control 3.2757311 0.9332595 0.06674052
#> 55   SKST012_3_G102950         control 2.9107180 0.9018265 0.09817355
#> 56   SKST012_4_G102954         control 3.2504997 0.9313359 0.06866413
#> 57   SKST012_5_G102992         control 2.7658572 0.8999757 0.10002425
#> 58   SKST012_6_G102997         control 3.0290341 0.9162627 0.08373725
#> 59   SKST014_1_G102976             IBD 0.6842360 0.4911153 0.50888466
#> 60   SKST036_3_G103025         control 2.7133186 0.8846836 0.11531644
#> 61   SKST036_2_G103020         control 2.4866486 0.8723806 0.12761943
#> 62   SKST037_2_G103023         control 2.1749561 0.7924307 0.20756927
#> 63   SKST032_4_G103008             IBD 2.3228094 0.8444105 0.15558954
#> 64   SKST032_3_G102989             IBD 2.7601873 0.9161065 0.08389348
#> 65   SKST036_1_G103013         control 2.5246610 0.8584318 0.14156819
#> 66   SKST032_5_G103024             IBD 2.3957859 0.8529960 0.14700398
#> 67   SKST041_1_G103005         control 2.8809232 0.9108480 0.08915205
#> 68   SKST037_3_G103026         control 1.2527133 0.6787014 0.32129865
#> 69   SKST041_2_G103027         control 2.0987718 0.8033809 0.19661908
#> 70   SKST041_3_G103028         control 2.2962906 0.8474506 0.15254945
#> 71   SKST010_2_G102969         control 2.5482905 0.8826525 0.11734753
#> 72   SKST010_3_G102970         control 3.3414326 0.9396790 0.06032103
#> 73   SKST007_8_G102999         control 2.6317371 0.8958339 0.10416609
#> 74   SKST010_1_G102968         control 3.4896605 0.9544864 0.04551359
#> 75   SKST007_6_G102995         control 2.6634173 0.8338101 0.16618994
#> 76   SKST007_7_G103016         control 2.8152047 0.9110826 0.08891744
#> 77   SKST007_3_G102949         control 2.9240942 0.9226028 0.07739722
#> 78   SKST007_4_G102948         control 2.6810693 0.8931690 0.10683105
#> 79   SKST007_1_G102966         control 2.7810739 0.9093627 0.09063725
#> 80   SKST007_2_G102967         control 2.5204322 0.8741401 0.12585993
#> 81  SKST027_10_G102936             IBD 1.8598491 0.8024193 0.19758072
#> 82   SKST025_9_G103021             IBD 1.5556967 0.6776121 0.32238788
#> 83   SKST025_2_G102986             IBD 2.6696480 0.8958129 0.10418713
#> 84   SKST025_1_G102983             IBD 2.7781552 0.8919120 0.10808803
#> 85   SKST024_8_G103031         control 2.3069551 0.8079908 0.19200921
#> 86   SKST025_6_G103010             IBD 1.5780891 0.7437443 0.25625573
#> 87   SKST025_5_G103006             IBD 2.0191108 0.7401280 0.25987200
#> 88   SKST025_4_G102939             IBD 1.8767355 0.7931885 0.20681147
#> 89   SKST025_3_G102942             IBD 2.4100870 0.8393832 0.16061683
#> 90           p8883_mo6             IBD 2.7980886 0.9148663 0.08513370
#> 91           p9223_mo4         control 2.1881319 0.6885799 0.31142015
#> 92           p9223_mo3         control 2.7076171 0.8123768 0.18762317
#> 93           p9223_mo2         control 2.7358612 0.8100018 0.18999815
```

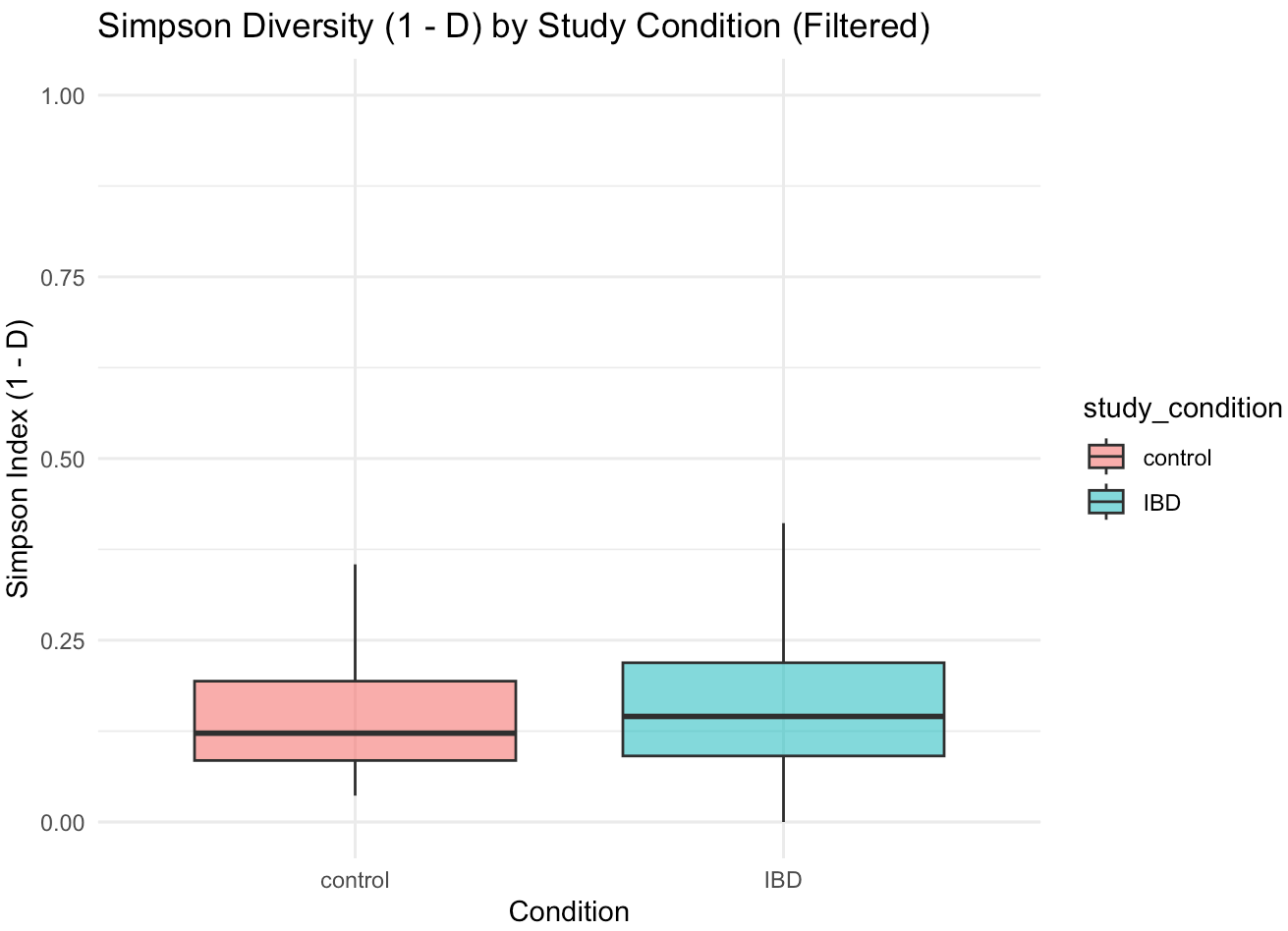
```
#> 94          p9223_mo1         control 2.0431248 0.6344728 0.36552717
#> 95          p9220_mo7         control 2.9987451 0.9184255 0.08157447
#> 96          p9220_mo6         control 2.7864065 0.8938321 0.10616795
#> 97          p9220_mo5         control 3.2924001 0.9469014 0.05309862
#> 98          p9220_mo4         control 3.1979251 0.9360076 0.06399240
#> 99          p9220_mo3         control 3.2498214 0.9433278 0.05667221
#> 100         p9220_mo2         control 3.1767250 0.9310763 0.06892374
#> 101         p8646_mo8             IBD 3.2712163 0.9467655 0.05323450
#> 102         p8646_mo9             IBD 3.1069294 0.9333765 0.06662346
#> 103         p8646_mo6             IBD 2.8863611 0.8884233 0.11157671
#> 104         p8646_mo7             IBD 2.9868318 0.9208884 0.07911163
#> 105        p8649_mo11             IBD 2.1949472 0.7365005 0.26349953
#> 106        p8649_mo12             IBD 2.7410957 0.8342998 0.16570021
#> 107         p8649_mo1             IBD 3.3293681 0.9484852 0.05151476
#> 108        p8649_mo10             IBD 3.2217033 0.9339763 0.06602372
#> 109         p8649_mo2             IBD 3.1776585 0.9329343 0.06706573
#> 110         p8649_mo3             IBD 2.9045244 0.8851852 0.11481484
#> 111         p8582_mo5             IBD 3.2352955 0.9424675 0.05753252
#> 112         p8582_mo7             IBD 3.2676648 0.9412434 0.05875662
#> 113         p8582_mo6             IBD 3.2571216 0.9458514 0.05414862
#> 114         p9281_mo5         control 3.1516610 0.9288301 0.07116990
#> 115         p9281_mo6         control 1.9488341 0.7903650 0.20963496
#> 116         p9223_mo7         control 2.7323455 0.8154577 0.18454233
#> 117         p9223_mo8         control 2.8639164 0.8480329 0.15196706
#> 118         p9223_mo5         control 3.1969248 0.9219603 0.07803965
#> 119         p9223_mo6         control 2.7585885 0.8478928 0.15210718
#> 120         p9281_mo3         control 3.2784516 0.9429546 0.05704545
#> 121         p9281_mo4         control 3.0690481 0.9239806 0.07601941
#> 122         p9281_mo1         control 3.3366552 0.9415510 0.05844896
#> 123         p9281_mo2         control 3.1349278 0.9249857 0.07501427
#> 124         p8646_mo3             IBD 2.8212167 0.8770671 0.12293292
#> 125         p8646_mo2             IBD 2.6068442 0.8508834 0.14911661
#> 126        p8646_mo12             IBD 3.1970799 0.9379695 0.06203050
#> 127        p8646_mo11             IBD 3.1906148 0.9385366 0.06146342
#> 128        p8646_mo10             IBD 3.1904190 0.9328649 0.06713509
#> 129         p8646_mo1             IBD 2.4563344 0.8190123 0.18098771
#> 130         p8600_mo9             IBD 2.7306945 0.8812102 0.11878980
#> 131         p8600_mo8             IBD 2.5054908 0.8435909 0.15640911
#> 132         p8646_mo5             IBD 3.3252302 0.9431623 0.05683770
#> 133         p8646_mo4             IBD 3.2608190 0.9272229 0.07277711
#> 134         p8808_mo7             IBD 2.6817387 0.8918650 0.10813503
#> 135         p8808_mo6             IBD 2.8267354 0.9175881 0.08241192
#> 136         p8808_mo9             IBD 2.6598647 0.8968506 0.10314940
#> 137         p8808_mo8             IBD 2.7542698 0.9090572 0.09094278
#> 138        p8808_mo12             IBD 2.1685183 0.8268748 0.17312520
#> 139        p8808_mo11             IBD 2.0175372 0.7927332 0.20726684
#> 140         p8883_mo2             IBD 2.9641448 0.9164362 0.08356380
#> 141         p8808_mo4             IBD 2.4652800 0.8516828 0.14831722
#> 142        p8816_mo10             IBD 2.8854457 0.9153551 0.08464486
#> 143         p8816_mo1             IBD 3.2642581 0.9372518 0.06274820
#> 144         p8883_mo3             IBD 2.8320962 0.8885733 0.11142674
#> 145         p8600_mo1             IBD 2.3235612 0.8419463 0.15805366
```

