Example Quarto Site for R

Paul Villanueva

8/13/2021

Table of Contents

# 1 Intro

This is an example Quarto project made for the Reproducible Research with R workshop at the R/Medicine conference on 8/25/2021.

This is a Quarto book project. A book project is just like a site, so it uses the same styling and structure as the site, but it has a few extra things. Most notably, you’re able to download the entire site as a single document by clicking the download button in the upper right hand corner.

# 2 A Basic Quarto Doc

This is an example Quarto document. Note the qmd extension - this tells Quarto that this is a Markdown files that contains computations.

Since Quarto based on Markdown, we can **bold** and *italicize* text. We can also make headers.

# 3 Let’s make a table

| Meal | Food |
| --- | --- |
| Breakfast | Coffee |
| Lunch | Leftovers |
| Dinner | Spam Musubi |

# 4 Example images

|  |
| --- |
| My support system  My support system |

|  |  |
| --- | --- |
| Lunch: Leftovers  Lunch: Leftovers | Dinner: Spam musubi  Dinner: Spam musubi |

# 5 Code and chunk options

Quarto is based on **R**Markdown, so you can do all the R stuff you’re used to as well.

library(tidyverse)  
  
standard\_curves <- readxl::read\_xlsx('data/std\_curve.xlsx', sheet = "everything") %>%   
 janitor::clean\_names() %>%   
 filter(amoa < 40)  
  
lm\_eqn = function(df){  
 m = lm(log\_qty ~ ct, df);  
 data.frame(  
 a = format(as.numeric(coef(m)[1]), digits = 2),  
 b = format(as.numeric(coef(m)[2]), digits = 2),  
 r2 = format(summary(m)$r.squared, digits = 3)  
 )  
}  
  
st\_splits <- standard\_curves %>%   
 group\_by(amoa, run) %>%   
 group\_split()   
  
eqs <- st\_splits %>%   
 lapply(., lm\_eqn) %>%   
 bind\_rows()  
  
labels <- lapply(st\_splits, slice\_head, n = 1) %>%   
 bind\_rows() %>%   
 select(amoa, run) %>%   
 bind\_cols(eqs) %>%   
 mutate(amoa = paste0("amoA\_AOB\_p", amoa)) %>%  
 mutate(eq\_label = paste0("y = ", a, " - ", abs(as.numeric(b)), "x<br>r^2 = ", r2))  
  
  
standard\_curves %>%   
 mutate(amoa = paste0("amoA\_AOB\_p", amoa)) %>%   
 ggplot(aes(log\_qty, ct)) +  
 geom\_point() +  
 facet\_grid(run ~ amoa, scales = "free") +   
 theme(  
 panel.border = element\_rect(color = "black", size = 1, fill = NA),  
 panel.grid.minor.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.major.x = element\_line(color = "gray", size = 0.5, linetype = "dashed"),  
 panel.grid.major.y = element\_line(color = "gray", size = 0.5, linetype = "dashed"),  
 panel.spacing = unit(0.5, "lines"),  
 panel.background = element\_blank(),  
 strip.background = element\_rect(color = "black", size = 1, fill = NA),  
 ) +   
 labs(  
 x = "Log(gene copies per reaction)",  
 y = "Ct"  
 ) +   
 scale\_x\_continuous(limits = c(0, 7), breaks = seq(0, 7, 1), expand = c(0, 0)) +   
 scale\_y\_continuous(limits = c(0, 25)) +  
 geom\_smooth(aes(group=1), method="lm", se=FALSE) +   
 ggtext::geom\_richtext(data = labels, aes(x = 3, y = 5, label = eq\_label),  
 size = 4, fontface = "bold", inherit.aes = FALSE)

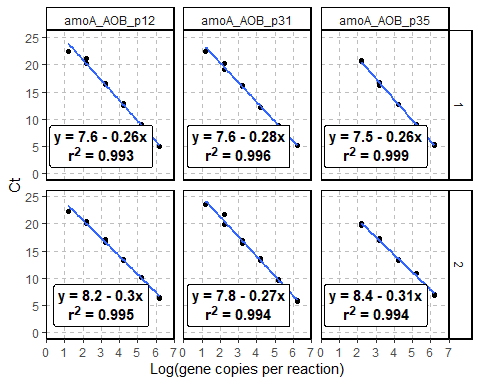


Figure 5.1: Standard curves for the LAMPS crop priming experiment

We can also throw some Python in here:

xs = [x for x in range(10)]  
  
print(\*(f'{x} squared is {x ^ 2}.' for x in xs), sep='\n')

0 squared is 2.  
1 squared is 3.  
2 squared is 0.  
3 squared is 1.  
4 squared is 6.  
5 squared is 7.  
6 squared is 4.  
7 squared is 5.  
8 squared is 10.  
9 squared is 11.

import matplotlib.pyplot as plt  
import numpy as np  
  
Z = np.random.rand(6, 10)  
x = [x + 0.5 for x in xs]  
y = np.arange(4.5, 11, 1)   
  
fig, ax = plt.subplots();  
ax.pcolormesh(x, y, Z)

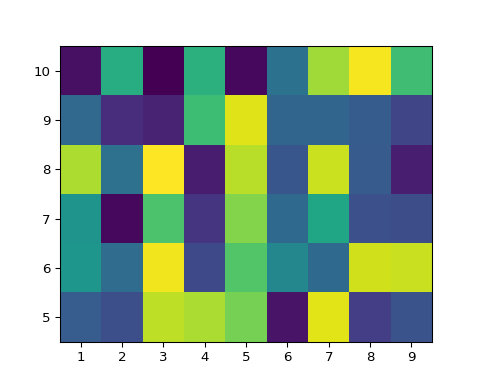


Figure 5.2: That’s a heatmap, baby!

## 5.1 Math stuff

We can also write math stuff! For example, here is a definition:

**Definition 5.1 (Continuity)** The function is *continuous at a point* if for all there exists such that if , then . If this is true for all such in the domain of , we say that is a *continuous function.*

#### 5.1.0.1 Example

Define by:

Prove that is not a continuous function.

**Proof.** Let and choose any . For any , we can find some such that since the irrationals are dense in . But then , showing that is not continuous at .

## 5.2 Adding references

We can also add references. For instance, the following definition of -partially colored comes from this paper: (Blair et al. 2020)

**Definition 5.2 (-partially colored)** Let be a diagram of a link with crossings. We call -*partially colored* if we have specified a subset of the strands of and a function . We refer to this partial coloring by the tuple . Given -partial colorings and of , we say is the result of a coloring move on if

1. and for some strand in ;
2. ;
3. is adjacent to at some crossing , and ;
4. the over-strand at is an element of ;
5. .

## 5.3 Cross-references

Along the way, we’ve been giving each of the items above labels. The Visual Editor knows about these labels and we can call them up for cross referencing. For example:

* We were pretty happy about the standard curves in [fig. 5.1](#fig-std-curves)
* I love me some heatmaps like [fig. 5.2](#fig-tiles)
* [def. 5.2](#def-merid-coloring) is trivially true for the unknot.

# 6 Bibliography

# 7 A Pure Python Doc

This is a pure Python qmd document. Since there are no R code chunks, it is executed via the Jupyter kernel. Because it’s written in a qmd file, all of the Rmarkdown/knitr functionality you’re used to can be used here. Here is some example code below.

