Alpha Diversity Norm and Transform

Michael Sieler

17 May, 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.

If you use this code, please cite Keaton Stagaman's package phyloseqCompanion: `Stagaman, K. phyloseqCompanion: Provides additional functions to work with phyloseq objects. (2022). github.com/kstagaman/phyloseqCompanion`.

Setup Environment

```r
# 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(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(flextable)  # for making pretty tables, flextable()
library(gridExtra)  # use marrangeGrob, for combining plots
library(ggbeeswarm)  # Pretty dots on box plots

## Microbiome Analysis
library(broom)  # Helps for tidying up datatables. tidy()
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

## Add last so, other packages don't "mask" tidy functions
library(tidyverse)  # Making your code look pretty and tidy
```

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
```

```r
# colnames(sample_data(ps.all)) <- c()  # c("colName1", "colName2", ...)

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

Create sample data tables/frames

```r
# 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.

```r
# 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 f
  phylo.div = T  # Only set to false if you don't have phylogenetic information attached to your phylos
){

  # Calculates alpha scores
  tmp.dt <- phyloseq::estimate_richness(
    physeq = physeq,  # Physeq object
    measures = methods
  ) %>% 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)){

    print("Calculating phylogenetic diversity, takes a while...")

    # 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)) %>%
      select(-SR) %>%  # Deselects "SR" (Species Richness) since we already included it.
      rename(Phylogenetic = PD) %>%  # renames "PD" to "Phylogenetic"
      as.data.table(keep.rownames = smpl.col.name) %>% setkeyv(smpl.col.name)  # set col name for sampl

    tmp.dt <- tmp.dt %>%
      inner_join(., phy.dt, by = smpl.col.name)
    # return to sender
    return (tmp.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("Observed", "Shannon", "Simpson", # Non-phylogenetic measures, add additional measur
                   "Phylogenetic") %>%  # Phylogenetic measures
                  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

## [1] "Calculating phylogenetic diversity, takes a while..."
```

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

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

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 = 
      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
## 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: Observed"
##
##     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"
```

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

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

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 %>%
                      inner_join(., dt.alphaScores.norm.all, by = "Sample")

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

```
##     Sample  Primer Final_Barcode Barcode_truncated_plus_T Barcode_full_length
## 1:     CL3 ILBC_01        AACGCA                   TGCGTT         CTAGCGTGCGT
## 2:     CC1 ILBC_02        AACTCG                   CGAGTT         CATCGACGAGT
## 3:     SV1 ILBC_03        AACTGT                   ACAGTT         GTACGCACAGT
## 4: M31Fcsw ILBC_04        AAGAGA                   TCTCTT         TCGACATCTCT
## 5: M11Fcsw ILBC_05        AAGCTG                   CAGCTT         CGACTGCAGCT
## 6: M31Plmr ILBC_07        AATCGT                   ACGATT         CGAGTCACGAT
##    SampleType                                Description  Observed   Shannon
## 1:       Soil   Calhoun South Carolina Pine soil, pH 4.9 0.9743701 0.9809566
## 2:       Soil   Cedar Creek Minnesota, grassland, pH 6.1 1.0000000 1.0000000
## 3:       Soil Sevilleta new Mexico, desert scrub, pH 8.3 0.9104283 0.9732006
## 4:      Feces    M3, Day 1, fecal swab, whole body study 0.3692686 0.5070572
## 5:      Feces   M1, Day 1, fecal swab, whole body study  0.3259024 0.3159624
## 6:       Skin    M3, Day 1, right palm, whole body study 0.5628513 0.6306641
##      Simpson Phylogenetic
## 1: 0.9828823    0.9793102
## 2: 0.9886749    1.0000000
## 3: 1.0000000    0.8735045
## 4: 0.4627942    0.1262508
## 5: 0.3716700    0.1620847
## 6: 0.5230616    0.3989199
```

Combine Alpha Scores into One Column

```
# Melt data table for easy plotting and statistical analysis

dt.alphaPlus.all.melt <- dt.alphaPlus.all %>%
                            pivot_longer(cols = methods.alpha,    # List of column names containing alp
                                         names_to = "Alpha.Metric",   # Column name for alpha metrics
                                         values_to = "Alpha.Score"  # Column name for alpha scores
                                         ) %>%
                            arrange(Alpha.Metric)  # Sort datatable by alpha metric
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(methods.alpha)` instead of `methods.alpha` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

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

