Microbiota Analysis in R

Kim Dill-McFarland

March 20, 2017

Updated April 15, 2017

Online version available at http://rpubs.com/dillmcfarlan/R_microbiotaSOP (http://rpubs.com/dillmcfarlan/R_microbiotaSOP)

Tips for this workshop

1. If you have any issues in R, type ??command into the console where “command” is the function you are having issues with and a help page will come up.
2. Lines starting with `#` are comments that are for the reader’s benefit. These lines are not code and do not need to be entered into the console.
3. GREY boxes contain code that you can copy and paste to run on your machine.

```
#GREY box
```

4. WHITE boxes contain sample output of this code, and nothing will happen if you try to copy it into your console.

```
WHITE box
```

5. Basic R code you may find useful:
    a. Matrices/data frames are designated by [,] where it is [rows, columns]
    b. | is or
    c. & is and

Introduction

Written for R v3.3.2 in RStudio v1.0.136

Goal

The goal of this tutorial is to demonstrate basic analyses of microbiota data to determine if and how communities differ by variables of interest. In general, this pipeline can be used for any microbiota data set that has been clustered into operational taxonomic units (OTUs).

This tutorial assumes some basic statistical knowledge. Please consider if your data fit the assumptions of each test (normality? equal sampling? Etc.). If you are not familiar with statistics at this level, we strongly recommend collaborating with someone who is. The incorrect use of statistics is a pervasive and serious problem in the sciences so don’t become part of the problem! That

said, this is an introductory tutorial and there are many, many further analyses that can be done with microbiota data. Hopefully, this is just the start for your data!

Data

The data used here were created using 2x250 bp amplicon sequencing of the bacterial V4 region of the 16S rRNA gene on the Illumina MiSeq platform. The full data set is in Dill-McFarland *et al*. Sci Rep 7: 40864 (https://www.ncbi.nlm.nih.gov/pubmed/28098248). Here, we will use a subset of samples. Specifically, we will be correlating the fecal bacterial microbiota of 8 dairy calves at different ages (2 weeks, 8 weeks, 1 year) to variables like weight gain (average daily gain in kg, ADGKG) and gastrointestinal short chain fatty acids (SCFA).

Files

We will use the following files created using the Microbiota Processing in mothur: Standard Operating Procedure (SOP) (https://rpubs.com/dillmcfarlan/mothurSOP).

- example.final.nn.unique_list.0.03.norm.shared (OTU table)
- example.final.nn.unique_list.0.03.cons.taxonomy (Taxonomy of OTUs)

We will also be using tab-delimited metadata and SCFA files created in Excel. The metadata includes our metadata (like age and ADGKG) as well as alpha-diversity metrics from `example.final.nn.unique_list.0.03.norm.groups.summary` calculated in mothur. The SCFA table is the mM concentrations of different SCFAs in rumen (stomach) liquids from 1-year-old animals.

- example.metadata.txt
- example.SCFA.txt

Finally, we will be loading a number of custom scripts from `Steinberger_scripts` and some a pre-calculated OTU tree `NJ.tree.RData`. The information for creating this tree is provided in this tutorial.

Get set up

Download and install

- Base R: http://cran.mtu.edu/ (http://cran.mtu.edu/)
- RStudio: https://www.rstudio.com/products/rstudio/download3/ (https://www.rstudio.com/products/rstudio/download3/)
- Packages: Open RStudio on your computer. If you have not already downloaded these packages, go to the lower right quadrant of your screen and open the Package tab. Click “download” and search for the package you want to download.
    - `ape`
    - `dplyr`
    - `ggplot2`
    - `gplots`
    - `lme4`
    - `phangorn`
    - `plotly`
    - `tidyr`
    - `vegan`
    - `VennDiagram`
    - `venneuler`
    - `phyloseq` (`phyloseq` is not on CRAN, so we have to call it manually. See below.)

Copy and paste the following into your console.

```
source("https://bioconductor.org/biocLite.R")
```

```
## Bioconductor version 3.4 (BiocInstaller 1.24.0), ?biocLite for help
```

```
biocLite("phyloseq")
```

```
## BioC_mirror: https://bioconductor.org
```

```
## Using Bioconductor 3.4 (BiocInstaller 1.24.0), R 3.3.3 (2017-03-06).
```

```
## Installing package(s) 'phyloseq'
```

```
## package 'phyloseq' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Kim\AppData\Local\Temp\RtmpWsR86d\downloaded_packages
```

```
## installation path not writeable, unable to update packages: cluster,
##   lattice, survival
```

```
## Old packages: 'Biostrings', 'curl', 'IRanges', 'S4Vectors', 'XVector'
```

Note: If you are having trouble installing packages, turn off your computer's firewall temporarily.

Organization

All of our analyses will be organized into a "Project".

Make a new project by selecting File->New project. Select "New Directory" and "Empty Project". Name the project "Microbiota_Analysis_BRC" and save the project to your Desktop. Place all of your files for this analysis in the folder created on the Desktop

Create a new R script (File->New file->R script) to save your code. This file will automatically be saved in the project folder.

Now your screen should look like this

- Upper left: Where you type and save the code you want to run.
- Upper right: Files you load into and create in R. To view one, click on it and it will open in the upper left pane.
- Lower left: The console. Where commands and outputs run (similar to the one mothur window).
- Lower right: Variable. Explore the different tabs.

Data manipulation

Load Packages

The "library" command tells R to open the package you want to use. You need to do this every time you open R.

```
#Analyses of Phylogenetics and Evolution package. Required for tree calculations to be used with phyloseq
library(ape)

#This package will also help us more easily manipulate our data
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
#Graphing package used in phyloseq. To edit the default setting of a plot, you need to use functions in this
 package.
library(ggplot2)

#This package is used to calculate and plot Venn diagrams as well as heatmaps
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
#Linear mixed-effects models like repeated measures analysis
library(lme4)
```

```
## Loading required package: Matrix
```

```
#used to read in mothur-formatted files
library(phangorn)

#The phyloseq package seeks to address issues with multiple microbiome analysis packages by providing a set
 of functions that internally manage the organizing, linking, storing, and analyzing of phylogenetic sequenc
ing data. In general, this package is used for UniFrac analyses.
library(phyloseq)

#A package to create interactive web graphics of use in 3D plots
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

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

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
#This package will help us more easily manipulate our data, which are matrices
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:Matrix':
##
##     expand
```

```
#The vegan package provides tools for descriptive community ecology. It has most basic functions of diversit
y analysis, community ordination and dissimilarity analysis. In general, this package is used for Bray-Curti
s and Jaccard analyses.
library(vegan)
```

```
## Loading required package: permute
```

```
## Loading required package: lattice
```

```
## This is vegan 2.4-3
```

```
##
## Attaching package: 'vegan'
```

```
## The following objects are masked from 'package:phangorn':
##
##     diversity, treedist
```

```
#Pretty Venn disgrams
library(VennDiagram)
```

```
## Loading required package: grid
```

```
## Loading required package: futile.logger
```

```
##
## Attaching package: 'VennDiagram'
```

```
## The following object is masked from 'package:ape':
##
##     rotate
```

```
library(venneuler)
```

```
## Loading required package: rJava
```

Load Data

In the code, the text before = is what the file will be called in R. Make this short but unique as this is how you will tell R to use this file in later commands.

- header: tells R that the first row is column names, not data
- row.names: tells R that the first column is row names, not data
- sep: tells R that the data are tab-delimited. If you had a comma-delimited file, you would us `sep=","`

```
#OTU table (shared file)
OTU = read.table("example.final.an.unique_list.0.03.norm.shared", header=TRUE, sep="\t")

#Taxonomy of each OTU
tax = read.table("example.final.an.unique_list.0.03.cons.taxonomy", header=TRUE, sep="\t")

#Metadata. Since we made this in Excel, not mothur, we can use the "row.names" modifier to automatically nam
e the rows by the values in the first column (sample names)
meta = read.table("example.metadata.txt", header=TRUE, row.names=1, sep="\t")

#SCFA data
SCFA = read.table("example.SCFA.txt", header=TRUE, row.names=1, sep="\t")
```

Clean up the data

You can look at your data by clicking on it in the upper-right quadrant “Environment”

There are several unneeded columns and incorrect formatting in the tables as they were output by mothur. We will now fix them.

OTU table

We need to use the “Group” column as the row names so that it will match our metadata

```
row.names(OTU) = OTU$Group
```

We then need to remove the “label”, “numOTUs”, and “Group” columns as they are not OTU counts like the rest of the table

```
OTU.clean = OTU[,-which(names(OTU) %in% c("label", "numOtus", "Group"))]
```

Taxonomy table

For the taxonomy table, we name the rows by the OTU #

```
row.names(tax) = tax$OTU
```

Remove all the OTUs that don’t occur in our OTU.clean data set

```
tax.clean = tax[row.names(tax) %in% colnames(OTU.clean),]
```

We then need to separate the “taxonomy” column so that each level (*i.e.* Domain, Phylum, etc) is in it’s own column. We do this with a special command “separate” from the `tidyr` package

```
tax.clean = separate(tax.clean, Taxonomy, into = c("Domain", "Phylum", "Class", "Order", "Family", "Genus",
"Species", "Strain"), sep=";")
```

Finally, we remove the “Size” and “Strain” columns as well as “OTU” since these are now the row names

```
tax.clean = tax.clean[,-which(names(tax.clean) %in% c("Size", "Strain", "OTU"))]
```

Metadata and SCFA tables

These tables do not require any modification since I created them in Excel exactly as I need them for this R analysis.

Order the data

To make viewing and using the data easier, we will make sure our tables have samples (rows) in the same order. Since OTU.clean, meta, and SCFA have sample names as row names, we order by these.

```
OTU.clean = OTU.clean[order(row.names(OTU.clean)),]
meta = meta[order(row.names(meta)),]
SCFA = SCFA[order(row.names(SCFA)),]
```

Our taxonomy table is already in order from OTU1 to OTUN so we do not need to order it.

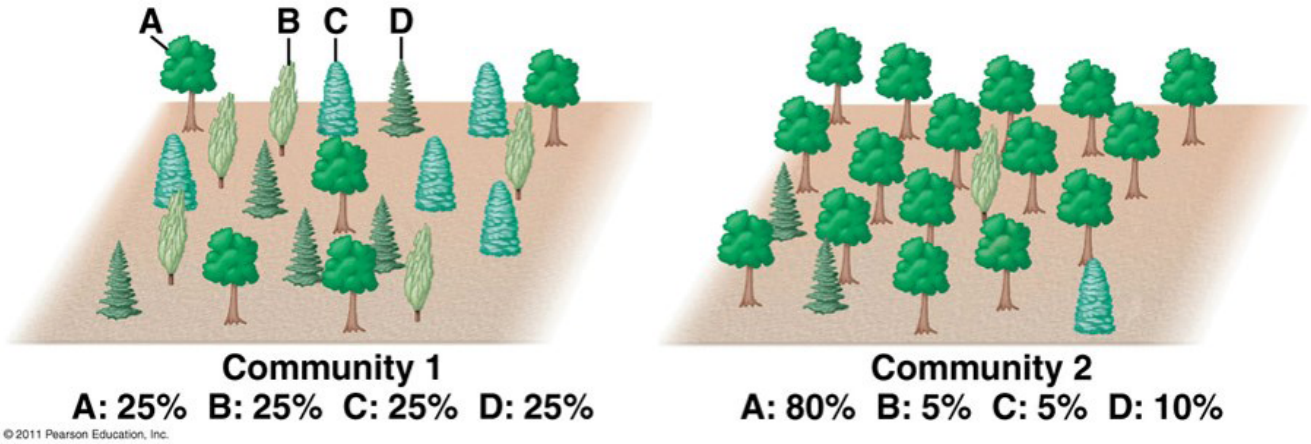
Set seed

We will be running some processes that rely on the random number generater. To make your analysis reproducible, we set the random seed.

```
set.seed(8765)
```

Alpha-diversity

Alpha-diversity is within sample diversity. It is how many different species (OTUs) are in each sample (richness) and how evenly they are distributed (evenness), which together are diversity. Each sample has one value for each metric.

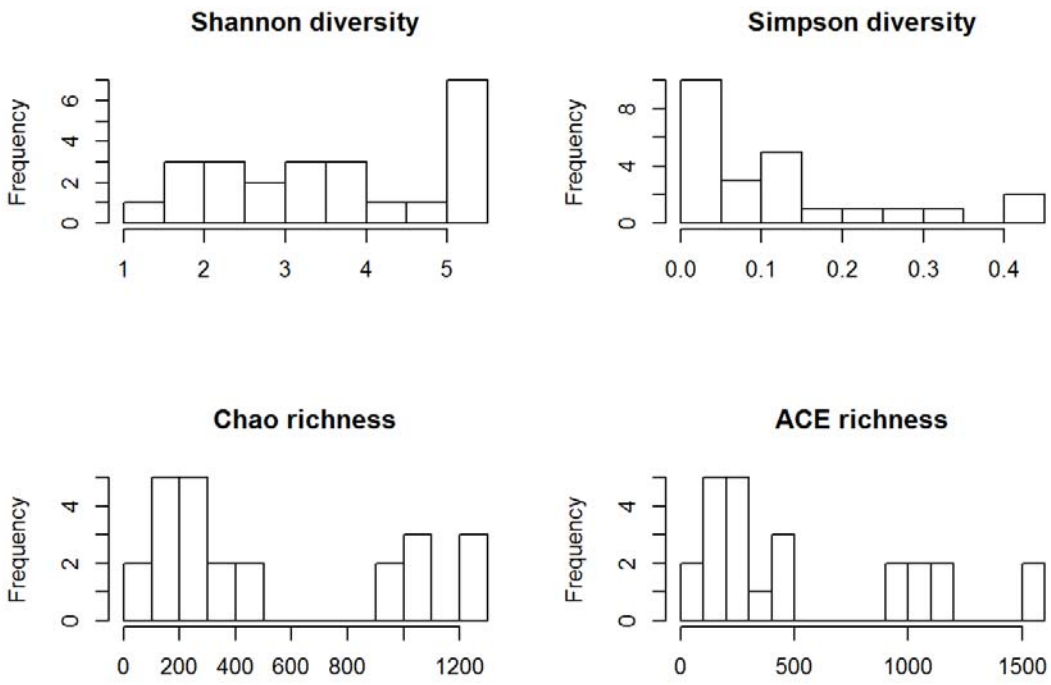


This image illustrates richness vs. diversity. Both forests have the same richness (4 tree species) but Community 1 has much more even distribution of the 4 species while Community 2 is dominated by tree species A. This makes Community 1 more diverse than Community 2.

Explore alpha metrics

Now we will start to look at our data. We will first start with alpha-diversity and richness. Let's plot some common ones here.

```
#Create 2x2 plot environment so that we can see all 4 metrics at once.
par(mfrow = c(2, 2))

#Then plot each metric.
hist(meta$shannon, main="Shannon diversity", xlab="", breaks=10)
hist(meta$simpson, main="Simpson diversity", xlab="", breaks=10)
hist(meta$chao, main="Chao richness", xlab="", breaks=15)
hist(meta$ace, main="ACE richness", xlab="", breaks=15)
```



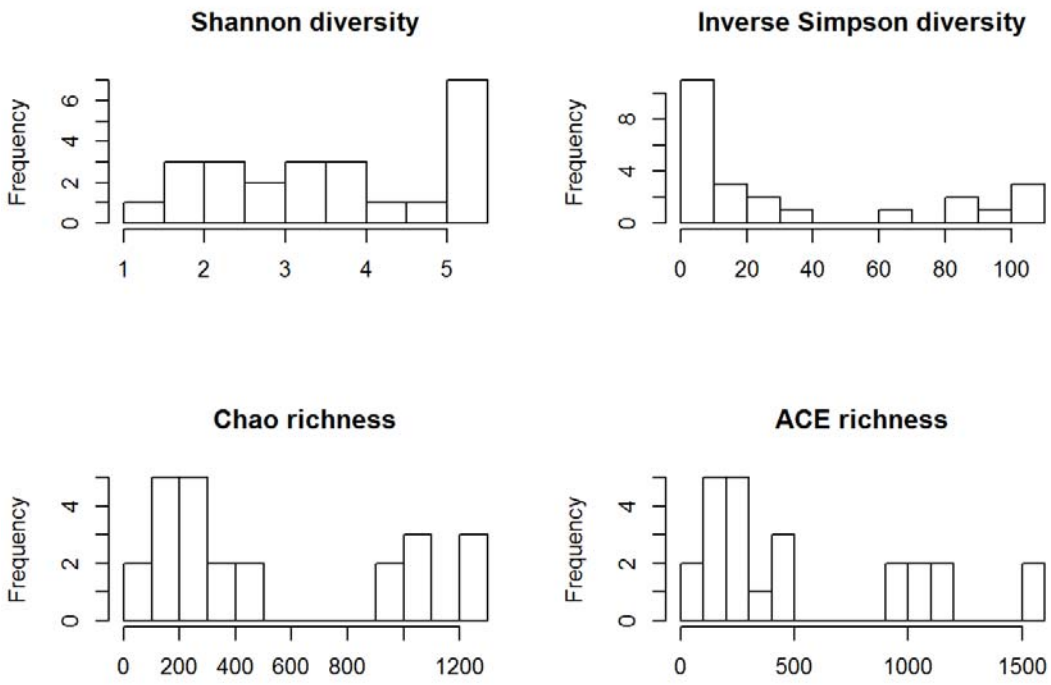
You want the data to be roughly normal so that you can run ANOVA or t-tests. If it is not normally distributed, you will need to consider non-parametric tests such as Kruskal-Wallis.

Here, we see that none of the data are normally distributed. This occurs with the subset but not the full data set because I've specifically selected samples with divergent alpha metrics. In general, you will see roughly normal data for Shannon's diversity as well as most richness metrics. Simpson's diversity, on the other hand, is usually skewed as seen here.

So most will use inverse Simpson (1/Simpson) instead. This not only increases normalcy but also makes the output more logical as a higher inverse Simpson value corresponds to higher diversity.

Let's look at inverse Simpson instead.

```
#Create 2x2 plot environment
par(mfrow = c(2, 2))

#Plots
hist(meta$shannon, main="Shannon diversity", xlab="", breaks=10)
hist(1/meta$simpson, main="Inverse Simpson diversity", xlab="", breaks=10)
hist(meta$chao, main="Chao richness", xlab="", breaks=15)
hist(meta$ace, main="ACE richness", xlab="", breaks=15)
```



Now we see a bimodal distribution for Simpson similar to the richness metrics.

To test for normalcy statistically, we can run the Shapiro-Wilk test of normality.

```
shapiro.test(meta$shannon)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  meta$shannon
## W = 0.91511, p-value = 0.0456
```

```
shapiro.test(1/meta$simpson)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  1/meta$simpson
## W = 0.74821, p-value = 4.69e-05
```

```
shapiro.test(meta$chao)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  meta$chao
## W = 0.80636, p-value = 0.0003749
```

```
shapiro.test(meta$ace)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  meta$ace
## W = 0.83017, p-value = 0.0009573
```

We see that, as expected from the graphs, none are normal.

However, our sample size is small and normalcy tests are very sensitive for small data-sets. In fact, you can run Shapiro-Wilk on a list of 50 values randomly sampled from the R-generated normal distribution and find that they are not normal (even though we know that they are!)

So, what does this mean for our purposes? Well, we should run statistical tests that don't assume our data is normal, because we don't have any evidence (graphs, Shapiro-Wilk) that it is normal. For demonstration purposes, though, we will run other tests as well.

Overall, for alpha-diversity:

- ANOVA, t-test, or general linear models with the normal distribution are used when the data is roughly normal
- Kruskal-Wallis, Wilcoxon rank sum test, or general linear models with another distribution are used when the data is not normal

Our main variables of interest are

- AgeGroup: 2w, 8w, 1yr
- ADGKG: 0.05-1.56 kg gained per day (average daily gain kg)

Categorical variables

Now that we know which tests can be used, let's run them.

Normally distributed metrics

Since it's the closest to normalcy, we will use **Shannon's diversity** as an example. First, we will test age, which is a categorical variable with more than 2 levels. Thus, we run ANOVA. If age were only two levels, we could run a t-test

Does age impact the Shannon diversity of the fecal microbiota?

```
#Run the ANOVA and save it as an object
aov.shannon.age = aov(shannon ~ AgeGroup, data=meta)
#Call for the summary of that ANOVA, which will include P-values
summary(aov.shannon.age)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## AgeGroup     2  42.98  21.489   103.4 1.35e-11 ***
## Residuals   21   4.36   0.208
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

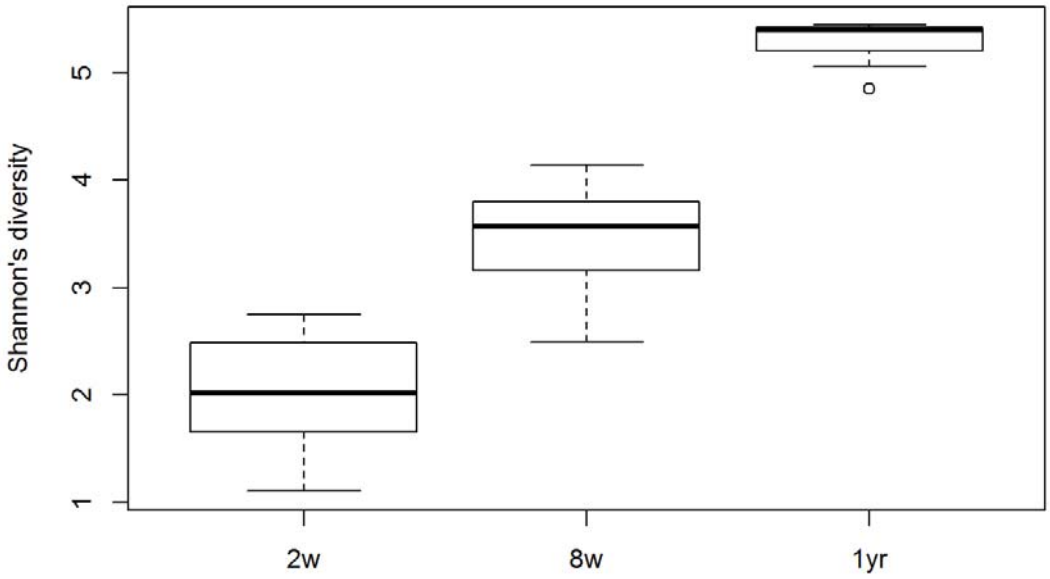
To do all the pairwise comparisons between groups and correct for multiple comparisons, we run Tukey's honest significance test of our ANOVA.

```
TukeyHSD(aov.shannon.age)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = shannon ~ AgeGroup, data = meta)
##
## $AgeGroup
##             diff        lwr       upr   p adj
## 2w-1yr -3.270063 -3.8446230 -2.695503 0.0e+00
## 8w-1yr -1.830903 -2.4054628 -1.256342 2.0e-07
## 8w-2w   1.439160  0.8646001  2.013720 8.5e-06
```

We clearly see that all age groups have significantly different diversity. When we plot the data, we see that diversity increases as the animals age.

```
#Re-order the groups because the default is 1yr-2w-8w
meta$AgeGroup.ord = factor(meta$AgeGroup, c("2w","8w","1yr"))
#Return the plot area to 1x1
par(mfrow = c(1, 1))
#Plot
boxplot(shannon ~ AgeGroup.ord, data=meta, ylab="Shannon's diversity")
```



Non-normally distributed metrics

We will use **Chao's richness estimate** here. Since age is categorical, we use Kruskal-Wallis (non-parametric equivalent of ANOVA). If we have only two levels, we would run Wilcoxon rank sum test (non-parametric equivalent of t-test)

```
kruskal.test(chao ~ AgeGroup, data=meta)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  chao by AgeGroup
## Kruskal-Wallis chi-squared = 19.28, df = 2, p-value = 6.507e-05
```

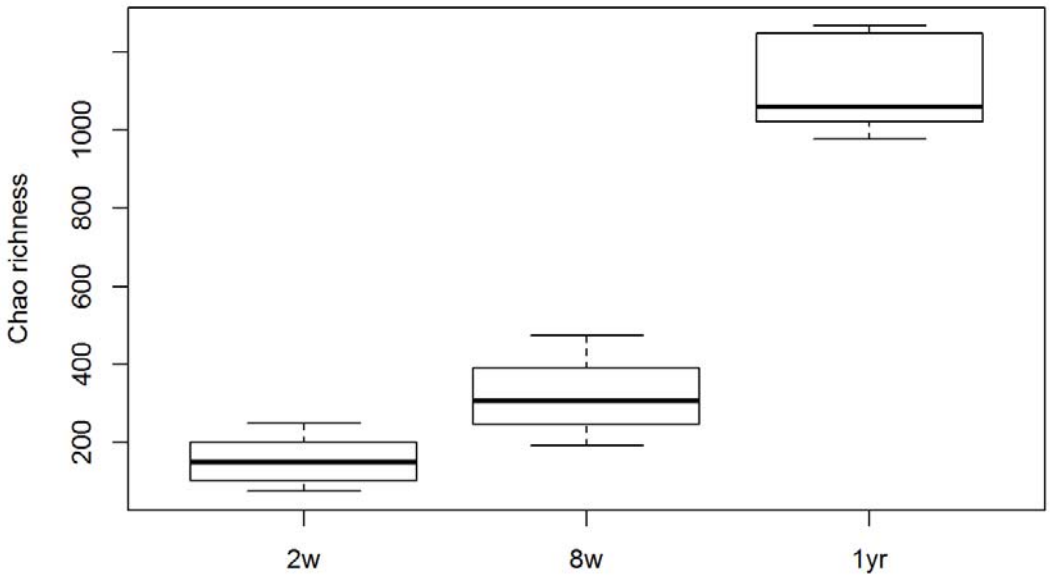
We can test pairwise within the age groups with Wilcoxon Rank Sum Tests. This test has a slightly different syntax than our other tests

```
pairwise.wilcox.test(meta$chao, meta$AgeGroup, p.adjust.method="fdr")
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test
##
## data:  meta$chao and meta$AgeGroup
##
##    1yr     2w
## 2w 0.00023 -
## 8w 0.00023 0.00186
##
## P value adjustment method: fdr
```

Like diversity, we see that richness also increases with age.

```
#Create 1x1 plot environment
par(mfrow = c(1, 1))
#Plot
boxplot(chao ~ AgeGroup.ord, data=meta, ylab="Chao richness")
```



Continuous variables

For continuous variables, we use general linear models, specifying the distribution that best fits our data.

Normally distributed metrics

Since ADG is a continuous variable, we run a general linear model. We will again use Shannon's diversity as our roughly normal metric. The default of `glm` and `lm` is the normal distribution so we don't have to specify anything.

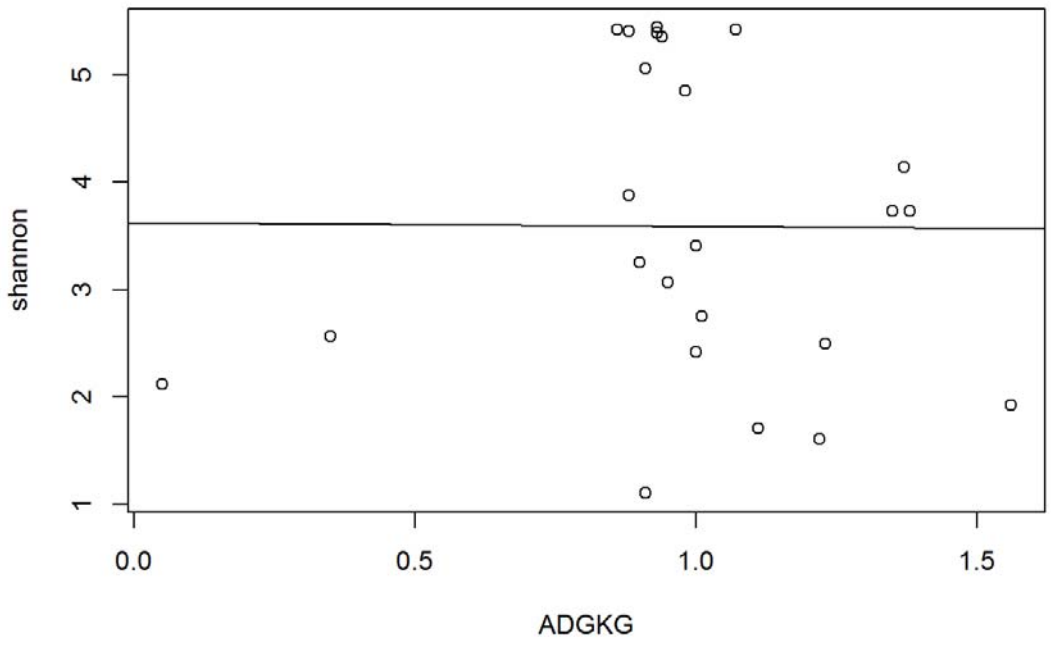
Does ADG impact the Shannon diversity of the fecal microbiota?

