Lesson 12

Vicki Hertzberg

4/2/2017

Microbial Ecology - The Theory

In ecology, scientists collect quantitative data to determine the resemblance between either the objects under study (objects) or the variables describing them (species or other descriptors). The analysis of the association between objects and descriptors is the first step in the analysis.

There are three main types of association:

- Similarity
- Distance
- Dependence

After creating these measures of association, common tasks are to cluster or to ordinate.

To cluster: Sets of objects (or descriptors) can be partitioned into two or more subsets (clusters) using pre-established rules of agglomeration.

To ordinate: Objects (or descriptors) are represented in a space containing fewer dimensions than the original dataset such that the positions of the objects (or descriptors) with respect to one another may also be used to cluster them.

Microbial Ecology - The Practice

As scientists interested in the microbiome, we are now all microbial ecologists. We typically talk about objects as subjects from whom samples have been taken (or even sites within subjects, e.g., oral, vaginal, rectal) while the descriptors are the list of abundances of each species (or other taxonomic classification) found.

So we become interested in how similar subjects are to each other, say, within group i, and how distant they are from, say, group j. Then we might want to know if that distance is associated with some other difference between the groups, for instance, are groups that are closer together in site pH levels also more similar (i.e., less distant).

Other questions of interest: how do sites (or species) cluster together? Also, can you use a multivariate dimensionality reduction technique to describe differences while preserving distances?

Recall if it is appropriate to determine distances between two points with Euclidean distance, the ensuing multidimensional reduction technique is principal component analysis (PCA). If the data are counts then the distance should be calculated in terms of the χ^2 distance, and then the reduction technique is canonical

component analysis (CCA). If the distance metrics take on some other value (e.g., dissimilarity), then principal coordinate analysis (PCoA), which is really PCA of the squared distance matrix.

These are the types of questions that we can use `phyloseq` to address with our OTUtables and our sample data.

Beyond the OTU Table

`phyloseq` is an R package for efficient interactive and reproducible analyses of OTU-clustered high-throughput phylogenetic sequencing data. We can take the sequence variant table produced by `dada2` and use it, along with taxonomic classification and data about the samples, in these analyses. `Phyloseq` imports, stores, analyzes, and displays these data.

In our last lesson we showed how to hand off data to `phyloseq` from `dada2` , using the `phyloseq` command to create a `phyloseq` object from an OTU table, a taxonomic table, and a sample dataset.

In this lesson we will some of the other functionality of `phyloseq` :

- explore some of the example datasets that are included;
- creating `phyloseq` objects from the output of other programs;
- graphics in `phyloseq'.

Load Packages and Example Data

To get started we will need to load both `phyloseq` and `ggplot2` .

```
# Load packages

library(phyloseq)
packageVersion("phyloseq")
```

```
## [1] '1.19.1'
```

```
library(ggplot2)
packageVersion("ggplot2")
```

```
## [1] '2.2.1'
```

```
library(RColorBrewer)
packageVersion("RColorBrewer")
```

```
## [1] '1.1.2'
```

Load example data using the `data()` command.

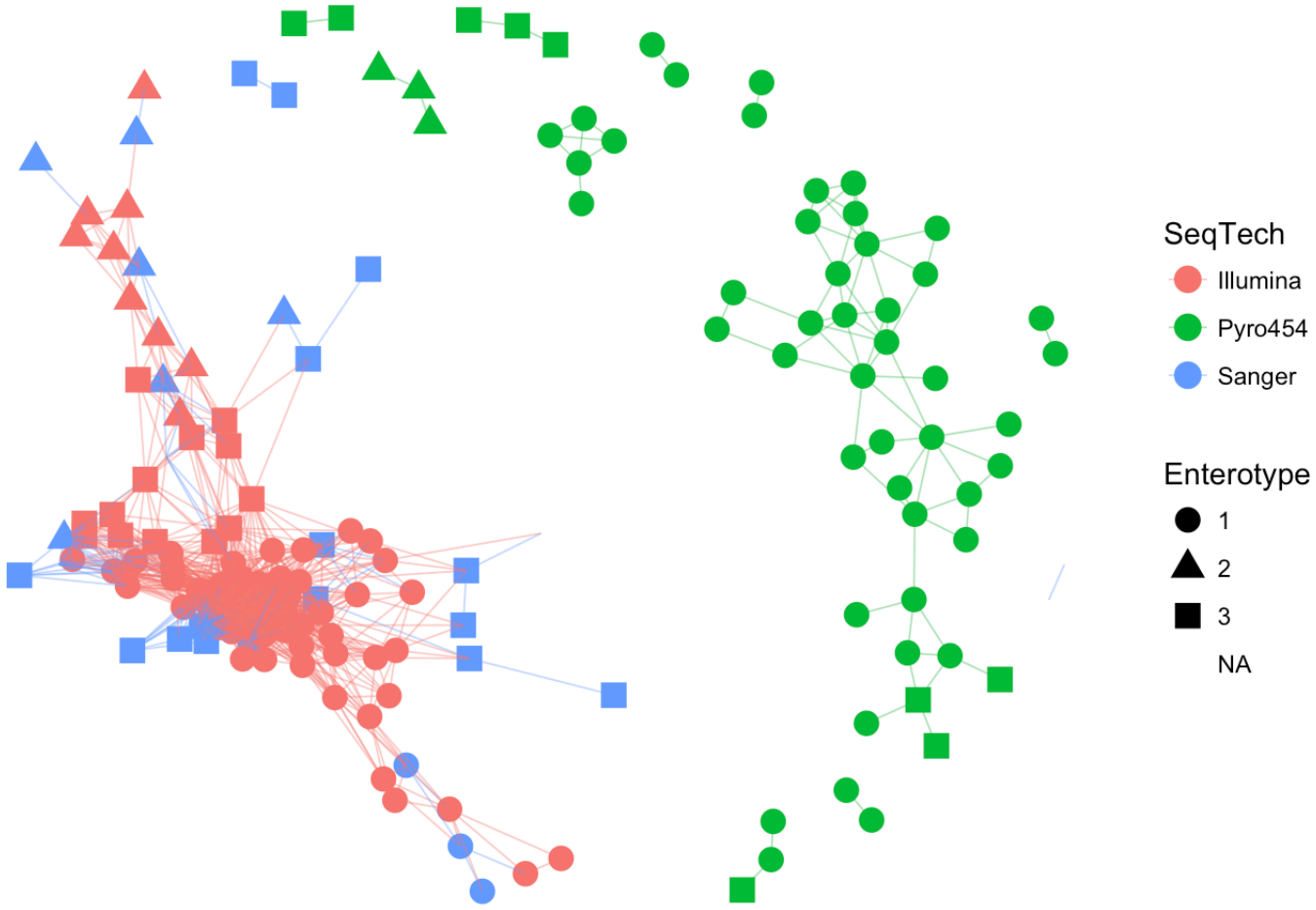
```
# Load example data

data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

You can try these examples using the `example()` command.

```
# Use the example() command on the enterotype database

example(enterotype, ask=FALSE)
```

```
##
## entrty> data(enterotype)
##
## entrty> ig <- make_network(enterotype, "samples", max.dist=0.3)
##
## entrty> plot_network(ig, enterotype, color="SeqTech", shape="Enterotype", line_wei
ght=0.3, label=NULL)
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```



You can also import data from other packages such as MG-RAST or QIIME. The newer versions of QIIME produce files formatted according to the *biom* standard (see http://biom-format.org (http://biom-format.org)).

The phyloseq package has small examples of biom files with different levels and organization of data. The following chunk illustrates how to import each of the 4 main types of biom files. Note that in practice you won't know what type your file is, so best to include all 4. In addition, the `import_biom` funciton allows you to import an associated phylogenetic tree file and reference sequence file (e.g., fasta file).

To do this you must first define the file paths. For this example, this should be within the phyloseq package. Thus we use the `system.file` command to get the paths.

```
# Load the files
rich_dense_biom  <- system.file("extdata", "rich_dense_otu_table.biom",  package="phy
loseq")
rich_sparse_biom <- system.file("extdata", "rich_sparse_otu_table.biom", package="phy
loseq")
min_dense_biom   <- system.file("extdata", "min_dense_otu_table.biom",   package="phy
loseq")
min_sparse_biom  <- system.file("extdata", "min_sparse_otu_table.biom",  package="phy
loseq")
treefilename <- system.file("extdata", "biom-tree.phy",  package="phyloseq")
refseqfilename <- system.file("extdata", "biom-refseq.fasta",  package="phyloseq")
```

Next we use these paths as argments to the `import_biom` function. The tree and reference sequence files are both suitable for any of the biom file types, so we only need one path for each.

```
# Import via the paths

import_biom(rich_dense_biom, treefilename, refseqfilename, parseFunction = parse_taxo
nomy_greengenes)
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5 taxa and 6 samples ]
## sample_data() Sample Data:       [ 6 samples by 4 sample variables ]
## tax_table()   Taxonomy Table:    [ 5 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 5 tips and 4 internal nodes ]
## refseq()      DNAStringSet:      [ 5 reference sequences ]
```

```
import_biom(rich_sparse_biom, treefilename, refseqfilename, parseFunction = parse_tax
onomy_greengenes)
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5 taxa and 6 samples ]
## sample_data() Sample Data:       [ 6 samples by 4 sample variables ]
## tax_table()   Taxonomy Table:    [ 5 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 5 tips and 4 internal nodes ]
## refseq()      DNAStringSet:      [ 5 reference sequences ]
```

```
import_biom(min_dense_biom, treefilename, refseqfilename, parseFunction = parse_taxon
omy_greengenes)
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5 taxa and 6 samples ]
## phy_tree()    Phylogenetic Tree: [ 5 tips and 4 internal nodes ]
## refseq()      DNAStringSet:      [ 5 reference sequences ]
```

```
import_biom(min_sparse_biom, treefilename, refseqfilename, parseFunction = parse_taxo
nomy_greengenes)
```

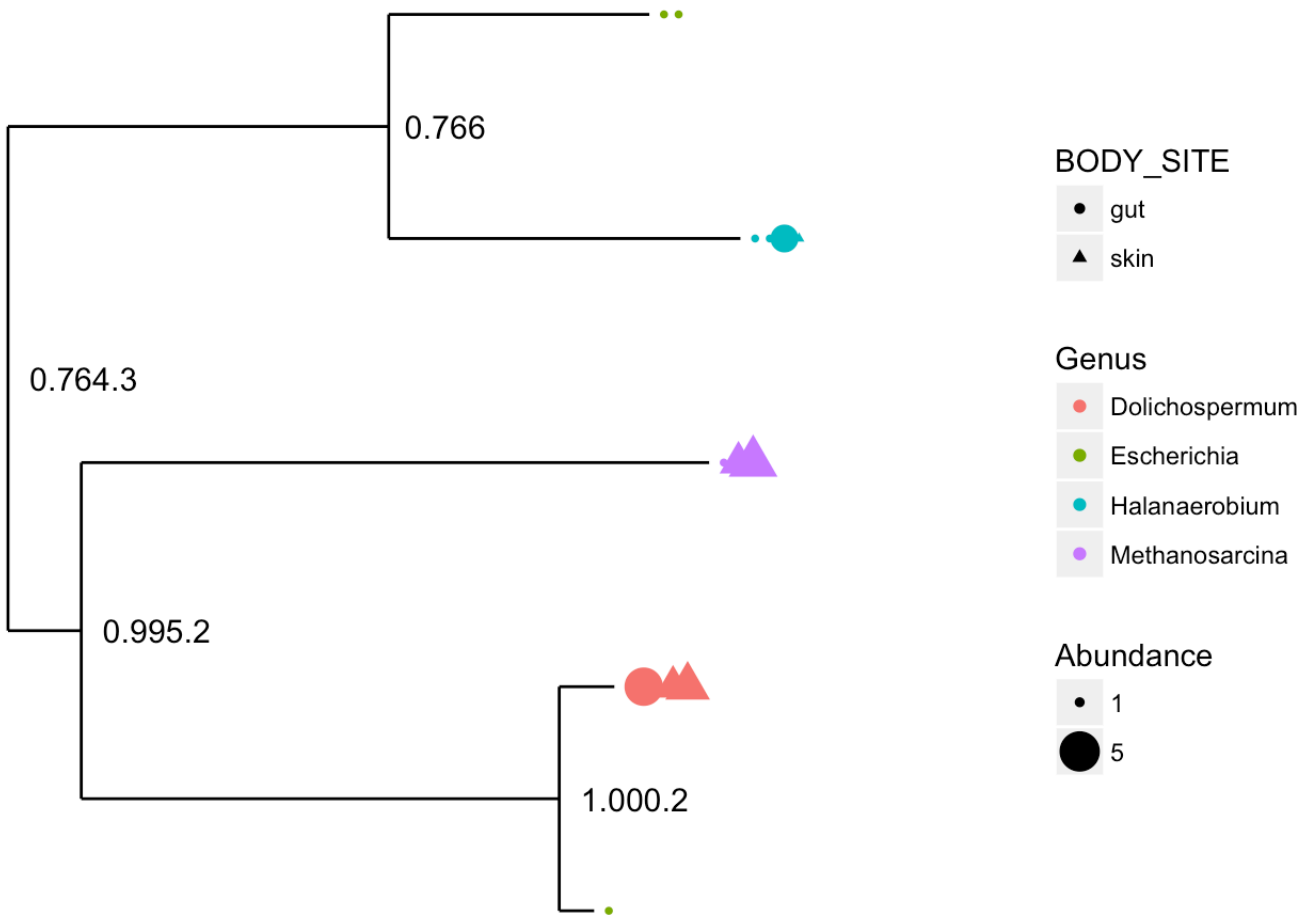
```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5 taxa and 6 samples ]
## phy_tree()    Phylogenetic Tree: [ 5 tips and 4 internal nodes ]
## refseq()      DNAStringSet:      [ 5 reference sequences ]
```

In practice you will store the result of your import as some variable name, like `myData` , then use this object in downstream analyses. For examples,

```
# Example of storing results of import and manipulating it

myData <- import_biom(rich_dense_biom, treefilename, refseqfilename, parseFunction =
parse_taxonomy_greengenes)
myData
```

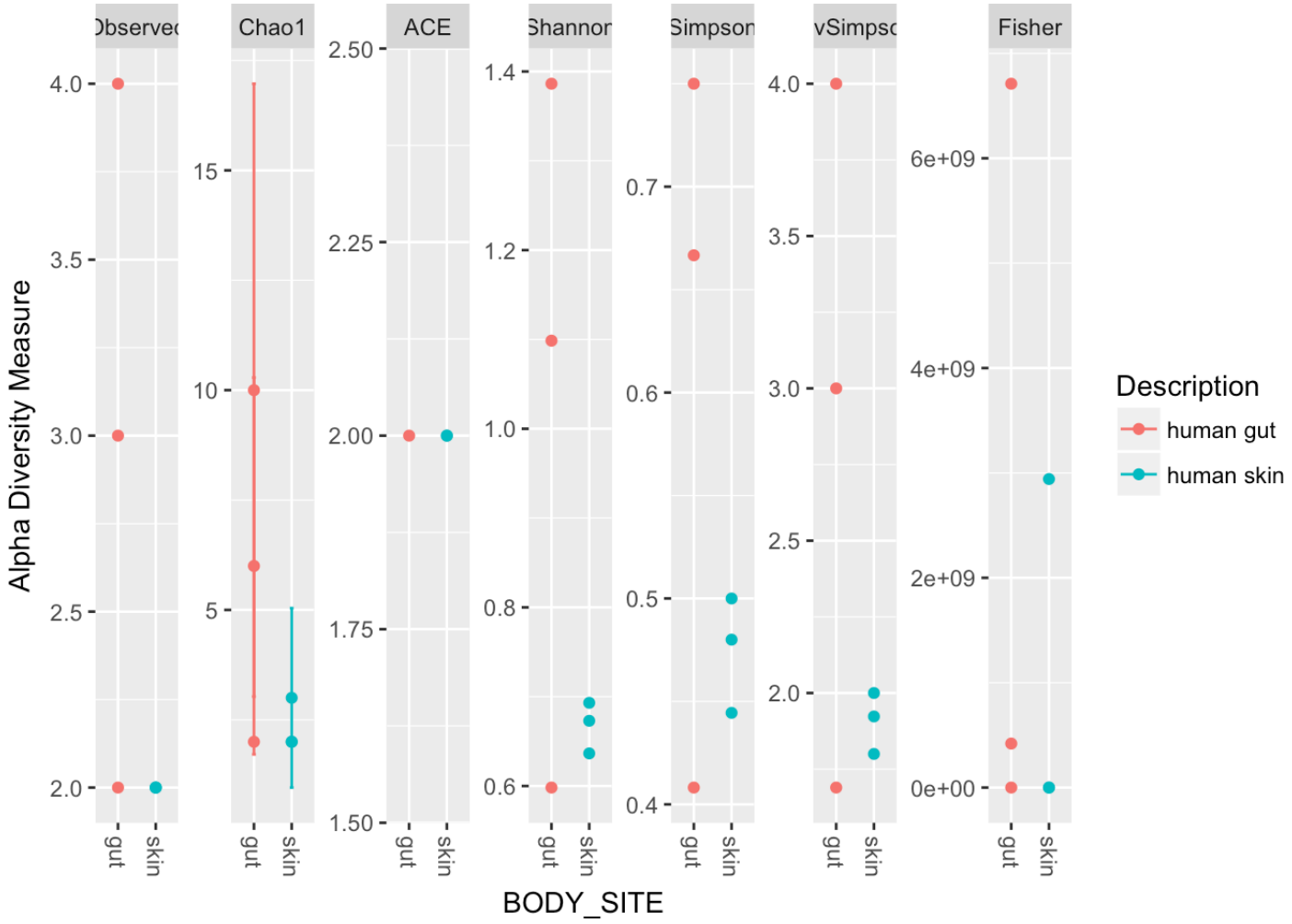
```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5 taxa and 6 samples ]
## sample_data() Sample Data:       [ 6 samples by 4 sample variables ]
## tax_table()   Taxonomy Table:    [ 5 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 5 tips and 4 internal nodes ]
## refseq()      DNAStringSet:      [ 5 reference sequences ]
```

```
plot_tree(myData, color = "Genus", shape = "BODY_SITE", size = "abundance")
```

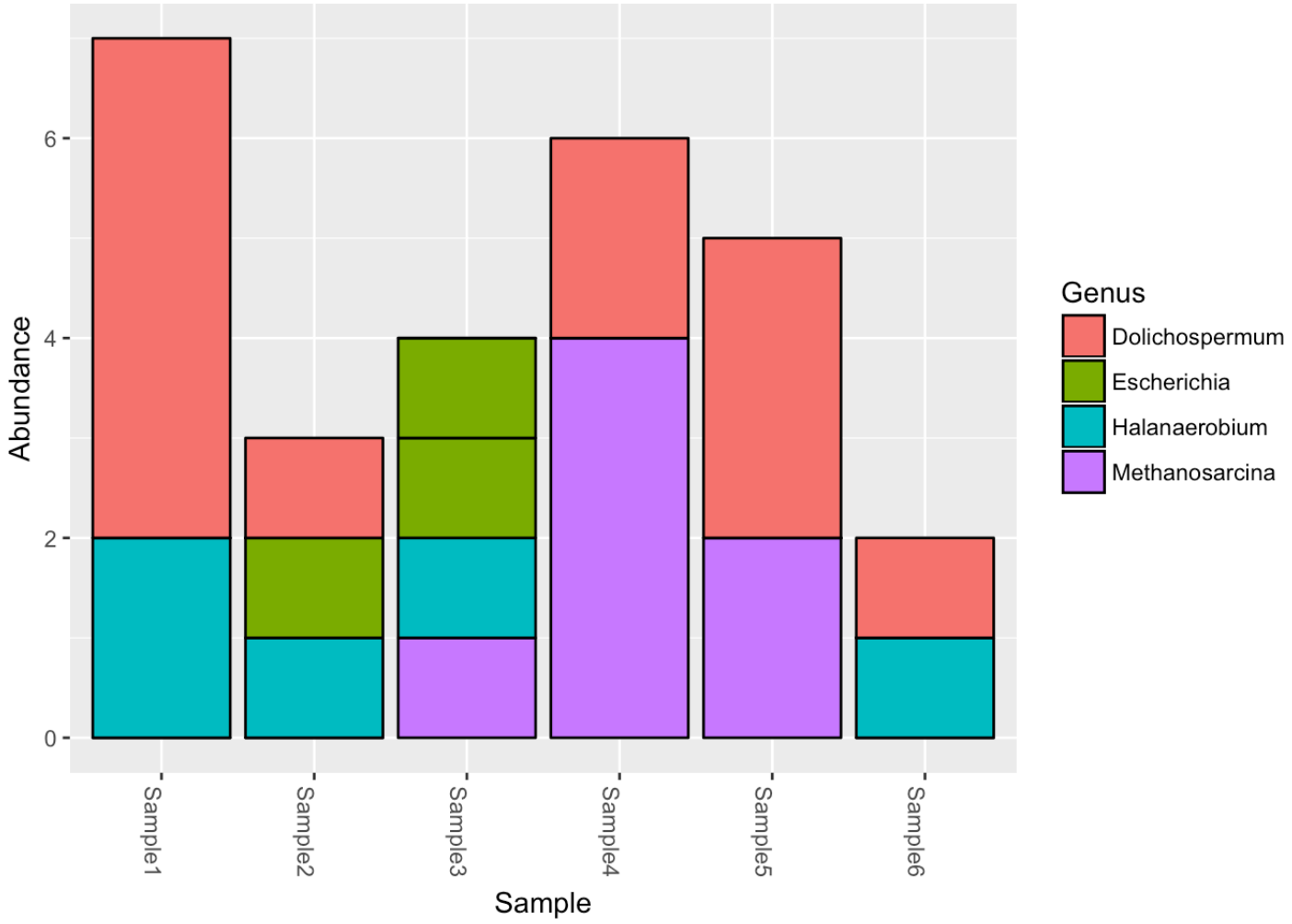



```
plot_richness(myData, x="BODY_SITE", color="Description")
```

```
## Warning: Removed 33 rows containing missing values (geom_errorbar).
```




```
plot_bar(myData, fill="Genus")
```




```
refseq(myData)
```

```
##   A DNAStringSet instance of length 5
##     width seq                                               names
## [1]   334 AACGTAGGTCACAAGCGTTGT...TTCCGTGCCGGAGTTAACAC GG_OTU_1
## [2]   465 TACGTAGGGAGCAAGCGTTAT...CCTTACCAGGGCTTGACATA GG_OTU_2
## [3]   249 TACGTAGGGGGCAAGCGTTAT...GGCTCGAAAGCGTGGGGAGC GG_OTU_3
## [4]   453 TACGTATGGTGCAAGCGTTAT...AAGCAACGCGAAGAACCTTA GG_OTU_4
## [5]   178 AACGTAGGGTGCAAGCGTTGT...GGAATGCGTAGATATCGGGA GG_OTU_5
```

The good people at `phyloseq` have also downloaded data from the Human Microbiome Project and made it available as an R dataset. So let's look at it

```
# Get the HMP data
# You will need to change your path

load("~/Documents/NRSG_741/Lesson12/HMPv35.RData")

HMPv35
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 45336 taxa and 4743 samples ]
## sample_data() Sample Data:       [ 4743 samples by 9 sample variables ]
## tax_table()   Taxonomy Table:    [ 45336 taxa by 6 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 45336 tips and 45099 internal nodes ]
## refseq()      DNAStringSet:      [ 45336 reference sequences ]
```

Merge Data

`phyloseq` supports two types of merging.

1. You can merge OTUs or samples in a phyloseq object, based on a taxonomic or sample variable using the `merge_samples()` or `merge_taxa()` commands.
2. YOu can merge two or more data objects from the same experiment so that their data become part of the same phyloseq object using the `merge_phyloseq()` commend.

Merging Samples

Merging samples with `merge_samples()` can be a useful means of reducing noise or excess features, say by removing the individual effects between replicates or between samples for a particular explanatory variable. With `merge_samples()` the abundance values for the merged samples are summed.

As an example, let's first remove unobserved OTUs (those that sum 0 across all samples) and add an explanatory variable with which to organize later in plots, using the GlobalPatterns dataset.

```
# Remove empty samples

GP <- GlobalPatterns
GP <- prune_taxa(taxa_sums(GlobalPatterns) > 0, GlobalPatterns)
humantypes <- c("Feces", "Mock", "Skin", "Tongue")
sample_data(GP)$human <- get_variable(GP, "SampleType") %in% humantypes
```

Now on to merging samples:

```
# Merge Samples

mergedGP <- merge_samples(GP, "SampleType")
SD <- merge_samples(sample_data(GP), "SampleType")
print(SD[, "SampleType"])
```

```
##                     SampleType
## Feces                        1
## Freshwater                   2
## Freshwater (creek)           3
## Mock                         4
## Ocean                        5
## Sediment (estuary)           6
## Skin                         7
## Soil                         8
## Tongue                       9
```

```
print(mergedGP)
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 18988 taxa and 9 samples ]
## sample_data() Sample Data:       [ 9 samples by 8 sample variables ]
## tax_table()   Taxonomy Table:    [ 18988 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 18988 tips and 18987 internal nodes ]
```

```
sample_names(GP)
```

```
##  [1] "CL3"      "CC1"      "SV1"      "M31Fcsw"  "M11Fcsw"  "M31Plmr"
##  [7] "M11Plmr"  "F21Plmr"  "M31Tong"  "M11Tong"  "LMEpi24M" "SLEpi20M"
## [13] "AQC1cm"   "AQC4cm"   "AQC7cm"   "NP2"      "NP3"      "NP5"
## [19] "TRRsed1"  "TRRsed2"  "TRRsed3"  "TS28"     "TS29"     "Even1"
## [25] "Even2"    "Even3"
```

```
sample_names(mergedGP)
```

```
## [1] "Feces"             "Freshwater"        "Freshwater (creek)"
## [4] "Mock"              "Ocean"             "Sediment (estuary)"
## [7] "Skin"              "Soil"              "Tongue"
```

```
identical(SD, sample_data(mergedGP))
```

```
## [1] TRUE
```

The OTU abundances of merged samples are summed. Let's investigate this ourselves looking at just the top 10 most abundant OTUs.

```
# Look at top 10 most abundant OTUs

OTUnames10 <- names(sort(taxa_sums(GP), TRUE)[1:10])
GP10 <- prune_taxa(OTUnames10, GP)
mGP10 <- prune_taxa(OTUnames10,mergedGP)
ocean_samples <- sample_names(subset(sample_data(GP), SampleType == "Ocean"))
print(ocean_samples)
```

```
## [1] "NP2" "NP3" "NP5"
```

```
otu_table(GP10)[, ocean_samples]
```

```
## OTU Table:          [10 taxa and 3 samples]
##                      taxa are rows
##         NP2   NP3   NP5
## 329744   91   126   120
## 317182 3148 12370 63084
## 549656 5045 10713  1784
## 279599  113   114   126
## 360229   16    83   786
## 94166    49   128   709
## 550960   11    86    65
## 158660   13    39    28
## 331820   24   101   105
## 189047    4    33    29
```

```
rowSums(otu_table(GP10)[, ocean_samples])
```

```
## 329744 317182 549656 279599 360229  94166 550960 158660 331820 189047
##    337  78602  17542    353    885    886    162     80    230     66
```

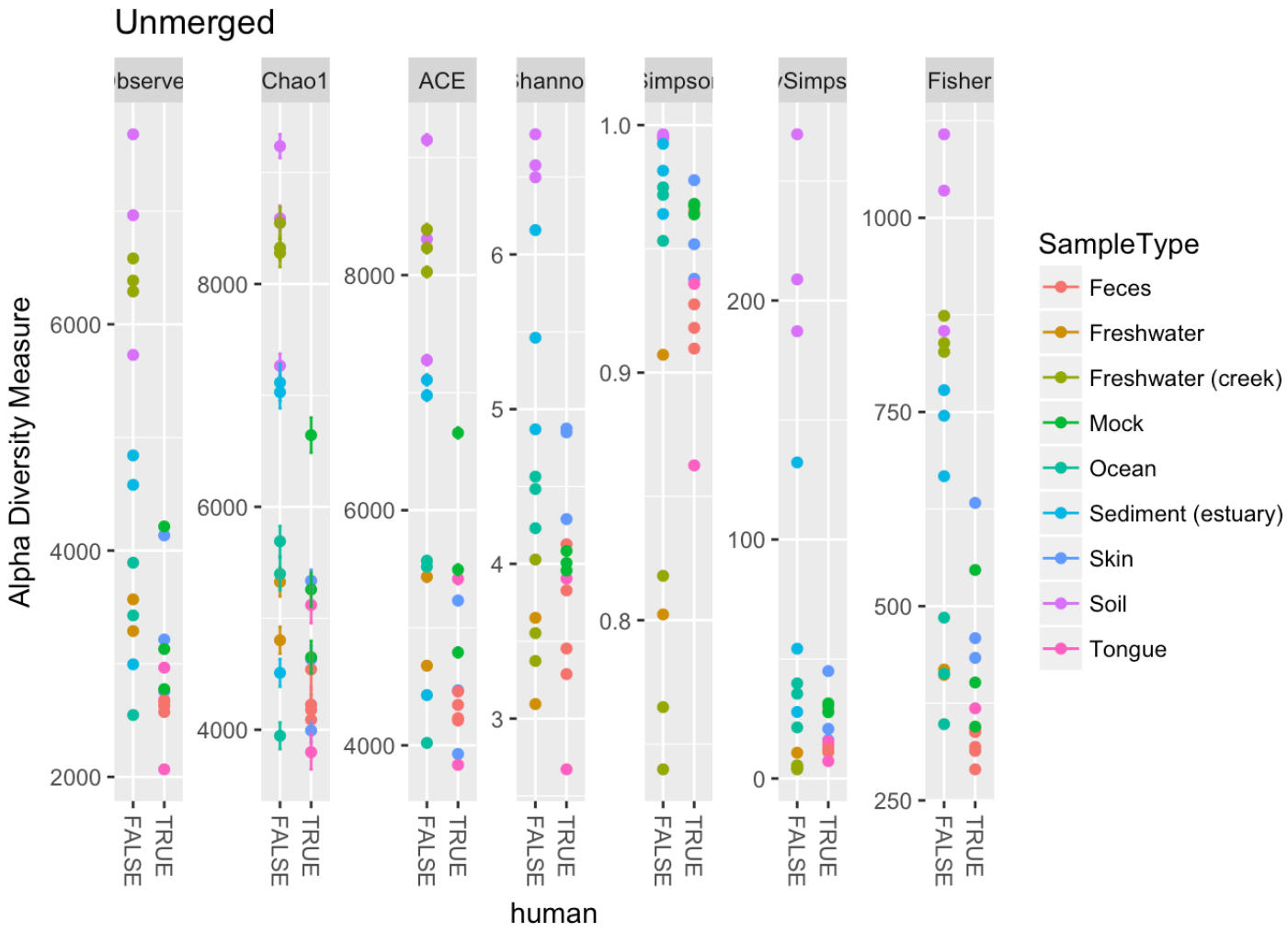
```
otu_table(mGP10)["Ocean", ]
```

```
## OTU Table:          [10 taxa and 1 samples]
##                      taxa are columns
##       329744 317182 549656 279599 360229 94166 550960 158660 331820 189047
## Ocean    337  78602  17542    353    885   886    162     80    230     66
```

Now let's look at the merge graphically between two richnes estimate summary plots.

```
# Richness estimate plots

plot_richness(GP, "human", "SampleType", title="Unmerged")
```

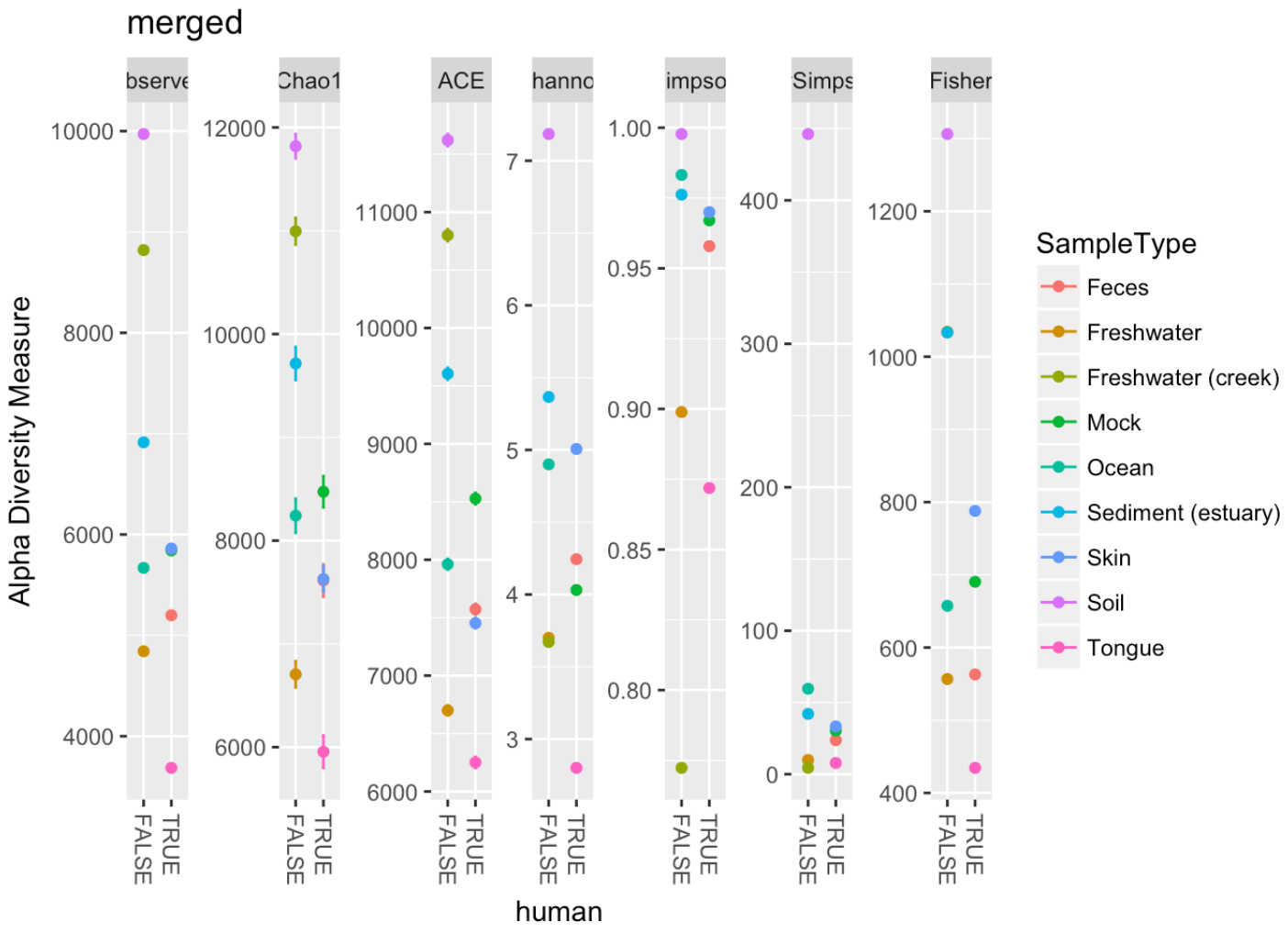
```
## Warning: Removed 130 rows containing missing values (geom_errorbar).
```



The merge can do some weird things to samples variables, so let's re-add these variables to the `sample_data` before we plot.

```
# Re-add sample data and plot richness plots again.

sample_data(mergedGP)$SampleType <- sample_names(mergedGP)
sample_data(mergedGP)$human <- sample_names(mergedGP) %in% humantypes
plot_richness(mergedGP, "human", "SampleType", title="merged")
```

```
## Warning: Removed 45 rows containing missing values (geom_errorbar).
```

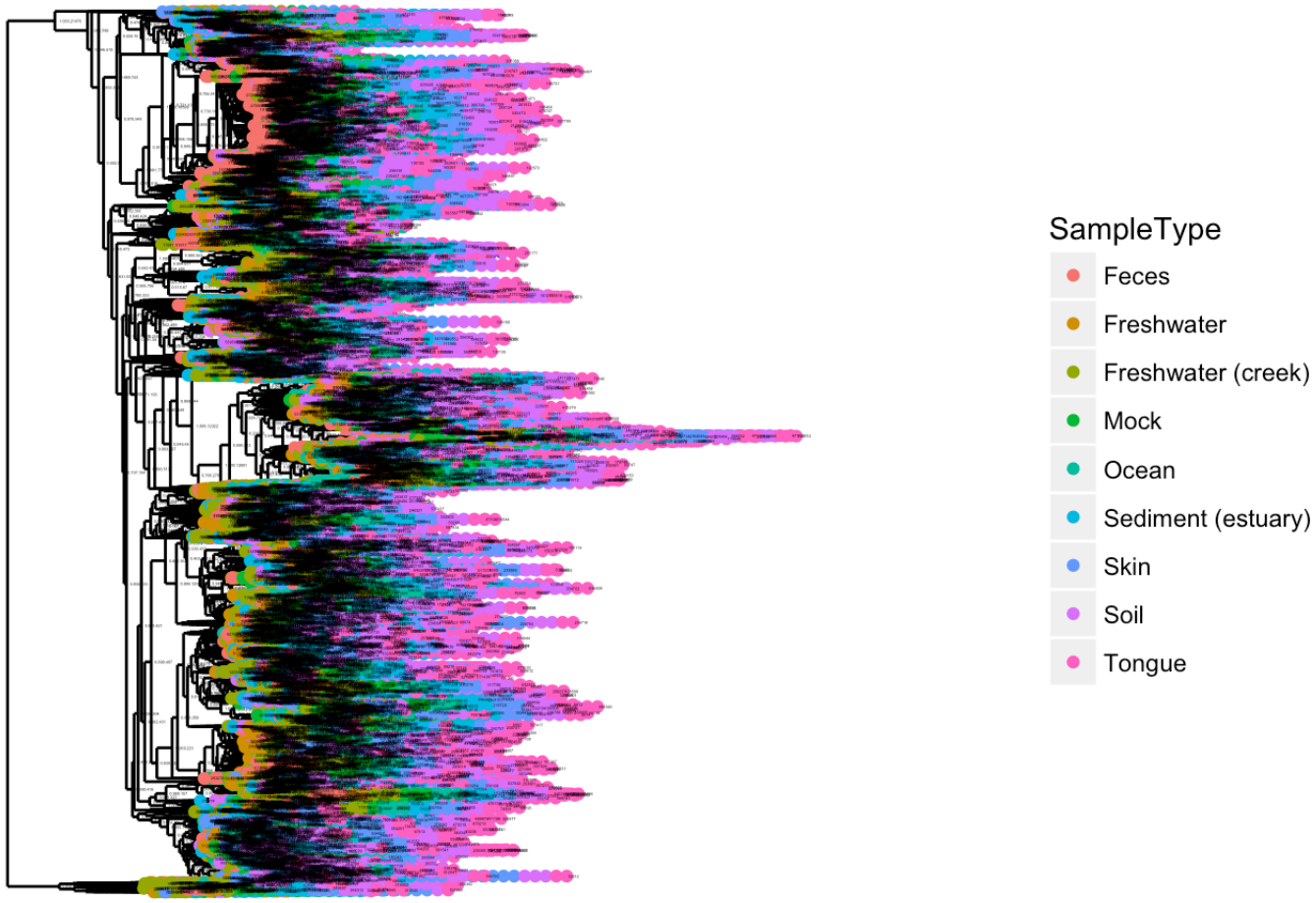


When we combine the abundances of non-replicate samples from the same environment, the estimates of absolute richness increase for each environment. But, more to the point, the new plot is easier to read and interpret, one of the reasons one might use the `merge_samples` function.

Merging Taxa

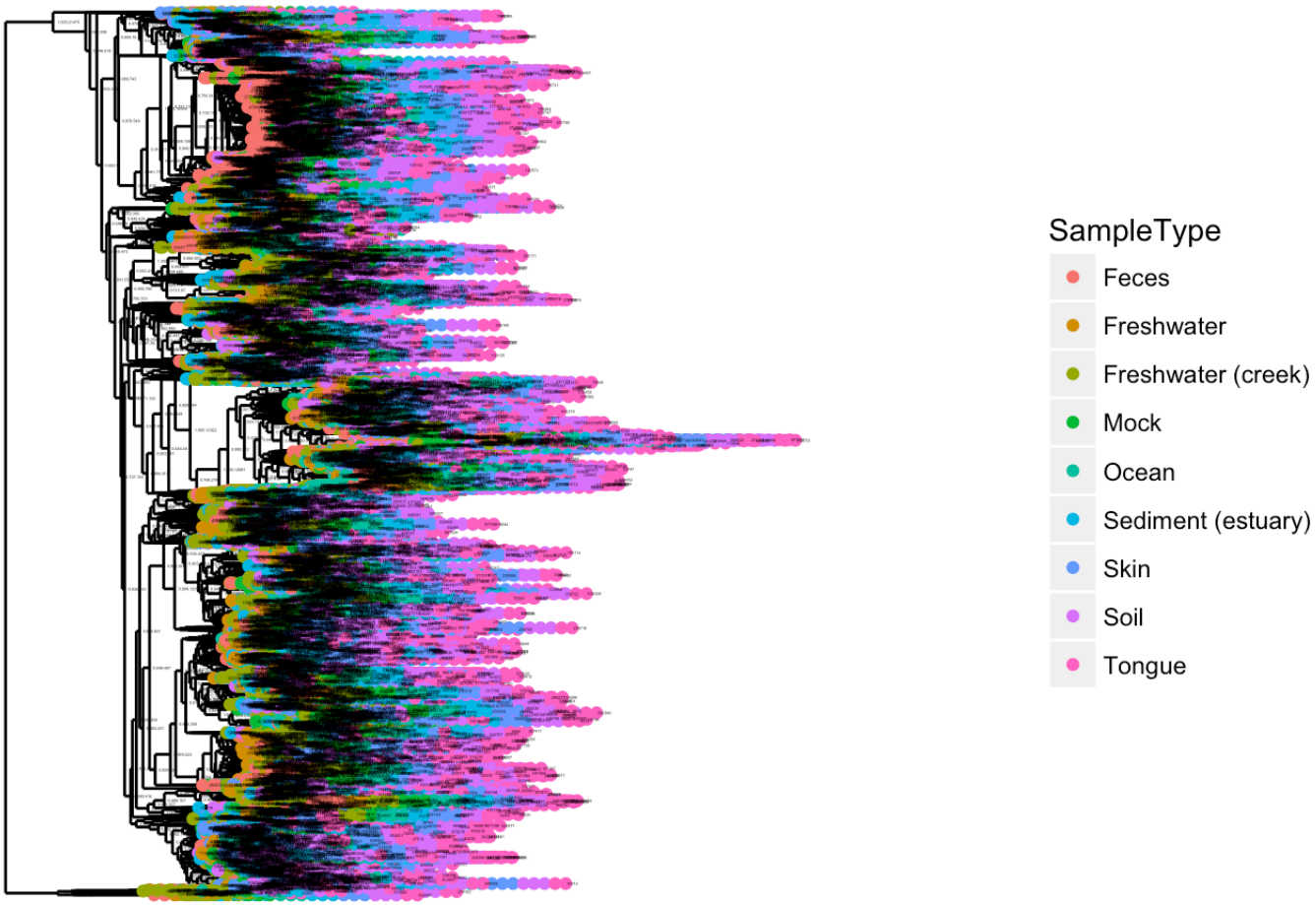
One of the issues with microbial census data is that a fine-scaled definition for OTU may blur a pattern that might otherwise be evident if we considered a higher taxonomic rank. The best way to deal with this is to agglomerate phylogenetic tree tips or taxa by using the agglomeration functions `tip_glom` or `tax_glom`. Let's use the object we imported earlier from the biom format files, myData. First the original tree:

```
# Plot phylogenetic tree of GlobalPatterns object
plot_tree(GP, color="SampleType", sizebase=2, label.tips="taxa_name")
```



What a mess. Let's try using `merge_taxa` to merge the first 100 OTUs into one new OTU. The option 2 in the call means to combine the counts of these 100 OTUs into the index for the new OTU.

```
# Combine the tree tips and plot again

x1 <- merge_taxa(GP, taxa_names(GP)[1:100], 2)
plot_tree(x1, color="SampleType", sizebase=2, label.tips="taxa_name")
```



You can barely see the difference, but it is there.

Merging Objects

You can also merge phyloseq objects. We will demonstrate with a somewhat trivial example by extracting the components of the GlobalPatterns object and then building them back to the original form using the command `merge_phyloseq`.

```
# Split apart GlobalPatterns

data(GlobalPatterns)
tree <- phy_tree(GlobalPatterns)
tax <- tax_table(GlobalPatterns)
otu <- otu_table(GlobalPatterns)
sam <- sample_data(GlobalPatterns)

# Create new phyloseq object

otutax <- phyloseq(otu, tax)
otutax
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
```

You can se that our new `otutax` object has just the OTU table and the taxonomy table. Let's use `merge_phyloseq` to build up the original `GlobalPatterns` object, and compare to make sure they are identical.

```
# Build back up

GP2 = merge_phyloseq(otutax, sam, tree)

# Test for identity

identical(GP2, GlobalPatterns)
```

```
## [1] TRUE
```

Accessing and (Pre)Processing Data

Accessors

We have already seen a bit above about how to access some of the data within a `phyloseq` object. We will comprehensively cover it now.

Let's use the `GlobalPatterns` dataset.

```
# Load data object and see what is there

data("GlobalPatterns")
GlobalPatterns
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

```
ntaxa(GlobalPatterns)
```

```
## [1] 19216
```

```
nsamples(GlobalPatterns)
```

```
## [1] 26
```

```
sample_names(GlobalPatterns)[1:5]
```

```
## [1] "CL3"     "CC1"     "SV1"     "M31Fcsw" "M11Fcsw"
```

```
rank_names(GlobalPatterns)
```

```
## [1] "Kingdom" "Phylum"  "Class"   "Order"   "Family"  "Genus"   "Species"
```

```
sample_variables(GlobalPatterns)
```

```
## [1] "X.SampleID"               "Primer"
## [3] "Final_Barcode"            "Barcode_truncated_plus_T"
## [5] "Barcode_full_length"      "SampleType"
## [7] "Description"
```

```
otu_table(GlobalPatterns)[1:5, 1:5]
```

```
## OTU Table:          [5 taxa and 5 samples]
##                      taxa are rows
##        CL3 CC1 SV1 M31Fcsw M11Fcsw
## 549322   0   0   0       0       0
## 522457   0   0   0       0       0
## 951      0   0   0       0       0
## 244423   0   0   0       0       0
## 586076   0   0   0       0       0
```

```
tax_table(GlobalPatterns)[1:5, 1:4]
```

```
## Taxonomy Table:     [5 taxa by 4 taxonomic ranks]:
##        Kingdom   Phylum          Class          Order
## 549322 "Archaea" "Crenarchaeota" "Thermoprotei" NA
## 522457 "Archaea" "Crenarchaeota" "Thermoprotei" NA
## 951    "Archaea" "Crenarchaeota" "Thermoprotei" "Sulfolobales"
## 244423 "Archaea" "Crenarchaeota" "Sd-NA"        NA
## 586076 "Archaea" "Crenarchaeota" "Sd-NA"        NA
```

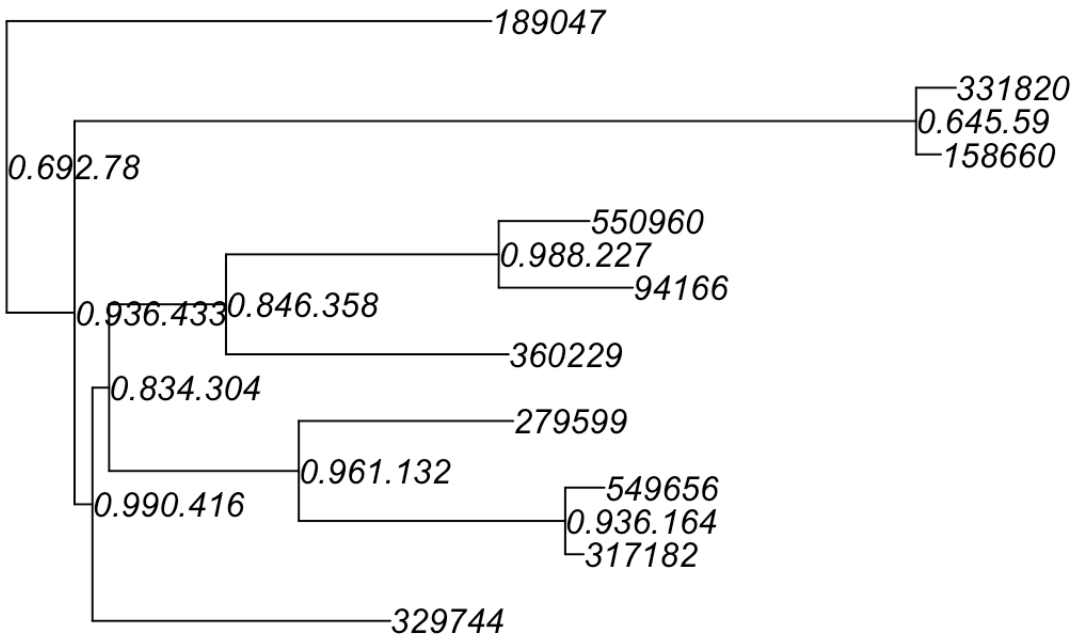
```
phy_tree(GlobalPatterns)
```

```
##
## Phylogenetic tree with 19216 tips and 19215 internal nodes.
##
## Tip labels:
##  549322, 522457, 951, 244423, 586076, 246140, ...
## Node labels:
##  , 0.858.4, 1.000.154, 0.764.3, 0.995.2, 1.000.2, ...
##
## Rooted; includes branch lengths.
```

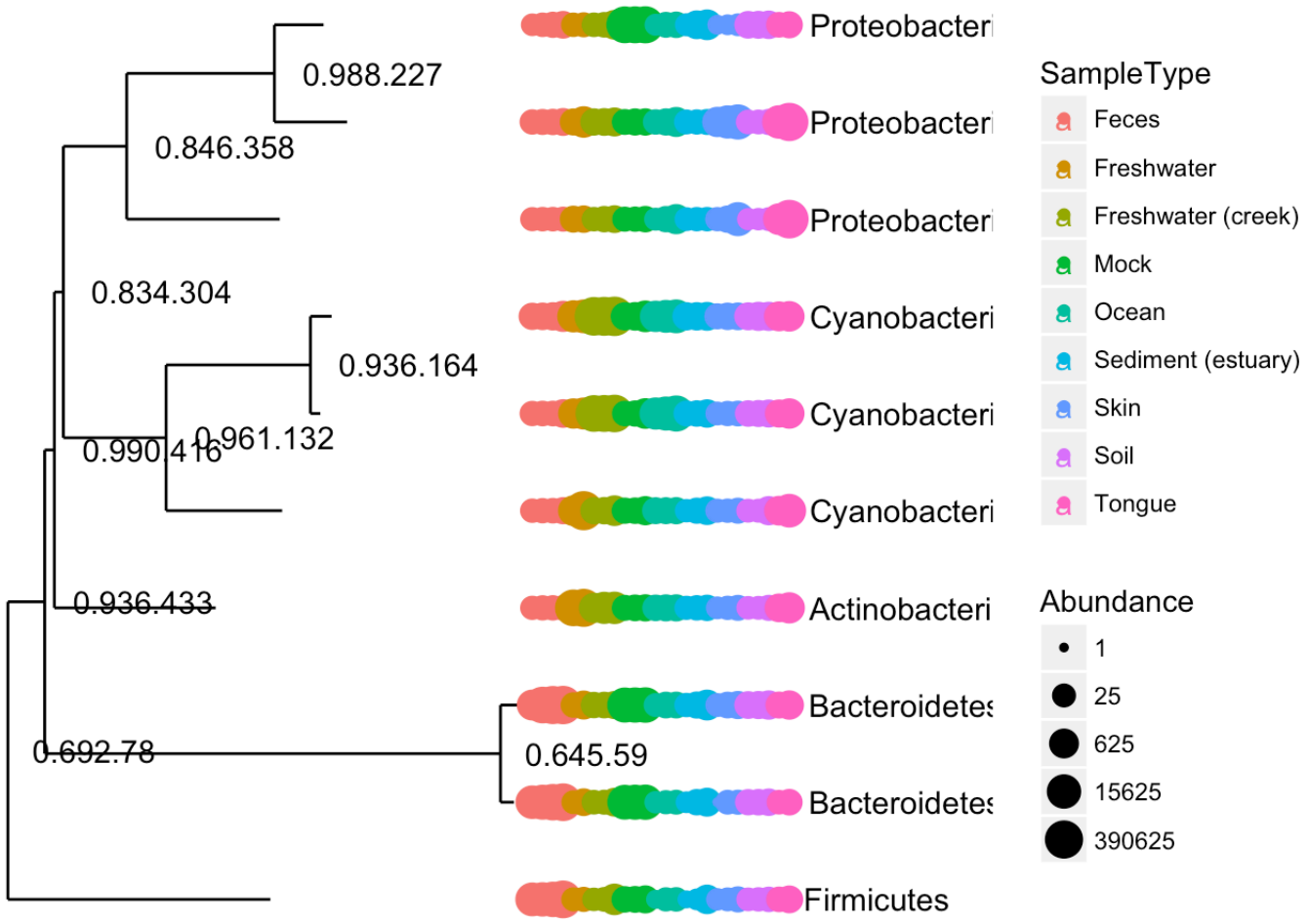
```
taxa_names(GlobalPatterns)[1:10]
```

```
##  [1] "549322" "522457" "951"    "244423" "586076" "246140" "143239"
##  [8] "244960" "255340" "144887"
```

```
myTaxa <- names(sort(taxa_sums(GlobalPatterns), decreasing = TRUE)[1:10])
ex1 <- prune_taxa(myTaxa, GlobalPatterns)
plot(phy_tree(ex1), show.node.label = TRUE)
```




```
plot_tree(ex1, color = "SampleType", label.tips = "Phylum", ladderize = "left", justi
fy = "left", size = "Abundance")
```



Preprocessing

`phyloseq` also includes functions for filtering, subsetting, and merging abundance data. Let's filter!

Filtering

Filtering is designed in a modular fashion. The functions `prune_taxa` and `prune_samples` will directly remove unwanted indices. The functions `filterfun_sample` and `genefilter_sample` will allow you to build arbitrarily complex sample-wise filtering criteria. The function `filter_taxa` allows for taxa-wise filtering.

In the following example we will use the GlobalPatterns object again, first transforming it to relative abundance in a new `GPr` object, then filtering that object to remove all OTUs that have a mean abundance $< 10^-5$, creating another new object.