```
#> 146         p8600_mo10          IBD 2.5144408 0.8568160 0.14318402
#> 147         p8600_mo11          IBD 2.4295687 0.8235414 0.17645859
#> 148         p8600_mo12          IBD 2.5358425 0.8374550 0.16254496
#> 149          p8600_mo2          IBD 2.7809702 0.9032680 0.09673202
#> 150          p8600_mo3          IBD 2.8533362 0.9127022 0.08729784
#> 151          p8600_mo4          IBD 2.5195770 0.8483133 0.15168668
#> 152          p8600_mo5          IBD 2.9446139 0.9161902 0.08380984
#> 153          p8600_mo6          IBD 3.1102496 0.9345201 0.06547994
#> 154          p8600_mo7          IBD 2.3905778 0.8205733 0.17942666
#> 155          p8775_mo7          IBD 2.9502307 0.9082173 0.09178271
#> 156          p8775_mo8          IBD 3.2512044 0.9410491 0.05895087
#> 157          p8775_mo5          IBD 3.0296708 0.9141398 0.08586023
#> 158          p8775_mo6          IBD 3.2244045 0.9381442 0.06185582
#> 159          p8775_mo3          IBD 3.0645091 0.9189255 0.08107447
#> 160          p8775_mo4          IBD 3.0852510 0.9217384 0.07826159
#> 161         p8775_mo12          IBD 2.9004152 0.9077146 0.09228544
#> 162          p8775_mo2          IBD 2.7901970 0.8848275 0.11517247
#> 163          p8775_mo9          IBD 2.6037354 0.8563312 0.14366876
#> 164         p8808_mo10          IBD 2.7467400 0.9204208 0.07957923
#> 165          p8816_mo8          IBD 2.8811862 0.9125761 0.08742389
#> 166          p8816_mo9          IBD 2.9347897 0.9134674 0.08653263
#> 167          p8816_mo4          IBD 3.0869504 0.9222314 0.07776861
#> 168          p8816_mo5          IBD 2.9759955 0.9165706 0.08342943
#> 169          p8816_mo6          IBD 2.9136666 0.9110562 0.08894382
#> 170          p8816_mo7          IBD 2.9174809 0.9160573 0.08394267
#> 171         p8816_mo11          IBD 3.0211472 0.9228923 0.07710768
#> 172         p8816_mo12          IBD 2.9358613 0.9036371 0.09636293
#> 173          p8816_mo2          IBD 2.7700241 0.8799520 0.12004803
#> 174          p8816_mo3          IBD 3.2272053 0.9394574 0.06054260
#> 175          p8585_mo9          IBD 3.0697299 0.9066927 0.09330725
#> 176          p8585_mo8          IBD 2.5014031 0.7884440 0.21155603
#> 177          p8585_mo4          IBD 3.1355768 0.9298240 0.07017595
#> 178          p8585_mo3          IBD 2.7160162 0.9054014 0.09459857
#> 179          p8585_mo7          IBD 2.7432172 0.8466796 0.15332037
#> 180          p8585_mo6          IBD 2.8369090 0.8679917 0.13200829
#> 181          p8582_mo9          IBD 3.2366453 0.9410905 0.05890952
#> 182          p8582_mo8          IBD 3.3240788 0.9418381 0.05816191
#> 183          p8585_mo2          IBD 2.9459631 0.9122657 0.08773433
#> 184          p8585_mo1          IBD 2.6820190 0.9014596 0.09854038
#> 185         p8883_mo12          IBD 2.5724065 0.8754515 0.12454848
#> 186         p8883_mo11          IBD 2.8374424 0.9141775 0.08582252
#> 187          p8855_mo3          IBD 2.9524064 0.9244398 0.07556015
#> 188         p8855_mo12          IBD 2.6117483 0.8739644 0.12603559
#> 189         p8855_mo11          IBD 2.5363042 0.8779739 0.12202607
#> 190         p8855_mo10          IBD 2.0196478 0.7770400 0.22295998
#> 191         p8883_mo10          IBD 2.9630591 0.9199234 0.08007657
#> 192          p8855_mo9          IBD 2.4258523 0.8606390 0.13936098
#> 193          p8855_mo8          IBD 2.6717697 0.8905552 0.10944483
#> 194          p8855_mo4          IBD 3.0695902 0.9328922 0.06710784
#> 195          p8808_mo5          IBD 2.7017634 0.9061237 0.09387631
#> 196          p8883_mo9          IBD 2.6951496 0.8936411 0.10635891
#> 197          p9061_mo1          IBD 1.9915813 0.7012513 0.29874866
```

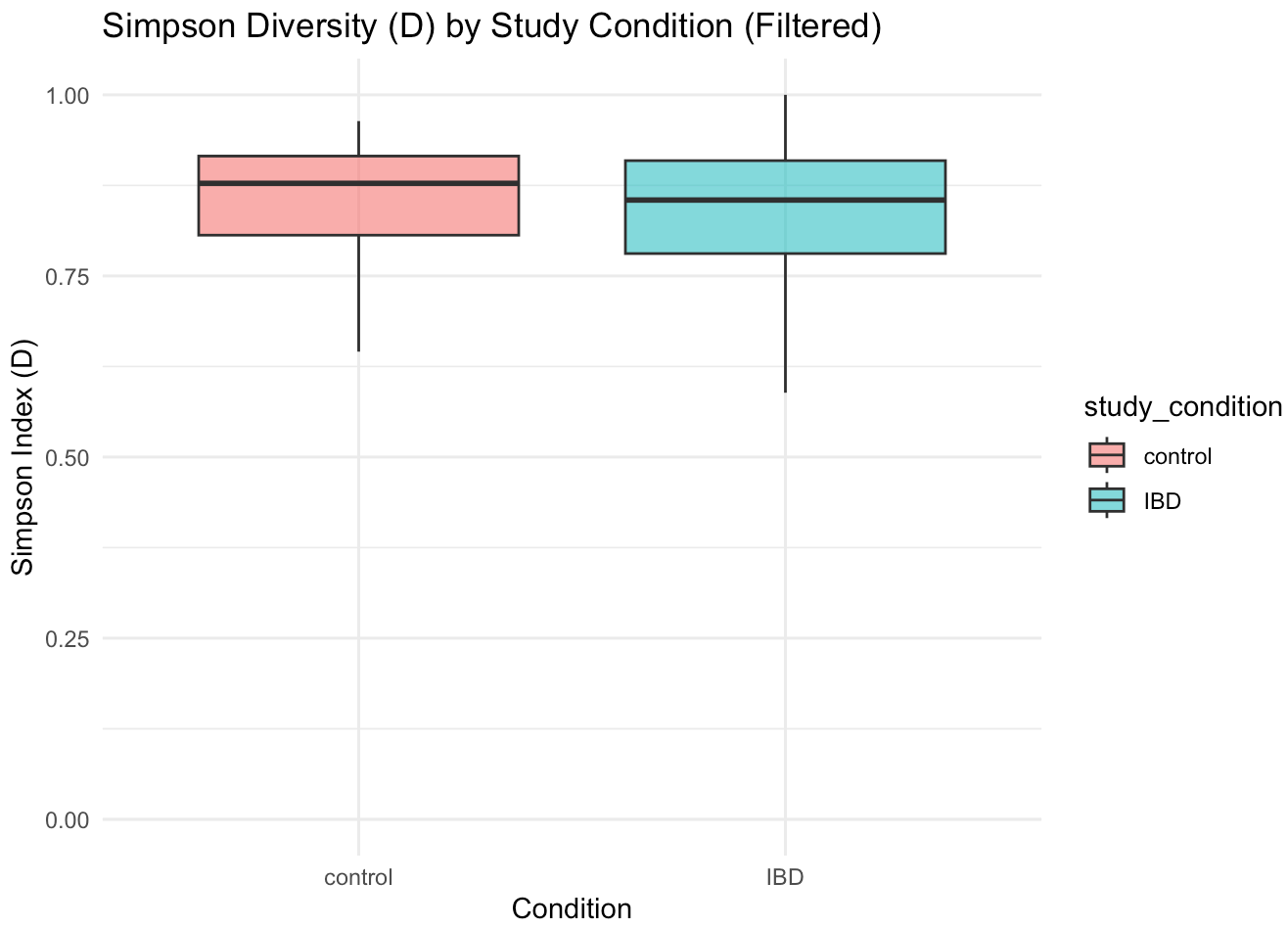
```
#> 198          p8883_mo7             IBD 2.8513694 0.9146848 0.08531518
#> 199          p8883_mo8             IBD 2.7596145 0.9069852 0.09301479
#> 200          p8883_mo4             IBD 2.8667172 0.9232376 0.07676240
#>  [ reached 'max' / getOption("max.print") -- omitted 2687 rows ]