## 7.1 Adding days per month from date range to a dataframe

Suppose you have a dataset with a column of start dates and column of end dates. For example:

import pandas as pd  
import calendar  
  
date\_df = pd.DataFrame({  
 "START\_TM": ['2/15/2010', '2/15/2010', '3/16/2010'],  
 "END\_TM": ['4/18/2010', '2/18/2010', '5/20/2010']  
})  
date\_df["START\_TM"] = date\_df["START\_TM"].astype('datetime64')  
date\_df["END\_TM"] = date\_df["END\_TM"].astype('datetime64')  
date\_df

START\_TM END\_TM  
0 2010-02-15 2010-04-18  
1 2010-02-15 2010-02-18  
2 2010-03-16 2010-05-20

Our goal is to count the number of days in each month this range of dates falls over.

We start by adding columns for each month:

months = {calendar.month\_name[i]:[0 for \_ in range(date\_df.shape[0])] for i in range(1, 13)}  
for m in months:  
 date\_df[m] = [0 for \_ in range(date\_df.shape[0])]  
date\_df

START\_TM END\_TM January February March April May June July \  
0 2010-02-15 2010-04-18 0 0 0 0 0 0 0   
1 2010-02-15 2010-02-18 0 0 0 0 0 0 0   
2 2010-03-16 2010-05-20 0 0 0 0 0 0 0   
  
 August September October November December   
0 0 0 0 0 0   
1 0 0 0 0 0   
2 0 0 0 0 0

### 7.1.1 Helper functions

def insert\_days\_per\_month(outer\_row):  
 dpm = days\_per\_month(outer\_row)  
 for index, inner\_row in dpm.iterrows():  
 outer\_row[inner\_row['Month']] = inner\_row['NumDays']  
 return(outer\_row)  
  
def days\_per\_month(row):  
 s = pd.Series(index = pd.date\_range(row[0], row[1]))[1: ]   
 days\_in\_month = s.resample('MS').size().to\_period('m').\  
 rename\_axis('Month').reset\_index(name = 'NumDays')  
 days\_in\_month['Month'] = days\_in\_month['Month'].apply(  
 lambda x: calendar.month\_name[x.month])  
 return(days\_in\_month)

We can get the desired result with apply:

date\_df = date\_df.apply(lambda x: insert\_days\_per\_month(x), axis = 1)  
date\_df

DeprecationWarning:  
  
The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.

START\_TM END\_TM January February March April May June July \  
0 2010-02-15 2010-04-18 0 13 31 18 0 0 0   
1 2010-02-15 2010-02-18 0 3 0 0 0 0 0   
2 2010-03-16 2010-05-20 0 0 15 30 20 0 0   
  
 August September October November December   
0 0 0 0 0 0   
1 0 0 0 0 0   
2 0 0 0 0 0

## 7.2 Interactive Widgets

Quarto has support for interactive documents. Supported formats include:

* JavaScript: Observable JS
* R: Shiny
* Python: Jupyter Widgets are all supported, such as IPyLeaflet and Plotly

Here are some examples:

## 7.3 IPyLeaflet

from ipyleaflet import Map, Marker  
  
good\_eats = {  
 "Cham Soot Gol": (33.772819, -117.9694484),  
 "The Boiling Crab": (33.6996179, -117.8905689),  
 "Tan Hoang Huong": (33.7446965, -117.9629173)  
}  
  
cham\_soot\_gol = Map(center=good\_eats["Cham Soot Gol"], scroll\_wheel\_zoom=True)  
for place in good\_eats:  
 cham\_soot\_gol.add\_layer(Marker(location=good\_eats[place], title=place))  
cham\_soot\_gol

Map(center=[33.772819, -117.9694484], controls=(ZoomControl(options=['position', 'zoom\_in\_text', 'zoom\_in\_titl…

## 7.4 Plotly

import plotly.express as px  
df = px.data.iris()  
fig = px.scatter(df, x="sepal\_width", y="sepal\_length",   
 color="species",   
 marginal\_y="violin", marginal\_x="box",   
 trendline="ols", template="simple\_white")  
fig.show()

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

# 8 Example Jupyter Notebook

The editing experience with Jupyter Notebook is very similar to the RStudio editing experience: when we make changes and save them, the preview window opened by quarto preview (if your document is not in a project) and quarto serve (if it is) will update.

Here’s some code:

for x in range(10):  
 print(f'{x} squared is {x ^2}.')

0 squared is 2.  
1 squared is 3.  
2 squared is 0.  
3 squared is 1.  
4 squared is 6.  
5 squared is 7.  
6 squared is 4.  
7 squared is 5.  
8 squared is 10.  
9 squared is 11.

What’s different about using Quarto to render Jupyter notebooks is that we can use Knitr code chunk options in the code cells. For example, we can use the label and fig.cap code chunk options in this cell below.

#| fig-cap: "A straight line projected into polar coordinates. "  
#| label: fig-polar

import numpy as np  
import matplotlib.pyplot as plt  
  
r = np.arange(0, 2, 0.01)  
theta = 2 \* np.pi \* r  
fig, ax = plt.subplots(subplot\_kw={'projection': 'polar'})  
ax.plot(theta, r)  
ax.set\_rticks([0.5, 1, 1.5, 2])  
  
ax.grid(True)  
plt.show()

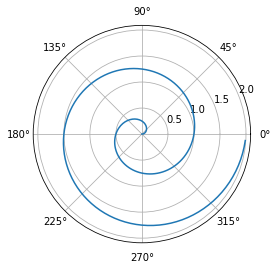


Figure 8.1: A straight line projected into polar coordinates.

And we can add chunk options just like we did in RStudio.

See [fig. 8.1](#fig-polar) for an example of a projection of a straight line into polar coordinates.

# 9 DNA String Stuff

Below are some functions to do some basic DNA string calculations. But this is a lot of function definitions and code testing that a lot of people probably don’t care about. We can hide the code beneath a fold using code-fold and provide a comment with code-summary:

#| code-fold: true  
#| code-summary: "Function definitions and doctests"

import pandas as pd  
def reverse\_complement(nuc\_sequence: str) -> str:  
 """  
 Returns the reverse complement of a nucleotide sequence.  
 >>> reverse\_complement('ACGT')  
 'ACGT'  
 >>> reverse\_complement('ATCGTGCTGCTGTCGTCAAGAC')  
 'GTCTTGACGACAGCAGCACGAT'  
 >>> reverse\_complement('TGCTAGCATCGAGTCGATCGATATATTTAGCATCAGCATT')  
 'AATGCTGATGCTAAATATATCGATCGACTCGATGCTAGCA'  
 """  
 complements = {  
 "A": "T",  
 "C": "G",  
 "G": "C",  
 "T": "A"  
 }  
 rev\_seq = "".join([complements[s] for s in nuc\_sequence.upper()[::-1]])  
 return rev\_seq  
  
def gc\_content(nuc\_sequence: str) -> float:  
 """  
 Calculates the GC content of a nucleotide sequence.  
 >>> gc\_content('ACGT')  
 0.5  
 """  
 gc\_tally = 0  
 for nuc in nuc\_sequence.lower():  
 if nuc == 'g' or nuc == 'c':  
 gc\_tally += 1  
 return gc\_tally / len(nuc\_sequence)  
  