```
# Create the object with the relative abundance data

GPr <- transform_sample_counts(GlobalPatterns, function(x) x / sum(x))
GPfr <- filter_taxa(GPr, function(x) mean(x) > 1e-5, TRUE)

GlobalPatterns
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

```
GPr
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

```
GPfr
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 4624 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 4624 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 4624 tips and 4623 internal nodes ]
```

Subsetting

We see that the final object, `GPfr` is highly subsetted containing just a little less than a quarter of the original OTUs (4624 / 19216).

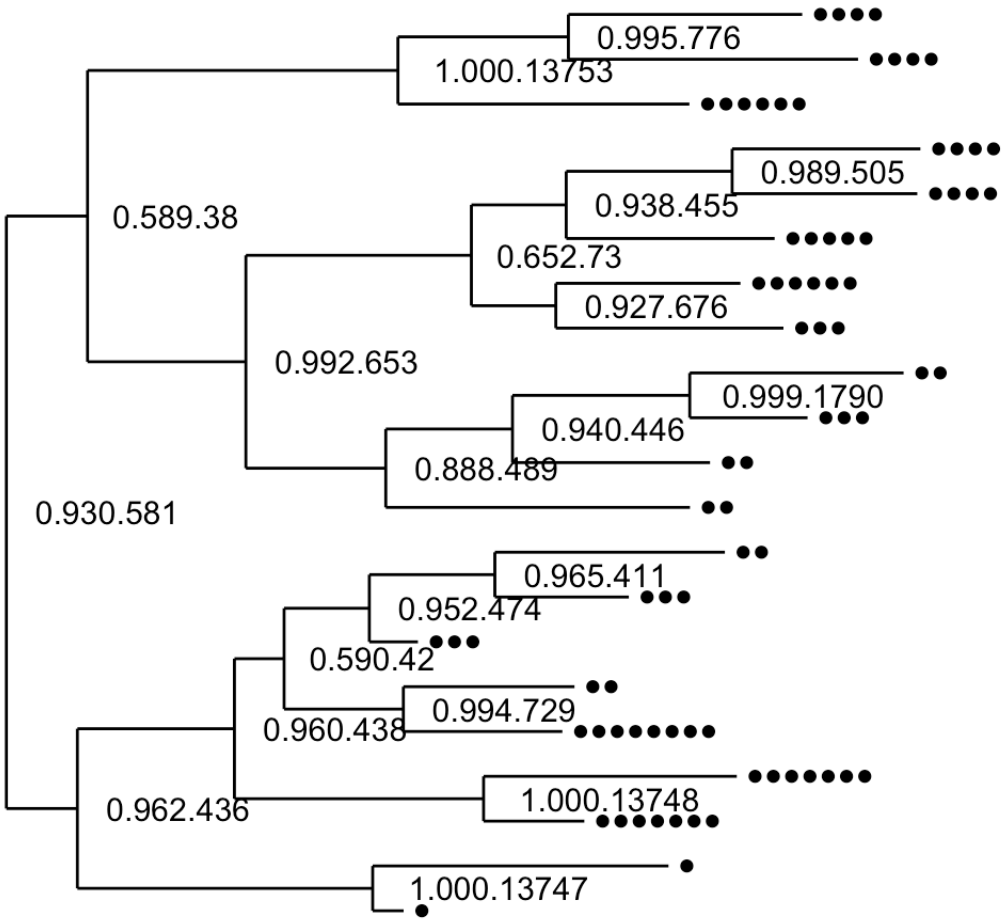
The functions `prune_sample` and `prune_taxa` are to be used in cases where the complete subset of OTUS or samples is directly available. Alternatively the `subsets_taxa` and `subset_samples` are best for subsetting based on other data contained in the taxonomy and sample datasets respectively.

As an example we will obtain the subset of the GlobalPatterns dataset that is part of the Chlamydiae phylum, then remove samples with less than 20 reads.

```
# Subset for Chlamydiae

GP.ch1 <- subset_taxa(GlobalPatterns, Phylum == "Chlamydiae")

# Eliminate samples whose total reads are less than 20

GP.ch1 <- prune_samples(sample_sums(GP.ch1) >= 20, GP.ch1)

GP.ch1
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 21 taxa and 9 samples ]
## sample_data() Sample Data:       [ 9 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 21 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 21 tips and 20 internal nodes ]
```

```
plot_tree(GP.ch1)
```



Merging

Merging methods include `merge_taxa` and `merge_samples` for merging specific OTUs or samples, respectively.

The following chunk shows the merge of the first 5 OTUs in teh Chlamydiae-only dataset.

```
# Merge first 5 OTUSs in Chlamydiae-only dataset

GP.ch1.merged <- merge_taxa(GP.ch1, taxa_names(GP.ch1)[1:5])
```

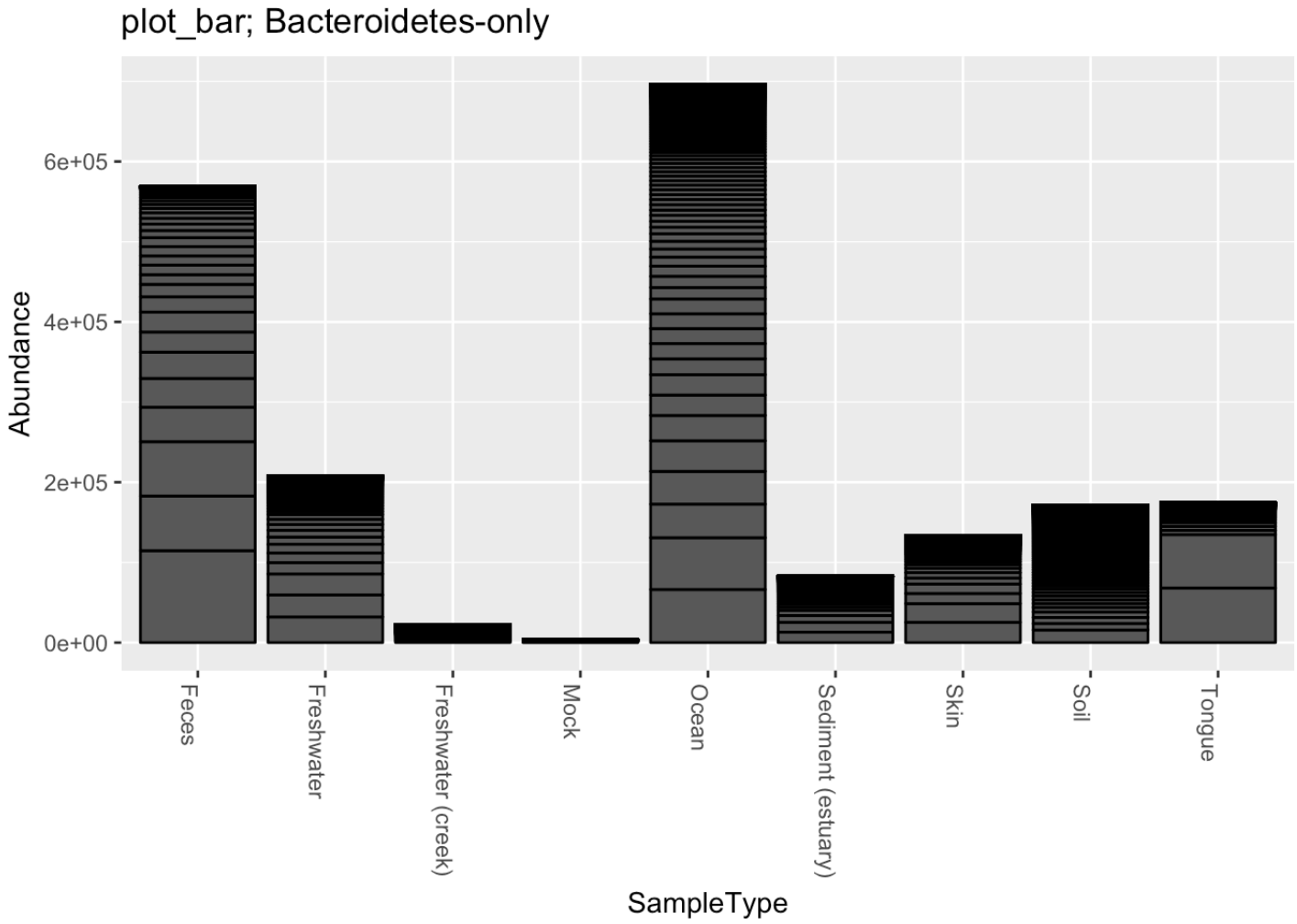
Let's do one final set of preprocessing steps

```
# Final preprocess for GlobalPatterns: filter out taxa not see more than 3 times in a
t least
# 20% of samples

GP <- filter_taxa(GlobalPatterns, function(x) sum(x > 3) > (0.2*length(x)), TRUE)

# Define human v non-human and add to sample data:

sample_data(GP)$human <- factor(get_variable(GP, "SampleType") %in% c("Feces", "Mock"
, "Skin", "Tongue"))

# Standardize abundances to median sequencing depth

total <- median(sample_sums(GP))
standf <- function(x, t=total) round(t * (x / sum(x)))
gps <- transform_sample_counts(GP, standf)

# Filter out taxa with CV > 3.0

gpsf <- filter_taxa(gps, function(x) sd(x) / mean(x) > 3.0, TRUE)

# Subset to Bacteroidetes

gpsfb <- subset_taxa(gpsf, Phylum == "Bacteroidetes")
```

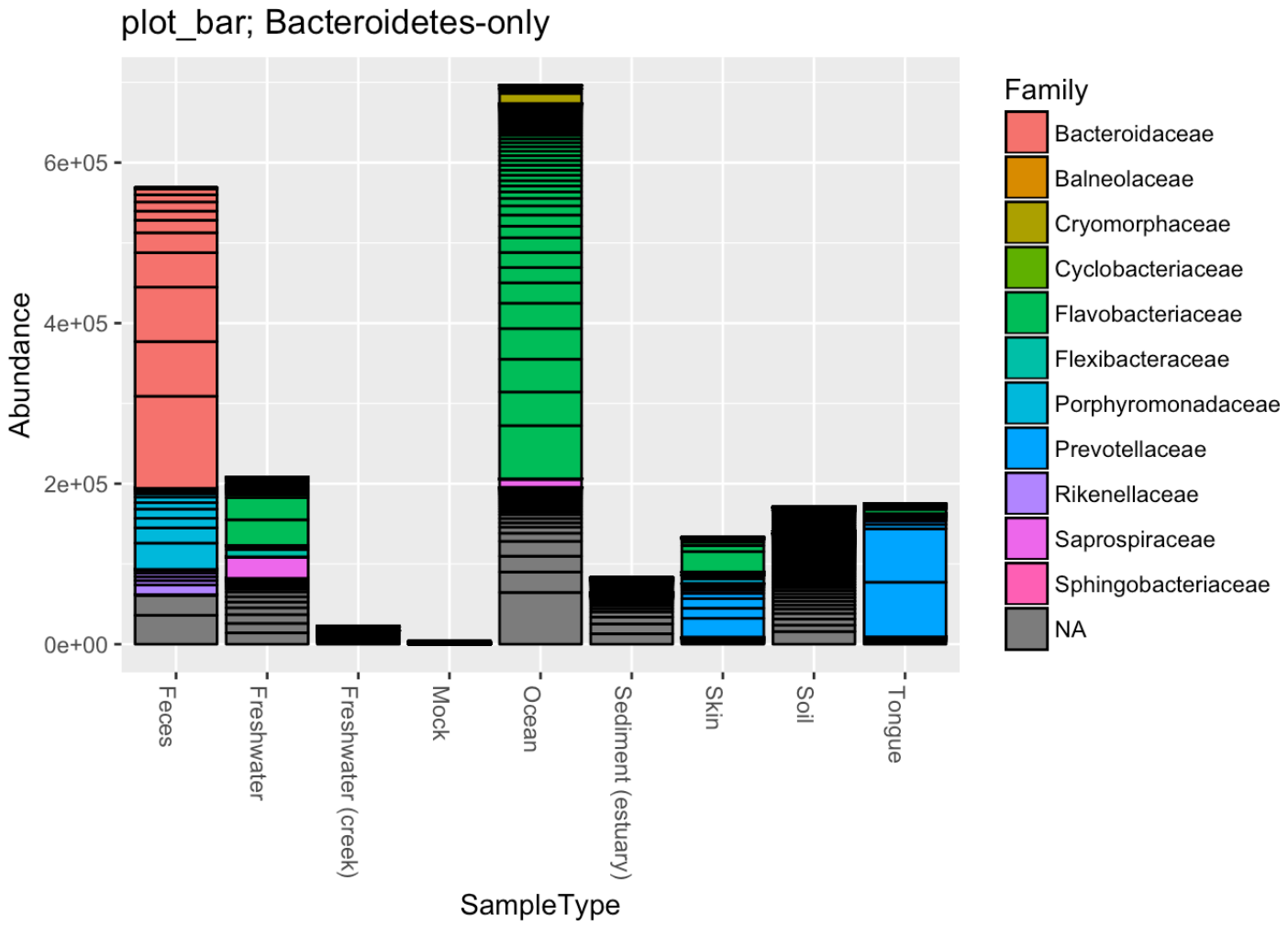
Graphical Display

Let's graph this slice of data

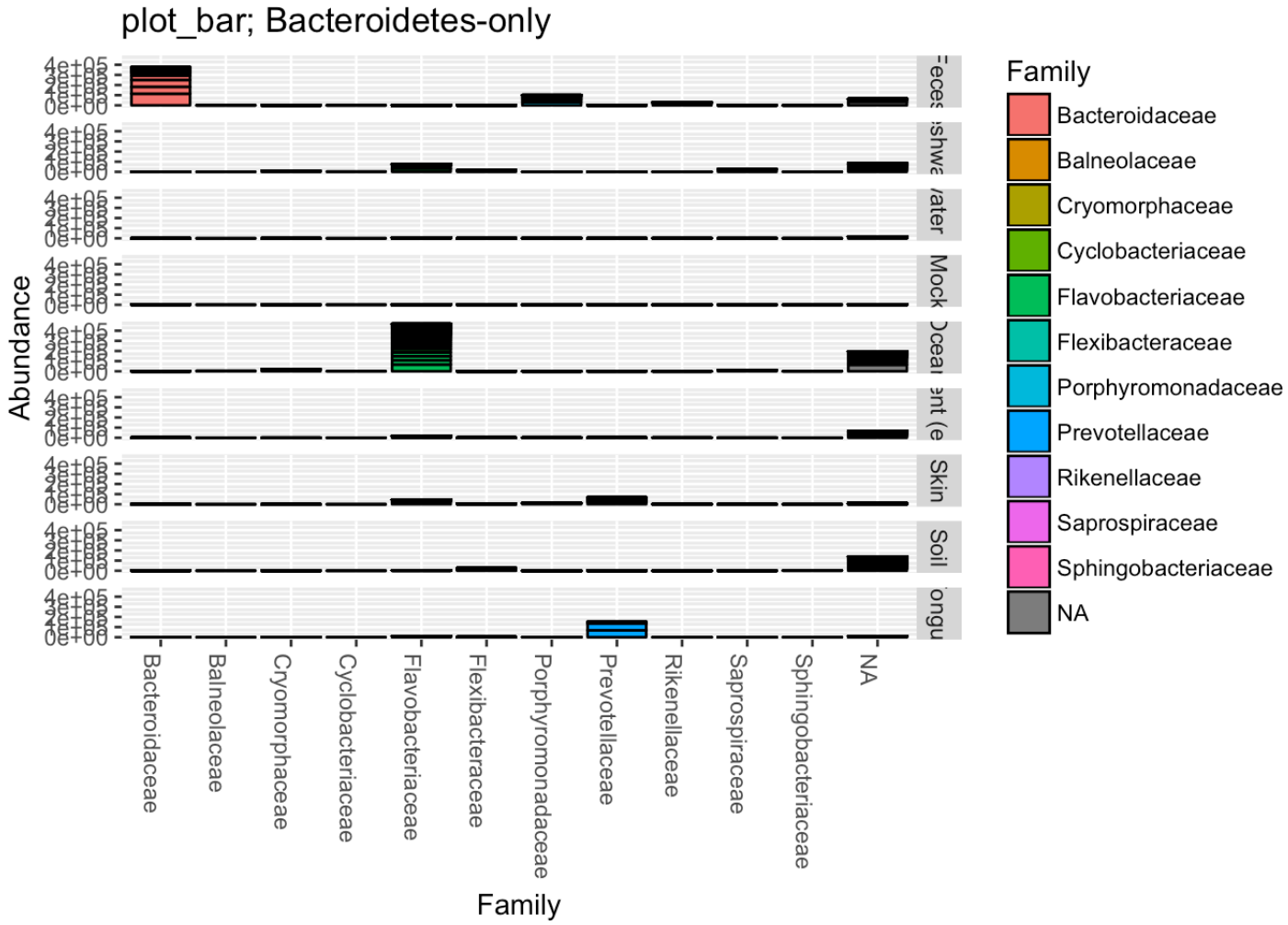
```
# Graph the slice of data

# First a simple bar graph

title = "plot_bar; Bacteroidetes-only"
plot_bar(gpsfb, "SampleType", "Abundance", title = title)
```




```
# Now colored by family

plot_bar(gpsfb, "SampleType", "Abundance", "Family", title = title)
```

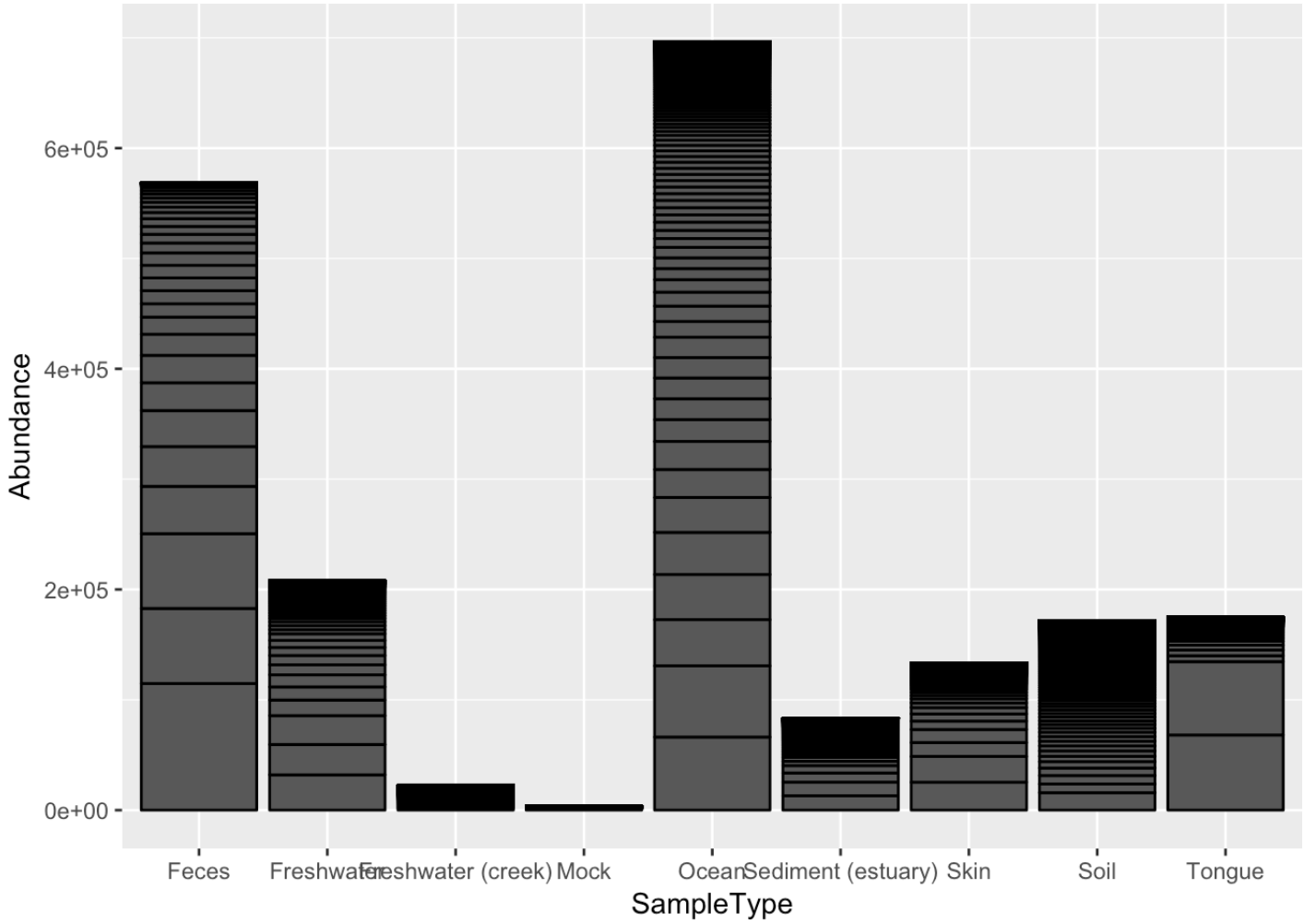



```
# Now faceted with type of samples

plot_bar(gpsfb, "Family", "Abundance", "Family", title = title, facet_grid = "SampleT
ype~.")
```

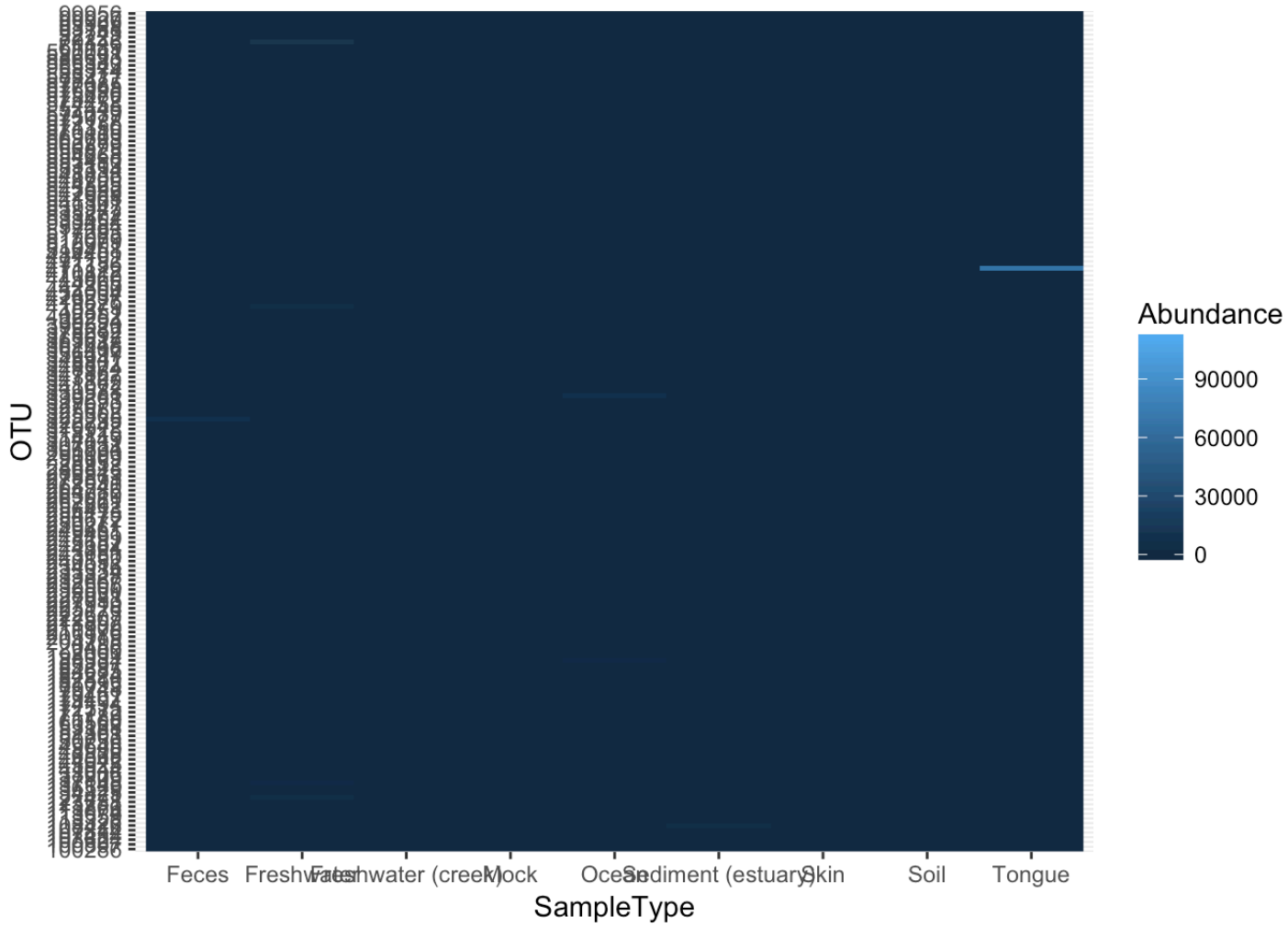


You can also define your own graphics "from scratch" using `psmelt` Here are two examples:

```
# Example 1
mdf <- psmelt(gpsfb)
# Simple bar plot

ggplot(mdf, aes(x=SampleType,
                y=Abundance)) +
  geom_bar(stat="identity", position = "stack", color="black")
```




```
# Simple heat map

ggplot(mdf, aes(x=SampleType,
                y=OTU,
                fill=Abundance)) +
  geom_raster()
```



Distance Functions

The `distance` function takes a `phyloseq` object and a method and returns a `dist` -class distance object. Currently `phyloseq` supports 44 different method options. For the complete list, type `distanceMethodList` . Use the documentation on `distance` for further details.

The usage is as follows:

distance(phyloseq-object, method="methodname", type="type")

Because this function organizes distance calculations into one function, it is easy to calculate all distance methods and investigate the results. We are going to do so on the `Enterotypes` dataset, then perform multidimensional scaling, plot the first two axes, shading and shaping the points in each plot according to sequencing technology and assigned "Enterotype" label.

First some preliminaries, loading `plyr` , setting the `ggplot2` theme, and loading the dataset:

```
# get plyr
library(plyr)

# set ggplot2 theme
theme_set(theme_bw())

# load enterotype data
data(enterotype)
```

Some preliminary filtering to remove OTUs that have unassigned sequences ("-1"):

```
# Remove unassigned OTUs

enterotype <- subset_taxa(enterotype, Genus != "-1")
```

Let's see the available distance methods coded in `distance`:

```
dist_methods <- unlist(distanceMethodList)
print(dist_methods)
```

```
##     UniFrac1     UniFrac2        DPCoA          JSD     vegdist1
##    "unifrac"   "wunifrac"      "dpcoa"        "jsd"  "manhattan"
##     vegdist2     vegdist3     vegdist4     vegdist5     vegdist6
##  "euclidean"   "canberra"       "bray" "kulczynski"    "jaccard"
##     vegdist7     vegdist8     vegdist9    vegdist10    vegdist11
##      "gower"   "altGower"   "morisita"       "horn"  "mountford"
##    vegdist12    vegdist13    vegdist14    vegdist15   betadiver1
##       "raup"   "binomial"       "chao"        "cao"          "w"
##   betadiver2   betadiver3   betadiver4   betadiver5   betadiver6
##         "-1"          "c"         "wb"          "r"          "I"
##   betadiver7   betadiver8   betadiver9  betadiver10  betadiver11
##          "e"          "t"         "me"          "j"        "sor"
##  betadiver12  betadiver13  betadiver14  betadiver15  betadiver16
##          "m"         "-2"         "co"         "cc"          "g"
##  betadiver17  betadiver18  betadiver19  betadiver20  betadiver21
##         "-3"          "l"         "19"         "hk"        "rlb"
##  betadiver22  betadiver23  betadiver24        dist1        dist2
##        "sim"         "gl"          "z"    "maximum"     "binary"
##        dist3   designdist
##  "minkowski"        "ANY"
```

Remove distance methods that depend on a phylogenetic tree, because the `enterotype` object that we started with does not have one. These would be the two unifrac methods and DPCoA.

```
# Remove methods that need a phylogenetic tree

dist_methods[(1:3)]
```

```
##   UniFrac1   UniFrac2      DPCoA
##  "unifrac" "wunifrac"    "dpcoa"
```

```
dist_methods <- dist_methods[-(1:3)]
print(dist_methods)
```

```
##          JSD     vegdist1     vegdist2     vegdist3     vegdist4
##        "jsd"  "manhattan"  "euclidean"   "canberra"       "bray"
##     vegdist5     vegdist6     vegdist7     vegdist8     vegdist9
## "kulczynski"    "jaccard"      "gower"   "altGower"   "morisita"
##    vegdist10    vegdist11    vegdist12    vegdist13    vegdist14
##       "horn"  "mountford"       "raup"   "binomial"       "chao"
##    vegdist15   betadiver1   betadiver2   betadiver3   betadiver4
##        "cao"          "w"         "-1"          "c"         "wb"
##   betadiver5   betadiver6   betadiver7   betadiver8   betadiver9
##          "r"          "I"          "e"          "t"         "me"
##  betadiver10  betadiver11  betadiver12  betadiver13  betadiver14
##          "j"        "sor"          "m"         "-2"         "co"
##  betadiver15  betadiver16  betadiver17  betadiver18  betadiver19
##         "cc"          "g"         "-3"          "l"         "19"
##  betadiver20  betadiver21  betadiver22  betadiver23  betadiver24
##         "hk"        "rlb"        "sim"         "gl"          "z"
##        dist1        dist2        dist3   designdist
##    "maximum"     "binary"  "minkowski"        "ANY"
```

```
# This is the unser-defined method:

dist_methods["designdist"]
```

```
## designdist
##      "ANY"
```

```
# Remove it

dist_methods <- dist_methods[-which(dist_methods == "ANY")]
print(dist_methods)
```

```
##          JSD     vegdist1     vegdist2     vegdist3     vegdist4
##        "jsd"  "manhattan"  "euclidean"   "canberra"       "bray"
##     vegdist5     vegdist6     vegdist7     vegdist8     vegdist9
## "kulczynski"    "jaccard"      "gower"   "altGower"   "morisita"
##    vegdist10    vegdist11    vegdist12    vegdist13    vegdist14
##       "horn"  "mountford"       "raup"   "binomial"       "chao"
##    vegdist15   betadiver1   betadiver2   betadiver3   betadiver4
##        "cao"          "w"         "-1"          "c"         "wb"
##   betadiver5   betadiver6   betadiver7   betadiver8   betadiver9
##          "r"          "I"          "e"          "t"         "me"
##  betadiver10  betadiver11  betadiver12  betadiver13  betadiver14
##          "j"        "sor"          "m"         "-2"         "co"
##  betadiver15  betadiver16  betadiver17  betadiver18  betadiver19
##         "cc"          "g"         "-3"          "l"         "19"
##  betadiver20  betadiver21  betadiver22  betadiver23  betadiver24
##         "hk"        "rlb"        "sim"         "gl"          "z"
##        dist1        dist2        dist3
##    "maximum"     "binary"  "minkowski"
```

Now we can loop through this list, calculate distances according to each methods, plot the distances, and save each plot to a plist (`plist`).

```
# Prepare to receive plots
plist <- vector("list", length(dist_methods))
names(plist) <- dist_methods

for(i in dist_methods ){
  # Calculate distance matrix
  iDist <- distance(enterotype, method=i)
  #Calculate ordination
  iMDS <- ordinate(enterotype, "MDS", distance = iDist)
  ## Make plot
  # Don't carry over previous plot (if errror, p will be blank)
  p <- NULL
  # Create plot, store as temp variable p
  p <- plot_ordination(enterotype, iMDS, color="SeqTech", shape="Enterotype")
  # Add title to each plot
  p <- p + ggtitle(paste("MDS using distance method ", i, sep=""))
  # Save the graphic to file
  plist[[i]] <- p
}
```

```
## Warning in vegdist(structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, :
## results may be meaningless with non-integer data in method "morisita"
```

```
## Warning in vegdist(structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, :
## results may be meaningless with non-integer data in method "chao"
```

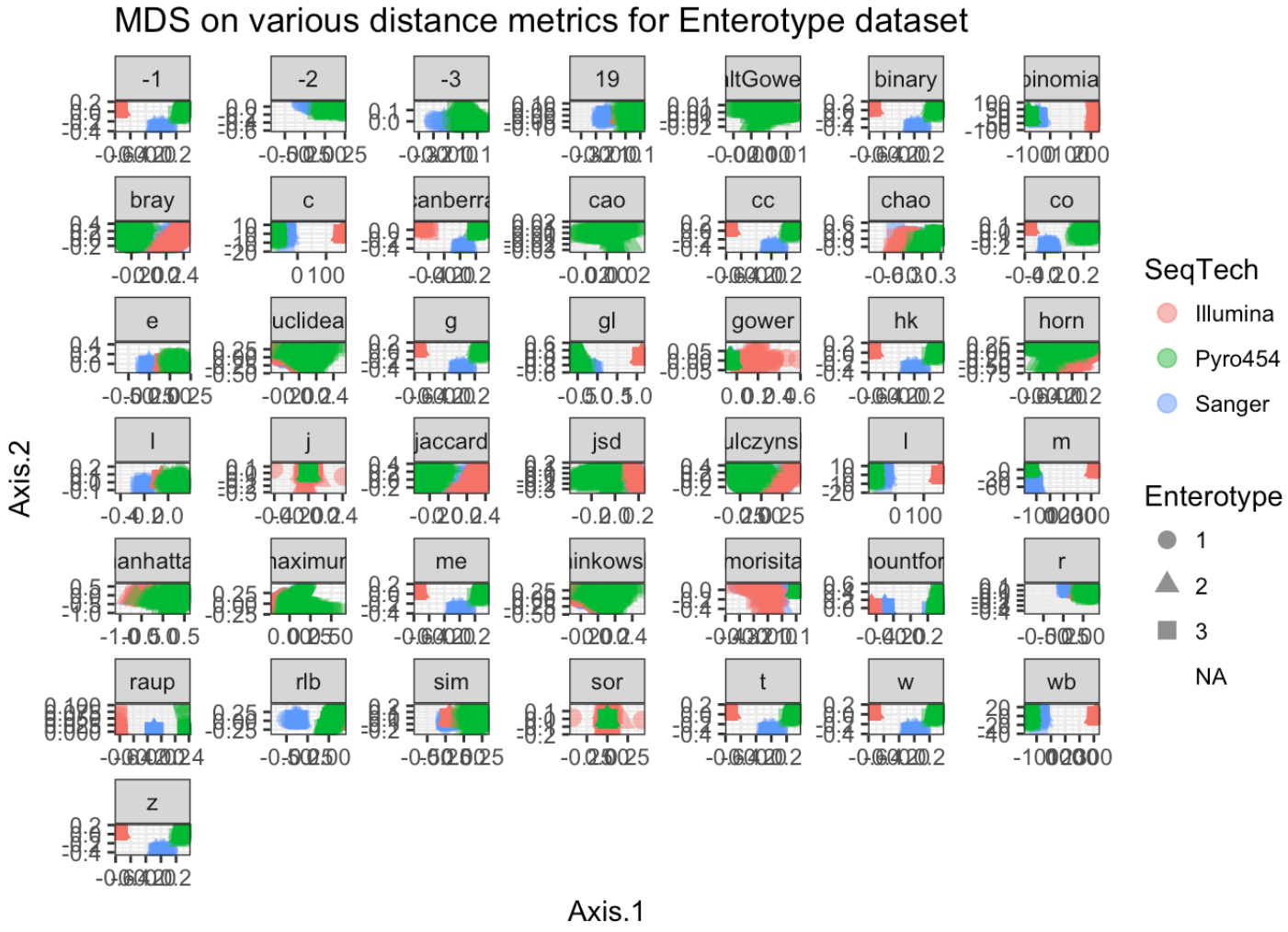
```
## Warning in vegdist(structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, :
## results may be meaningless with non-integer data in method "cao"
```

Combine Results

Shade according to sequencing technology:

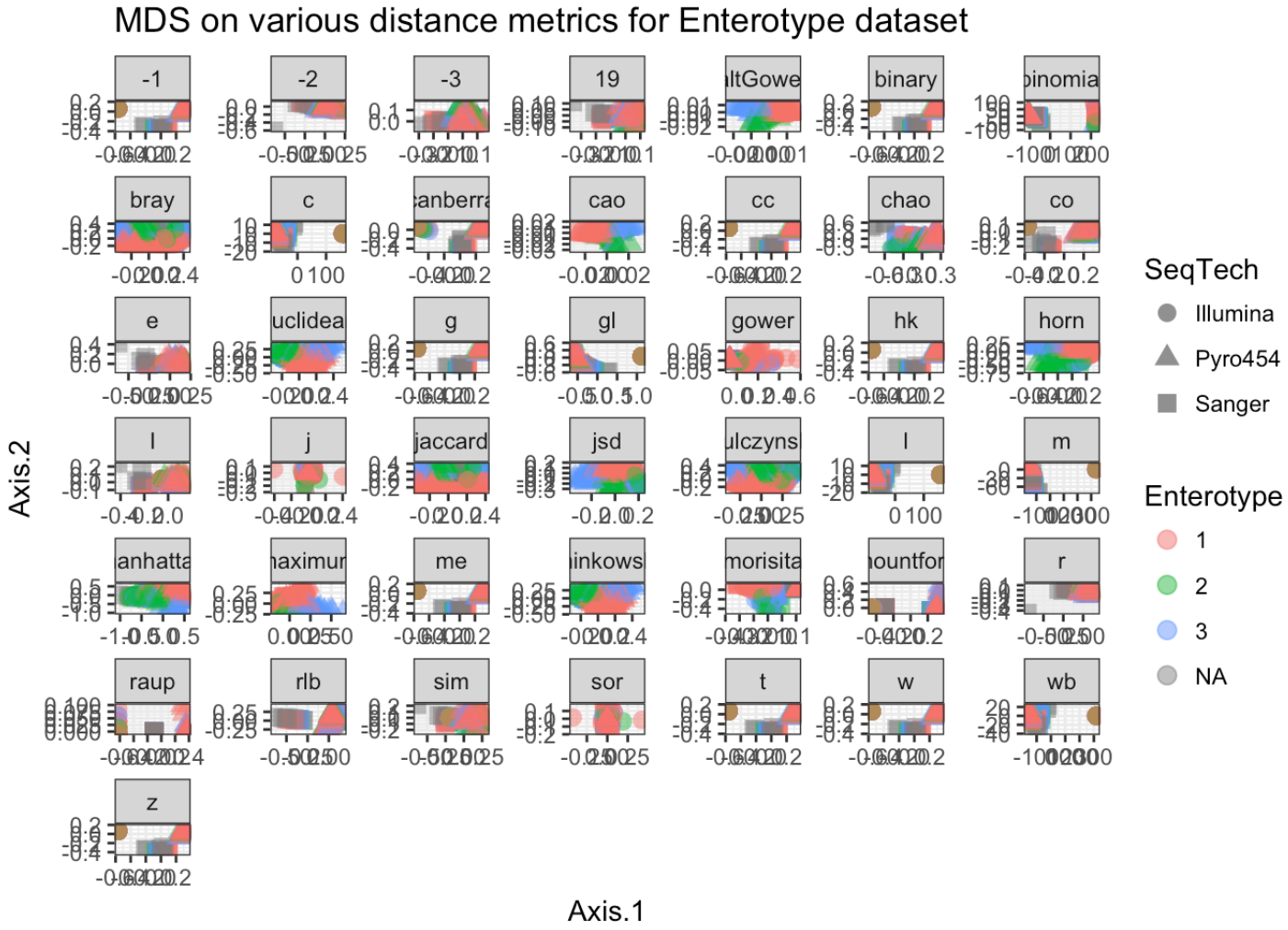
```
# Shade according to sequencing technology:

df <- ldply(plist, function(x) x$data)
names(df)[1] <- "distance"
p <- ggplot(df, aes(Axis.1, Axis.2, color=SeqTech, shape=Enterotype))
p <- p + geom_point(size = 3, alpha = 0.5)
p <- p + facet_wrap(~distance, scales="free")
p <- p + ggtitle("MDS on various distance metrics for Enterotype dataset")
p
```

```
## Warning: Removed 387 rows containing missing values (geom_point).
```



Now, shade instead according to assigned Enterotype

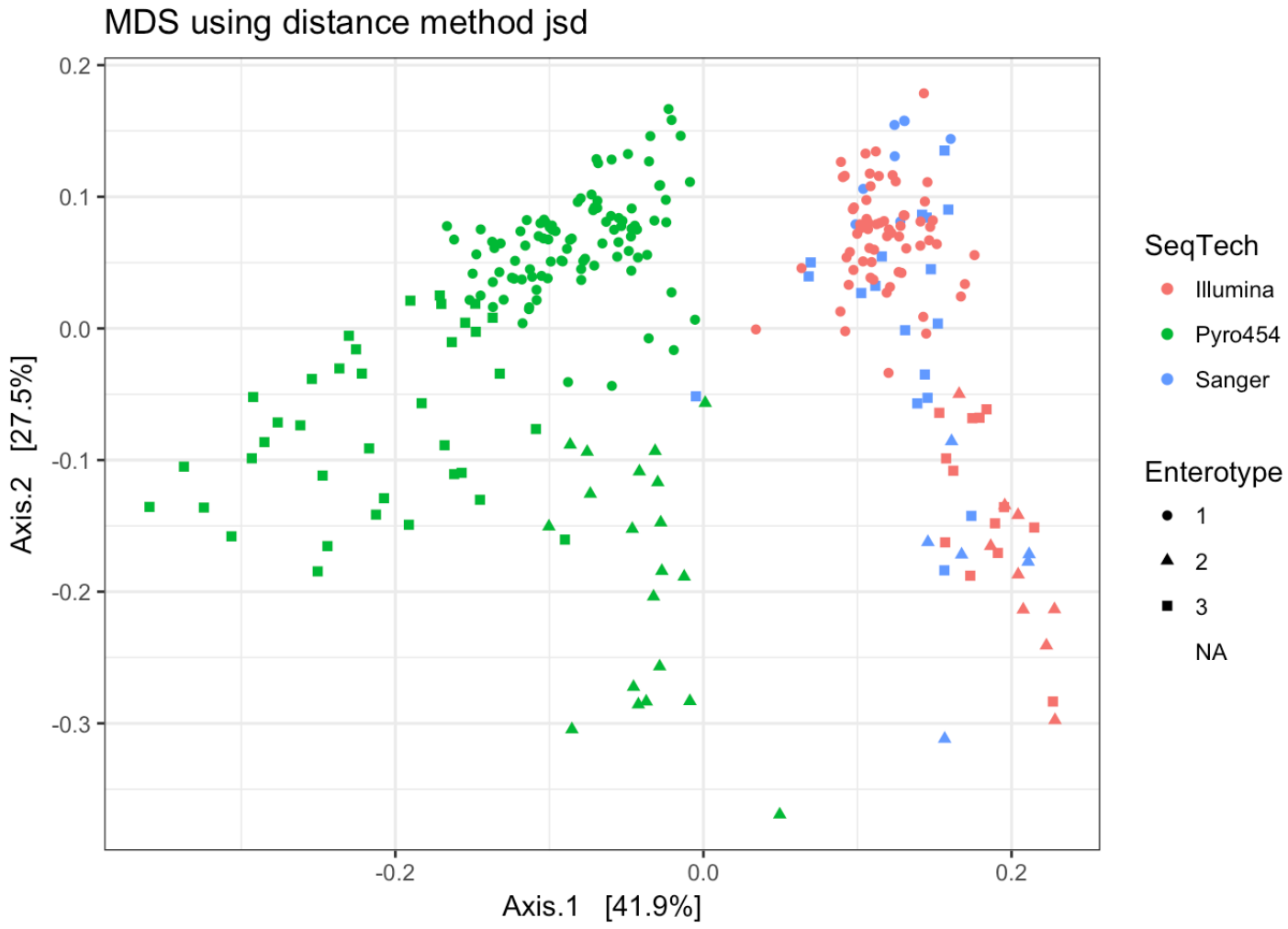
```
# Shade according to Enterotype:

df <- ldply(plist, function(x) x$data)
names(df)[1] <- "distance"
p <- ggplot(df, aes(Axis.1, Axis.2, color=Enterotype, shape=SeqTech))
p <- p + geom_point(size = 3, alpha = 0.5)
p <- p + facet_wrap(~distance, scales="free")
p <- p + ggtitle("MDS on various distance metrics for Enterotype dataset")
p
```



Compare Results

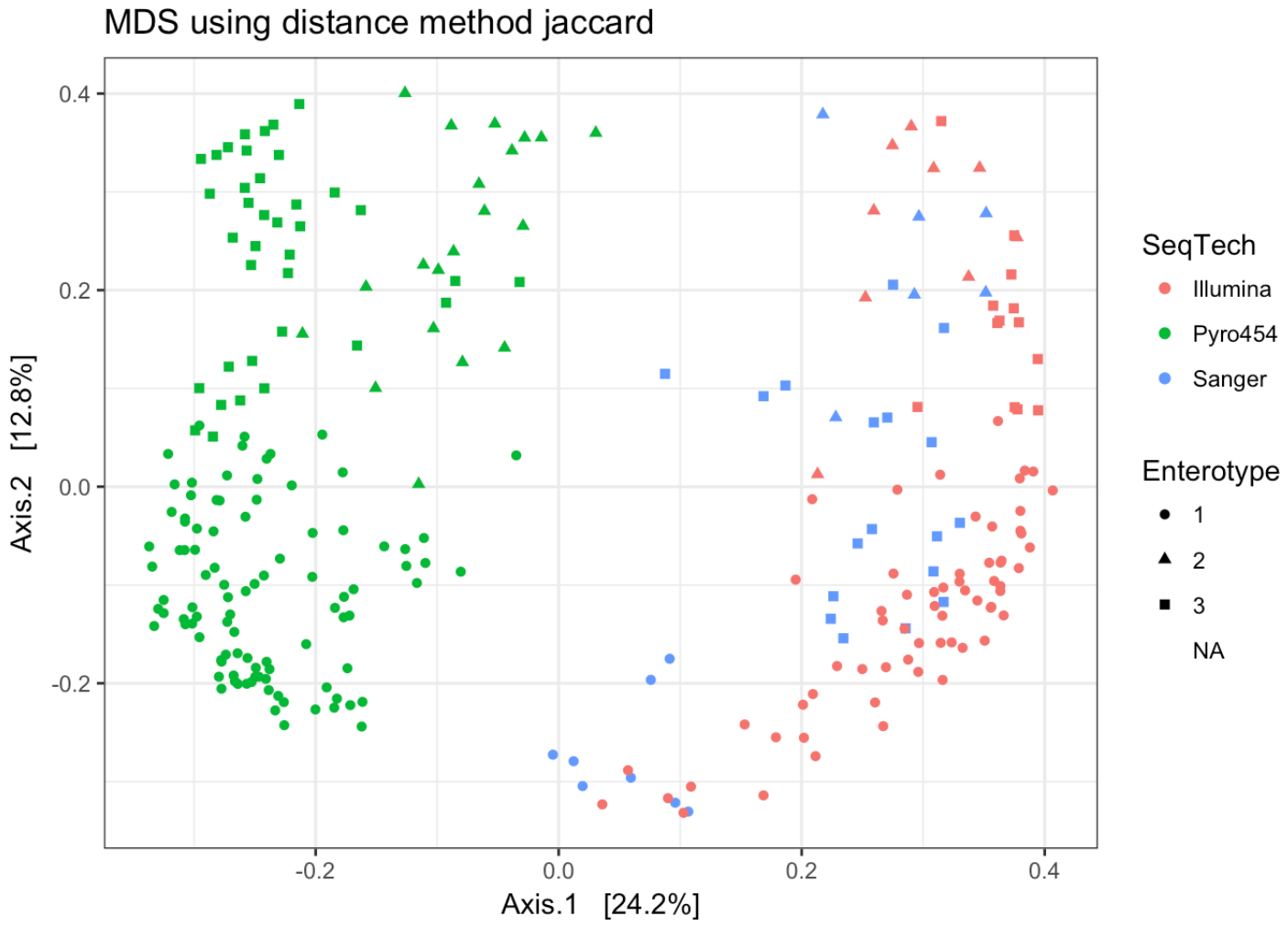
Some selected examples among the created plots. First the Jensen-Shannon Divergence.

