alpha-diversity-normalization

Michael Sieler

03 March, 2022

Contents

The code should be able to be ran as is after you've installed the required libraries. I included a built-in dataset from the `Phyloseq` package called `GlobalPatterns` that will allow you to test the script before using your own phyloseq object.

Setup Environment

```
# Load Libraries

## General Purpose
library(knitr)  # For knitting documents to HTML or PDF formats
library(tinytex)  # LaTex stuff for Rmarkdown
library(data.table)  # Handle data tables
library(tidyverse)  # Making your code look pretty and tidy
library(reshape2)  # for reshaping data using melt()
library(rcompanion)  # various functions, transformTukey()

## Figures/Tables
library(ggplot2)  # For plotting pretty graphs
library(CoDaSeq)  # For CLR transformations, PCOA plots
library(gridExtra)  # use marrangeGrob, for combining plots
library(ggbeeswarm)  # Pretty dots on box plots

## Microbiome Analysis
library(phyloseq)  # For microbiome analysis and plotting functions
library(phyloseqCompanion)  # helper functions for manipulating phyloseq objects
library(nortest)  # Allows us to run ad.test()
library(picante)  # Allows us to use pd() to calc phylogenetic diversity
```

Background

Here is some context to the script, that hopefully adds some clarity to why things are done the way they are.

Code Chunks

I've split the code into the major steps, and then further by "Save Function" and "Run Function" because normally I save my functions to a separate script that I call using `source()`, but for simplicity I've included everything into one document.

Naming Conventions

I follow my own brand of naming convetions, which may differ from yours.

Variables:

Generally: `<highLevelVarType>.<subtype>.<subset>.<variables>.<Modifications>`

- Names separated by `.`, going from broad to specific
- Words following the first word between `.`'s are capitalized
  - Ex: `thereIsMoreThanOneWord.betweenThePeriods.inThisLongVariableName`
- Make variable names as succinct as possible
  - Ex: `multiWord.btPeriod.inVar`

Examples:

- Datatables: `dt.<subtype>.<subset>.<modifications>` # datatables
  - Ex: `dt.control.time0`
  - Ex: `dt.exposed.time0`
- Dataframes: `df.<subtype>.<subset>.<modifications>` # dataframes
- Plots/figures: `plot.<subset>.<y-var>.<x-vars...>` # plots/figures
  - plot.betaDiv.
- Tables: `table.<subset>.<y-var>.<x-vars...>` # tables
- Models (lm, glm, etc.): `mod.<subtype>.<subset>.<y>.<x-vars...>` # models (lm, glm, etc.)
- Phyloseq objects: `ps.<subtype>.<subset>` # phyloseq objects

Functions:

- Same concept with variables, but use _'s instead of .'s
- Should be descriptive, but short enough to know the main task of the function

Import and Clean Data

Phyloseq Object

```
# Example data
data(GlobalPatterns)

# Load phyloseq object
ps.all <- GlobalPatterns

# View sample data
view(sample.data.frame(ps.all))
```

Clean PS Obj

If you need to clean phyloseq object for whatever reason (e.g., rarefying, normalizing, update column names etc.), you'd want to do that ahead of making data tables/frames.

```
# Remove columns, if rownames are already sample names, no need to have an extra sample column
sample_data(ps.all) <- sample_data(ps.all)[, c(-1)]  # [rows, cols]


# Rename sample column one at a time by name
# colnames(sample_data(ps.all))[colnames(sample_data(ps.all)) == "X.SampleID"] <- "Sample"


# Rename all columns
# colnames(sample_data(ps.all)) <- c()  # c("colName1", "colName2", ...)

# Check that renaming worked
# view(sample.data.frame(ps.all))
```

Create sample data tables/frames

```
# Load Sample Data
df.all <- sample.data.frame(ps.all)  # Dataframe
dt.all <- sample.data.table(ps.all)  # Datatable
```