```
glm.shannon.ADG = glm(shannon ~ ADGKG, data=meta)
summary(glm.shannon.ADG)
```

```
##
## Call:
## glm(formula = shannon ~ ADGKG, data = meta)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.49110  -1.11216  -0.01749   1.53658   1.84728
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.62565    1.01390   3.576  0.00169 **
## ADGKG       -0.03407    0.97805  -0.035  0.97253
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 2.151815)
##
##     Null deviance: 47.343  on 23  degrees of freedom
## Residual deviance: 47.340  on 22  degrees of freedom
## AIC: 90.412
##
## Number of Fisher Scoring iterations: 2
```

The output let's us know that the intercept of our model is significantly different from 0 but our slope (*e.g.* our variable of interest) is not. This makes sense when we look at the data.

```
plot(shannon ~ ADGKG, data=meta)
#Add the glm best fit line
abline(glm.shannon.ADG)
```



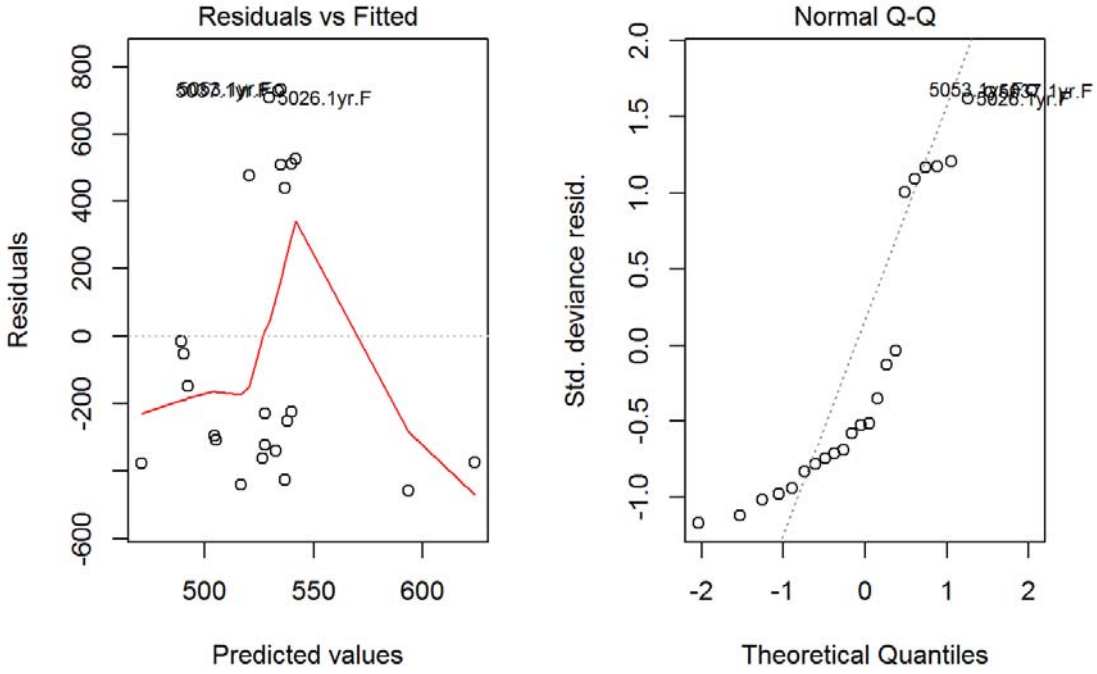
Non-normally distributed metrics

We will again use a general linear model for our non-normally distributed metric Chao. However, this time, we change the distribution from normal to something that fits the data better.

But which distribution should we choose? In statistics, there is no one "best" model. There are only good and better models. We will use the plot() function to compare two models and pick the better one.

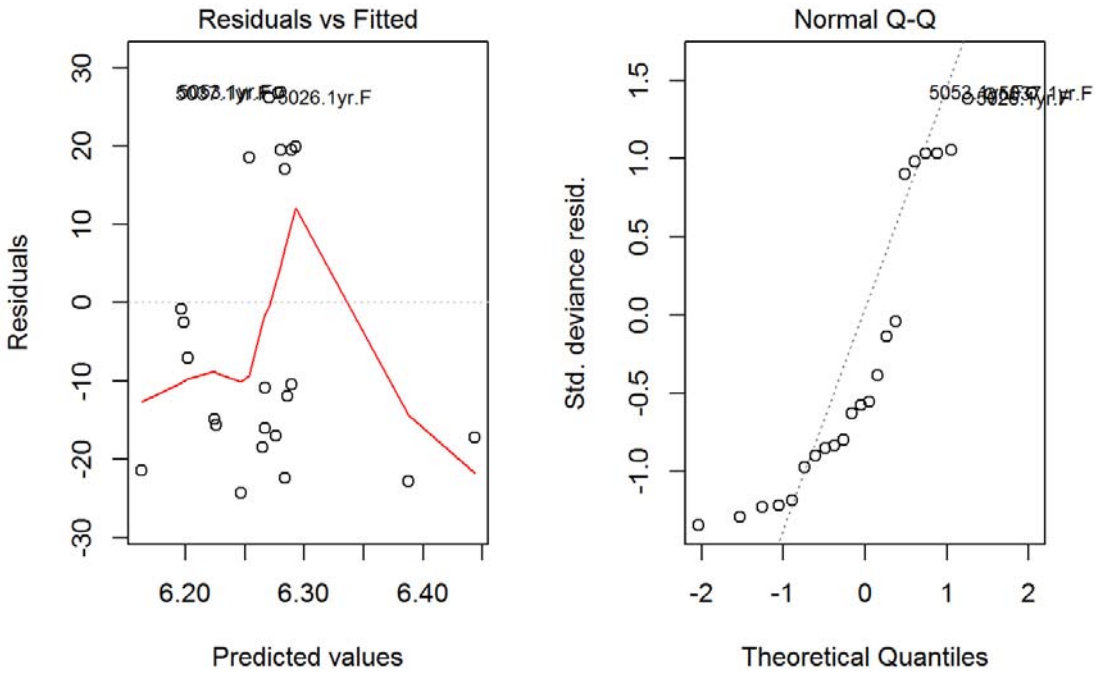
First, the Gaussian (normal) distribution, which we already know is a bad fit.

```
gaussian.chao.ADG = glm(chao ~ ADGKG, data=meta, family="gaussian")
par(mfrow = c(1,2))
plot(gaussian.chao.ADG, which=c(1,2))
```



Quasipoisson (log) distribution

```
qp.chao.ADG = glm(chao ~ ADGKG, data=meta, family="quasipoisson")
par(mfrow = c(1,2))
plot(qp.chao.ADG, which=c(1,2))
```



What we're looking for is no pattern in the Residuals vs. Fitted graph ("stars in the sky"), which shows that we picked a good distribution family to fit our data. We also want our residuals to be normally distributed, which is shown by most/all of the points falling on the line in the Normal Q-Q plot.

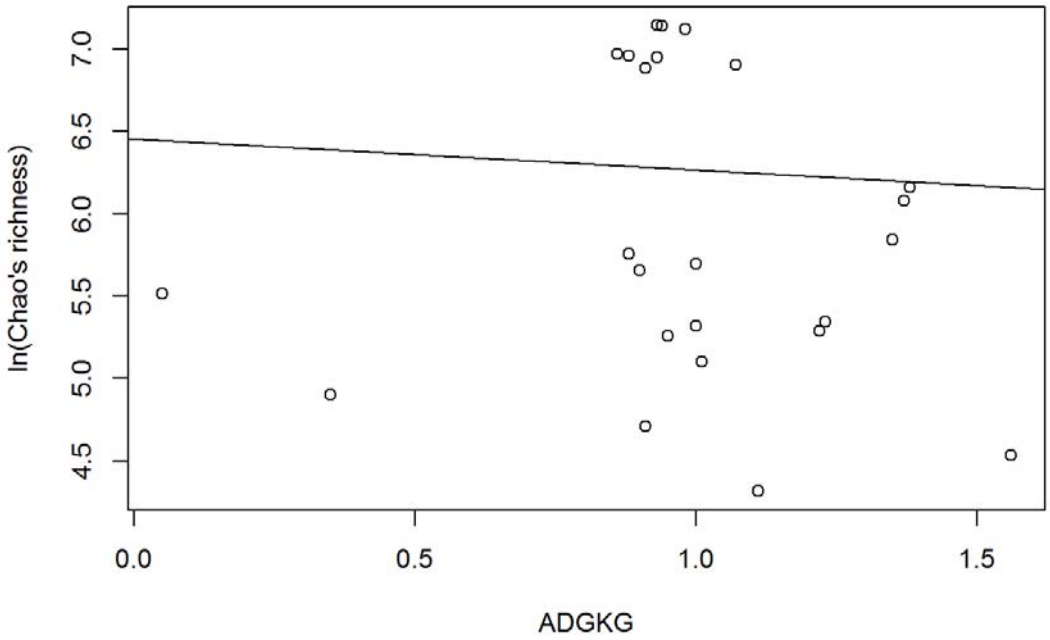
While it's still not perfect, the quasipoisson fits much better with residuals on the order of 30 whereas gaussian was on the order of 600. So, we will use quasipoisson and see that ADG does not to correlate to Chao richness.

```
summary(qp.chao.ADG)
```

```
##
## Call:
## glm(formula = chao ~ ADGKG, family = "quasipoisson", data = meta)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -24.36  -17.05  -10.66   18.81   26.91
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.4528     0.5561  11.605 7.54e-11 ***
## ADGKG        -0.1859     0.5438  -0.342    0.736
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 374.2485)
##
##     Null deviance: 8117.2  on 23  degrees of freedom
## Residual deviance: 8074.4  on 22  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

Plotting this we see that, indeed, there is not signficant correlation between Chao and ADG.

```
#Return the plot area to 1x1
par(mfrow = c(1, 1))
#Plot
plot(log(chao) ~ ADGKG, data=meta, ylab="ln(Chao's richness)")
abline(qp.chao.ADG)
```



Mixed models

Our two variables may not be fully independent and therefore, running them in two separate tests may not be correct. That is to say, age may impact ADG. In fact, I know this is the case because calves (2w, 8w) gain weight more quickly than heifers (1yr).

Think about your variables and what they mean “in the real world.” Logically combine them into as few ANOVA tests as possible. In the end, it’s better to test a meaningless interaction than not test a meaningful one.

We can test if the interaction of age and ADG impacts diversity with a model that includes both of our variables. The `*` symbol is a shortcut for models. A*B is equivalent to A + B + A:B

```
aov.shannon.all = aov(shannon ~ AgeGroup*ADGKG, data=meta)
summary(aov.shannon.all)
```

```
##                Df Sum Sq Mean Sq F value   Pr(>F)
## AgeGroup        2  42.98  21.489  95.472 2.61e-10 ***
## ADGKG           1   0.05   0.054   0.239    0.631
## AgeGroup:ADGKG  2   0.26   0.130   0.576    0.572
## Residuals      18   4.05   0.225
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can see that the interaction of age and ADG doesn’t significantly impact Shannon diversity, So we should remove that variable to simplify our model. If you had many interaction terms, you would step-wise remove the one with the highest P-value until you had the simplest model with only individual variables and significant interaction terms.

```
aov.shannon.all2 = aov(shannon ~ AgeGroup+ADGKG, data=meta)
summary(aov.shannon.all2)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## AgeGroup     2  42.98  21.489   99.70 3.96e-11 ***
## ADGKG        1   0.05   0.054    0.25    0.623
## Residuals   20   4.31   0.216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Overall, the ANOVA test tells us that only age impacts Shannon diversity but it does not tell us which age groups differ from one another. If all of our variables were categorical, we could run TukeyHSD like we did with age only.

```
TukeyHSD(aov.shannon.all)
```

```
## Warning in replications(paste("~", xx), data = mf): non-factors ignored:
## ADGKG
```

```
## Warning in replications(paste("~", xx), data = mf): non-factors ignored:
## AgeGroup, ADGKG
```

```
## Warning in TukeyHSD.aov(aov.shannon.all): 'which' specified some non-
## factors which will be dropped
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = shannon ~ AgeGroup * ADGKG, data = meta)
##
## $AgeGroup
##             diff       lwr       upr    p adj
## 2w-1yr -3.270063 -3.875469 -2.664657 0.00e+00
## 8w-1yr -1.830903 -2.436309 -1.225496 1.20e-06
## 8w-2w   1.439160  0.833754  2.044567 2.81e-05
```

However, you will see that we don't get any data from ADG since it is continuous. There is an error denoting this as "non-factors ignored: ADGKG"

So, we should have run our test as a glm since we have at least one continuous variable. First, we will still include the interaction variable to see that type of output.

```
glm.shannon.all = glm(shannon ~ AgeGroup*ADGKG, data=meta)
summary(glm.shannon.all)
```

```
##
## Call:
## glm(formula = shannon ~ AgeGroup * ADGKG, data = meta)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0301  -0.2468   0.0894   0.1572   0.7624
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)        5.7123     2.5928   2.203   0.0409 *
## AgeGroup2w        -3.3969     2.6197  -1.297   0.2111
## AgeGroup8w        -2.9610     2.7554  -1.075   0.2967
## ADGKG             -0.4481     2.7599  -0.162   0.8728
## AgeGroup2w:ADGKG   0.1228     2.7848   0.044   0.9653
## AgeGroup8w:ADGKG   1.0750     2.8763   0.374   0.7130
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.22508)
##
##     Null deviance: 47.3425  on 23  degrees of freedom
## Residual deviance:  4.0514  on 18  degrees of freedom
## AIC: 39.413
##
## Number of Fisher Scoring iterations: 2
```

Now this output is saying the same thing as ANOVA but in a more complicated way. The function automatically picks a reference group for categorical variables (in this case, 1yr) to compare all other groups to. Let's go through each line

- (Intercept) - This is whether or not the y-intercept is 0. A significant P-value indicates that the intercept is not 0, and we wouldn't expect it to be for any alpha-diversity metric since 0 means nothing is there
- AgeGroup2w - the difference between Shannon when Age = 2w vs. 1yr (the same as testing "shannon ~ AgeGroup" and only looking at the 2w-1yr pairwise comparison)
- AgeGroup8w - the same as 2w but now looking at only the 8w-1yr comparison
- ADGKG - the slope of Shannon to ADGKG (the same as testing "shannon ~ ADGKG")
- AgeGroup2w:ADGKG - the difference in slope of shannon ~ ADG between ages 2w and 1yr
- AgeGroup8w:ADGKG - the difference in slope of shannon ~ ADG between ages 8w and 1yr

As we saw in ANOVA, none of the interaction terms are significant so we remove them.

```
glm.shannon.all2 = glm(shannon ~ AgeGroup+ADGKG, data=meta)
summary(glm.shannon.all2)
```

```
##
## Call:
## glm(formula = shannon ~ AgeGroup + ADGKG, data = meta)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.95299  -0.25858   0.07643   0.30409   0.74487
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.4459     0.3487  15.619 1.14e-12 ***
## AgeGroup2w   -3.2760     0.2324 -14.094 7.55e-12 ***
## AgeGroup8w   -1.7989     0.2408  -7.471 3.30e-07 ***
## ADGKG        -0.1639     0.3281  -0.500    0.623
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.2155447)
##
##     Null deviance: 47.3425  on 23  degrees of freedom
## Residual deviance:  4.3109  on 20  degrees of freedom
## AIC: 36.903
##
## Number of Fisher Scoring iterations: 2
```

Note: The full glm model with the interaction term included did not show age as significant. When we remove the interaction term, age is significant. This is why you should remove non-significant interactions terms as they can the mask main effects of individual variables.

We can run a similar test with non-normal data like Chao.

```
qp.chao.all = glm(chao ~ AgeGroup*ADGKG, data=meta, family="quasipoisson")
summary(qp.chao.all)
```

```
##
## Call:
## glm(formula = chao ~ AgeGroup * ADGKG, family = "quasipoisson",
##     data = meta)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -7.774  -3.430  -0.140   3.692   5.277
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       6.99825    0.71122   9.840 1.14e-08 ***
## AgeGroup2w       -1.61539    0.75272  -2.146   0.0458 *
## AgeGroup8w       -2.24498    0.86846  -2.585   0.0187 *
## ADGKG             0.01751    0.75699   0.023   0.9818
## AgeGroup2w:ADGKG -0.42295    0.80094  -0.528   0.6039
## AgeGroup8w:ADGKG  0.86269    0.86550   0.997   0.3321
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 18.86331)
##
##     Null deviance: 8117.2  on 23  degrees of freedom
## Residual deviance:  348.5  on 18  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 4
```

Remove the non-significant interaction.

```
qp.chao.all2 = glm(chao ~ AgeGroup+ADGKG, data=meta, family="quasipoisson")
summary(qp.chao.all2)
```

```
##
## Call:
## glm(formula = chao ~ AgeGroup + ADGKG, family = "quasipoisson",
##     data = meta)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -7.783  -3.452  -1.378   3.744   8.184
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.03944    0.23567  29.870  < 2e-16 ***
## AgeGroup2w  -1.98090    0.14862 -13.329 2.08e-11 ***
## AgeGroup8w  -1.24286    0.11926 -10.422 1.57e-09 ***
## ADGKG       -0.02643    0.24530  -0.108    0.915
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 23.74583)
##
##     Null deviance: 8117.20  on 23  degrees of freedom
## Residual deviance:  476.31  on 20  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 4
```

Repeated measure

Another thing to consider with this data is the fact that we sampled the same animals over time. So, we have a repeated measures design. There are a number of ways to do repeated measures in R. I personally like the `lme4` package used here.

We add the repeated measure component by adding a random effect for the individual animals with `(1|Animal)` in the `lmer` function.

```
rm.shannon.all = lmer(shannon ~ AgeGroup+ADGKG + (1|Animal), data=meta)
summary(rm.shannon.all)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: shannon ~ AgeGroup + ADGKG + (1 | Animal)
##    Data: meta
##
## REML criterion at convergence: 32.4
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -1.83117 -0.45932  0.09539  0.49972  1.53368
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Animal   (Intercept) 0.03793  0.1948
##  Residual             0.17819  0.4221
## Number of obs: 24, groups:  Animal, 8
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   5.3906     0.3520  15.313
## AgeGroup2w   -3.2739     0.2114 -15.486
## AgeGroup8w   -1.8104     0.2208  -8.201
## ADGKG        -0.1049     0.3321  -0.316
##
## Correlation of Fixed Effects:
##            (Intr) AgGrp2 AgGrp8
## AgeGroup2w -0.350
## AgeGroup8w -0.027  0.461
## ADGKG      -0.884  0.057 -0.293
```

We see that very little of the variance in the data is explained by the animal random effects (0.03793). So we actually don't need to include repeated measures in our final model, but it was necessary to check!

From all of this, we can conclude that the fecal microbiota increases in diversity and richness as dairy cows age. Animal growth as measured by ADG does not correlate with fecal community diversity or richness.

Beta-diversity

Beta-diversity is between sample diversity. It is how different every sample is from every other sample. Thus, each sample has more than one value. Some metrics take abundance into account (*i.e.* diversity: Bray-Curtis, weighted UniFrac) and some only calculate based on presence-absence (*i.e.* richness: Jaccard, unweighted UniFrac).

Beta-diversity appears like the following (completely made-up numbers)

.	sample1	sample2	sample3	…
sample1	0	0.345	0.194	…
sample2	0.345	0	0.987	…
sample3	0.194	0.987	0	…
…	…	…	…	…

Visualization

The best way to visualize beta-diversity, or how different samples are from each other, is by non-metric multidimensional scaling (nMDS). This is similar to principle coordinate analysis or PCA/PCoA if you've heard of that, only nMDS is more statistically robust with multiple iterations in the form of the `trymax` part of the command.

Each symbol on an nMDS plot represents the total microbial community of that sample. Symbols closer together have more similar microbiotas while those farther apart have less similar.

OTU-based metrics

There are two main type of beta-diversity measures. These OTU-based metrics treat every OTU as a separate entity without taking taxonomy into account. The distance between *Prevotella* OTU1 and *Prevotella* OTU2 is equivalent to the distance between *Prevotella* OTU1 and *Bacteroides* OTU1.

Dot plots

First, we calculate the nMDS values for a 2-axis `k=2` graph using the OTU-based Bray-Curtis metric that takes into account both the presence/absence and abundance of OTUs in your samples (*i.e.* diversity). This uses the `metaMDS` function from the package `vegan`.

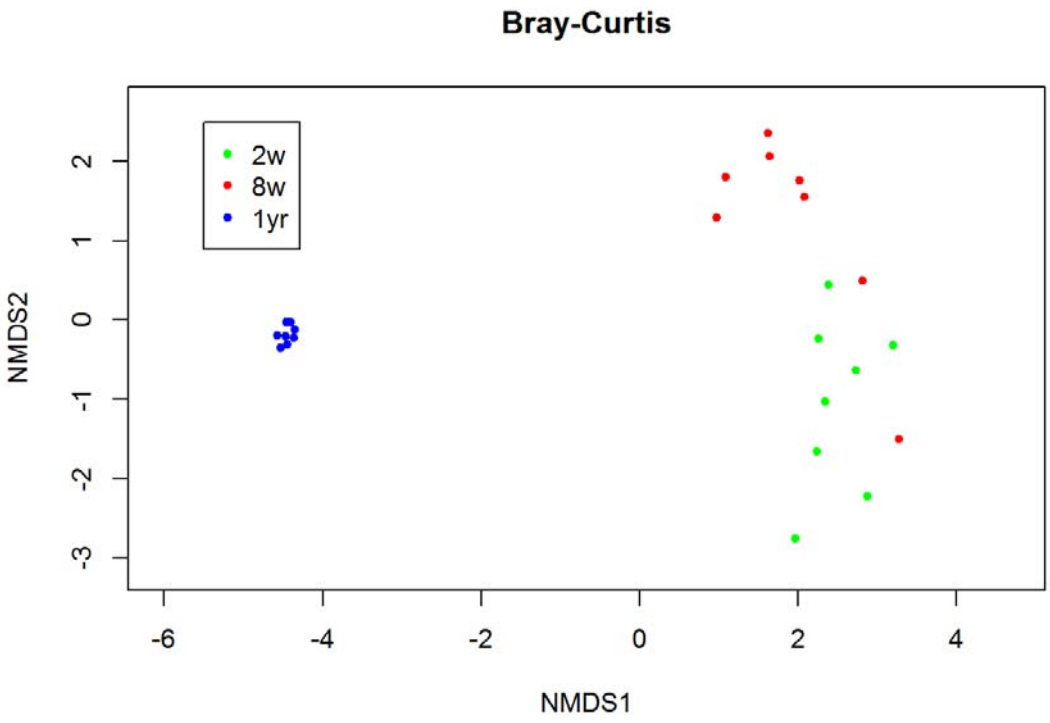
```
BC.nmds = metaMDS(OTU.clean, distance="bray", k=2, trymax=1000)
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.06208161
## Run 1 stress 0.06210668
## ... Procrustes: rmse 0.001636313  max resid 0.005662513
## ... Similar to previous best
## Run 2 stress 0.06208261
## ... Procrustes: rmse 0.0008174643  max resid 0.00186259
## ... Similar to previous best
## Run 3 stress 0.06208133
## ... New best solution
## ... Procrustes: rmse 0.000495613  max resid 0.001143981
## ... Similar to previous best
## Run 4 stress 0.06208228
## ... Procrustes: rmse 0.0002768028  max resid 0.0006083455
## ... Similar to previous best
## Run 5 stress 0.06208254
## ... Procrustes: rmse 0.0003377152  max resid 0.0007457908
## ... Similar to previous best
## Run 6 stress 0.06208233
## ... Procrustes: rmse 0.000285801  max resid 0.000626649
## ... Similar to previous best
## Run 7 stress 0.06210685
## ... Procrustes: rmse 0.001453303  max resid 0.005539077
## ... Similar to previous best
## Run 8 stress 0.062104
## ... Procrustes: rmse 0.001430176  max resid 0.005147467
## ... Similar to previous best
## Run 9 stress 0.06208351
## ... Procrustes: rmse 0.0005018534  max resid 0.00111944
## ... Similar to previous best
## Run 10 stress 0.06208269
## ... Procrustes: rmse 0.0003614257  max resid 0.0008024269
## ... Similar to previous best
## Run 11 stress 0.06208154
## ... Procrustes: rmse 0.0004861021  max resid 0.001120926
## ... Similar to previous best
## Run 12 stress 0.06212707
## ... Procrustes: rmse 0.001859292  max resid 0.005339963
## ... Similar to previous best
## Run 13 stress 0.3702005
## Run 14 stress 0.06210406
## ... Procrustes: rmse 0.001425256  max resid 0.00512563
## ... Similar to previous best
## Run 15 stress 0.06208142
## ... Procrustes: rmse 3.189023e-05  max resid 6.612762e-05
## ... Similar to previous best
## Run 16 stress 0.06210429
## ... Procrustes: rmse 0.001578454  max resid 0.005195898
## ... Similar to previous best
## Run 17 stress 0.06210796
## ... Procrustes: rmse 0.00155285  max resid 0.005626229
## ... Similar to previous best
## Run 18 stress 0.06208191
## ... Procrustes: rmse 0.0001981339  max resid 0.0004391198
## ... Similar to previous best
## Run 19 stress 0.06208168
## ... Procrustes: rmse 0.0001331311  max resid 0.000291077
## ... Similar to previous best
## Run 20 stress 0.06210592
## ... Procrustes: rmse 0.001396183  max resid 0.005412384
## ... Similar to previous best
## *** Solution reached
```

We see that we reached a convergent solution around 20 iterations and our stress is very low (0.06), meaning that 2-axis are sufficient to view the data.

Then plot the nMDS with different colors for your different groups of interest. We will use colors for our three ages

```
par(mfrow = c(1, 1))
#Create a blank plot for the nmds
plot(BC.nmds, type="n", main="Bray-Curtis")
#Add the points colored by age
points(BC.nmds, display="sites", pch=20, col=c("blue", "green", "red")[meta$AgeGroup])
#Add a legend
legend(-5.5, 2.5, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
```



This will create a plot in the lower right quadrant. If you want to get fancy, type “?plot” in the console to see other ways to modify the plot function.