```
# Plot ordination with Jensen-Shannon Divergence

print(plist[["jsd"]])
```



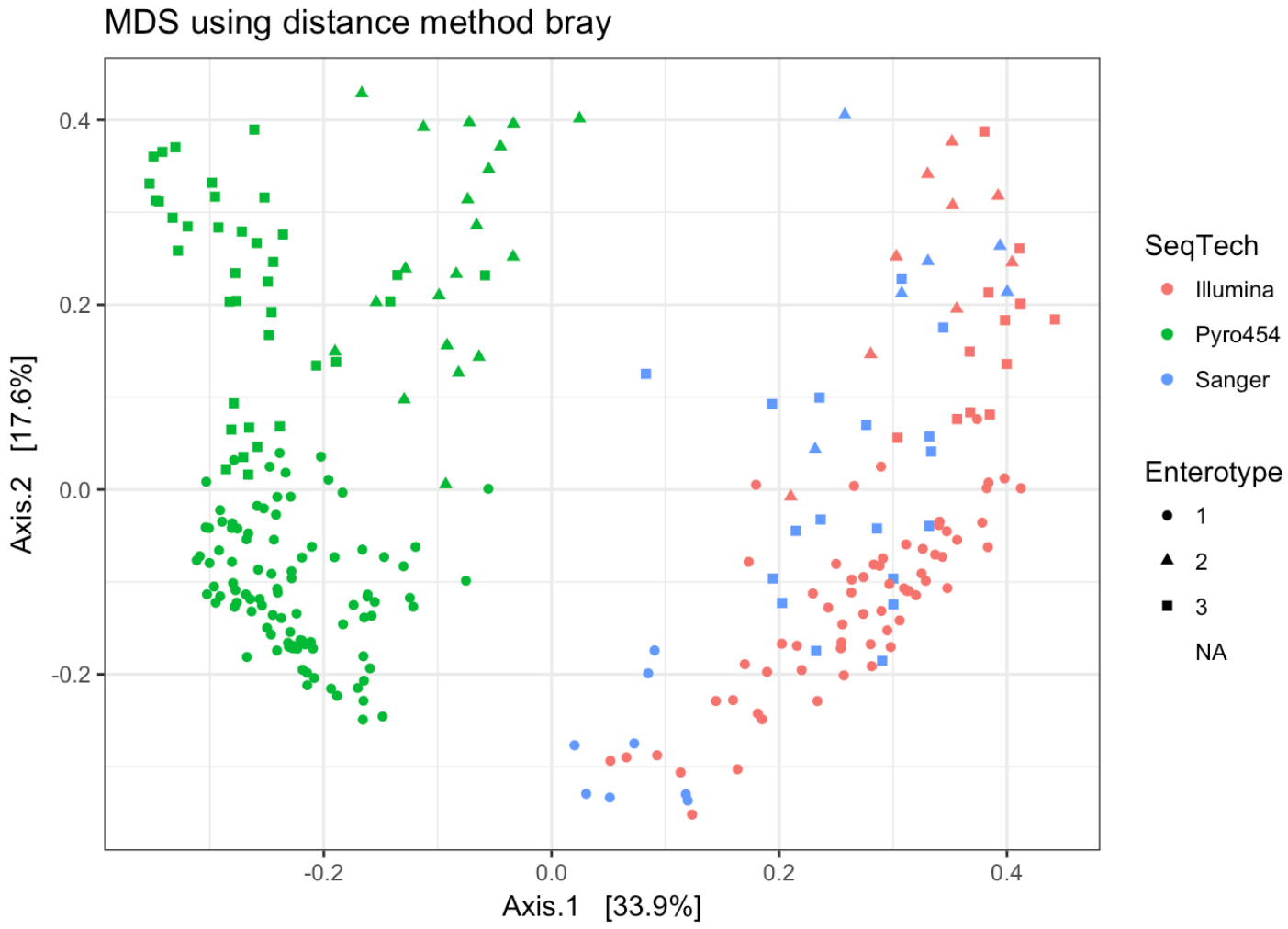
Jaccard distance

```
# Plot ordination with Jaccard distance

print(plist[["jaccard"]])
```



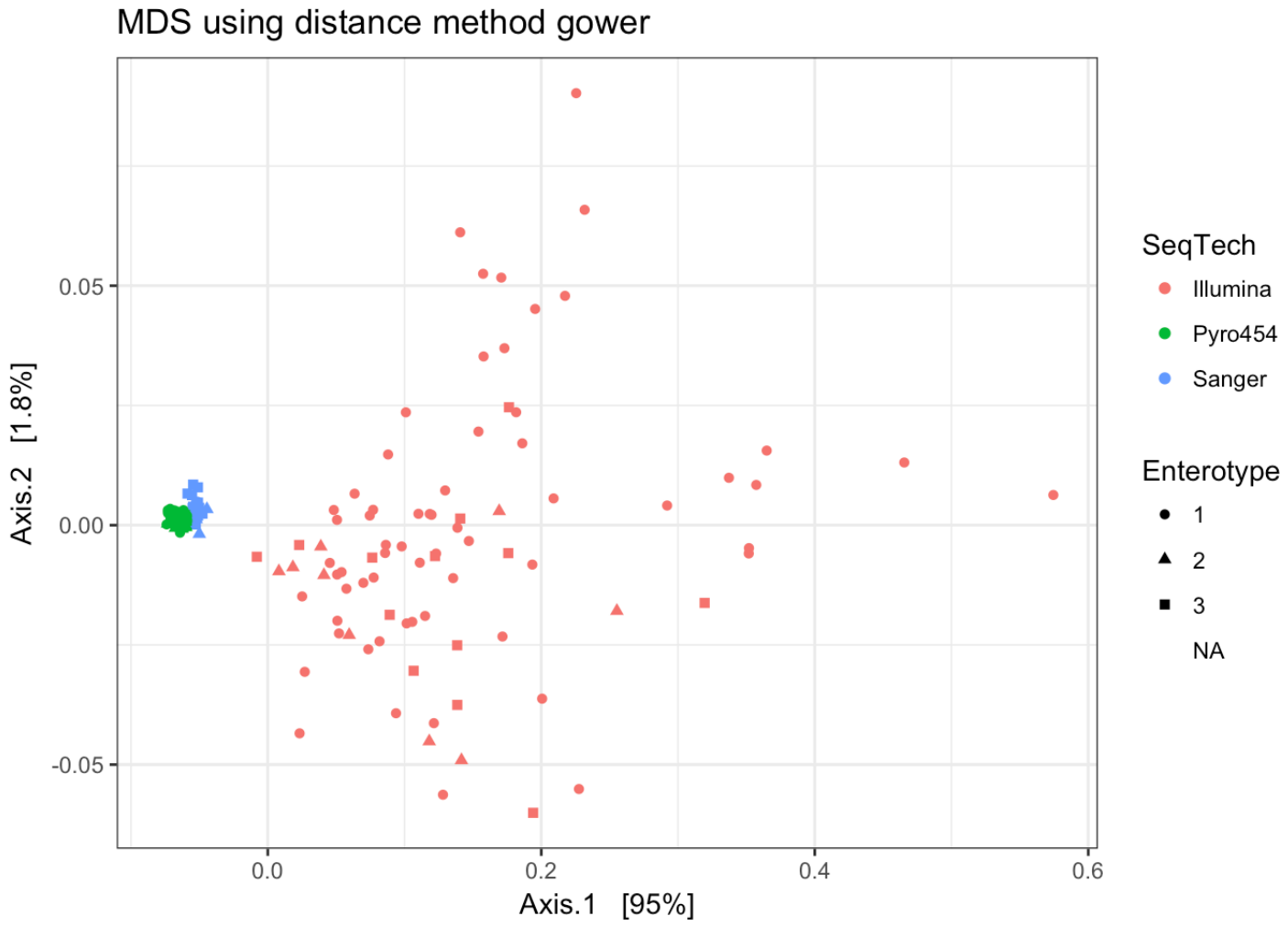
Bray-Curtis

```
# Plot ordination with Bray-Curtis

print(plist[["bray"]])
```



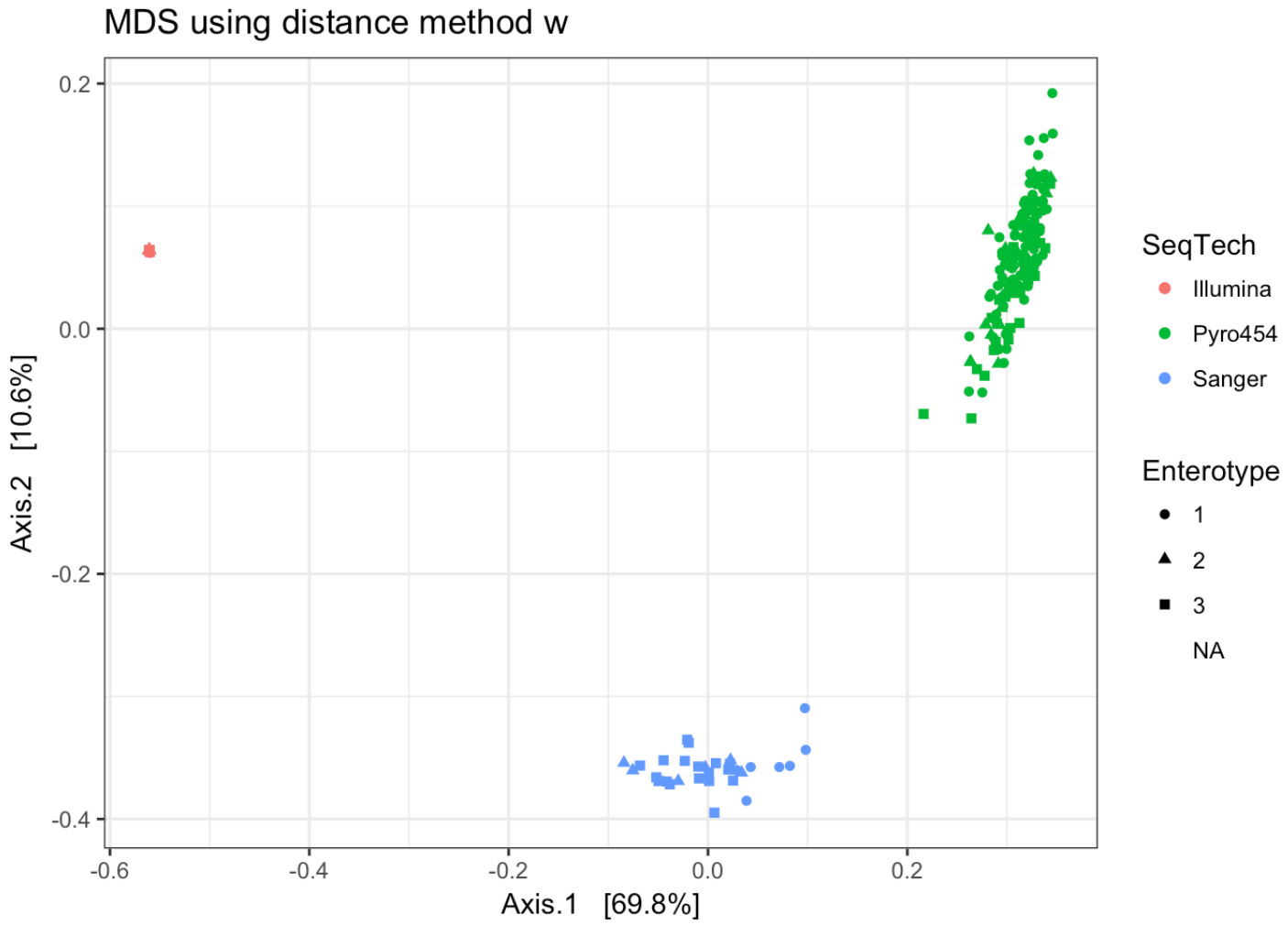
Gower

```
# Plot ordination with Gower

print(plist[["gower"]])
```



w

```
# Plot ordination with w

print(plist[["w"]])
```



Gap Statistic

How Many Clusters Are There?

The `clusGap` function from the `cluster` package calculates a goodness of clustering measure, the "gap" statistic. For each number of clusters `k`, it compares W(k) with $E[W(k)]$ where the latter is defined by bootstrapping.

Let's give this a try:

First: Ordination

Well, really, first, let's set things up, then ordinate:

```
# Set up

library(cluster)
packageVersion("cluster")
```

```
## [1] '2.0.6'
```

```
theme_set(theme_bw())
data(enterotype)

# Now ordinate
exord <- ordinate(enterotype, method = "MDS", distance = "jsd")
```

Compute Gap Statistic

```
# Compute gap statistic

paml = function(x, k){list(cluster=pam(x,k, cluster.only = TRUE))}
x = phyloseq:::scores.pcoa(exord, distplay="sites")
gskmn = clusGap(x[,1:2], FUN=paml, K.max = 6, B=50)
gskmn
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = x[, 1:2], FUNcluster = paml, K.max = 6, B = 50)
## B=50 simulated reference sets, k = 1..6; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 4
##          logW   E.logW       gap     SE.sim
## [1,] 2.995599 3.112782 0.1171833 0.02022403
## [2,] 2.209852 2.771530 0.5616781 0.02149033
## [3,] 1.922188 2.586401 0.6642123 0.02361989
## [4,] 1.685798 2.415390 0.7295919 0.02788847
## [5,] 1.601025 2.282325 0.6813004 0.01895680
## [6,] 1.480640 2.180412 0.6997716 0.02749864
```

Plot Results

Define a plot method for results

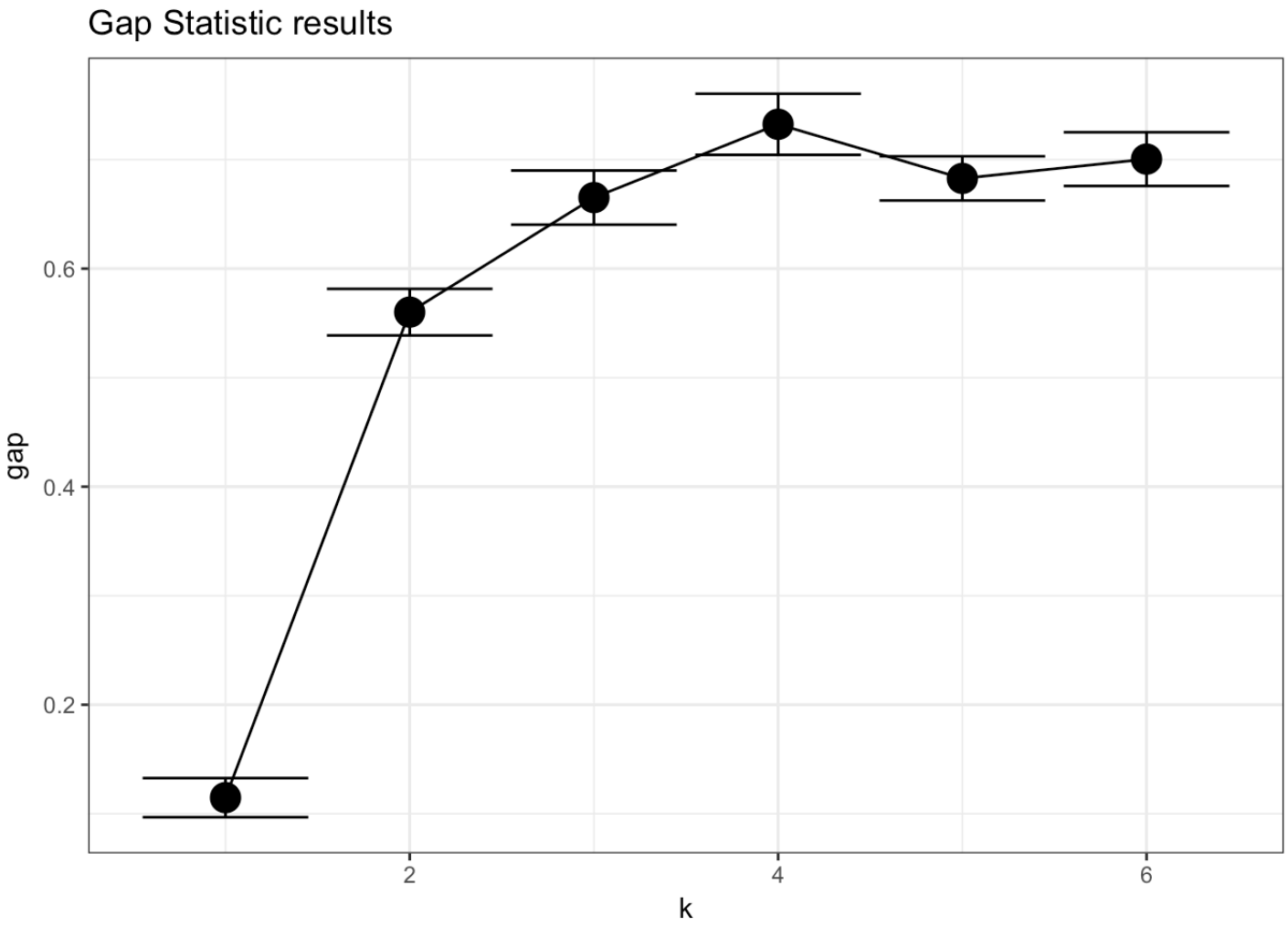
```
# Define plot method

plot_glusgap <- function(clusgap, title="Gap Statistic calculation results"){
  require("ggplot2")
  gstab <- data.frame(clusgap$Tab, k=1:nrow(clusgap$Tab))
  p <- ggplot(gstab, aes(k, gap)) + geom_line() + geom_point(size=5)
  p <- p + geom_errorbar(aes(ymax = gap+SE.sim, ymin = gap - SE.sim))
  p <- p + ggtitle(title)
  return(p)
}

# Define a wrapper function

gap_statistic_ordination <- function(ord, FUNcluster, type="sites", K.max=6, axes=c(1
:2), B=500, verbose=interactive(), ...){
    require("cluster")
    #   If "paml" was chosen, use this internally defined call to pam
    if(FUNcluster == "paml"){
        FUNcluster = function(x,k) list(cluster = pam(x, k, cluster.only=TRUE))
    }
    # Use the scores function to get the ordination coordinates
    x = phyloseq:::scores.pcoa(ord, display=type)
    #   If axes not explicitly defined (NULL), then use all of them
    if(is.null(axes)){axes = 1:ncol(x)}
    #   Finally, perform, and return, the gap statistic calculation using cluster::cl
usGap
    clusGap(x[, axes], FUN=FUNcluster, K.max=K.max, B=B, verbose=verbose, ...)
}
```

Now try out this function

```
# Make the plot

gs <- gap_statistic_ordination(exord, "paml", B=50, verbose = FALSE)
print(gs, method="Tibs2001SEmax")
```

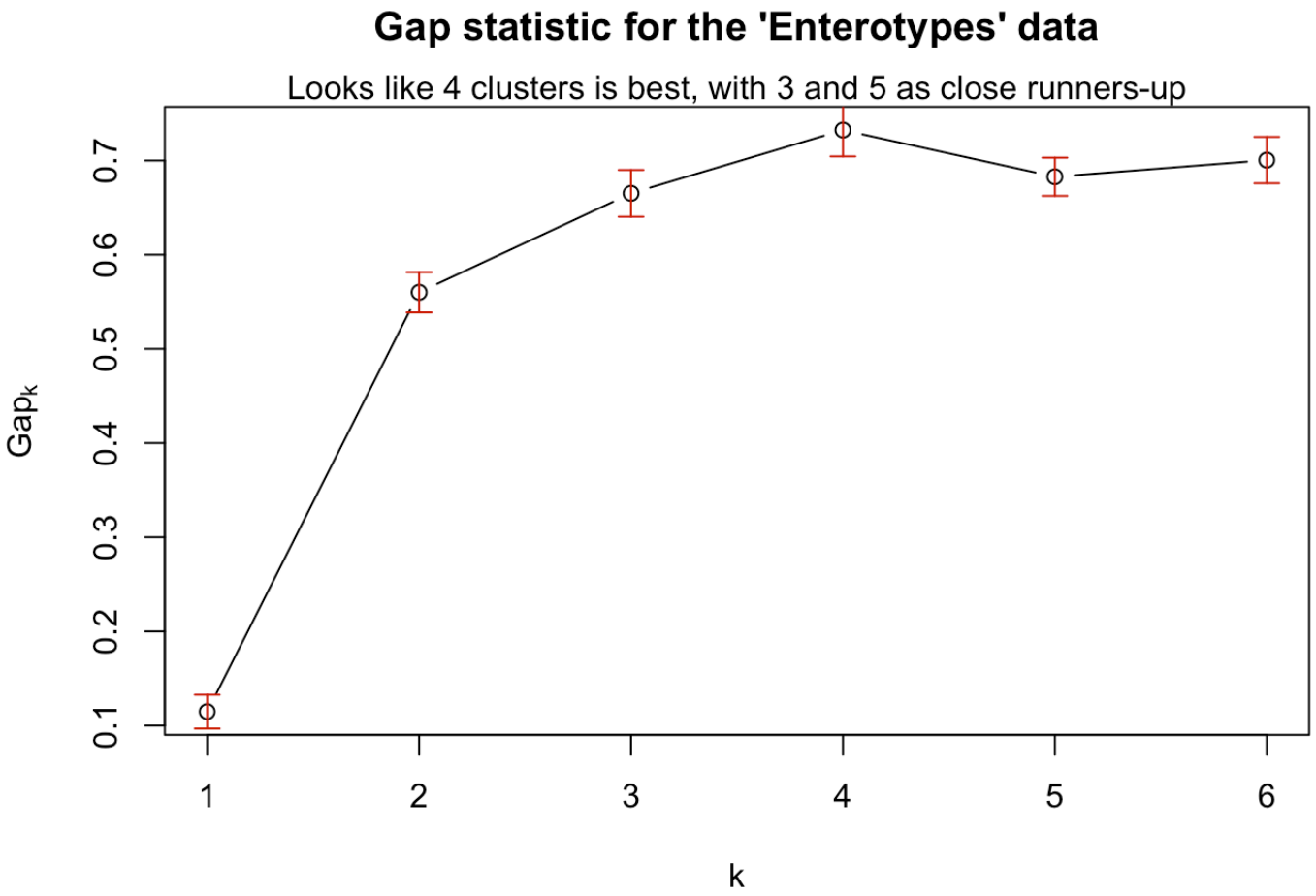
```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = x[, axes], FUNcluster = FUNcluster, K.max = K.max,     B = B, verbose
= verbose)
## B=50 simulated reference sets, k = 1..6; spaceH0="scaledPCA"
##  --> Number of clusters (method 'Tibs2001SEmax', SE.factor=1): 4
##          logW   E.logW       gap     SE.sim
## [1,] 2.995599 3.110399 0.1147998 0.01794265
## [2,] 2.209852 2.769942 0.5600904 0.02134329
## [3,] 1.922188 2.587328 0.6651399 0.02483978
## [4,] 1.685798 2.418179 0.7323813 0.02803585
## [5,] 1.601025 2.283779 0.6827548 0.02028888
## [6,] 1.480640 2.181093 0.7004524 0.02463325
```

```
plot_clusgap(gs)
```



For comparison, let's see what happens if we only use plotting as available in `base` R:

```
# Base R plotting

plot(gs, main = "Gap statistic for the 'Enterotypes' data")
mtext("Looks like 4 clusters is best, with 3 and 5 as close runners-up")
```



Ordination Plots

We want to filter low-occurrence, poorly-represented OTUs from these data, because they are essentially noise variables for purposes of this lesson. In practice, you should probably perform clearly-document well-justified pre-processing steps.

First we want to remove OTUs that do not appear more than 5 times in more than half the samples.

```
# Remove OTUs that don't appear more than 5 times in more than half the samples

GP <- GlobalPatterns
wh0 <- genefilter_sample(GP, filterfun_sample(function(x) x>5), A = 0.5*nsamples(GP))
GP1 <- prune_taxa(wh0, GP)
```

Transform to an even sampling depth.

```
# Transform to an even sampling depth

GP1 <- transform_sample_counts(GP1, function(x) 1e6 * x/sum(x))
```

Keep only the 5 most abundant phyla.

```
# Keep the top 5 most abundant phyla

phylum.sum <- tapply(taxa_sums(GP1), tax_table(GP1)[, "Phylum"], sum, na.rm = TRUE)
top5phyla <- names(sort(phylum.sum, TRUE))[1:5]
GP1 <- prune_taxa((tax_table(GP1)[, "Phylum"] %in% top5phyla), GP1)
summary(GP1)
```

```
##   Length    Class     Mode
##        1 phyloseq       S4
```

```
GP1
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 204 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 204 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 204 tips and 203 internal nodes ]
```

So we still have 204 taxa that are in the dataset `GP1` that we will use going forward.

We will want to investigate human-associated microbiomes vs not, so we will define as follows:

```
# Define human-associate samples
human <- get_variable(GP1, "SampleType") %in% c("Feces", "Mock", "Skin", "Tongue")
sample_data(GP1)$human <- factor(human)
```

Four Main Types of Ordination Plots

There are four basic representations of an ordination.

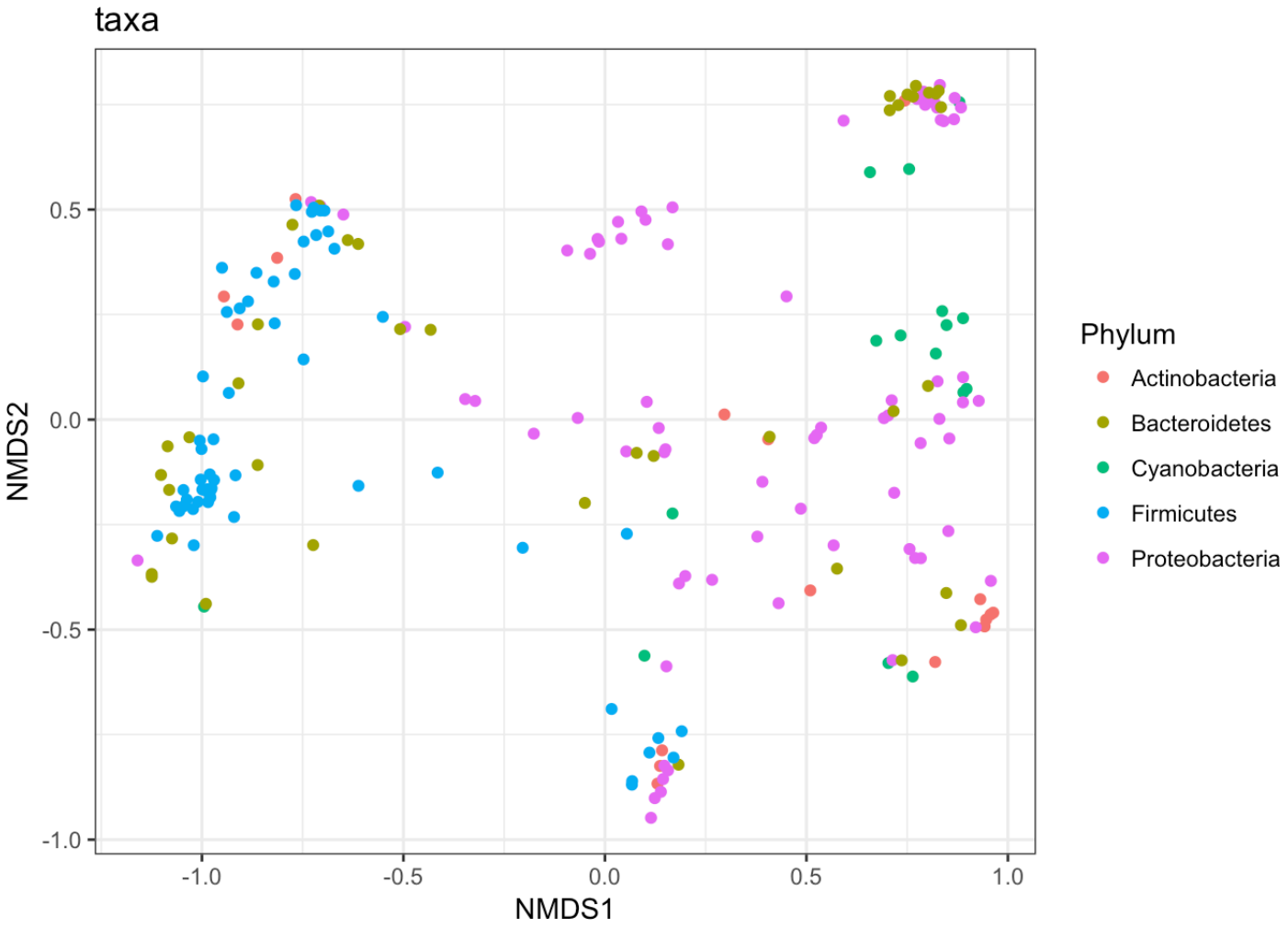
Just OTUs

Let's plot just the OTUs and shade by Phylum. Remember there will `ntaxa(GP1)` = 204 OTUs in this plot.

```
# Plot ordination for just OTUs

GP.ord <- ordinate(GP1, "NMDS", "bray")
```

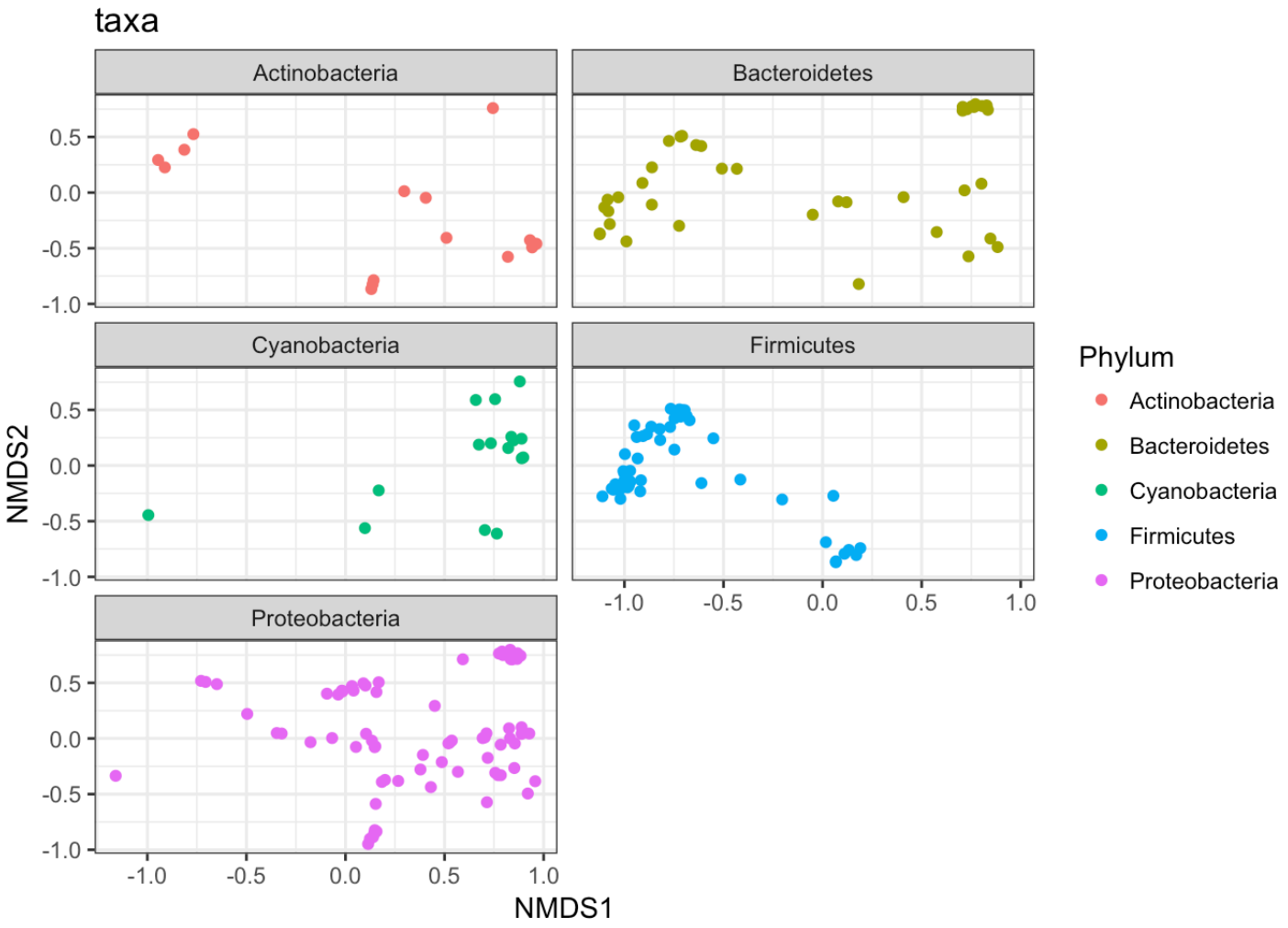
```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1333468
## Run 1 stress 0.1448829
## Run 2 stress 0.1333468
## ... Procrustes: rmse 2.095302e-06  max resid 5.470897e-06
## ... Similar to previous best
## Run 3 stress 0.3783527
## Run 4 stress 0.1570722
## Run 5 stress 0.1333468
## ... Procrustes: rmse 1.085015e-05  max resid 3.243687e-05
## ... Similar to previous best
## Run 6 stress 0.1333468
## ... New best solution
## ... Procrustes: rmse 1.200093e-06  max resid 3.579923e-06
## ... Similar to previous best
## Run 7 stress 0.1690105
## Run 8 stress 0.1469145
## Run 9 stress 0.1460272
## Run 10 stress 0.1385323
## Run 11 stress 0.1488936
## Run 12 stress 0.1680691
## Run 13 stress 0.1333468
## ... New best solution
## ... Procrustes: rmse 4.668917e-06  max resid 1.4577e-05
## ... Similar to previous best
## Run 14 stress 0.1506169
## Run 15 stress 0.1644341
## Run 16 stress 0.1486045
## Run 17 stress 0.1333468
## ... Procrustes: rmse 4.092483e-06  max resid 9.214519e-06
## ... Similar to previous best
## Run 18 stress 0.1585099
## Run 19 stress 0.1333468
## ... Procrustes: rmse 2.013768e-05  max resid 5.999941e-05
## ... Similar to previous best
## Run 20 stress 0.1333468
## ... Procrustes: rmse 6.569186e-06  max resid 1.939565e-05
## ... Similar to previous best
## *** Solution reached
```

```
p1 <- plot_ordination(GP1, GP.ord, type="taxa", color="Phylum", title = "taxa")
print(p1)
```



There is a lot going on here, and a lot of points are superimposed on one another. One way to deal with that is to use facetting.

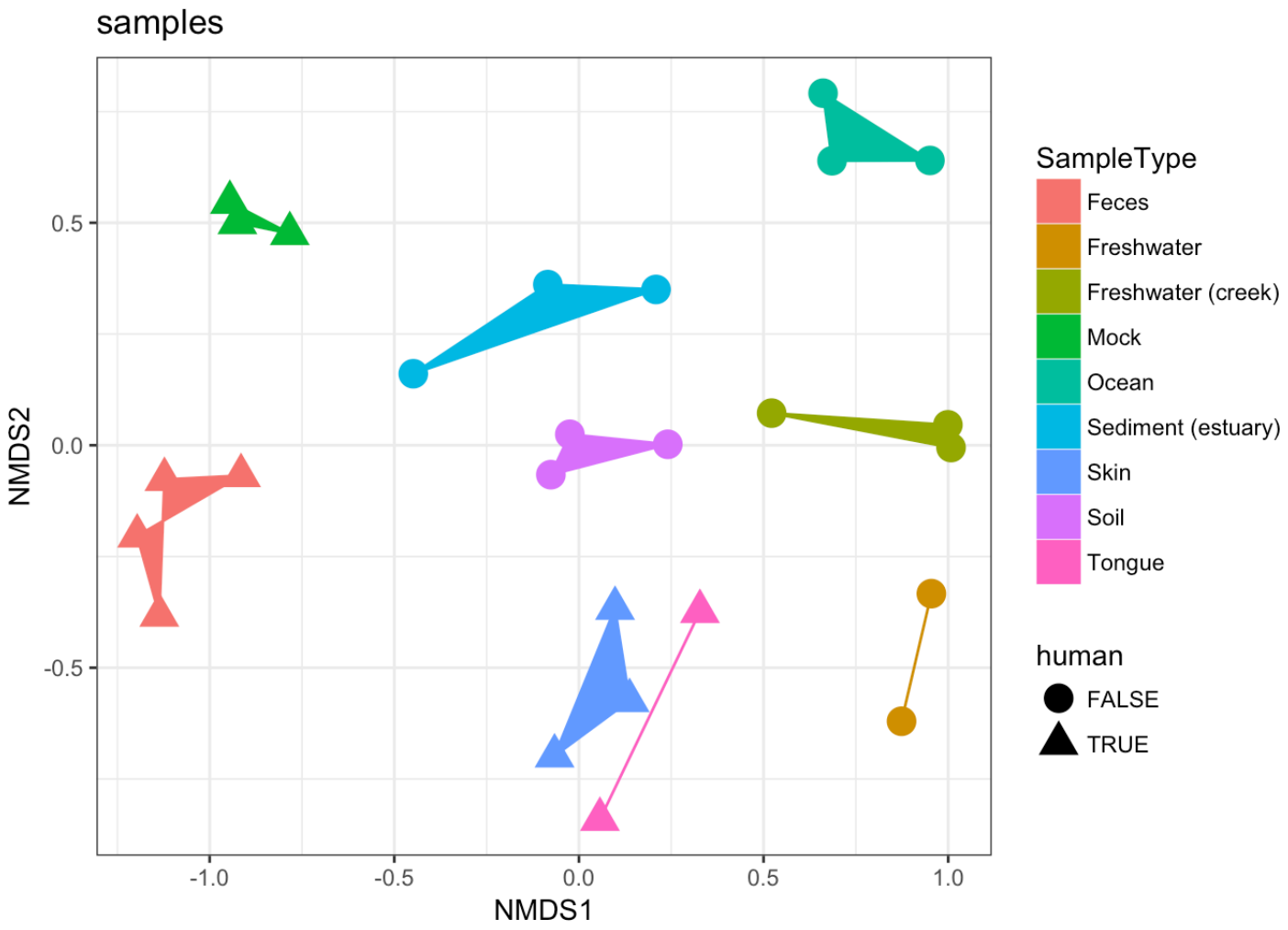
```
# Facet-wrapping for previous plot

p1 + facet_wrap(~Phylum, 3)
```



Just Samples

Another way to plot is to consider the samples, and shade by “SampleType”, modifying shape according to if they are human associated.

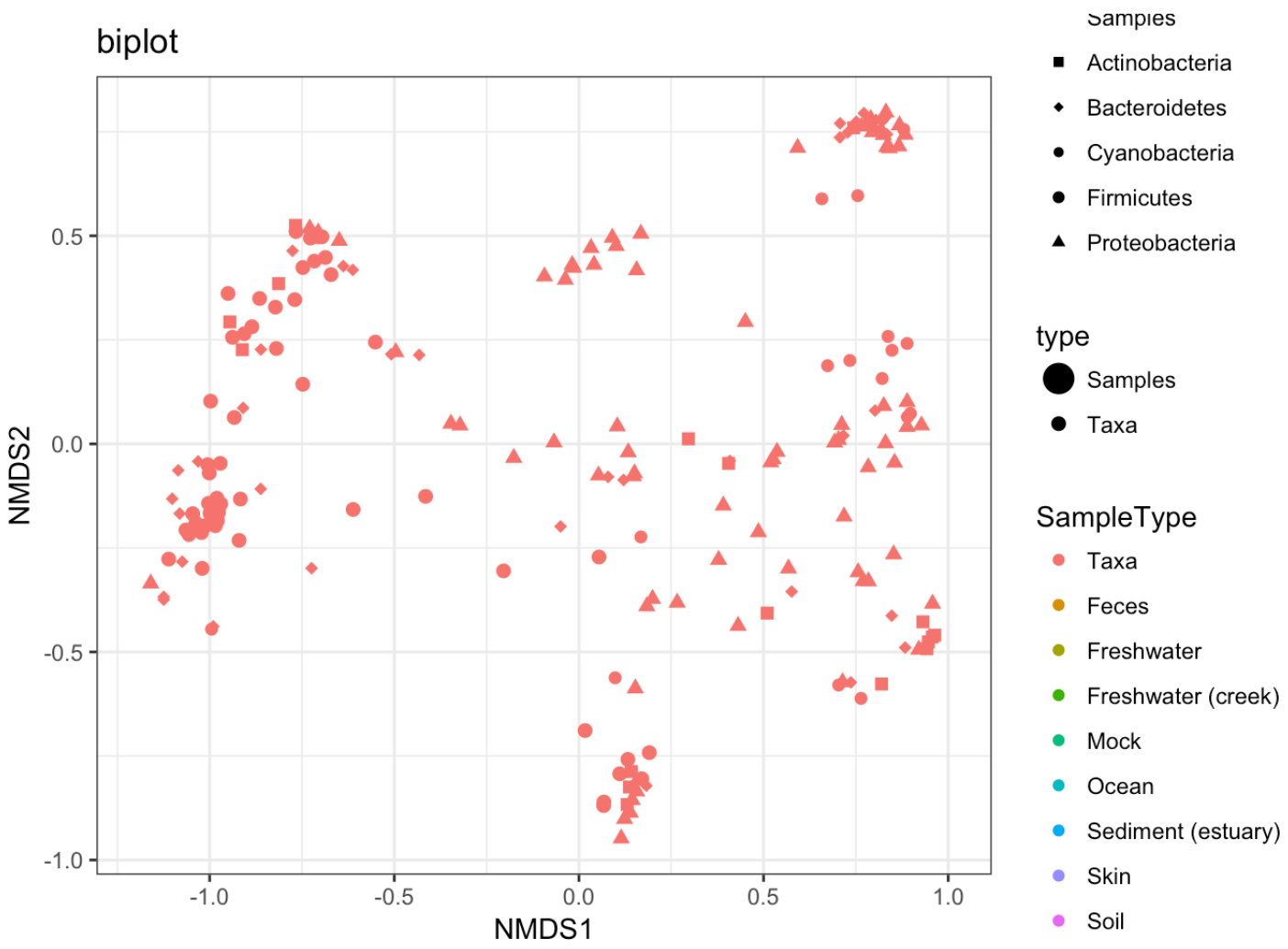
```
# Plot just the Samples

p2 <- plot_ordination(GP1, GP.ord, type="samples", color="SampleType", shape="human")
p2 + geom_polygon(aes(fill=SampleType)) +
  geom_point(size=5) +
  ggtitle("samples")
```



Biplot Graphic of OTUs and Samples Together

The `plot_ordination` function can also automatically create two different graphic layouts in which both the sampels and OTUs are plotted together in one “biplot”. This will not work for methods that are intrinsically samples-only ordinations (e.g., UniFrac/PCoA).

```
# Biplot Graph

p3 <- plot_ordination(GP1, GP.ord, type="biplot", color="SampleType", shape="Phylum",
title = "biplot")
 # The following will modify the automatic shape scale
GP1.shape.names <- get_taxa_unique(GP1, "Phylum")
GP1.shapes <- 15:(15 + length(GP1.shape.names) - 1)
names(GP1.shapes) <- GP1.shape.names
GP1.shapes["samples"] <- 16

p3 + scale_shape_manual(values=GP1.shapes)
```



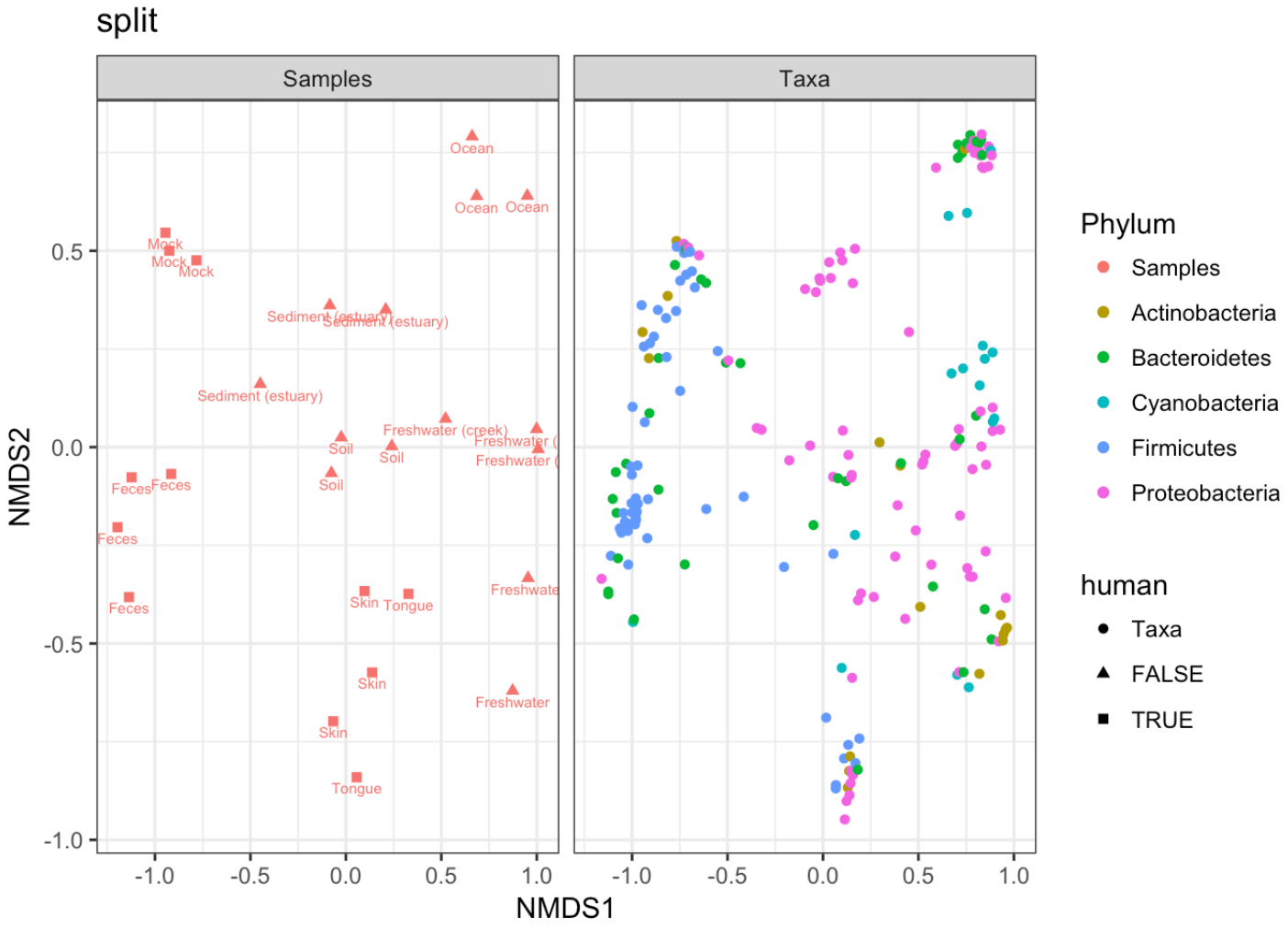
Split Graphic

We have the same problem as before, with a lot of overlay that might obscure meanint. Let's try to separate into side-by-side panels.

```
# Split graphic into panels

p4 <- plot_ordination(GP1, GP.ord, type="split", color="Phylum", shape = "human", lab
el = "SampleType", title = "split")
```

```
## Warning: Ignoring unknown aesthetics: na.rm
```

```
p4
```



Ordination Methods

Now we are going o loop through different `method` parameter options to the `plot_ordination` function, store the plot results, then plot in a combined graphic.

```
# First use different ordination methods and plot to list

dist <- "bray"
ord_meths <- c("DCA", "CCA", "RDA", "DPCoA", "NMDS", "MDS", "PCoA")
plist <- llply(as.list(ord_meths), function(i, physeq, dist){
  ordi <- ordinate(physeq, method=i, distance = dist)
  plot_ordination(physeq, ordi, "samples", color="SampleType")
}, GP1, dist)
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.1333468
## Run 1 stress 0.182469
## Run 2 stress 0.1450956
## Run 3 stress 0.1634732
## Run 4 stress 0.1385324
## Run 5 stress 0.1518734
## Run 6 stress 0.1531745
## Run 7 stress 0.1486045
## Run 8 stress 0.1831025
## Run 9 stress 0.1677043
## Run 10 stress 0.1385324
## Run 11 stress 0.1513368
## Run 12 stress 0.1471619
## Run 13 stress 0.1518734
## Run 14 stress 0.1465222
## Run 15 stress 0.1385323
## Run 16 stress 0.2805829
## Run 17 stress 0.1385324
## Run 18 stress 0.1333468
## ... Procrustes: rmse 6.41087e-06  max resid 1.867542e-05
## ... Similar to previous best
## Run 19 stress 0.3693297
## Run 20 stress 0.1449081
## *** Solution reached
```

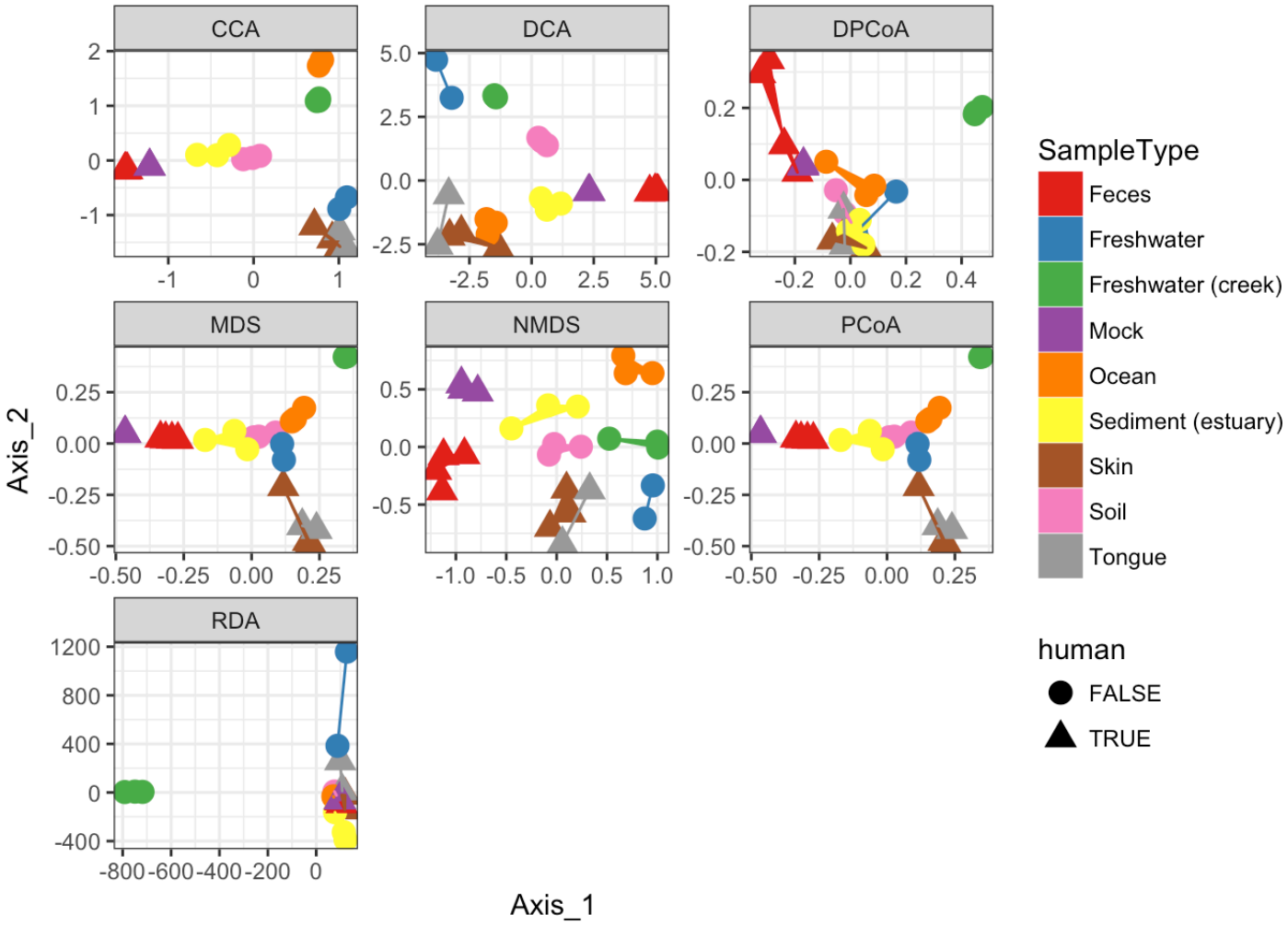
```
names(plist) <- ord_meths
```

Now we will extract the data from each of the individual plots and put them back together in one big data.frame suitable for including all plots in one graphic.