def random\_dna\_string(seq\_length: int = 10) -> str:  
 """  
 Generates a random DNA string seq\_length bp long  
 >>> len(random\_dna\_string())  
 10  
 >>> len(random\_dna\_string(20))  
 20  
 """  
 from random import choice  
   
 dna\_string = ""  
 for \_ in range(seq\_length):  
 dna\_string += choice("ACGT")  
 return dna\_string  
  
def make\_strings\_df(num\_strings: int = 10, str\_length: int = 10) -> pd.DataFrame:  
 """  
 Generates a pandas dataframe with num\_strings DNA sequences of length str\_length with   
 columns "Sequence", "GC Content", "Reverse Complement"  
 >>> df = make\_strings\_df(100, 37)  
 >>> df.shape  
 (100, 3)  
 >>> len(df['Sequence'][0])  
 37  
 """  
 dna\_strings\_list = [random\_dna\_string(str\_length) for \_ in range(num\_strings)]  
 strings\_df = pd.DataFrame({  
 "Sequence": dna\_strings\_list  
 })  
 strings\_df['GC Content'] = strings\_df['Sequence'].apply(gc\_content)  
 strings\_df['Reverse Complement'] = strings\_df['Sequence'].apply(reverse\_complement)  
 return strings\_df  
  
import doctest  
doctest.testmod(verbose=0)

TestResults(failed=0, attempted=9)

Let’s use the function and create a histogram of the GC contents for the simulated sequences.

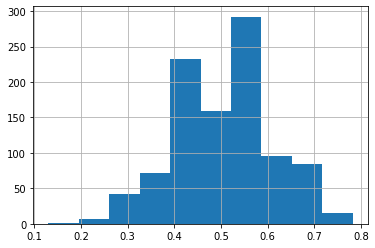
strings\_df = make\_strings\_df(1000, 23)  
print(f'strings\_df has {strings\_df.shape[0]} rows and {strings\_df.shape[1]} columns.')

strings\_df has 1000 rows and 3 columns.

strings\_df.head(10)

Sequence GC Content Reverse Complement  
0 TAATAATGGGCTAAACTATGTTT 0.260870 AAACATAGTTTAGCCCATTATTA  
1 GACCGTGACCCAAGGCAGATGGG 0.652174 CCCATCTGCCTTGGGTCACGGTC  
2 TAGGGTTGTGCTTTACCTTACAT 0.391304 ATGTAAGGTAAAGCACAACCCTA  
3 GCAAGGCCGGATACGCGTATAAT 0.521739 ATTATACGCGTATCCGGCCTTGC  
4 ACCACTCCTCAAACGTTACTGAT 0.434783 ATCAGTAACGTTTGAGGAGTGGT  
5 CCTCGTCAGTTGTCACTTCTATG 0.478261 CATAGAAGTGACAACTGACGAGG  
6 ACAATGATCGCAGCCGAGGTATA 0.478261 TATACCTCGGCTGCGATCATTGT  
7 GTTGGATATTCCGCAGCAGAGGA 0.521739 TCCTCTGCTGCGGAATATCCAAC  
8 CGCTTAAAATCCCTGCATAGACC 0.478261 GGTCTATGCAGGGATTTTAAGCG  
9 AGACCACTACTGGGTGGAGACGG 0.608696 CCGTCTCCACCCAGTAGTGGTCT

strings\_df['GC Content'].hist();



# 10 Sample Analysis

This is some preliminary analysis I did for an experiment we’re doing to quantify the effects of crop priming on nitrogen fixation rates. You can see the latest version of this analysis [at my research journal](https://pommevilla.github.io/crop_priming/analysis_so_far.html) and learn more about the experiment at [my website](https://pommevilla.netlify.app/project/20210823_icrop_priming/). Also note that you can view the source code of this page by click on the Code button to the right of the title of this page.

### 10.0.1 Data

data.priming <- read.csv("data/priming\_amoA\_deltaCt.csv", header = T) %>%   
 rename(sample\_id = X)   
  
data.raw <- read.csv("data/priming\_amoA\_rawCt.csv", header = T) %>%   
 rename(sample\_id = X)  
  
data.priming.long <- data.priming %>%   
 pivot\_longer(cols = starts\_with("amoA"), names\_to = "amoA", values\_to = "deltaCT")  
  
data.raw.long <- data.raw %>%   
 pivot\_longer(cols = starts\_with("amoA"), names\_to = "amoA", values\_to = "CT")  
  
data.priming.long$sample\_id <- fct\_reorder(data.priming.long$sample\_id, parse\_number(data.priming.long$sample\_id))  
  
df <- data.priming[, -1]  
rownames(df) <- data.priming[, 1]  
  
metadata <- df %>%   
 select(fert\_level:field\_rep) %>%  
 mutate(across(everything(), as.factor))  
  
  
amoa\_counts <- df %>%   
 select(starts\_with("amoA"))

data.priming contains the data for our experiment. There are rows for samples, columns for the delta CTs of the different amoAs, and some metadata.

data.priming[1:5, 1:5]

sample\_id amoA.001 amoA.002 amoA.003 amoA.004  
1 2b 10.119249 27.41268 8.764504 8.992937  
2 35b 9.837943 27.51089 9.300077 10.445448  
3 52f 26.345485 26.34548 26.345485 26.345485  
4 34f 26.914432 26.91443 26.914432 26.914432  
5 16f 8.337293 25.94591 8.371314 25.945907

data.raw contains the same columns but lists the raw CT values instead of the 16s-normalized ones.

data.raw[1:5, 1:5]

sample\_id amoA.001 amoA.002 amoA.003 amoA.004  
1 2b 22.70657 40 21.35182 21.58026  
2 35b 22.32706 40 21.78919 22.93456  
3 52f 40.00000 40 40.00000 40.00000  
4 34f 40.00000 40 40.00000 40.00000  
5 16f 22.39139 40 22.42541 40.00000

THe long versions of these dataframes contains the same info but in long format to play nicely with ggplot.

## 10.1 Removing amoAs

We’ll start by removing those amoAs from our data that are not present in over 30 samples across both treatments.

We’ll first start by counting the non-detects for each amoA.

non\_detect\_counts <- data.raw.long %>%  
 group\_by(fert\_level, amoA) %>%   
 count(CT == 40) %>%   
 rename(non\_detect = `CT == 40`) %>%  
 filter(non\_detect == TRUE)

Finding the amoAs that are not detected in > 30 across both samples

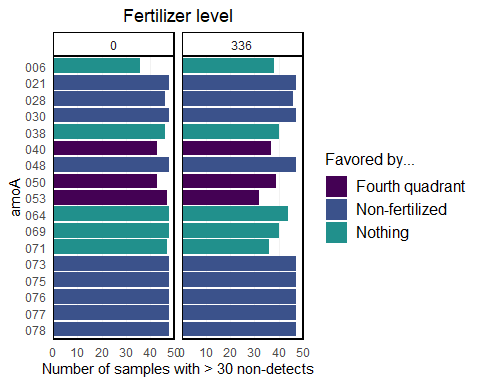
removes <- non\_detect\_counts %>%   
 pivot\_wider(names\_from = fert\_level, values\_from = n, names\_prefix = "fert.") %>%  
 filter(fert.0 > 30 & fert.336 > 30) %>%  
 pivot\_longer(cols = fert.0:fert.336, names\_to = "fert\_level", values\_to = "n")