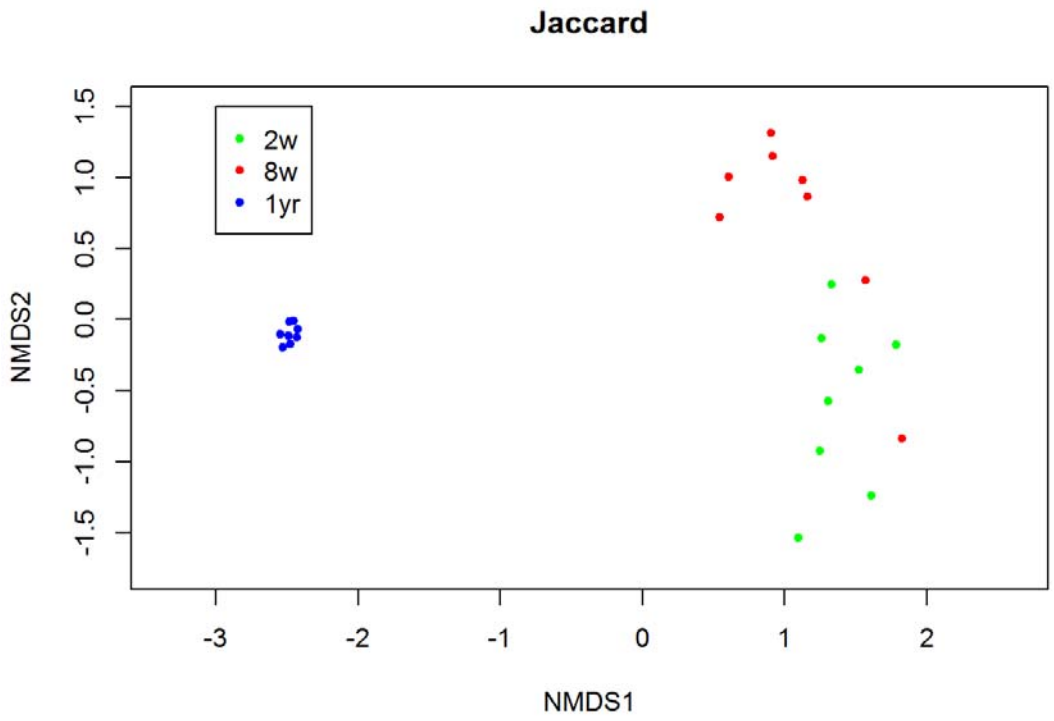
A similar thing can be done for the Jaccard metric, which only takes into account presence/absence (*i.e.* richness).

```
J.nmds = metaMDS(OTU.clean, distance="jaccard", k=2, trymax=1000)
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.0620818
## Run 1 stress 0.06208178
## ... New best solution
## ... Procrustes: rmse 0.0007016851  max resid 0.001623036
## ... Similar to previous best
## Run 2 stress 0.06210633
## ... Procrustes: rmse 0.001409348  max resid 0.005467011
## ... Similar to previous best
## Run 3 stress 0.06210745
## ... Procrustes: rmse 0.001470069  max resid 0.00557513
## ... Similar to previous best
## Run 4 stress 0.06208144
## ... New best solution
## ... Procrustes: rmse 0.0001309513  max resid 0.0002717662
## ... Similar to previous best
## Run 5 stress 0.06208156
## ... Procrustes: rmse 5.349512e-05  max resid 0.0001195792
## ... Similar to previous best
## Run 6 stress 0.06208137
## ... New best solution
## ... Procrustes: rmse 2.027381e-05  max resid 4.710602e-05
## ... Similar to previous best
## Run 7 stress 0.06208345
## ... Procrustes: rmse 0.0004560942  max resid 0.001010311
## ... Similar to previous best
## Run 8 stress 0.06210681
## ... Procrustes: rmse 0.001448074  max resid 0.005531499
## ... Similar to previous best
## Run 9 stress 0.06208334
## ... Procrustes: rmse 0.0004470347  max resid 0.000984174
## ... Similar to previous best
## Run 10 stress 0.06208155
## ... Procrustes: rmse 7.705878e-05  max resid 0.0001651192
## ... Similar to previous best
## Run 11 stress 0.06208217
## ... Procrustes: rmse 0.0002412108  max resid 0.0005340427
## ... Similar to previous best
## Run 12 stress 0.06210429
## ... Procrustes: rmse 0.001420012  max resid 0.005133791
## ... Similar to previous best
## Run 13 stress 0.06208263
## ... Procrustes: rmse 0.0002884997  max resid 0.0006395557
## ... Similar to previous best
## Run 14 stress 0.06208166
## ... Procrustes: rmse 0.0001135875  max resid 0.0002424163
## ... Similar to previous best
## Run 15 stress 0.06210651
## ... Procrustes: rmse 0.001438738  max resid 0.005503184
## ... Similar to previous best
## Run 16 stress 0.06208137
## ... New best solution
## ... Procrustes: rmse 6.516686e-05  max resid 0.0001605969
## ... Similar to previous best
## Run 17 stress 0.06208244
## ... Procrustes: rmse 0.0002976643  max resid 0.0007159927
## ... Similar to previous best
## Run 18 stress 0.06208222
## ... Procrustes: rmse 0.0002618419  max resid 0.0006358936
## ... Similar to previous best
## Run 19 stress 0.06208197
## ... Procrustes: rmse 0.000208525  max resid 0.0005678922
## ... Similar to previous best
## Run 20 stress 0.0620832
```

```
## ... Procrustes: rmse 0.0004189108  max resid 0.0009707012
## ... Similar to previous best
## *** Solution reached
```

```
plot(J.nmds, type="n", main="Jaccard")
points(J.nmds, display="sites", pch=20, col=c("blue", "green", "red")[meta$AgeGroup])
legend(-3, 1.5, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
```



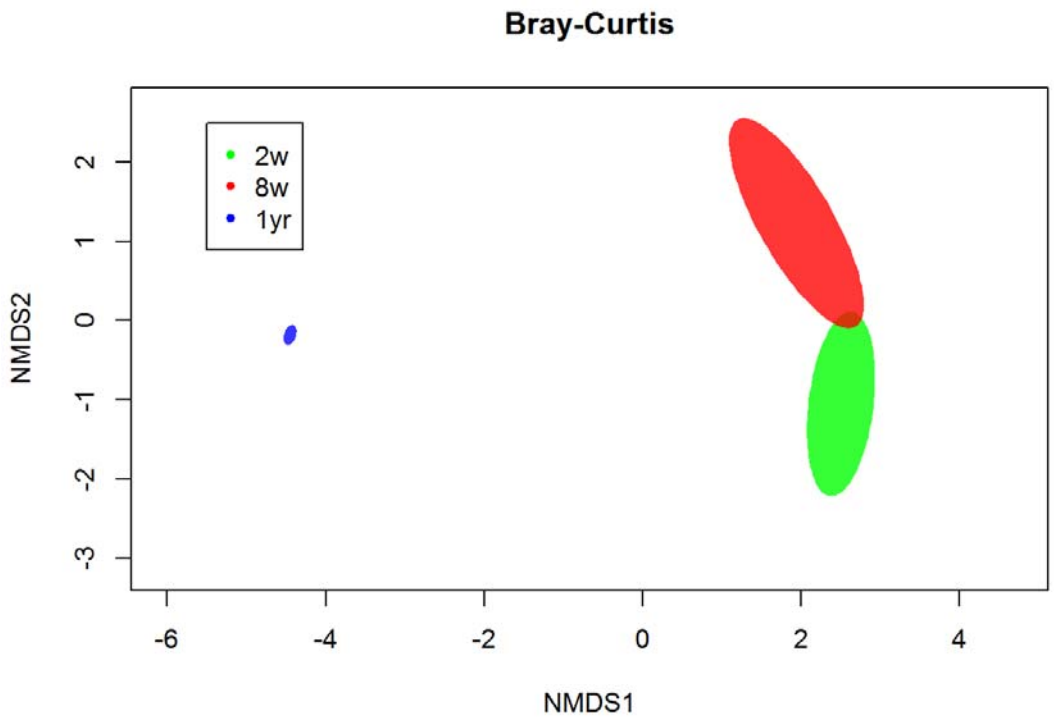
You see that the values are very different for Jaccard but the pattern of points is very similar to Bray-Curtis. This is because Jaccard is a transformation of Bray-Curtis with J = 2BC/(1+BC)

Ellipses

You can also plot standard error (se) ellipses for your nmds data instead of showing all of the individual points. Here, we will plot 99% confidence se ellipses for the Bray-Curtis metric using `ordiellipse` from `vegan`.

Code courtesy of Madison Cox.

```
plot(BC.nmds, type="n", main="Bray-Curtis")
legend(-5.5, 2.5, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)

#Add an ellipse for 2w
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="green",
draw="polygon", alpha=200, show.groups = c("2w"), border=FALSE)

#Add an ellipse for 8w
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="red", dr
aw="polygon", alpha=200, show.groups = c("8w"), border=FALSE)

#Add an ellipse for 1yr
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="blue", d
raw="polygon", alpha=200, show.groups = c("1yr"), border=FALSE)
```



We clearly see in both the dot and ellipse plots that age significantly impacts the overall structure (Bray-Curtis) and composition (Jaccard) of the fecal bacterial microbiota.

3D plots

If your stress is high (like over 0.3) for your `metaMDS` calculation, you probably need to increase to 3 axes `k=3`. Graphing a 3D plot is much more complicated, and there are a number of packages that could be used. Here, we will use one option from the `plotly` package to visualize a 3D Bray-Curtis plot.

```
#Calculate the Bray-Curtis nMDS for 3-axis
BC.nmds.3D = metaMDS(OTU.clean, distance="bray", k=3, trymax=1000)
```

```
## Square root transformation
## Wisconsin double standardization
## Run 0 stress 0.04686346
## Run 1 stress 0.04741659
## Run 2 stress 0.04673425
## ... New best solution
## ... Procrustes: rmse 0.01073904  max resid 0.0344814
## Run 3 stress 0.05061835
## Run 4 stress 0.04740131
## Run 5 stress 0.04984642
## Run 6 stress 0.04747801
## Run 7 stress 0.05226505
## Run 8 stress 0.05295437
## Run 9 stress 0.04741387
## Run 10 stress 0.0457586
## ... New best solution
## ... Procrustes: rmse 0.03868237  max resid 0.1296728
## Run 11 stress 0.05094992
## Run 12 stress 0.04719303
## Run 13 stress 0.05012352
## Run 14 stress 0.04750204
## Run 15 stress 0.0479423
## Run 16 stress 0.04579561
## ... Procrustes: rmse 0.004692476  max resid 0.01495666
## Run 17 stress 0.05069634
## Run 18 stress 0.0485804
## Run 19 stress 0.05058189
## Run 20 stress 0.04859459
## Run 21 stress 0.04996713
## Run 22 stress 0.04740079
## Run 23 stress 0.04747632
## Run 24 stress 0.04675455
## Run 25 stress 0.04747574
## Run 26 stress 0.0486171
## Run 27 stress 0.04575823
## ... New best solution
## ... Procrustes: rmse 0.0005374711  max resid 0.0008831403
## ... Similar to previous best
## *** Solution reached
```

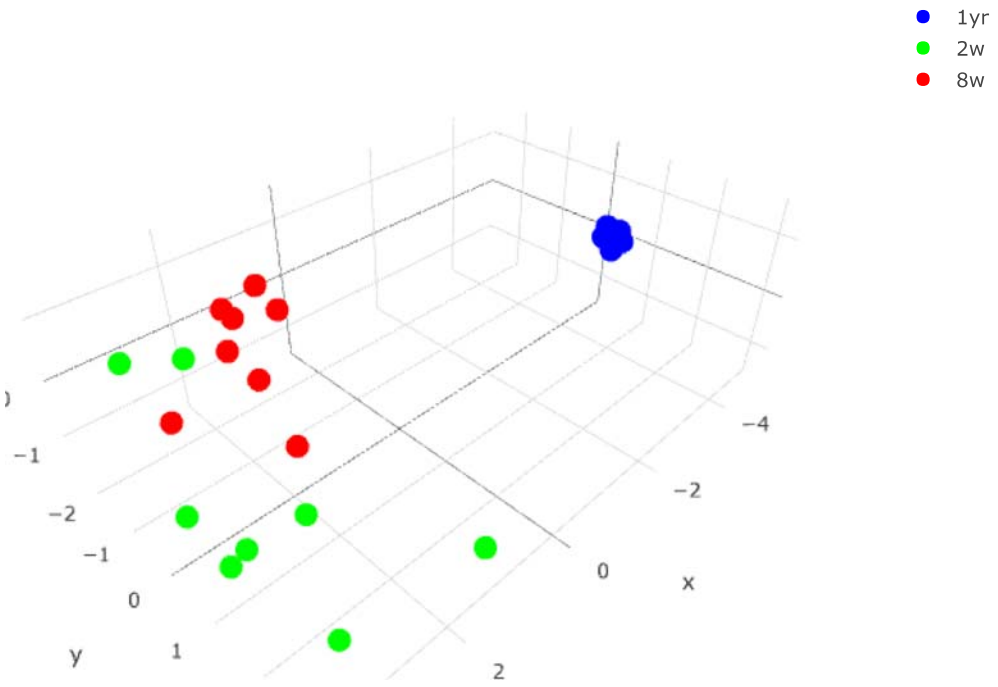
Extract x-y-z values for this nmds

```
BCxyz = scores(BC.nmds.3D, display="sites")
#This is a table that looks like
BCxyz
```

```
##                 NMDS1       NMDS2        NMDS3
## 5017.1yr.F -4.7973931  0.33029806 -0.211481225
## 5017.2w.F   3.1867260  0.06208276  1.484970505
## 5017.8w.F   1.0614871 -2.13025264 -1.218243774
## 5020.1yr.F -4.7579235  0.24440345 -0.002888360
## 5020.2w.F   3.4979230 -1.00981047  1.015200903
## 5020.8w.F   1.5897780 -1.93435391  0.464128291
## 5026.1yr.F -4.7720517  0.20611823  0.214815994
## 5026.2w.F   3.3976411  1.10010056 -0.616957559
## 5026.8w.F   3.1483050  2.07715934  1.478767471
## 5031.1yr.F -4.8021402  0.44250394  0.202447638
## 5031.2w.F   3.3537430  0.48376070 -1.490408346
## 5031.8w.F   0.8577869 -1.64300786  0.250766536
## 5037.1yr.F -4.8522745  0.48898068 -0.004218580
## 5037.2w.F   3.6593056  0.26886383 -0.507062657
## 5037.8w.F   3.1326413 -0.82210579 -0.024946820
## 5041.1yr.F -4.7724198  0.28335210  0.060469429
## 5041.2w.F   3.1661815  2.43615798 -1.218459457
## 5041.8w.F   1.0947996 -2.58325770 -0.236659085
## 5045.1yr.F -4.7522029  0.16444286  0.004405471
## 5045.2w.F   1.5110480  3.11956405 -0.469494555
## 5045.8w.F   1.4900615 -2.17087166 -0.450930039
## 5053.1yr.F -4.8259682  0.39929033 -0.016428020
## 5053.2w.F   3.2932453  2.30299477  0.813801957
## 5053.8w.F   0.8917011 -2.11641360  0.478404284
```

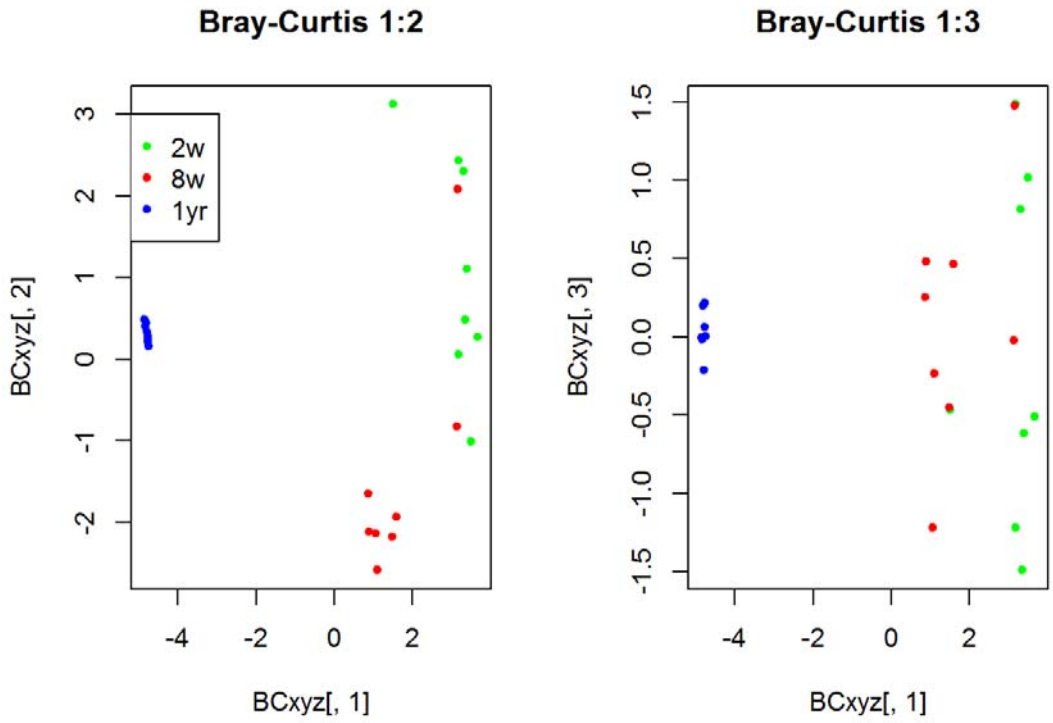
Plot the xyz coordinates and color by age

```
plot_ly(x=BCxyz[,1], y=BCxyz[,2], z=BCxyz[,3], type="scatter3d", mode="markers", color=meta$AgeGroup,
colors=c("blue", "green", "red"))
```



Note: Since 3D plots are difficult to interpret in printed journal articles, many authors choose to create two separate 2D plots to show the 3D data like so.

```
par(mfrow=c(1,2))
#Axis 1 and 2 (x and y)
plot(BCxyz[,1], BCxyz[,2], main="Bray-Curtis 1:2", pch=20, col=c("blue", "green", "red")[meta$AgeGroup])
legend(-5.4, 3, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Axis 1 and 3 (x and z)
plot(BCxyz[,1], BCxyz[,3], main="Bray-Curtis 1:3", pch=20, col=c("blue", "green", "red")[meta$AgeGroup])
```



Phylogentic-based metrics

The most common of this type of beta-diversity metrics is UniFrac. The strength of UniFrac over Bray-Curtis or Jaccard is that it takes into account phylogenetic relationships of the species present in the microbiota. Thus, samples with different OTUs from the same genus will be more similar by UniFrac that those with OTUs from different genera. The weakness is that UniFrac is more sensitive to low abundance OTUs and those that a very phylogenetically distant.

Your choice will depend on how much you personally feel phylogenetic relationships vs. sensitively matter in your data.

Just as above, UniFrac can be plotted as an nMDS. You just need to use a different R package, and thus, slightly different commands.

Create physeq object

To start, you must make a `phyloseq` object which includes the OTU.clean, meta, and tax.clean data. We tell R which tables are each type

```
OTU.UF = otu_table(as.matrix(OTU.clean), taxa_are_rows=FALSE)
tax.UF = tax_table(as.matrix(tax.clean))
meta.UF = sample_data(meta)
```

We then merge these into an object of class phyloseq.

```
physeq = phyloseq(OTU.UF, tax.UF, meta.UF)
```

To add the phylogenetic component to UniFrac, we calculate a rooted phylogenetic tree of our OTUs. This takes a long time so we have provided the tree for you.

However, if we were to calculate a tree, first, we import a distance matrix created from representative sequences of our OTUs. We would use `phangorn` to read the file as it was created in mothur as seen under "Trees of OTUs" here (https://rpubs.com/dillmcfarlan/mothurSOP).

DO NOT RUN THIS

```
dist.mat = import_mothur_dist("clean_repFasta.phylip.dist")
```

We would then calculate a rooted neighbor-joining tree from the distance matrix using the `ape` package.

DO NOT RUN THIS

```
NJ.tree = bionj(dist.mat)
```

Instead, we have pre-calculated this tree and you can load is with

```
load("NJ.tree.Rdata")
```

Then, add this tree to your physeq object. This object will be what is used in UniFrac calculations.

```
physeq.tree = merge_phyloseq(physeq, NJ.tree)
```

We can look at this object and see its components.

```
physeq.tree
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5002 taxa and 24 samples ]
## sample_data() Sample Data:       [ 24 samples by 9 sample variables ]
## tax_table()   Taxonomy Table:    [ 5002 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 5002 tips and 5000 internal nodes ]
```

Dot plots

Calculate weighted UniFrac (*i.e.* diversity) distances and ordinate into an nMDS. We specify weighted with `weighted=TRUE`.

```
wUF.ordu = ordinate(physeq.tree, method="NMDS", distance="unifrac", weighted=TRUE)
```

```
## Warning in UniFrac(physeq, ...): Randomly assigning root as -- Otu00062 --
## in the phylogenetic tree in the data you provided.
```

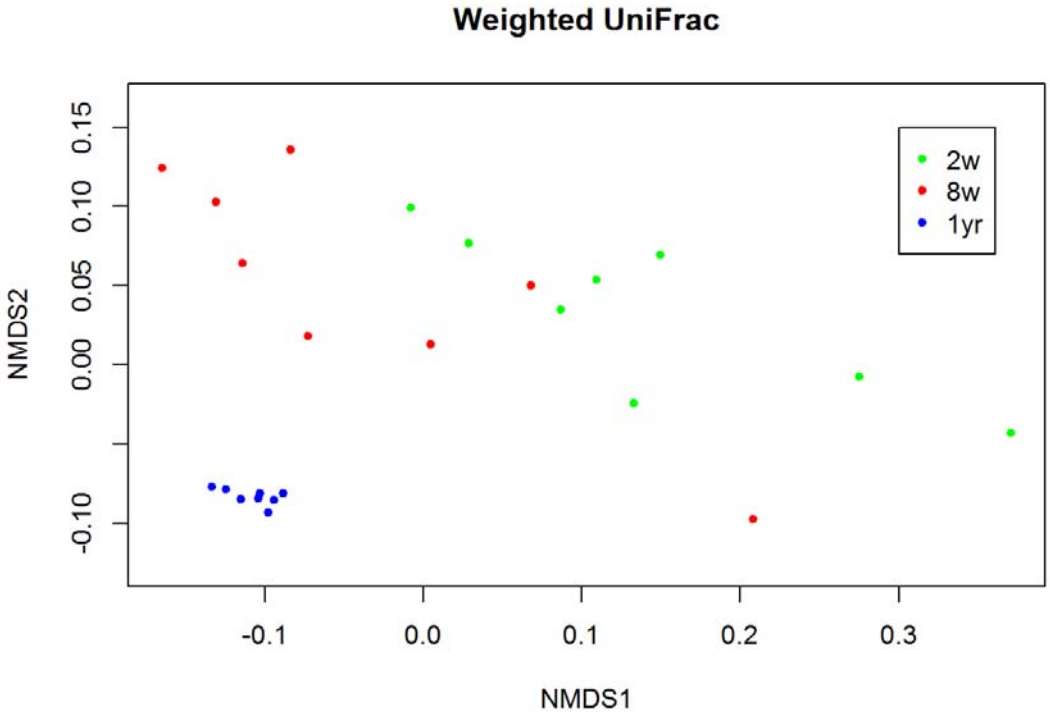
```
## Run 0 stress 0.0864543
## Run 1 stress 0.08645377
## ... New best solution
## ... Procrustes: rmse 0.0001213931  max resid 0.0003141587
## ... Similar to previous best
## Run 2 stress 0.1335727
## Run 3 stress 0.1463023
## Run 4 stress 0.08645329
## ... New best solution
## ... Procrustes: rmse 0.0007206919  max resid 0.001920389
## ... Similar to previous best
## Run 5 stress 0.1270238
## Run 6 stress 0.1157455
## Run 7 stress 0.1143571
## Run 8 stress 0.1317677
## Run 9 stress 0.08645345
## ... Procrustes: rmse 5.804039e-05  max resid 0.0001620988
## ... Similar to previous best
## Run 10 stress 0.08808605
## Run 11 stress 0.08645348
## ... Procrustes: rmse 0.000642139  max resid 0.001706552
## ... Similar to previous best
## Run 12 stress 0.1157451
## Run 13 stress 0.0864534
## ... Procrustes: rmse 4.051435e-05  max resid 0.0001125382
## ... Similar to previous best
## Run 14 stress 0.1143564
## Run 15 stress 0.08659435
## ... Procrustes: rmse 0.004251655  max resid 0.01804703
## Run 16 stress 0.1295296
## Run 17 stress 0.0864538
## ... Procrustes: rmse 0.000161137  max resid 0.0004585026
## ... Similar to previous best
## Run 18 stress 0.1347981
## Run 19 stress 0.08645297
## ... New best solution
## ... Procrustes: rmse 0.0003657154  max resid 0.0008934259
## ... Similar to previous best
## Run 20 stress 0.08808625
## *** Solution reached
```

You can plot UniFrac nMDS using the basic `plot` function as we've done before.

```
par(mfrow=c(1,1))
plot(wUF.ordu, type="n", main="Weighted UniFrac")
```

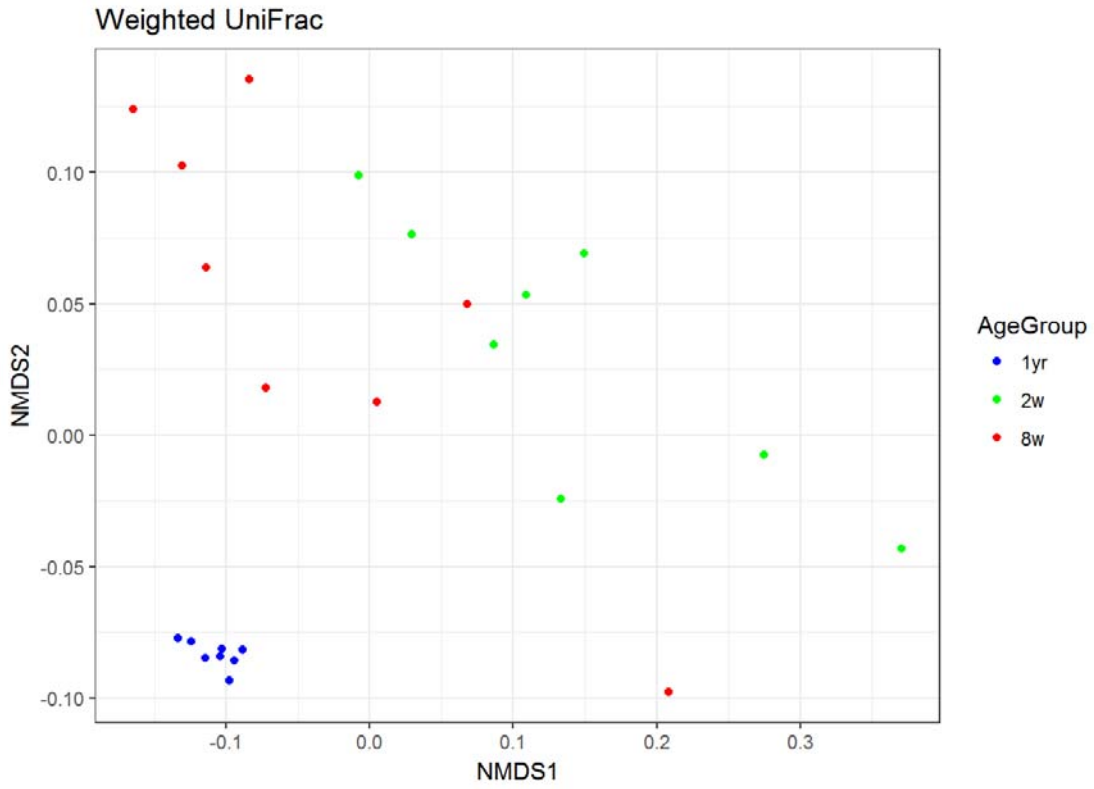
```
## Warning in ordiplot(x, choices = choices, type = type, display = display, :
## Species scores not available
```

```
points(wUF.ordu, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(0.3,0.15, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
```



But let's also look at the `ggplot2` package. This package is incredibly powerful and can be customized in many ways. This document (https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf) has many helpful tips.

```
plot_ordination(physeq.tree, wUF.ordu, type="sites", color="AgeGroup") +
  scale_colour_manual(values=c("2w"="green", "8w"="red", "1yr"="blue")) +
  theme_bw() +
  ggtitle("Weighted UniFrac")
```



Unweighted UniFrac (*i.e.* richness) can be visualized in the same way. We specify unweighted with `weighted=FALSE`.