Calculate Alpha Scores

Save Function

This is the function that does the alpha diversity score calculations.

```
# Calculate Alpha Scores ----------------------------------------------------
#   Description: Generates alpha diversity scores from a list of alpha methods
#   Input: phyloseq object, list of alpha div. methods,
#   Output: dataframe of alpha-diversity scores

alpha_base <- function(
  physeq,  # Phyloseq object
  methods,  # List of alpha methods (e.g., c(Shannon, Simpson, Observed) )
  smpl.col.name = "Sample",  # Default is "Sample" but you can change it to whatever when you call the
  phylo.div = F  # Only set to true if you have phylogenetic information attached to your phyloseq obje
  ){

  # Calculates alpha scores
  tmp.dt <- phyloseq::estimate_richness(
    physeq = physeq,  # Physeq object
    measures = methods[1:(length(methods)-1)]
  ) %>% as.data.table(keep.rownames = smpl.col.name) %>% setkeyv(smpl.col.name)  # Sets sample column n
  tmp.dt[, se.chao1 := NULL] # No idea what this does, but it's from Keatons code and I think it's impo

  # If you have phylogenetic information in your phyloseq object you'll want to set phylo.div to true
  if(isTRUE(phylo.div)){

    # Calculate the sum of the total phylogenetic branch length for one or multiple samples. See ?pican
    #   - Returns a dataframe of the PD and species richness (SR) values for all samples
    phy.dt <- picante::pd(samp = otu.matrix(physeq), tree = phyloseq::phy_tree(physeq)) %>%
      as.data.table(keep.rownames = smpl.col.name) %>% setkeyv(smpl.col.name)  # set col name

    # Names the columns by alpha method
    # names(phy.dt)[(length(phy.dt)-1):length(phy.dt)] <- methods[length(methods)]  #methods[(length(me
    names(phy.dt)[2:3] <- methods[(length(methods)-1):length(methods)]
    tmp.dt[phy.dt] # adds the columns to the other dataframe with the previously calculated alpha score

    # return to sender
    return (tmp.dt[phy.dt])
  }

  # Returns alpha scores datatable
  return (tmp.dt)
}
```

Running Function

Set alpha methods:

```
# Which alpha indices do you want to use?
#   - Uncomment out the code that you want to use

## Here is an example for phyloseq object without phylogenetic data
# methods.alpha <- c("Observed", "Shannon", "Simpson") %>%  # Add additional measures here. You can fin
#   purrr::set_names()


## Here is an example for phyloseq objects with phylogenetic data
#    - You can find different measures here: ?estimate_richness()

methods.alpha <- c("Shannon", "Simpson", # Non-phylogenetic measures, add additional measures here
                   "Phylogenetic", "Richness") %>%  # Don't add measures additional. You can find diffe
                  purrr::set_names()  # Set's names list elements in "alpha.methods"
```

Calculate alpha scores:

```
# Calculate raw alpha scores
#   Note: if you don't include se.chao1, it will throw a warning. No biggie

dt.alphaScores.all <- alpha_base(physeq = ps.all,  # Phyloseq object
                                 methods = methods.alpha,  # List of alpha methods
                                 smpl.col.name = "Sample",  # Default is "Sample" but you can change it
                                 phylo.div = T  # Set to true if your physeq obj has phylogenetic infor
                                 )
```

```
## Warning in `[.data.table`(tmp.dt, , `:=`(se.chao1, NULL)): Column 'se.chao1'
## does not exist to remove
```

```
# Check that scores were calculated
head(dt.alphaScores.all)
```

```
##    Sample  Shannon   Simpson Phylogenetic Richness
## 1: AQC1cm 3.552736 0.7648870     247.2830     6290
## 2: AQC4cm 3.372495 0.7397659     253.2101     6582
## 3: AQC7cm 4.027716 0.8179374     245.1008     6386
## 4:    CC1 6.776603 0.9952117     262.2629     7679
## 5:    CL3 6.576517 0.9946561     250.5354     6964
## 6:  Even1 4.083665 0.9681981     179.9377     4213
```