```
# Extract data from each of the individual plots and put them all together in one big dataframe

pdataframe <- ldply(plist, function(x){
  df <- x$data[, 1:2]
  colnames(df) <- c("Axis_1", "Axis_2")
  return(cbind(df, x$data))
})

names(pdataframe)[1] <- "method"
```

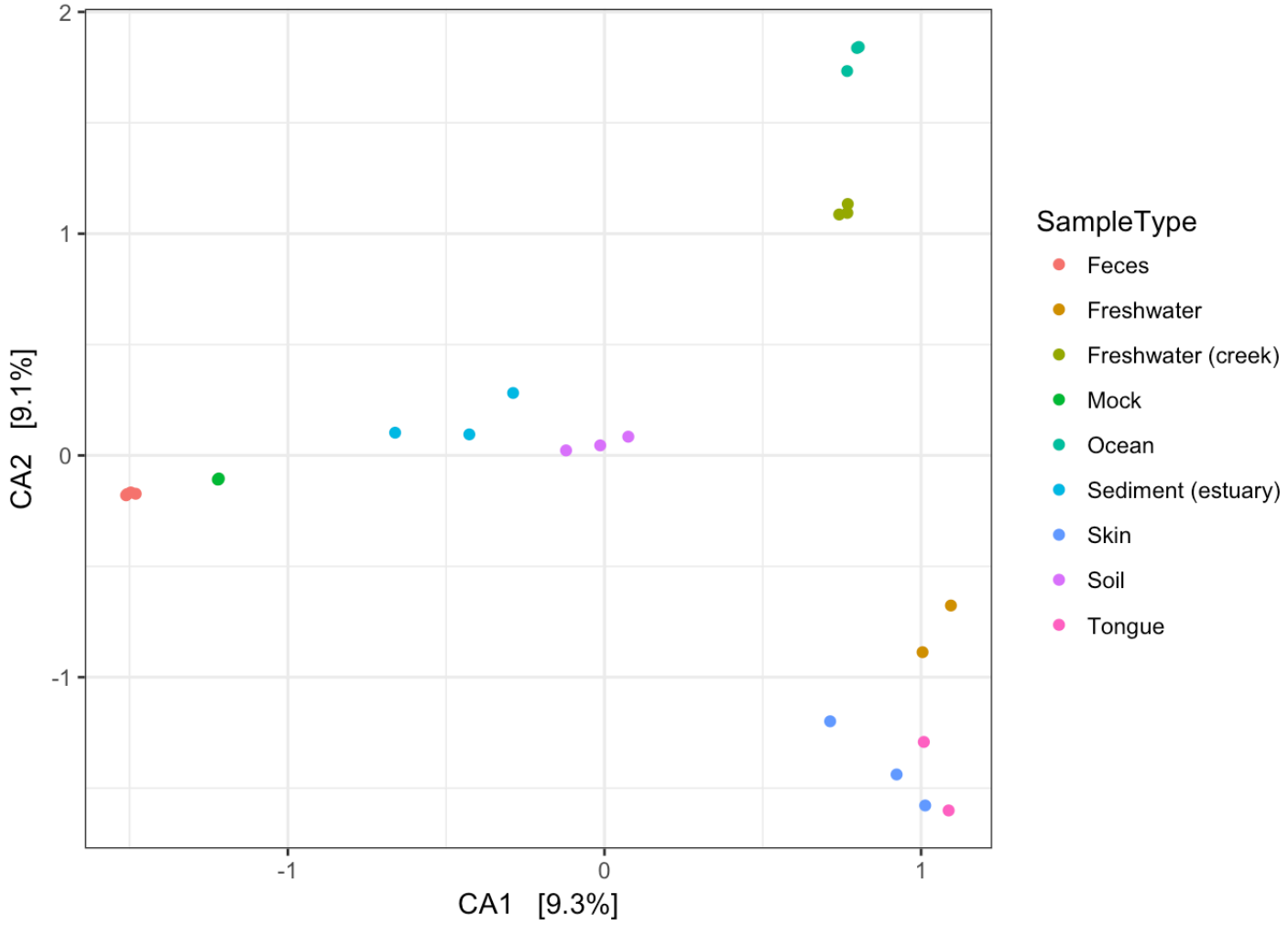
Now with all ordination results combined in one dataframe, called `pdataframe` , we can make a standard faceted scatterplot with ggplot2.

```
# Make faceted scatterplot

p <- ggplot(pdataframe, aes(Axis_1, Axis_2, color = SampleType, shape=human, fill=Sam
pleType))
p <- p + geom_point(size=4) +
          geom_polygon()
p <- p + facet_wrap(~method, scales="free")
p <- p + scale_fill_brewer(type="qual", palette = "Set1")
p <- p + scale_colour_brewer(type="qual", palette = "Set1")
p
```



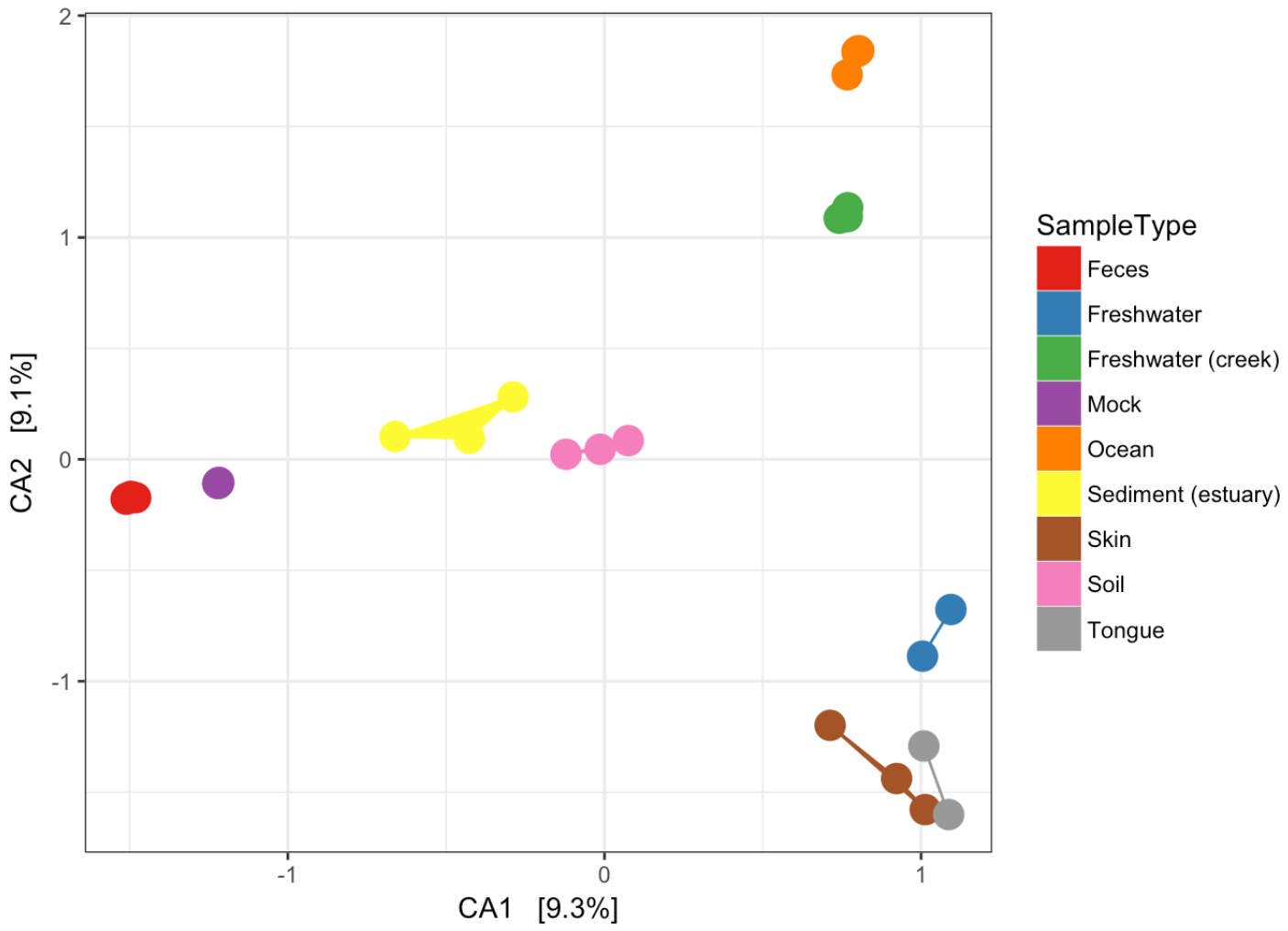
If you want t larger version of an individual plot, print from the original `plist` from which the `pdataframe` was mde. For instance plot the detrended correspondence analysis (DCA), which is the second element of the list.

```
# Print DCA from original plist


plist[[2]]
```



Make it look nicer.

```
#Fluff it up

p <- plist[[2]] + scale_colour_brewer(type="qual", palette = "Set1")
p <- p + scale_fill_brewer(type="qual", palette = "Set1")
p <- p + geom_point(size = 5) +
         geom_polygon(aes(fill=SampleType))
p
```

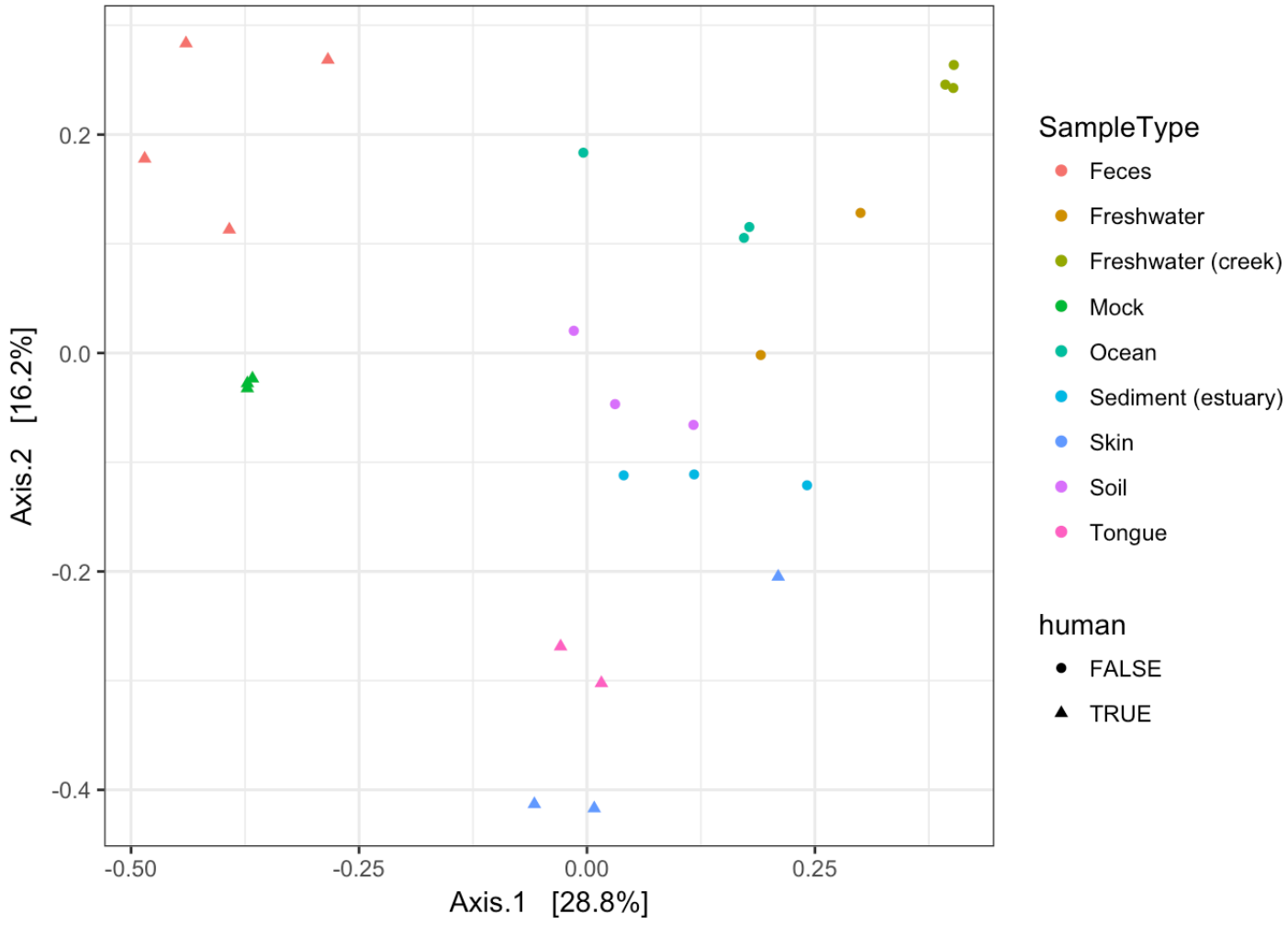


PCoA on UniFrac Distances

You can also do PCoA on UniFrac distances

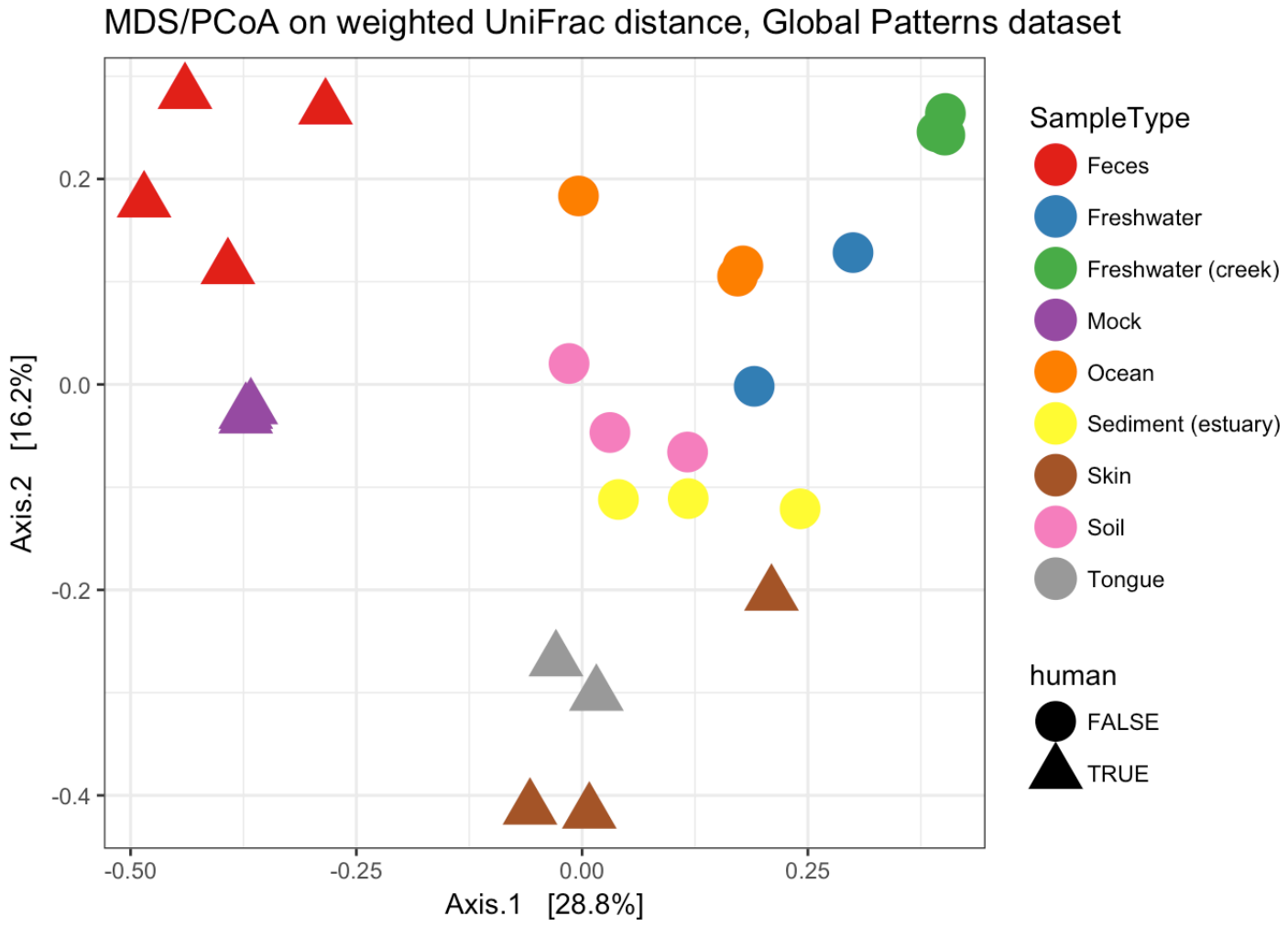
Use the `ordinate` function to perform weighted UniFrac then perform PCoA on the distance matrix. Then pass the data and the ordination results to `plot_ordination` to create the `ggplot2` graphic withe default settings.

```
# get unifrac distances and pcoa them

ordu <- ordinate(GP1, "PCoA", "unifrac", weighted=TRUE)

# plot it

plot_ordination(GP1, ordu, color="SampleType", shape = "human")
```



Let's pretty it up.

```
# Fluff up the graph

p <- plot_ordination(GP1, ordu, color="SampleType", shape="human")
p <- p + geom_point(size=7)
p <- p + scale_colour_brewer(type="qual", palette = "Set1")
p + ggtitle("MDS/PCoA on weighted UniFrac distance, Global Patterns dataset")
```



Alpha Diversity Graphics

Although the command is to the `plot_richness` function, the function in fact calculates to more than to richness, in fact to all descriptions of alpha diversity.

First some setup

```
# Some setup for graphics

theme_set(theme_bw())
pal <- "Set1"
scale_colour_discrete <-  function(palname=pal, ...){
  scale_colour_brewer(palette=palname, ...)
}
scale_fill_discrete <-  function(palname=pal, ...){
  scale_fill_brewer(palette=palname, ...)
}
```

Prepare the Data

It is probably not a bad idea to prune OTUs that are not present in any of the samples (somehow that happens) but *don't trim more than that!* Many richness estimates are modeled on singletons and doubletons, and you need to leave them in the dataset to get meaningful estimates.

```
# Prune off species = 0 across all samples

GP <- prune_taxa(taxa_sums(GlobalPatterns) > 0, GlobalPatterns)
```

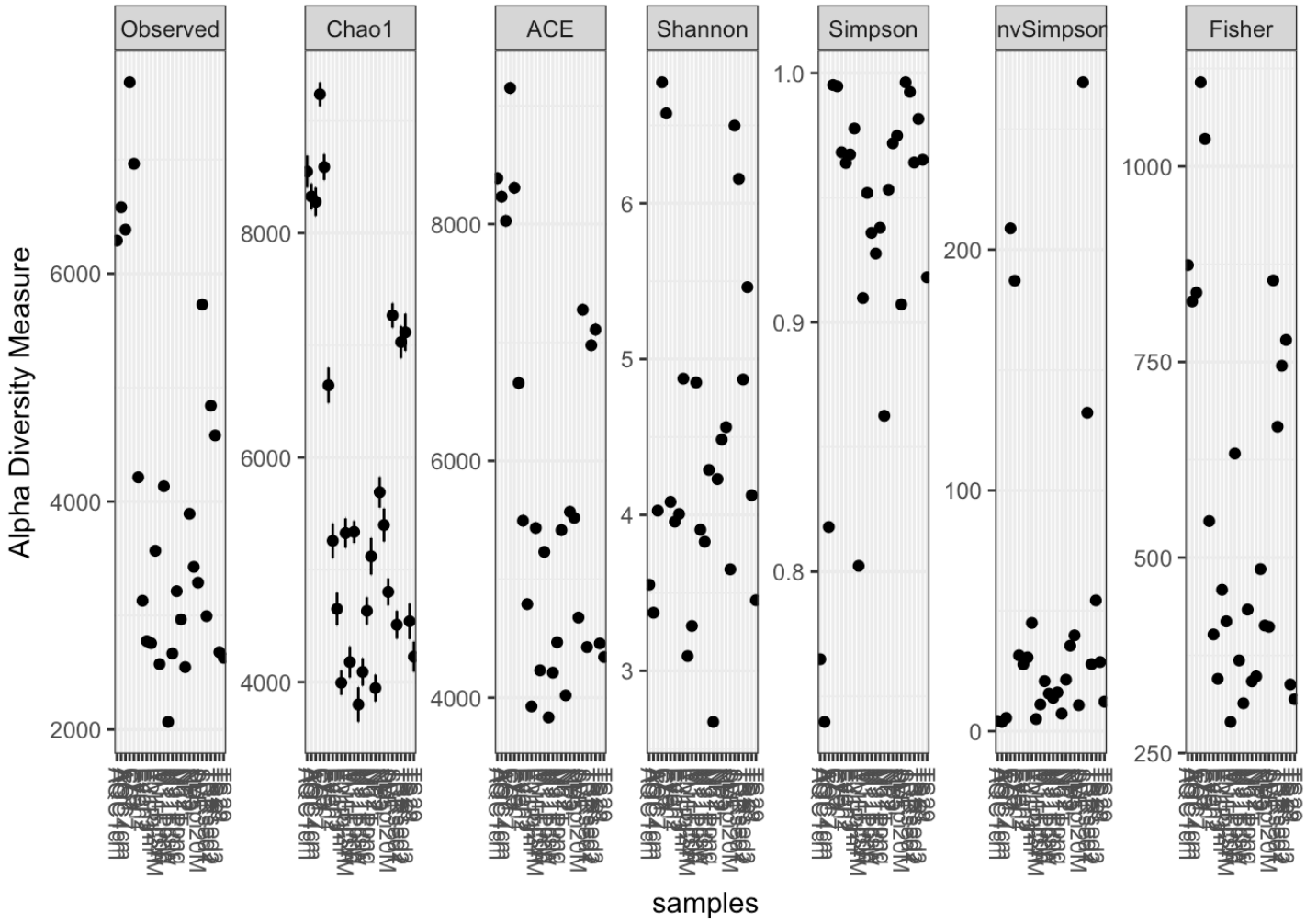
Plot Examples

The Default Graphic

The `plot_richness` function produces the default graphic. Let's look at the `GP` example dataset.

```
# Default plot

plot_richness(GP)
```

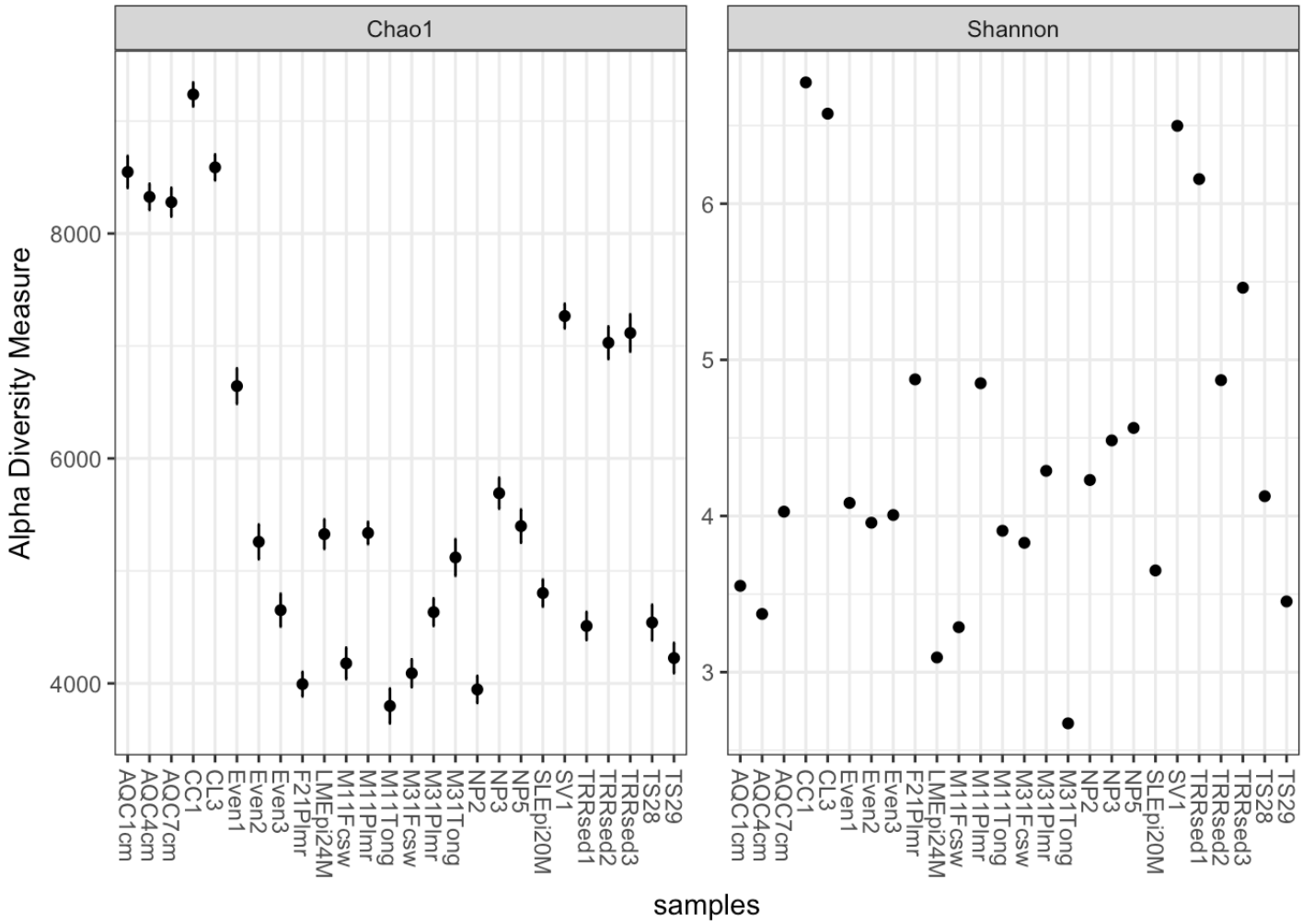
```
## Warning: Removed 130 rows containing missing values (geom_errorbar).
```



We can, of course, just select a couple of measures to plot.

```
# Plot only Chao1, Shannon

plot_richness(GP, measures = c("Chao1", "Shannon"))
```

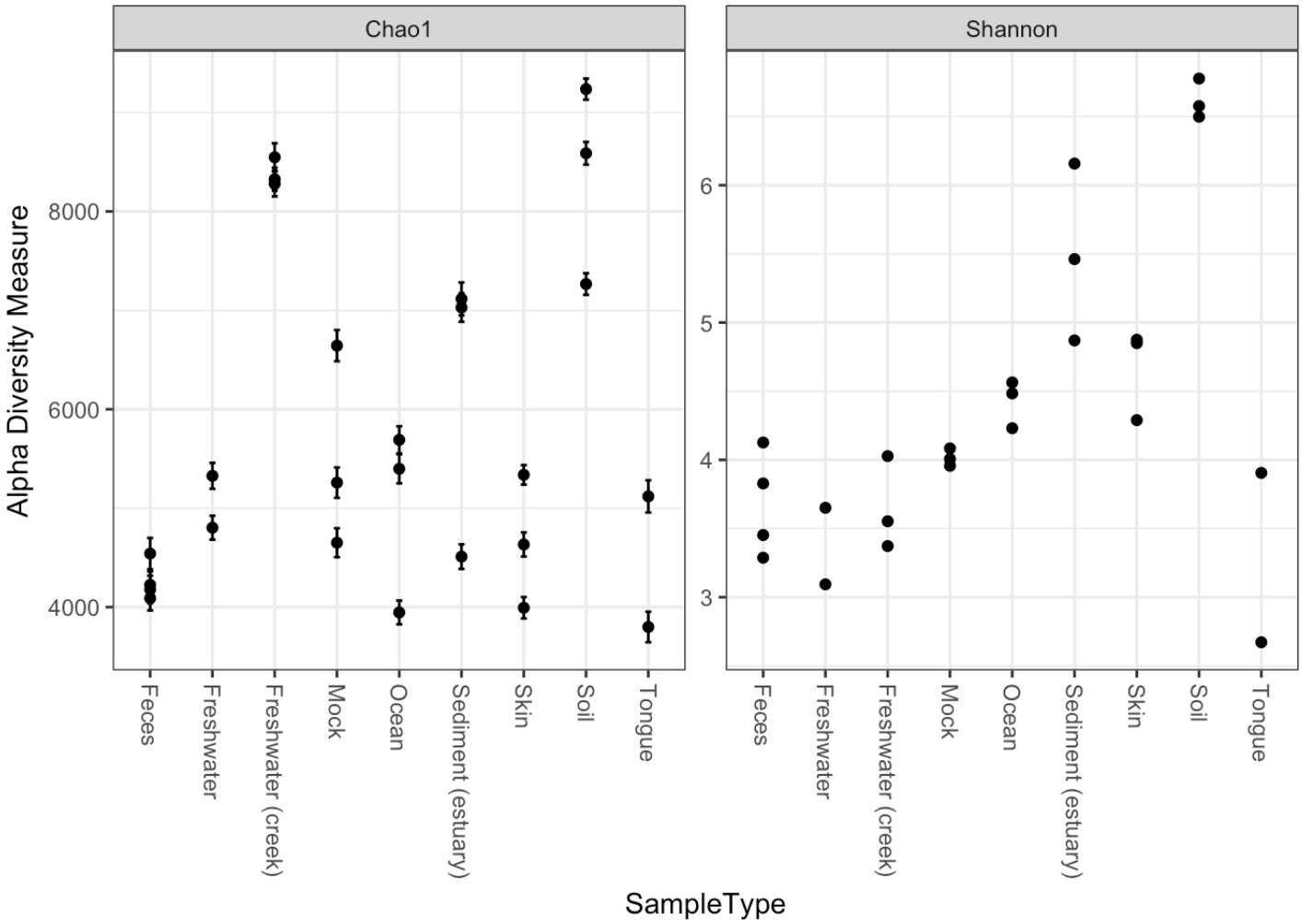
```
## Warning: Removed 26 rows containing missing values (geom_errorbar).
```



We can also specify a sample variable on which to organize the samples along the x-axis. An experimentally meaningful categorical variable is usually a good choice.

```
# Plot only Chao1, Shannon

plot_richness(GP, x="SampleType", measures = c("Chao1", "Shannon"))
```

```
## Warning: Removed 26 rows containing missing values (geom_errorbar).
```



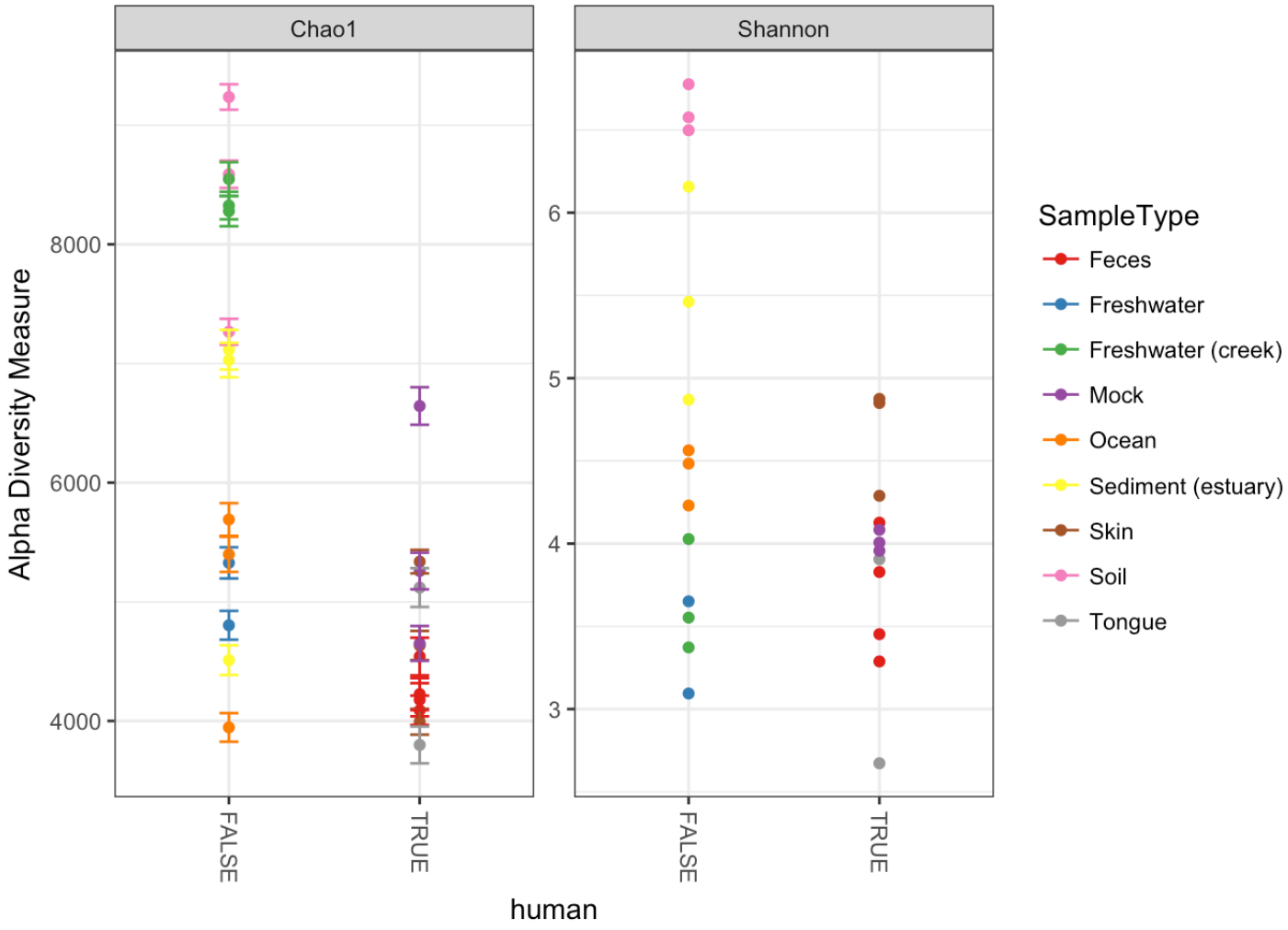
Suppose we wanted to organize around an external variable that is not in the dataset already. As an example, think of a variable indicating if the sample was human or not. Let's define that variable first.

```
# Create indicator variable if sample is human or not.

sample_data(GP)$human <- get_variable(GP, "SampleType") %in% c("Feces", "Mock", "Skin
", "Tongue")
```

Now we tell `plot_richness` to map the new human variable on the x-axis, and shaade the points in different color groups accoring to SampleType.

```
# Use indicator for human as x-axis, shade according to sample type

plot_richness(GP, x="human", color="SampleType", measures = c("Chao1", "Shannon"))
```

```
## Warning: Removed 26 rows containing missing values (geom_errorbar).
```



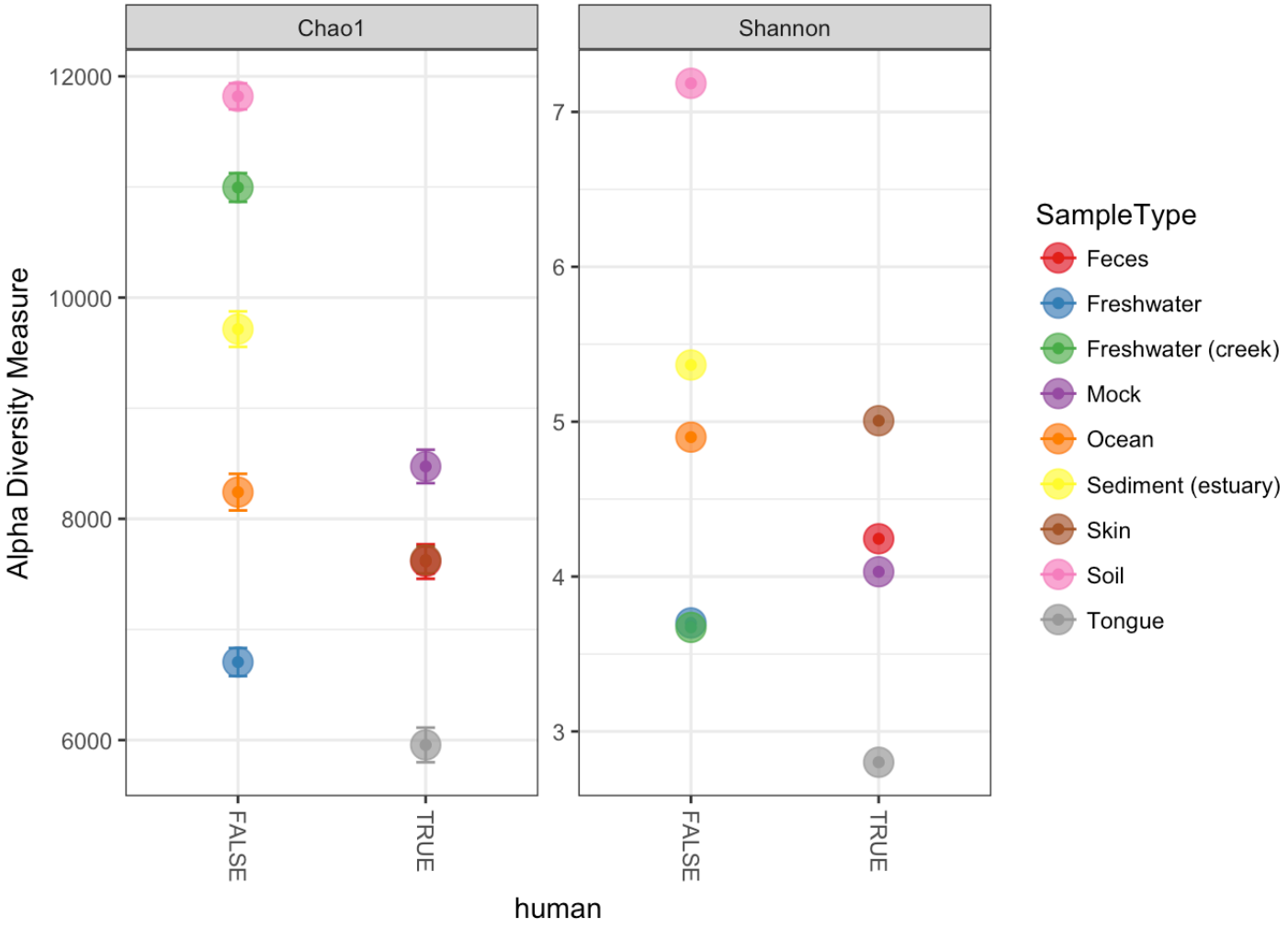
We can merge samples that are from the environment (SampleType) and make the points bigger with `ggplot2`. First, merge samples.

```
# Merge samples

GPst <- merge_samples(GP, "SampleType")

# Repair variables damaged during merge (coerced to numeric)
sample_data(GPst)$SampleType <- factor(sample_names(GPst))
sample_data(GPst)$human <- as.logical(sample_data(GPst)$human)
```

Now to plot

```
p <- plot_richness(GPst, x="human", color="SampleType", measures = c("Chao1", "Shanno
n"))
p <- p + geom_point(size=5, alpha=0.7)
p
```

```
## Warning: Removed 9 rows containing missing values (geom_errorbar).
```



Tree Graphics

The `phyloseq` package also has powerful graphics for displaying and manipulating phylogenetic trees with the `plot_tree` function.

Example

We want to plot trees, but we notice that the node labels for the GlobalPatterns dataset are, well, strange. Here they are:

```
# Node labels for phylogenetic tree of GlobalPatterns

head(phy_tree(GlobalPatterns)$node.label, 10)
```

```
##  [1] ""          "0.858.4"   "1.000.154" "0.764.3"   "0.995.2"
##  [6] "1.000.2"   "0.943.7"   "0.971.6"   "0.766"     "0.611"
```

These might be bootstrap values, but what is with the two decimals?

Let's just take the first 4 characters.

```
# Take first 4 characters of labels

phy_tree(GlobalPatterns)$node.label <- substr(phy_tree(GlobalPatterns)$node.label, 1,
4)
head(phy_tree(GlobalPatterns)$node.label)
```

```
## [1] ""     "0.85" "1.00" "0.76" "0.99" "1.00"
```

Much better, these look like bootstrap values now.

The dataset has many OTUs, more than will fit nicely on a tree graphic.

```
# Number of taxa in dataset

ntaxa(GlobalPatterns)
```

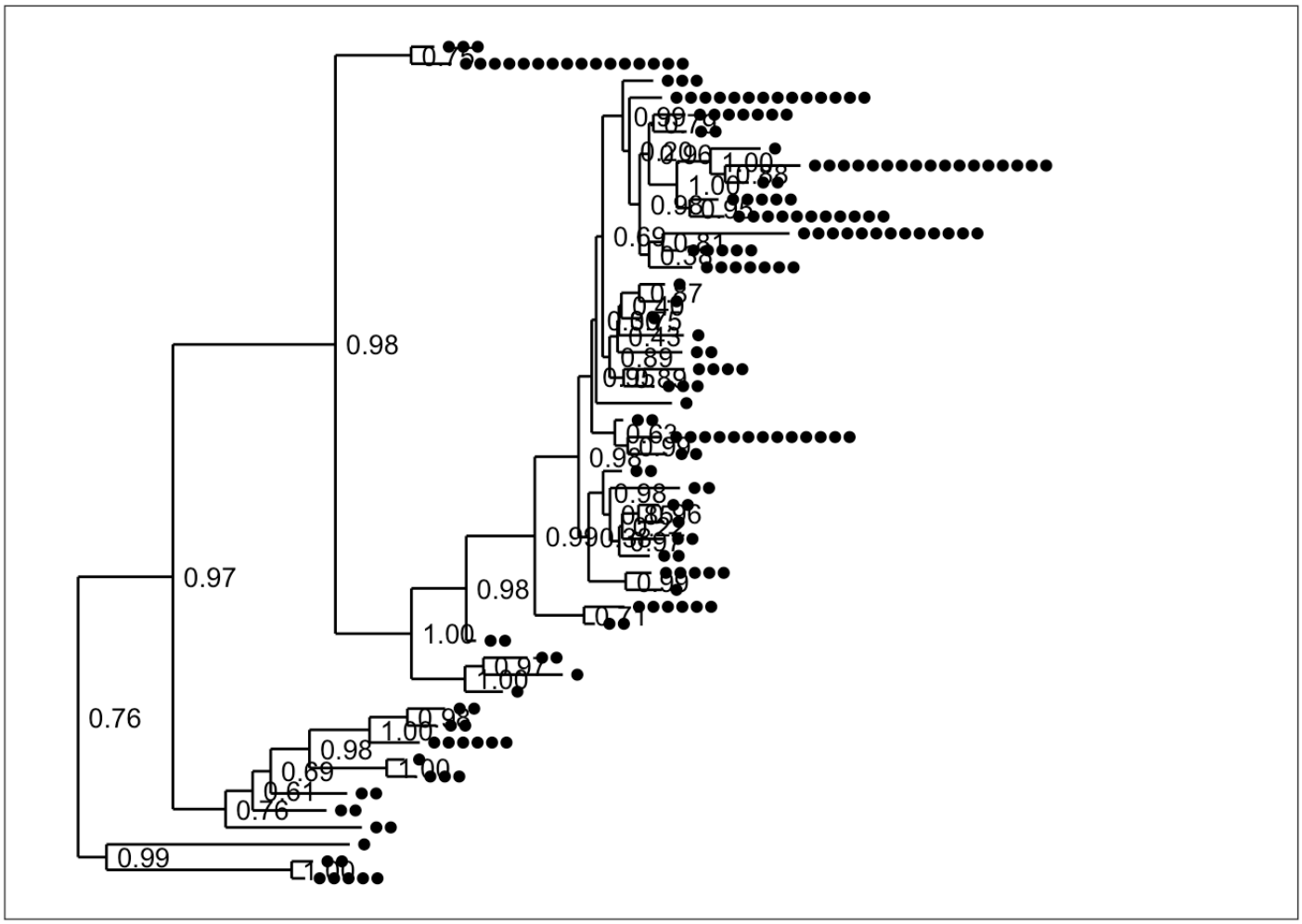
```
## [1] 19216
```

Let's arbitarily prune to the first 50 OTUs in the dataset and store.

```
# Prune to first 50 OTUs

physeq <- prune_taxa(taxa_names(GlobalPatterns)[1:50], GlobalPatterns)
```

If we just use the default `plot_tree` setting, look at what we get with this pruned tree.

```
# Plot the tree

plot_tree(physeq)
```

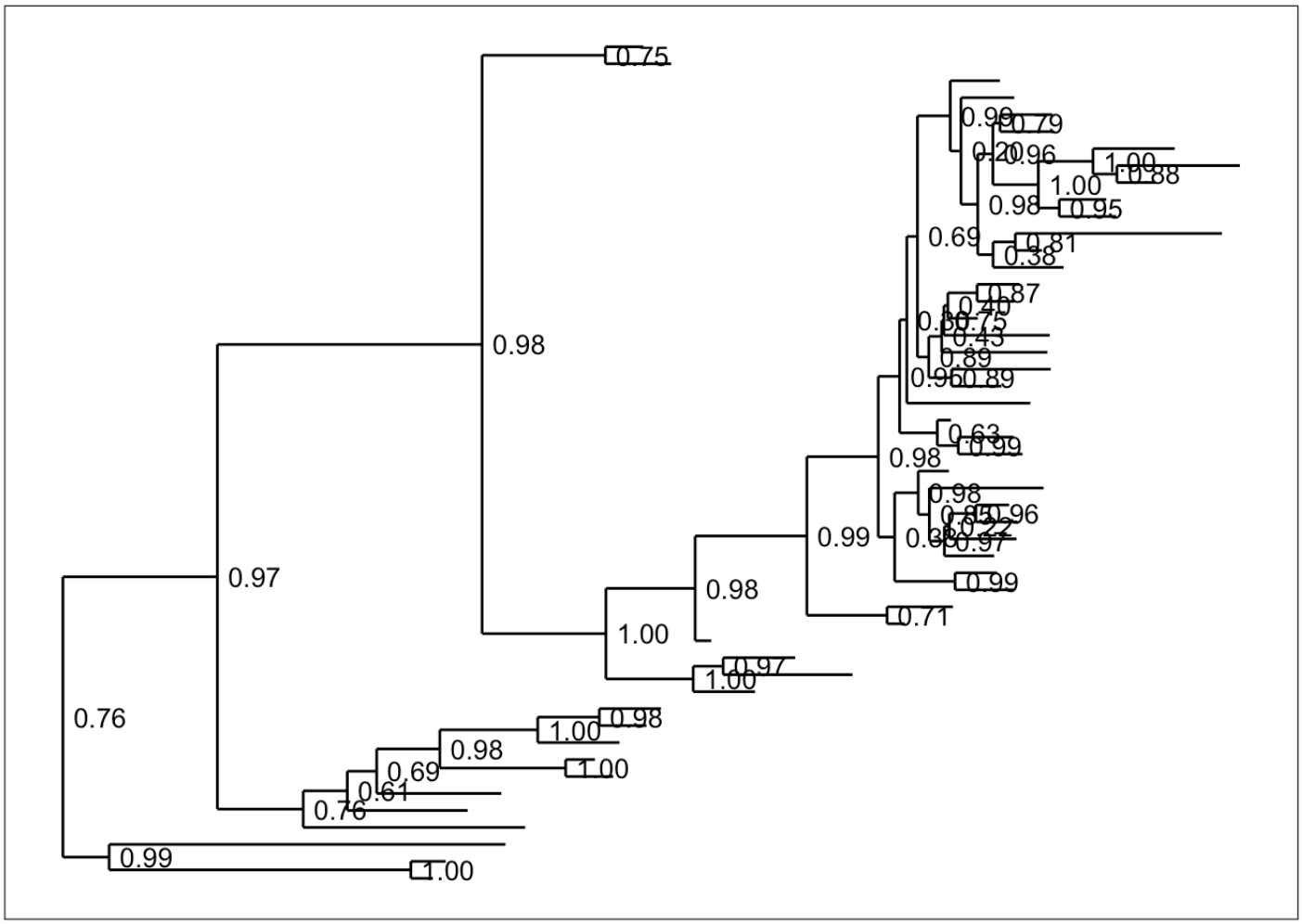


Kind of ugly. We can do beter.

The black dots next to the tips are one for each sample in which that OTU was observed. Some have more dots than others. The node labels are also next to the tips, making everything pretty messy. How can we clean this up?

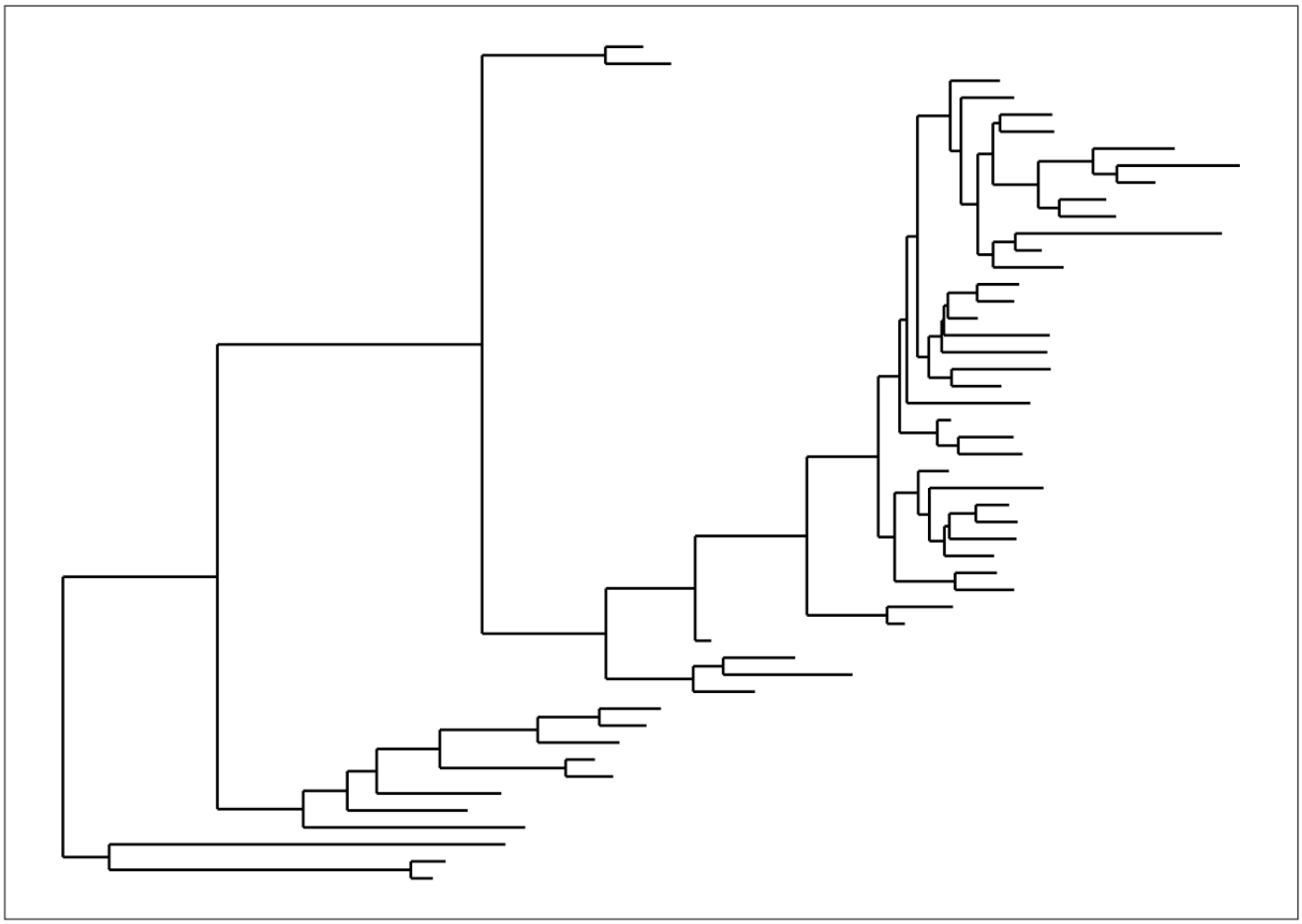
First a tree with labels only.