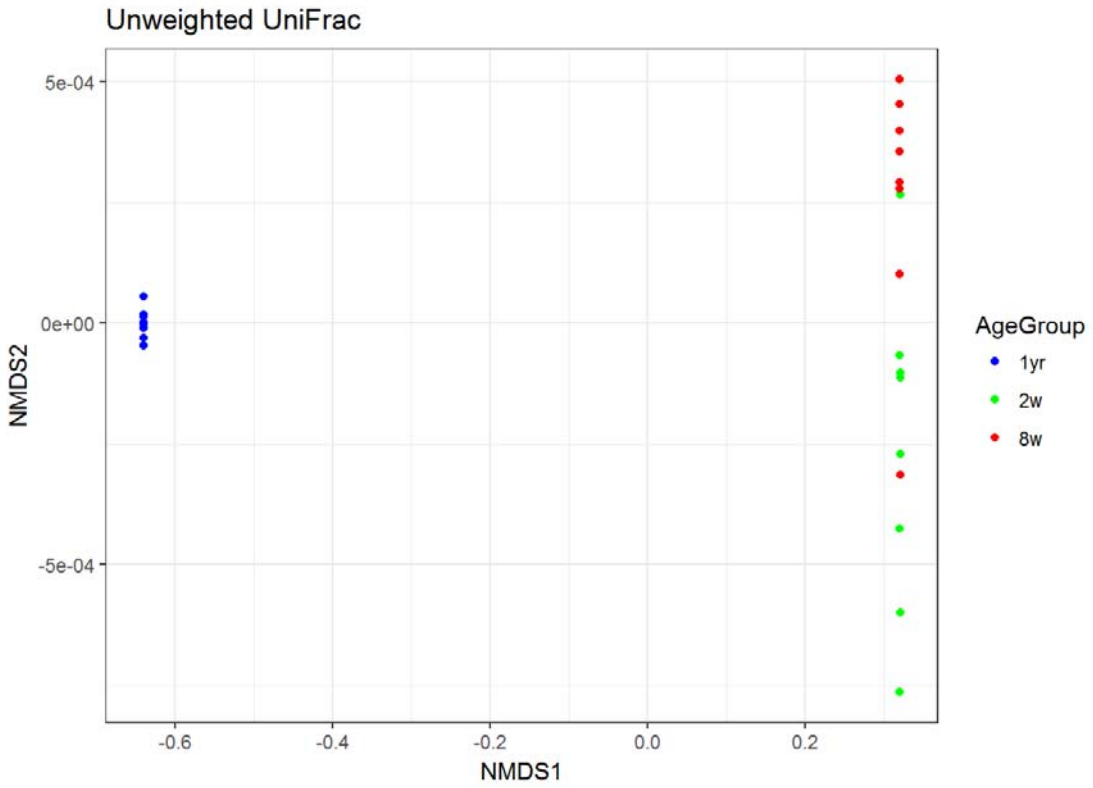
```
uwUF.ordu = ordinate(physeq.tree, method="NMDS", distance="unifrac", weighted=FALSE)
```

```
## Warning in UniFrac(physeq, ...): Randomly assigning root as -- Otu00541 --
## in the phylogenetic tree in the data you provided.
```

```
## Run 0 stress 9.695153e-05
## Run 1 stress 9.657832e-05
## ... New best solution
## ... Procrustes: rmse 7.750783e-05  max resid 0.0002776914
## ... Similar to previous best
## Run 2 stress 9.871795e-05
## ... Procrustes: rmse 8.086551e-05  max resid 0.0002819207
## ... Similar to previous best
## Run 3 stress 9.488623e-05
## ... New best solution
## ... Procrustes: rmse 7.261501e-05  max resid 0.0002642816
## ... Similar to previous best
## Run 4 stress 9.862006e-05
## ... Procrustes: rmse 1.701217e-05  max resid 5.025527e-05
## ... Similar to previous best
## Run 5 stress 9.806631e-05
## ... Procrustes: rmse 0.0001070473  max resid 0.0002353732
## ... Similar to previous best
## Run 6 stress 9.757454e-05
## ... Procrustes: rmse 3.985665e-05  max resid 0.0001388531
## ... Similar to previous best
## Run 7 stress 9.826177e-05
## ... Procrustes: rmse 9.722135e-05  max resid 0.0002191936
## ... Similar to previous best
## Run 8 stress 9.695708e-05
## ... Procrustes: rmse 7.448687e-05  max resid 0.0002751687
## ... Similar to previous best
## Run 9 stress 9.907648e-05
## ... Procrustes: rmse 9.310993e-05  max resid 0.0002388289
## ... Similar to previous best
## Run 10 stress 9.984534e-05
## ... Procrustes: rmse 3.384419e-05  max resid 0.0001260377
## ... Similar to previous best
## Run 11 stress 9.684607e-05
## ... Procrustes: rmse 0.0001319037  max resid 0.0003356478
## ... Similar to previous best
## Run 12 stress 9.69891e-05
## ... Procrustes: rmse 8.404145e-06  max resid 2.447679e-05
## ... Similar to previous best
## Run 13 stress 0.0002969569
## ... Procrustes: rmse 0.0003866364  max resid 0.0006715474
## ... Similar to previous best
## Run 14 stress 9.723199e-05
## ... Procrustes: rmse 3.731826e-05  max resid 0.0001336343
## ... Similar to previous best
## Run 15 stress 9.99257e-05
## ... Procrustes: rmse 0.0001270356  max resid 0.0003614341
## ... Similar to previous best
## Run 16 stress 9.955355e-05
## ... Procrustes: rmse 6.056256e-05  max resid 0.0001673759
## ... Similar to previous best
## Run 17 stress 9.589429e-05
## ... Procrustes: rmse 1.686683e-05  max resid 4.596185e-05
## ... Similar to previous best
## Run 18 stress 9.633493e-05
## ... Procrustes: rmse 3.660483e-05  max resid 0.0001324208
## ... Similar to previous best
## Run 19 stress 9.921893e-05
## ... Procrustes: rmse 1.085938e-05  max resid 1.669484e-05
## ... Similar to previous best
## Run 20 stress 9.637055e-05
## ... Procrustes: rmse 6.450683e-05  max resid 0.0001970587
## ... Similar to previous best
## *** Solution reached
```

```
## Warning in metaMDS(ps.dist): Stress is (nearly) zero - you may have
## insufficient data
```

```
plot_ordination(physeq.tree, uwUF.ordu, type="sites", color="AgeGroup") +
  scale_colour_manual(values=c("2w"="green", "8w"="red", "1yr"="blue")) +
  theme_bw() +
  ggtitle("Unweighted UniFrac")
```



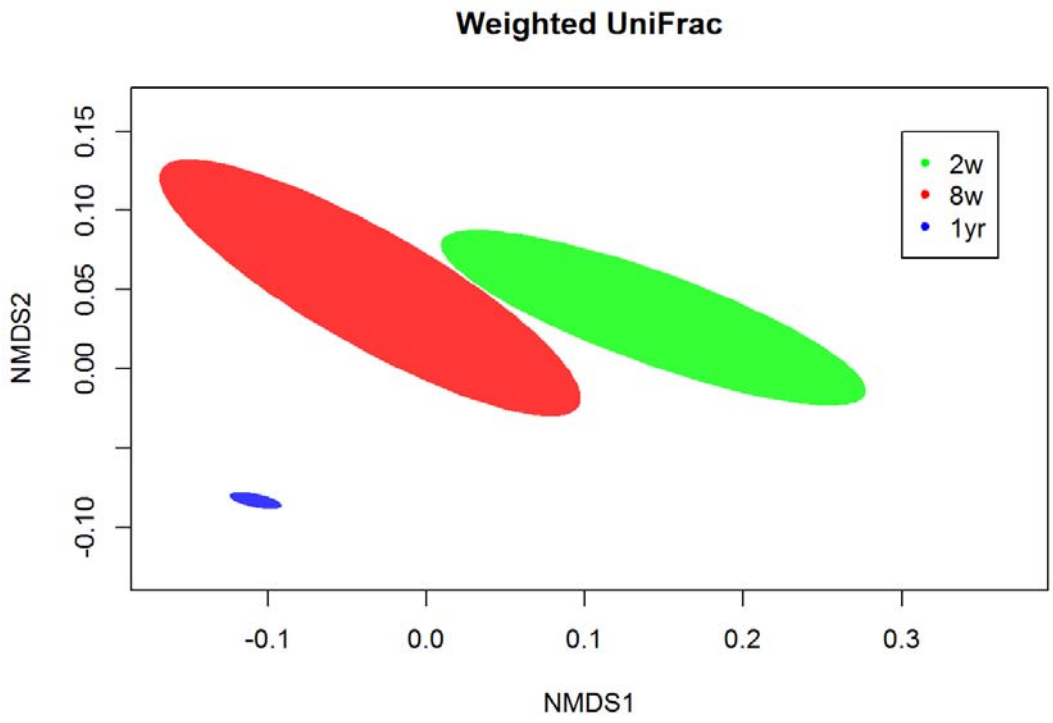
Ellipses

Ellipses can be plotted instead of points as well. With the basic plot function:

```
plot(wUF.ordu, type="n", main="Weighted UniFrac")
```

```
## Warning in ordiplot(x, choices = choices, type = type, display = display, :
## Species scores not available
```

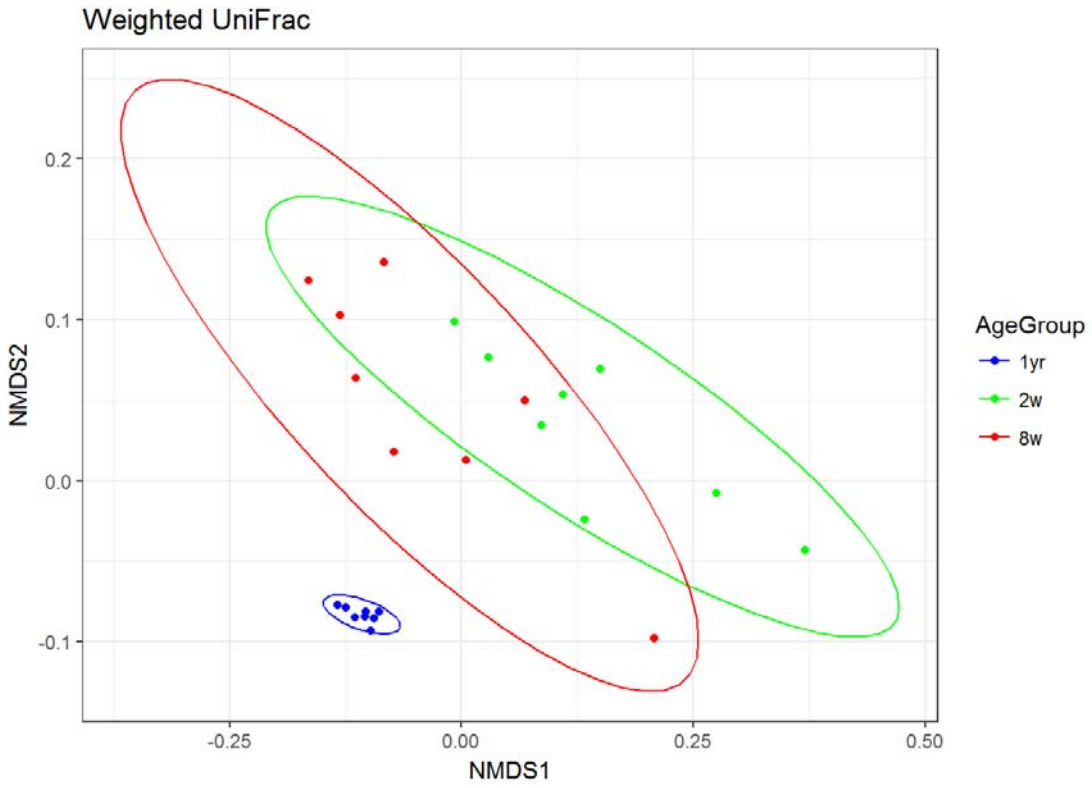
```
legend(0.3, 0.15, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)

#Add an ellipse for 2w
ordiellipse(wUF.ordu, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="green",
 draw="polygon", alpha=200, show.groups = c("2w"), border=FALSE)

#Add an ellipse for 8w
ordiellipse(wUF.ordu, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="red", d
raw="polygon", alpha=200, show.groups = c("8w"), border=FALSE)

#Add an ellipse for 1yr
ordiellipse(wUF.ordu, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="blue",
draw="polygon", alpha=200, show.groups = c("1yr"), border=FALSE)
```



We can also plot ellipses in `ggplot2`. However, these ellipses are not the exact same at the standard error ellipses used with OTU-based metrics as they use different underlying calculations. However, they get at the same question of confidence intervals for groups of points on an nMDS.

We plot ellipses with `ggplot2` by adding the `stat_ellipse` function to our plot.

```
plot_ordination(physeq.tree, wUF.ordu, type="sites", color="AgeGroup") +
  scale_colour_manual(values=c("2w"="green", "8w"="red", "1yr"="blue")) +
  theme_bw() +
  stat_ellipse() +
  ggtitle("Weighted UniFrac")
```



3D plots

3D UniFrac ordinations are not currently supported by `phyloseq`. We see that our ordinations only include 2 dimensions.

```
wUF.ordu
```

```
##
## Call:
## metaMDS(comm = ps.dist)
##
## global Multidimensional Scaling using monoMDS
##
## Data:     ps.dist
## Distance: user supplied
##
## Dimensions: 2
## Stress:     0.08645297
## Stress type 1, weak ties
## Two convergent solutions found after 20 tries
## Scaling: centring, PC rotation
## Species: scores missing
```

But we can instead calculate UniFrac distances using `UniFrac` and ordinating for 3-axes with `metaMDS`.

```
wUF.dist = UniFrac(physeq.tree, weighted=TRUE, normalized=TRUE)
```

```
## Warning in UniFrac(physeq.tree, weighted = TRUE, normalized = TRUE):
## Randomly assigning root as -- Otu03194 -- in the phylogenetic tree in the
## data you provided.
```

```
wUF.nmds.3D = metaMDS(wUF.dist, method="NMDS", k=3)
```

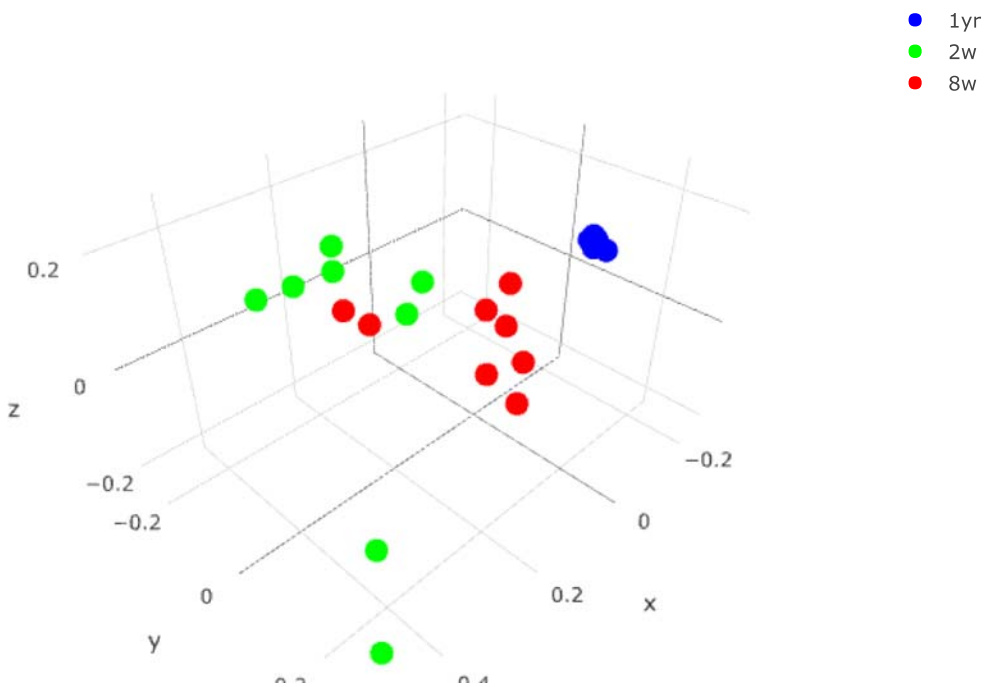
```
## Run 0 stress 0.04217486
## Run 1 stress 0.05952471
## Run 2 stress 0.05952709
## Run 3 stress 0.042174
## ... New best solution
## ... Procrustes: rmse 0.0003317483  max resid 0.0007893038
## ... Similar to previous best
## Run 4 stress 0.04217542
## ... Procrustes: rmse 0.0005403913  max resid 0.0014387
## ... Similar to previous best
## Run 5 stress 0.0421741
## ... Procrustes: rmse 0.0001810271  max resid 0.000555628
## ... Similar to previous best
## Run 6 stress 0.05952602
## Run 7 stress 0.04217451
## ... Procrustes: rmse 0.0003976044  max resid 0.001227917
## ... Similar to previous best
## Run 8 stress 0.06815104
## Run 9 stress 0.05952564
## Run 10 stress 0.04217457
## ... Procrustes: rmse 0.0004479109  max resid 0.001435945
## ... Similar to previous best
## Run 11 stress 0.04217428
## ... Procrustes: rmse 0.0003207273  max resid 0.0009212836
## ... Similar to previous best
## Run 12 stress 0.04217476
## ... Procrustes: rmse 0.0004904995  max resid 0.001357519
## ... Similar to previous best
## Run 13 stress 0.04217443
## ... Procrustes: rmse 0.0003308483  max resid 0.0008748533
## ... Similar to previous best
## Run 14 stress 0.04217414
## ... Procrustes: rmse 0.0002102509  max resid 0.000611423
## ... Similar to previous best
## Run 15 stress 0.04217491
## ... Procrustes: rmse 0.0005257634  max resid 0.001791904
## ... Similar to previous best
## Run 16 stress 0.04217454
## ... Procrustes: rmse 0.0003986916  max resid 0.001121447
## ... Similar to previous best
## Run 17 stress 0.04217553
## ... Procrustes: rmse 0.0004447142  max resid 0.001546131
## ... Similar to previous best
## Run 18 stress 0.04217399
## ... New best solution
## ... Procrustes: rmse 0.0001824097  max resid 0.0005684325
## ... Similar to previous best
## Run 19 stress 0.04217406
## ... Procrustes: rmse 7.68744e-05  max resid 0.0001772352
## ... Similar to previous best
## Run 20 stress 0.04217417
## ... Procrustes: rmse 0.0001240512  max resid 0.0002862878
## ... Similar to previous best
## *** Solution reached
```

Then, similar to what we did with Bray-Curtis/Jaccard, we pull out the xyz values and plot with `plotly`.

```
wUFxyz = scores(wUF.nmds.3D, display="sites")
#This is a table that looks like
wUFxyz
```

```
##                  NMDS1        NMDS2       NMDS3
## 5017.1yr.F -0.19591424  0.107765310  0.07968290
## 5017.2w.F   0.40329083  0.187040546 -0.11891085
## 5017.8w.F  -0.06738145  0.046058811 -0.21927277
## 5020.1yr.F -0.21311918  0.100813200  0.06833139
## 5020.2w.F  -0.02918765 -0.163606283 -0.02929884
## 5020.8w.F   0.03375300  0.054503745 -0.09099989
## 5026.1yr.F -0.22482781  0.066613100  0.05594134
## 5026.2w.F   0.13241677 -0.217029557  0.08745439
## 5026.8w.F   0.38996273  0.135464299  0.24011205
## 5031.1yr.F -0.19996967  0.080398029  0.09445703
## 5031.2w.F   0.19084848 -0.256852240  0.01563640
## 5031.8w.F  -0.13587208 -0.042300350 -0.02591350
## 5037.1yr.F -0.21800838  0.076413856  0.07189119
## 5037.2w.F   0.05187202 -0.120151694 -0.04223782
## 5037.8w.F   0.14227112 -0.115591151 -0.01897721
## 5041.1yr.F -0.20911338  0.081709200  0.07441520
## 5041.2w.F   0.27813371 -0.237693762  0.03647625
## 5041.8w.F  -0.13928666 -0.001531998 -0.18656755
## 5045.1yr.F -0.23328251  0.051043269  0.06274834
## 5045.2w.F   0.49259170  0.294540193 -0.14634317
## 5045.8w.F  -0.16902451 -0.126094687 -0.13841874
## 5053.1yr.F -0.21539833  0.077884489  0.08008741
## 5053.2w.F   0.27502987 -0.030380383  0.17559141
## 5053.8w.F  -0.13978439 -0.049015941 -0.12588496
```

```
plot_ly(x=wUFxyz[,1], y=wUFxyz[,2], z=wUFxyz[,3], type="scatter3d", mode="markers", color=meta$AgeGroup, col
ors=c("blue", "green", "red"))
```



Vectors for continuous variables

While it is easy to visualize categorical groups with coloring in nMDS, it is difficult to achieve the same effect with continuous variables. Instead, we can fit these variables as a vector on our nMDS plots.

To do this, we first fit the variables to our distances using the `envfit` function in `vegan`. You can do Bray-Curtis, Jaccard, weighted or unweighted UniFrac. Here, we will demonstrate with Bray-Curtis and weighted UniFrac.

```
fit.BC = envfit(BC.nmds, meta)
fit.BC
```

```
##
## ***VECTORS
##
##             NMDS1    NMDS2     r2 Pr(>r)
## AgeExact -0.99887 -0.04744 0.9765  0.001 ***
## ADGKG     0.12503  0.99215 0.0770  0.444
## chao     -0.98567  0.16868 0.9599  0.001 ***
## shannon  -0.69400  0.71997 0.9469  0.001 ***
## simpson   0.42087 -0.90712 0.7353  0.001 ***
## ace      -0.99746  0.07129 0.9078  0.001 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
##
## ***FACTORS:
##
## Centroids:
##                   NMDS1   NMDS2
## Animalcow5017   -0.1841  0.5449
## Animalcow5020    0.0059  0.6577
## Animalcow5026    0.4243 -0.8826
## Animalcow5031   -0.2442  0.1175
## Animalcow5037    0.4946 -0.0566
## Animalcow5041    0.0500 -0.0290
## Animalcow5045   -0.1374 -0.3384
## Animalcow5053   -0.4090 -0.0134
## AgeGroup1yr     -4.4470 -0.1800
## AgeGroup2w       2.5047 -1.0509
## AgeGroup8w       1.9422  1.2309
## AgeGroup.ord2w   2.5047 -1.0509
## AgeGroup.ord8w   1.9422  1.2309
## AgeGroup.ord1yr -4.4470 -0.1800
##
## Goodness of fit:
##                  r2 Pr(>r)
## Animal       0.0248  0.997
## AgeGroup     0.9134  0.001 ***
## AgeGroup.ord 0.9134  0.001 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
```

We see that it has automatically fit every variable in our meta table.

The simplest way around this is to just ask envfit to run on only the variables you want.

```
fit.BC = envfit(BC.nmds, meta[,c("AgeGroup", "ADGKG")])
fit.BC
```

```
##
## ***VECTORS
##
##         NMDS1   NMDS2    r2 Pr(>r)
## ADGKG 0.12503 0.99215 0.077  0.452
## Permutation: free
## Number of permutations: 999
##
## ***FACTORS:
##
## Centroids:
##               NMDS1   NMDS2
## AgeGroup1yr -4.4470 -0.1800
## AgeGroup2w   2.5047 -1.0509
## AgeGroup8w   1.9422  1.2309
##
## Goodness of fit:
##              r2 Pr(>r)
## AgeGroup 0.9134  0.001 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
```

We repeat for weighted UniFrac

```
fit.wUF = envfit(wUF.ordu, meta[,c("AgeGroup", "ADGKG")])
fit.wUF
```

```
##
## ***VECTORS
##
##          NMDS1    NMDS2     r2 Pr(>r)
## ADGKG -0.17846  0.98395 0.0398   0.66
## Permutation: free
## Number of permutations: 999
##
## ***FACTORS:
##
## Centroids:
##               NMDS1   NMDS2
## AgeGroup1yr -0.1076 -0.0834
## AgeGroup2w   0.1432  0.0322
## AgeGroup8w  -0.0356  0.0511
##
## Goodness of fit:
##              r2 Pr(>r)
## AgeGroup 0.5588  0.001 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
```

For categorical variables, envfit will label the centroid of the data for each group in the nMDS with that group's name. For continuous variables, it adds an arrow in the direction from smallest to largest value.

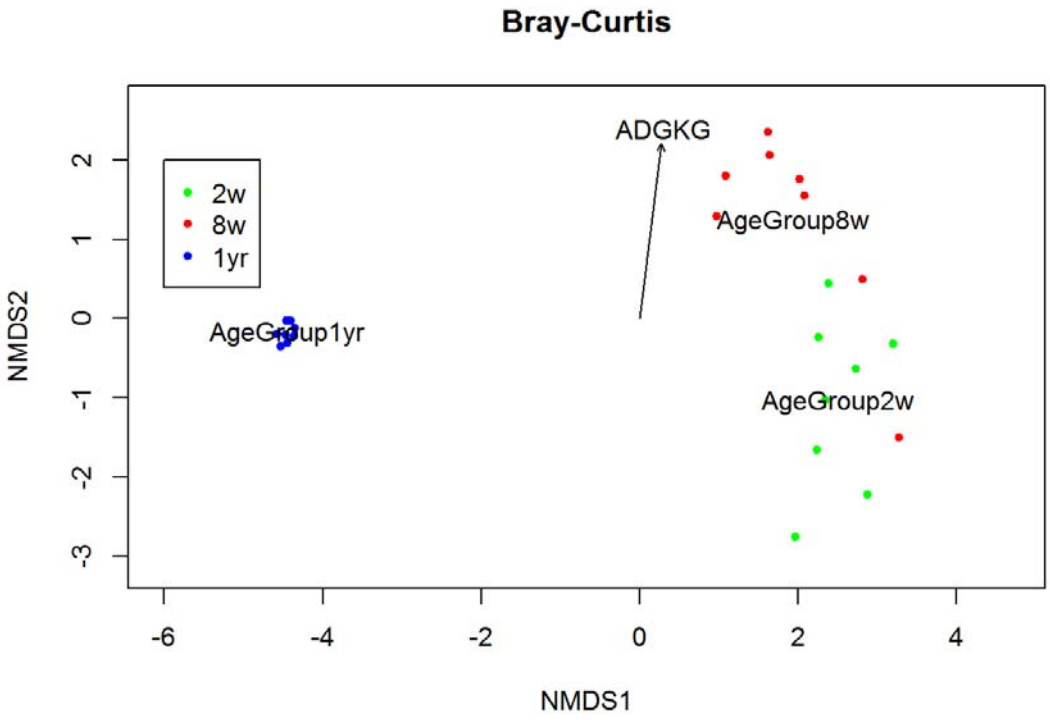
Note: The P-values for variables in `envfit` are not equivalent to the P-values for our ANOVA/Kruskal/GLM tests. Instead, `envfit` P-values tell you how well the arrow or centroids fit the *x-y data of the nMDS*, not the underlying distance matrix. In general, if your nMDS is a good representation of the data (low stress value) and the variable was significant in its appropriate ANOVA/Kruskal/GLM test, the fitted arrow/centroids will also be significant. And if your nMDS is a good representation of the data and the variable was *not* significant, the fitted arrow/centroids will also *not* be significant. We see this type of result here, but this will not always be the case.

However, if your nMDS stress was borderline or not great and/or your variable was borderline significant or not, you may see divergent results for the arrow/centroid. This does not mean that the result you got in ANOVA/Kruskal/GLM was invalid. It just means that it's difficult to visualize this result as a simple arrow or centroids on a 2D plot. Regardless, non-significant variables in `envfit` that you know are signficant in other tests may still be represented on an nMDS as a visual aid.

Thus, we plot our 2D nMDS colored by age with an arrow for the ADG variable even though that arrow was not significant. Since the ADG variable was also not significant in GLM, we probably won't use these plot in a publication, but it is good practice.