Normalize Alpha Scores

Save Function

```r
# Normalize Alpha Scores ------------------------------------------------
#   Description: normalizes alpha diversity scores based on their distributions
#   Input: dataframe of alpha diversity scores, metadata table
#   Output: normalized datatable of alpha scores (0 to 1)

norm_alpha_score <- function(
  alpha.base,
  sample.df,
  methods,
  smpl.col.name = "Sample"
  ){

  # Makes a copy of the dataframe you input and adds a column for your sample IDs
  model.data.base <- copy(alpha.base[alpha.base[[smpl.col.name]] %in%
                                       row.names(sample.df)])

  # Loops through the different alpha methods
  for (alpha in methods) {

    # ad.test(): Performs the Anderson-Darling test for the composite hypothesis of normality
    #   - Basically checking to see if the alpha score distribution that was calculated previously foll
    #   - Check this out for more info: ?ad.test()
    if (nortest::ad.test(model.data.base[[alpha]])$p.value <= 0.05) {

      # If the alpha scores do not follow a normal distribution, then you transform it using Tukey's (n
      #   - This will transform your data as closely to a normal distribution

      # Sub-function to transform data that isn't normally distributed using Tukey's (not Turkey's) pow
      #   - Check this out for more info: ?transformTukey()
      trans <- rcompanion::transformTukey(model.data.base[[alpha]], plotit = F, quiet = F, statistic = 1
      trans <- (trans-min(trans))/(max(trans)-min(trans))   # Fixes normalization 0 to 1

      # Runs ad.test again to see if data returns higher than 0.05 p-value, if true then it transforms
      if (nortest::ad.test(trans)$p.value > 0.05) {
        model.data.base[[alpha]] <- trans  # Transorm data with transformTukey() above
        print(paste0("Finished: ", alpha))  # Letting you know what it's working on

        # If your data is now normally distributed it will return < 0.05 p.val, and then it uses max/mi
      } else {
        model.data.base[[alpha]] <- (model.data.base[[alpha]] - min(model.data.base[[alpha]] ))/(max(mo
        print(paste0("Finished: ", alpha))  # Letting you know what it's working on
      }

    # If your data is already normally distributed, then it uses max/min values to distribute the score
    } else {
      model.data.base[[alpha]] <- (model.data.base[[alpha]] - min(model.data.base[[alpha]] ))/(max(model
      print(paste0("Finished: ", alpha))  # Letting you know what it's working on
    }
  }

  # Sends your data back normalized from 0 to 1
  return(model.data.base)
}
```

Run Function

Normalizing scores from 0 to 1 for easier comparison across metrics.

Under the hood, we use the functions `descdist` and `fitdist` (`fitdistrplus` package) to determine that the best distribution for the alpha-diversity metric scores were almost always the beta distribution. This distribution is not directly supported by the `glm` function (used in later data analysis) but is approximated by the `quasibinomial` family. These distributions only take values from 0 to 1, so we divide all alpha-diversity scores by the max score for each metric.

```
# Normalize scores from 0 to 1 for easier comparison across metrics
#   - The test will out put some statistical information about which transformations were done

dt.alphaScores.norm.all <- norm_alpha_score(alpha.base = dt.alphaScores.all,  # Unnormalized alpha scor
                                            sample.df = df.all,  # sample data frame
                                            methods = methods.alpha,  # list of alpha methods
                                            smpl.col.name = "Sample"  # Default is "Sample" but you can
                                            )
```

```
##
##     lambda      W Shapiro.p.value      A Anderson.p.value
## 358 -1.075 0.9725          0.6894 0.2223           0.8088
##
## if (lambda >  0){TRANS = x ^ lambda}
## if (lambda == 0){TRANS = log(x)}
## if (lambda <  0){TRANS = -1 * x ^ lambda}
##
## [1] "Finished: Shannon"
##
##     lambda      W Shapiro.p.value      A Anderson.p.value
## 800  9.975 0.9293         0.07446 0.5681           0.1267
##
## if (lambda >  0){TRANS = x ^ lambda}
## if (lambda == 0){TRANS = log(x)}
## if (lambda <  0){TRANS = -1 * x ^ lambda}
##
## [1] "Finished: Simpson"
##
##     lambda      W Shapiro.p.value      A Anderson.p.value
## 319  -2.05 0.9384          0.1231 0.4748           0.2207
##
## if (lambda >  0){TRANS = x ^ lambda}
## if (lambda == 0){TRANS = log(x)}
## if (lambda <  0){TRANS = -1 * x ^ lambda}
##
## [1] "Finished: Phylogenetic"
##
##     lambda    W Shapiro.p.value      A Anderson.p.value
## 341   -1.5 0.95          0.2316 0.4027           0.3332
##
## if (lambda >  0){TRANS = x ^ lambda}
## if (lambda == 0){TRANS = log(x)}
## if (lambda <  0){TRANS = -1 * x ^ lambda}
##
## [1] "Finished: Richness"
```

```
# View
head(dt.alphaScores.norm.all)
```

```
##    Sample   Shannon    Simpson Phylogenetic  Richness
## 1: AQC1cm 0.4171836 0.02138575    0.9730353 0.9433648
## 2: AQC4cm 0.3501525 0.00000000    0.9842873 0.9577726
## 3: AQC7cm 0.5641219 0.09325515    0.9686822 0.9482835
## 4:    CC1 1.0000000 0.98867491    1.0000000 1.0000000
## 5:    CL3 0.9809566 0.98288231    0.9793102 0.9743701
## 6:  Even1 0.5790998 0.73835900    0.7548947 0.7628842
```