# Boxplot Shannon by group
ggplot(div_df, aes(x = study_condition, y = Shannon, fill = study_condition)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Shannon Diversity by Study Condition (Filtered)",
       x = "Condition", y = "Shannon Index")
```




```
# Boxplot Simpson Diversity (1-D) by group
ggplot(div_df, aes(x = study_condition, y = InvSimpson, fill = study_condition)) +
  geom_boxplot(outlier.shape = NA, alpha = 0.6) +
  theme_minimal() +
  labs(title = "Simpson Diversity (1 - D) by Study Condition (Filtered)",
       x = "Condition", y = "Simpson Index (1 - D)")
```




```
# Boxplot Simpson Diversity (D) by group
ggplot(div_df, aes(x = study_condition, y = Simpson_D, fill = study_condition)) +
  geom_boxplot(outlier.shape = NA, alpha = 0.6) +
  theme_minimal() +
  labs(title = "Simpson Diversity (D) by Study Condition (Filtered)",
       x = "Condition", y = "Simpson Index (D)")
```



Shannon Index

- IBD Median Shannon Index < control Median Shannon Index $\implies$ less microbial diversity than control
- Low-Shannon outliers in IBD: could reflect patients whose gut communities have lost many taxa (very few species or highly uneven communities), often interpreted as dysbiosis

Simpson Diversity (1 - D)

- Visually shows us that we have a relatively low diversity (i.e., 1 - D is closer to 0).

Statistical test

We use the Wilcoxon-rank test.

H_0: The Shannon distributions in control and IBD come from the same population (i.e., no difference in median diversity).

H_a: The Shannon distributions in control and IBD differ (i.e., one group tends to have higher or lower Shannon).

```
div_df %>%
  wilcox_test(Shannon ~ study_condition) %>%
  add_significance("p")
#> # A tibble: 1 × 8
#>   .y.     group1  group2    n1    n2 statistic        p p.signif
#>   <chr>   <chr>   <chr>  <int> <int>     <dbl>    <dbl> <chr>
#> 1 Shannon control IBD      799  2088   959496. 3.97e-10 ****
```

```
div_df %>%
  # requires "coin" package
  wilcox_effsize(  # rank-biserial correlation
    Shannon ~ study_condition,
    alternative = "two.sided",
    conf.level  = 0.95,
    ci          = TRUE
  )
#> # A tibble: 1 × 9
#>   .y.     group1  group2 effsize    n1    n2 conf.low conf.high magnitude
#> * <chr>   <chr>   <chr>    <dbl> <int> <int>    <dbl>     <dbl> <ord>
#> 1 Shannon control IBD      0.116   799  2088     0.08      0.15 small
```

Given our extremely small p-value, i.e., $p \ll 0.05$, we reject the null hypothesis. There is statistically significant evidence that the Shannon index in controls is not drawn from the same distribution as in IBD.

This confirms that IBD patients in our cohort exhibit significantly reduced gut-microbial α-diversity (Shannon index) compared to controls.

Rank-biserial correlation

- can be interpreted roughly like a Pearson r.
    - |r| ≈ 0.1 (small), ≈ 0.3 (medium), ≈ 0.5 (large)

effsize = 0.116 $\implies$ small rank-biserial correlation, indicating that control samples tend to have higher Shannon than IBD samples.

CLR-Transform

Microbiome count tables are compositional — each sample's taxa counts are only meaningful *relative* to one another and sum to an arbitrary total. The **centered log-ratio (CLR) transform** maps those compositions into real space so that standard multivariate methods (e.g. distance measures, classifiers) behave properly.

In the context of the Random Forest model, the CLR-transform removes sequencing-depth effects, stablizes variance, centers the features (so that each is on a comparable scale across samples), and respects compositional structure (ensuring measured distances between feature vectors reflect relative changes in taxa, not absolute counts).

```
# clr-transform
phy_clr <- microbiome::transform(phy_filtered, "clr")
message("After CLR-transform: ntaxa = ", ntaxa(phy_clr), ", nsamples = ", nsamples(phy_clr
#> After CLR-transform: ntaxa = 388, nsamples = 2887
```

Train

Define Random Forest Model

```
train_rf <- function(phy_clr,
                     n_folds    = 5,
                     n_repeats  = 3,
                     seed       = 43,    # for reproducibility
                     trees      = 500) { # number of trees

  set.seed(seed)

  # extract feature matrix (samples by taxa) and metadata
  otu_mat  <- t(as(otu_table(phy_clr), "matrix"))
  meta_df  <- as(sample_data(phy_clr), "data.frame") %>%
    as_tibble(rownames = "SampleID") %>%
    mutate(
      Outcome   = factor(study_condition, levels = c("control","IBD")),
      SubjectID = as.character(subject_id)  # ensure character
    )

  # sanitize names so randomForest() doesn't yell
  colnames(otu_mat) <- make.names(colnames(otu_mat))

  data_all <- as_tibble(otu_mat, rownames = "SampleID") %>%
    dplyr::left_join(meta_df, by = "SampleID")

  message("Full data: ", nrow(data_all), " samples × ", ncol(otu_mat), " taxa; ",
          length(unique(data_all$SubjectID)), " subjects.")