```
## # A tibble: 6 x 9
##   Sample  Primer  Final_Barcode Barcode_truncated_p~ Barcode_full_le~ SampleType
##   <chr>   <fct>   <fct>         <fct>                <fct>            <fct>
## 1 CL3     ILBC_01 AACGCA        TGCGTT               CTAGCGTGCGT      Soil
## 2 CC1     ILBC_02 AACTCG        CGAGTT               CATCGACGAGT      Soil
## 3 SV1     ILBC_03 AACTGT        ACAGTT               GTACGCACAGT      Soil
## 4 M31Fcsw ILBC_04 AAGAGA        TCTCTT               TCGACATCTCT      Feces
## 5 M11Fcsw ILBC_05 AAGCTG        CAGCTT               CGACTGCAGCT      Feces
## 6 M31Plmr ILBC_07 AATCGT        ACGATT               CGAGTCACGAT      Skin
## # ... with 3 more variables: Description <fct>, Alpha.Metric <chr>,
## #   Alpha.Score <dbl>
```

DONE!

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

Plotting

Assign data

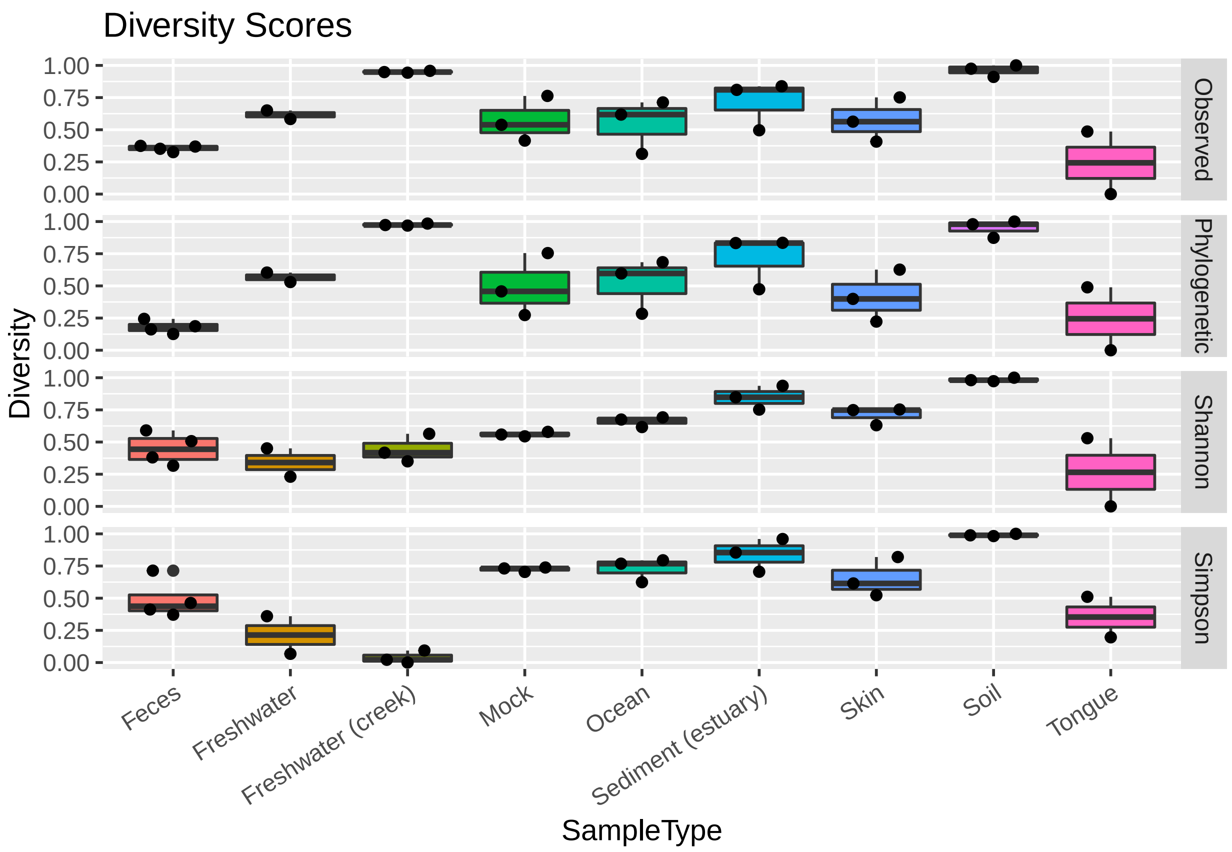
I like to assign temporary data variables for each plot/statistical analysis chunk, because in my experience it keeps the code nimble and flexible.

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
```

```r
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
```