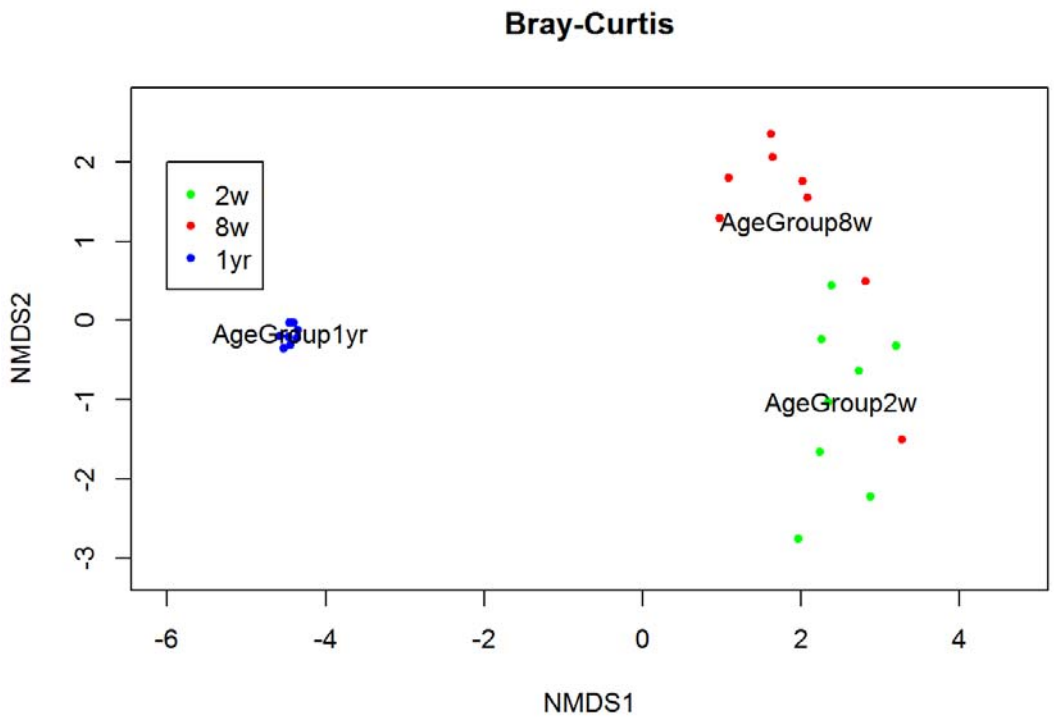
For Bray-Curtis:

```
plot(BC.nmds, type="n", main="Bray-Curtis")
points(BC.nmds, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(-6, 2, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Add fitted variables
plot(fit.BC, col="black")
```



You could also ask it to only plot variables with a fit P-value < 0.05. So we would only see the centroids

```
plot(BC.nmds, type="n", main="Bray-Curtis")
points(BC.nmds, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(-6, 2, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Add fitted variables
plot(fit.BC, col="black", p.max=0.05)
```

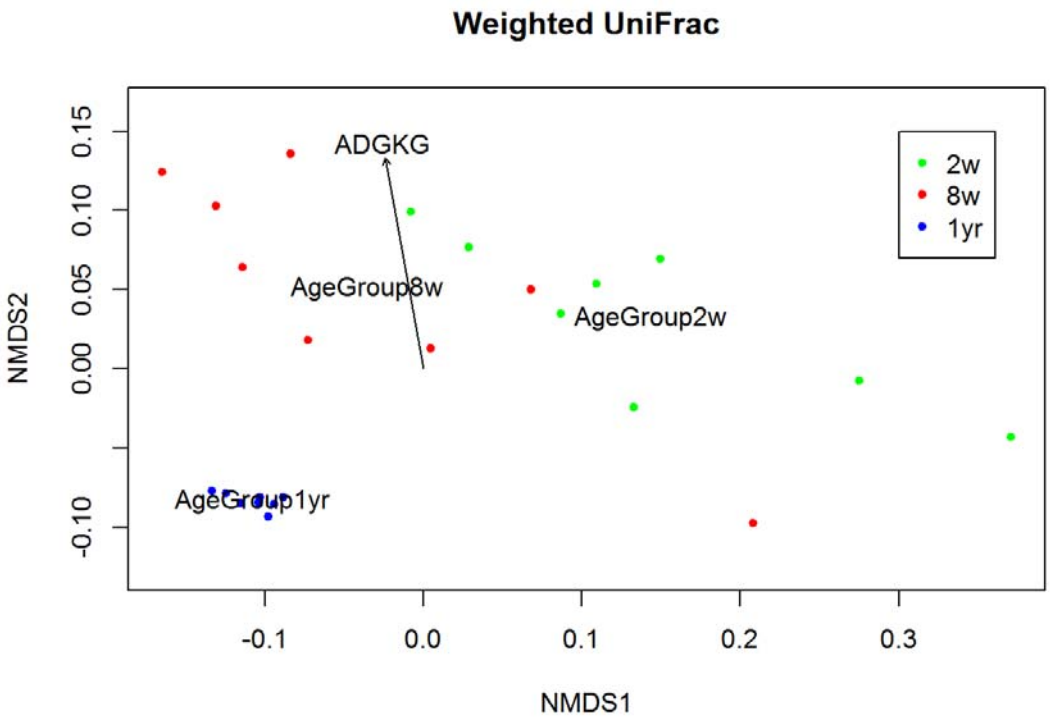


Weighted UniFrac

```
plot(wUF.ordu, type="n", main="Weighted UniFrac")
```

```
## Warning in ordiplot(x, choices = choices, type = type, display = display, :
## Species scores not available
```

```
points(wUF.ordu, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(.3,.15, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Add fitted variables
plot(fit.wUF, col="black")
```



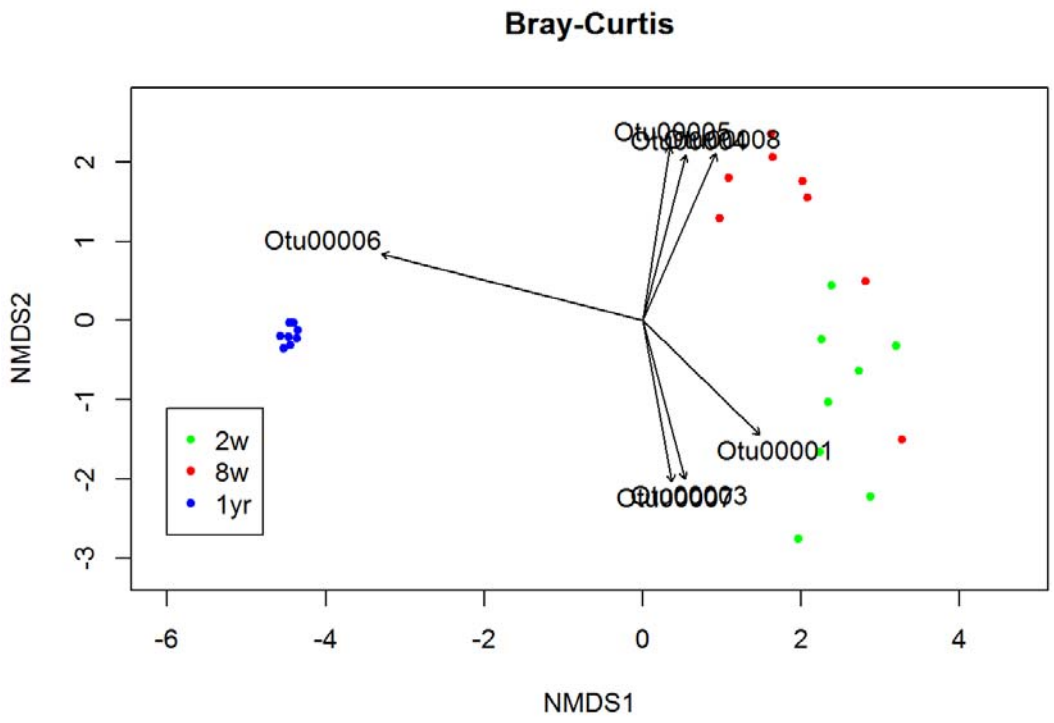
You could also fit your OTU.clean table to the nMDS to add arrow(s) for specific OTUs within the plot. OTU arrows that, say, go in the same direction as an age group centroid tend to increase in abundance in that age group. The opposite direction would indicate that an OTU decreases in abundance in that age group.

Fitting all OTUs would take awhile so we will only fit the first 10 in our table.

```
fit.BC.OTU = envfit(BC.nmds, OTU.clean[,1:10])
fit.BC.OTU
```

```
##
## ***VECTORS
##
##             NMDS1    NMDS2     r2 Pr(>r)
## Otu00001  0.71738 -0.69668 0.2478  0.033 *
## Otu00002  0.46984 -0.88275 0.2109  0.057 .
## Otu00003  0.25719 -0.96636 0.2503  0.021 *
## Otu00004  0.25006  0.96823 0.2738  0.030 *
## Otu00005  0.15473  0.98796 0.2910  0.003 **
## Otu00006 -0.96867  0.24837 0.6743  0.001 ***
## Otu00007  0.17991 -0.98368 0.2488  0.009 **
## Otu00008  0.40157  0.91583 0.3108  0.016 *
## Otu00009  0.26275 -0.96487 0.1894  0.062 .
## Otu00010  0.33868 -0.94090 0.1552  0.078 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
```

```
#We will only plot significant arrows in this case
plot(BC.nmds, type="n", main="Bray-Curtis")
points(BC.nmds, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(-6, -1.1, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Add fitted variables
plot(fit.BC.OTU, col="black", p.max=0.05)
```

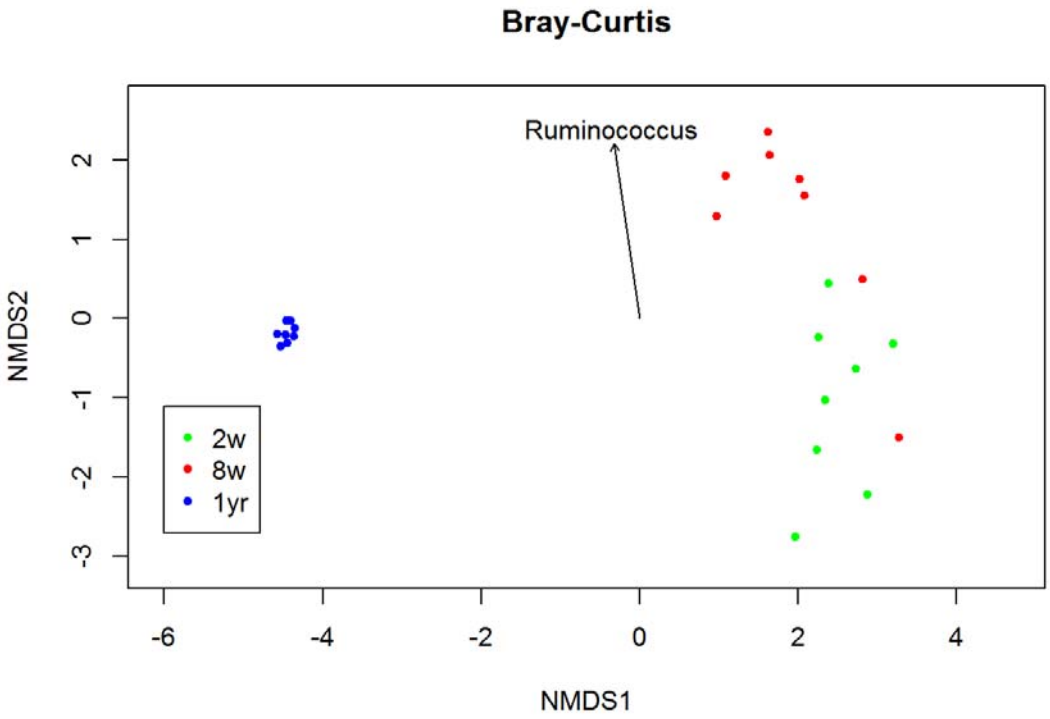


You could also think about plotting higher taxonomic levels like summed genera or family groups of OTUs.

```
#Extract all OTUs within the genus Ruminococcus
OTU.Rumino = OTU.clean[,tax.clean$Genus == "g__Ruminococcus"]
#Sum the abundances of the Ruminococcaceae OTUs into one variable (column)
OTU.Rumino$Rumino.sum = rowSums(OTU.Rumino)

#Fit the new Ruminococcaceae group
fit.BC.Rumino = envfit(BC.nmds, OTU.Rumino$Rumino.sum)
fit.BC.Rumino
```

```
##
## ***VECTORS
##
##         NMDS1    NMDS2     r2 Pr(>r)
## [1,] -0.14506  0.98942 0.6621  0.001 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Permutation: free
## Number of permutations: 999
```

```
#Plot
plot(BC.nmds, type="n", main="Bray-Curtis")
points(BC.nmds, pch=20, display="sites", col=c("blue", "green", "red")[meta$AgeGroup])
legend(-6, -1.1, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
#Add fitted variables
plot(fit.BC.Rumino, col="black", labels=c("Ruminococcus"))
```



Statistically test beta-diversity

While nMDS gives us a visual of beta-diversity, it does not test for statistical differences. We do this with permutational analysis of variance (PERMANOVA) or analysis of similarity (ANOSIM). These test whether the overall microbial community differs by your variable of interest.

You can run them with Bray-Curtis, Jaccard, weighted or unweighted UniFrac to answer different questions. For example, if your variable is significant for Bray-Curtis/weighted UniFrac but not Jaccard/unweighted UniFrac, this means your groups tend to have the same OTUs (richness) but different abundances of those OTUs (diversity). When variables are signficant for Bray-Curtis/Jaccard but not UniFrac, this indicates that your samples have different specific OTUs but similar taxa. Like group 1 has a lot of *Prevotella* OTU1 and group 2 has a lot of *Prevotella* OTU2, but they are both *Prevotella* so UniFrac treats them as being very similar.

PERMANOVA

For Bray-Curtis or Jaccard, we use the `vegan` package to calculate distances and run PERMANOVA. As with ANOVA/glm of alpha-diversity, we want to include all variables that could interact in one model.

Note: adonis cannot handle or account for NA or blanks in your data. Subset to only samples with complete metadata before running `vegdist` if these exist.

```
#Calculate distance and save as a matrix
BC.dist=vegdist(OTU.clean, distance="bray")
#Run PERMANOVA on distances.
adonis(BC.dist ~ AgeGroup*ADGKG, data = meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = BC.dist ~ AgeGroup * ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##                Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup        2    3.9720 1.98600  8.0116 0.44481 0.000999 ***
## ADGKG           1    0.1979 0.19791  0.7984 0.02216 0.618382
## AgeGroup:ADGKG  2    0.2976 0.14881  0.6003 0.03333 0.929071
## Residuals      18    4.4620 0.24789         0.49969
## Total          23    8.9296                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Similarly for Jaccard

```
J.dist=vegdist(OTU.clean, distance="jaccard")
adonis(J.dist ~ AgeGroup*ADGKG, data = meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = J.dist ~ AgeGroup * ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##                Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup        2    3.9720 1.98600  8.0116 0.44481 0.000999 ***
## ADGKG           1    0.1979 0.19791  0.7984 0.02216 0.632368
## AgeGroup:ADGKG  2    0.2976 0.14881  0.6003 0.03333 0.920080
## Residuals      18    4.4620 0.24789         0.49969
## Total          23    8.9296                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that the interaction is not significant so we remove it.

```
adonis(BC.dist ~ AgeGroup+ADGKG, data = meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = BC.dist ~ AgeGroup + ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##           Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup   2    3.9720 1.98600  8.3451 0.44481 0.000999 ***
## ADGKG      1    0.1979 0.19791  0.8316 0.02216 0.616384
## Residuals 20    4.7597 0.23798         0.53302
## Total     23    8.9296                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
adonis(J.dist ~ AgeGroup+ADGKG, data = meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = J.dist ~ AgeGroup + ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##           Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup   2    3.9720 1.98600  8.3451 0.44481 0.000999 ***
## ADGKG      1    0.1979 0.19791  0.8316 0.02216 0.566434
## Residuals 20    4.7597 0.23798         0.53302
## Total     23    8.9296                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For UniFrac, we use the `phyloseq` package to calculate distances and then `vegan` to run PERMANOVA.

```
wUF.dist = UniFrac(physeq.tree, weighted=TRUE, normalized=TRUE)
```

```
## Warning in UniFrac(physeq.tree, weighted = TRUE, normalized = TRUE):
## Randomly assigning root as -- Otu00842 -- in the phylogenetic tree in the
## data you provided.
```

```
adonis(wUF.dist ~ AgeGroup*ADGKG, data=meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = wUF.dist ~ AgeGroup * ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##                Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup        2   0.71682 0.35841  7.6290 0.43422 0.000999 ***
## ADGKG           1   0.03281 0.03281  0.6984 0.01988 0.665335
## AgeGroup:ADGKG  2   0.05553 0.02777  0.5910 0.03364 0.871129
## Residuals      18   0.84564 0.04698         0.51226
## Total          23   1.65080                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
uwUF.dist = UniFrac(physeq.tree, weighted=FALSE, normalized=TRUE)
```

```
## Warning in UniFrac(physeq.tree, weighted = FALSE, normalized = TRUE):
## Randomly assigning root as -- Otu01729 -- in the phylogenetic tree in the
## data you provided.
```

```
adonis(uwUF.dist ~ AgeGroup*ADGKG, data=meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = uwUF.dist ~ AgeGroup * ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##               Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup        2    3.4956 1.74781  9.1479 0.46952 0.000999 ***
## ADGKG           1    0.2434 0.24343  1.2741 0.03270 0.218781
## AgeGroup:ADGKG  2    0.2669 0.13344  0.6984 0.03585 0.832168
## Residuals      18    3.4391 0.19106         0.46193
## Total          23    7.4450                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Remove non-significant interaction term

```
adonis(wUF.dist ~ AgeGroup+ADGKG, data=meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = wUF.dist ~ AgeGroup + ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##           Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup   2   0.71682 0.35841  7.9543 0.43422 0.000999 ***
## ADGKG      1   0.03281 0.03281  0.7282 0.01988 0.626374
## Residuals 20   0.90117 0.04506         0.54590
## Total     23   1.65080                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
adonis(uwUF.dist ~ AgeGroup+ADGKG, data=meta, permutations = 1000)
```

```
##
## Call:
## adonis(formula = uwUF.dist ~ AgeGroup + ADGKG, data = meta, permutations = 1000)
##
## Permutation: free
## Number of permutations: 1000
##
## Terms added sequentially (first to last)
##
##           Df SumsOfSqs MeanSqs F.Model      R2   Pr(>F)
## AgeGroup   2    3.4956 1.74781  9.4324 0.46952 0.000999 ***
## ADGKG      1    0.2434 0.24343  1.3137 0.03270 0.206793
## Residuals 20    3.7060 0.18530         0.49778
## Total     23    7.4450                 1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

ANOSIM

If you have very different group sizes, you may consider analysis of similarities (ANOSIM) instead of PERMANOVA. This test does not assume equal group variances. However, it only allows simple 1 variable models with no interactions and can only be used for categorical (AgeGroup), not continuous (ADG) variables. So, ANOSIM has a lot of limitations and should only be used if you group sizes are *very, very* different, like 10 vs 100.

For example, Bray-Curtis:

```
anosim(BC.dist, meta$AgeGroup, permutations = 1000)
```

```
##
## Call:
## anosim(dat = BC.dist, grouping = meta$AgeGroup, permutations = 1000)
## Dissimilarity: bray
##
## ANOSIM statistic R: 0.8467
##       Significance: 0.000999
##
## Permutation: free
## Number of permutations: 1000
```

Overall, from the nMDS of various beta-diversity metrics (OTU- and phylogenetic-based) and statistical analyses, it is clear that age significantly impacts the fecal microbiota of dairy cows.

2D variables

These analyses are for comparing the microbiota to metadata that cannot fit in a single column and therefore, must be represented as a matrix of its own. For example, PERMANOVA can only tell you that the microbiota differs according to a single short chain fatty acid (SCFA), but other tests can tell you that the microbiota differs according to the overall SCFA profile. This section is also useful for comparing data if you have multiple OTU tables, like for bacteria, archaea, and fungi.

`Mantel` from `vegan` tests if two distance matrices co-vary *e.g.* does the data in matrix 1 change in the same way as the data in matrix 2. Like PERMANOVA, this test only tells you that the overall data co-vary, not which specific OTUs or SCFAs matter.

You can only compare samples were you have both types of data so we must subset our OTU table to only the samples that we also have SCFA for. The names are a little different between the tables so we also add ".F" to the SCFA names to make them match

```
OTU.SCFA = OTU.clean[row.names(OTU.clean) %in% paste(row.names(SCFA), ".F", sep=""),]
```

We then calculate distance matrices separately for each matrix. It is not necessary to do Bray-Curtis, Jaccard and UniFrac here since our SCFA data does not have any taxonomy to it.

```
dist1 = vegdist(OTU.SCFA)
dist2 = vegdist(SCFA)
```

Run a Mantel test comparing the 2 matrices.

```
mantel(dist1, dist2, permutations=100)
```

```
## 'nperm' >= set of all permutations: complete enumeration.
```

```
## Set of permutations < 'minperm'. Generating entire set.
```

```
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## mantel(xdis = dist1, ydis = dist2, permutations = 100)
##
## Mantel statistic r: -0.02423
##       Significance: 0.54167
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.540 0.552 0.596 0.629
## Permutation: free
## Number of permutations: 23
```

We see that the overall OTU table and SCFA tables do not co-vary.

You can also run Mantel on 3 matrices at once like so

Do not run as we do not have 3 matrices here

```
mantel.partial(dist1, dist2, dist3, permutations=100)
```

Beta dispersion

Sometimes it will be clear from nMDS that one group tends to vary more (be more spread out) than another group. You can test this statistically with multivariate homogeneity of group dispersion (variances).

Here is an example for Bray-Curtis. We use the same distance matrix we calculated for PERMANOVA/ANOSIM

Calculate dispersion (variances) within each group.

```
disp.age = betadisper(BC.dist, meta$AgeGroup)
```

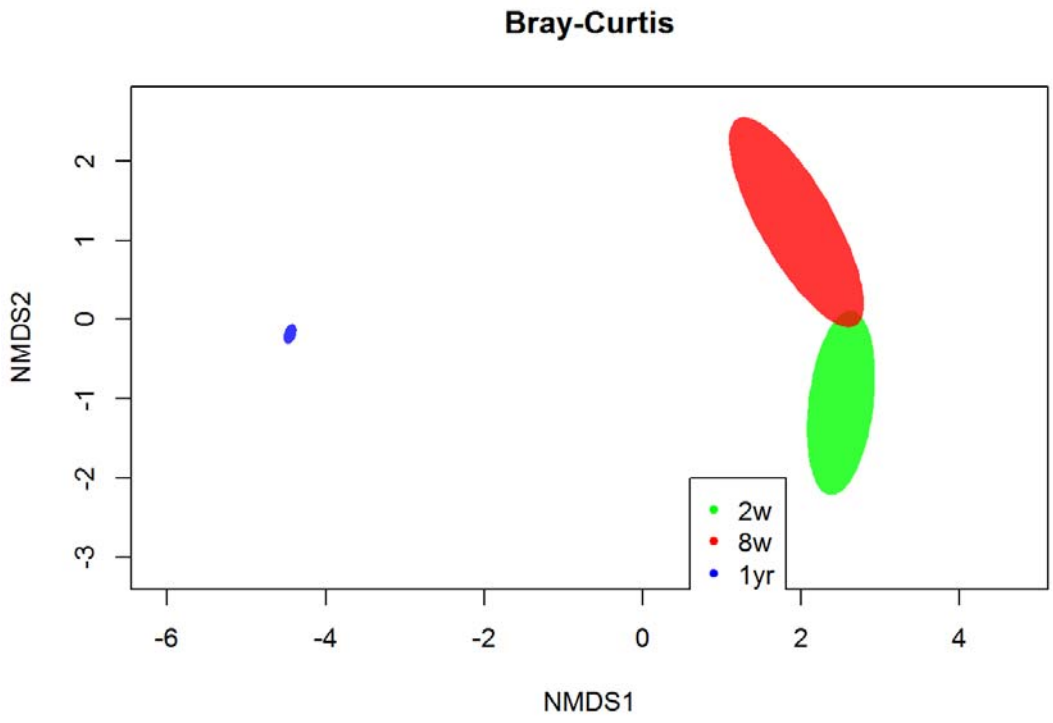
Perform an ANOVA-like test to determine if the variances differ by groups.

```
permutest(disp.age, pairwise=TRUE, permutations=1000)
```

```
##
## Permutation test for homogeneity of multivariate dispersions
## Permutation: free
## Number of permutations: 1000
##
## Response: Distances
##           Df  Sum Sq  Mean Sq     F N.Perm   Pr(>F)
## Groups     2 0.47459 0.237293 30.93   1000 0.000999 ***
## Residuals 21 0.16111 0.007672
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Pairwise comparisons:
## (Observed p-value below diagonal, permuted p-value above diagonal)
##            1yr         2w     8w
## 1yr            9.9900e-04 0.0010
## 2w  4.8556e-06            0.7902
## 8w  1.2886e-06 7.7206e-01
```

Combining this with our plot,

```
plot(BC.nmds, type="n", main="Bray-Curtis")
legend(.6,-2, legend=c("2w","8w","1yr"), col=c("green","red","blue"), pch=20)
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="green",
draw="polygon", alpha=200, show.groups = c("2w"), border=FALSE)
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="red", dr
aw="polygon", alpha=200, show.groups = c("8w"), border=FALSE)
ordiellipse(BC.nmds, groups=meta$AgeGroup, display="sites", kind="se", conf=0.99, label=FALSE, col="blue", d
raw="polygon", alpha=200, show.groups = c("1yr"), border=FALSE)
```



we see that 2 week and 8 week calves have similar variability in their fecal microbiotas but that both 2- and 8-week calves have more variable fecal microbiotas than 1-year heifers.

OTUs that differ by

Categorical variables

Just because the overall microbiota does or does not differ between age groups, does not mean specific OTUs do or don't differ by age. However, it is inadvisable to just test all OTUs in your data set against all variables of interest. Since you are running multiple similar tests, you need to apply a false discovery rate (fdr) correctios and correcting across all OTUs (5002 in this data set) will most likely result in no significant results after fdr correction. Also, you don't want to look at over 5000 P-values, do you?

There are a number of way to decrease the number of OTUs you're looking at

1. Don't use OTUs. Add together genus or family groups and test if all or some of these taxa differ across variables of interest
2. Apply an abundance cutoff such as only looking at OTUs/taxa that are at least 1% abundance in at least one sample
3. Apply a frequency cutoff such as only looking at OTUs/taxa that occur in at least 50% of samples
4. Combine 2 and 3

However, some of these methods are somewhat arbitrary. How do you pick an abundance or frequency cutoff? What if a low abundant OTU is of interest? And what if you are interested in possible species-level differences (OTUs) so high taxonomic levels aren't useful?

So, one way to non-arbitrarily select OTUs/taxa of interest is similarity percentages (SIMPER). SIMPER identifies the OTUs that most contribute to beta-diversity measures. These OTUs are the most abundant and/or most variable OTUs in the data set. **Note**: SIMPER outputs all pairwise comparisons (A-B, B-C, A-C, etc.) and thus, only works for categorical variables.

SIMPER's output is a list of OTUs which cumulatively explain 70%+ of the variation between each comparison. The numbers below the OTUs are **cumulative**, so to get each OTU's contribution, you must subtract the previous OTU's value.

For example

```
simper(OTU.clean, meta$AgeGroup, permutations=100)
```

```
## cumulative contributions of most influential species:
##
## $`1yr_2w`
##  Otu00002  Otu00001  Otu00003  Otu00007  Otu00011  Otu00006  Otu00009
## 0.0983761 0.1627191 0.2225335 0.2657879 0.2982889 0.3271508 0.3514210
##  Otu00014  Otu00022  Otu00018  Otu00012  Otu00016  Otu00004  Otu00021
## 0.3660756 0.3793171 0.3924608 0.4048922 0.4171422 0.4283988 0.4385280
##  Otu00008  Otu00025  Otu00028  Otu00023  Otu00037  Otu00013  Otu00035
## 0.4479076 0.4565849 0.4646081 0.4723795 0.4790690 0.4857141 0.4920793
##  Otu00055  Otu00030  Otu00036  Otu00040  Otu00042  Otu00010  Otu00049
## 0.4983615 0.5045449 0.5106265 0.5166717 0.5226378 0.5274331 0.5321886
##  Otu00046  Otu00033  Otu00031  Otu00081  Otu00051  Otu00064  Otu00056
## 0.5368030 0.5413764 0.5458188 0.5500936 0.5543565 0.5582465 0.5620674
##  Otu00032  Otu00052  Otu00062  Otu00026  Otu00020  Otu00074  Otu00069
## 0.5657989 0.5695078 0.5730822 0.5765920 0.5799406 0.5831741 0.5864067
##  Otu00066  Otu00077  Otu00148  Otu00073  Otu00067  Otu00065  Otu00076
## 0.5895953 0.5927428 0.5958511 0.5989588 0.6020549 0.6051241 0.6081334
##  Otu00075  Otu00091  Otu00048  Otu00097  Otu00068  Otu00050  Otu00084
## 0.6111073 0.6140400 0.6169121 0.6196512 0.6223697 0.6250661 0.6277023
##  Otu00100  Otu00019  Otu00063  Otu00039  Otu00086  Otu00071  Otu00101
## 0.6303356 0.6329664 0.6355752 0.6381709 0.6406744 0.6431362 0.6455850
##  Otu00089  Otu00096  Otu00095  Otu00108  Otu00088  Otu00103  Otu00094
## 0.6480310 0.6504700 0.6528884 0.6553007 0.6576757 0.6600472 0.6624184
##  Otu00098  Otu00116  Otu00090  Otu00105  Otu00104  Otu00099  Otu00059
## 0.6647575 0.6670589 0.6693444 0.6716046 0.6738590 0.6760506 0.6781917
##  Otu00106  Otu00115  Otu00102  Otu00110  Otu00119  Otu00118  Otu00034
## 0.6803196 0.6824245 0.6844633 0.6865021 0.6884972 0.6904775 0.6924261
##  Otu00114  Otu00093  Otu00124  Otu00045
## 0.6943714 0.6962690 0.6981558 0.7000319
##
## $`1yr_8w`
##   Otu00001   Otu00005   Otu00006   Otu00004   Otu00010   Otu00017
## 0.03765603 0.07335078 0.10010930 0.12226268 0.14087762 0.15688502
##   Otu00008   Otu00009   Otu00015   Otu00018   Otu00016   Otu00014
## 0.17205091 0.18718833 0.20107546 0.21456235 0.22713556 0.23964967
##   Otu00029   Otu00019   Otu00021   Otu00025   Otu00024   Otu00037
## 0.25102468 0.26162658 0.27202671 0.28093293 0.28829315 0.29516652
##   Otu00035   Otu00044   Otu00055   Otu00027   Otu00036   Otu00040
## 0.30170335 0.30821052 0.31465848 0.32109529 0.32733731 0.33354206
##   Otu00042   Otu00020   Otu00013   Otu00041   Otu00003   Otu00043
## 0.33966556 0.34564370 0.35158279 0.35717451 0.36261926 0.36799345
##   Otu00038   Otu00026   Otu00034   Otu00049   Otu00070   Otu00046
## 0.37334038 0.37836130 0.38334135 0.38822230 0.39310161 0.39783775
##   Otu00012   Otu00058   Otu00011   Otu00051   Otu00054   Otu00045
## 0.40234701 0.40670755 0.41102172 0.41521298 0.41939306 0.42353985
##   Otu00047   Otu00064   Otu00056   Otu00052   Otu00048   Otu00002
## 0.42764688 0.43163954 0.43556497 0.43937178 0.44313291 0.44683135
##   Otu00062   Otu00031   Otu00057   Otu00061   Otu00053   Otu00074
## 0.45050368 0.45405112 0.45759807 0.46109474 0.46455875 0.46787762
##   Otu00069   Otu00066   Otu00077   Otu00073   Otu00067   Otu00079
## 0.47119548 0.47447192 0.47770248 0.48089214 0.48406988 0.48721802
##   Otu00083   Otu00078   Otu00076   Otu00075   Otu00091   Otu00121
## 0.49033806 0.49342871 0.49651735 0.49956976 0.50257978 0.50549547
##   Otu00097   Otu00092   Otu00032   Otu00084   Otu00129   Otu00050
## 0.50830678 0.51111612 0.51389884 0.51660098 0.51922111 0.52181856
##   Otu00100   Otu00101   Otu00096   Otu00108   Otu00095   Otu00086
## 0.52434751 0.52686095 0.52936793 0.53184756 0.53429667 0.53674109
##   Otu00089   Otu00088   Otu00103   Otu00094   Otu00098   Otu00116
## 0.53918547 0.54162316 0.54405719 0.54649097 0.54889172 0.55125394
##   Otu00105   Otu00104   Otu00143   Otu00123   Otu00082   Otu00039
## 0.55357747 0.55589135 0.55819397 0.56049152 0.56278380 0.56503978
##   Otu00099   Otu00130   Otu00090   Otu00106   Otu00107   Otu00115
## 0.56728918 0.56953083 0.57176616 0.57395024 0.57611979 0.57828018
##   Otu00087   Otu00153   Otu00102   Otu00110   Otu00119   Otu00118
## 0.58042631 0.58252590 0.58461849 0.58671108 0.58875879 0.59079874
```