  # grab feature cols
  feature_cols <- colnames(otu_mat)

  # pre-allocate lists for storing ROC objects & probabilities
  roc_list   <- list()   # pROC::roc objects
  probs_list <- list()   # numeric vectors of predicted probs
  labels_list <- list()  # test-set labels
  idx        <- 1        # running index


  # function to do one round of grouped k-fold CV
  one_repeat <- function(rep_id) {
    # shuffle and split unique subjects into folds
    subs      <- unique(data_all$SubjectID)
    subs      <- sample(subs)
    fold_ids  <- cut(seq_along(subs), breaks = n_folds, labels = FALSE)
    subject_folds <- split(subs, fold_ids)

    # for each fold, train, test, compute AUC
    purrr::map_dfr(seq_along(subject_folds), function(k) {
```

```
    test_subj <- subject_folds[[k]]

    train_df <- filter(data_all, !SubjectID %in% test_subj) %>%
                  dplyr::select(all_of(feature_cols), Outcome)
    test_df  <- filter(data_all,  SubjectID %in% test_subj) %>%
                  dplyr::select(all_of(feature_cols), Outcome)

    rf_mod <- randomForest(
      Outcome ~ .,
      data    = train_df,
      ntree   = trees
    )

    # predict probabilities for positive class "IBD"
    probs   <- predict(rf_mod, test_df, type = "prob")[, "IBD"]
    roc_obj <- roc(test_df$Outcome, probs, quiet = TRUE)

    # store for later
    probs_list[[idx]]    <<- probs
    roc_list[[idx]]      <<- roc_obj
    labels_list[[idx]]   <<- test_df$Outcome
    idx                  <<- idx + 1

    tibble(
      repeats    = rep_id,
      fold       = k,
      n_train    = nrow(train_df),
      n_test     = nrow(test_df),
      auc        = as.numeric(auc(roc_obj))
    )
  })
}

# run repeats in parallel
cv_results <- future_map_dfr(seq_len(n_repeats), one_repeat, .options = furrr_options(se

# summarize
summary_df <- cv_results %>%
  group_by(repeats) %>%
  summarize(
    mean_auc = mean(auc),
    sd_auc   = sd(auc),
    .groups  = "drop"
  ) %>%
  ungroup() %>%
  mutate(overall_mean = mean(mean_auc),
         overall_sd   = sd(mean_auc))

list(
  cv_folds    = cv_results,   # df of AUCs per fold
  cv_summary  = summary_df,   # summary of AUCs
  probs_list  = probs_list,   # list of numeric vectors of probabilities
```

```
    roc_list    = roc_list,     # list of pROC::roc objects
    labels_list = labels_list   # list of test-set labels
  )
}
```

Train + Test model

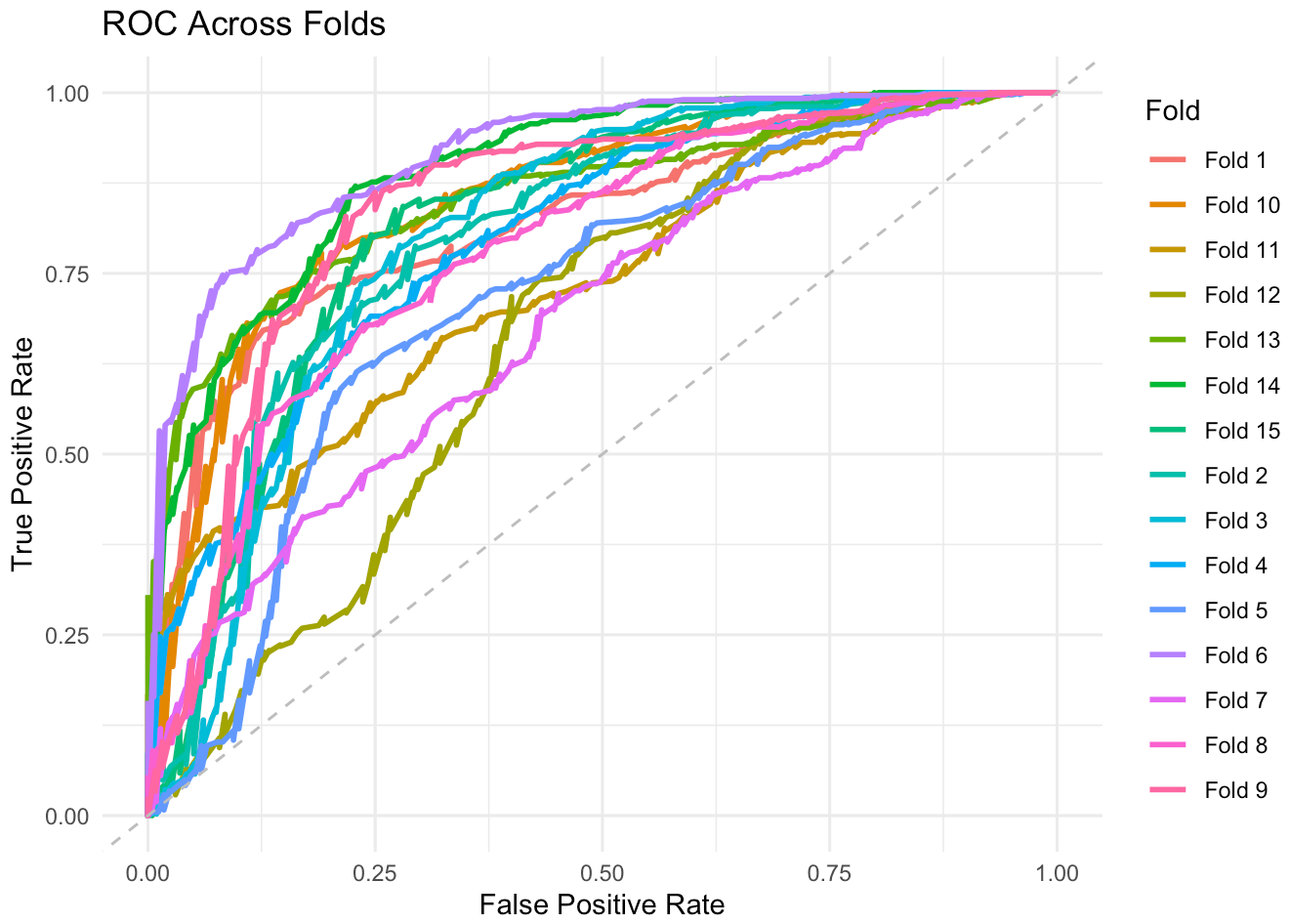
```
# train the model
rf_workflow_fit <- train_rf(phy_clr)
#> Full data: 2887 samples × 388 taxa; 986 subjects.
```

Results

Prediction of IBD vs control

Plot ROC across folds

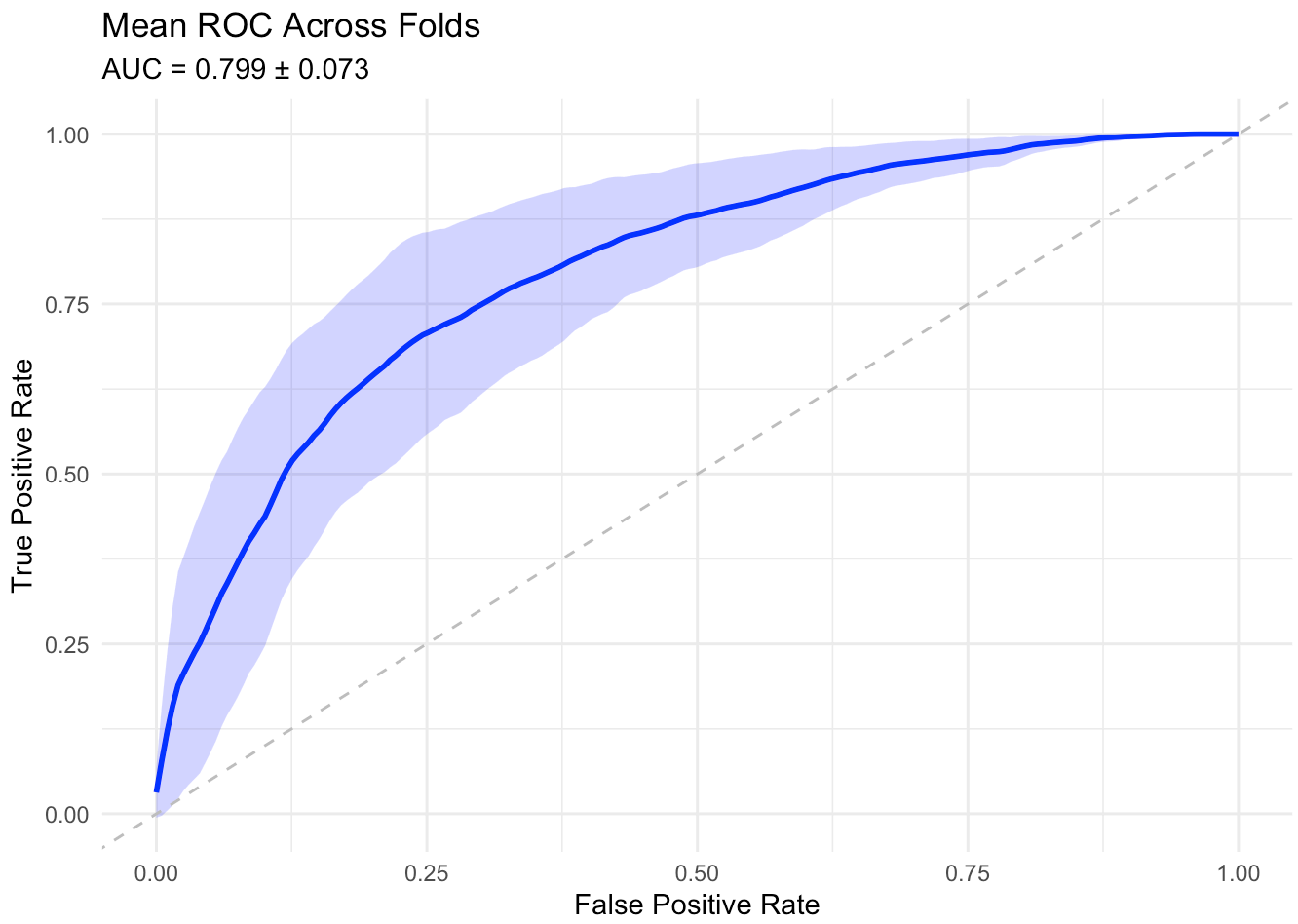
```
# create ROC df
roc_df <- purrr::imap_dfr(rf_workflow_fit$roc_list, function(roc_obj, i) {
  data.frame(
    Fold       = paste0("Fold ", i),
    FPR        = 1 - roc_obj$specificities,
    TPR        = roc_obj$sensitivities
  )
})