Prepare for Plotting

This creates a combined datatable with your metadata and alpha diversity scores

```
# Create a datatable of alpha scores for melting
dt.alphaPlus.all <- dt.all[dt.alphaScores.norm.all, on = "Sample"] %>% setkeyv("Sample")

# View
head(dt.alphaPlus.all)
```

```
##    Sample  Primer Final_Barcode Barcode_truncated_plus_T Barcode_full_length
## 1: AQC1cm ILBC_16        ACAGCA                   TGCTGT         GACCACTGCTG
## 2: AQC4cm ILBC_17        ACAGCT                   AGCTGT         CAAGCTAGCTG
## 3: AQC7cm ILBC_18        ACAGTG                   CACTGT         ATGAAGCACTG
## 4:    CC1 ILBC_02        AACTCG                   CGAGTT         CATCGACGAGT
## 5:    CL3 ILBC_01        AACGCA                   TGCGTT         CTAGCGTGCGT
## 6:  Even1 ILBC_27        ACCGCA                   TGCGGT         TGACTCTGCGG
##            SampleType                                Description   Shannon
## 1: Freshwater (creek)             Allequash Creek, 0-1cm depth 0.4171836
## 2: Freshwater (creek)            Allequash Creek, 3-4 cm depth 0.3501525
## 3: Freshwater (creek)            Allequash Creek, 6-7 cm depth 0.5641219
## 4:               Soil Cedar Creek Minnesota, grassland, pH 6.1 1.0000000
## 5:               Soil Calhoun South Carolina Pine soil, pH 4.9 0.9809566
## 6:               Mock                                    Even1 0.5790998
##       Simpson Phylogenetic  Richness
## 1: 0.02138575    0.9730353 0.9433648
## 2: 0.00000000    0.9842873 0.9577726
## 3: 0.09325515    0.9686822 0.9482835
## 4: 0.98867491    1.0000000 1.0000000
## 5: 0.98288231    0.9793102 0.9743701
## 6: 0.73835900    0.7548947 0.7628842
```

Save Function

```
# Melt Data Table ------------------------------------------------------------
#   Description: Melts sample data table for easy plotting
#   Input: normalized alpha scores, alpha methods, variable names
#   Output: a melted datatable
```

```
melt_to_datatable_2 <- function(datatable1, datatable2, vars, var.name, samp.name = "Sample", val.name =

  comb.data <- datatable1[datatable2, on = (samp.name)] %>% setkeyv(samp.name)

  return( melt(
    comb.data,  # combined datatables from above
    measure.vars = vars, # Measure variables for melting. Can be missing, vector, list, or pattern-base
    variable.name = var.name, # Name for the measured variable names column.
    value.name = val.name  # Name for the molten data values column(s).
  )
  )
}
```

Run Function

```
# Melt data table for easy plotting and statistical analysis
dt.alphaPlus.all.melt <- melt_to_datatable_2(datatable1 = dt.all,
                                             datatable2 = dt.alphaScores.norm.all,
                                             vars = methods.alpha,
                                             var.name = "Alpha.Metric",
                                             samp.name = "Sample",
                                             val.name = "Alpha.Score")

# View
head(dt.alphaPlus.all.melt)
```

```
##   Sample  Primer Final_Barcode Barcode_truncated_plus_T Barcode_full_length
## 1 AQC1cm ILBC_16        ACAGCA                   TGCTGT         GACCACTGCTG
## 2 AQC4cm ILBC_17        ACAGCT                   AGCTGT         CAAGCTAGCTG
## 3 AQC7cm ILBC_18        ACAGTG                   CACTGT         ATGAAGCACTG
## 4    CC1 ILBC_02        AACTCG                   CGAGTT         CATCGACGAGT
## 5    CL3 ILBC_01        AACGCA                   TGCGTT         CTAGCGTGCGT
## 6  Even1 ILBC_27        ACCGCA                   TGCGGT         TGACTCTGCGG
##           SampleType                                Description Alpha.Metric
## 1 Freshwater (creek)             Allequash Creek, 0-1cm depth      Shannon
## 2 Freshwater (creek)            Allequash Creek, 3-4 cm depth      Shannon
## 3 Freshwater (creek)            Allequash Creek, 6-7 cm depth      Shannon
## 4               Soil Cedar Creek Minnesota, grassland, pH 6.1      Shannon
## 5               Soil Calhoun South Carolina Pine soil, pH 4.9      Shannon
## 6               Mock                                    Even1      Shannon
##   Alpha.Score
## 1   0.4171836
## 2   0.3501525
## 3   0.5641219
## 4   1.0000000
## 5   0.9809566
## 6   0.5790998
```

DONE!

Now if you want to go on to plotting, here are some examples.