```
##   Otu00022   Otu00072   Otu00080   Otu00093   Otu00124   Otu00112
## 0.59281824 0.59481609 0.59678509 0.59873275 0.60067308 0.60260107
##   Otu00122   Otu00131   Otu00132   Otu00134   Otu00128   Otu00125
## 0.60450552 0.60639869 0.60828362 0.61014314 0.61199594 0.61383412
##   Otu00133   Otu00159   Otu00139   Otu00127   Otu00114   Otu00137
## 0.61566158 0.61747930 0.61928689 0.62106367 0.62282385 0.62455846
##   Otu00136   Otu00194   Otu00138   Otu00144   Otu00142   Otu00135
## 0.62629042 0.62801571 0.62974033 0.63143945 0.63312281 0.63480281
##   Otu00147   Otu00120   Otu00188   Otu00126   Otu00028   Otu00211
## 0.63647550 0.63814069 0.63980299 0.64140642 0.64300322 0.64457174
##   Otu00154   Otu00146   Otu00173   Otu00156   Otu00158   Otu00157
## 0.64612078 0.64764950 0.64917769 0.65068721 0.65217234 0.65364696
##   Otu00060   Otu00168   Otu00140   Otu00163   Otu00171   Otu00113
## 0.65508066 0.65651008 0.65793253 0.65931862 0.66069801 0.66207484
##   Otu00178   Otu00200   Otu00165   Otu00170   Otu00164   Otu00187
## 0.66344999 0.66480785 0.66616041 0.66748648 0.66881018 0.67012189
##   Otu00151   Otu00213   Otu00149   Otu00183   Otu00192   Otu00167
## 0.67141176 0.67269928 0.67397558 0.67525135 0.67652371 0.67778788
##   Otu00177   Otu00181   Otu00180   Otu00236   Otu00186   Otu00199
## 0.67904574 0.68029263 0.68151160 0.68272731 0.68393783 0.68512983
##   Otu00253   Otu00150   Otu00204   Otu00169   Otu00218   Otu00189
## 0.68632029 0.68750539 0.68867418 0.68982822 0.69097221 0.69210846
##   Otu00182   Otu00184   Otu00226   Otu00270   Otu00172   Otu00225
## 0.69323878 0.69436709 0.69548866 0.69660494 0.69770318 0.69878699
##   Otu00185   Otu00203
## 0.69986670 0.70093653
##
## $`2w_8w`
##  Otu00002  Otu00001  Otu00003  Otu00007  Otu00009  Otu00005  Otu00011
## 0.1101390 0.1804133 0.2466786 0.2952479 0.3351854 0.3745198 0.4100899
##  Otu00004  Otu00010  Otu00017  Otu00008  Otu00012  Otu00015  Otu00022
## 0.4397781 0.4641945 0.4818672 0.4987872 0.5154942 0.5307997 0.5454777
##  Otu00029  Otu00013  Otu00019  Otu00020  Otu00028  Otu00006  Otu00023
## 0.5580145 0.5704325 0.5824230 0.5910912 0.5996473 0.6081657 0.6166261
##  Otu00024  Otu00027  Otu00031  Otu00044  Otu00030  Otu00041  Otu00043
## 0.6247348 0.6322130 0.6396626 0.6468237 0.6539027 0.6600291 0.6659522
##  Otu00038  Otu00032  Otu00026  Otu00070  Otu00033  Otu00034  Otu00047
## 0.6718453 0.6776585 0.6834157 0.6887933 0.6940870 0.6992933 0.7044391
```

We see a number of OTUs that may differ between 1 or more age comparisons. However, these are just the OTUs that most contribute to Bray-Curtis measures between our age groups. *They are not necessarily significantly different.*

To test significance, we compare the relative abundance of an OTU across our age groups with Kruskal-Wallis (OTU abundance is never normally distributed, trust me). For example, OTU1 occurs in all SIMPER age comparisons and does, in fact, significantly differ by age.

```
kruskal.test(OTU.clean$Otu00001 ~ meta$AgeGroup)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  OTU.clean$Otu00001 by meta$AgeGroup
## Kruskal-Wallis chi-squared = 15.994, df = 2, p-value = 0.0003364
```

In contrast, OTU17 occurs in SIMPER but does not actually significantly differ by age group

```
kruskal.test(OTU.clean$Otu00017 ~ meta$AgeGroup)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  OTU.clean$Otu00017 by meta$AgeGroup
## Kruskal-Wallis chi-squared = 4.9767, df = 2, p-value = 0.08305
```

Note: These P-values have not been corrected from false discovery rate (fdr) yet.

Now, it would be very tedious to individually test every variable of interest in SIMPER and then test every SIMPER OTU in Kruskal-Wallis. So, Andrew Steinberger (Suen lab) has written two scripts to simplify both SIMPER and Kruskal-Wallis of SIMPER OTUs. The latest versions can be found on his GitHub page (https://github.com/asteinberger9/seq_scripts) and we have provided them for this workshop in `/Steinberger_scripts`

Disclaimer *Andrew has provided these scripts out of the goodness of his heart and provides no guarentee that they will work for your exact data set or with new versions of R/RStudio/vegan. You may contact him through GitHub with issues or errors, but it is not his job to troubleshoot for you. He may or may not address your concerns in an updated version of the scripts at a later time.*

The use of these scripts are as follows (from Steinberger GitHub with some modifications)

simper_pretty.R

This script is meant to rapidly perform the SIMPER function from the R package `vegan` for all comparisons of interest in a data set. Inputs are OTU and metadata tables, and the output is a .csv. User can tailor contents of .csv by setting perc_cutoff, low_cutoff, and low_val. This function can also handle taxonomic levels instead of OTU, but currently only select formats are compatible. Requires installation of the R package 'vegan'.

Usage:

simper.pretty(x, metrics, c('interesting'), perc_cutoff=0.5, low_cutoff = 'y', low_val=0.01, 'output_name')

Inputs:

- x: OTU table
- metrics: metadata table
- interesting: a list of the column headers for the columns of interest in the metrics file. e.g. c('int1','int2','int3')
- perc_cutoff: % cutoff for output OTUs, as decimal (i.e. write 50% as 0.5), larger % increases number OTUs in output.
- low_cutoff: 'y' if want to REMOVE OTUs that contribute less than 1%
- low_val: set value of low cutoff (0.01), ignored if low_cutoff='n'.
- output_name: the name that is appended to the output filename "_clean_simper.csv".

R_krusk.R

This script takes the output .csv of `simper_pretty.R` , and the OTU/metadata/taxonomy tables, and performs the non-parametric Kruskal-Wallis rank-sum test on each OTU in the .csv file. Output is a .csv file containing the same contents of simper.pretty output with the following info: p-value, fdr corrected p-value, OTU taxonomic classification (if applicable), mean rel. abund and std dev of otu/tax_lvl in group 1 of comparison, and mean rel. abund and std dev of otu/tax_lvl in group 2 of comparison. Requires installation of R packages 'vegan' and 'dplyr'.

Usage:

kruskal.pretty(x, metrics, csv, c('interesting'), 'output_name', taxonomy)

Inputs:

- x: OTU table
- metrics: metadata table
- csv: output from simper.pretty, must be imported as data.frame. e.g. csv= data.frame(read.csv("PATH to name_clean_simper.csv"))
- interesting: a list of the column headers for the columns of interest in the metrics file, should be same as simper.pretty inputs. e.g. c('int1','int2','int3')
- output_name= the name that is appended to the output filename "_krusk_simper.csv".
- taxonomy: The .taxonomy file output from classify.otu command in mothur. This is the UNALTERED tax file, not tax.clean (optional)

First, we load these functions into R.

```
source("Steinberger_scripts/simper_pretty.r")
source("Steinberger_scripts/R_krusk.r")
```

Then, we apply them to our data. We will ask for all SIMPER OTUs (`perc_cutoff = 1` , meaning up to cumulative 100%) but cutoff any OTUs that individually contribute less than 1% to SIMPER (`low_val=0.01`). You may want to consider different cutoffs for your data.

```
simper.pretty(OTU.clean, meta, c('AgeGroup'), perc_cutoff=1, low_cutoff = 'y', low_val=0.01, 'Age')

simper.results = data.frame(read.csv("Age_clean_simper.csv"))
kruskal.pretty(OTU.clean, meta, simper.results, c('AgeGroup'), 'Age', tax)
```

If we import the Kruskal-Wallis back into R and select only OTUs there were significantly different after fdr correction (fdr_krusk_p.val)…

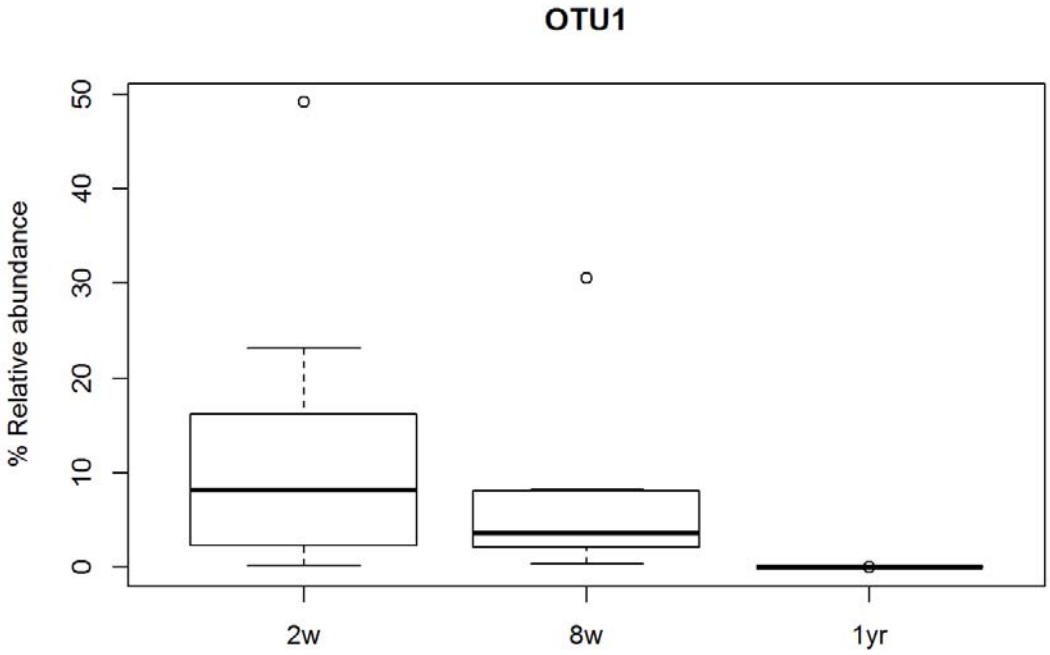
```
#Import
KW.results = data.frame(read.csv("Age_krusk_simper.csv"))
#Remove non-significant
KW.results.signif = KW.results[KW.results$fdr_krusk_p.val < 0.05,]
#Order by OTU#
KW.results.signif = KW.results.signif[with(KW.results.signif, order(OTU)),]
head(KW.results.signif)
```

```
##     X Comparison     SIMPER      OTU  krusk_p.val fdr_krusk_p.val
## 2   2     1yr_2w 0.06434298 Otu00001 0.0004510953     0.001383359
## 15 15     1yr_8w 0.03765603 Otu00001 0.0004510953     0.001383359
## 1   1     1yr_2w 0.09837610 Otu00002 0.0004510953     0.001383359
## 30 30      2w_8w 0.11013903 Otu00002 0.0208625823     0.029989962
## 3   3     1yr_2w 0.05981442 Otu00003 0.0003310658     0.001383359
## 32 32      2w_8w 0.06626526 Otu00003 0.0356919001     0.044373714
##
         Taxonomy
## 2          k__Bacteria;p__Firmicutes;c__Clostridia;o__Clostridiales;f__Ruminococcaceae;g__Faecalibacteriu
m;s__prausnitzii;
## 15         k__Bacteria;p__Firmicutes;c__Clostridia;o__Clostridiales;f__Ruminococcaceae;g__Faecalibacteriu
m;s__prausnitzii;
## 1          k__Bacteria;p__Firmicutes;c__Clostridia;o__Clostridiales;f__Ruminococcaceae;g__Faecalibacteriu
m;s__prausnitzii;
## 30         k__Bacteria;p__Firmicutes;c__Clostridia;o__Clostridiales;f__Ruminococcaceae;g__Faecalibacteriu
m;s__prausnitzii;
## 3  k__Bacteria;p__Actinobacteria;c__Coriobacteriia;o__Coriobacteriales;f__Coriobacteriaceae;g__Collinsell
a;s__aerofaciens;
## 32 k__Bacteria;p__Actinobacteria;c__Coriobacteriia;o__Coriobacteriales;f__Coriobacteriaceae;g__Collinsell
a;s__aerofaciens;
##    Left.mean.abund   Left.stdev Right.mean.abund Right.stdev
## 2     7.109140e-06 2.010768e-05      0.128370197  0.16351829
## 15    7.109140e-06 2.010768e-05      0.073292635  0.09803742
## 1     7.118451e-06 2.013402e-05      0.196185324  0.23796423
## 30    1.961853e-01 2.379642e-01      0.007205221  0.01601067
## 3     0.000000e+00 0.000000e+00      0.119333403  0.18000346
## 32    1.193334e-01 1.800035e-01      0.010598818  0.02126522
```

we see a number of OTU that significantly differ by age group.

Looking at OTU1 as relative abundance

```
#Calculate abundance
abund = OTU.clean/rowSums(OTU.clean)*100
#plot
boxplot(abund$Otu00001 ~ meta$AgeGroup.ord, ylab="% Relative abundance", main="OTU1")
```



and using the P-values in KW.results.signif, we can say that OTU1 is significantly less abundant in 1yr animals compared to either 2w or 8w calves.

Continuous variables

For continuous variables, there is no simple test like SIMPER to pull out OTUs likely to differ across your variable. You could run linear models `glm` of the OTU abundances with different distributions `family=` similar to what we did with Chao richness. However, OTU abundance data is not normal nor does it fit well with other standard distributions due to its many zeros. So, you will need to test a number of distributions and transformations of the data to find a suitable model.

Correlations

So, you can also approach continuous variables as correlations. Generally, only strong correlations (r > 0.5 or r < -0.5) should be reported and if you have a lot that fall into the “strong” category, you can up the cut off, say, to r > 0.75 or r < -0.75. There are many correlation options. I like Kendall-Tau because it does not assume linearity or normality. Type ??cor in the R console to learn others that are available.

Also, consider options to decrease the number of OTUs tested or you will be dealing with a huge table. Like only ones at >X% abundance? Only ones found in SIMPER and/or KW analyses of other important variables?

Here, we will correlate ADG to OTUs with at least 5% relative abundance in at least one sample in our data set.

```
#Remember we calculated abundance before with
#abund = OTU.clean/rowSums(OTU.clean)*100

#Subset OTUs to abundance cutoff
OTU.abund = OTU.clean[, apply(abund, MARGIN=2, function(x) any(x > 5))]

cor.kendall = cor(OTU.abund, meta$ADGKG, method = "kendall")
cor.kendall
```

```
##                 [,1]
## Otu00001  0.189852125
## Otu00002  0.211764129
## Otu00003  0.027397313
## Otu00004  0.275867615
## Otu00005  0.165056323
## Otu00006 -0.114462240
## Otu00007  0.143930930
## Otu00008  0.211764129
## Otu00009 -0.177517901
## Otu00010  0.176299258
## Otu00011  0.208334326
## Otu00012  0.017236256
## Otu00013  0.269669049
## Otu00015  0.018077538
## Otu00016 -0.257293680
## Otu00017  0.284293111
## Otu00019  0.172479145
## Otu00020  0.102188122
## Otu00022 -0.034040152
## Otu00023  0.004106646
## Otu00024  0.073416202
## Otu00027  0.412640807
## Otu00029  0.076924424
## Otu00030 -0.077670805
## Otu00031  0.286002668
## Otu00038 -0.271163072
## Otu00041  0.125193349
## Otu00043  0.189645652
## Otu00044  0.239065695
## Otu00053 -0.217652255
## Otu00055 -0.112428004
## Otu00070 -0.037317590
```

In this case, we don't see any strong correlations. However, if we did, we could use those OTUs as our list of ones that are of interest to check for significance with glm.

Next, we will correlate SCFAs with OTUs with at least 1% relative abundance in at least one sample in our data set. We will use only samples for which we also have SCFA data.

```
#Calculate abundances
abund.SCFA = OTU.SCFA/rowSums(OTU.SCFA)*100

#Subset OTUs to abundance cutoff
OTU.SCFA.abund = OTU.SCFA[, apply(abund.SCFA, MARGIN=2, function(x) any(x > 1))]

cor.kendall = cor(OTU.SCFA.abund, SCFA, method = "kendall")
cor.kendall
```

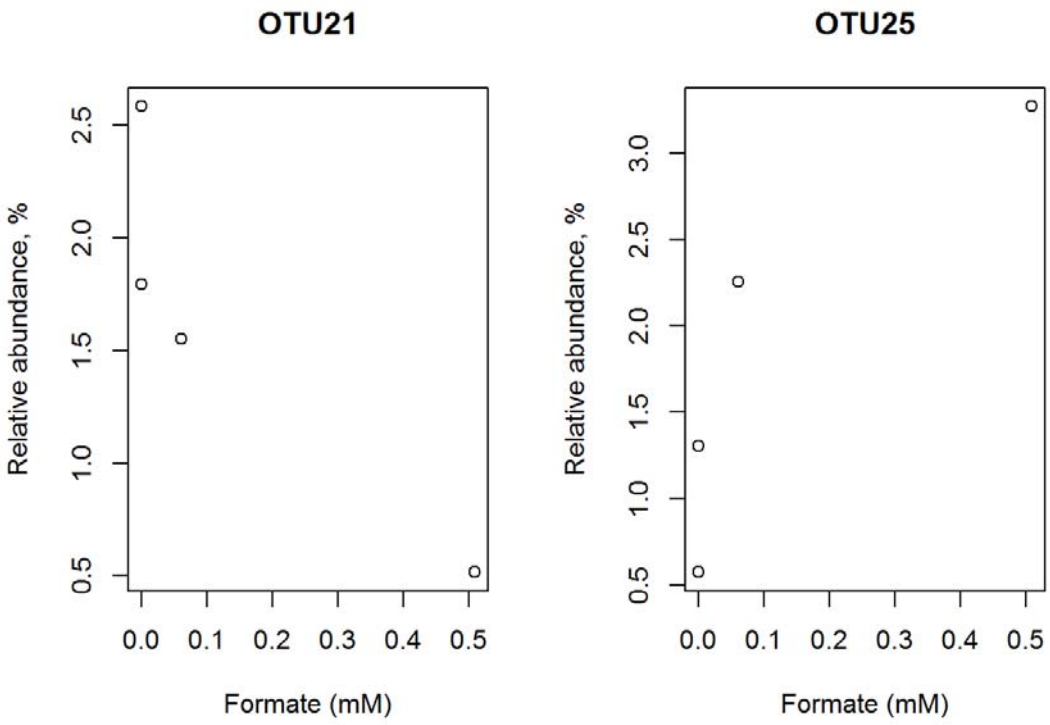
```
##             Formate    Acetate Propionate Isobutyrate   Butyrate
## Otu00006  0.0000000  0.1825742  0.1825742   0.1825742  0.1825742
## Otu00014  0.1825742  0.3333333  0.3333333   0.0000000  0.3333333
## Otu00016 -0.1825742 -0.3333333 -0.3333333  -0.6666667 -0.3333333
## Otu00018 -0.1825742 -0.3333333 -0.3333333  -0.6666667 -0.3333333
## Otu00021 -0.9128709 -0.6666667 -0.6666667  -0.3333333 -0.6666667
## Otu00025  0.9128709  0.6666667  0.6666667   0.3333333  0.6666667
## Otu00035 -0.5477226 -0.6666667 -0.6666667  -1.0000000 -0.6666667
## Otu00036 -0.5477226 -0.6666667 -0.6666667  -0.3333333 -0.6666667
## Otu00037 -0.1825742  0.0000000  0.0000000   0.3333333  0.0000000
## Otu00040 -0.5477226 -0.6666667 -0.6666667  -1.0000000 -0.6666667
## Otu00042  0.1825742  0.3333333  0.3333333   0.0000000  0.3333333
## Otu00046 -0.1825742 -0.3333333 -0.3333333  -0.6666667 -0.3333333
## Otu00049 -0.1825742 -0.3333333 -0.3333333   0.0000000 -0.3333333
## Otu00051  0.5477226  0.3333333  0.3333333   0.6666667  0.3333333
## Otu00052 -0.5477226 -0.6666667 -0.6666667  -1.0000000 -0.6666667
## Otu00056 -0.1825742 -0.3333333 -0.3333333  -0.6666667 -0.3333333
## Otu00064 -0.5477226 -0.3333333 -0.3333333  -0.6666667 -0.3333333
## Otu00066 -0.5477226 -0.6666667 -0.6666667  -1.0000000 -0.6666667
## Otu00067  0.1825742  0.0000000  0.0000000   0.3333333  0.0000000
## Otu00069  0.5477226  0.3333333  0.3333333   0.6666667  0.3333333
## Otu00074  0.5477226  0.6666667  0.6666667   0.3333333  0.6666667
## Otu00077  0.1825742  0.3333333  0.3333333   0.6666667  0.3333333
## Otu00088  0.1825742  0.0000000  0.0000000  -0.3333333  0.0000000
## Otu00089  0.1825742  0.0000000  0.0000000  -0.3333333  0.0000000
## Otu00097 -0.1825742  0.0000000  0.0000000   0.3333333  0.0000000
## Otu00100 -0.1825742  0.0000000  0.0000000   0.3333333  0.0000000
## Otu00113 -0.5477226 -0.6666667 -0.6666667  -0.3333333 -0.6666667
## Otu00192  0.5477226  0.6666667  0.6666667   1.0000000  0.6666667
## Otu00295  0.2581989  0.2357023  0.2357023   0.7071068  0.2357023
##            iVal.2MB   Valerate
## Otu00006 -0.1825742  0.1825742
## Otu00014 -0.3333333  0.0000000
## Otu00016 -0.3333333 -0.6666667
## Otu00018 -0.3333333 -0.6666667
## Otu00021 -0.6666667 -0.3333333
## Otu00025  0.6666667  0.3333333
## Otu00035 -0.6666667 -1.0000000
## Otu00036  0.0000000 -0.3333333
## Otu00037  0.0000000  0.3333333
## Otu00040 -0.6666667 -1.0000000
## Otu00042 -0.3333333  0.0000000
## Otu00046 -0.3333333 -0.6666667
## Otu00049  0.3333333  0.0000000
## Otu00051  1.0000000  0.6666667
## Otu00052 -0.6666667 -1.0000000
## Otu00056 -0.3333333 -0.6666667
## Otu00064 -1.0000000 -0.6666667
## Otu00066 -0.6666667 -1.0000000
## Otu00067  0.6666667  0.3333333
## Otu00069  1.0000000  0.6666667
## Otu00074  0.0000000  0.3333333
## Otu00077  0.3333333  0.6666667
## Otu00088  0.0000000 -0.3333333
## Otu00089  0.0000000 -0.3333333
## Otu00097  0.0000000  0.3333333
## Otu00100  0.0000000  0.3333333
## Otu00113  0.0000000 -0.3333333
## Otu00192  0.6666667  1.0000000
## Otu00295  0.7071068  0.7071068
```

If the data table is too large to view in R, you can write it to a table in your project folder.

```
write.table(cor.kendall, file = "cor_kendall.csv", sep = ",")
```

We see that some OTUs strongly correlation with a SCFAs. For example, Otu00021 and Otu00025 with Formate

```
par(mfrow = c(1, 2))
plot(abund.SCFA$Otu00021 ~ SCFA$Formate, xlab="Formate (mM)", ylab="Relative abundance, %", main="OTU21")
plot(abund.SCFA$Otu00025 ~ SCFA$Formate, xlab="Formate (mM)", ylab="Relative abundance, %", main="OTU25")
```



Clearly we don't have enough data points to make strong conclusions here and the correlations are being driven by one animal with very high formate. However, we could further test the list of OTUs that correlate strongly with SCFAs. We will assume a normal distribution here, but you should assess your models with plot() to make sure they are a good fit.

