Whole-Genome Shotgun Sequencing for Pollen Identification

Anya Cutler and Berry Brosi

started 16 August 2018; most recent edits 23 December 2020

Contents

Overview **2**

Analysis approach **2**
non-independence of data . . . 2
comparing WGS to amplicon data . . . 2
structure of analyses: false negatives and false positives . . . 3
structure of analyses: quantitative matching . . . 3

Data import **4**

data formatting / setup **5**
Basic Kraken data prep . . . 5
combine Kraken & sample metadata . . . 6
remove Kraken reads below the contamination threshold . . . 6
Kraken data: format for false negative and quantitative analyses . . . 7
combine amplicon and Kraken data for false negative analysis . . . 8
format kraken data for false positive analysis . . . 9
false positives: format amplicon data . . . 10
combine amplicon and Kraken data for false positive analysis . . . 13

Analysis **14**
false negatives . . . 14
sample complexity: false negatives . . . 14
Amplicon vs. Shotgun comparison: false negatives . . . 16
by marker (*rbcL* vs ITS2)—Amplicon vs. Shotgun false negatives . . . 17
markers combined—Amplicon vs. Shotgun false negatives . . . 18
false positives . . . 19
sample complexity: false positives . . . 19
Amplicon vs. Shotgun comparison, false positives . . . 22
Amplicon markers considered separately (*rbcL* and ITS2) . . . 22
amplicon markers combined . . . 23
quantitative matching . . . 24
WGS data only: quantitative matching . . . 25
Amplicon vs. WGS: quantitative matching . . . 26
data prep for quant matching amplicon vs. WGS . . . 27
residual-based analysis for quant matching amplicon vs. WGS . . . 27
linear-model based analysis for quant matching amplicon vs. WGS . . . 28

Figures **29**
Figure 1: true positives by taxon . . . 29
Figure 2: true positives vs. species richness . . . 31
Figure X: true positives vs. relatedness . . . 36
Figure Y: true positives vs. rarity (binned) . . . 41

Figure S1: true positives combined . . . 46
Figure Z: true positives by all 3 complexity variables; species level only . . . 48
Figure 3: true positives vs. rarity by taxon . . . 48
Figure 4: quantitative matching by taxon . . . 52
Figure 6: false negatives by source . . . 58
data prep for false positives by sample complexity . . . 63
Figure I: false positives vs. species richness . . . 64
Figure J: false positives vs. relatedness . . . 65
Figure K: false positives vs. rarity (binned) . . . 67
Figure S2: false positives combined . . . 69

Session Information **70**

update Dec 2020

- this version of the Rmarkdown is updated from previous versions to reflect that we are now using merged forward and reverse reads. The Rmarkdown should mostly or entirely reflect that.

Overview

This analysis is set up to assess how whole-genome shotgun (WGS) sequencing of known pollen samples performs, both on its own and relative to amplicon-based metabarcoding (with ITS2 and *rbcL*) in terms of:

1. **False negatives** (i.e. true positives)—if a species is present in a sample, is that species detected?
2. **False positives**—how many species that are *not* present in a sample are erroneously detected?
3. **Quantitative matching**—what is the quantitative correspondence between the number of pollen grains going in to a sample and the number of sequence reads coming out?

These three questions form the basic structure of the analysis, which is paralleled by the structure of this file.

The samples that were sequenced with WGS were from constructed "mock community" pollen samples of known composition. We used the exact same DNA isolations as in Bell et al. 2019 *Molecular Ecology*. This analysis is largely (though not entirely) based on the analyses in that paper.