```
# Plot tree, no points

plot_tree(physeq, "treeonly")
```



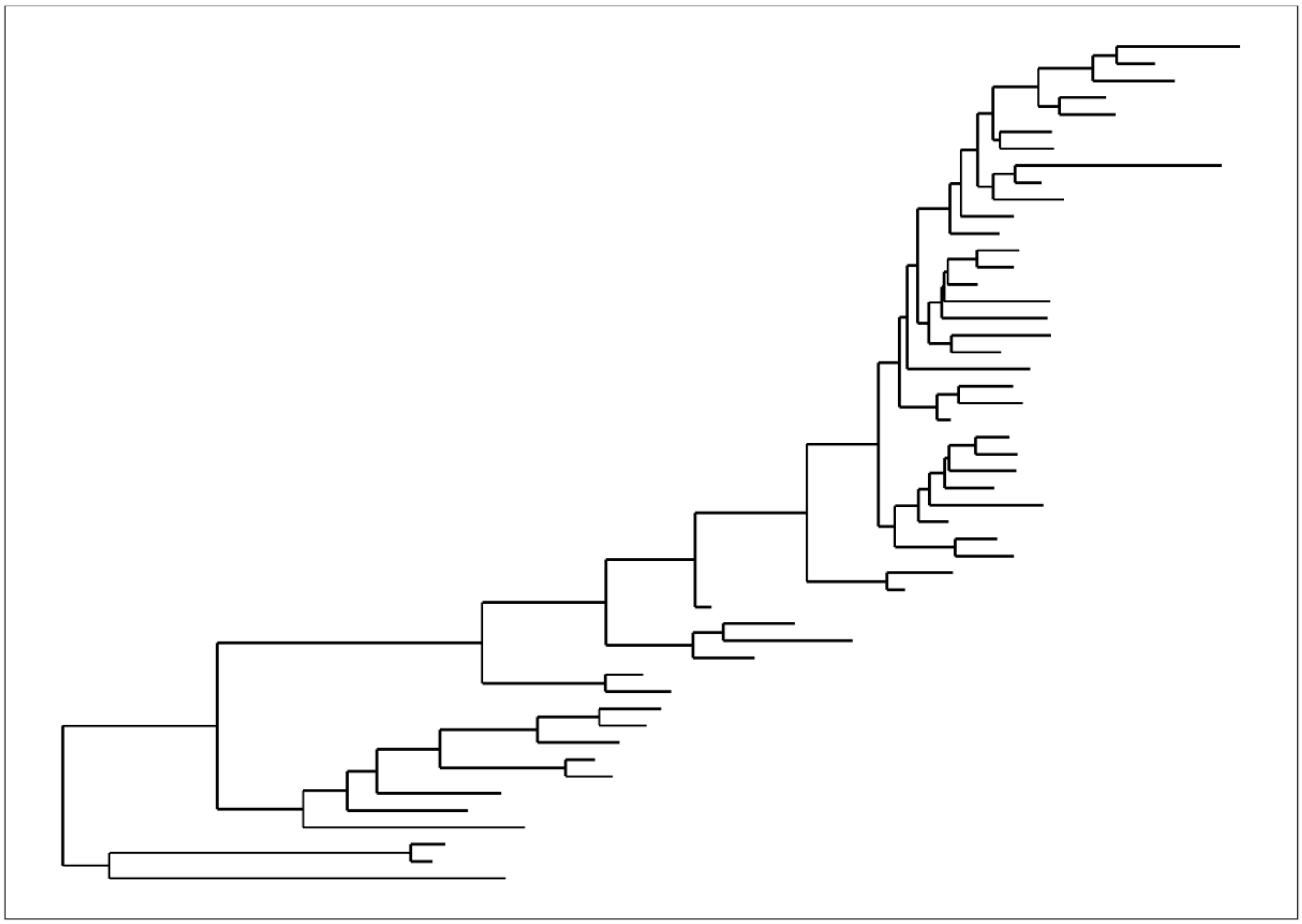
What about without the labels?

```
# Plot with no points and no labels

plot_tree(physeq, "treeonly", nodeplotblank)
```

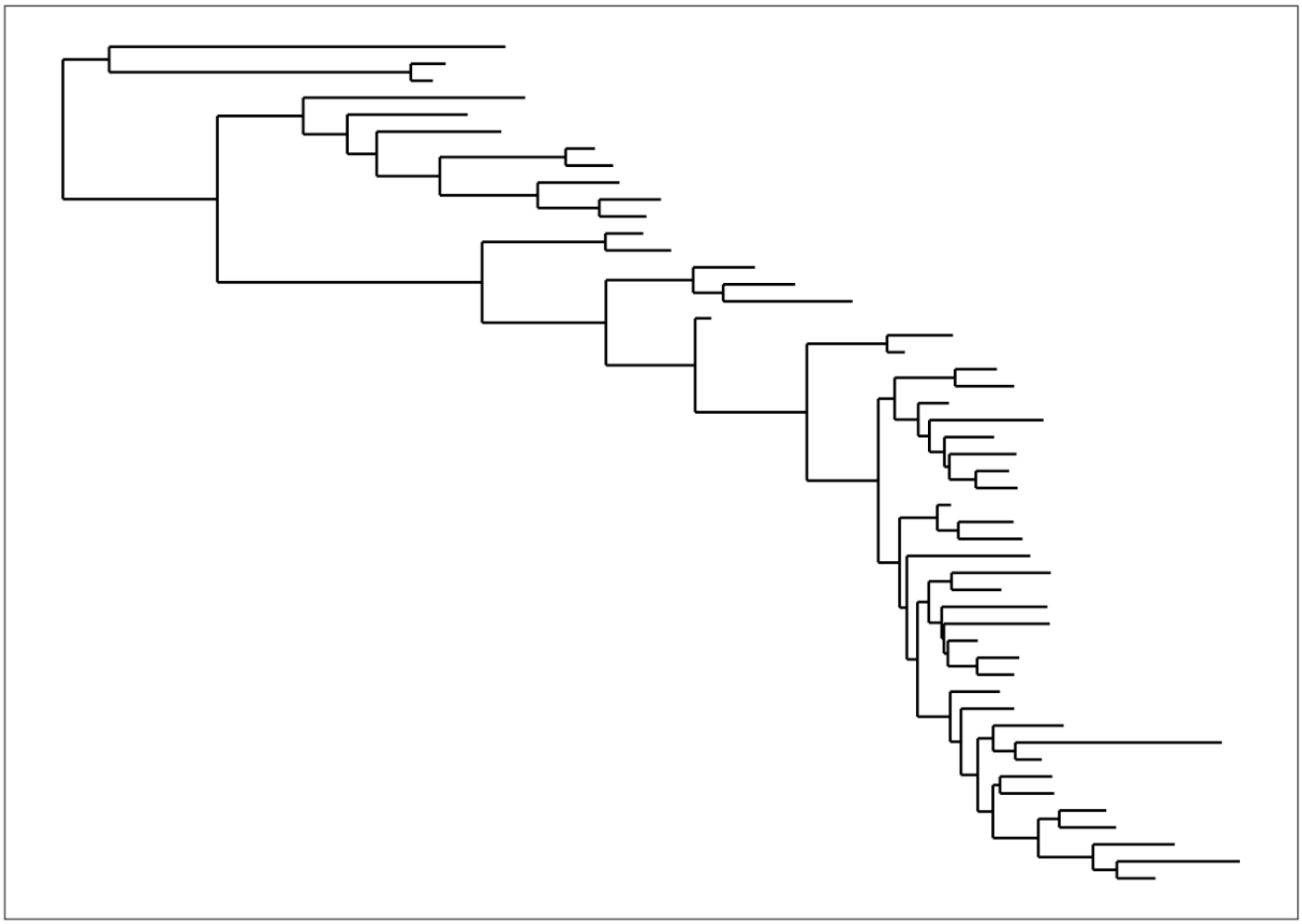


We can adjust the way the branches are rotated to improve the looks using the `ladderize` parameter.

```
# Ladderize the tree above

plot_tree(physeq, "treeonly", nodeplotblank, ladderize="left")
```

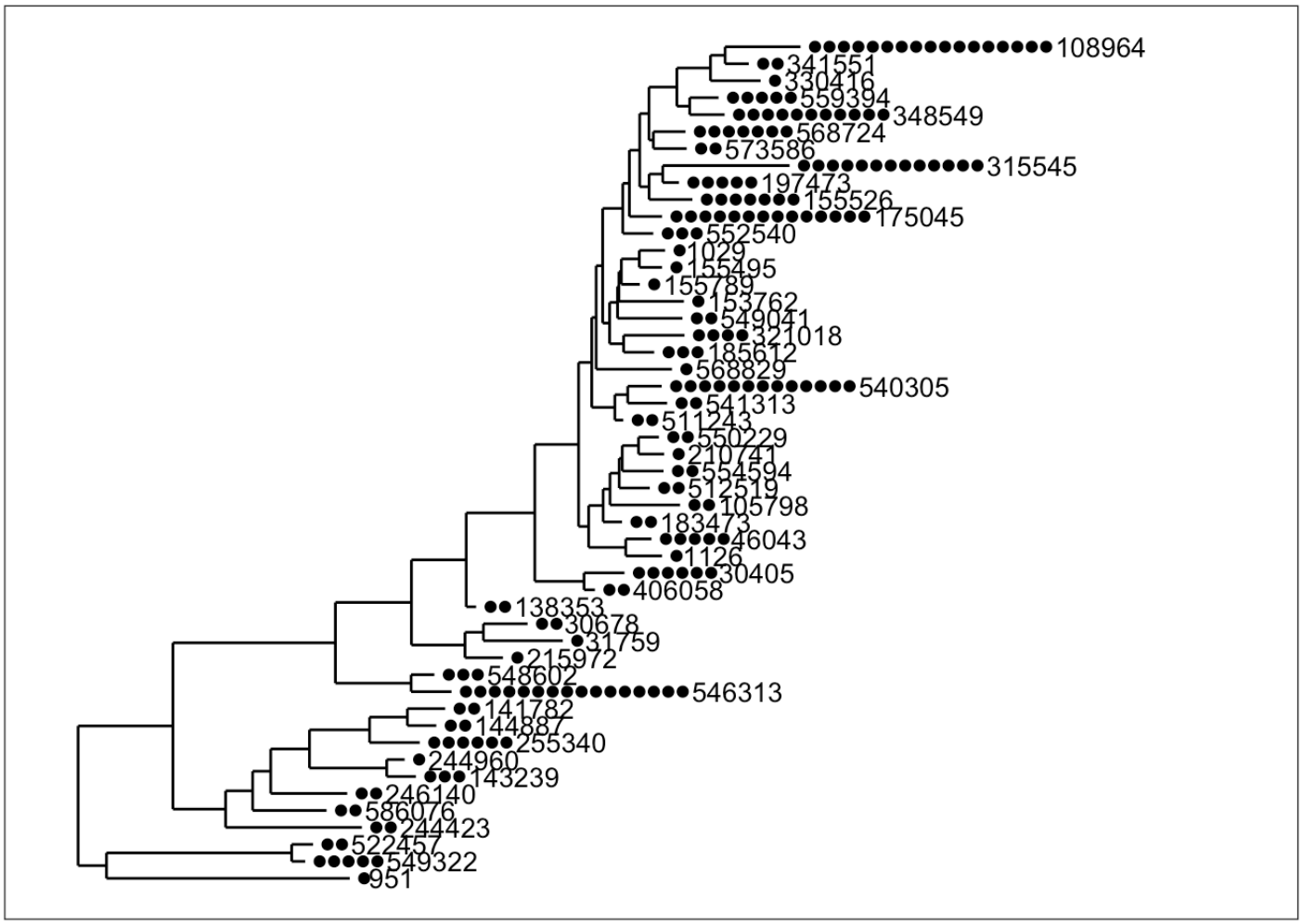


```
plot_tree(physeq, "treeonly", nodeplotblank, ladderize = TRUE)
```



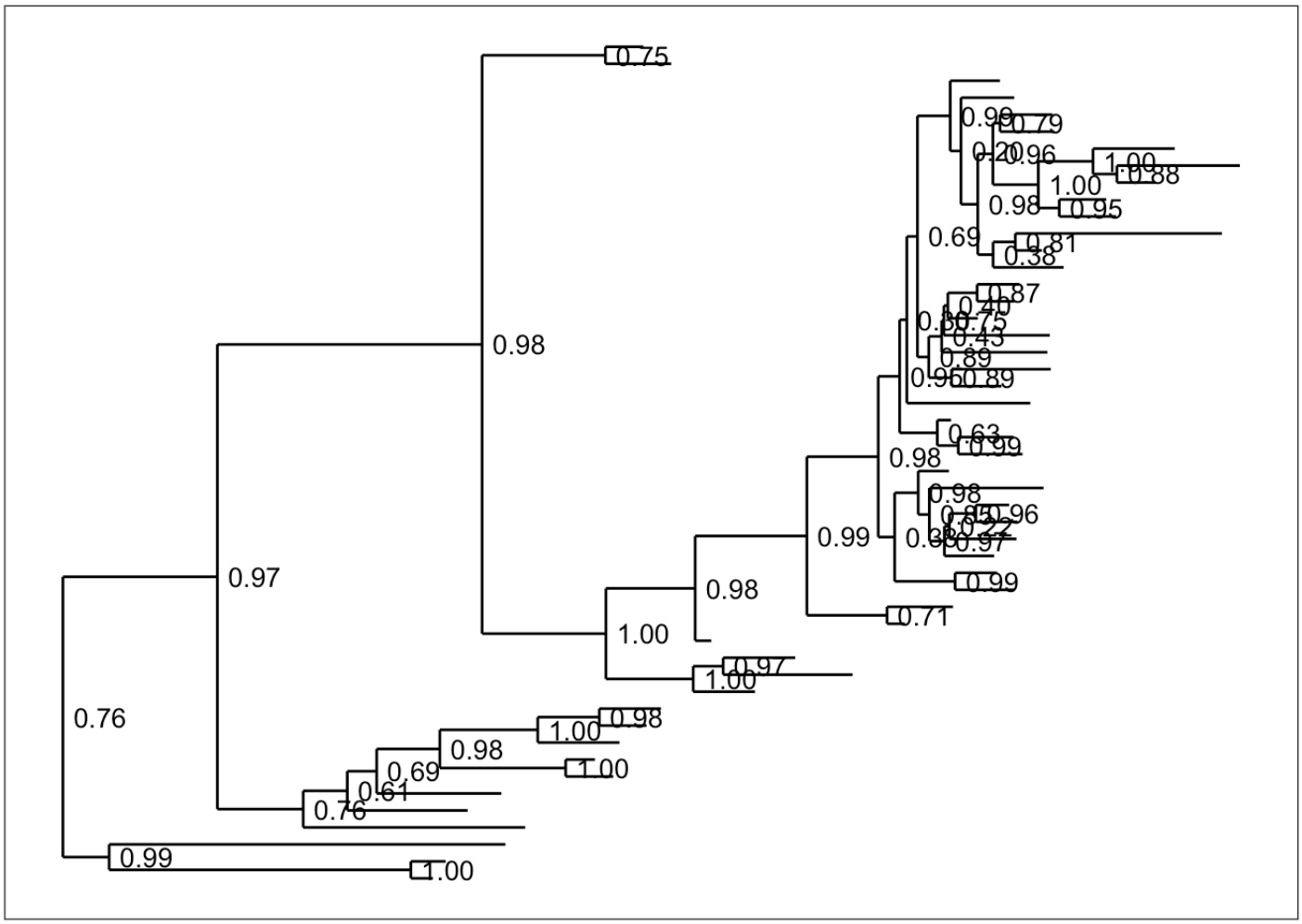
Now let's add the OTU labels back to the tips?

```
# Add tip labels back to tree above

plot_tree(physeq, nodelabf=nodeplotblank, label.tips="taxa_names", ladderize="left")
```



Whoa, we also added back the points. Let's take those away.

```
# Tree above with labels, without points

plot_tree(physeq, "anythingelse")
```



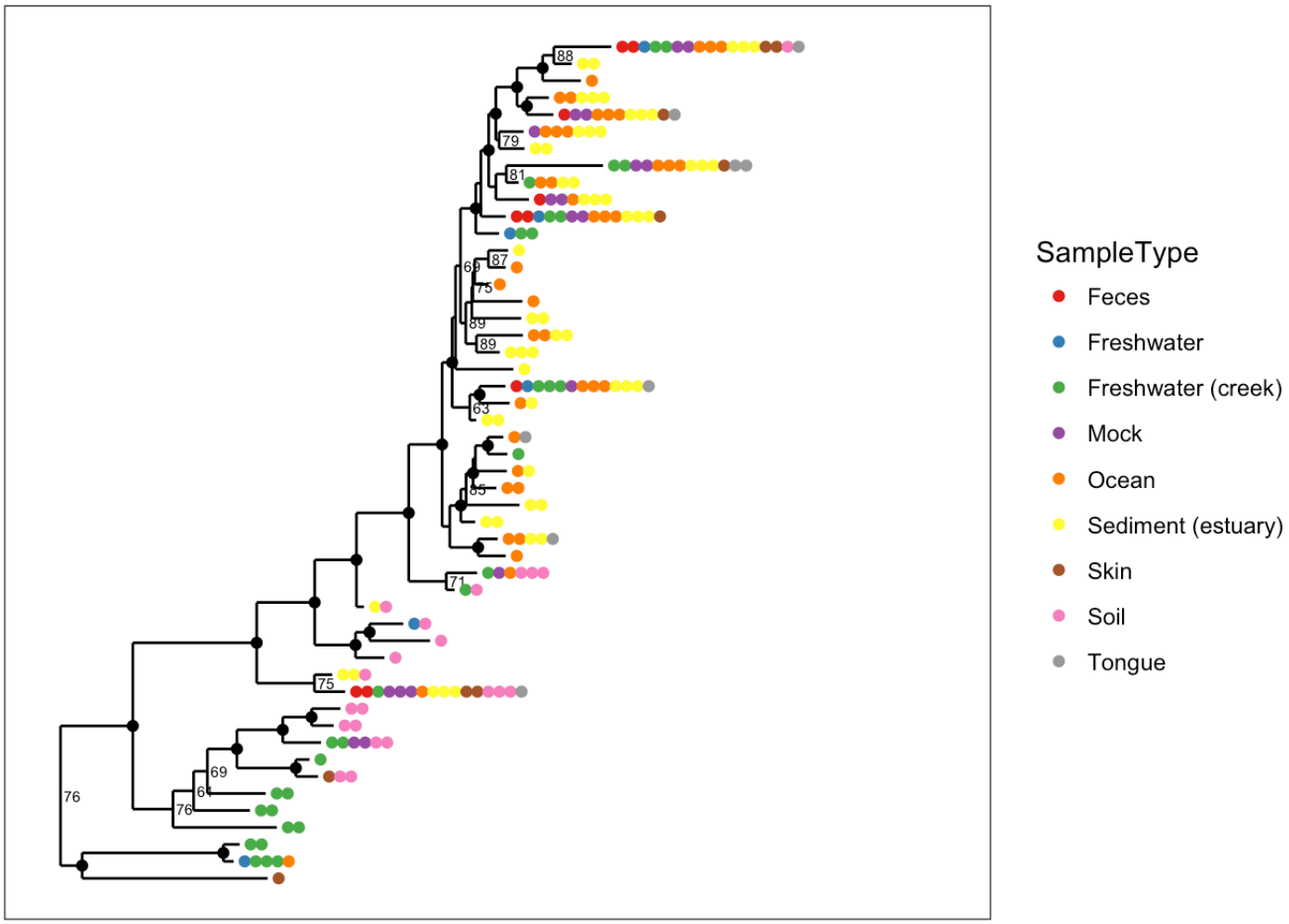
Mapping Variables in Data

For the default argument to the method parameter, “sampledodge”, a point is added next to each OTU tip for every sample in which that OTU was observed. Now we can add aesthetics to these points.

Add Color

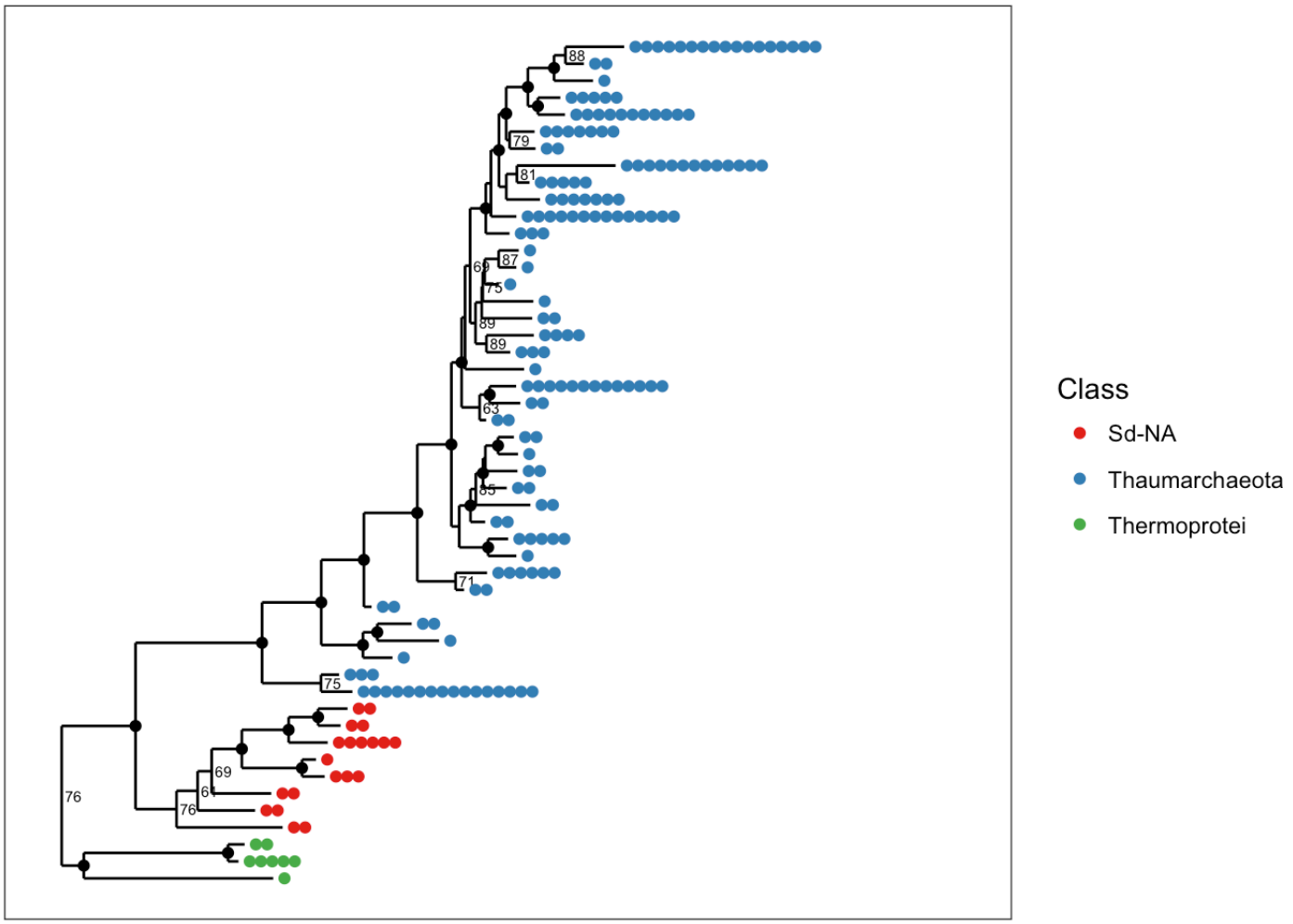
Let’s map color to the type of sample collected (location)

```
# Map color to type of sample collected

plot_tree(physeq, nodelabf = nodeplotboot(), ladderize = "left", color = "SampleType"
)
```



Or we can color according to the taxonomic level of Class

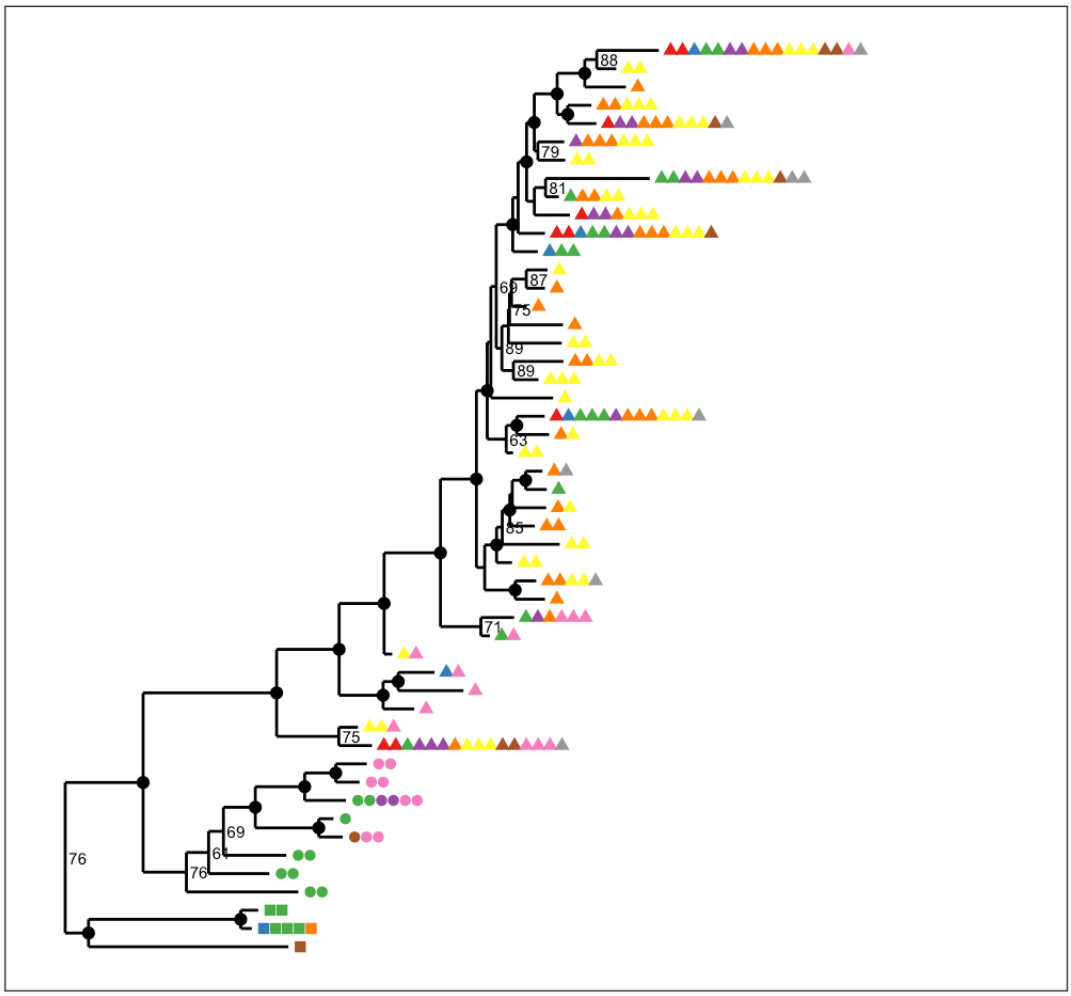
```
# Color according to taxonomic Class

plot_tree(physeq, nodelabf = nodeplotboot(), ladderize = "left", color = "Class")
```



Shape

We can also map to a point shape if the variable has 6 or less categories, even when color is also mapped. Let's add point shape to indicate taxonomic Class while color indicates SampleType.

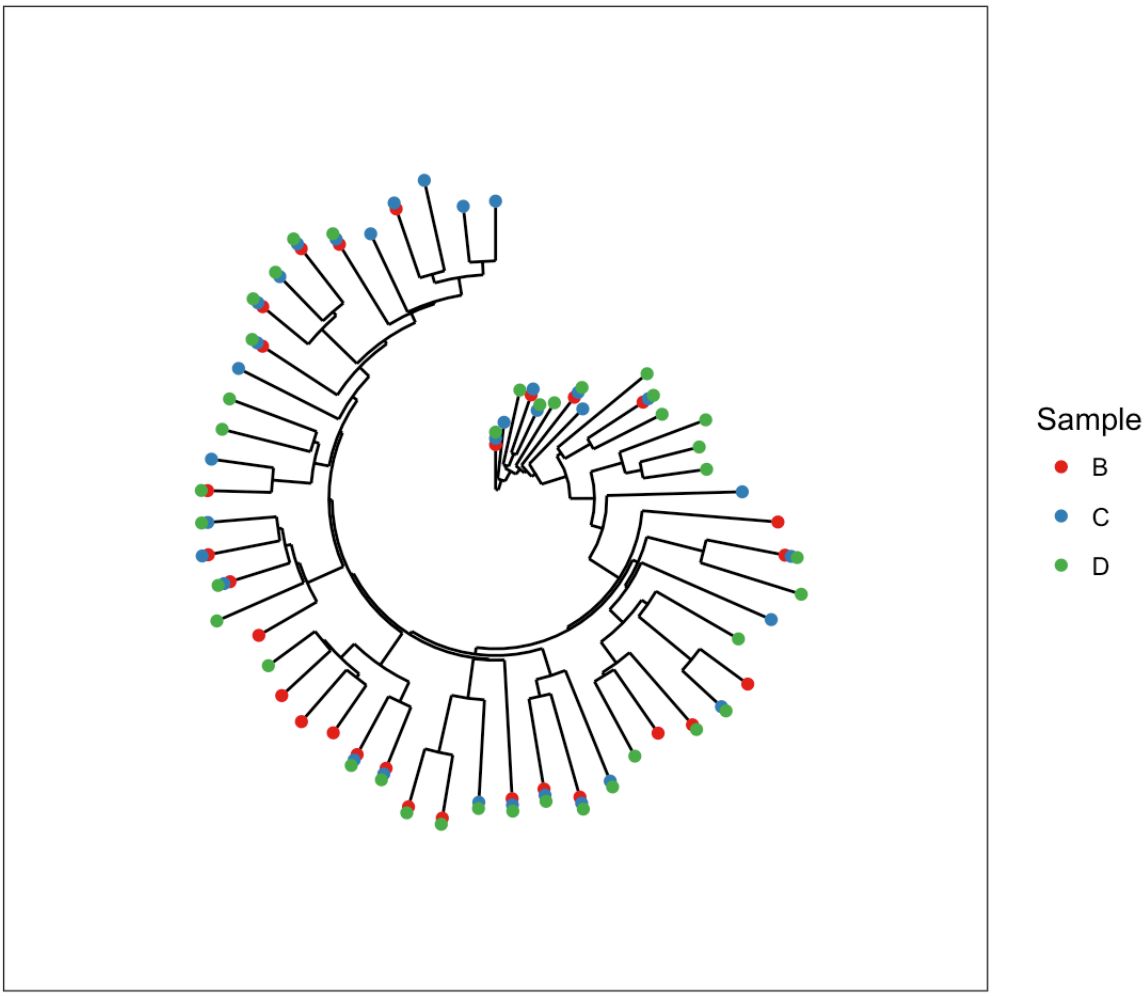
```
# add shape to points

plot_tree(physeq, nodelabf = nodeplotboot(), ladderize = "left", color = "SampleType"
, shape = "Class")
```



Radial Tree

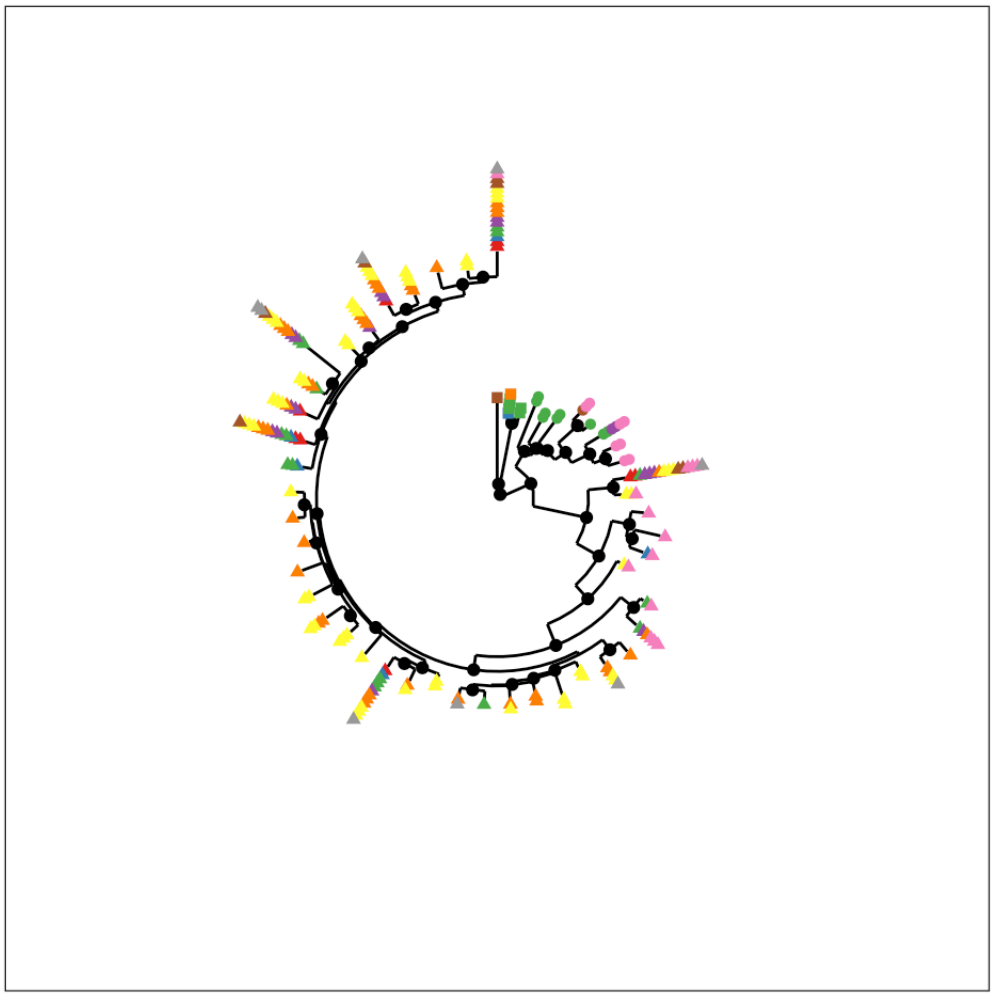
Making a radial tree is surprisingly easy with `ggplot2`, as long as you remember that our vertically oriented tree is a cartesian mapping of the data to a graphic. A radial tree is the same mapping, but with polar, instead of cartesian, coordinates. To wit

```
# Make radial phylogenetic tree for esophagus dataset

data (esophagus)
plot_tree(esophagus, color = "Sample", ladderize = "left") + coord_polar(theta = "y")
```



We can make a more elaborate tree with the GlobalPatterns dataset, but will need to do some preliminary loading and trimming.

```
# Make radial tree for GlobalPatterns

plot_tree(physeq, nodelabf = nodeplotboot(60, 60, 3), color = "SampleType", shape = "
Class", ladderize = "left") + coord_polar(theta = "y")
```



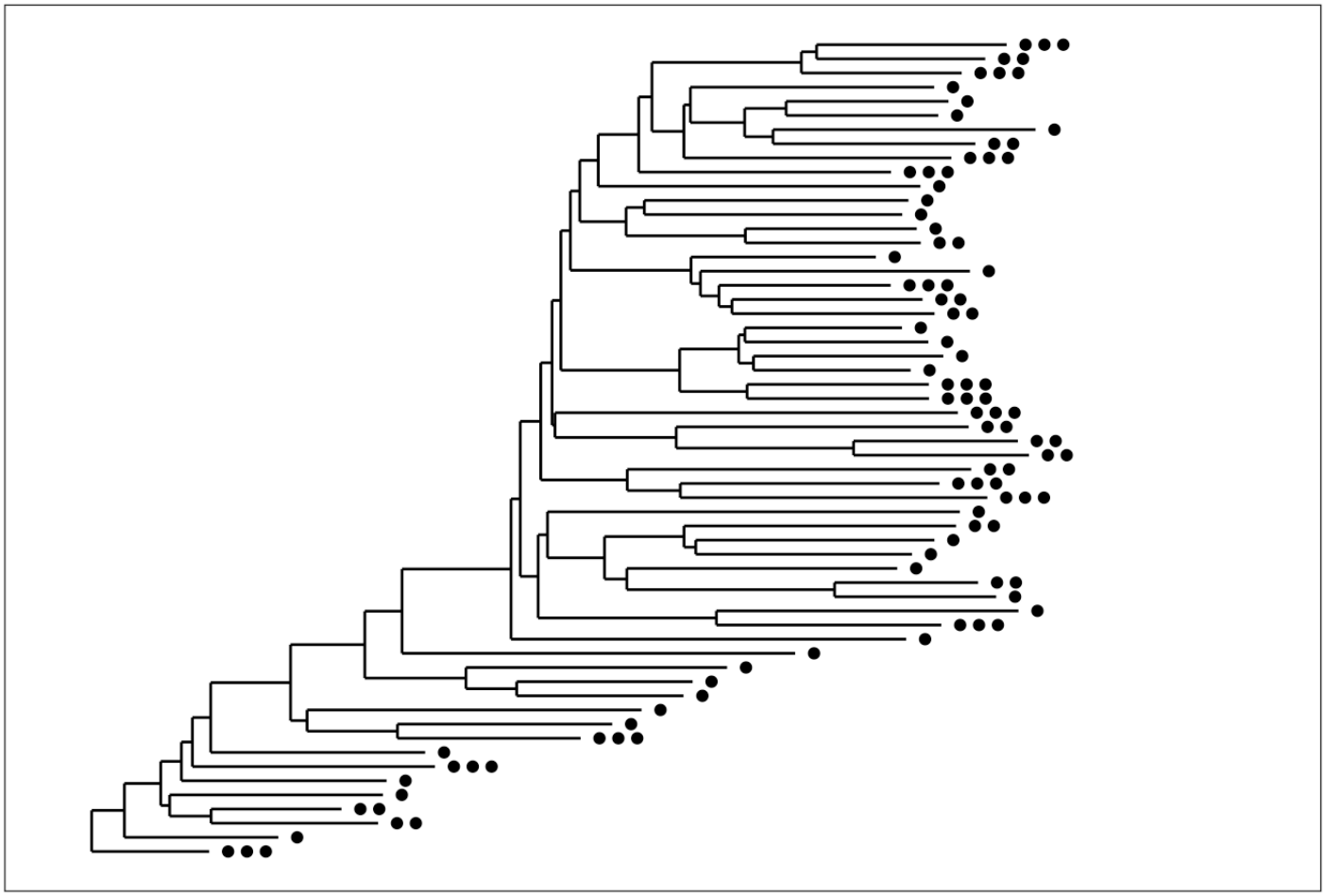
More Examples with the Esophagus Dataset

The Esophagus dataset is really simple - just 3 samples, no sample data, and a modest amount of sequencing per sample.

Let's look at the default tree

```
# Default tree for Esophagus dataset

plot_tree(esophagus, title="Default Tree.")
```

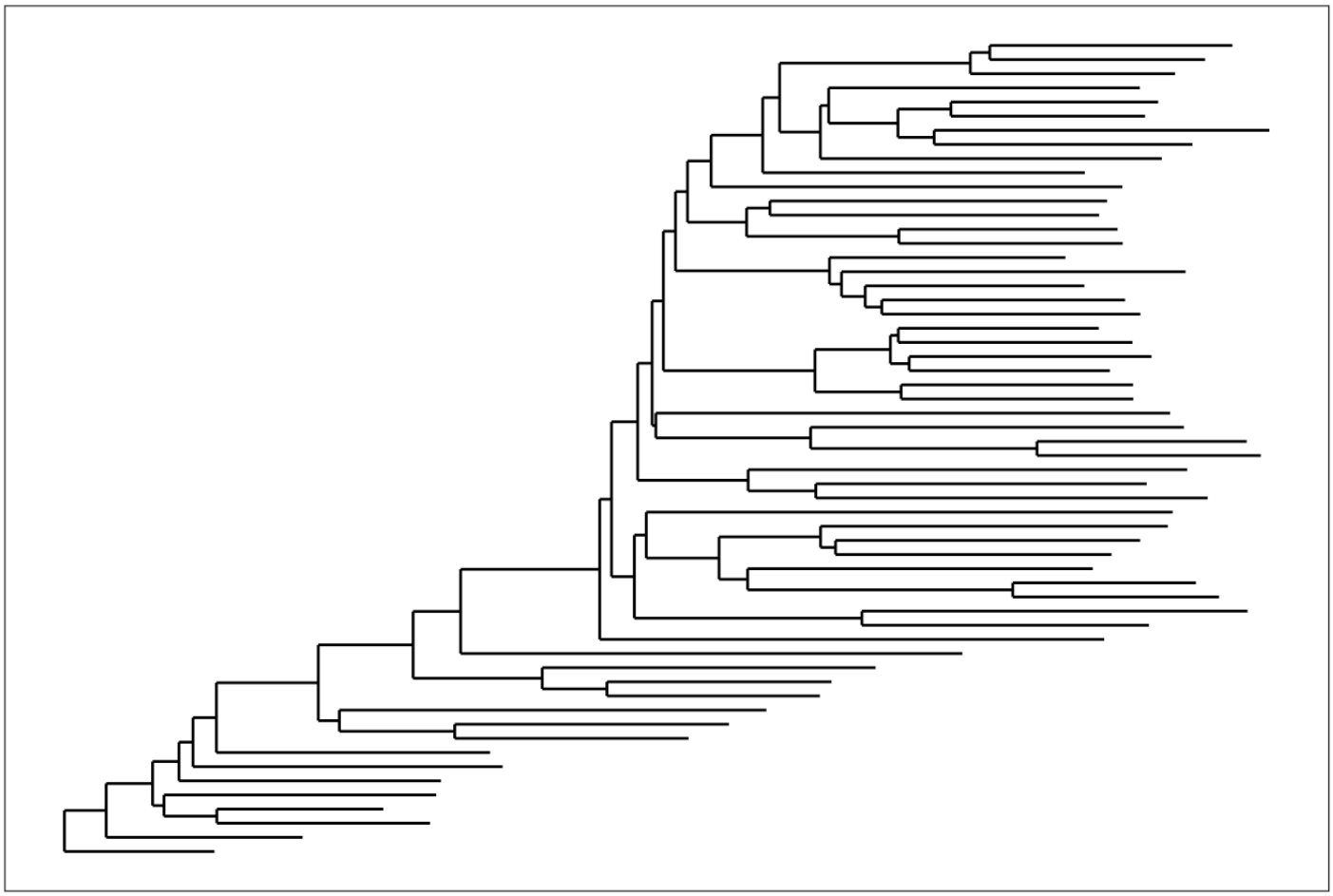
Default Tree.



If you want an unadorned tree, the `treeonly` method can be used.

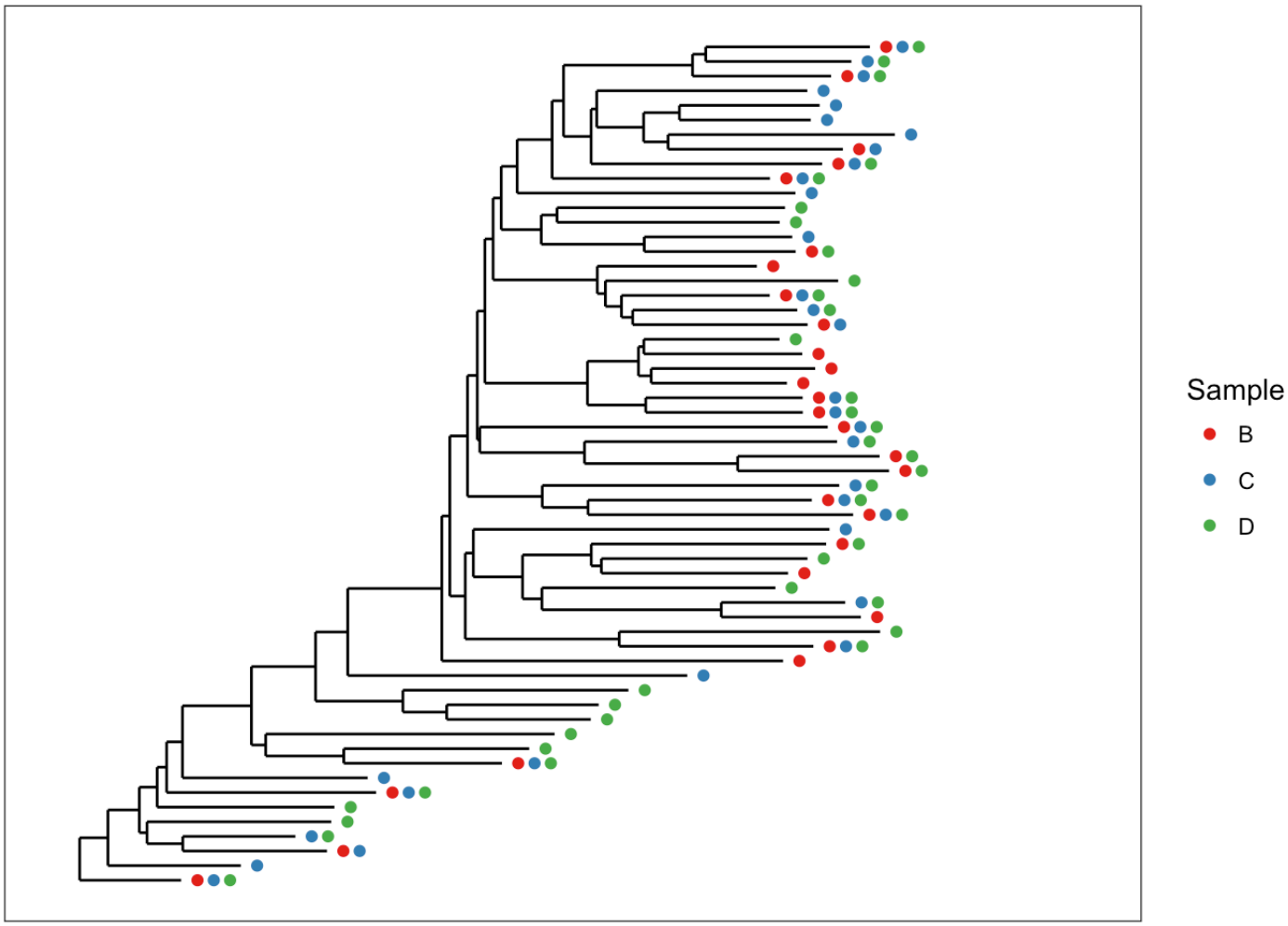
```
# Simplest tree

plot_tree(esophagus, "treeonly", title="method = \"treeonly\"")
```

method = "treeonly"



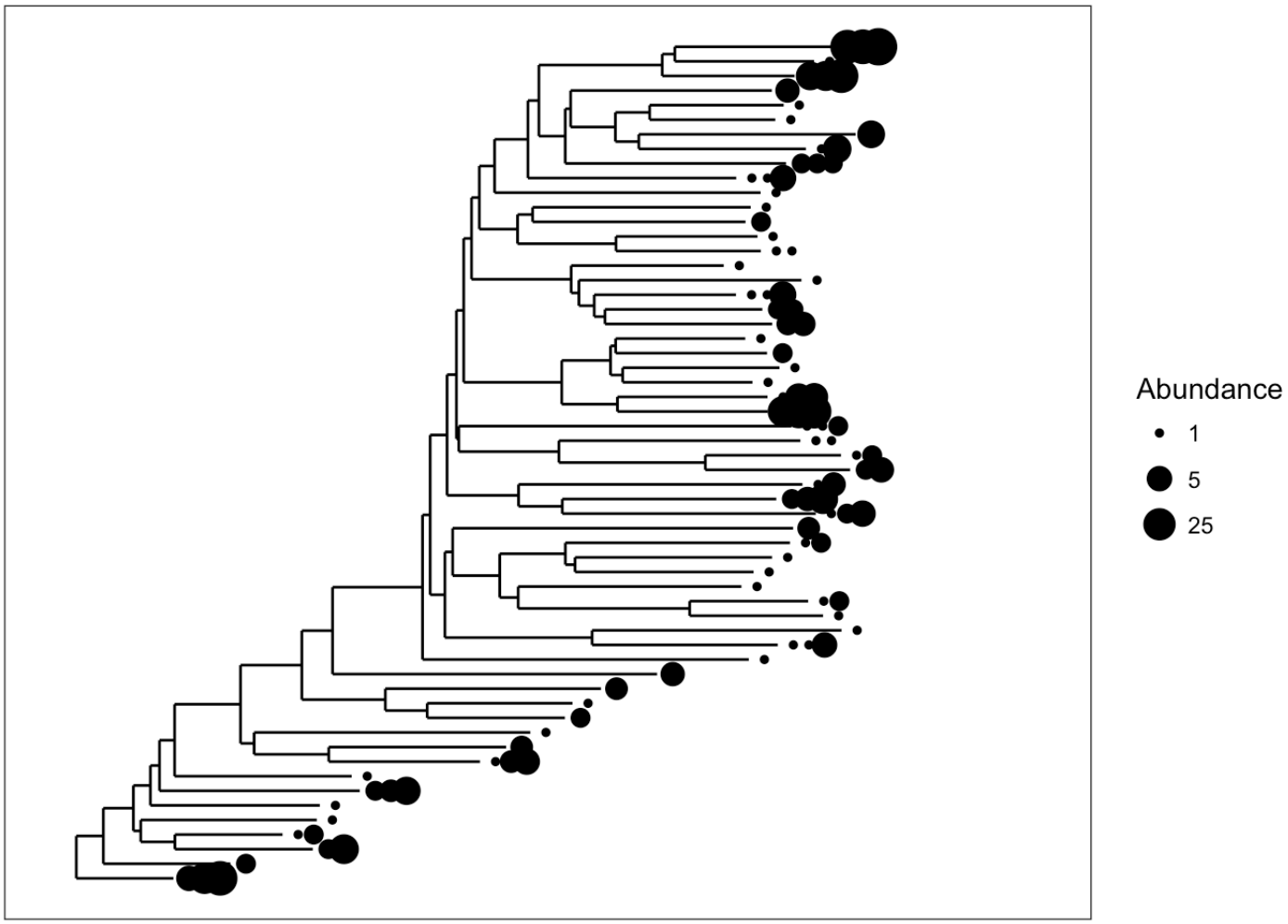
Shading tips where OTUs are observed:

```
# Shade tips where OTUs are observed

plot_tree(esophagus, color="samples")
```



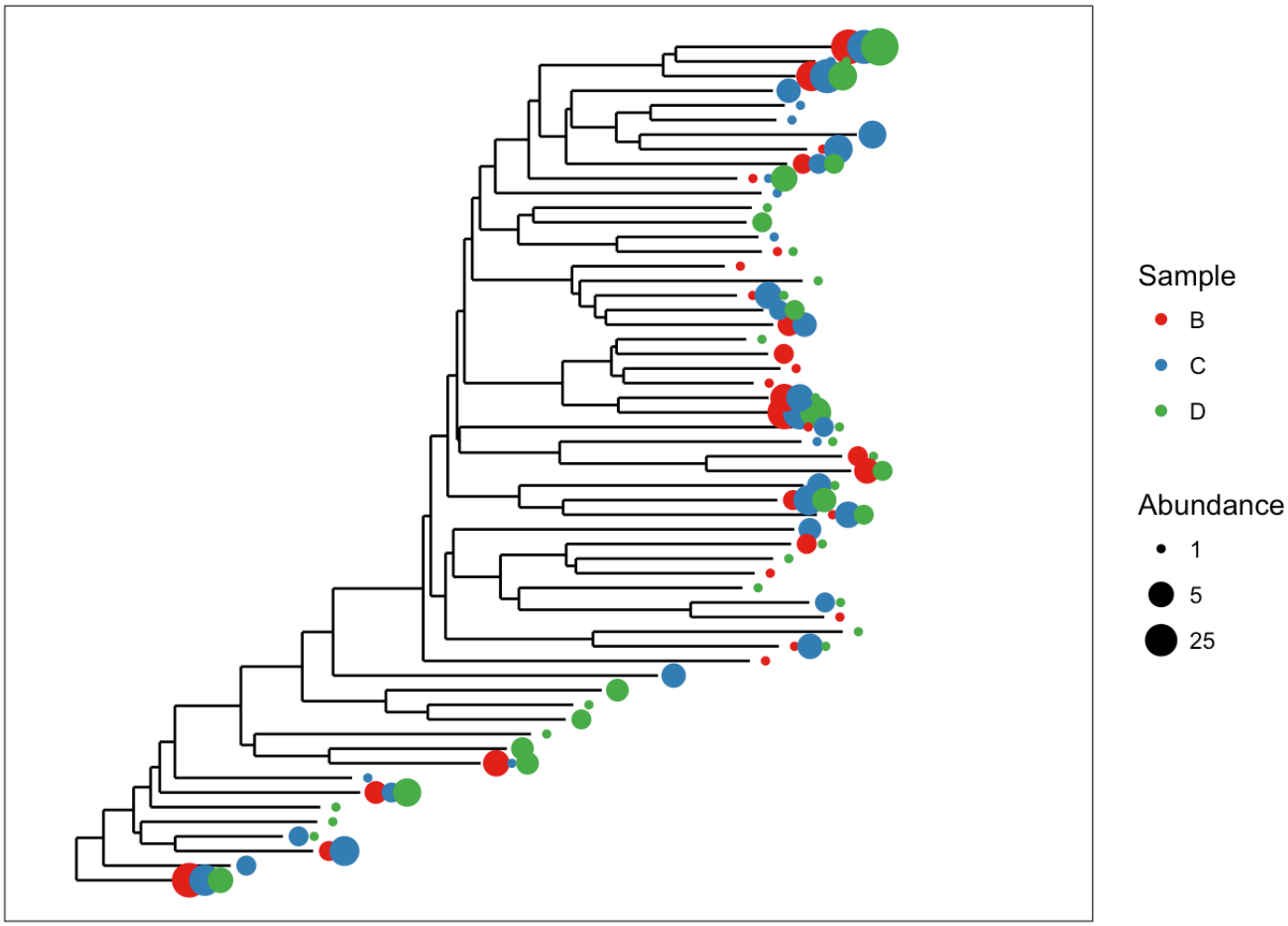
Scale the tips according to abundance.