We’ll now reduce data.priming by removing those amoAs that are largely non-detects. We’ll also update the long version while we’re at it

data.priming.reduced <- data.priming %>%   
 select(-one\_of(removes$amoA))  
  
data.priming.reduced.long <- data.priming.reduced %>%   
 select(-sample\_id, field\_rep) %>%   
 pivot\_longer(cols = contains("amoa"))

Here’s a barchart of what we’re removing:

removes %>%   
 mutate(amoA = str\_sub(amoA, -3)) %>%   
 mutate(favored = case\_when(  
 amoA %in% c("006", "038", "064", "069", "071") ~ "Nothing",  
 amoA %in% c("021", "028", "030", "048", "073", "075", "076", "077", "078") ~ "Non-fertilized",  
 amoA %in% c("040", "050", "053") ~ "Fourth quadrant",  
 TRUE ~ "First quadrant"  
 )) %>%   
 mutate(fert\_level = str\_sub(fert\_level, start = 6)) %>%   
 ggplot(aes(amoA, n, fill = favored )) +  
 geom\_col() +  
 facet\_wrap(~ fert\_level) +   
 theme(  
 plot.title = element\_text(hjust = 0.5),  
 legend.text = element\_markdown(size = 12),  
 legend.title = element\_markdown(size = 12, hjust = 0),  
 strip.background = element\_rect(size = 1, color = "black", fill = "NA"),  
 panel.grid = element\_line(color = "gray95"),  
 panel.grid.major.y = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.minor.x = element\_blank(),  
 panel.border = element\_rect(color = "black", size = 1, fill = NA)  
 ) +  
 scale\_fill\_viridis\_d(begin = 0, end = 0.5) +  
 scale\_y\_continuous(limits = c(0, 50), expand = expansion(add = c(0, 0))) +  
 scale\_x\_discrete(limits = rev) +   
 coord\_flip() +   
 labs(  
 y = "Number of samples with > 30 non-detects",  
 title = "Fertilizer level",  
 fill = "Favored by..."  
 )



Note that most of the non-detects that we’re removing are from the non-fertilized group.

Next, we’ll convert the CT values to presence/absence for use in later analysis.

amoA\_presence\_absence <- data.raw %>%   
 select(sample\_id, starts\_with("amoA")) %>%  
 mutate(across(starts\_with("amoA"), ~ ifelse(.x == 40, 0, 1)))

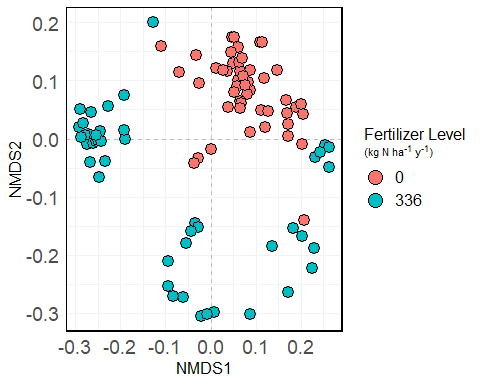
## 10.2 Ordination

Calculating the NMDS (positioning the sites):

mds.priming = metaMDS(data.priming.reduced %>% select(contains("amoa")), distance = "bray", k = 3)  
  
site.scores <- as.data.frame(scores(mds.priming, display = "sites")) %>%   
 mutate(sample\_id = data.priming.reduced$sample\_id,  
 Crop = data.priming.reduced$crop,  
 Fert\_Level = as.factor(data.priming.reduced$fert\_level),  
 Day = as.factor(data.priming.reduced$doe),  
 Substrate\_Addition = as.factor(data.priming.reduced$addition))

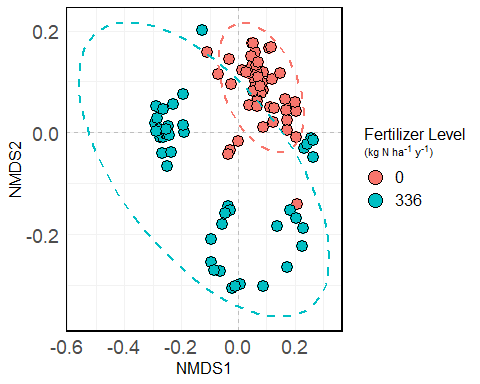
This is enough to plot a basic NMDS:

nmds.plot <- site.scores %>%   
 ggplot(aes(NMDS1, NMDS2, fill = Fert\_Level)) +   
 geom\_hline(yintercept = 0.0,  
 colour = "grey",  
 lty = 2) +  
 geom\_vline(xintercept = 0.0,  
 colour = "grey",  
 lty = 2) +  
 geom\_point(size = 4, shape = 21) +   
 theme(  
 plot.title = element\_text(hjust = 0.5),  
 legend.text = element\_markdown(size = 12),  
 legend.title = element\_markdown(size = 12, hjust = 0),  
 axis.text.x = element\_text(size = 14),  
 axis.text.y = element\_text(size = 14),  
 axis.title.x = element\_text(size = 12),  
 axis.title.y = element\_text(size = 12),  
 panel.grid = element\_line(color = "gray95"),  
 panel.border = element\_rect(color = "black", size = 1, fill = NA)  
 ) +  
 scale\_fill\_discrete(name = "Fertilizer Level<br>  
 <span style = 'font-size:8pt;'>  
 (kg N ha<sup>-1</sup> y<sup>-1</sup>)  
 </span>") +  
 guides(  
 fill = guide\_legend(override.aes = list(shape = 21, size = 5))  
 )   
  
nmds.plot



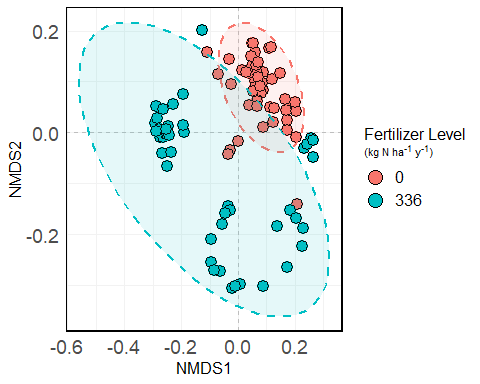
We can also add ellipses to the plot indicate confidence intervals if you’re interested in that:

nmds.plot +  
 stat\_ellipse(aes(color = Fert\_Level), size = 1, linetype = "dashed", show.legend = FALSE)



And again with shading:

nmds.plot +  
 stat\_ellipse(aes(color = Fert\_Level), size = 1, linetype = "dashed", show.legend = FALSE) +  
 stat\_ellipse(aes(fill = Fert\_Level), size = 1, linetype = "dashed", show.legend = FALSE, geom = "polygon", alpha = 0.1)