# plot ROC
ggplot(roc_df, aes(x = FPR, y = TPR, color = Fold)) +
  geom_line(linewidth = 1) +
  geom_abline(linetype = "dashed", color = "gray") +
  labs(
    title = "ROC Across Folds",
    x     = "False Positive Rate",
    y     = "True Positive Rate"
  ) +
  theme_minimal()
```



We see the each fold's ROC trade-off between TPR (sensitivity) and FPR (1-specificity). Looking across all folds helps us gauge the stability of our classifier. Given the wide spread signals, our classifier might depend on the training/test split. Let's investigate.

Plot Average ROC with Confidence Bands

```r
# define grid
fpr_grid <- seq(0, 1, length.out = 200)

# create per-fold interpolated TPRs at correct grid
roc_df <- map_dfr(rf_workflow_fit$roc_list, ~{
  roc_obj <- .
  # interpolate TPR at desired FPRs via approx():
  fpr     <- 1 - roc_obj$specificities
  tpr     <- roc_obj$sensitivities
  interp  <- approx(x = fpr, y = tpr, xout = fpr_grid, ties = mean)
  tibble(FPR = fpr_grid, TPR = interp$y)
}, .id = "Fold")

# summarize mean and sd
summary_roc <- roc_df %>%
  group_by(FPR) %>%
  summarize(
    mean_TPR = mean(TPR, na.rm = TRUE),
    sd_TPR   = sd(TPR, na.rm = TRUE),
    .groups  = "drop"
  )

# plot mean ROC
ggplot(summary_roc, aes(x = FPR, y = mean_TPR)) +
  geom_ribbon(aes(ymin = mean_TPR - sd_TPR,
                  ymax = mean_TPR + sd_TPR),
              fill  = "blue", alpha = 0.2) +
  geom_line(color = "blue", size = 1) +
  geom_abline(linetype = "dashed", color = "gray") +
  labs(
    title    = "Mean ROC Across Folds",
    subtitle = sprintf("AUC = %.3f ± %.3f",
                       mean(sapply(rf_workflow_fit$roc_list, auc)),
                       sd  (sapply(rf_workflow_fit$roc_list, auc))),
    x        = "False Positive Rate",
    y        = "True Positive Rate"
  ) +
  theme_minimal()
```



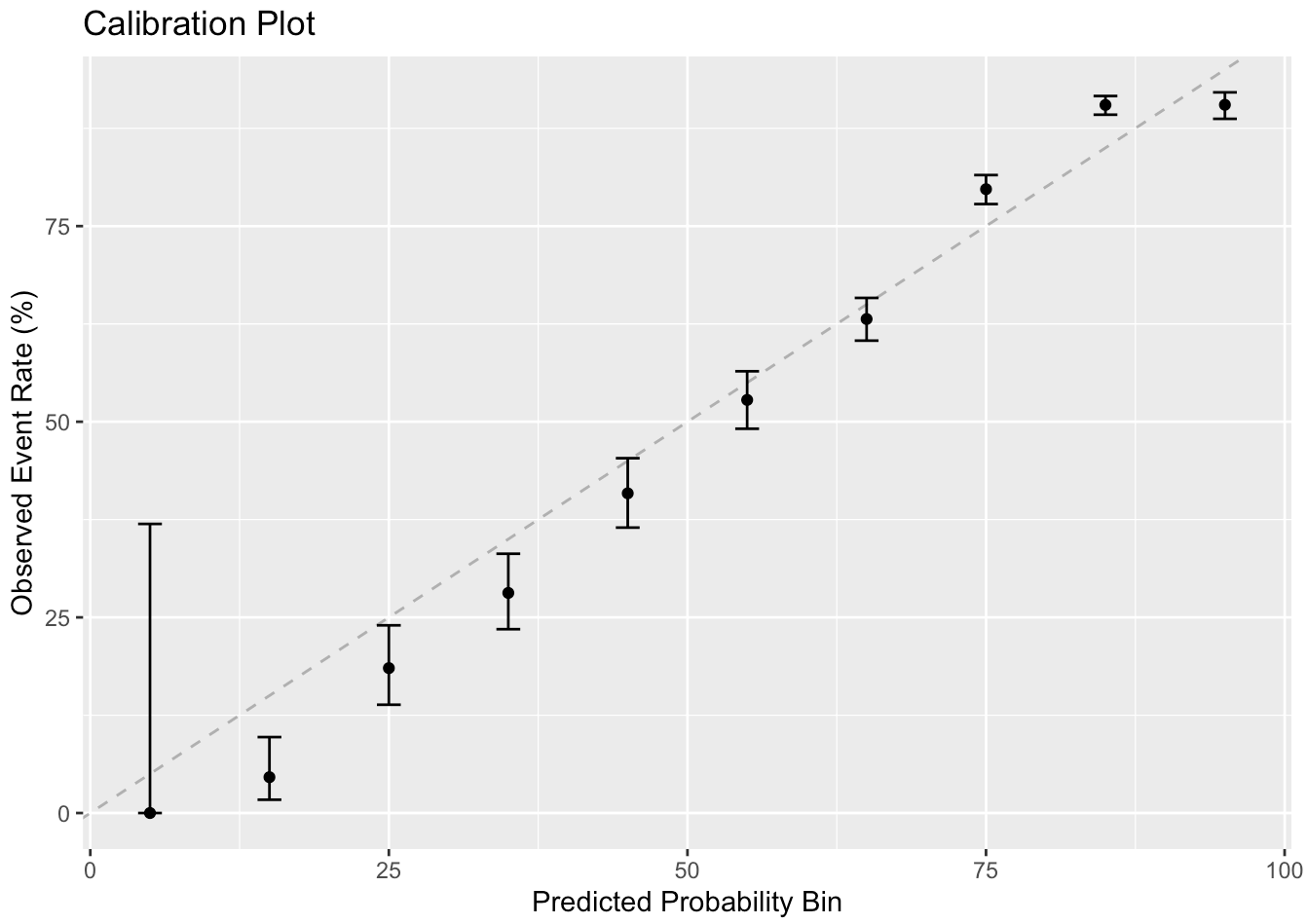
Quantify AUC Variability

```
# per-fold AUC vector
aucs <- rf_workflow_fit$cv_folds$auc

# basic descriptors
mean_auc <- mean(aucs); sd_auc <- sd(aucs)
cat(sprintf("Mean AUC = %.3f ± %.3f\n", mean_auc, sd_auc))
#> Mean AUC = 0.799 ± 0.073

# 95% CI via bootstrapping across folds
boot_ci <- boot::boot(aucs, statistic = function(u, i) mean(u[i]), R = 2000)
ci_vals <- boot::boot.ci(boot_ci, type = "perc")$percent[4:5]
cat(sprintf("Bootstrap 95%% CI: [%.3f, %.3f]\n", ci_vals[1], ci_vals[2]))
#> Bootstrap 95% CI: [0.763, 0.833]
```

Calibration Curves

Calibration shows how predicted probabilities map to observed frequencies.