Plot

Statistical Analysis

Functions

```r
# gen_glm_anova ---------------------------------------------------
#   Description: Runs an anova test on a generalized linear model
#   Input: model, alpha methods, filter pvalues
#   Output: anova statistical results

gen_glm_anova <- function(tmp.mod, tmp.metric, filt.pval = 1){
  return(Anova(tmp.mod, type = 2) %>%
           tidy() %>%
           mutate(sig = ifelse(p.value <= 0.05, "*", "")) %>%
           mutate(metric = tmp.metric, .before = 1) %>%
           filter((p.value < filt.pval | is.na(p.value)) & df > 0) %>%
           arrange(desc(statistic))  # highest to lowest effect size
  )
}


# Stats Table -------------------------------------------------------
#   Description: produces a statistical table from anova stats
#   Input: anova results, variables, terms
#   Output: statistical table of anova results

stats_table <- function(dataframe, terms = NA, hline.num = NA, formula = NA, stat.desc = F){

  # Arrange dataframe by most to least significant
  if(!"sig" %in% colnames(dataframe)){
    dataframe<- dataframe %>%
      tidy() %>%
      mutate(sig = ifelse(p.value <= 0.05, "*", "")) %>%
      # arrange(desc(statistic))  # highest to lowest effect size
      arrange(if(!isTRUE(stat.desc)) TRUE else desc(statistic))
  }

  if(is.na(hline.num)){
    hline.num = seq(1, nrow(dataframe) -1)
  }


  # caption <- paste0("beta score ~ (", paste0(var, collapse = "+"), ")^", terms)
  return (dataframe %>%
            flextable() %>%
            set_caption(caption = ifelse(is.na(formula), "", formula)) %>%
            #set_caption(caption = table.caption(caption)) %>%  # Uses autoNumCaptions
            # align(j = c(1, ncol(dataframe)), align = "left") %>%
            align(j = 2:5, align = "right") %>%
            colformat_double(j = 3, digits = 2) %>%
            colformat_double(j = 5, digits = 3) %>%
            merge_v(j = 1) %>%
            hline(i = hline.num, j = NULL, border = NULL, part = "body") %>%
            set_formatter(values = list("p.value" = p_val_format) ) %>%
            autofit()
  )
}


# P-value Format ----------------------------------------------------------
#   Description: Formats P-values for summary statistic tables
#     * P-values below a certain threshold will appear as "<0.001"
#   Input:
#   Output:

p_val_format <- function(x){
  z <- scales::pvalue_format()(x)
  z[!is.finite(x)] <- ""
  z
}
```