## 10.3 Arrows!

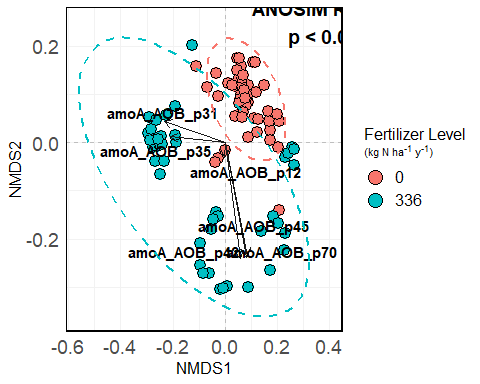
Let’s calculate the loading factors of the individual amoas:

dune\_dist <- vegdist(data.priming %>% select(starts\_with('amoA')))  
  
amoa\_anosim <- anosim(dune\_dist, data.priming$fert\_level)  
  
mds.spp.fit <- envfit(mds.priming, data.priming.reduced %>% select(contains("amoa")), permutations = 999)  
  
spp.scrs <- as.data.frame(scores(mds.spp.fit, display = "vectors"))   
spp.scrs <- cbind(spp.scrs, Species = rownames(spp.scrs))   
spp.scrs <- cbind(spp.scrs, pval = mds.spp.fit$vectors$pvals)  
  
spp.scores <- as.data.frame(scores(mds.spp.fit, display = "vectors")) %>%   
 mutate(Species = rownames(.),  
 pval = mds.spp.fit$vectors$pvals)

This is enough to plot arrows on the NMDS. We’ll show the loadings of some amoAs of interest that we identified in a previous analysis.

special <- c("amoA.012", "amoA.031", "amoA.035", "amoA.042", "amoA.045", "amoA.070")  
  
special\_arrows <- spp.scores %>%   
 rownames\_to\_column() %>%   
 filter(rowname %in% special) %>%   
 mutate(x = -0.25 \* NMDS1,  
 y = -0.25 \* NMDS2,  
 assay = str\_sub(rowname, -2),  
 assay = paste0("amoA\_AOB\_p", assay)  
 )  
  
nmds.plot +  
 geom\_segment(data = special\_arrows,  
 aes(x = 0, xend = -0.3 \* NMDS1,  
 y = 0, yend = -0.3 \* NMDS2),  
 size = 0.66,  
 arrow = arrow(length = unit(0.25, "cm")),  
 color = "grey10", lwd = 0.3,  
 inherit.aes = FALSE) +  
 ggrepel::geom\_text\_repel(  
 data = special\_arrows,  
 aes(x \* 1, y \* 1, label = assay),  
 fontface = "bold",  
 size = 4,  
 inherit.aes = FALSE,  
 force = 1,  
 nudge\_x = -0.001  
 ) +  
 annotate(  
 "text",  
 label = paste0("ANOSIM R = ", round(amoa\_anosim$statistic, 2),  
 "\np < 0.001"),  
 x = 0.4,  
 y = 0.25,  
 size = 5,  
 fontface = 2  
 ) +  
 stat\_ellipse(aes(color = Fert\_Level), size = 1, linetype = "dashed", show.legend = FALSE)

Warning: Duplicated aesthetics after name standardisation: size



## 10.4 Statistics

### 10.4.1 Which factors have an impact on overall community composition?

X <- data.priming.reduced %>%   
 select(-c(contains("amoa")))  
Y <- data.priming.reduced %>%   
 select(c(contains("amoa")))  
  
adonis(Y ~ X$fert\_level + X$addition + X$crop + X$timepoint)

Call:  
adonis(formula = Y ~ X$fert\_level + X$addition + X$crop + X$timepoint)   
  
Permutation: free  
Number of permutations: 999  
  
Terms added sequentially (first to last)  
  
 Df SumsOfSqs MeanSqs F.Model R2 Pr(>F)   
X$fert\_level 1 1.4426 1.44255 56.092 0.37093 0.001 \*\*\*  
X$addition 2 0.0327 0.01633 0.635 0.00840 0.701   
X$crop 1 0.0882 0.08823 3.431 0.02269 0.031 \*   
X$timepoint 1 0.0367 0.03671 1.427 0.00944 0.224   
Residuals 89 2.2889 0.02572 0.58855   
Total 94 3.8890 1.00000   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

This tells us that fertilization level is very significant and explains ~37% of the variation in our samples. Crop is also a significant factor on community composition, though it only explains 2.3% of the variation.

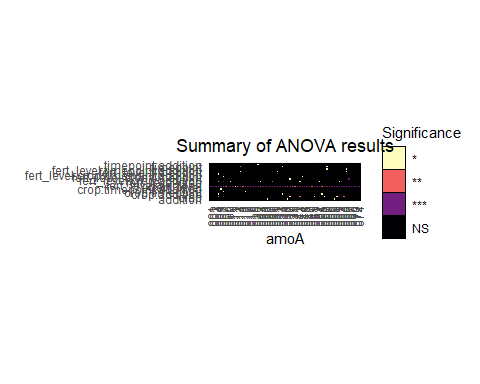
### 10.4.2 How do the treatment factors affect the “abundance” of genes on an individual level?

All the code below does is perform an ANOVA of the gene’s abundance against all the terms and all of their interactions.

formulae <- lapply(colnames(data.priming.reduced %>% select(starts\_with("amoA"))), function(x) as.formula(paste0(x, " ~ fert\_level \* crop \* timepoint \* addition")))  
  
res <- lapply(formulae, function(x) broom::tidy(aov(x, data = data.priming.reduced)))  
names(res) <- format(formulae)  
names(res) <- str\_sub(names(res), end = 8)  
  
anova\_results <- lapply(seq\_along(res), function(i) res[[i]] %>% mutate(gene = names(res)[[i]])) %>%   
 bind\_rows() %>%   
 filter(term != "Residuals") %>%   
 mutate(sig = case\_when(  
 p.value < 0.05 & p.value > 0.01 ~ "\*",  
 p.value < 0.01 & p.value > 0.001 ~ "\*\*",  
 p.value < 0.001 ~ "\*\*\*",  
 TRUE ~ "NS"  
 ))

Visualization:

anova\_results %>%   
 mutate(gene = str\_sub(gene, -3)) %>%   
 ggplot(aes(gene, term, fill = sig)) +   
 geom\_tile(color = "black") +   
 coord\_equal() +   
 labs(y = "",  
 x = "amoA",  
 title = "Summary of ANOVA results",  
 fill = "Significance ") +   
 theme(  
 plot.title = element\_text(hjust = 0.5),  
 axis.text.x = element\_text(angle = 90, hjust = 0, vjust = 0.5)  
 ) +   
 scale\_fill\_viridis\_d(option = "magma", direction = -1)

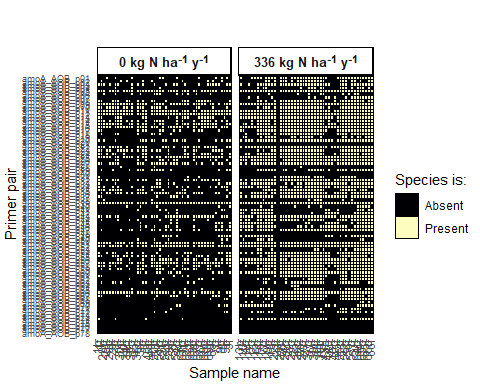


Overall, we see that, again, fertilization level has a significant impact on abundance levels of the individual genes, and it’s not even really that close. There are other factors that might be worth investigating on a gene-by-gene basis, too, but that’s for later.