```
# create calibration df
calib_df <- purrr::imap_dfr(
  rf_workflow_fit$probs_list,
  ~ tibble(
      Prob = .x,
      Obs  = rf_workflow_fit$labels_list[[.y]]
    )
)

# caret::calibration object
calib_obj <- caret::calibration(
  Obs ~ Prob,
  data   = calib_df,
  class  = "IBD",
  cuts   = 10  # number of bins
)

# plot calibration
ggplot(calib_obj, bwidth = 2, dwidth = 3) +
  labs(
    title = "Calibration Plot",
    x     = "Predicted Probability Bin",
    y     = "Observed Event Rate (%)"
  )
```



The diagonal reference line represents perfect calibration: predicted probability = observed frequency. Points above the line means the model's predictions in that bin are too low (it underestimates risk), while points below the line means predictions are too high (it overestimates risk).

The width of the each error bar indicates how reliable the calibration is in that bin. Note that the wide error bars in the first bin is between 0-37%, which means that there may be very few IBD cases in that range, so we should be cautious in our interpretation.

However, looking at our bins, we actually see that there are *no* predictions in the first bin, which explains the very wide confidence bound since confidence intervals were forced on zero counts.

```
calib_obj$data
#>    calibModelVar       bin   Percent     Lower     Upper Count midpoint
#> 1           Prob   [0,0.1]  0.000000  0.000000 36.941665     0        5
#> 2           Prob (0.1,0.2]  4.580153  1.699134  9.702458     6       15
#> 3           Prob (0.2,0.3] 18.518519 13.839956 23.983100    45       25
#> 4           Prob (0.3,0.4] 28.125000 23.487475 33.134896    99       35
#> 5           Prob (0.4,0.5] 40.853659 36.474839 45.343059   201       45
#> 6           Prob (0.5,0.6] 52.789116 49.106960 56.448805   388       55
#> 7           Prob (0.6,0.7] 63.128039 60.366941 65.826407   779       65
#> 8           Prob (0.7,0.8] 79.720280 77.819356 81.526941  1482       75
#> 9           Prob (0.8,0.9] 90.484140 89.238333 91.629918  2168       85
#> 10          Prob   (0.9,1] 90.503716 88.711493 92.096239  1096       95
```

Let's look at bins by risk.

- **Low-Risk Bins** (left side, 10-30% predicted):
    - Beginning at bin 2, where we actually observed outcomes in that probability range, our model is slightly over-confident: it gives 15% when the true rate is ~2%.
- **Mid-Risk Bins** (around 25–55% predicted):
    - The observed rates lie almost exactly on the diagonal, with narrow confidence intervals. Our model is **well calibrated** in the mid-probability range, leading to confidence in its risk estimates around "coin-flip" probabilities.
- **High-Risk Bins** (right side, 65-95% predicted):
    - There is a slight underestimation of risk around the 75–95% region (points just below the diagonal), meaning the model is a bit **under-confident**: it gives 95% when the true rate is ~88 %, and 75% when the true rate is ~78 %.
    - However, these deviations are modest and well within the narrow confidence bands, indicating acceptable performance even at extreme probabilities.

Let's further assess our model by checking how accurate our predictions are, on average. We will use the Brier score, which quantifies overall calibration, with lower values indicating better calibration.

Brier Score

```
calib_df <- calib_df %>%
  mutate(
    OutcomeNum = as.integer(Obs == "IBD")  # 1 when Obs == "IBD", 0 otherwise
  )

# manual calculation
brier_score <- calib_df %>%
  summarise(Brier = mean((Prob - OutcomeNum)^2)) %>%
  pull(Brier)

brier_score
#> [1] 0.1532861
```

Our Brier score is closer to 0, which implies overall good calibration and discrimination. Let's now compare how much better this is than baseline, i.e., calculate Brier Skill Score (BSS).

```
# calculate IBD prevalence
prevalence <- mean(calib_df$OutcomeNum)
cat("IBD prevalence:", round(prevalence * 100, 1), "%\n")
#> IBD prevalence: 72.3 %

# calculate BS_{ref}
BS_ref <- prevalence * (1 - prevalence)
cat("BS_ref:", BS_ref, "\n")
#> BS_ref: 0.200163

# calculate BSS
BSS = 1 - (brier_score)/BS_ref
cat("BSS:", BSS)
#> BSS: 0.2341937
```

BS_ref can be viewed as a "no-skill" baseline, in which this naive model always predicts the overall IBD prevalence 20%. Our result, 15.3%, is less than BS_ref, with BSS indicating a 23% improvement over baseline.

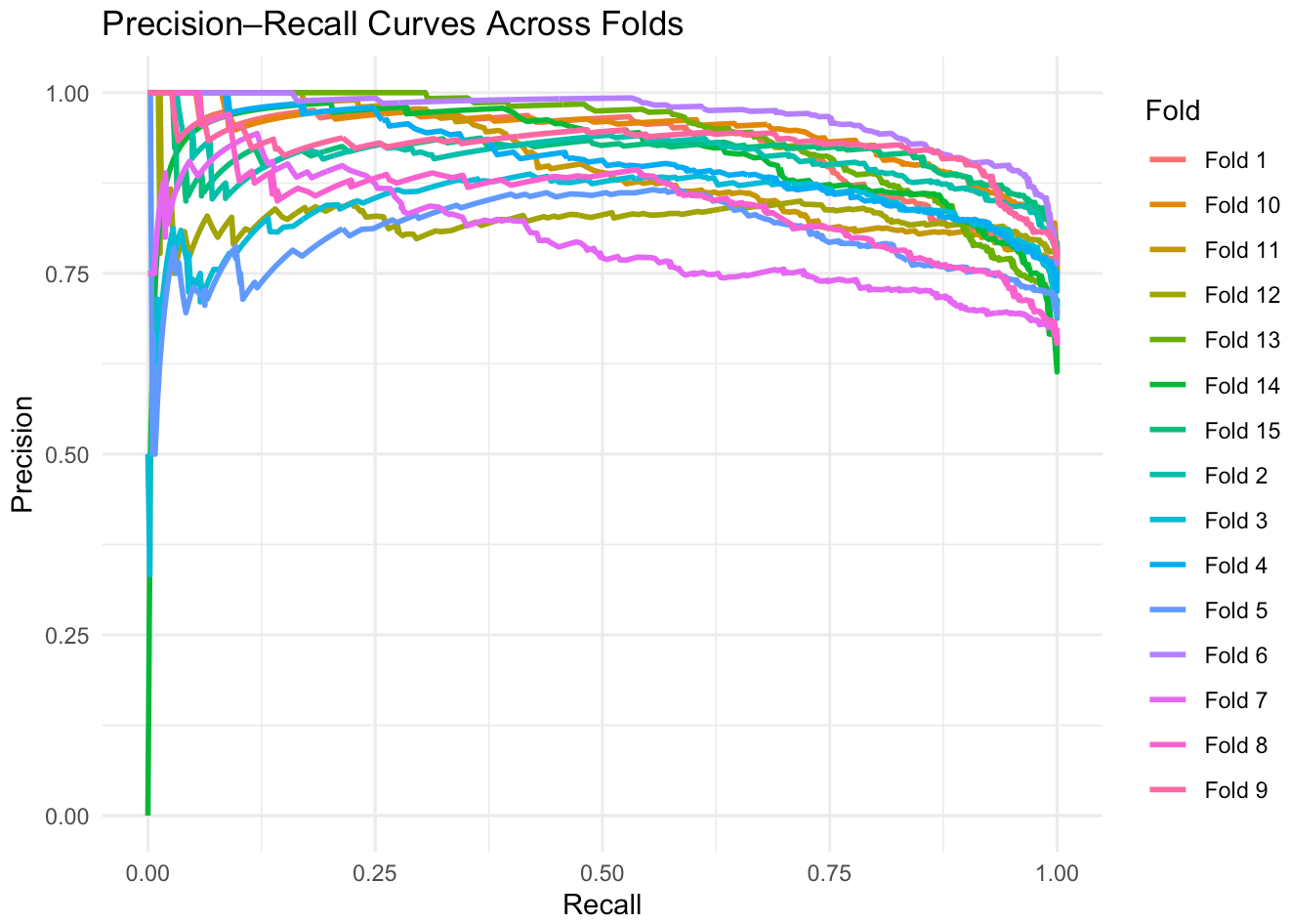
Precision-Recall Curves

Let's first quantify class imbalance.

```
# flatten labels_list into one factor vector
all_labels <- unlist(rf_workflow_fit$labels_list, use.names = FALSE)

tab   <- table(all_labels)
pct   <- prop.table(tab) * 100

tab
#> all_labels
#> control     IBD
#>    2397    6264

pct
#> all_labels
#>  control      IBD
#> 27.67579 72.32421
```

So we see that IBD dominates control (72% vs 28%).