```
# Scale tips by abundance

plot_tree(esophagus, size="abundance")
```



Color and scale on the same tree!

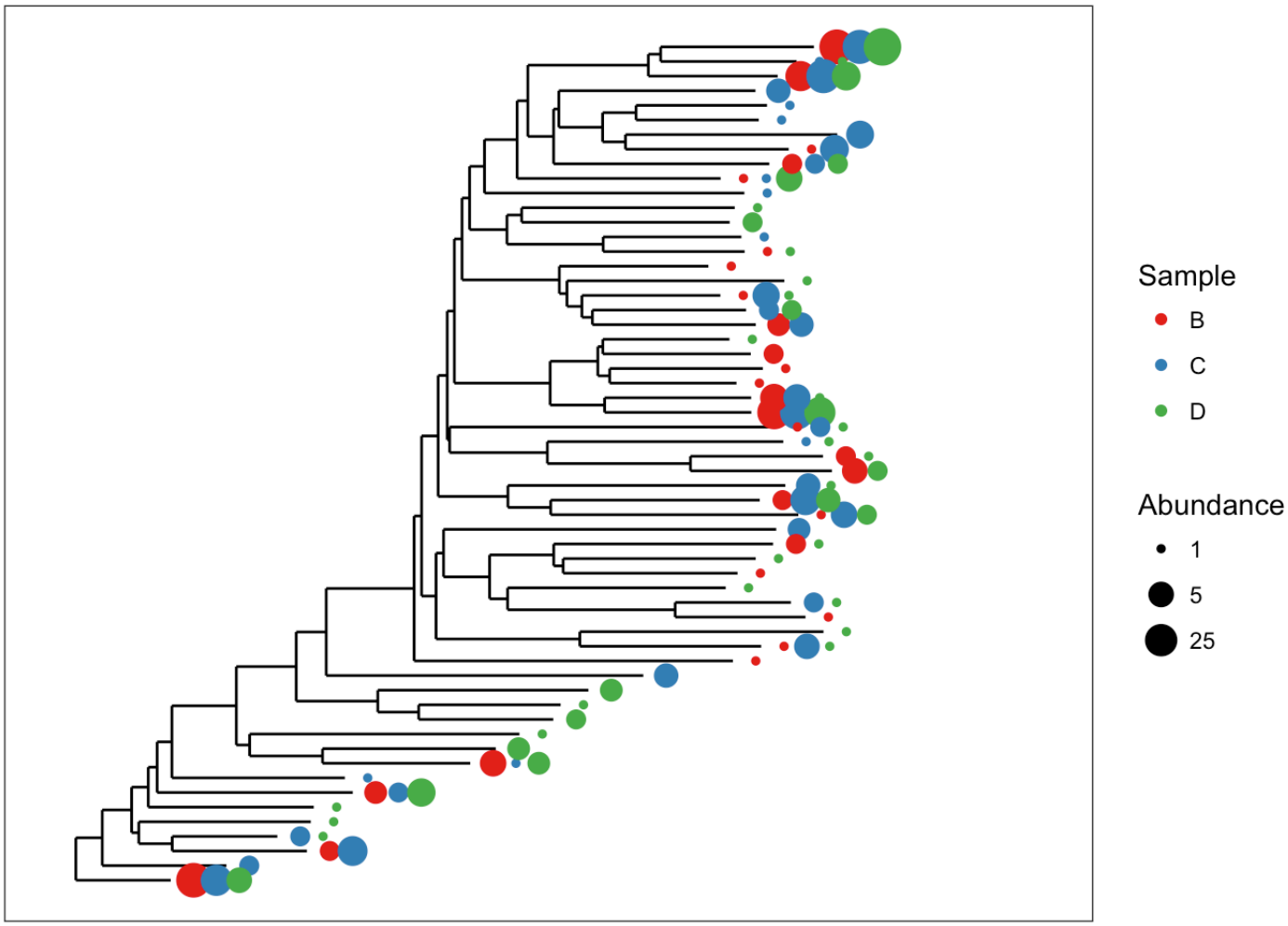
```
# Shade tips where OTUs are observed

plot_tree(esophagus, color="samples", size = "abundance")
```



Oh my!

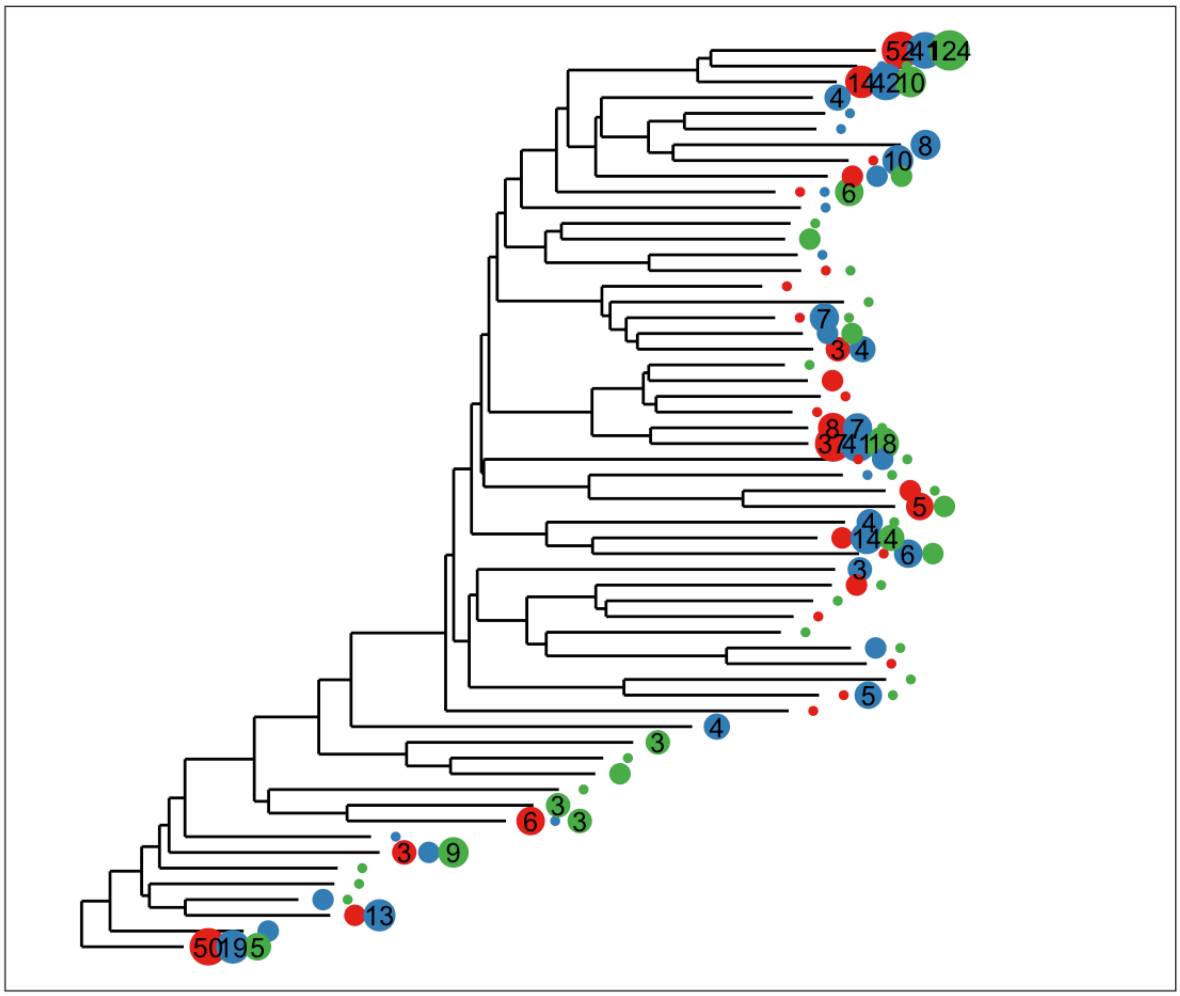
There is some overlap of tip points. Let's see if we can adjust spacing to spread them out a bit.

```
# Shade tips where OTUs are observed

plot_tree(esophagus, color="samples", size = "abundance", base.spacing = 0.03)
```



Can we display the actual numeric value of abundances that occurred 3 or more times in a sample?

```
# Shade tips where OTUs are observed

plot_tree(esophagus, color="samples", size = "abundance", base.spacing = 0.03, min.ab
undance = 3)
```



More Examples with GlobalPatterns Dataset

Let's subset the GlobalPatterns dataset to just the observed Archaea.

```
# Subset GlobalPatterns to just Archaea

gpa <- subset_taxa(GlobalPatterns, Kingdom == "Archaea")
ntaxa(gpa)
```

```
## [1] 208
```

The number of taxa is small enough for a decent tree plot.

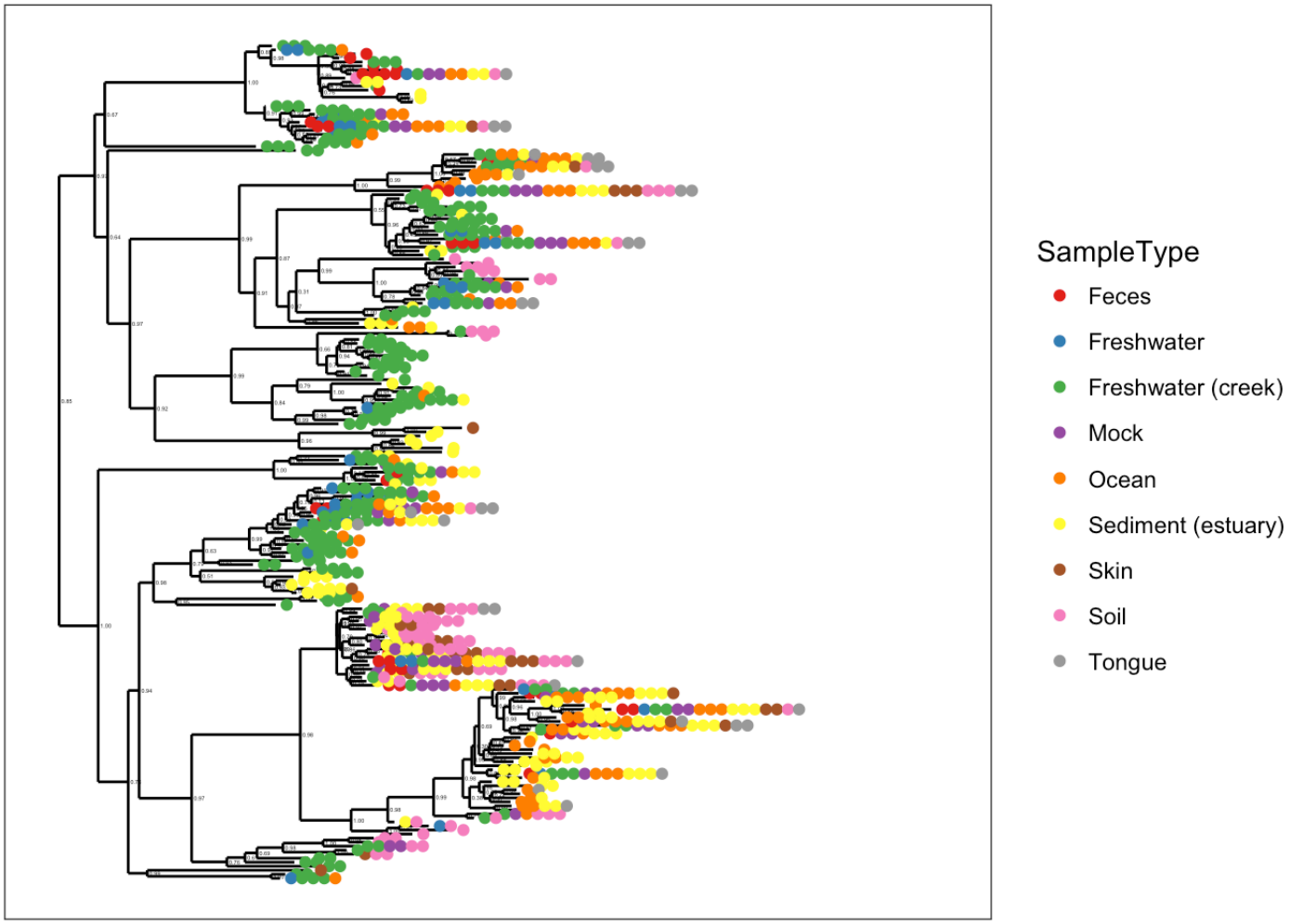
Too many OTUs means a tree that is pointless to displa in its entirety in one graphic of a standard size and font. So the whole GlobalPatterns dataset is probably a bad idea.

```
# Number of taxa in GlobalPatterns dataset

ntaxa(GlobalPatterns)
```

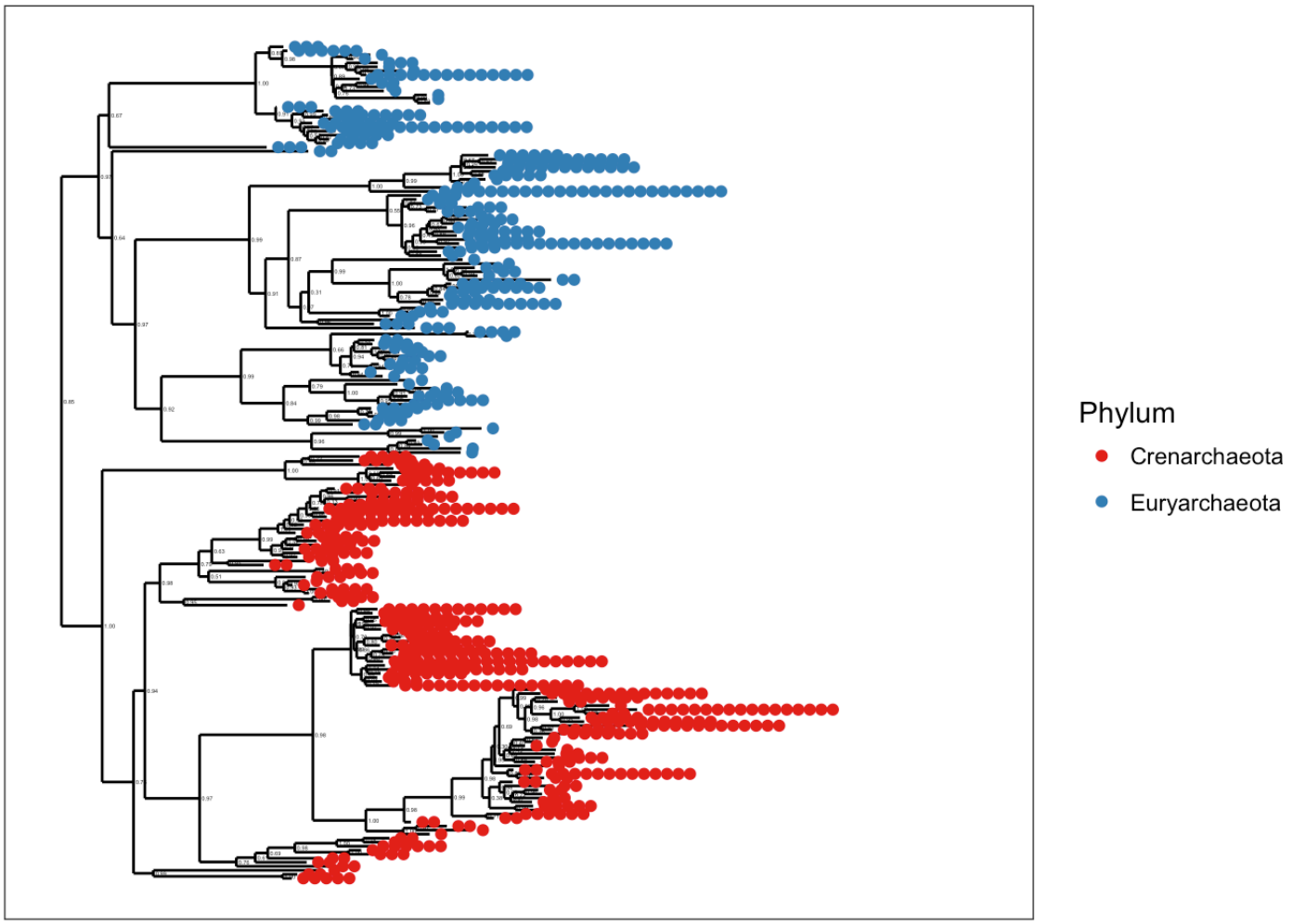
```
## [1] 19216
```

Let's look at the basic tree with just a minimal set of parameter choices.

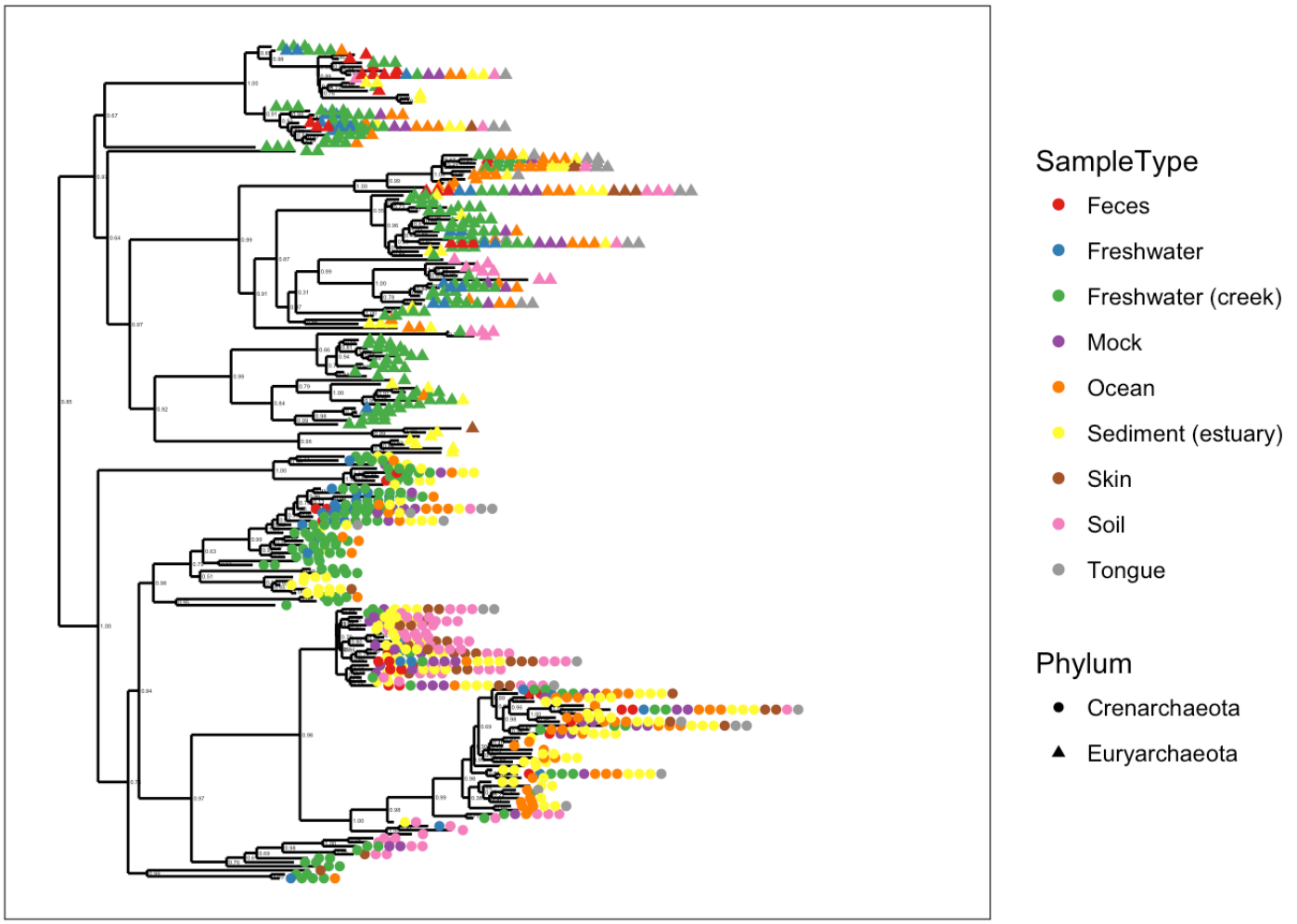
```
# Almost minimal trees for GlobalPatterns Archaea subset

plot_tree(gpa, color = "SampleType")
```



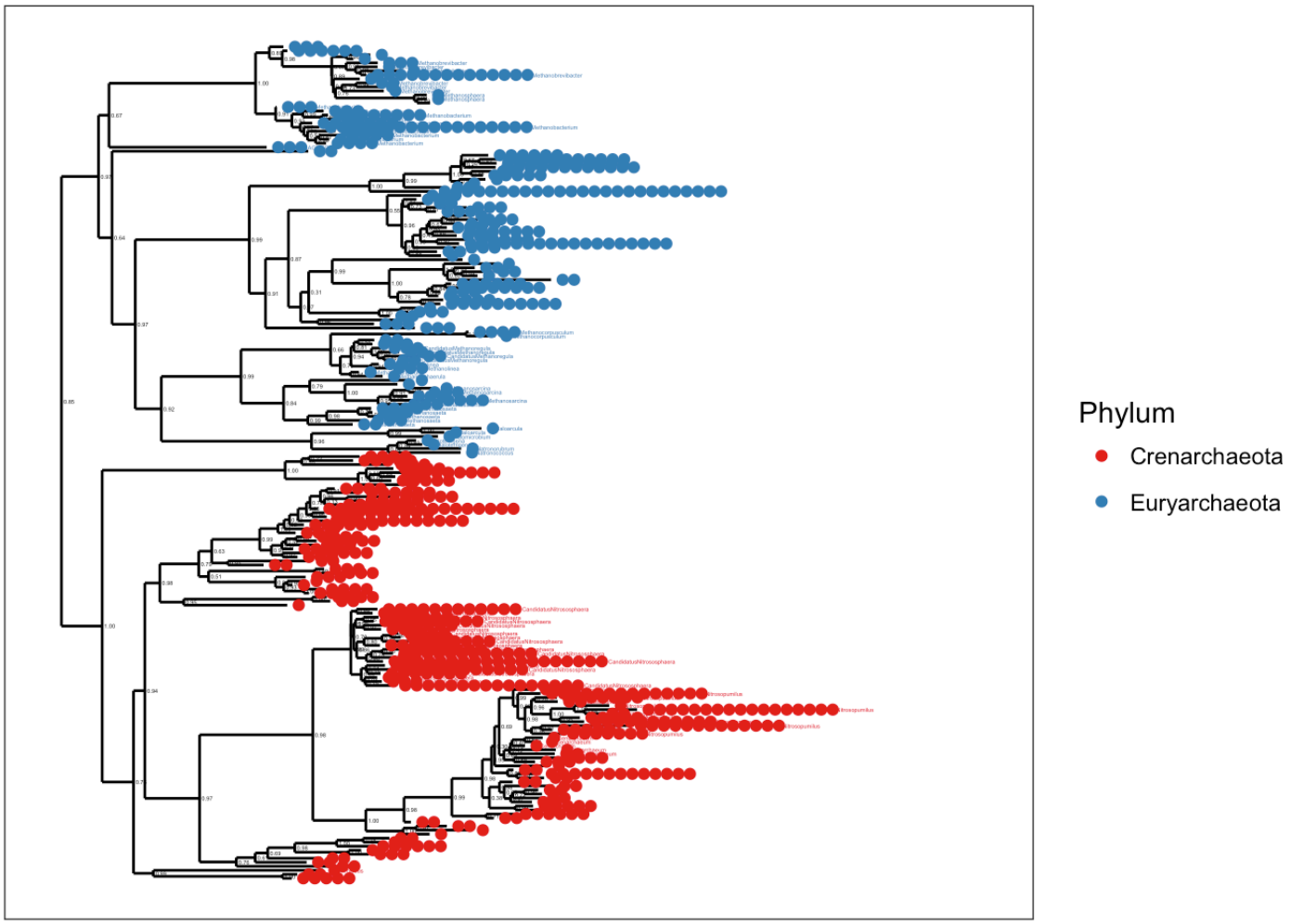

```
plot_tree(gpa, color = "Phylum")
```




```
plot_tree(gpa, color = "SampleType", shape = "Phylum")
```

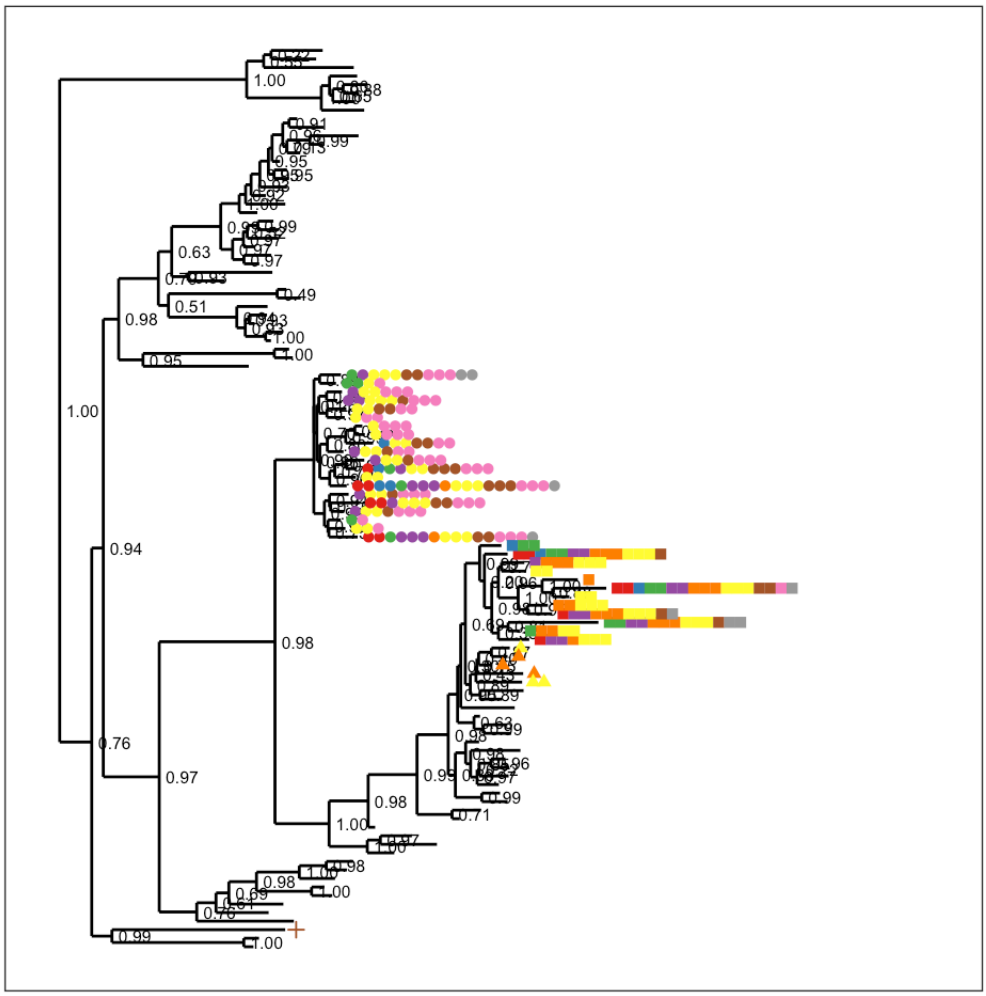



```
plot_tree(gpa, color = "Phylum", label.tips = "Genus")
```

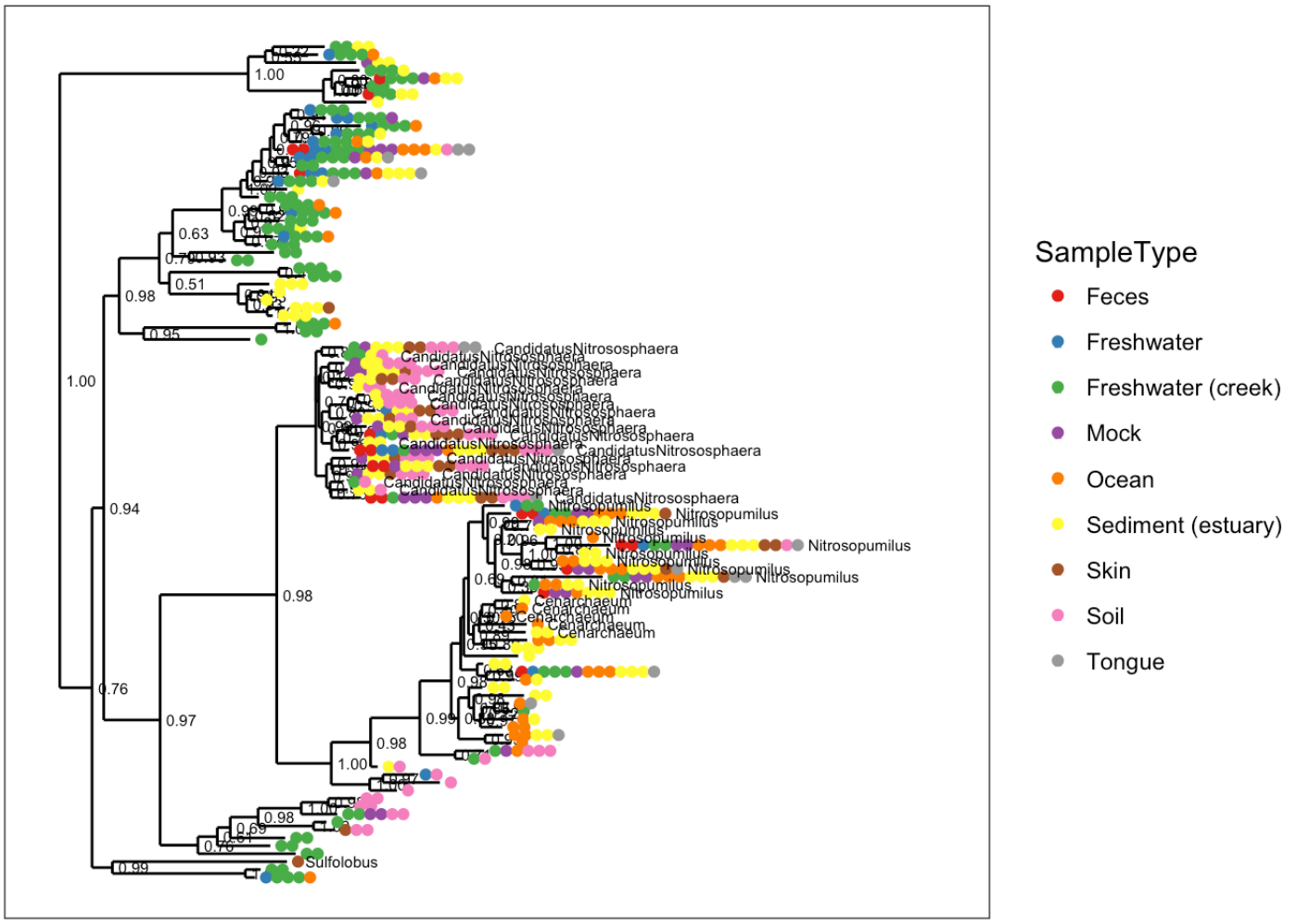


These ~200 taxa still make for a crowded graphic. So let's subset further to just the Crenarcheota.

```
# Trees for just Crenarchaeota

gpac <- subset_taxa(gpa, Phylum=="Crenarchaeota")
plot_tree(gpac, color="SampleType", shape="Genus")
```

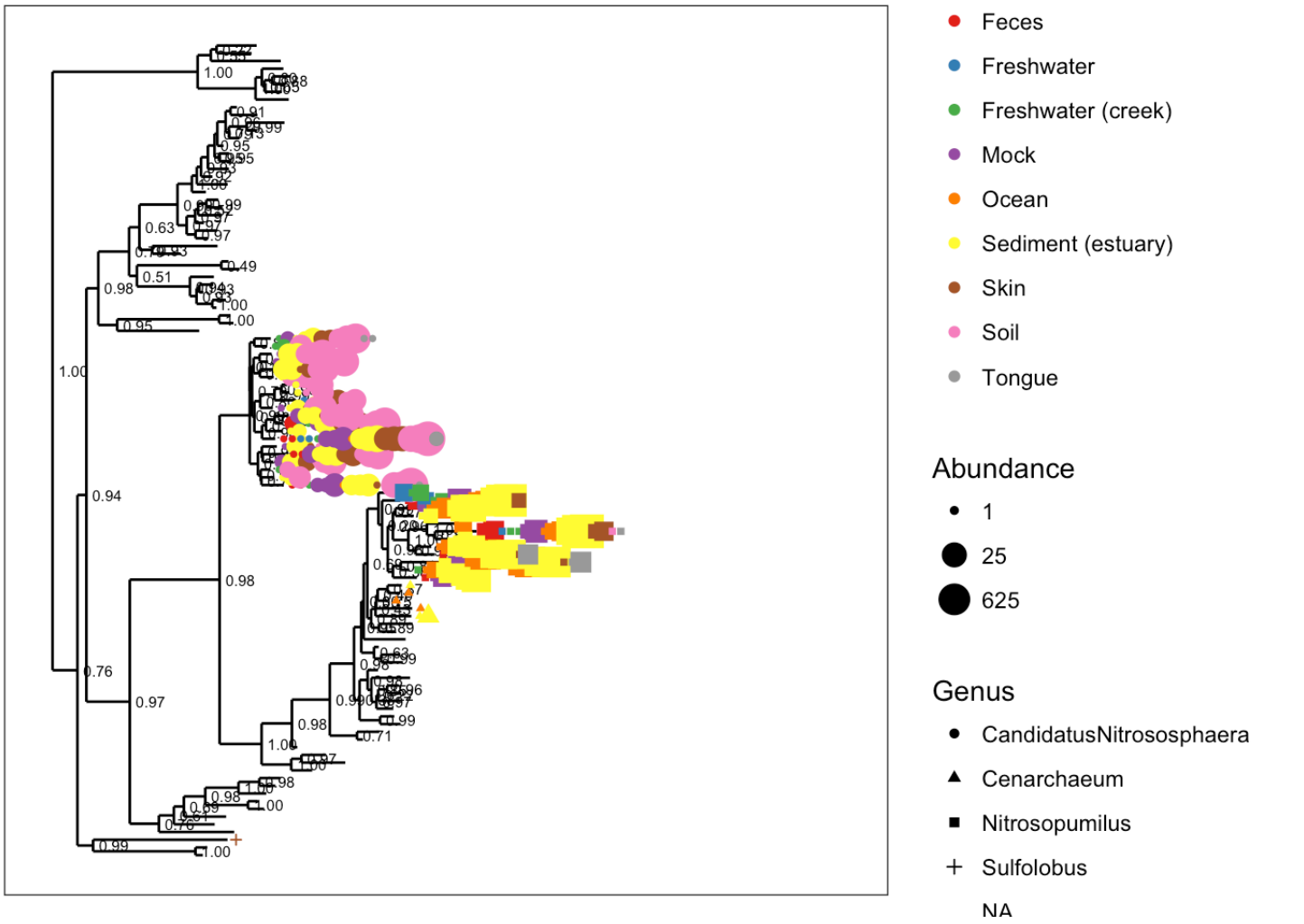



```
plot_tree(gpac, color = "SampleType", label.tips = "Genus")
```

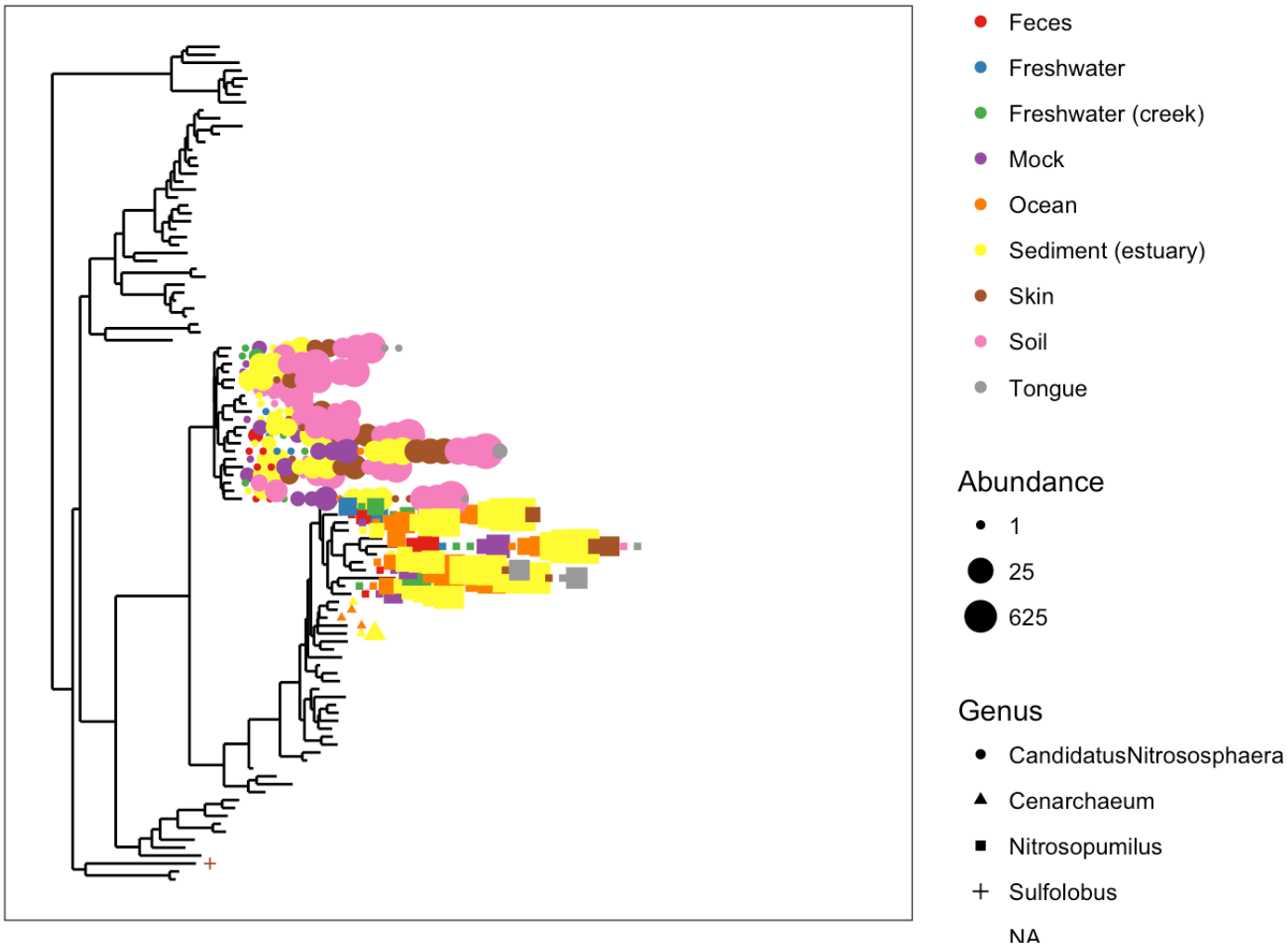


Add some abundance information, then spread it out:

```
# Add abundance inforamtion then spread it out

plot_tree(gpac, color="SampleType", shape="Genus", size="abundance", plot.margin=0.4)
```



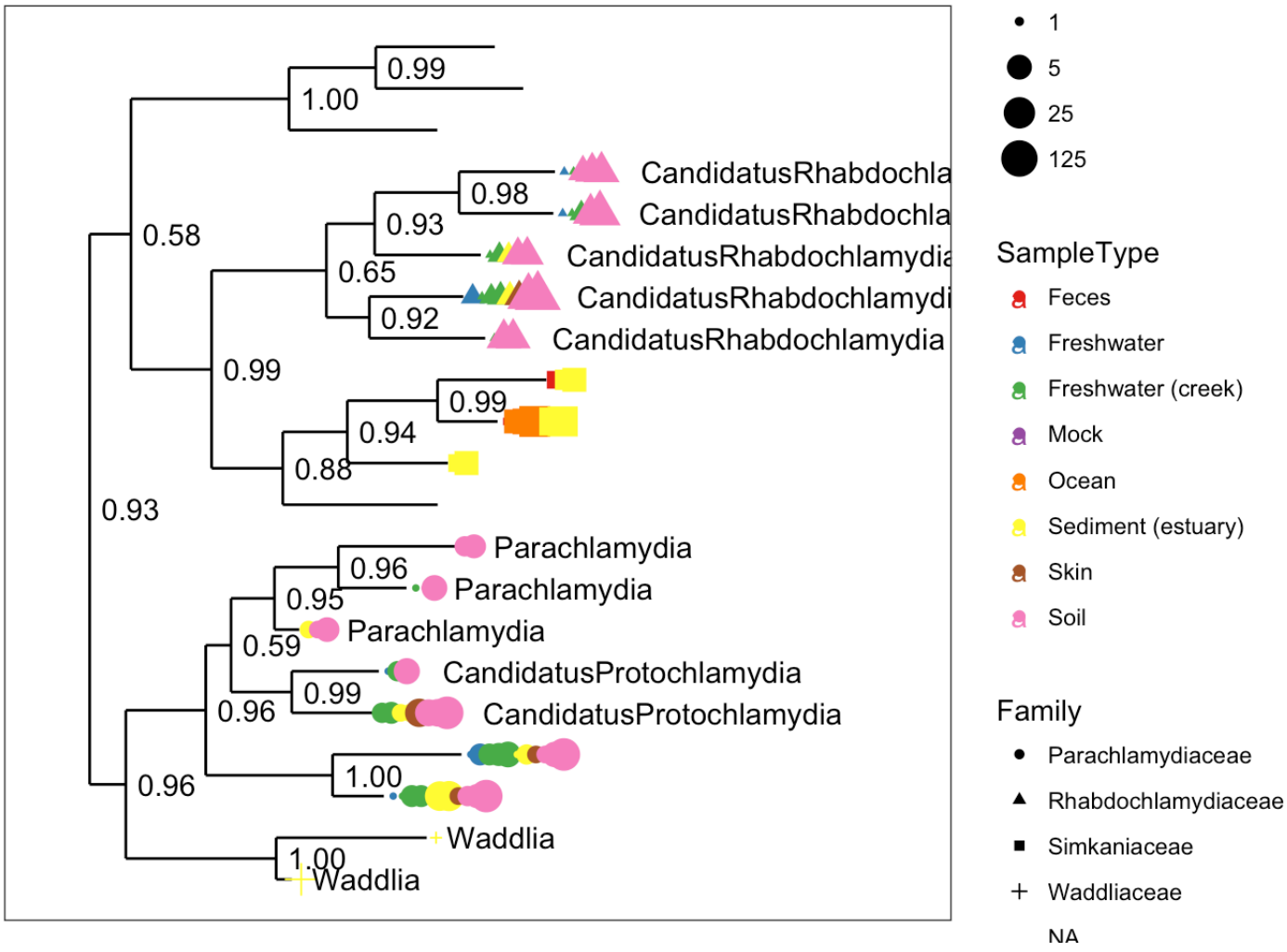

```
plot_tree(gpac, nodelabf=nodeplotblank, color="SampleType", shape="Genus", size="abundance", base.spacing=0.04, plot.margin=0.4)
```



Chlamydiae-only Tree

Let's use some of these same techniquest with the Chlamydiae-only tree that we had created earlier.

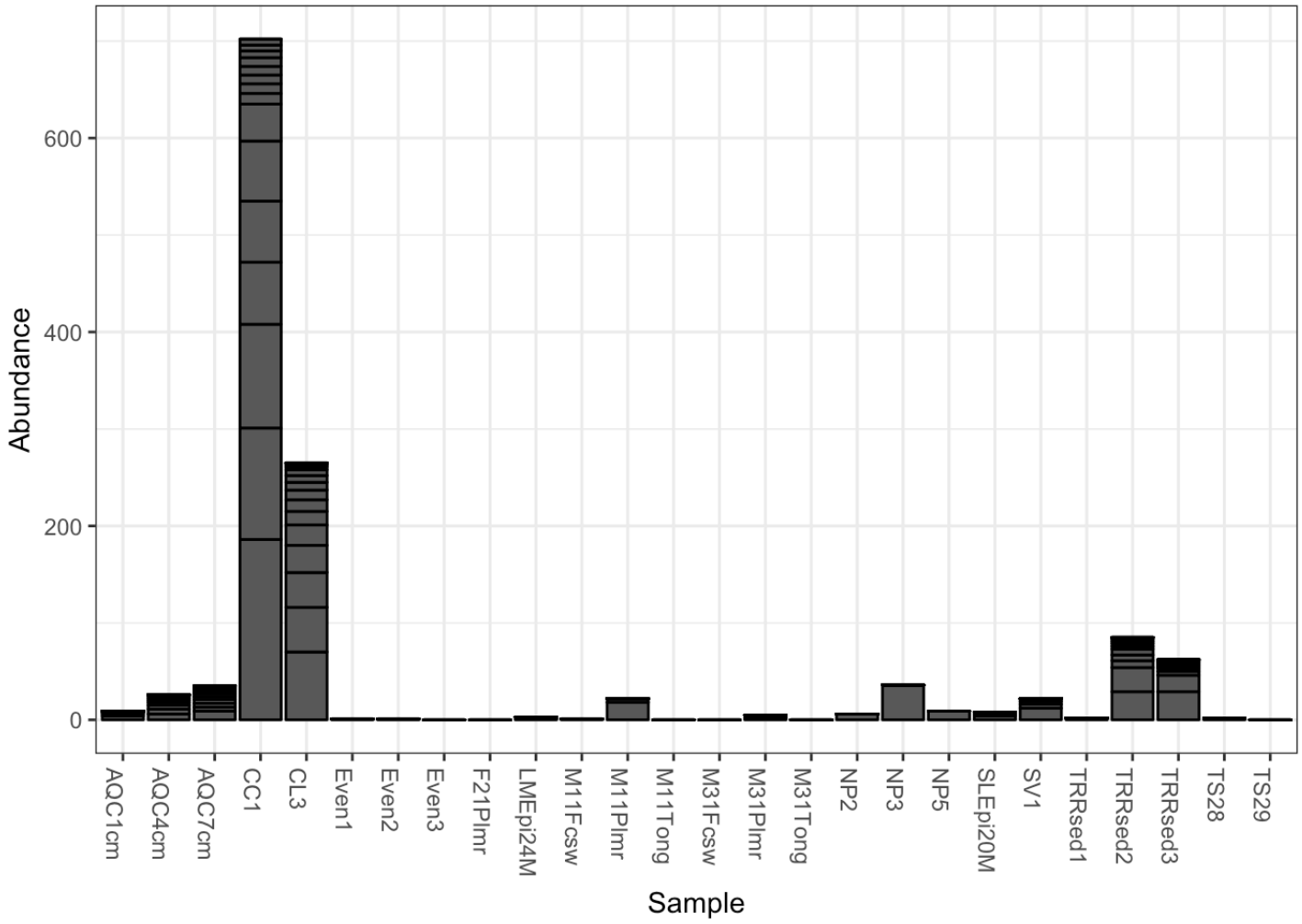
```
# Create Chlamydiae-only tree and plot again

gp.ch <- subset_taxa(GlobalPatterns, Phylum=="Chlamydiae")
plot_tree(gp.ch, color="SampleType", shape="Family", label.tips="Genus", size="abunda
nce", plot.margin=0.6)
```



Bar Plots

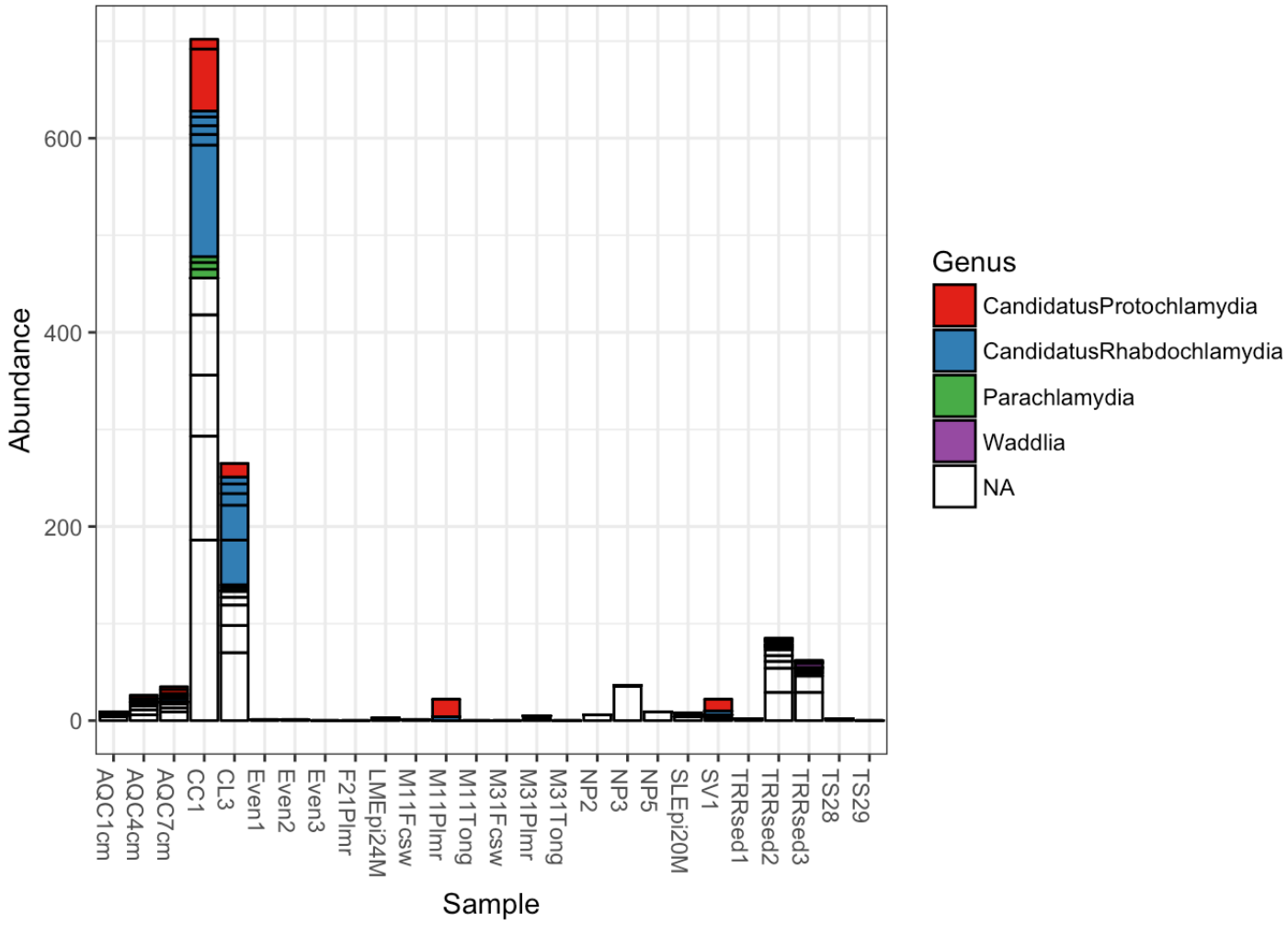
Let's start again with the Chlamydiae-only subset of GlobalPatterns, and look at the basic bar plot.