## 10.5 Biodiversity

Let’s start by visualizing the presence/absence table:

amoA\_presence\_absence %>%   
 pivot\_longer(cols = starts\_with("amoA"), names\_to = "amoA", values\_to = "presence") %>%   
 mutate(amoA = str\_sub(amoA, -2),  
 amoA = paste0("amoA\_AOB\_p", amoA),  
 presence = as.factor(presence)) %>%   
 left\_join(metadata %>% rownames\_to\_column(var = "sample\_id")) %>%   
 mutate(strip\_label = paste0(fert\_level, " kg N ha<sup>-1</sup> y<sup>-1</sup>")) %>%   
 ggplot(aes(sample\_id, amoA, fill = presence)) +   
 geom\_tile(color = "black") +  
 labs(  
 x = "Sample name",  
 y = "Primer pair",  
 fill = "Species is:",  
 title = "",  
 subtitle = ""  
 ) +   
 scale\_fill\_viridis\_d(labels = c("Absent", "Present"),  
 begin = 0, end = 1,  
 option = "magma") +  
 theme(  
 axis.text.x = element\_text(angle = 90, hjust = 1, vjust = 0.25),  
 axis.text.y = element\_text(size = 7),  
 plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5),  
 strip.text = element\_markdown(size = 10, face = "bold"),  
 strip.background = element\_rect(size = 1, color = "blaCk", fill = NA),  
 plot.margin = unit(c(0, 0.1, 0.1, 0.1), "cm")  
 ) +   
 scale\_y\_discrete(limits = rev) +   
 facet\_grid(~ strip\_label, scales = "free")



## 10.6 Reading in the best BLAST hit info:

amoA\_organism\_info <- readxl::read\_xlsx("data/amoa\_mfp\_qpcr\_org\_accessions.xlsx", sheet = 5) %>%   
 select(-c(contains(c("forward", "reverse", "notes"))))

Counts of best BLAST hits:

amoA\_organism\_info %>%   
 count(best\_blast\_hits, sort = TRUE)

# A tibble: 25 x 2  
 best\_blast\_hits n  
 <chr> <int>  
 1 Nitrosolobus multiformis AmoA (amoA) gene 10  
 2 Nitrosospira sp. En13 AmoA 9  
 3 Nitrosospira multiformis ATCC 25196 7  
 4 Nitrosospira sp. Wyke8 AmoA 7  
 5 Nitrosospira lacus strain APG3 6  
 6 Nitrosospira sp. Np39-19 6  
 7 Nitrosospira sp. Wyke2 4  
 8 Nitrosospira sp. NpAV 3  
 9 Nitrosomonas sp. JL21 2  
10 Nitrosospira briensis 2  
# ... with 15 more rows

## 10.7 Creating a phyloseq object

amoa\_tax\_table <- amoA\_organism\_info %>%   
 select(array\_name, best\_blast\_hits) %>%   
 column\_to\_rownames(var = "array\_name") %>%   
 tax\_table()  
  
rownames(amoa\_tax\_table) <- amoA\_organism\_info$array\_name

ps <- phyloseq(  
 otu\_table(amoA\_presence\_absence %>% column\_to\_rownames(var = "sample\_id"), taxa\_are\_rows = FALSE),  
 sample\_data(metadata),  
 amoa\_tax\_table  
)

## 10.8 Richness analysis

How does observed richness and evenness change with treatment level? This is a modified diversity function that does a bunch of nice stuff that phyloseq::estimate\_richness doesn’t do.

estimate\_richness\_mod <- function(physeq, split=TRUE, measures=NULL){  
   
  
 if( !split ){  
 OTU <- taxa\_sums(physeq)   
 } else if( split ){  
 OTU <- as(otu\_table(physeq), "matrix")  
 if( taxa\_are\_rows(physeq) ){ OTU <- t(OTU) }  
 }  
   
  
 renamevec = c("Observed", "Chao1", "ACE", "Shannon", "Pielou", "Simpson", "InvSimpson", "SimpsonE", "Fisher")  
 names(renamevec) <- c("S.obs", "S.chao1", "S.ACE", "shannon", "pielou", "simpson", "invsimpson", "simpsone", "fisher")  
  
 if( is.null(measures) ){  
 measures = as.character(renamevec)  
 }  
  
 if( any(measures %in% names(renamevec)) ){  
 measures[measures %in% names(renamevec)] <- renamevec[names(renamevec) %in% measures]  
 }  
   
  
 if( !any(measures %in% renamevec) ){  
 stop("None of the `measures` you provided are supported. Try default `NULL` instead.")  
 }  
   
  
 outlist = vector("list")  
  
 estimRmeas = c("Chao1", "Observed", "ACE")  
 if( any(estimRmeas %in% measures) ){  
 outlist <- c(outlist, list(t(data.frame(estimateR(OTU)))))  
 }  
 if( "Shannon" %in% measures ){  
 outlist <- c(outlist, list(shannon = diversity(OTU, index="shannon")))  
 }  
 if( "Pielou" %in% measures){  
 #print("Starting Pielou")  
 outlist <- c(outlist, list(pielou = diversity(OTU, index = "shannon")/log(estimateR(OTU)["S.obs",])))  
 }  
 if( "Simpson" %in% measures ){  
 outlist <- c(outlist, list(simpson = diversity(OTU, index="simpson")))  
 }  
 if( "InvSimpson" %in% measures ){  
 outlist <- c(outlist, list(invsimpson = diversity(OTU, index="invsimpson")))  
 }  
 if( "SimpsonE" %in% measures ){  
  
 outlist <- c(outlist, list(simpsone = diversity(OTU, index="invsimpson")/estimateR(OTU)["S.obs",]))  
 }  
 if( "Fisher" %in% measures ){  
 fisher = tryCatch(fisher.alpha(OTU, se=TRUE),  
 warning=function(w){  
 warning("phyloseq::estimate\_richness: Warning in fisher.alpha(). See `?fisher.fit` or ?`fisher.alpha`. Treat fisher results with caution")  
 suppressWarnings(fisher.alpha(OTU, se=TRUE)[, c("alpha", "se")])  
 }  
 )  
 if(!is.null(dim(fisher))){  
 colnames(fisher)[1:2] <- c("Fisher", "se.fisher")  
 outlist <- c(outlist, list(fisher))  
 } else {  
 outlist <- c(outlist, Fisher=list(fisher))  
 }  
 }  
 out = do.call("cbind", outlist)  
  
 namechange = intersect(colnames(out), names(renamevec))  
 colnames(out)[colnames(out) %in% namechange] <- renamevec[namechange]  
  
 colkeep = sapply(paste0("(se\\.){0,}", measures), grep, colnames(out), ignore.case=TRUE)  
 out = out[, sort(unique(unlist(colkeep))), drop=FALSE]  
  
 out <- as.data.frame(out)  
 return(out)  
}

metrics <- c("Observed", "Shannon")  
richness <- estimate\_richness\_mod(ps, measures = metrics) %>%   
 rownames\_to\_column(var = "sample\_id") %>%   
 mutate(sample\_id = str\_sub(sample\_id, start = 2))   
  
richness <- left\_join(sample\_data(ps) %>% data.frame() %>% rownames\_to\_column(var = "sample\_id"), richness) %>%   
 pivot\_longer(cols = Observed:Shannon, names\_to = "Metric", values\_to = "Value")

Joining, by = "sample\_id"

## 10.9 Statistical tests

### 10.9.1 Significance test of fertilization level on richness.

(sig\_rich\_fert <- kruskal.test(Value ~ fert\_level, data = richness %>% filter(Metric == "Observed")))

Kruskal-Wallis rank sum test  
  
data: Value by fert\_level  
Kruskal-Wallis chi-squared = 54.212, df = 1, p-value = 1.8e-13

The p-value < 0.001 gives us strong statistical evidence that richness is significantly different between fertilization treatment groups.