```
OTU21.Formate = glm(OTU.SCFA$Otu00021 ~ SCFA$Formate)
summary(OTU21.Formate)
```

```
##
## Call:
## glm(formula = OTU.SCFA$Otu00021 ~ SCFA$Formate)
##
## Deviance Residuals:
##       1        2        3        4
## -56.173   96.253  -46.747    6.668
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    357.75      51.46   6.952   0.0201 *
## SCFA$Formate  -540.02     201.13  -2.685   0.1152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 7324.907)
##
##     Null deviance: 67454  on 3  degrees of freedom
## Residual deviance: 14650  on 2  degrees of freedom
## AIC: 50.175
##
## Number of Fisher Scoring iterations: 2
```

```
OTU25.Formate = glm(OTU.SCFA$Otu00025 ~ SCFA$Formate)
summary(OTU25.Formate)
```

```
##
## Call:
## glm(formula = OTU.SCFA$Otu00025 ~ SCFA$Formate)
##
## Deviance Residuals:
##        1         2         3         4
##  127.727  -118.783     6.217   -15.162
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    219.78      74.49   2.951   0.0982 .
## SCFA$Formate   721.00     291.12   2.477   0.1316
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 15346.04)
##
##     Null deviance: 124819  on 3  degrees of freedom
## Residual deviance:  30692  on 2  degrees of freedom
## AIC: 53.133
##
## Number of Fisher Scoring iterations: 2
```

So, we see that these two OTUs do not significantly differ with Formate concentration even though they had very strong Kendall correlations. This is similar to OTUs occuring in SIMPER that do not hold up to subsequent Kruskal-Wallis testing.

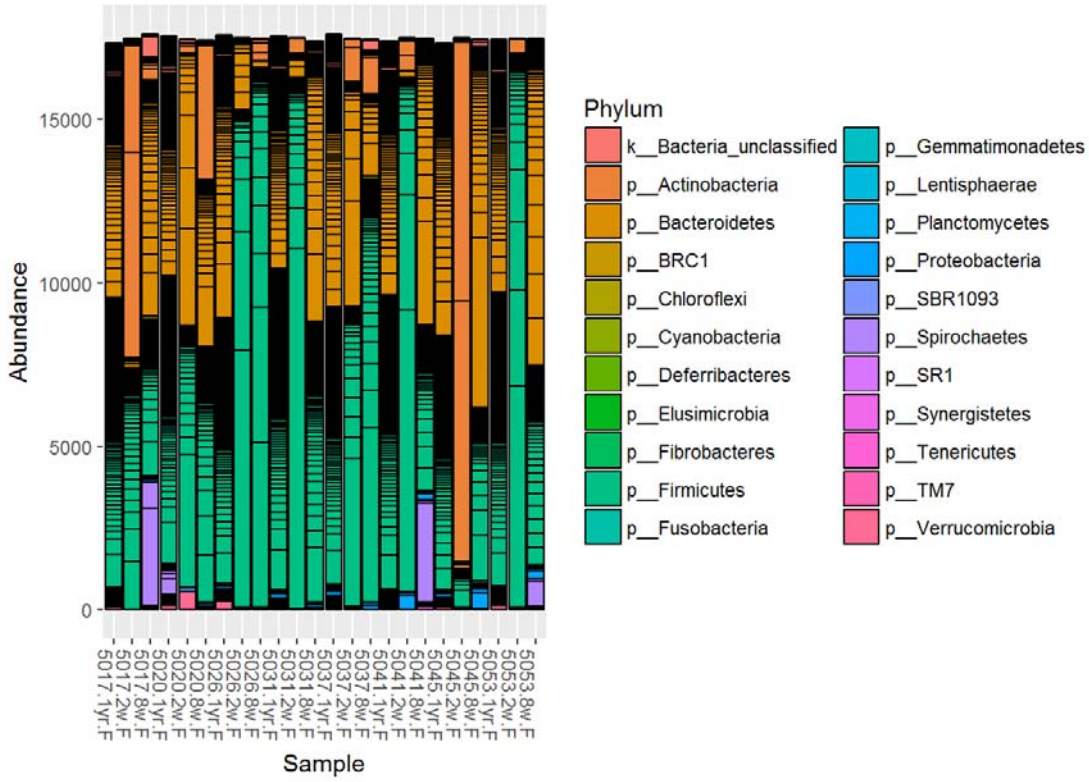
Other visualizations

Bar charts

The phyloseq object we created with our OTU, meta, tax, and tree data (physeq.tree) can also be used in a number of other plot functions in the `phyloseq` / `ggplot2` packages.

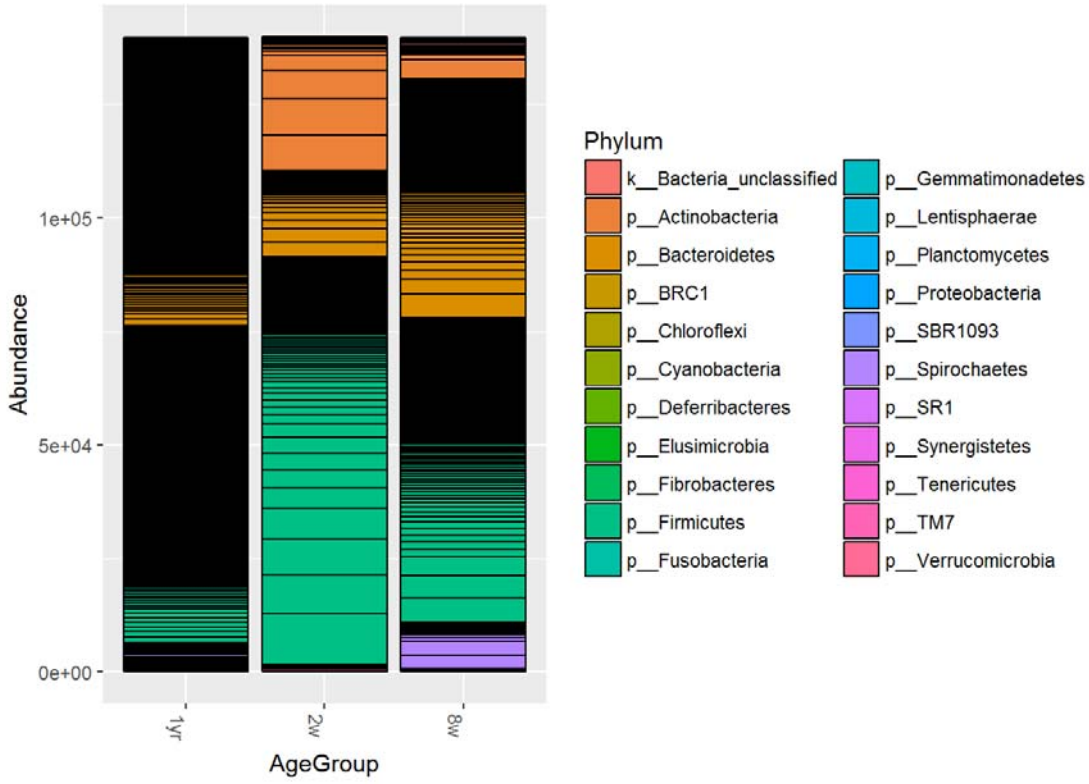
Let's explore some of the bar chart options. First, we'll make the classic additive bar chart for phyla in our samples

```
plot_bar(physeq.tree, fill="Phylum")
```



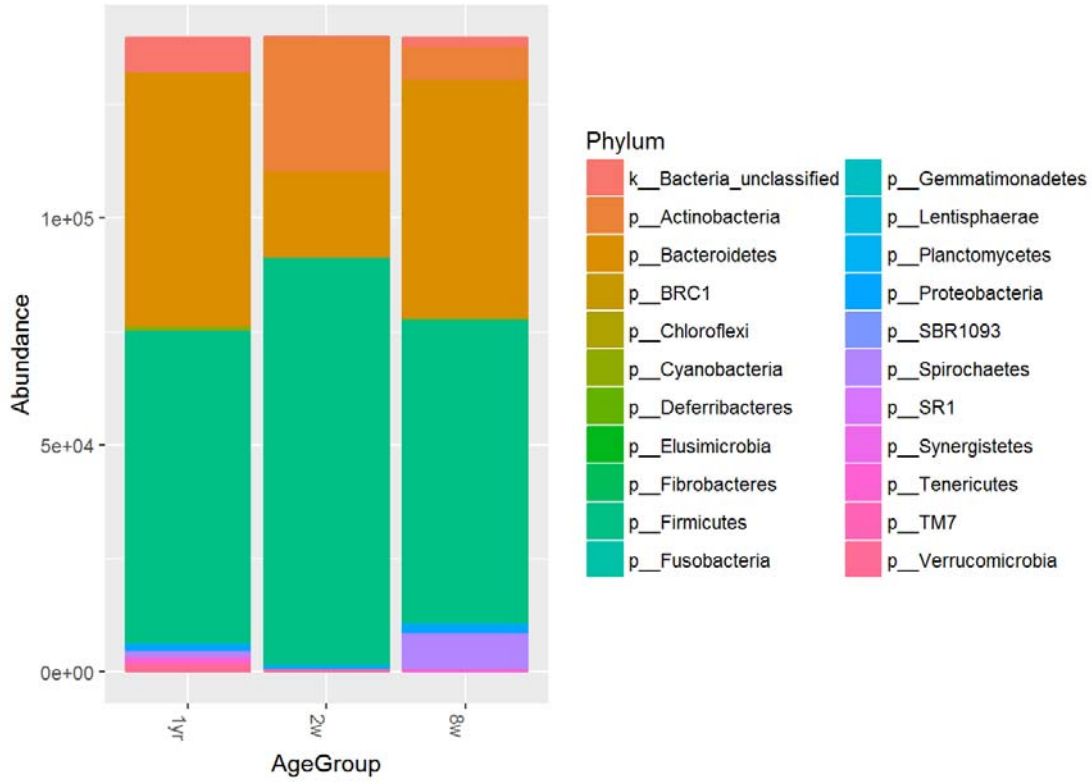
We can simplify by grouping our samples by age group

```
plot_bar(physeq.tree, x="AgeGroup", fill="Phylum")
```



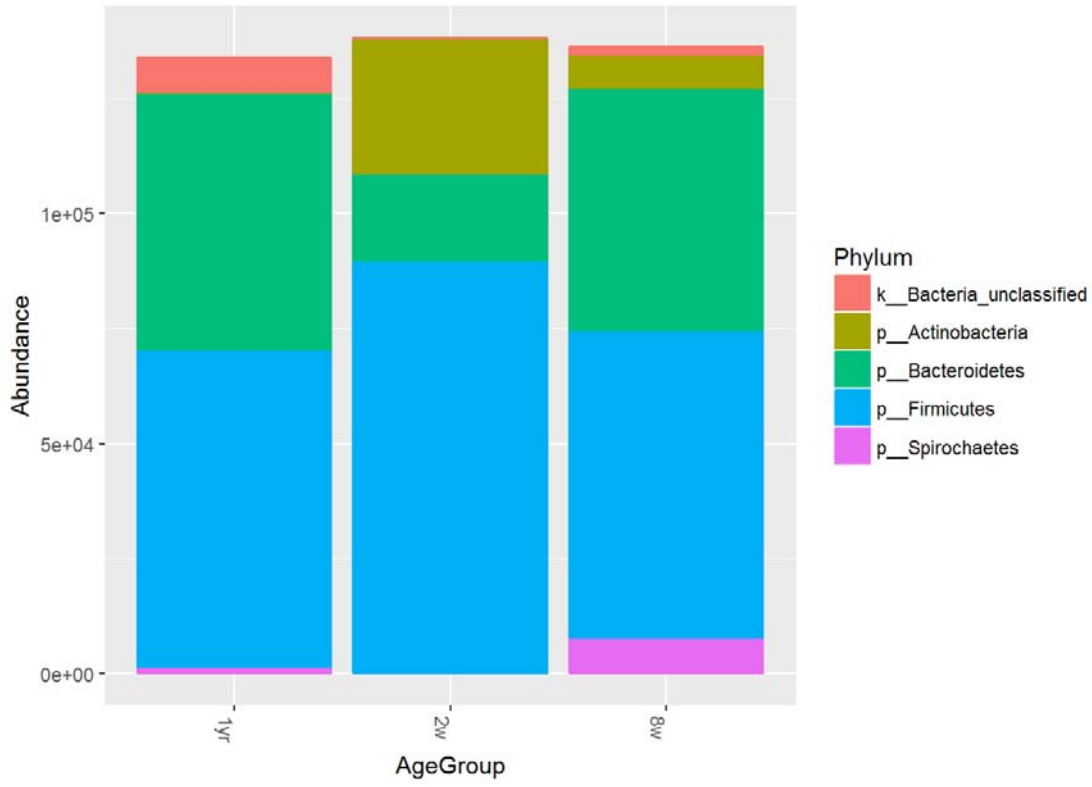
And removing the lines between OTUs in the bars

```
plot_bar(physeq.tree, x="AgeGroup", fill="Phylum") + geom_bar(aes(color=Phylum, fill=Phylum), stat="identit
y", position="stack")
```



And only showing the top 5 most abundant phyla

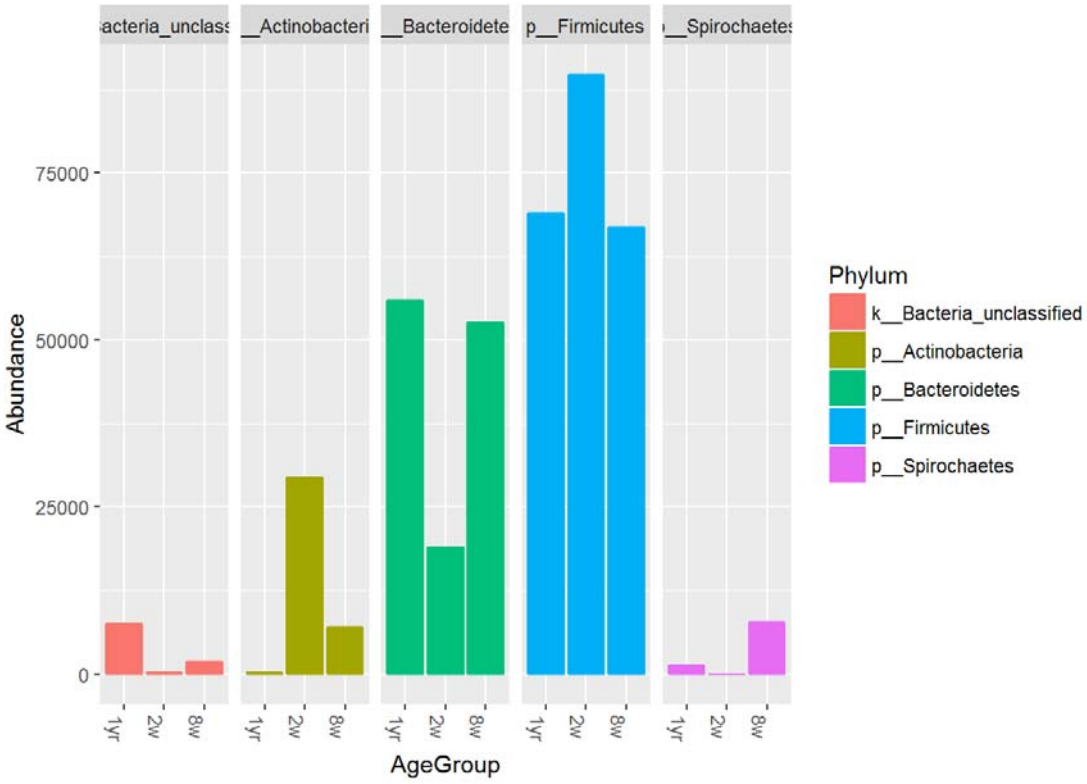
```
#Sort the Phyla by abundance and pick the top 5
top5P.names = sort(tapply(taxa_sums(physeq.tree), tax_table(physeq.tree)[, "Phylum"], sum), TRUE)[1:5]
#Cut down the physeq.tree data to only the top 10 Phyla
top5P = subset_taxa(physeq.tree, Phylum %in% names(top5P.names))
#Plot
plot_bar(top5P, x="AgeGroup", fill="Phylum") + geom_bar(aes(color=Phylum, fill=Phylum), stat="identity", pos
ition="stack")
```



There are many more options within `ggplot2` to alter this figure. This document (https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf) has many helpful tips.

Another way to simplify these bar plots is to not show all OTUs for one sample in one bar. We can do this with facet_grid

```
plot_bar(top5P, x="AgeGroup", fill="Phylum", facet_grid = ~Phylum) + geom_bar(aes(color=Phylum,
fill=Phylum), stat="identity", position="stack")
```



And you can break it down at any taxonomic level and color by any other level.

Trees

We can also plot phylogenetic trees and label/modify them by our variables of interest.

Let's look at the genus *Prevotella* in our data. We want to subset down to just this genus or else our plot would be too cluttered to read.

Subset by genus

```
prevotella = subset_taxa(physeq.tree, Genus == "g__Prevotella")
```

We can see that this worked by comparing the number of taxa in our subset and our original data

```
physeq.tree
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 5002 taxa and 24 samples ]
## sample_data() Sample Data:       [ 24 samples by 9 sample variables ]
## tax_table()   Taxonomy Table:    [ 5002 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 5002 tips and 5000 internal nodes ]
```

```
prevotella
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 106 taxa and 24 samples ]
## sample_data() Sample Data:       [ 24 samples by 9 sample variables ]
## tax_table()   Taxonomy Table:    [ 106 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 106 tips and 105 internal nodes ]
```

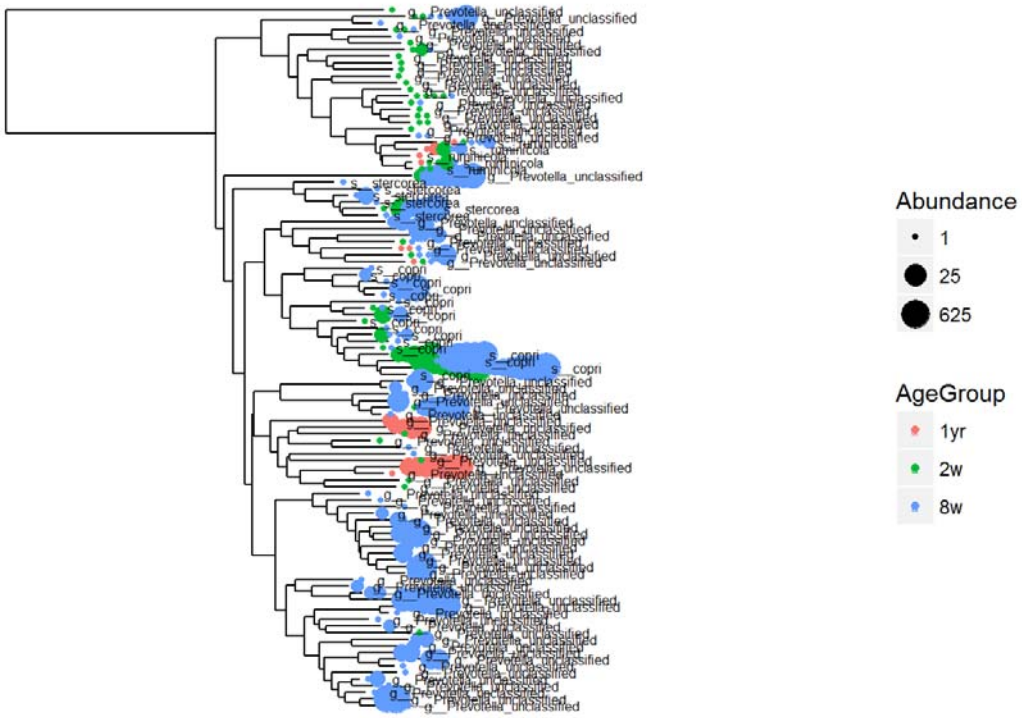
We can plot these OTUs on a tree.

```
plot_tree(prevotella, plot.margin = 0.5, ladderize = TRUE)
```

In the figure, each OTU is represented by the end branch of the tree. How many samples that OTU occurs in is represented by the black dots.

Let's make this figure a little more useful and add 1) Colors to the dots for our age groups, 2) Size to the dots to show OTU abundance, and 3) Species level labels for the OTUs

```
plot_tree(prevotella, color = "AgeGroup", label.tips = "Species", size = "abundance", plot.margin = 0.5, lad
derize = TRUE)
```



Already it's a little difficult to read. You can view a larger page by clicking "Zoom" above the figure. Or export the figure as a PDF and save as a full page size, 9.5x11.

There are even more customizable options in this figure. Type ?plot_tree into the console to see the help page explaining all the options.

Heat maps

There are some good options in both `phyloseq` and `gplots` to make heatmaps. We will go through `phyloseq` but know that the same things could be done in `gplots` with code specific to that package.

OTUs

We're going to just look at the 20 most abundant OTUs to make it more readable.

```
#Sort the OTUs by abundance and pick the top 20
top20OTU.names = names(sort(taxa_sums(physeq.tree), TRUE)[1:20])
#Cut down the physeq.tree data to only the top 10 Phyla
top20OTU = prune_taxa(top20OTU.names, physeq.tree)
```

We now see that we only have 20 taxa

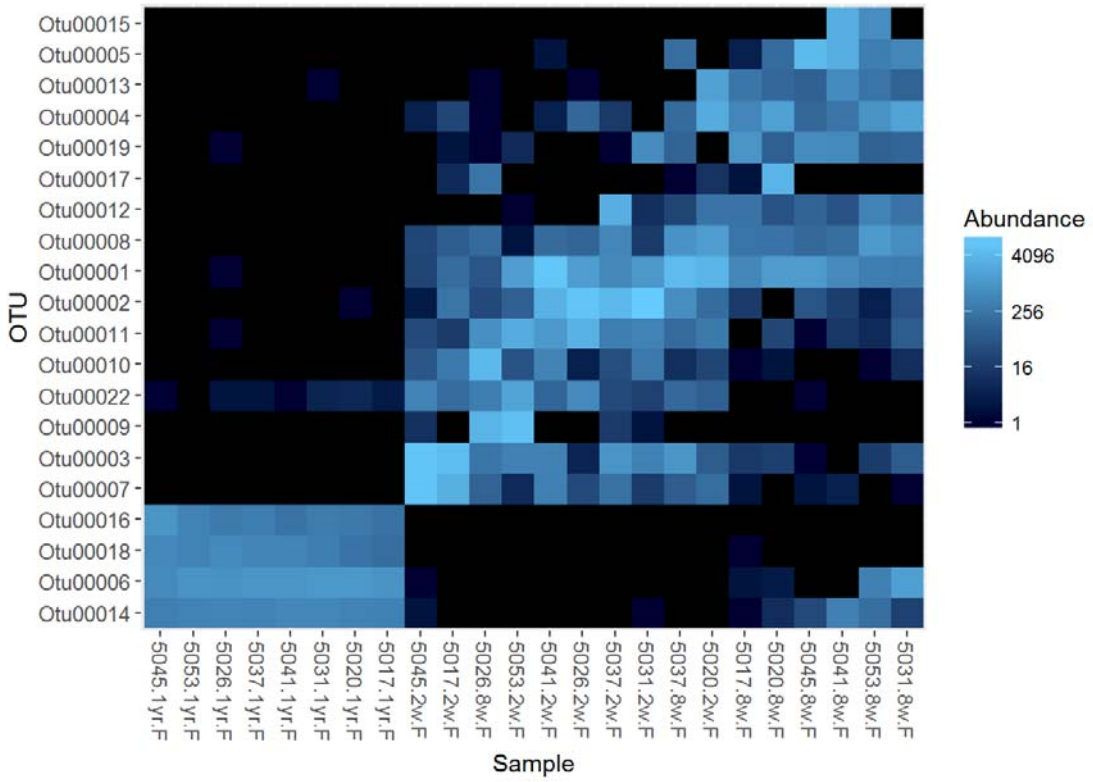
```
top20OTU
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 20 taxa and 24 samples ]
## sample_data() Sample Data:       [ 24 samples by 9 sample variables ]
## tax_table()   Taxonomy Table:    [ 20 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 20 tips and 19 internal nodes ]
```

First, you can make a heatmap of OTU abundance across all samples

```
plot_heatmap(top20OTU)
```

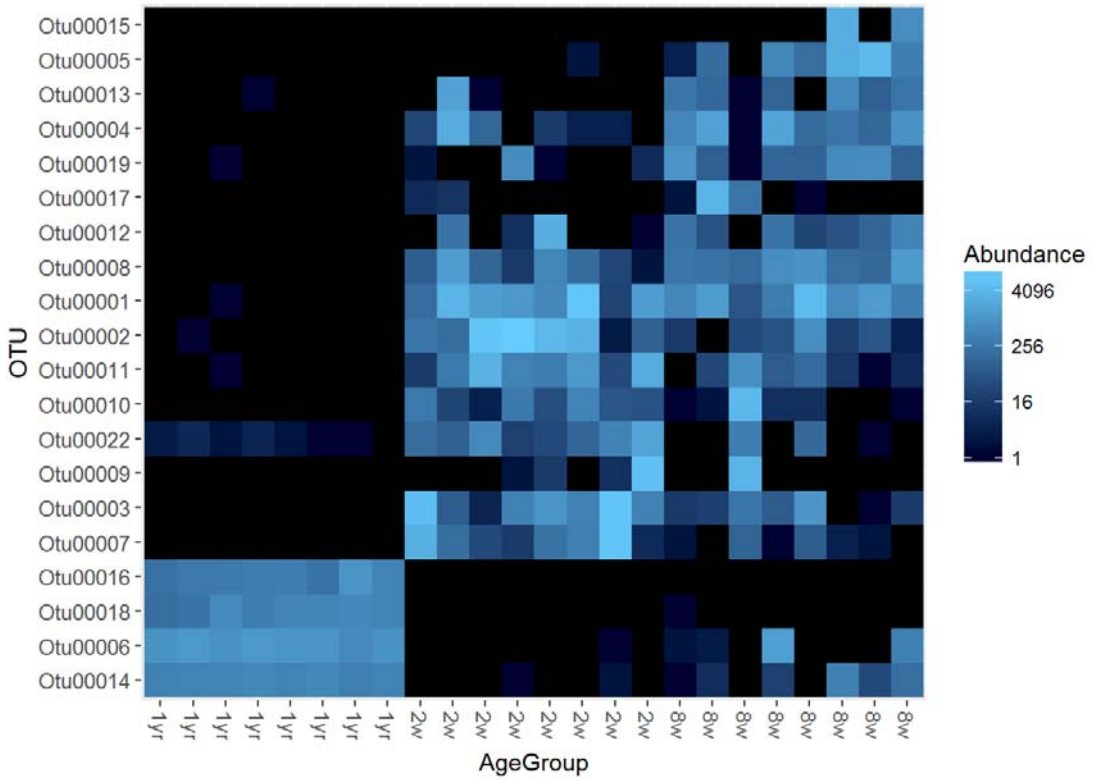
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



And grouped by our age groups

```
plot_heatmap(top20OTU, sample.label="AgeGroup", sample.order="AgeGroup")
```

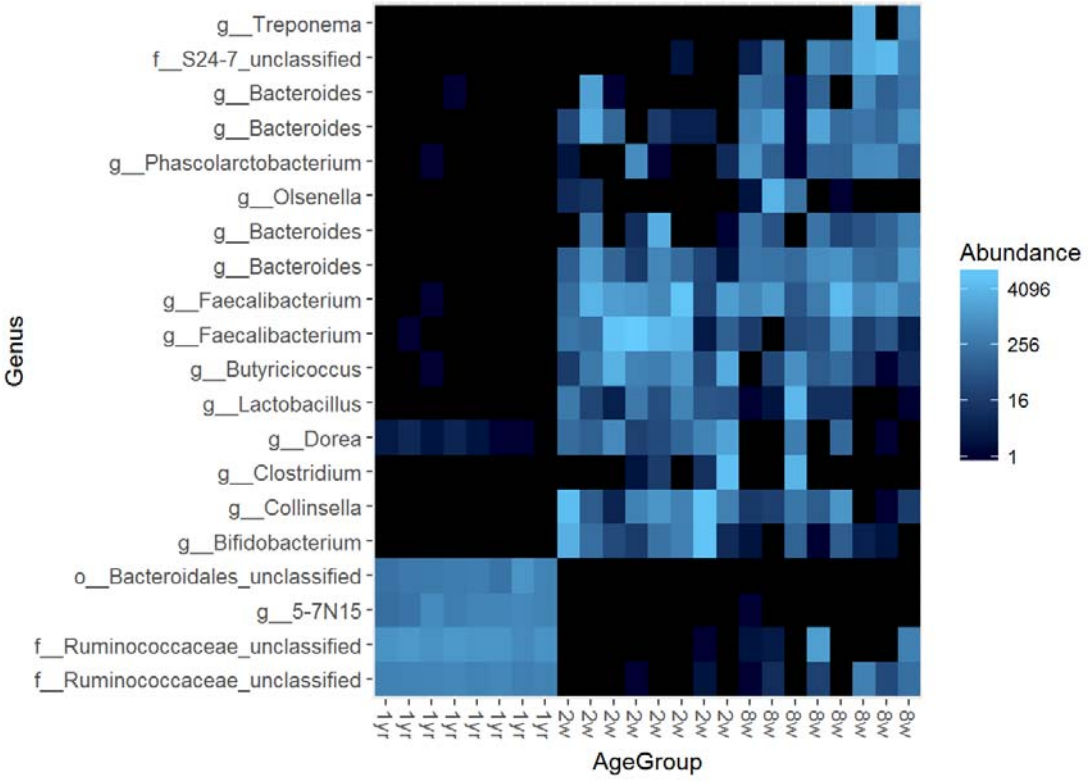
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



We can label the OTU taxa

```
plot_heatmap(top20OTU, sample.label="AgeGroup", sample.order="AgeGroup", taxa.label="Genus")
```

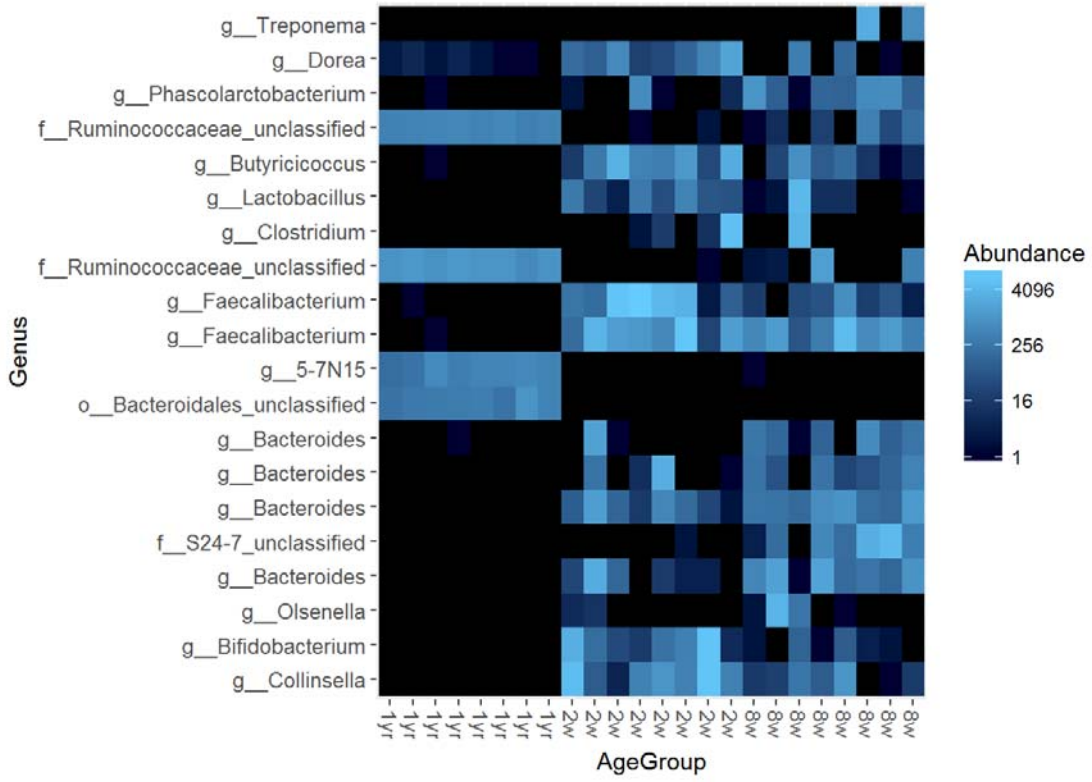
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



And group OTUs within the same Phyla

```
plot_heatmap(top20OTU, sample.label="AgeGroup", sample.order="AgeGroup", taxa.label="Genus", taxa.order="Phy
lum")
```

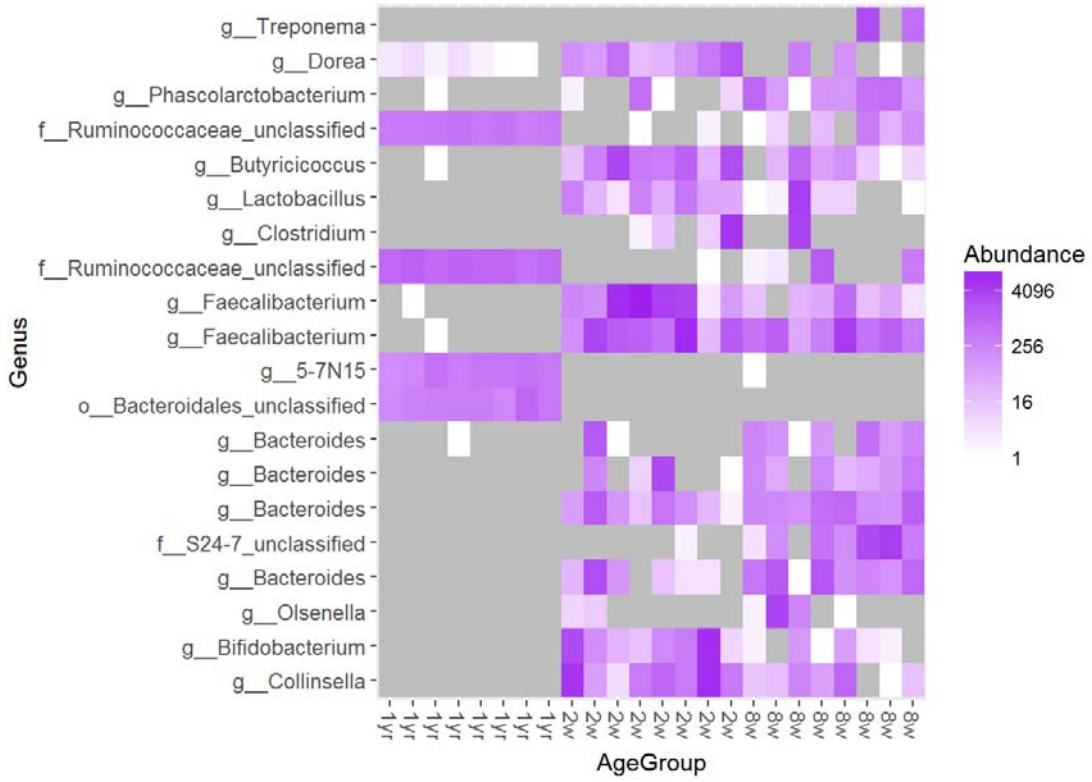
```
## Warning: Transformation introduced infinite values in discrete y-axis
```



We can also change the colors (white -> purple), including the 0s/NAs (grey).