```
# Basic plots with Chlamydiae subset

plot_bar(gp.ch)
```



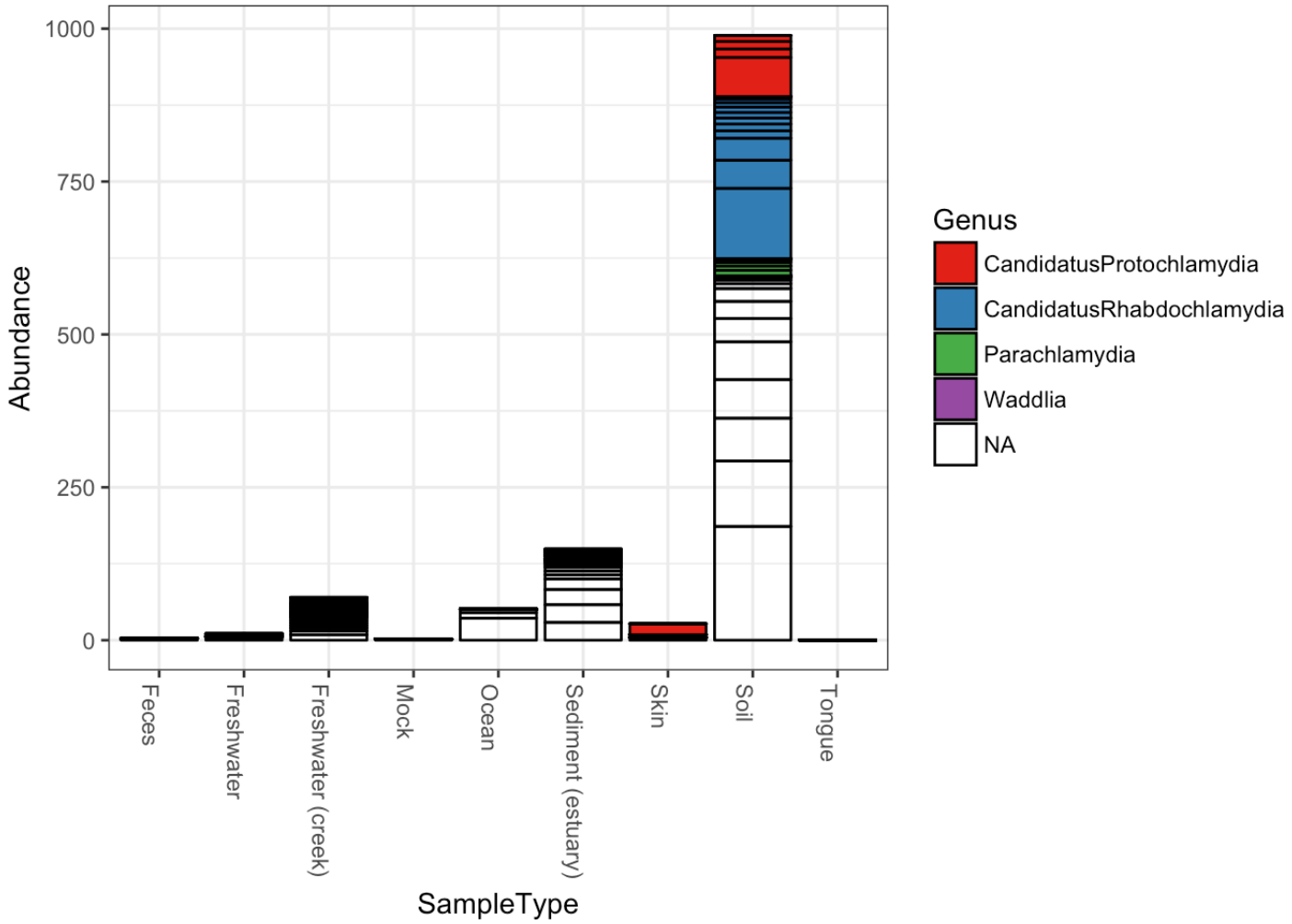
Add some fill color to represent the Genus to which the OTUs belong

```
# Add fill color for Genus to basic bar plot

plot_bar(gp.ch, fill = "Genus")
```



Keep same fill color, but group by SampleType.

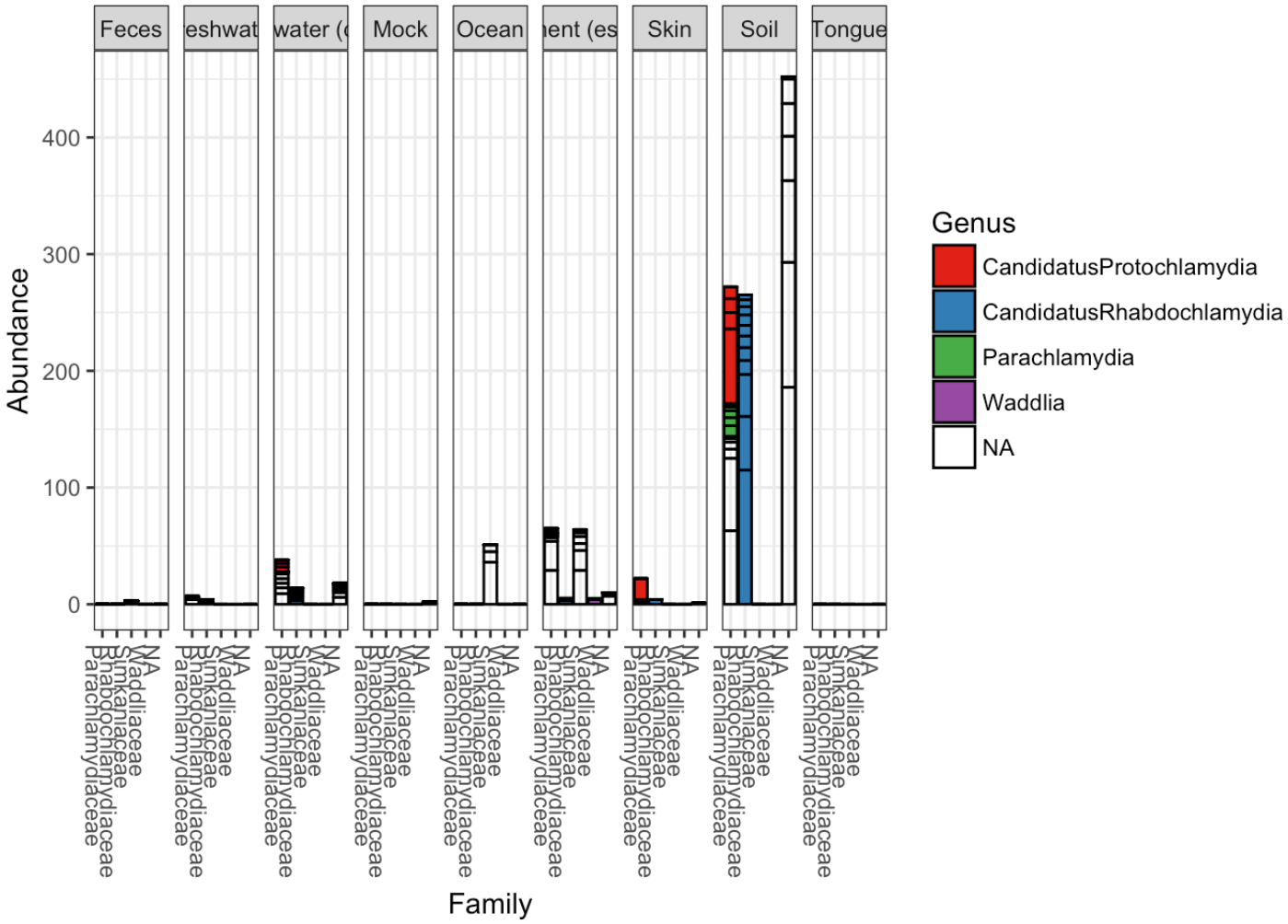
```
# Add fill color for Genus to basic bar plot

plot_bar(gp.ch, fill = "Genus", x = "SampleType")
```



Jazz It Up with Facets

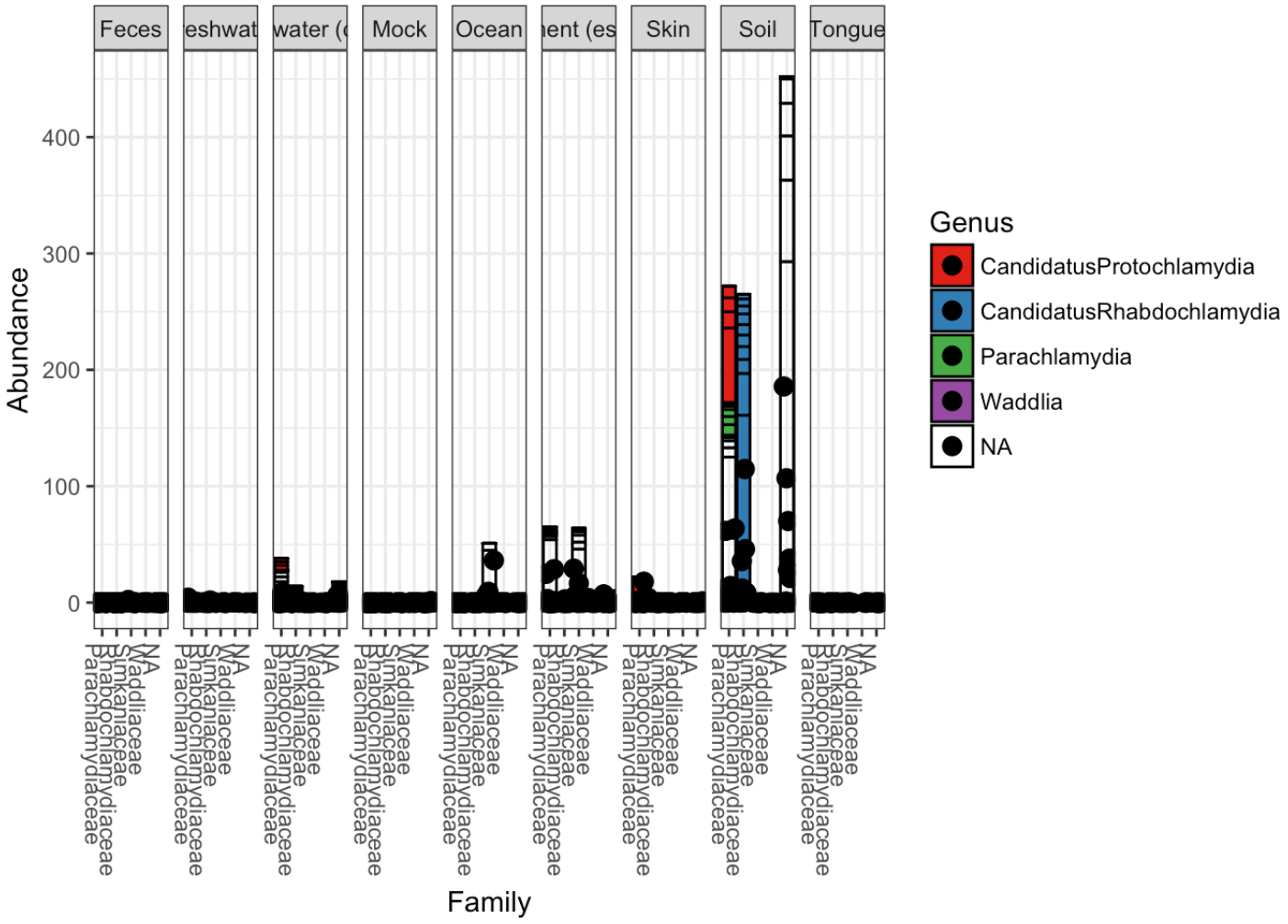
Organize further with facets, graphing the Families faceted by SampleType and filled by Genus.

```
# Add fill color for Genus to basic bar plot

plot_bar(gp.ch, "Family", fill = "Genus", facet_grid = ~SampleType)
```



Even fancier by adding a layer of jittered points.

```
# Add a layer of jittered points

p = plot_bar(gp.ch, "Family", fill="Genus", facet_grid=~SampleType)
p + geom_point(aes(x=Family, y=Abundance), color="black", position="jitter", size=3)
```



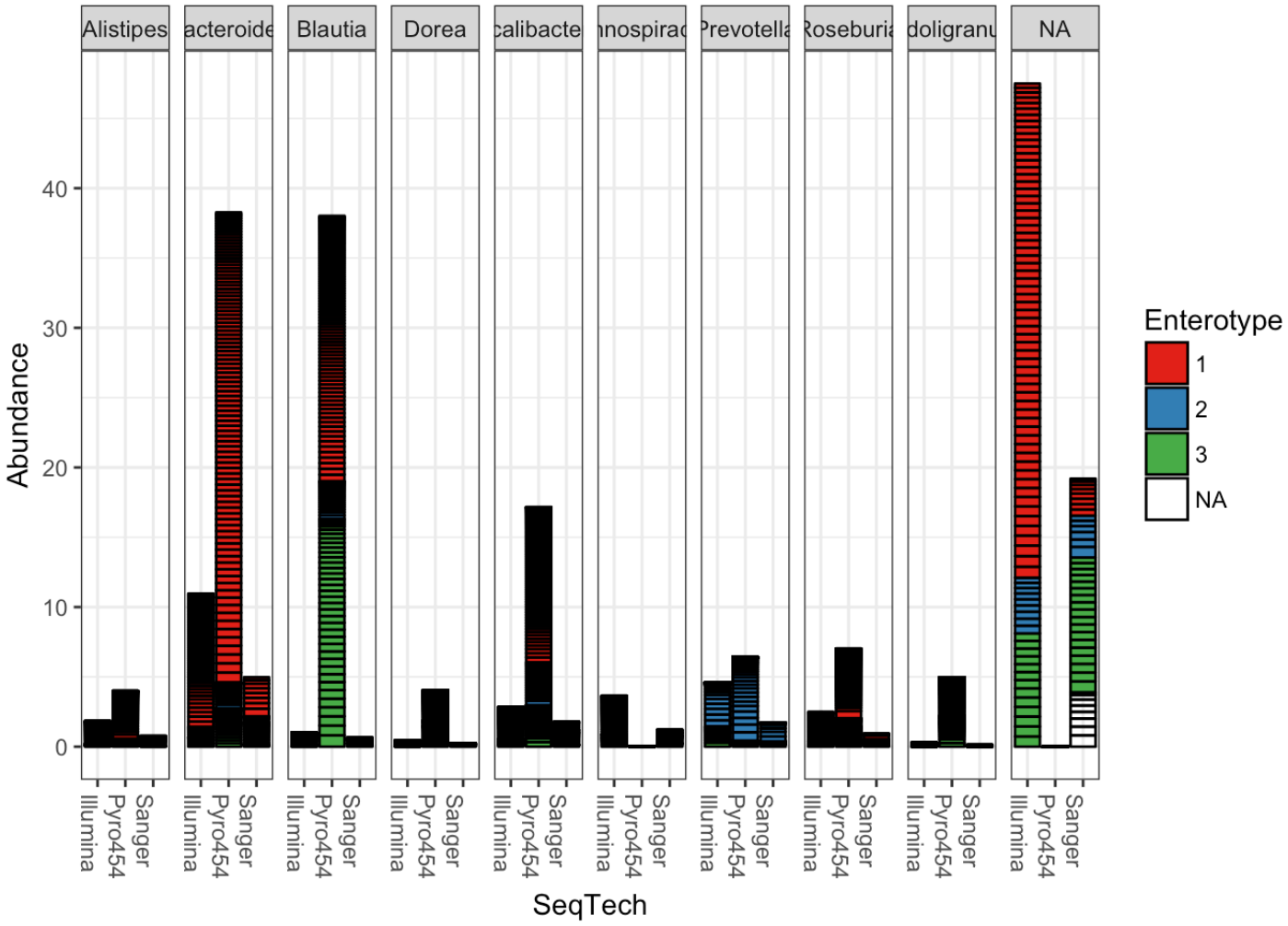
Enterotype Dataset

Let's first trim to the 10 most abundant genera.

```
TopNOTUs <- names(sort(taxa_sums(enterotype), TRUE)[1:10])
ent10   <- prune_taxa(TopNOTUs, enterotype)
```

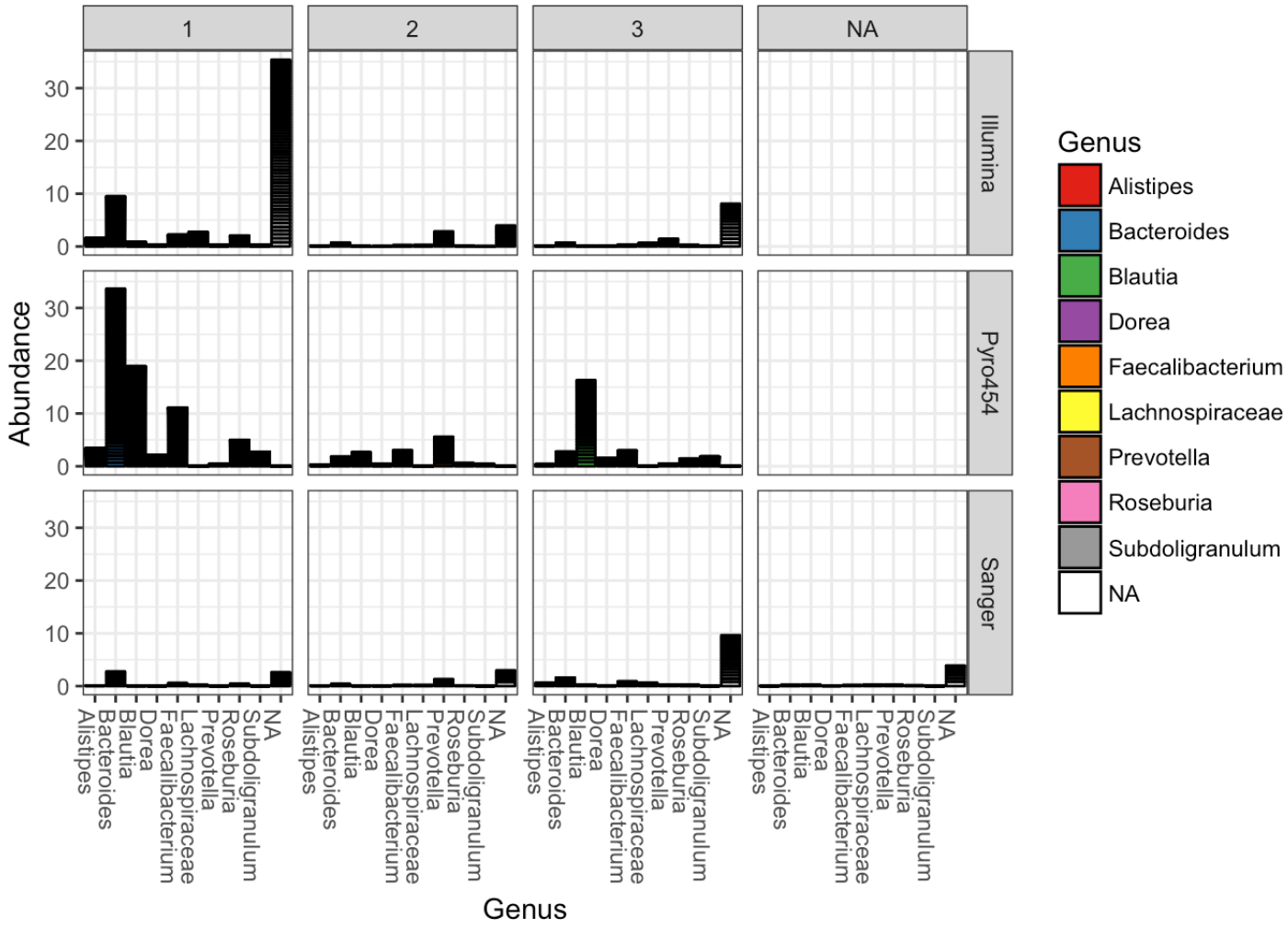
The settings in the following R-chunk were chosen after trial-and-error, looking to show off certain features of the data to the best advantage.

```
# Bar graph for Enterotype to look at sequencing technology v genus detection

plot_bar(ent10, "SeqTech", fill="Enterotype", facet_grid = ~Genus)
```



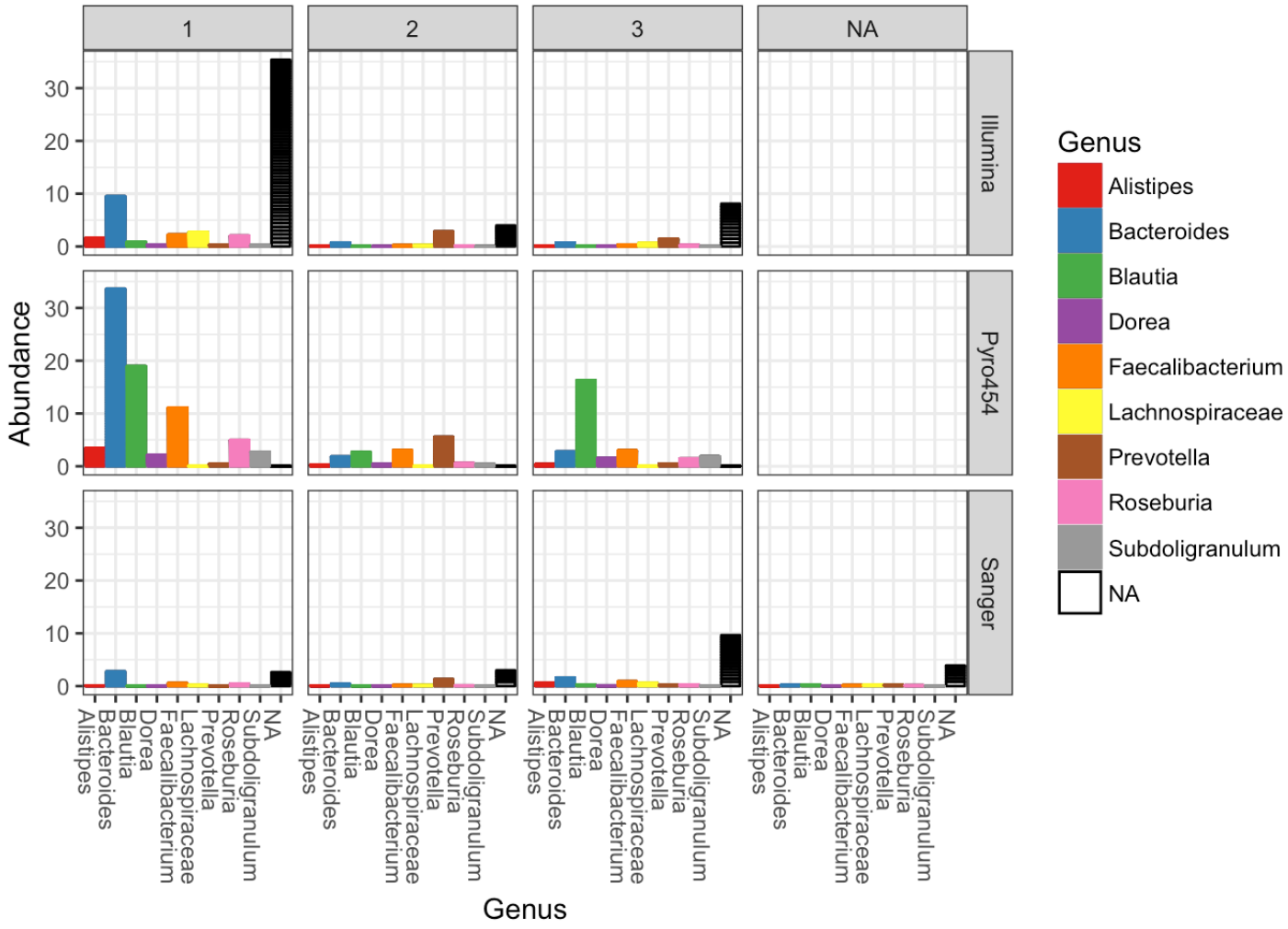
You could of course opt instead to separate enterotype designation and genus designation:

```
# Separate into finer grid

plot_bar(ent10, "Genus", fill="Genus", facet_grid = SeqTech~Enterotype)
```



We can remove those OTU separation lines as follows:

```
# Separate into finer grid

p <- plot_bar(ent10, "Genus", fill="Genus", facet_grid = SeqTech~Enterotype)
p + geom_bar(aes(color=Genus, fill=Genus), stat="identity", position="stack")
```



Heatmap Plots

You can also create heatmaps using ordination methods to organize rows and columns instead of hierarchical clustering. Such an organization typically makes a more interpretable display. Lucky for us we can do all of this within `phyloseq`!

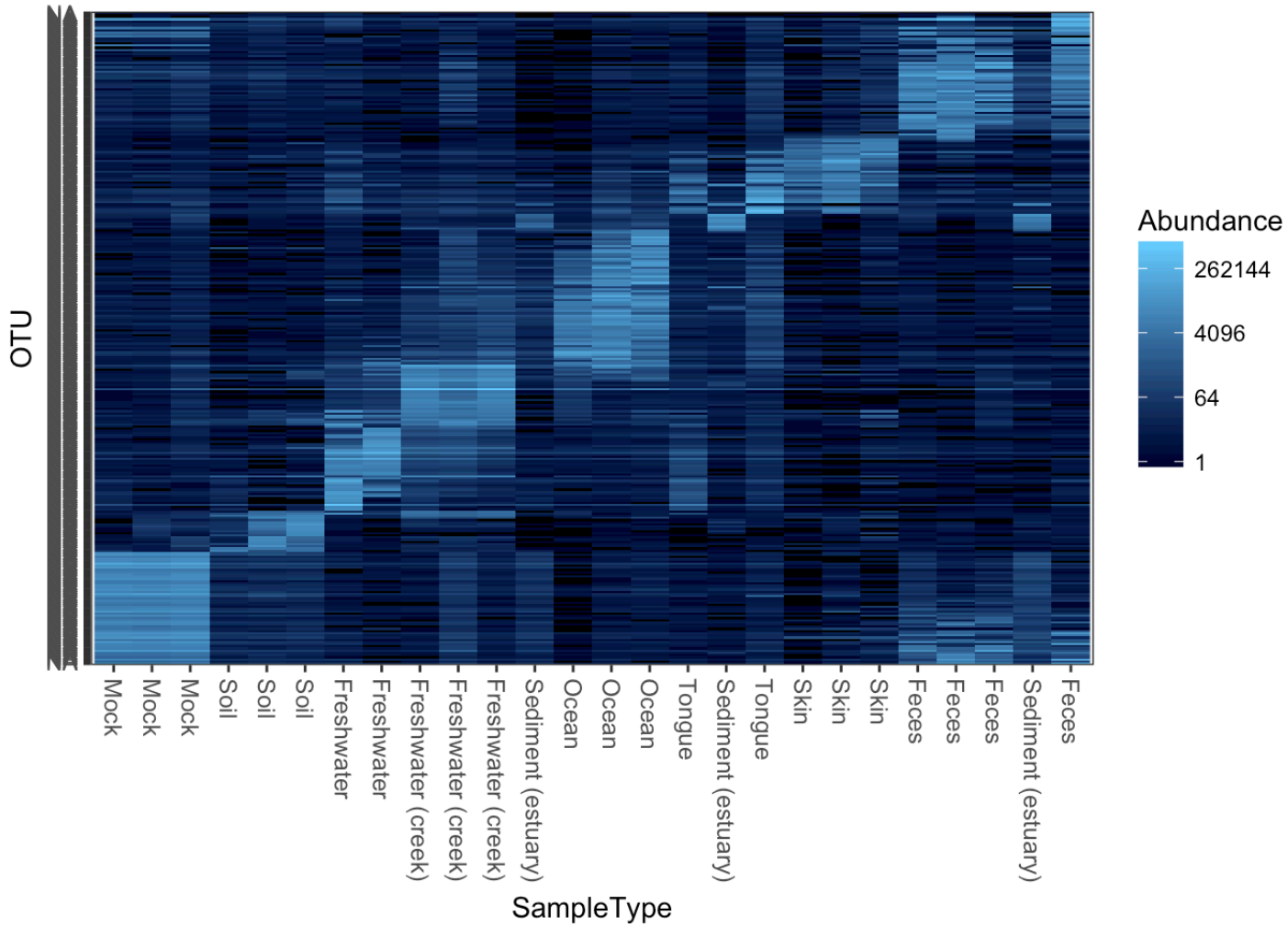
Traditionally heatmaps have been used to emphasize data that is above or below a threshold as “hot” or “cold” respectively. But when used with OTU abundance data, the need is to see the relative patterns of high abundance OTUs against a background of low abundance or absent groups. Thus the default color scheme is dark blue (low abundance) to very light blue for highest abundance, with black representing missing or zero abundance values. Thus you can change the color scheme by changing the `low`, `high`, and `na.value` arguments.

Plot a 300-taxa Dataset

et’s get the top 300 most abundant Bacteria taxa across all samples in the GlobalPatterns dataset and plot a heatmap. Quick and dirty.

```
# Get top 300 most abundant Bacteria taxa in GlobalPatterns

gpt <- subset_taxa(GlobalPatterns, Kingdom=="Bacteria")
gpt <- prune_taxa(names(sort(taxa_sums(gpt),TRUE)[1:300]), gpt)

# Plot the heatmap

plot_heatmap(gpt, sample.label="SampleType")
```

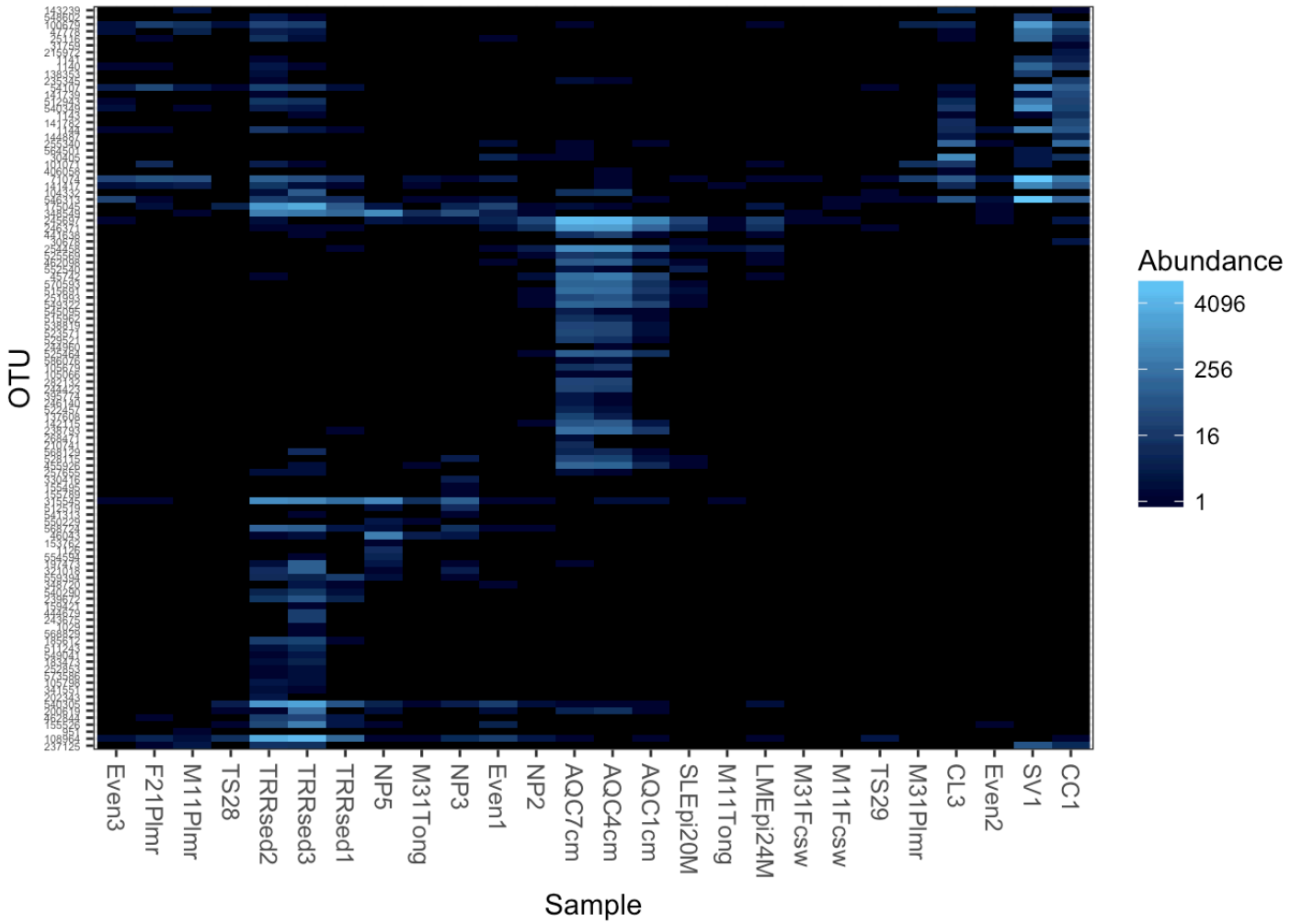
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



This is kind of noisy. Let's deal with something a bit more manageable, like the Crenarchaeota phylum of Archaea, which we created above as gpac. Now the default heatmap

```
# Default heatmap for Crenarchaeota

plot_heatmap(gpac)
```

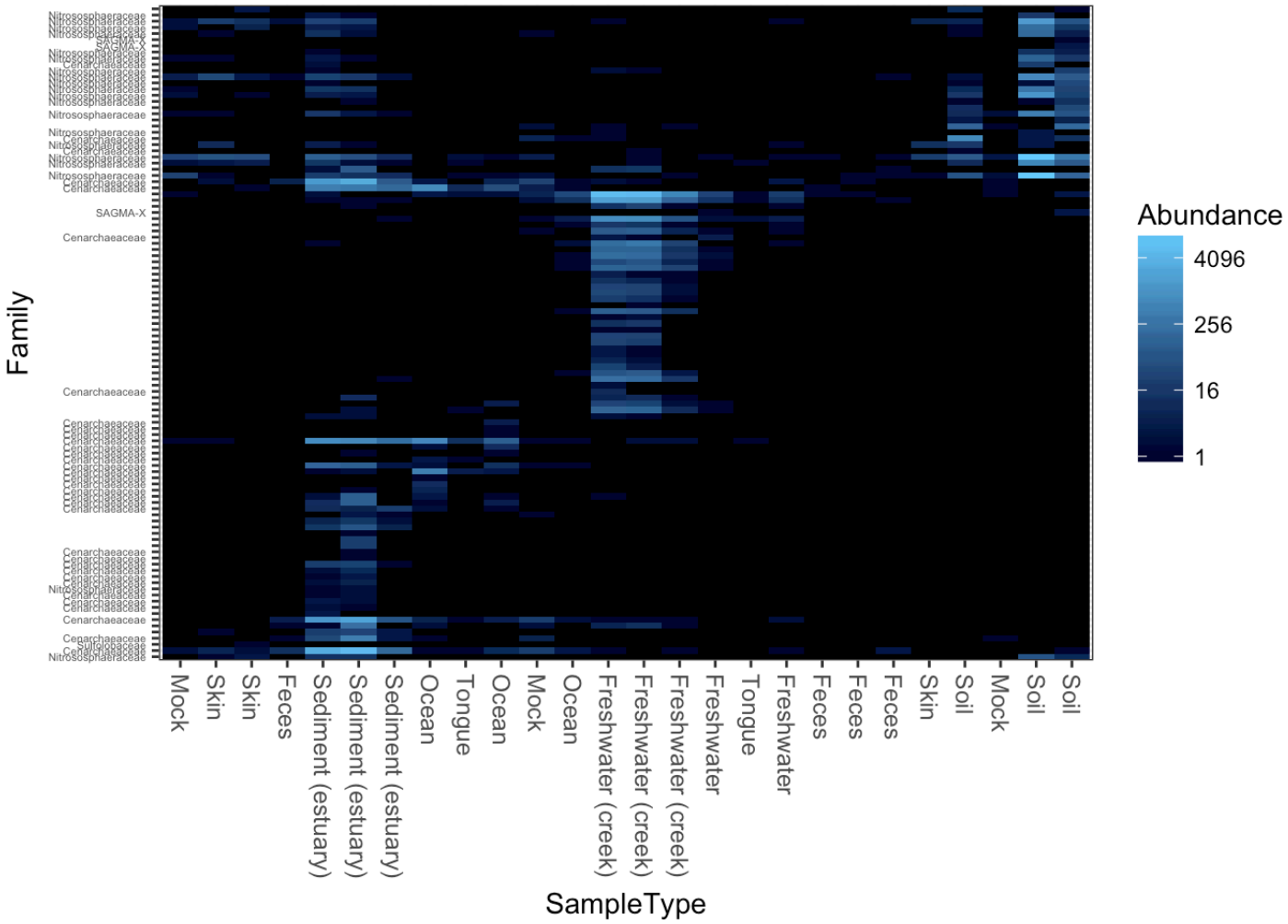
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Re-label by a sample variable and taxonomic family.

```
# Re-label and SampleType and Family

p <- plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family")
p
```

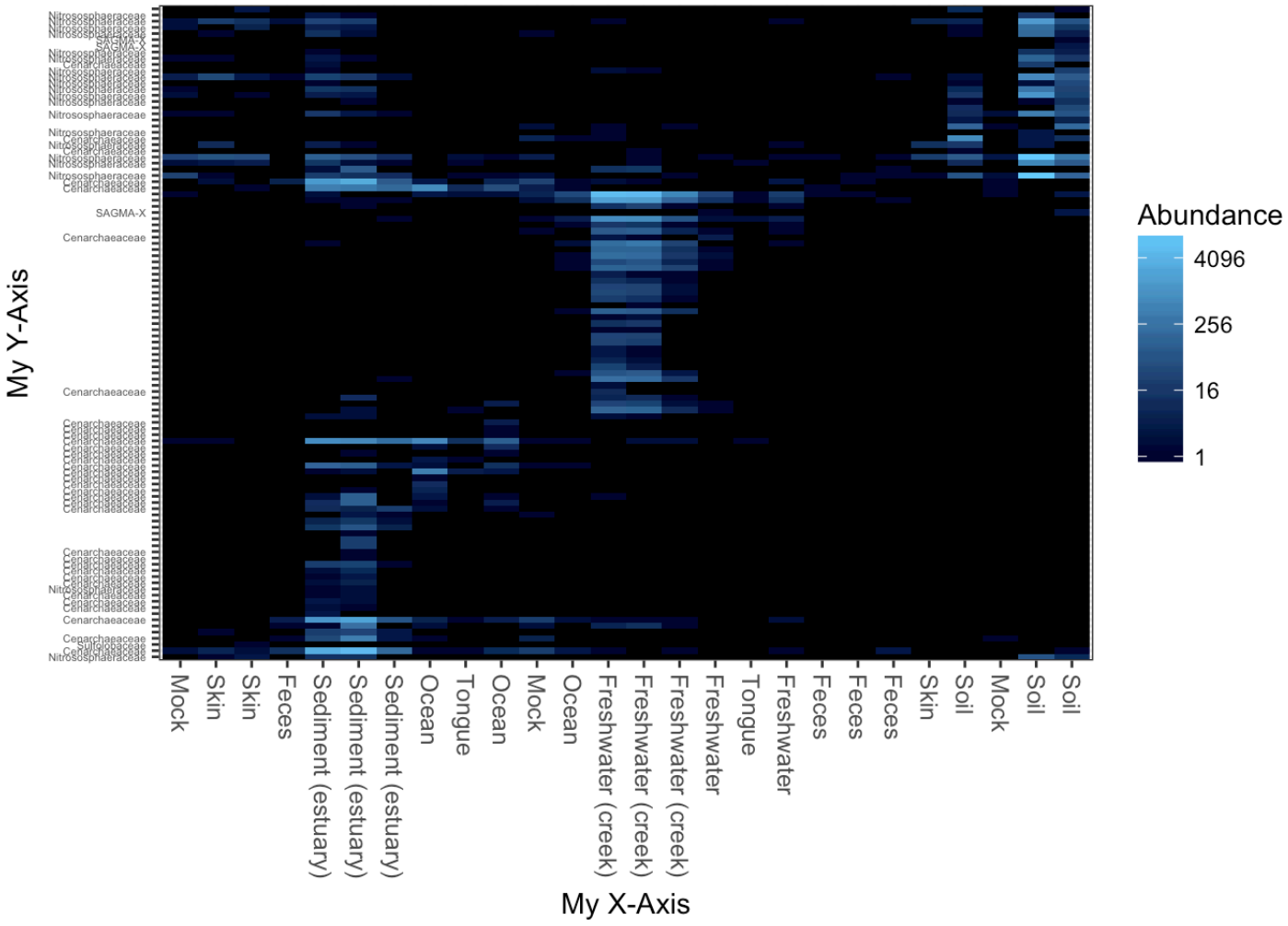
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



And if you wanted axis lables but not hte labels on individual features?

```
# Relabel

p$scales$scales[[1]]$name <- "My X-Axis"
p$scales$scales[[2]]$name <- "My Y-Axis"
print(p)
```

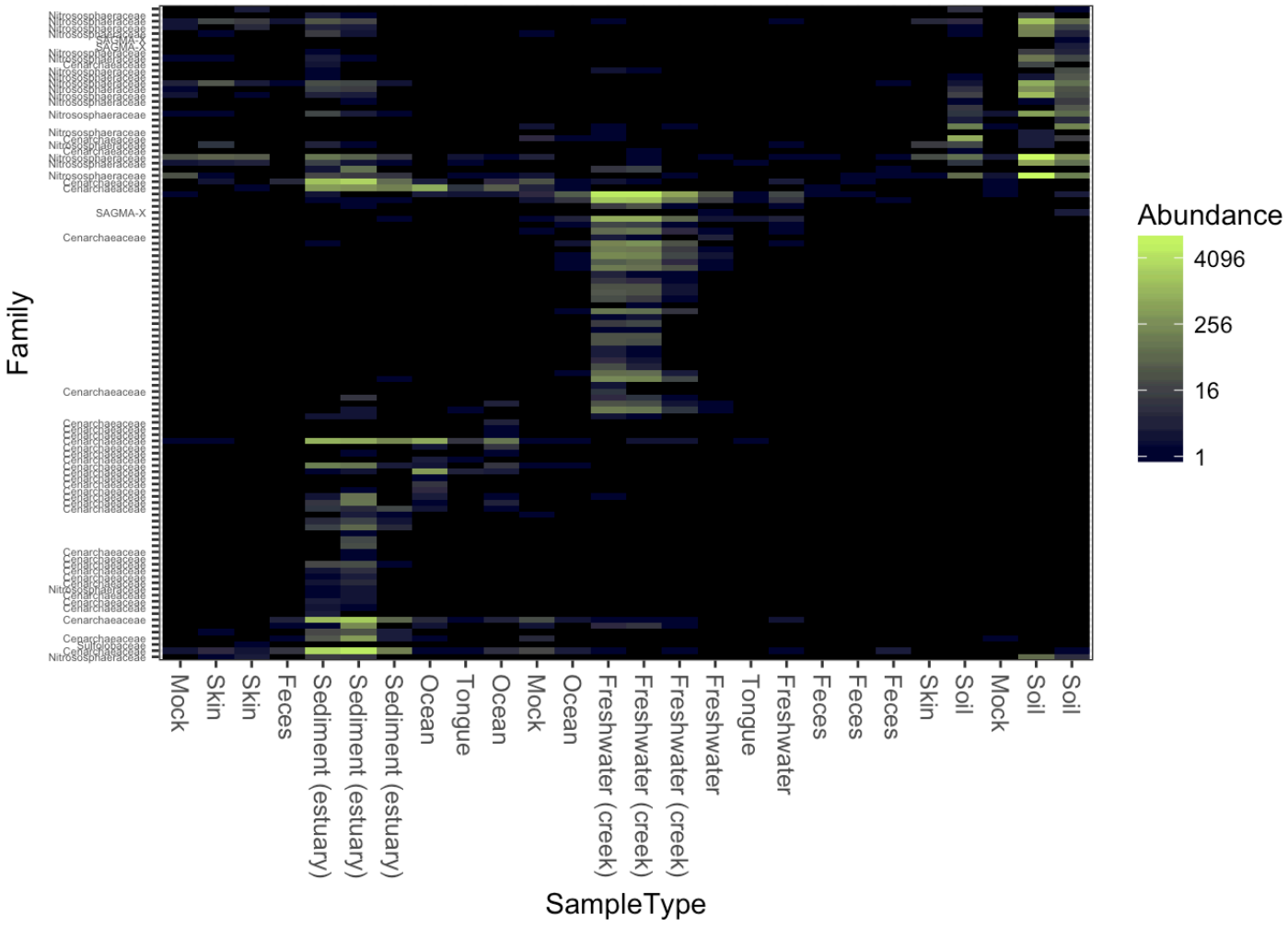
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



How about a different color scheme?