```
# create PRROC objects per fold
pr_list <- imap(rf_workflow_fit$probs_list, function(probs, i) {
  labels <- rf_workflow_fit$labels_list[[i]]
  fg     <- probs[labels == "IBD"]       # scores for the positive class
  bg     <- probs[labels == "control"]   # scores for the negative class

  pr.curve(
    scores.class0 = fg,
    scores.class1 = bg,
    curve         = TRUE
  )
})

# create PR df
pr_df <- purrr::imap_dfr(pr_list, function(pr, i) {
  as.data.frame(pr$curve) %>%
    setNames(c("Recall", "Precision", "Threshold")) %>%
    mutate(Fold = paste0("Fold ", i))
})

# plot PR
ggplot(pr_df, aes(x = Recall, y = Precision, color = Fold)) +
  geom_line(size = 1) +
  labs(
    title = "Precision–Recall Curves Across Folds",
    x     = "Recall",
    y     = "Precision"
  ) +
  theme_minimal()
```



The clustering of curves near the top-left indicates generally high precision even at moderate recall, while the spread between them reflects fold-to-fold variability in performance.

Let's look at our case broken out into three components: the high-precision, high-recall region, trade-off as recall increases, and fold-to-fold variability.

- High-precision, high-recall region
    - All folds achieve perfect precision at very low recall (selecting only the highest-confidence predictions). Where the curves run nearly flat, it means that we can increase recall (i.e., catch more true IBD cases) without sacrificing precision (few false positives).
- Trade-off as recall increases
    - Precision slowly declines as we increase recall, meaning that, if we want to capture more true positives, we have to admit more false positives. The slope of each fold's curve quantifies how rapidly precision deteriorates; flatter slopes are better (i.e., precision holds up longer).
- Fold-to-fold variability
    - The spread between curves reveals how stable the model's PR performance is across different train/test splits.

So how good is our model at capturing true positives, i.e., IBD prevalence?

AUPRC

Recall that IBD prevalence is 72%, which means that a random classifier has AUPRC = 0.72. Area Under the Precision–Recall Curve (AUPRC) summarizes a model's ability to retrieve true positives without overwhelming false positives by integrating precision over all recall levels. AUPRC's baseline equals the positive-class prevalence, making it especially informative in imbalanced settings.

```
auprcs <- purrr::map_dbl(pr_list, ~ .x$auc.integral)

# Summary across folds
mean_auprc <- mean(auprcs)
sd_auprc   <- sd(auprcs)

cat(sprintf("Mean AUPRC = %.3f ± %.3f\n", mean_auprc, sd_auprc))
#> Mean AUPRC = 0.892 ± 0.054
```

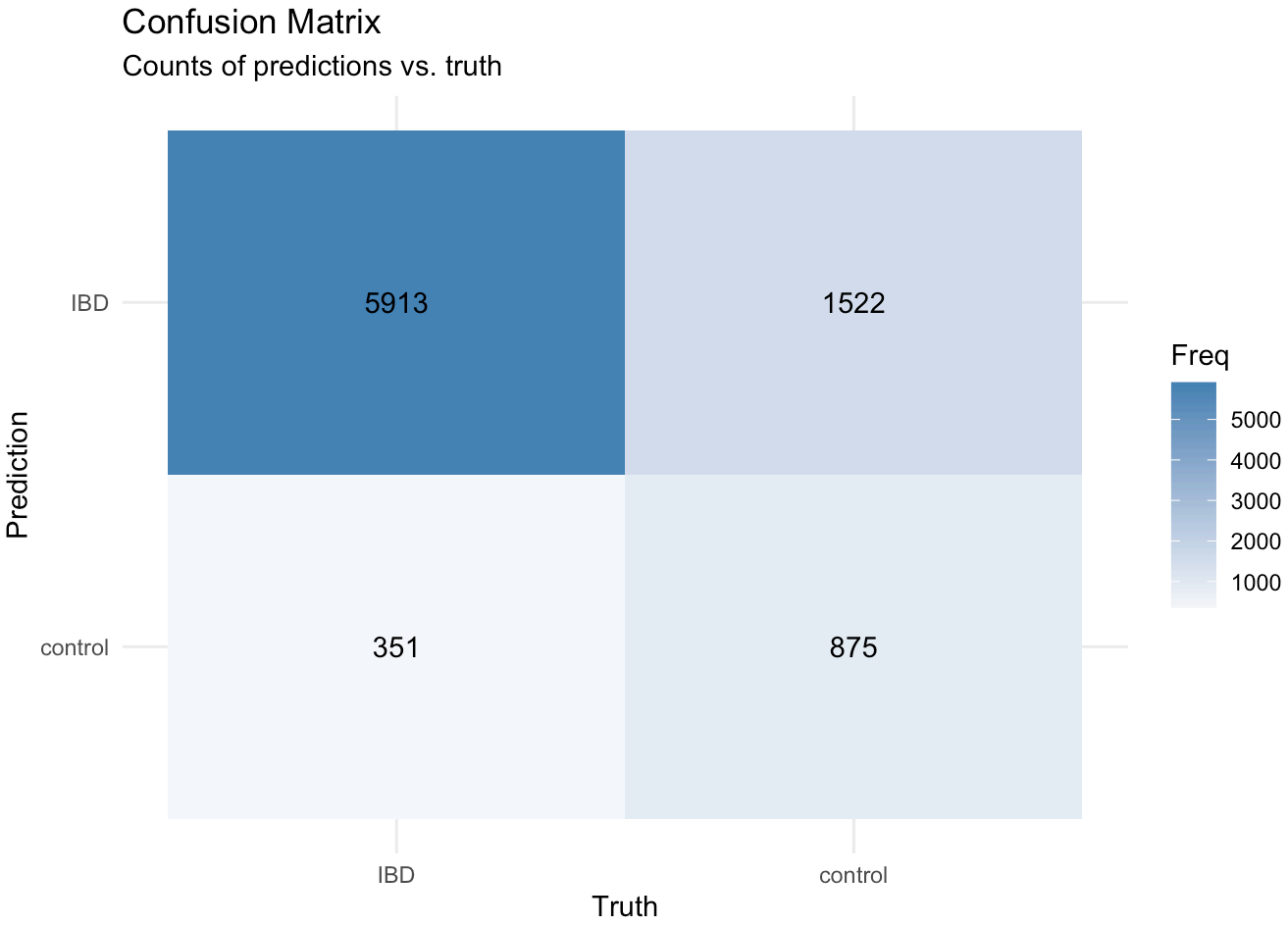
Since our mean AURPC = 0.892 ± 0.054 > 0.72, our model performs better than a random classifier.

Thus, a high AURPC alongside a low Brier score indicates both reliable probability estimates and strong discrimination.

Confusion Matrix

```
# get predictions and labels
results_df <- purrr::imap_dfr(
  rf_workflow_fit$probs_list,
  ~ tibble(
      Fold       = paste0("Fold ", .y),
      truth      = rf_workflow_fit$labels_list[[.y]],
      .pred_prob = .x
    )
) %>%
  # convert probabilities to class labels at 0.5 threshold
  mutate(.pred_class = factor(if_else(.pred_prob > 0.5, "IBD", "control"),
                              levels = levels(truth)))
# re-level so IBD comes first to make sure we calculate metrics for the correct "positive"
results_df <- results_df %>%
  mutate(
    truth = fct_relevel(truth, "IBD", "control"),
    .pred_class = fct_relevel(.pred_class, "IBD", "control")
  )

# compute the confusion matrix
conf_mat_res <- results_df %>%
  yardstick::conf_mat(truth = truth, estimate = .pred_class)

# calculate metrics for binary classification

# Sensitivity
sens_res <- sens_vec(results_df$truth, results_df$.pred_class)
# Specificity
spec_res <- spec_vec(results_df$truth, results_df$.pred_class)
# Precision
ppv_res  <- precision_vec(results_df$truth, results_df$.pred_class)
# F1-score
f1_res   <- f_meas_vec(results_df$truth, results_df$.pred_class)

data.frame(
  sens     = sens_res,
  spec     = spec_res,
  precision= ppv_res,
  F1        = f1_res
)
#>        sens      spec precision        F1
#> 1 0.9439655 0.3650396 0.7952925 0.8632747