### 10.9.2 Significance test of fertilization level on richness

(sig\_even\_fert <- kruskal.test(Value ~ fert\_level, data = richness %>% filter(Metric == "Shannon")))

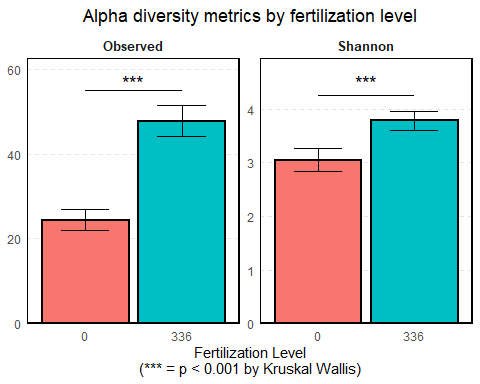
Kruskal-Wallis rank sum test  
  
data: Value by fert\_level  
Kruskal-Wallis chi-squared = 54.268, df = 1, p-value = 1.75e-13

The p-value < 0.001 gives us strong statistical evidence that Shannon diversity is significantly different between fertilization treatment groups.

## 10.10 Making nice plots for stat differences

Standard deviations, mean

summaries <- richness %>%   
 group\_by(Metric, fert\_level) %>%   
 summarize(mean\_val = mean(Value),   
 sd\_val = sd(Value),  
 n = n(),  
 .groups = "drop") %>%   
 mutate(se = abs((sd\_val / sqrt(n)) \* qt(0.025, n - 1) )) %>%   
 mutate(ymax = mean\_val + se,   
 ymin = mean\_val - se)   
  
this\_annotation <- data.frame(  
 Metric = c("Observed", "Shannon"),  
 lab = c("\*\*\*", "\*\*\*"),  
 x = 1.5,  
 y = c(50 + 5 + 2, 4 + 0.5),  
 lineheights = c(50 + 5, 4 + 0.25)  
)  
  
summaries %>%   
 ggplot(aes(fert\_level, mean\_val, fill = fert\_level)) +   
 geom\_col(color = "black", size = 1) +  
 facet\_wrap(~ Metric, scales = "free\_y") +   
 theme(  
 legend.position = "none",  
 strip.background = element\_blank(),  
 axis.title.y = element\_blank(),  
 strip.placement = "outside",  
 plot.title = element\_text(hjust = 0.5),  
 strip.text.y = element\_text(face = "bold", size = 10),  
 strip.text = element\_text(face = "bold", size = 10),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.major.y = element\_line(color = "gray90", linetype = "dashed"),  
 axis.ticks = element\_blank(),  
 panel.border = element\_rect(color = "black", size = 1, fill = "NA")  
 ) +  
 scale\_y\_continuous(expand = expansion(mult = c(0, 0.1))) +   
 geom\_errorbar(aes(ymin = ymin, ymax = ymax, width = 0.5)) +   
 geom\_text(  
 data = this\_annotation,  
 aes(x = x, y = y, label = lab),  
 inherit.aes = FALSE,  
 size = 5  
 ) +   
 geom\_segment(data = this\_annotation,  
 aes(x = 1,  
 xend = 2,   
 y = lineheights,  
 yend = lineheights),  
 inherit.aes = FALSE) +   
 labs(  
 x = "Fertilization Level\n(\*\*\* = p < 0.001 by Kruskal Wallis)",  
 title = "Alpha diversity metrics by fertilization level"  
   
 )



## 10.11 Beta diversity

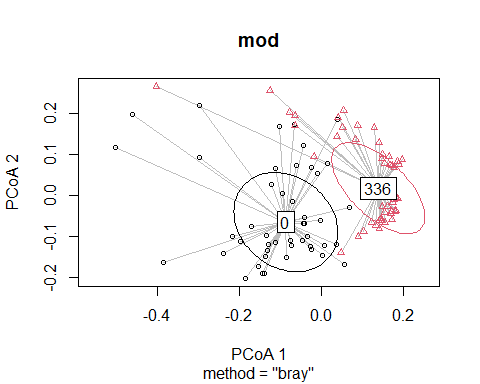
We’ll start beta diversity analysis off by doing an ADONIS/PERMANOVA to determine if treatment centroids/treatment variations are different between groups.

dis <- vegdist(otu\_table(ps))  
groups <- sample\_data(ps)$fert\_level  
mod <- betadisper(dis, groups)  
anova(mod)

Analysis of Variance Table  
  
Response: Distances  
 Df Sum Sq Mean Sq F value Pr(>F)   
Groups 1 0.52595 0.52595 24.589 3.228e-06 \*\*\*  
Residuals 92 1.96781 0.02139   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

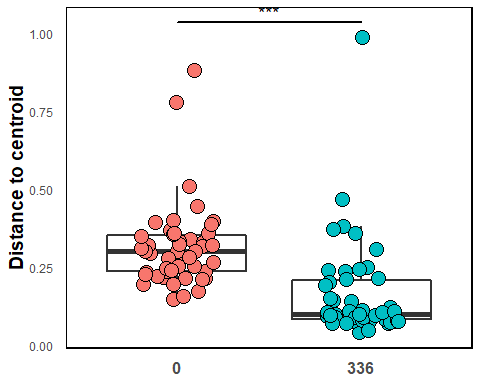
Since p <<<< 0.0001, there is strong evidence that the overall community compositions are significantly different (treatment centroid, distance to centroid, community variation) between the two groups. W can visualize this with a 1 SD ellipse:

plot(mod, ellipse = TRUE, hull = FALSE)



We see that there is clear separation between the two treatment centroids. Let’s do some more analysis on the distance-to-centroids that we’re seeing:

betadistances <- data.frame(  
 time\_frame = mod$group,  
 distance = mod$distances  
)  
  
betadistances %>%   
 ggplot(aes(time\_frame, distance)) +   
 geom\_boxplot(size = 1, outlier.shape = NA) +   
 geom\_jitter(aes(fill = time\_frame), size = 5, shape = 21, width = 0.2) +  
 theme(  
 legend.position = "none",  
 panel.grid.minor.x = element\_blank(),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.major.y = element\_blank(),  
 plot.title = element\_text(size = 17),  
 plot.subtitle = element\_text(size = 9),  
 axis.ticks.length = unit(0.25, "cm"),  
 axis.ticks.x = element\_blank(),  
 axis.text.x = element\_text(face = "bold", angle = 0, size = 12),  
 panel.border = element\_rect(color = "black", size = 1, fill = NA),  
 axis.title.x = element\_blank(),  
 axis.title.y = element\_text(size = 14, face = "bold"),  
   
 ) +   
 labs(  
 color = "",  
 y = "Distance to centroid"  
 ) +  
 ggsignif::geom\_signif(  
 map\_signif\_level = TRUE,  
 comparisons = list(c("0", "336")),  
 test = "t.test",  
 step\_increase = 0.1,  
 color = "black",  
 size = 1,  
 textsize = 5,  
 tip\_length = 0  
 )

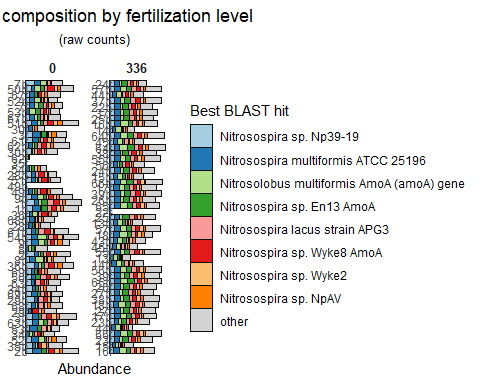


The significance bar is coming from the PERMANOVA test we did above. We see that there is actually less beta diversity (as meaasured by distance-to-centroid) in the fertilized group than in the non- fertilized group. We’ll see another visualization backing this up in the next section:

## 10.12 Composition

Let’s visualize the composition of the communities, separated by fertilization. We’ll start with raw counts - how many times was that best BLAST hit seen in that sample?