Test Results

```
# Build Statistical Model
mod.list <- lapply(methods.alpha, function(alpha){
  glm( formula = "Alpha.Score ~ SampleType",  # interaction: formula = "Alpha.Score ~ <Variable1>*<Vari
       data = subset(data, Alpha.Metric == alpha),
       family = "quasibinomial")
})


# Statistical Table of Model
lapply(methods.alpha, function(x){
  mod.list[[x]] %>% summary()
})
```

```
## $Observed
##
## Call:
## glm(formula = "Alpha.Score ~ SampleType", family = "quasibinomial",
##     data = subset(data, Alpha.Metric == alpha))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.74628  -0.06827   0.01093   0.20043   0.52522
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -0.5952     0.3563  -1.670  0.11314
## SampleTypeFreshwater            1.0705     0.6108   1.753  0.09767 .
## SampleTypeFreshwater (creek)    3.5356     0.9698   3.646  0.00200 **
## SampleTypeMock                  0.8865     0.5342   1.659  0.11536
## SampleTypeOcean                 0.7865     0.5325   1.477  0.15791
## SampleTypeSediment (estuary)    1.5133     0.5631   2.687  0.01558 *
## SampleTypeSkin                  0.8934     0.5344   1.672  0.11285
## SampleTypeSoil                  3.8157     1.0850   3.517  0.00265 **
## SampleTypeTongue               -0.5408     0.6657  -0.812  0.42780
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.1163446)
```

```
##
##     Null deviance: 8.5833  on 25  degrees of freedom
## Residual deviance: 2.2002  on 17  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 6
##
##
## $Shannon
##
## Call:
## glm(formula = "Alpha.Score ~ SampleType", family = "quasibinomial",
##     data = subset(data, Alpha.Metric == alpha))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.78456  -0.12604   0.00114   0.11163   0.56060
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.20651    0.28120  -0.734  0.47271
## SampleTypeFreshwater        -0.45348    0.50321  -0.901  0.38008
## SampleTypeFreshwater (creek) -0.01916    0.42980  -0.045  0.96496
## SampleTypeMock               0.45029    0.43006   1.047  0.30974
## SampleTypeOcean              0.87494    0.44213   1.979  0.06426 .
## SampleTypeSediment (estuary)  1.90778    0.52813   3.612  0.00215 **
## SampleTypeSkin               1.10490    0.45375   2.435  0.02620 *
## SampleTypeSoil               4.37226    1.34618   3.248  0.00473 **
## SampleTypeTongue            -0.81406    0.52911  -1.539  0.14232
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.07823719)
##
##     Null deviance: 7.4148  on 25  degrees of freedom
## Residual deviance: 1.5621  on 17  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 7
##
##
## $Simpson
##
## Call:
## glm(formula = "Alpha.Score ~ SampleType", family = "quasibinomial",
##     data = subset(data, Alpha.Metric == alpha))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.40325  -0.20996  -0.03204   0.14831   0.45434
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.03918    0.29078  -0.135  0.89441
## SampleTypeFreshwater         -1.26317    0.57965  -2.179  0.04367 *
## SampleTypeFreshwater (creek) -3.18642    0.92256  -3.454  0.00303 **
## SampleTypeMock                1.00661    0.47512   2.119  0.04915 *
## SampleTypeOcean               1.02900    0.47664   2.159  0.04545 *
## SampleTypeSediment (estuary)  1.69799    0.54246   3.130  0.00610 **
## SampleTypeSkin                0.66949    0.45697   1.465  0.16115
## SampleTypeSoil                4.68812    1.75629   2.669  0.01618 *
## SampleTypeTongue             -0.56465    0.51911  -1.088  0.29190
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.08452166)
##
##     Null deviance: 11.8718  on 25  degrees of freedom
## Residual deviance:  1.5201  on 17  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 7
##
##
## $Phylogenetic
##
## Call:
## glm(formula = "Alpha.Score ~ SampleType", family = "quasibinomial",
##     data = subset(data, Alpha.Metric == alpha))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.74892  -0.12786   0.00155   0.24847   0.53135
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -1.5191     0.4994  -3.042  0.00737 **
## SampleTypeFreshwater           1.7876     0.7408   2.413  0.02739 *
## SampleTypeFreshwater (creek)   5.1965     1.5120   3.437  0.00315 **
## SampleTypeMock                 1.5000     0.6674   2.247  0.03817 *
## SampleTypeOcean                1.6050     0.6677   2.404  0.02791 *
## SampleTypeSediment (estuary)   2.4336     0.6995   3.479  0.00287 **
## SampleTypeSkin                 1.1795     0.6717   1.756  0.09707 .
## SampleTypeSoil                 4.4835     1.1401   3.933  0.00107 **
## SampleTypeTongue               0.3912     0.8045   0.486  0.63299
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.147004)
##
##     Null deviance: 11.8272  on 25  degrees of freedom
## Residual deviance:  2.7324  on 17  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 6
```

```
# Statistical Power of Model
lapply(methods.alpha, function(x){
  mod.list[[x]] %>% Anova(type = 2)
})
```

```
## $Observed
## Analysis of Deviance Table (Type II tests)
##
## Response: Alpha.Score
##            LR Chisq Df Pr(>Chisq)
## SampleType   54.863  8  4.694e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $Shannon
## Analysis of Deviance Table (Type II tests)
##
## Response: Alpha.Score
##            LR Chisq Df Pr(>Chisq)
## SampleType   74.808  8   5.39e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $Simpson
## Analysis of Deviance Table (Type II tests)
##
## Response: Alpha.Score
##            LR Chisq Df Pr(>Chisq)
## SampleType   122.47  8  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $Phylogenetic
## Analysis of Deviance Table (Type II tests)
##
## Response: Alpha.Score
##            LR Chisq Df Pr(>Chisq)
## SampleType   61.868  8  2.002e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Publication quality table
stats_table( lapply(methods.alpha, function(x){
  gen_glm_anova(mod.list[[x]], x)
}) %>% bind_rows() )
```

```
## Warning: Warning: fonts used in `flextable` are ignored because the `pdflatex`
## engine is used and not `xelatex` or `lualatex`. You can avoid this warning
## by using the `set_flextable_defaults(fonts_ignore=TRUE)` command or use a
## compatible engine by defining `latex_engine: xelatex` in the YAML header of the
## R Markdown document.
```

Table 1:

metric	term	statistic	df	p.value	sig
Observed	SampleType	54.86	8	<0.001	*
Shannon	SampleType	74.81	8	<0.001	*
Simpson	SampleType	122.47	8	<0.001	*
Phylogenetic	SampleType	61.87	8	<0.001	*