The sequenced pollen samples were constructed to vary in complexity in three dimensions; we assessed how sample complexity affected the qualitative outcomes (false positive and false negative reads):

1. Question 1: **species richness** (1-9 species)
2. Question 2: **rarity** (actual proportion of grains this taxon has in a sample; from roughly half & half to $< 1\%$ of the rarer type)
3. Question 3: **taxonomic relatedness** (within genus to across broad clades in the seed plants)

Analysis approach

non-independence of data

Because all analyses included non-independent data (multiple replicates of the same pollen mixtures; pollen from the same plant species occurring in multiple mixtures), all of our analyses were conducted with mixed-effects modeling, using mixture identity and species identity as crossed random effects (modeled as random intercepts). For the analyses related to false positive matching, all species truly present in a mix were pooled together for the "true positives", so we did not use species identity (up to 9 different species) as a random effect in those analyses.

comparing WGS to amplicon data

Across our outcomes, in comparing WGS and amplicon performance, we pooled WGS and amplicon results together into a single data table and conducted analyses with method (WGS vs. amplicon) as a fixed effect.

structure of analyses: false negatives and false positives

The response variables for our first two outcomes are binomial (yes / no) in structure. We use binomial-errors mixed-effects models for these analyses.

For our first outcome of false negatives, we thus need to record—for each species present in a pollen mixture—whether or not that species was detected in that sample. To do this, we set up a datafile with each species truly present within each sample as its own row, which we subsequently scored as 0 / 1, with a zero for species that were present in the sample but *not* present in sequencing reads (above the contamination threshold), and a one for species that *were* present in the sequencing reads above the threshold.

For our second outcome of false positives, we wished to assess the proportion of true vs. false positives. To do this, we aggregated the data to one row per sample replicate, and summed the read counts of true positives (combining all species truly present in a particular mix) and false positives in two separate columns.

structure of analyses: quantitative matching

To assess the quantitative accuracy of WGS sequencing for our constructed mixtures, we tested the correlation between the known proportion of pollen grains in a sample (explanatory variable) and the proportion of WGS sequencing reads (response variable).

Ideally, we would like for there to be a perfect 1:1 fit between pollen grains going in and sequence reads coming out. This means a linear response between the two, with an intercept of 0 and a slope of 1. One could make an argument that instead of using a linear model, a binomial-errors model would be more appropriate given that the response variable is indeed proportional counts (of sequence reads). Moreover, even with a binomial-errors model, it is possible to get the equivalent of a line with slope = 1 and intercept = 0.

We considered this issue in detail and decided that a linear model is better in this context. In particular, a binomial model could for example predict that for a given taxon, it will have a very very low detection probability until it reaches say ~80% of the pollen grains in the sample at which point the logistic curve could swing up sharply and after ~80% the detection probability could be very close to 1. If the data matched such a model, it would return both a very low p-value and a very high R^2, even though it would not be giving us what we want. Therefore, we decided to stick with a linear model.

An alternative possibility is to specify the intercept (0) and slope (1) and see how well the data do at fitting that line. We also pursued this approach, by examining the *residuals* of WGS vs. amplicon data to that regression line. Smaller residuals are better. One thing to note about this approach is that it is probably dominated by the proportion of false positive reads—more false positives will mean that the proportion of true matches will be further from the 1:1 ratio. We thus used both of the approaches above to assess quantitative matching.

```
knitr::opts_chunk$set(echo = TRUE, error = TRUE)

suppressPackageStartupMessages(require(kableExtra))
suppressPackageStartupMessages(require(dplyr))
suppressPackageStartupMessages(require(stringr))
suppressPackageStartupMessages(require(reshape2))
suppressPackageStartupMessages(require(lme4))
suppressPackageStartupMessages(require(lmerTest))
suppressPackageStartupMessages(require(r2glmm))
suppressPackageStartupMessages(require(knitr))
suppressPackageStartupMessages(require(stringr))
suppressPackageStartupMessages(require(tidyr))
suppressPackageStartupMessages(require(tibble))
suppressPackageStartupMessages(require(data.table))
suppressPackageStartupMessages(require(binom))
suppressPackageStartupMessages(require(ggplot2))
suppressPackageStartupMessages(require(gridExtra))
suppressPackageStartupMessages(require(broom.mixed))
```

Data import

Three groups of data sets to import:

1. sample metadata
2. WGS Kraken empirical data
3. amplicon data

The **metadata** are the same for the WGS and amplicon data, as both were run from the exact same DNA extractions.