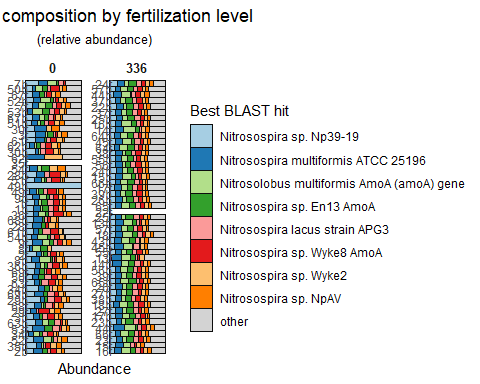
comp\_barplot(ps, "ta1",  
 facet\_by = "fert\_level",  
 sample\_order = "default",  
 tax\_transform\_for\_plot = "identity") +  
 coord\_flip() +   
 labs(  
 title = "Sample composition by fertilization level",  
 subtitle = "(raw counts)"  
 ) +   
 theme(  
 axis.text.x = element\_blank(),  
 axis.text.y = element\_text(margin = margin(r = -7)),  
 plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5, size = 10),  
 strip.text = element\_text(size = 10, face = "bold")  
 ) +  
 guides(  
 fill = guide\_legend(title = "Best BLAST hit", reverse = TRUE)  
 )



We see that overall the fertilized group appears to have more richness in it.

How about sample composition? IE, relative abundances?

comp\_barplot(ps, "ta1",  
 facet\_by = "fert\_level",  
 sample\_order = "default") +  
 coord\_flip() +   
 labs(  
 title = "Sample composition by fertilization level",  
 subtitle = "(relative abundance)"  
 ) +   
 theme(  
 axis.text.x = element\_blank(),  
 axis.text.y = element\_text(margin = margin(r = -7)),  
 plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5, size = 10),  
 strip.text = element\_text(size = 10, face = "bold")  
 ) +  
 guides(  
 fill = guide\_legend(title = "Best BLAST hit", reverse = TRUE)  
 )



Two big things pop out:

* Species distribution is more even in the fertilized group. This makes sense given previous results showing that Shannon entropy is higher and beta diversity is lower in the fertilized group. You can also see that the communities just look more like each other in the fertilized group, which manifests in shorter distance-to-centroids/lower community variation.
* There’s more green in the fertilized group, suggesting that Nitrosolobus multiformus and Nitrosospira sp. En13 abundances are significantly affected by fertilization addition.

## 10.13 Statistics on a best BLAST hit level

The next chunk is just doing some data transformation stuff to count the number of times each organism was seen in each sample in preparation for the statistical analysis.

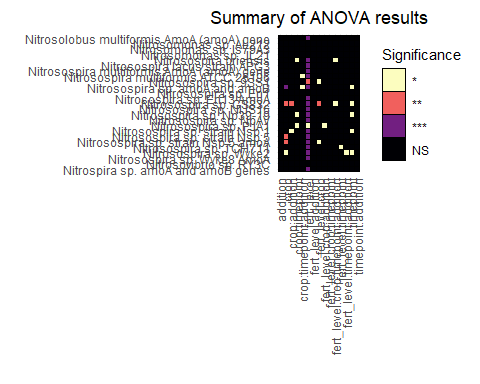
pa\_count <- ps %>%   
 otu\_table() %>%  
 data.frame %>%  
 rownames\_to\_column(var = "sample\_id") %>%   
 pivot\_longer(starts\_with("amoA"))   
  
  
org\_table <- tax\_table(ps) %>%   
 data.frame %>%   
 rownames\_to\_column(var = "name") %>%   
 rename(bbh = ta1) %>%   
 mutate(cleaned\_names = janitor::make\_clean\_names(bbh))  
  
bbh\_sample\_counts <- left\_join(pa\_count, org\_table, by = "name") %>%  
 group\_by(sample\_id, bbh) %>%   
 summarize(value = sum(value)) %>%   
 pivot\_wider(names\_from = "bbh", values\_from = value)  
  
bbh\_level\_counts <- left\_join(bbh\_sample\_counts,  
 sample\_data(ps) %>%   
 data.frame %>%   
 rownames\_to\_column(var = 'sample\_id') %>%   
 right\_join(bbh\_sample\_counts)  
) %>%   
 ungroup()

Here, we’re preparing formulas to feed to a lapply function to perform a Kruskal-Wallis test on all of the organisms.

formulae <- lapply(colnames(bbh\_sample\_counts %>% select(-sample\_id) %>% janitor::clean\_names()) , function(x) as.formula(paste0(x, " ~ fert\_level \* crop \* timepoint \* addition")))  
  
formulae[[1]] <- NULL  
  
res <- lapply(formulae, function(x) broom::tidy(aov(x, data = bbh\_level\_counts %>% janitor::clean\_names())))  
names(res) <- format(formulae)  
names(res) <- lapply(names(res), function(x) str\_split(x, "~")[[1]][1]) %>% unlist()  
  
anova\_results.counts <- lapply(seq\_along(res), function(i) res[[i]] %>% mutate(gene = names(res)[[i]])) %>%   
 bind\_rows() %>%   
 filter(term != "Residuals") %>%   
 mutate(gene = str\_trim(gene))

Visualizing the results again:

anova\_results.counts %>%  
 left\_join(org\_table, by = c("gene" ="cleaned\_names")) %>%  
 mutate(sig = case\_when(  
 p.value < 0.05 & p.value > 0.01 ~ "\*",  
 p.value < 0.01 & p.value > 0.001 ~ "\*\*",  
 p.value < 0.001 ~ "\*\*\*",  
 TRUE ~ "NS"  
 )) %>%   
 ggplot(aes(term, bbh, fill = sig)) +   
 geom\_tile(color = "black") +   
 labs(y = "",  
 x = "",  
 title = "Summary of ANOVA results",  
 fill = "Significance ") +   
 theme(  
 plot.title = element\_text(hjust = 0.5),  
 axis.text.x = element\_text(angle = 90, hjust = 1, vjust = 0.5),  
 axis.text.y = element\_text()  
 ) +   
 scale\_fill\_viridis\_d(option = "magma", direction = -1) +   
 scale\_y\_discrete(limits = rev) +  
 coord\_equal()



We see the same pattern at the organism level as when we did this at the gene level: fertilization level is by far the most significant factor affecting Presence/Absence of organisms.

Blair, R., A. Kjuchukova, R. Velazquez, and P. Villanueva. 2020. “Wirtinger Systems of Generators of Knot Groups.” *Communications in Analysis and Geometry* 28 (2): 243–62. <https://doi.org/10.4310/cag.2020.v28.n2.a2>.