Plotting

Assign data

I like to assign temporary data variables for each plot/statistical analysis chunk, because in my experience it keeps the code nimble and flexible.

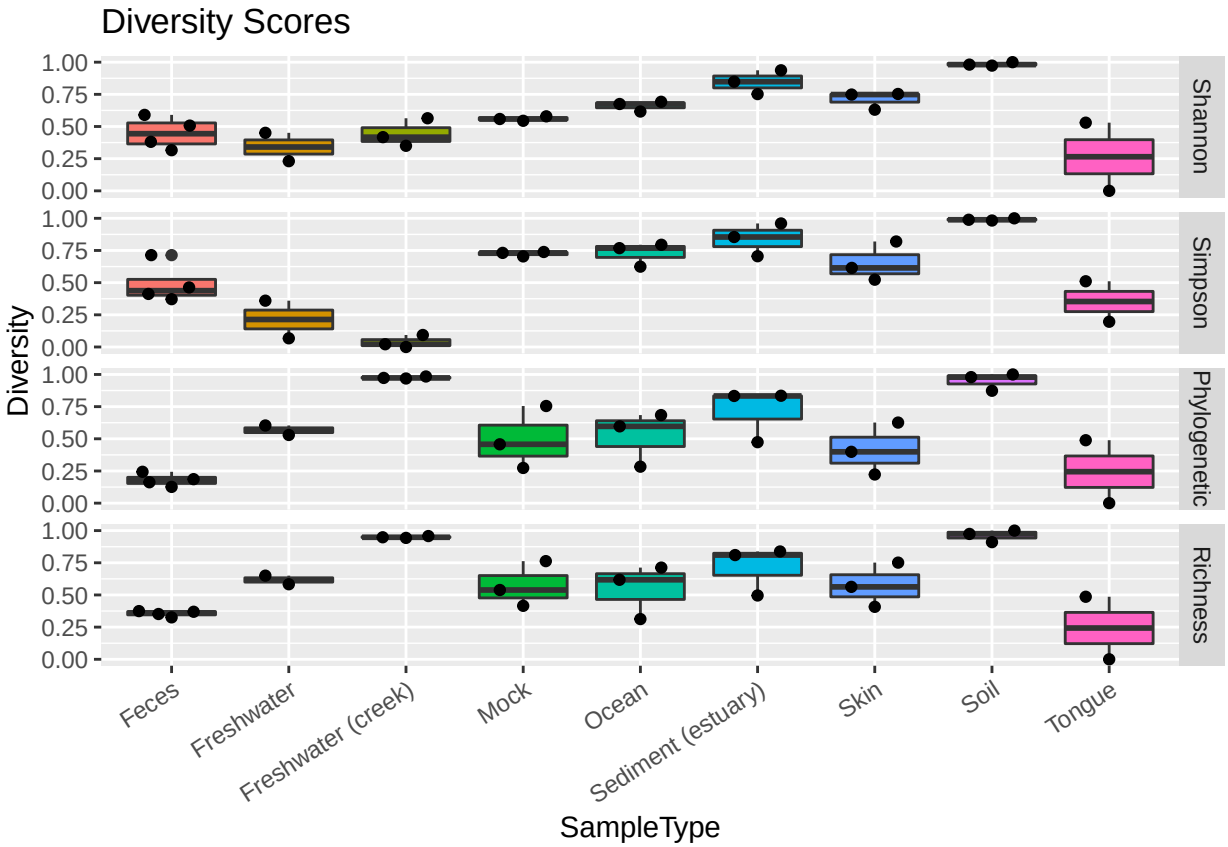
This avoids situations where you have dozens of variables for each figure, table, etc. Instead, you can add a function at the end of the chunks to export your variables, figures, and tables with unique names, if you want.

```
# Assign a temporary data variable
data <- dt.alphaPlus.all.melt
```

Plot

```
plot <- ggplot(data, aes(x = SampleType, y=Alpha.Score)) +
  geom_boxplot(aes(fill = SampleType)) +
  ggbeeswarm::geom_quasirandom() +  # spaces the dots out nicely
  facet_grid(Alpha.Metric ~ .) +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 33, hjust = 1, vjust=1)
        ) +
  labs(
    title = "Diversity Scores",
    # caption = "",
    y = "Diversity", #paste0("Diversity (",x ,")"),
    x = "SampleType"
  )

plot
```


Diversity Scores
Diversity
1.00
0.75
0.50
0.25
0.00
Shannon
Simpson
Phylogenetic
Richness
Feces
Freshwater
Freshwater (creek)
Mock
Ocean
Sediment (estuary)
Skin
Soil
Tongue
SampleType