```
plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family", low="#000033", high="#CCFF66")
```

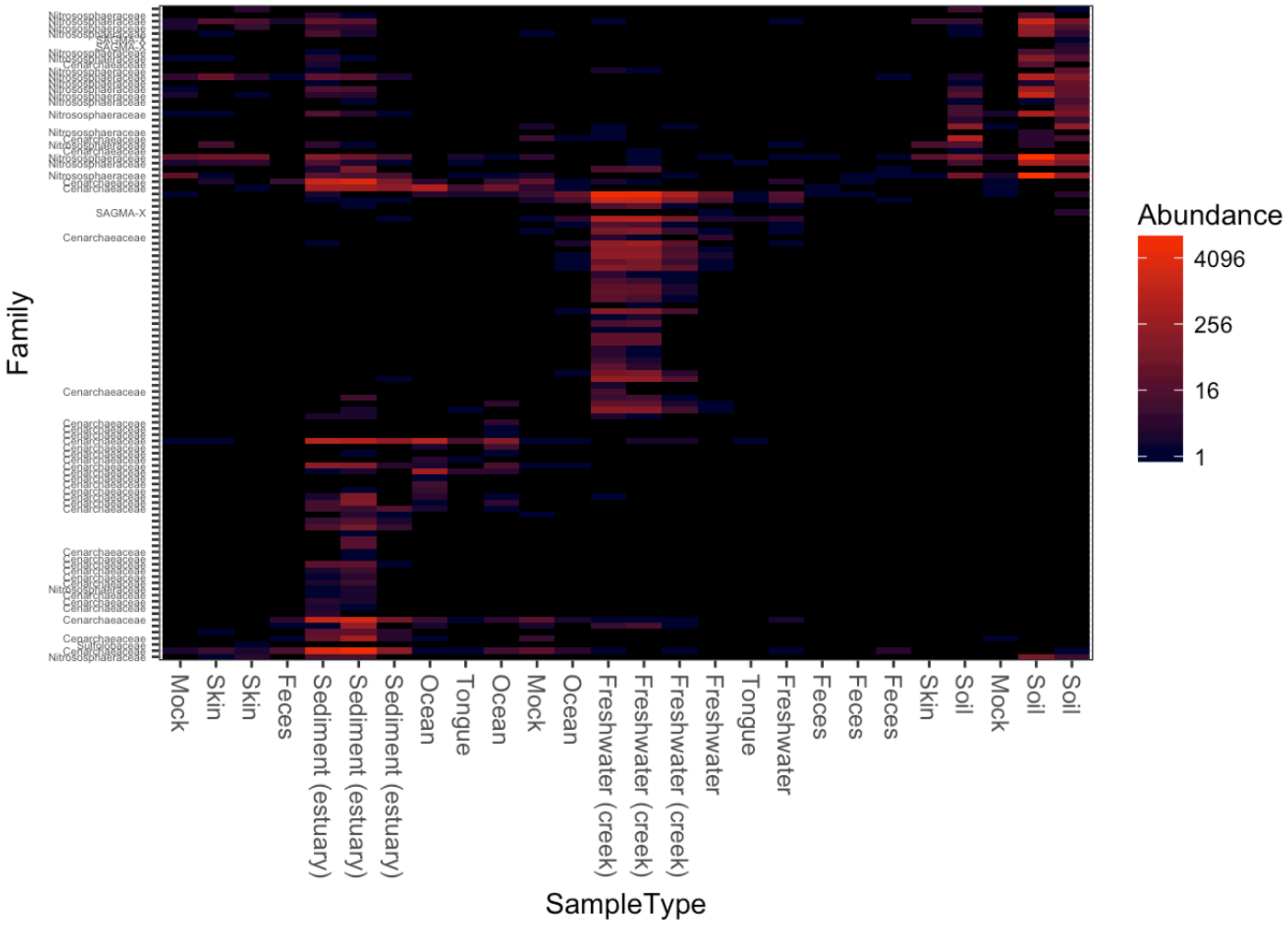
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Or a very dark blue to red scheme:

```
plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family", low="#000033", high="#FF33
00")
```

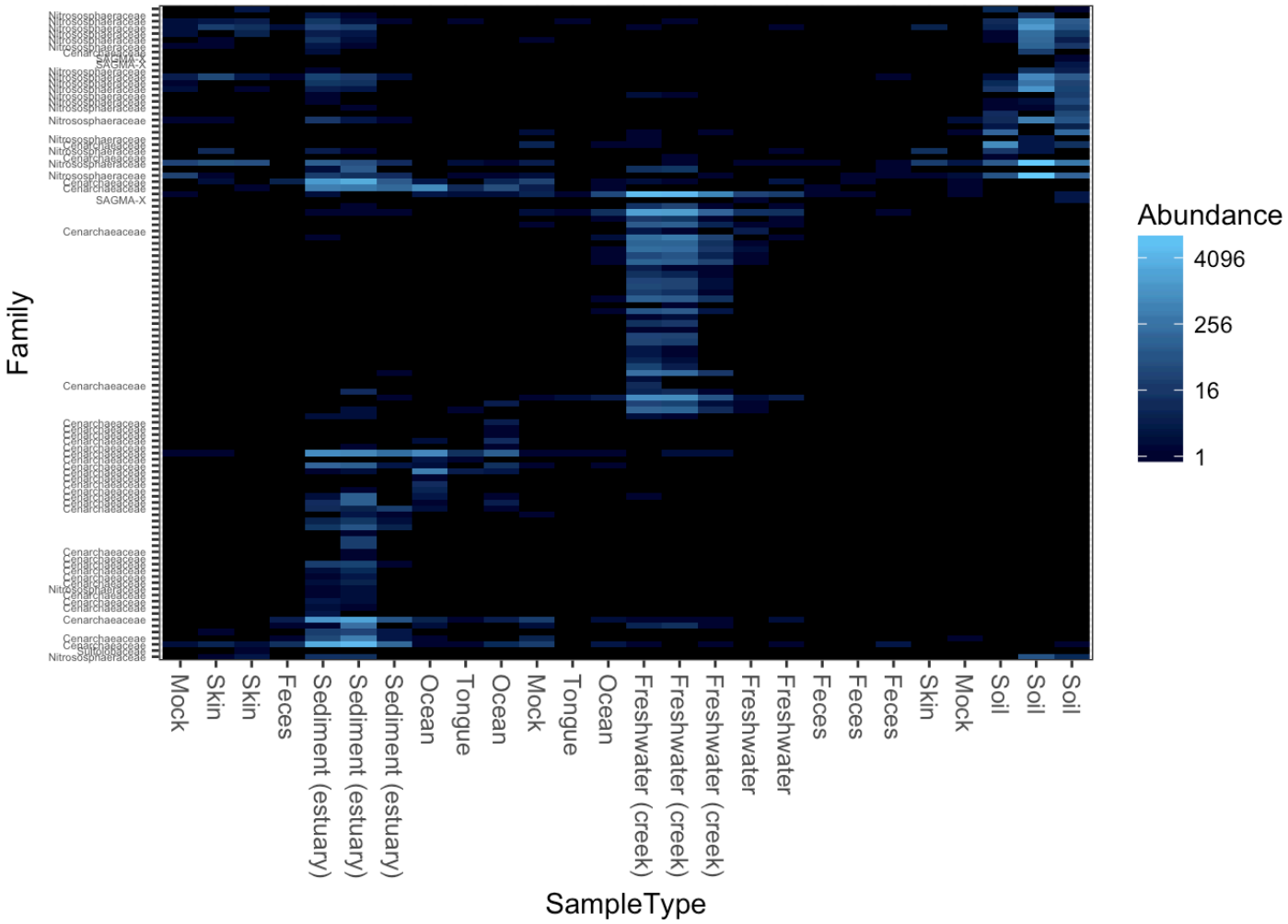
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Dark blue to light blue:

```
plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family", low="#000033", high="#66CC
FF")
```

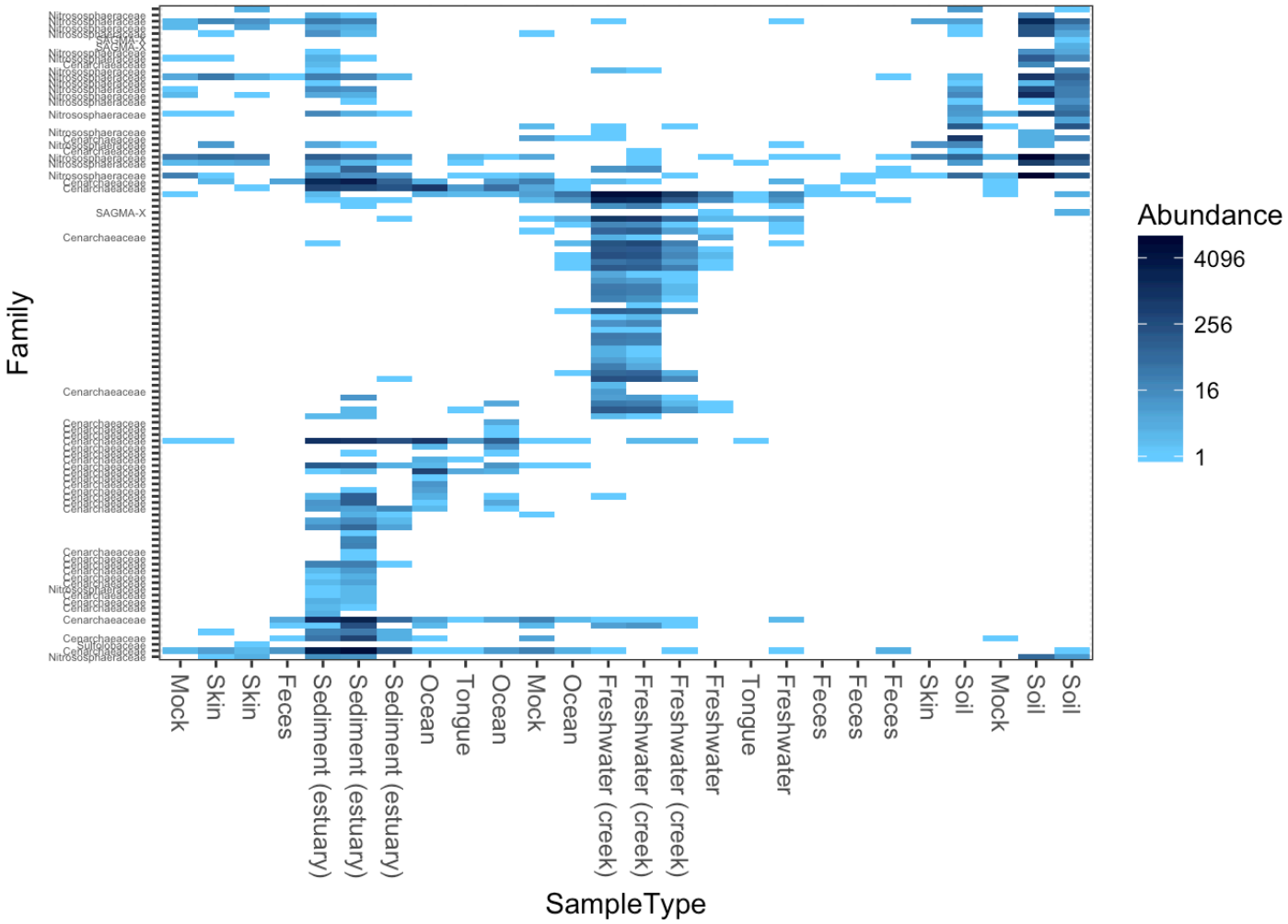
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Here is a “dark on light” color scheme, with the white value for NA and 0 elements.

```
plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family", low="#66CCFF", high="#0000
33", na.value = "white")
```

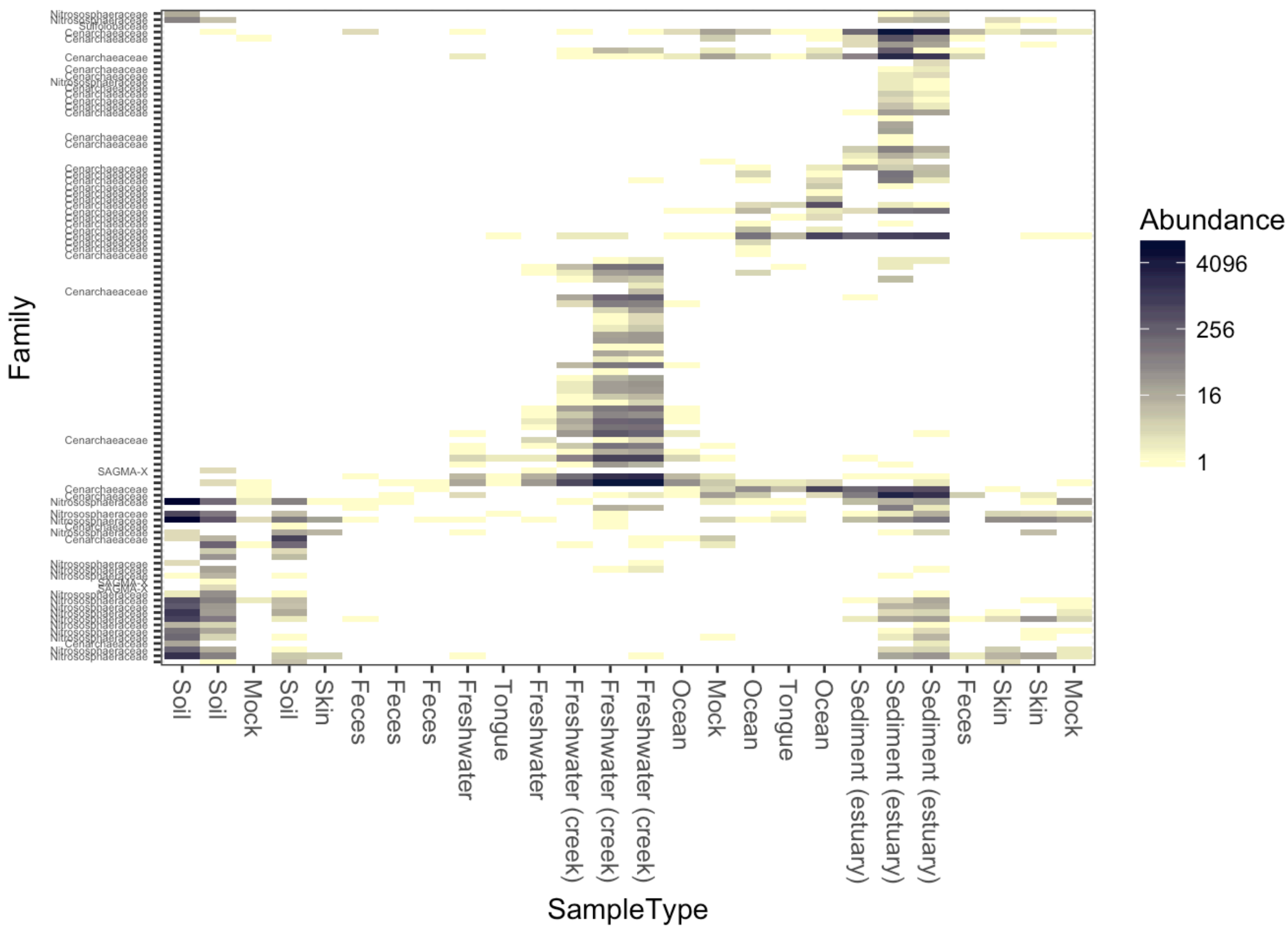
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



A somewhat similar color scheme to that above, but now the “near-zero” color is close to a cream color and the colors themselves closer to grey. Better contrast overall, but more subtle in a way, not as exciting.

```
plot_heatmap(gpac, "NMDS", "bray", "SampleType", "Family", low="#FFFFCC", high="#0000
33", na.value = "white")
```

```
## Warning: Transformation introduced infinite values in discrete y-axis
```

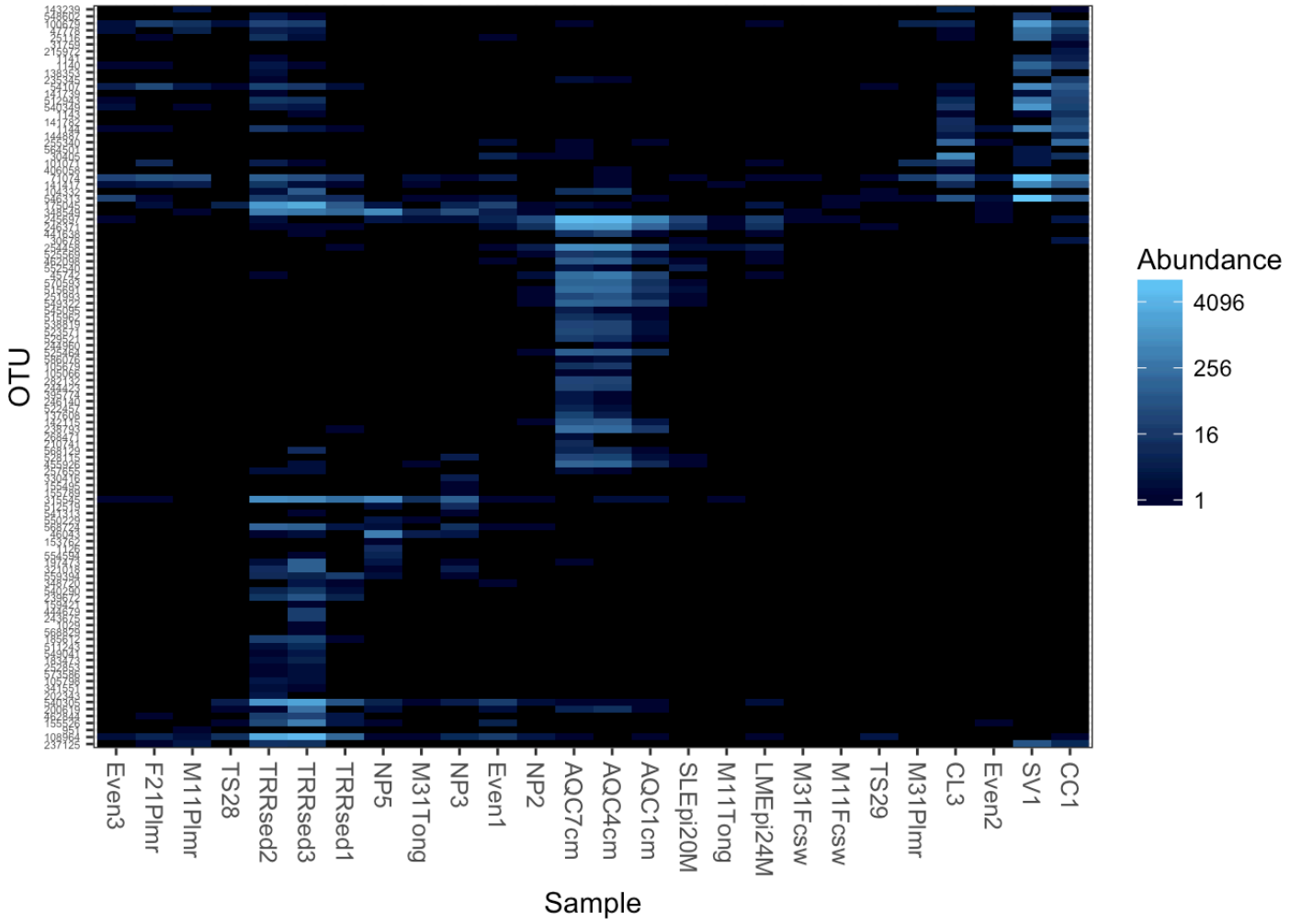


Now let's try with different distances, different ordination methods

First the default heatmap with NMDS ordination on the jaccard distance.

```
plot_heatmap(gpac, "NMDS", "jaccard")
```

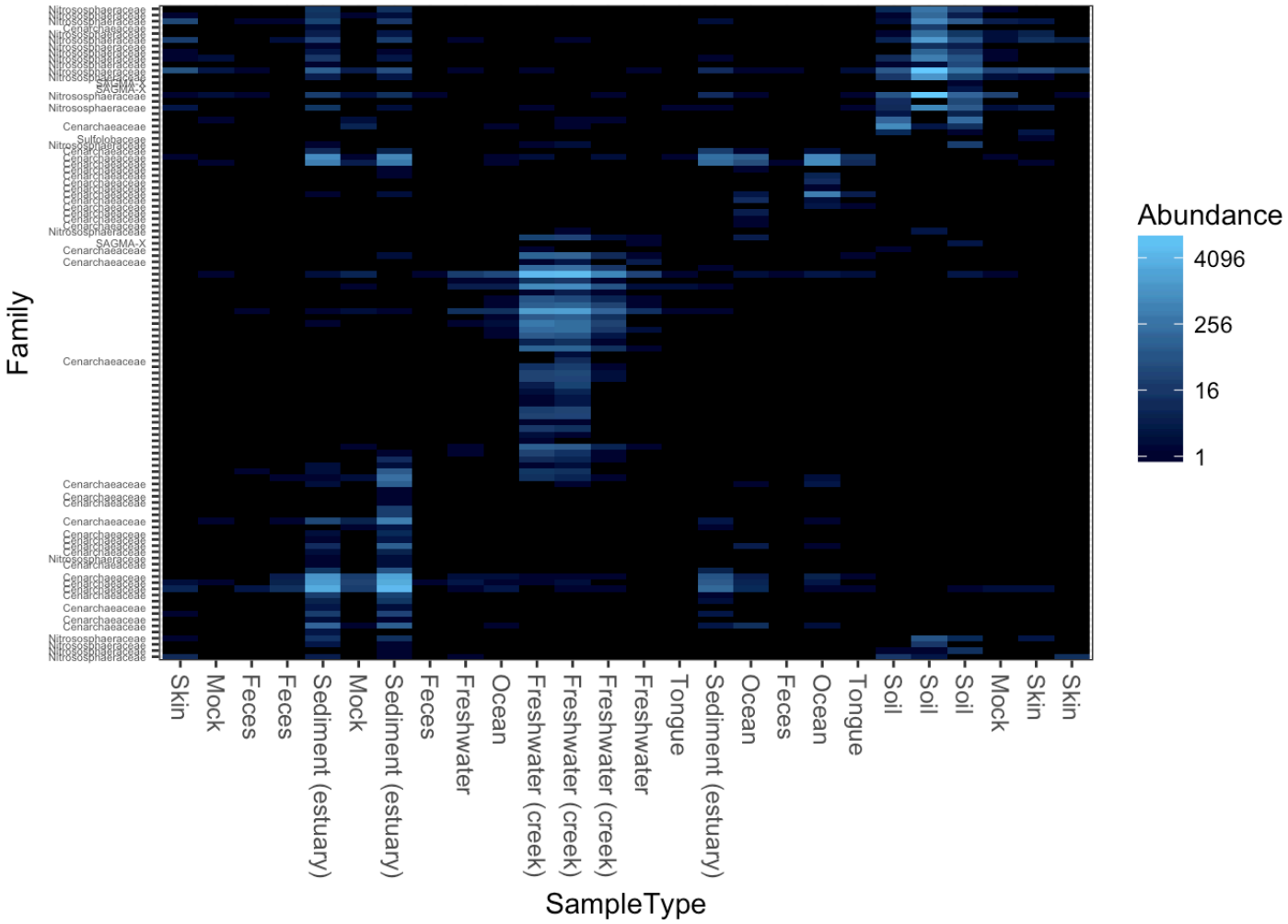
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Detrended correspondence analysis.

```
plot_heatmap(gpac, "DCA", "none", "SampleType", "Family")
```

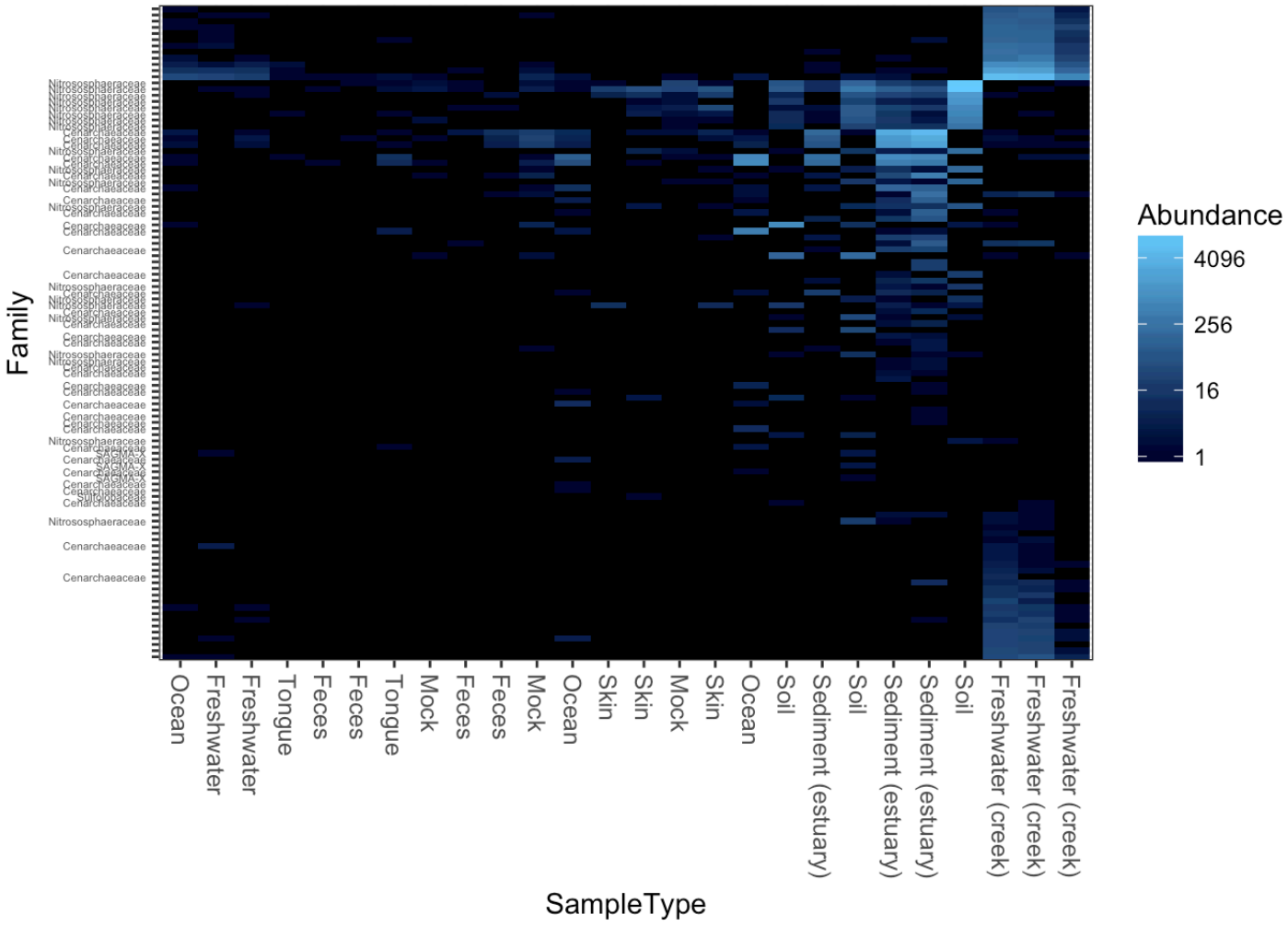
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Unconstrained redundancy analysis (PCA by any other name)

```
plot_heatmap(gpac, "RDA", "none", "SampleType", "Family")
```

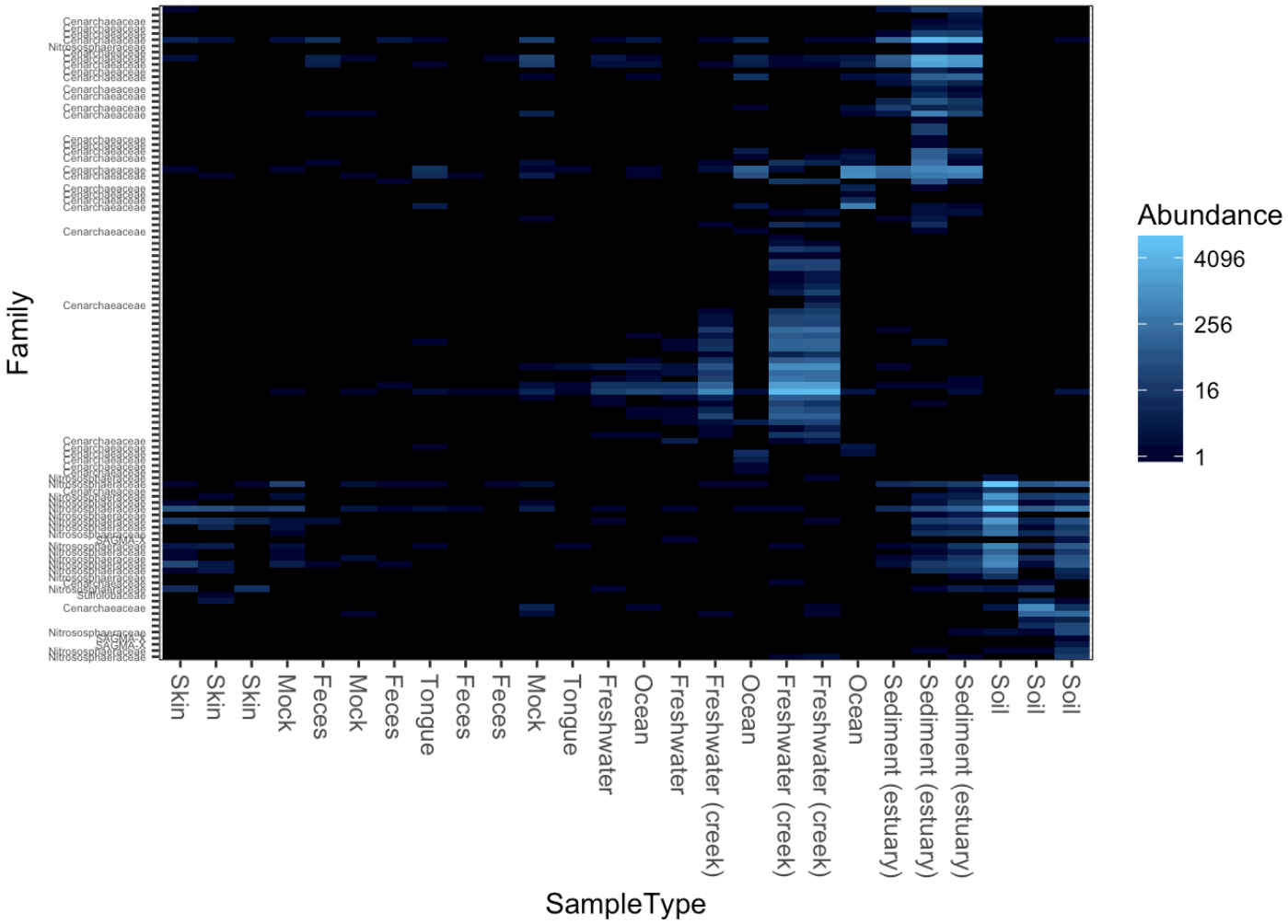
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



PCoA/MDS on the (default) Bray-Curtis distance.

```
plot_heatmap(gpac, "PCoA", "bray", "SampleType", "Family")
```

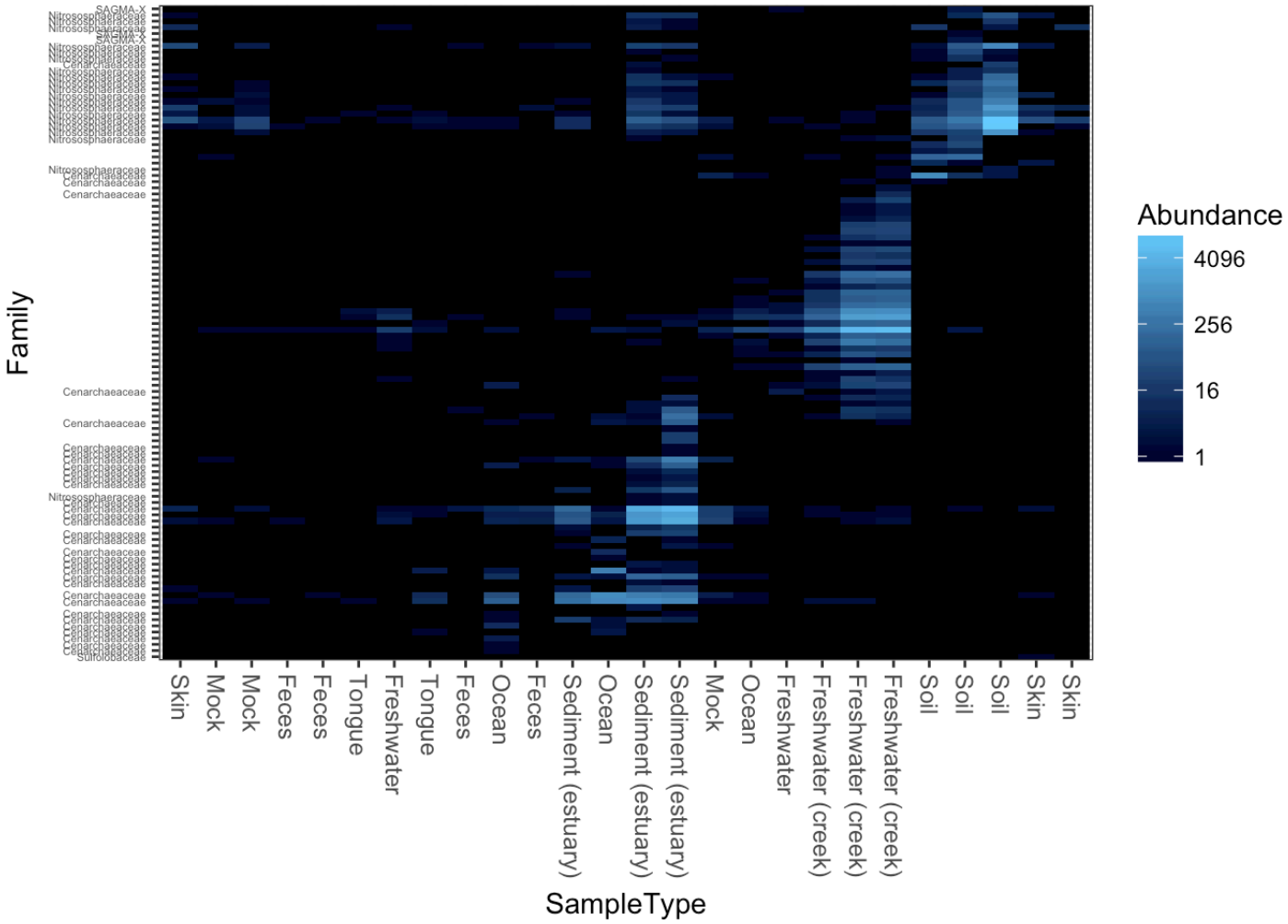
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



PCoA/MDS on unweighted UniFrac distance

```
plot_heatmap(gpac, "PCoA", "unifrac", "SampleType", "Family")
```

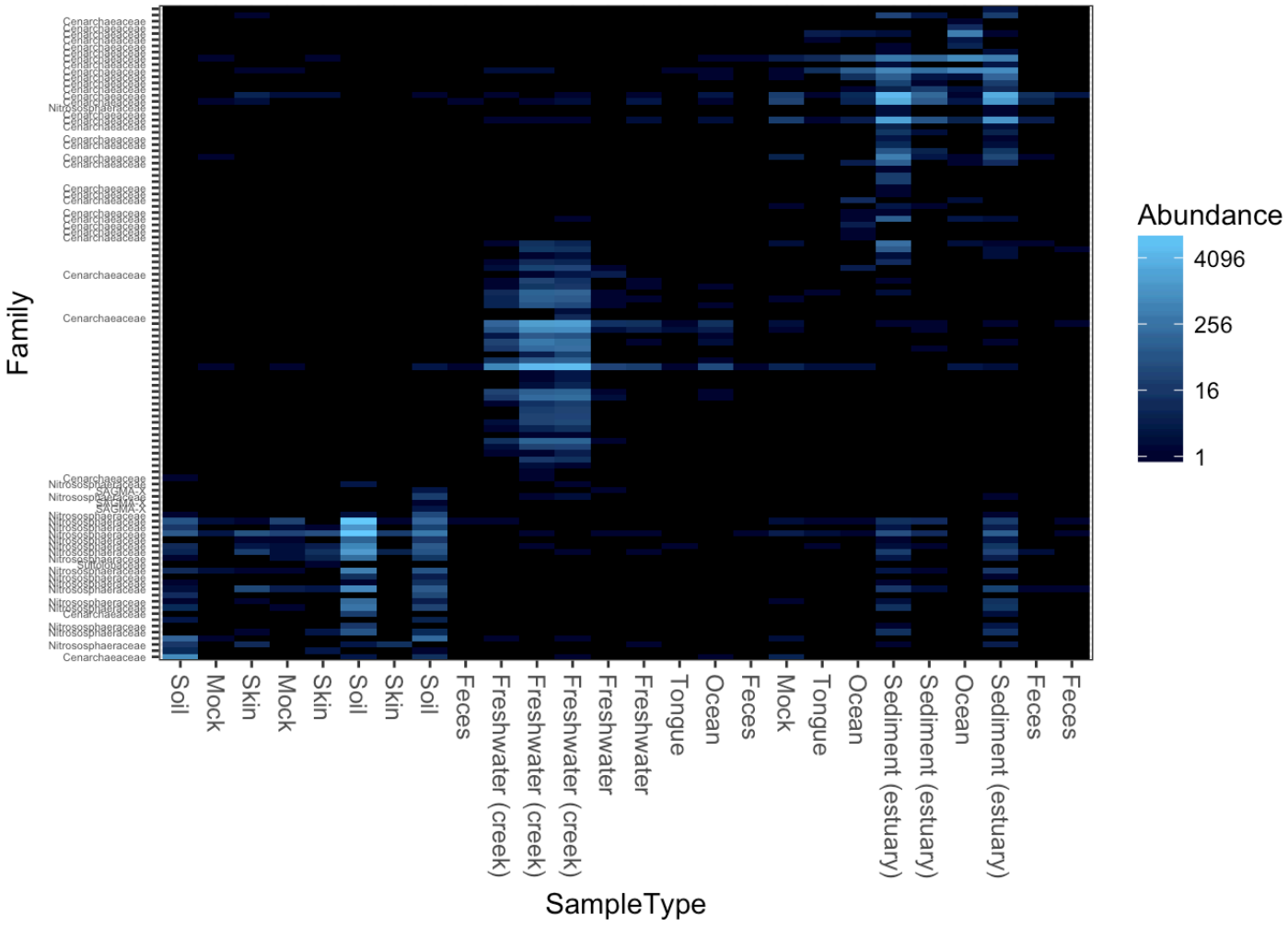
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



Same as above but with weighted UniFrac.

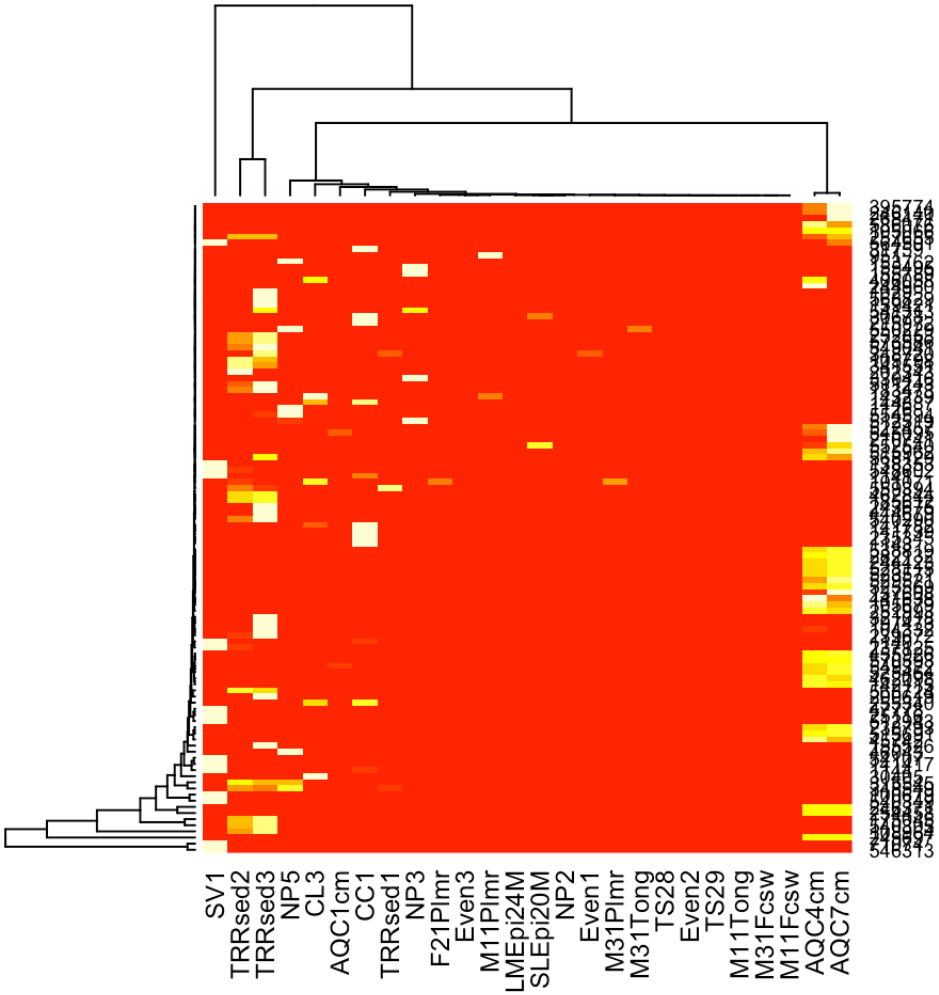
```
plot_heatmap(gpac, "PCoA", "unifrac", "SampleType", "Family", weighted=TRUE)
```

```
## Warning: Transformation introduced infinite values in discrete y-axis
```



And finally, just the regular heatmap produced by base-R graphics with a hierarchical clustering. Just in case you wanted to compare with `plot_heatmap`. Because I know you want to.

```
heatmap(otu_table(gpac))
```



Plot Microbiome Network

There is a random element to network layout methods. So we want this to be reproducible, so we will set the random number generator seed explicitly:

```
# Set the seed

set.seed(711L)
```

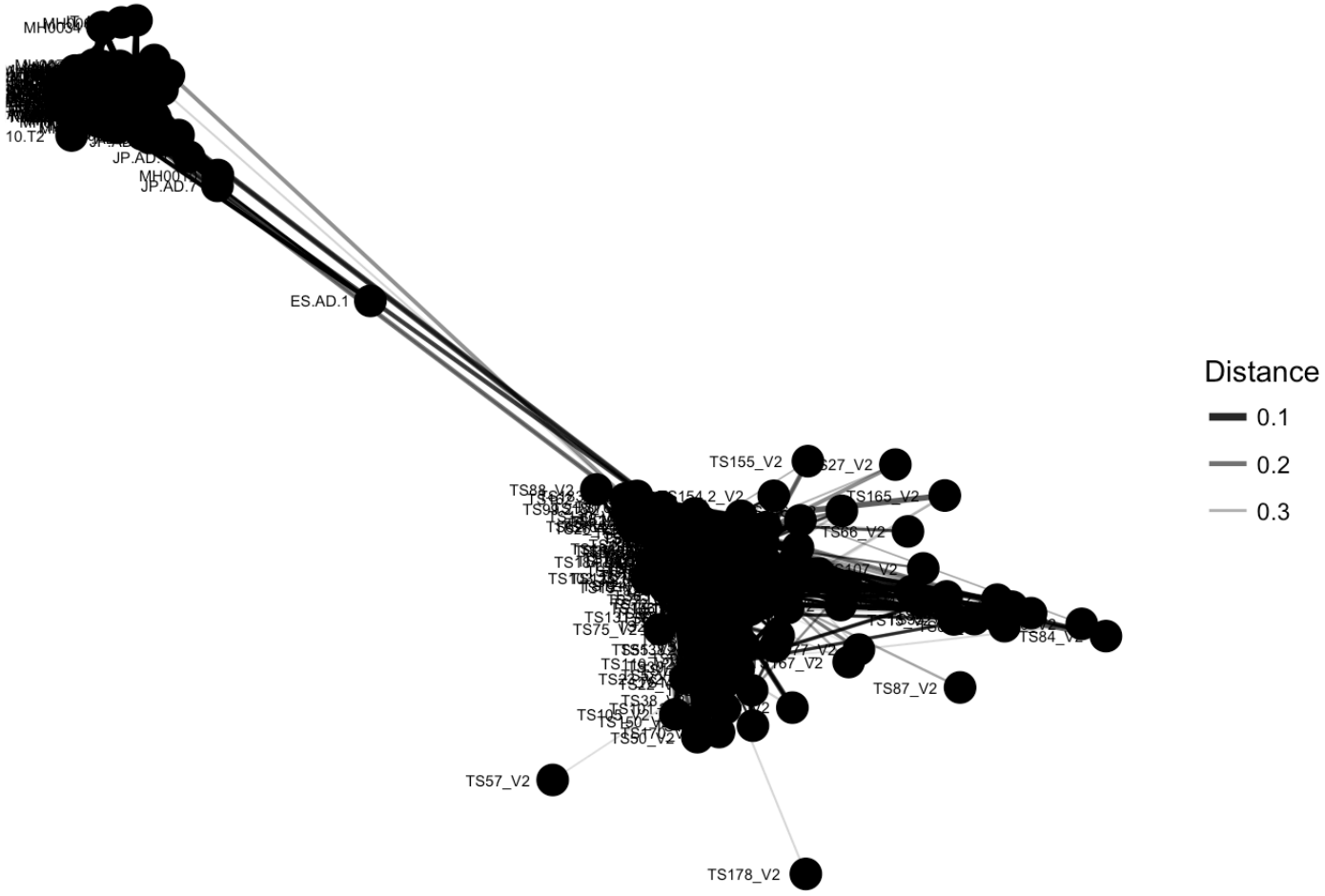
We are going to work first with the Enterotype dataset. We will first want to remove the 9 samples for which no enterotype designation was assigned.

```
# Remove samples with no enterotype designation

enterotype <- subset_samples(enterotype, !is.na(Enterotype))
```

We are going to create a network of samples in this dataset, connecting two samples when a distance between them is below some threshold.

Let's plot first with the default settings.

```
# Default microbiome network plot

plot_net(enterotype, maxdist = 0.4, point_label="Sample_ID")
```

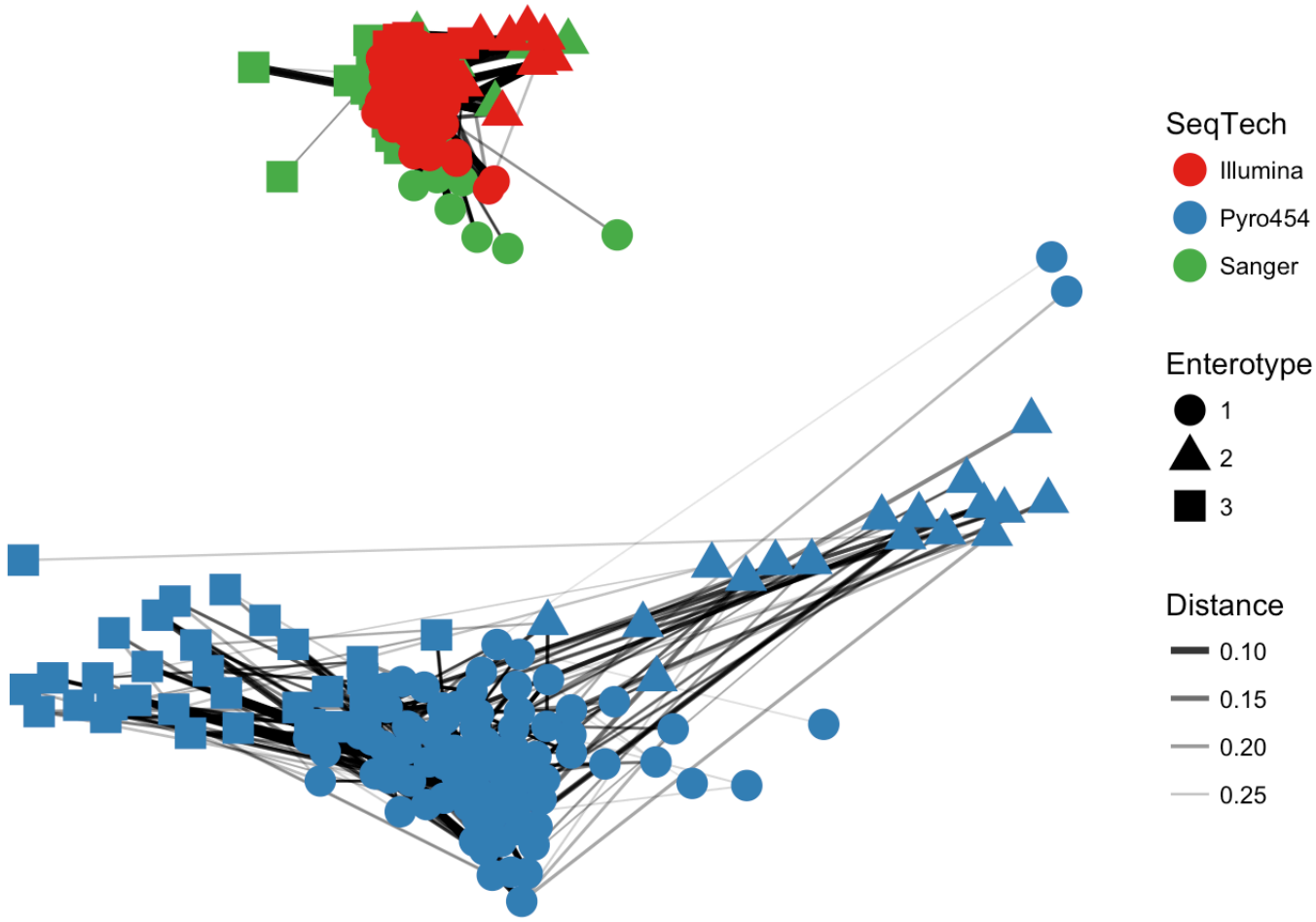


What do you notice here, other than a big mess?

Well there are two subgraphs that make up the majority of samples.

Let's use some color to see about aspects of the network.

```
# Exercise some options

plot_net(enterotype, maxdist = 0.3, color="SeqTech", shape="Enterotype")
```



What do you notice now?