# visualize the confusion matrix as heatmap
p <- autoplot(conf_mat_res, type = "heatmap")

# remove whatever scales (including 'fill') were already there to avoid warning
p$scales$scales <- list()

# add custom fill scale
```

```
p +
  scale_fill_gradient2(low = "white", high = "steelblue") +
  labs(title = "Confusion Matrix",
       subtitle = "Counts of predictions vs. truth") +
  theme_minimal()
```



- Sensitivity (Recall):
  - Calculation
    - $\frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{5,913}{5,913 + 351} = 0.944$
  - What is it?
    - It measure the model's ability to detect true cases. Our model correctly identifies 94.4% of all IBD cases. This is valuable for medical screening where missing a case is costly.
- Specificity:
  - Calculation
    - $\frac{\text{TN}}{\text{TN} + \text{TP}} = \frac{875}{875 + 1,522} = 0.365$
  - What is it?
    - It quantifies how well the model rules out negatives (controls). Only 36.5% of control samples are correctly classified, indicating a high false-positive rate. Misclassifying healthy samples as IBD could lead to unnecessary follow-up tests.

- Precision:
  - Calculation
    - $\frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{5,913}{5,913 + 1,522} = 0.795$
  - What is it?
    - It tells us the proportion of positive predictions that are actually correct. When our model predicts “IBD”, it is correct 79.5% of the time. Precision is moderate and can be improved to reduce false positives.
- F1:
  - Calculation
    - $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\ \text{TP}}{2\ \text{TP} + \text{FP} + \text{FN}} = \frac{2(5,913)}{2(5,913) + 1,522 + 351} = 0.863$
  - What is it?
    - The harmonic mean of precision and recall, giving equal weight to both metrics. The harmonic mean penalized extreme values, e.g., if either precision or recall is low, then the overall score will be driven down.
    - 1.0 $\implies$ perfect precision and recall.
      0.5 $\implies$ as good as random guessing.
      0.0 $\implies$ either precision or recall (or both) is zero.

Conclusion

We’ve classified subjects across several studies based on their gut microbiome samples, which may include repeat measurement samples, into two classes: IBD or control.

Our first step was to explore the counts data. We looked at the cohort characteristics of the data across the different studies. We then looked at α-diversity (Shannon/Simpson) and β-diversity to get a better understanding of the underlying data. We then proceeded to examine the data at the taxa-level to determine how to filter the data, ultimately deciding to keep taxa with reads ≥ 100 in at least 1% of samples.

Our second step was to investigate the filtered data. We performed another α-diversity analysis and ran the Wilcoxon-rank test, resulting in a rejection of the null hypothesis. That is, there is statistically significant evidence that the Shannon index in controls is not drawn from the same distribution as in IBD.

Our third step was to create, train, and test a Random Forest model. To account for repeated measurements across subjects, which violates the i.i.d. assumption underlying cross-validation, we incorporated repeated grouped k-fold CV, in our case, k = 5 with 3 repeats per fold. We fed the model CLR-transformed, which removes distortions of compositional bias, OTU abundances as features, which are now on an appropriate, comparable scale.

Our final step was to evaluate our model. We looked at ROC, AUC, calibration, precision-recall curves, AUPRC, and the confusion matrix and metrics. We determined that our model performs better than a random classifier, with a high AURPC alongside a low Brier score indicating both reliable probability estimates and strong discrimination.

Future work can look to improve the false positive rate for controls and compare different models for classification.

Appendix: Definitions and Explanations

α-diversity

Shannon Index

The Shannon index H quantifies both richness (number of taxa) and evenness (relative abundance distribution) in a single value. It is defined as

$$H = -\sum_{i=1}^{S} p_i \ln(p_i)$$

where

- S = total number of observed taxa (OTUs/ASVs) in the sample.
- p_i = proportion of reads belonging to taxon i (i.e. $p_i = \frac{n_i}{\sum_{j=1}^{S} n_j}$, where n_i is the count of taxon i).

Key properties:

- H increases as (1) the number of taxa S increases, and (2) the abundances become more even.
- Minimum $H = 0$ occurs when one taxon comprises 100% of the sample.
- Maximum $H = \ln(S)$ occurs when all S taxa are equally abundant ($p_i = 1/S$ for all i).

Simpson index

The **Simpson index** measures dominance (or conversely, diversity) by focusing on the probability of picking two reads from the same taxon. There are two common forms:

1. **Raw Simpson (D):**

$$D = \sum_{i=1}^{S} p_i^2$$

   where, D is the probability that two randomly drawn reads (with replacement) come from the same taxon.

   - $ D $
     - $D = 1$ if a single taxon dominates 100%.
     - $D = 1/S$ if all S taxa are equally abundant.

2. **Gini–Simpson or “Simpson’s Diversity” (1 – D):**

$$1 - D = 1 - \sum_{i=1}^{S} p_i^2$$

   - $1 - D \in \left[0,\ 1 - \frac{1}{S}\right]$
     - $1 - D = 0$ when one taxon dominates entirely.

- $1 - D$ approaches 1 as the community becomes more even and richly diverse.

Observed Richness

Observed richness simply counts how many taxa (OTUs/ASVs) are present in a sample:

$$S_{\text{obs}} = \sum_{i=1}^{S} \mathbb{1}(n_i > 0)$$

where $\mathbb{1}(\cdot)$ is an indicator that equals 1 if taxon i has at least one read. Thus, Observed richness measures raw taxon count without regard to abundance.

Chao1 Estimator

Chao1 is a non-parametric richness estimator that adjusts Observed richness for unseen (rare) taxa, using singletons (taxa with one read) and doubletons (two reads):

$$\widehat{S}_{\text{Chao1}} = S_{\text{obs}} + \frac{f_1^2}{2 f_2}$$

where

- S_{obs} = number of observed taxa.
- f_1 = count of taxa observed exactly once.
- f_2 = count of taxa observed exactly twice.

When $f_2 = 0$, one often uses a bias-corrected form or sets $\widehat{S}_{\text{Chao1}} = S_{\text{obs}} + \frac{f_1(f_1-1)}{2}$

CLR-transform

The centered log-ratio transform is a mapping of each composition x_i (either a taxon count or proportion) from the simplex S^d into the unconstrained (real) subspace $\{y \in \mathbb{R}^d : \sum_{i=1}^{d} y_i = 0\}$.

Formula Summary

For a sample with taxa counts (or proportions) $(x_1, x_2, \ldots, x_d)$, define the geometric mean

$$g = \left(\prod_{i=1}^{d} x_i\right)^{1/d}$$

Then the CLR-transformed vector is

$$\text{clr}(x)_i = \log\left(\frac{x_i}{g(x)}\right) \quad \text{for } i = 1, \ldots, d.$$

Because $\text{clr}(x)_i = 0$, the transform removes the simplex constraint (i.e., the simplex constraint is that each sample's taxa counts (or proportions) $(x_1, x_2, \ldots, x_d)$ satisfy:

1. Non-negativity: $x_i \geq 0 \quad \forall i \in \{1, \ldots, d\}$
2. Constant-sum (closure) constraint: $\sum_{i=1}^{d} x_i = 1$),

avoiding spurious correlations and misleading results that would arise from failing to meet the “unconstrained Euclidean space” assumption for PCA, Euclidean distances, and several classifiers.

Wilcoxon rank-sum test

The Wilcoxon rank-sum test (also called the Mann–Whitney U test) is a non-parametric method for comparing two independent groups. It is an alternative to a two-sample t-test and does not assume normality, only that the two groups are independent and that the outcome is at least ordinal.

Quick summary

1. Pool all observations from both groups and rank them from smallest to largest.
2. Sum the ranks within each group.
3. Compute the test statistic U.
4. Use either an exact table (small samples) or a normal approximation (large samples) to obtain a p-value under the null hypothesis that the two groups come from the same distribution.
    - Since we have moderate/large samples, we can calculate Z as

$$Z = \frac{U - \mu_U}{\sigma_U}$$

where

$$\mu_U = \frac{n_1 n_2}{2}, \quad \sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

Formula summary

- **Ranks**: assign ranks to the pooled data.
- **Rank-sum**:

$$R_1 = \sum_{\text{all observations in group 1}} (\text{rank})$$

- **U statistic** for group 1:

$$U_1 = R_1 - \frac{n_1(n_1 + 1)}{2}$$

- **Mean and variance of U under** H_0:

$$\mu_U = \frac{n_1 n_2}{2}, \quad \sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

- Z**-score approximation**:

$$Z = \frac{U - \mu_U}{\sigma_U}$$

- **Two-sided p-value** (for large samples):

$$p = 2 \times P\left(|Z| \geq |Z_{\text{observed}}|\right)$$

Brier score

A proper scoring rule that measures the accuracy of probabilistic predictions for binary outcomes by computing the mean squared difference between predicted probabilities and actual outcomes. A lower Brier score indicates better calibration and discrimination, with a perfect score of 0 and a worst score of 1.

Formula summary

For **binary** events, the Brier score is defined as:

$$\text{Brier Score} = \frac{1}{N} \sum_{i=1}^{N} (p_i - o_i)^2$$

where:

- p_i is the predicted probability of the positive class for instance i,
- o_i is the observed outcome (1 if the event occurred, 0 otherwise),
- N is the number of forecasts

This can be thought of as the mean squared error of the predicted probabilities.

References

1. Cao Q, Sun X, Rajesh K, Chalasani N, Gelow K, Katz B, Shah VH, Sanyal AJ, Smirnova E. Effects of Rare Microbiome Taxa Filtering on Statistical Analysis. Front Microbiol. 2021 Jan 12;11:607325. doi: 10.3389/fmicb.2020.607325. PMID: 33510727; PMCID: PMC7835481.