```
plot_heatmap(top20OTU, sample.label="AgeGroup", sample.order="AgeGroup", taxa.label="Genus", taxa.order="Phy
lum", low="white", high="purple", na.value="grey")
```

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

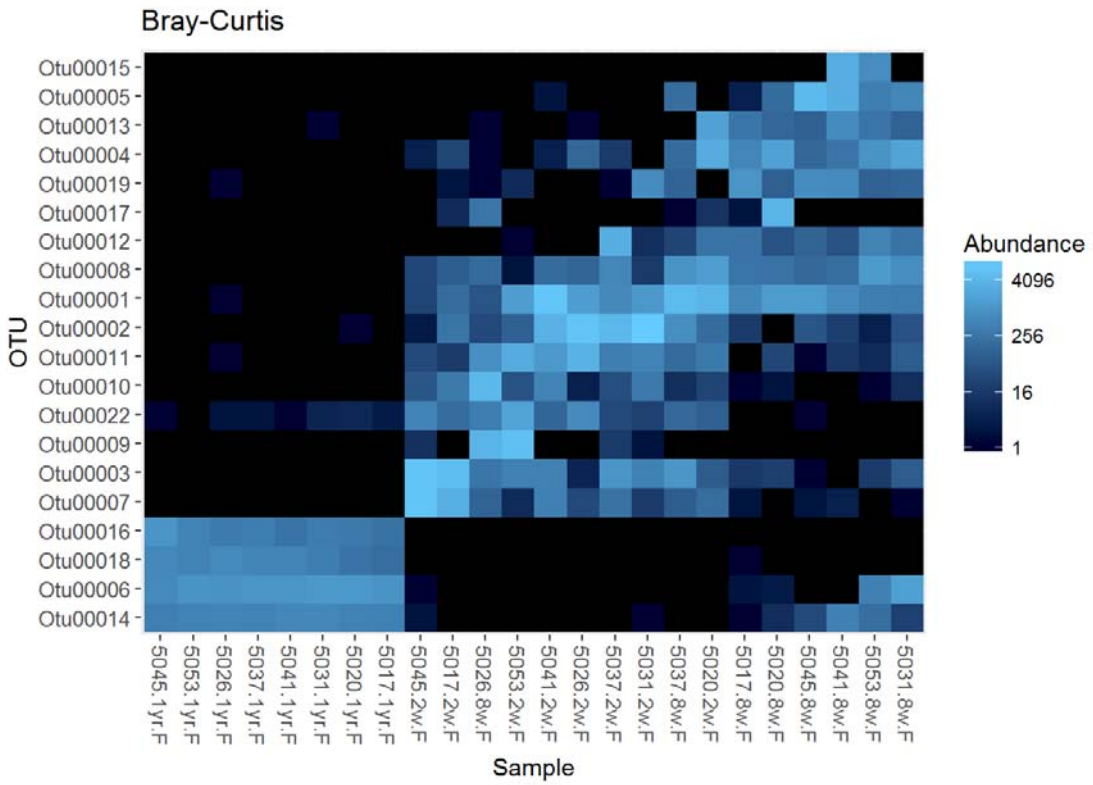


You can also have R automatically group your OTUs and samples by beta-diversity. This may yield the most easily interpreted heatmap but if you have a specific research question that is better addressed by your own ordering (like our age groups above), you should stick with that. We'll show Bray-Curtis as an example. Other options are

- bray
- jaccard
- wunifrac
- uwunifrac

```
plot_heatmap(top20OTU, "NMDS", "bray", title="Bray-Curtis")
```

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

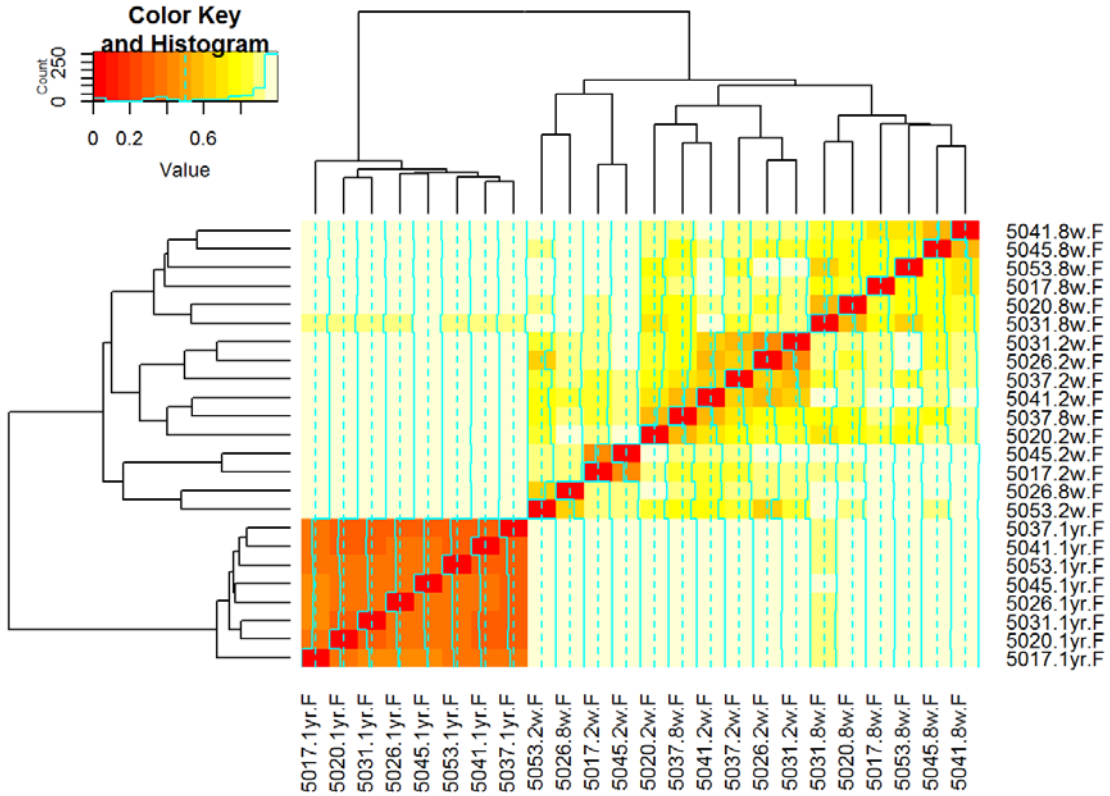


Beta-diversity

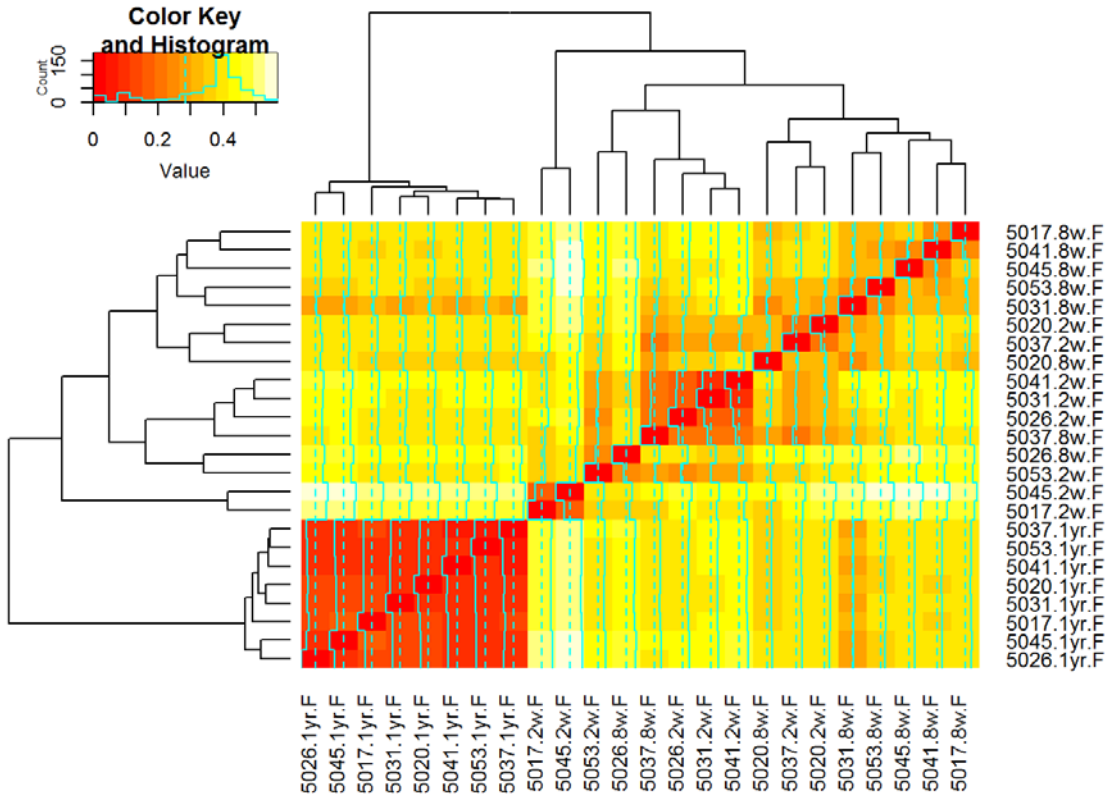
The other common use for heatmaps is to show distances between samples (*i.e.* beta-diversity) similar to what is shown in nMDS. We have all of the same metric options as we did for nMDS.

We do not want to use the plot_heatmap() function from `phyloseq` because it requires the input of a physeq object. Instead, we can use our distance matrices as inputs for a `gplots` command. This command will automatically group samples by similarity (trees)

```
#Bray-Curtis
heatmap.2(as.matrix(BC.dist))
```

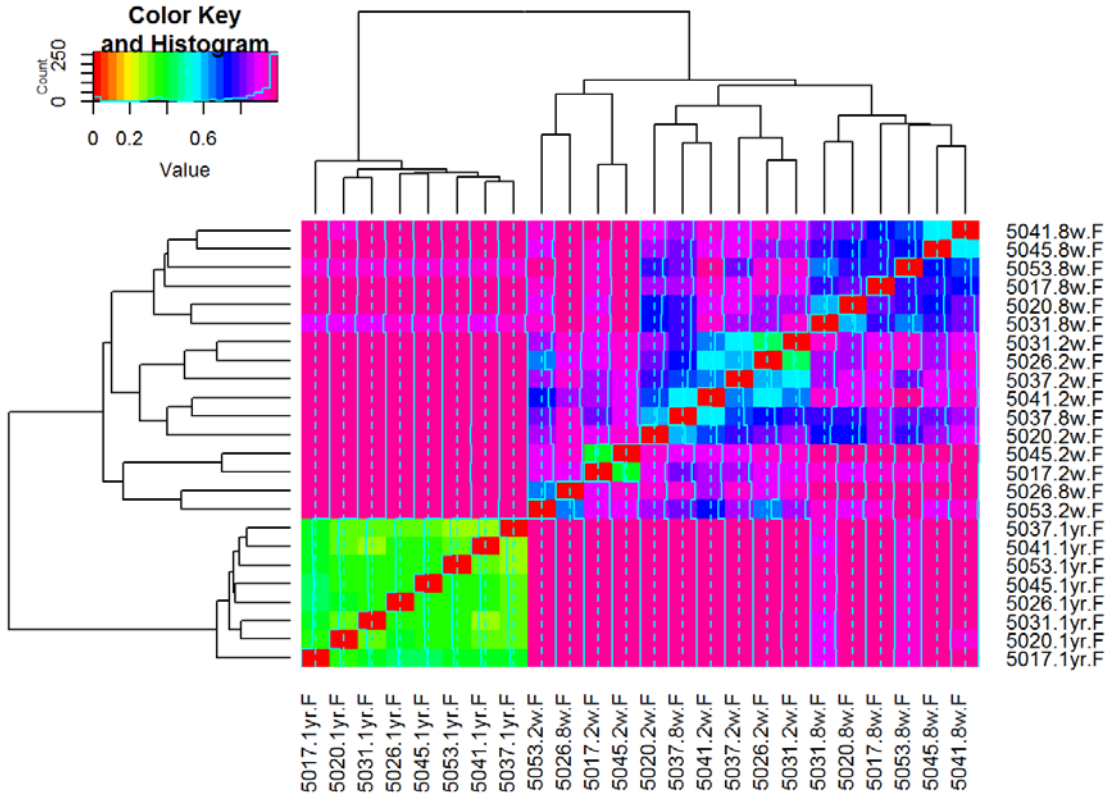



```
#UniFrac
heatmap.2(as.matrix(wUF.dist))
```



You could also change the colors

```
#Rainbow colors
rc <- rainbow(nrow(as.matrix(BC.dist)), start=0, end=0.9)
heatmap.2(as.matrix(BC.dist), col=rc)
```



As always, for further customization, explore with ?heatmap.2

Venn diagrams

Venn diagram of three samples: 5017.2w.F, 5017.8w.F, and 5017.1yr.F

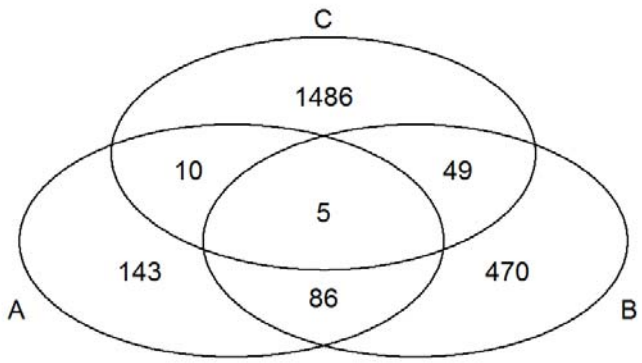
Create a list of OTUs that occur (count > 0) in each sample.

- We select for the row by name with *OTU.clean["name",]*
- We select the columns with a value >0 with *OTU.clean[,apply()]*

```
OTU.5017.2w = colnames(OTU.clean["5017.2w.F", apply(OTU.clean["5017.2w.F",], MARGIN=2, function(x) any(x
>0))])

OTU.5017.8w = colnames(OTU.clean["5017.8w.F", apply(OTU.clean["5017.8w.F",], MARGIN=2, function(x) any(x
>0))])

OTU.5017.1yr = colnames(OTU.clean["5017.1yr.F",apply(OTU.clean["5017.1yr.F",], MARGIN=2, function(x) any(x
>0))])
```

We can then use these lists of OTUs to plot a Venn diagram with venn() from the `gplots` package

```
venn(list(OTU.5017.2w, OTU.5017.8w, OTU.5017.1yr))
```

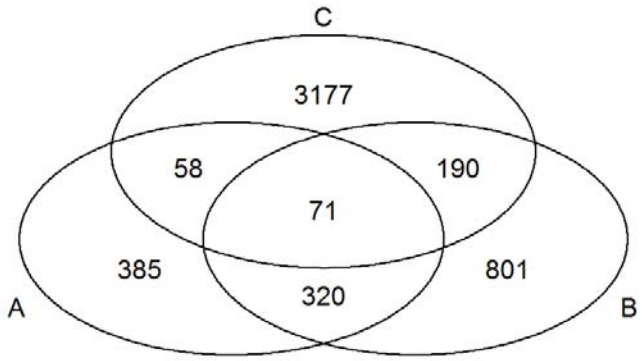


We can also do this for our age groups by selecting all samples where meta$AgeGroup = 2w, 8w, or 1yr

```
OTU.2w = colnames(OTU.clean[meta$AgeGroup == "2w", apply(OTU.clean[meta$AgeGroup == "2w",], MARGIN=2, functi
on(x) any(x >0))])

OTU.8w = colnames(OTU.clean[meta$AgeGroup == "8w", apply(OTU.clean[meta$AgeGroup == "8w",], MARGIN=2, functi
on(x) any(x >0))])

OTU.1yr = colnames(OTU.clean[meta$AgeGroup == "1yr", apply(OTU.clean[meta$AgeGroup == "1yr",], MARGIN=2, fun
ction(x) any(x >0))])
```

And plot

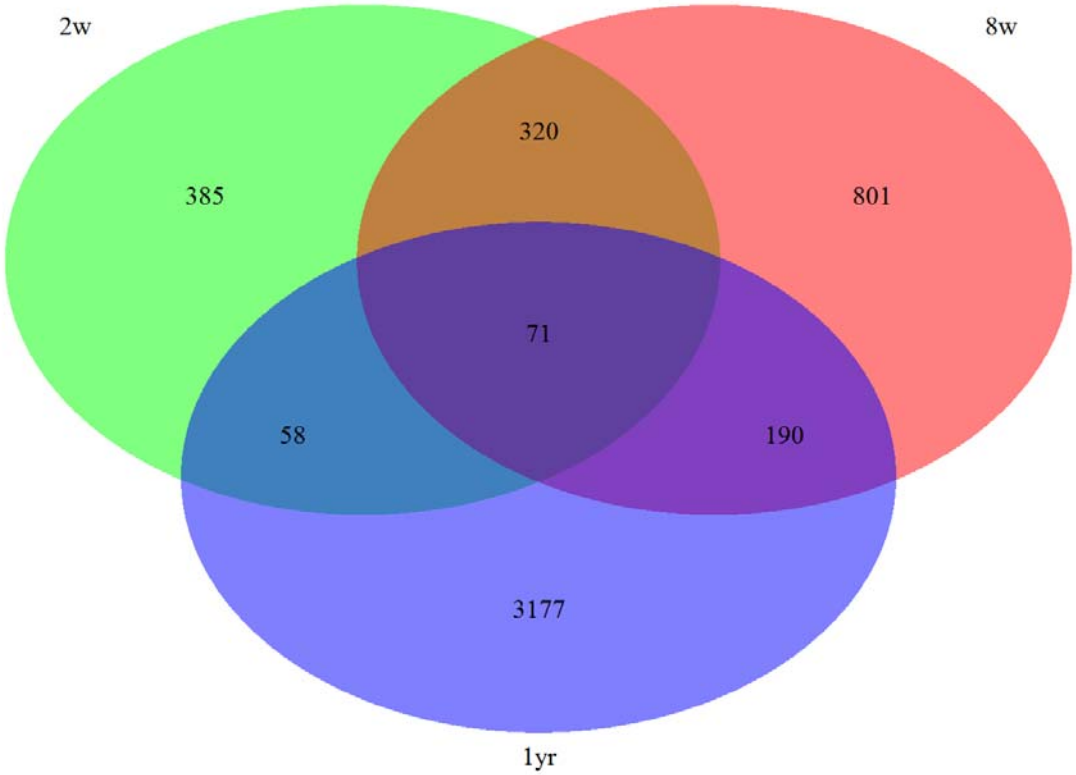
```
venn(list(OTU.2w, OTU.8w, OTU.1yr))
```



These are not the prettiest Venns, but they are the quickest way to calculate the values within a Venn.

Once you have these, you can use the VennDiagram package for more pretty graphing options. For example, the age groups venns would be

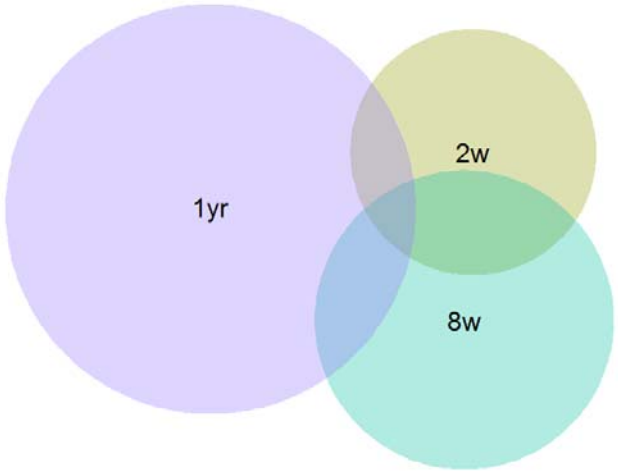
```
draw.triple.venn(area1 = 385+58+71+320, area2 = 801+190+320+71, area3 = 3177+190+58+71, n12 = 320+71, n23 =
190+71, n13 = 58+71, n123 = 71, category = c("2w", "8w", "1yr"), lty = "blank", fill = c("green", "red", "bl
ue"))
```



```
## (polygon[GRID.polygon.1343], polygon[GRID.polygon.1344], polygon[GRID.polygon.1345], polygon[GRID.polygo
n.1346], polygon[GRID.polygon.1347], polygon[GRID.polygon.1348], text[GRID.text.1349], text[GRID.text.1350],
text[GRID.text.1351], text[GRID.text.1352], text[GRID.text.1353], text[GRID.text.1354], text[GRID.text.135
5], text[GRID.text.1356], text[GRID.text.1357], text[GRID.text.1358])
```

Or with venneuler, you can scale the circles to be porportional to the total number of OTUs in that group

```
#Create a venneuler object
age.venn=venneuler(c('A' = 385+58+71+320, 'B' = 801+190+320+71, 'C' = 3177+190+58+71, 'A&B' = 320+71, 'B&C'
= 190+71, 'A&C' = 58+71, 'A&B&C' = 71))

#Add group names
age.venn$labels = c("2w", "8w", "1yr")

#Plot
plot(age.venn)
```



Or we can export the OTU lists and make Venns with this online tool http://bioinformatics.psb.ugent.be/webtools/Venn/ (http://bioinformatics.psb.ugent.be/webtools/Venn/). This tool is handy in that is gives you the list of OTUs within the Venn sections so that you can see which specific bacteria are shared.

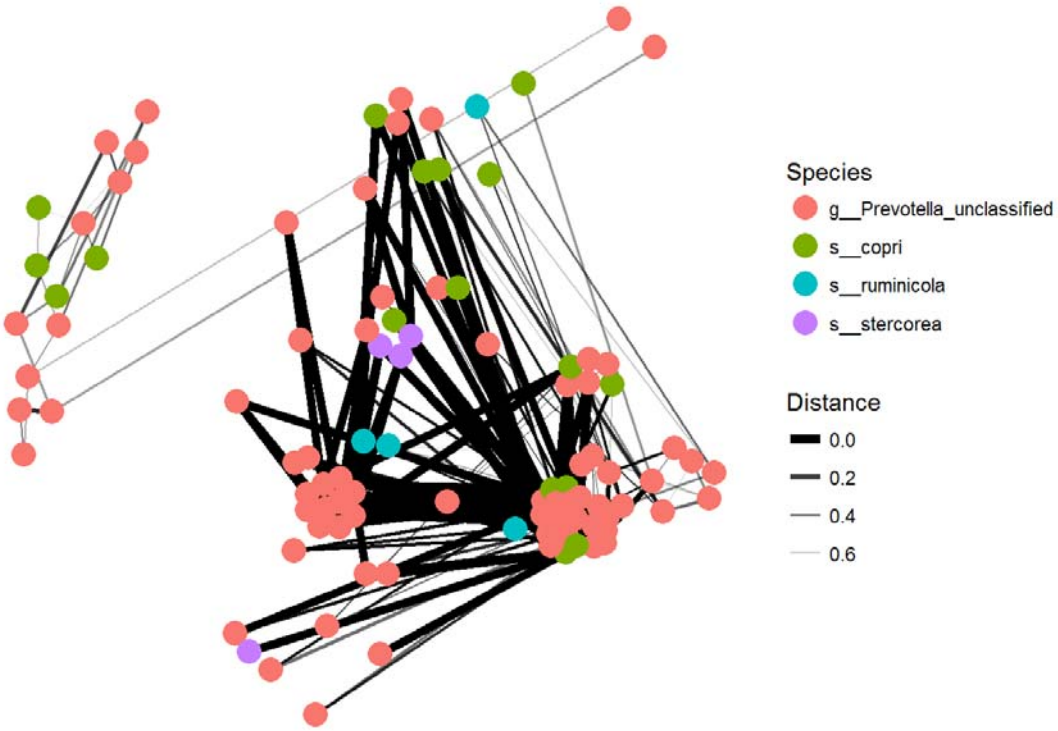
```
write.table(OTU.2w, "OTU.2w.csv", sep=",", row.names=FALSE, col.names=FALSE)
write.table(OTU.8w, "OTU.8w.csv", sep=",", row.names=FALSE, col.names=FALSE)
write.table(OTU.1yr, "OTU.1yr.csv", sep=",", row.names=FALSE, col.names=FALSE)
```

Networks

OTUs

You can plot the distances between OTUs as a network. It would be an unreadable mess to plot all the OTUs in our data set, so we will just use the smaller prevotella data set.

```
plot_net(prevotella, color="Species", type="taxa")
```



For co-occurrence networks of OTUs, I recommend Gephi (https://gephi.org/) or Cytoscape (http://www.cytoscape.org/). Thus far, I have not found an R package comparable to these other programs.

Beta-diversity

You can also plot beta-diversity as a network where the edges (lines) are the distances between samples. All metrics we've used here are supported (bray, jaccard, wunifrac, uwunifrac)

```
plot_net(physeq.tree, color="AgeGroup", distance="bray")
```