For the **WGS Kraken data**, Jamieson Botsch formatted these (25-Apr-2018) from the raw Kraken output to combine the two data sources (we sequenced WGS data both at Emory and at UGA), and to have the family / genus / species separated out into columns; see `Shotgun_data_prep.Rmd`. Unlike the QIIME Illumina data, we do not need to do any aggregation of read counts as this was done automatically by Kraken (nice touch); i.e. all read counts at the species level are also included in genus-level matches and so on. We do just a couple of steps of very basic / simple formatting with the data import here (removal of superfluous numeric column a the beginning of the data; removal of duplicate rows).

- in addition, at some point in 2020 Robert Petit merged the forward and reverse reads from the shotgun data in response to reviews, so that is a fairly substantial change from the previous version. Analyses in late 2020 reflect his update.

The **amplicon data** are represented by 7 data files: The first (`amp_all`) is already merged with the sample metadata and already formatted for false-negative analyses (with a separate row for each species within each sample). There are separate columns for the presence or absence of that taxon at three different taxonomic levels (species, genus, and family) and for each marker (ITS2 and *rbcL*).

This `amp_all` dataset, however, does not include data on false positives. Thus, the remaining six datafiles are there to provide data on how many of the hits were to true positives (known to be present in the sample) vs. false positives (not included in the sample). In contrast to `amp_all`, these are formatted as six separate files, one each for the two markers (ITS2 and *rbcL*) and three levels of taxonomic matching (species, genus, and family). The format of these files matches basically what comes from QIIME: taxa are the rows, and samples are the columns. We re-format these to the tidy format of having samples as rows later.

Across both the amplicon and the kraken data, one issue is that for different samples, matching the taxonomy at different levels leads to different numbers of rows. For example, we have two *Populus* species in some of the samples. For matching at the species level, that is two different rows. But for genus and family, it is just one row. There are some other samples that have two different genera in the same family (e.g., *Poa* and *Zea*, both in the Poaceae). Thus, these have to be either 1) stored as completely separate datafiles; or 2) stored in a single datafile (as in `amp_all`) but properly subset to not include duplicate rows for the same genus or family within samples. In this analysis we have split these out into separate files, which may not have been the best way to go (certainly not for memory management / speed; and also it makes the global environment rather messy). But given the aggregation that we had to do, it was probably the most straightforward way to go.

```
mixes = read.csv("pollen-mixes-proportions.csv")
amp_its_family = read.csv("Amplicon_ITS_Family.csv")
amp_its_genus = read.csv("Amplicon_ITS_Genus.csv")
amp_its_species = read.csv("Amplicon_ITS_Species.csv")
amp_rbc_family = read.csv("Amplicon_rbcL_Family.csv")
amp_rbc_genus = read.csv("Amplicon_rbcL_Genus.csv")
amp_rbc_species = read.csv("Amplicon_rbcL_Species.csv")
amp_all = read.csv("Amplicon_all.csv")

# 'mixes' comes in with some rows duplicated (because in the spreadsheet, each
# was assessed with both ITS2 and with rbcL); fix this here:
mixes = unique(mixes)

#Create list of kraken2 report files
reports = list.files(path="./kraken2_results", full.names=T)

#Read in files as table
```

```
kraken2=lapply(reports, read.table, header=FALSE, sep="\t")

#Create column with file name to get sampleID and bind it to other columns
for (i in 1:length(kraken2)){kraken2[[i]]<-cbind(reports[i],kraken2[[i]])}
kraken2_rbind <- do.call("rbind", kraken2)

#Rename column names
colnames(kraken2_rbind)[c(1,2,3,4,5,6,7)]<-c("sample.id", "perc.hit", "hits", "misses","tax","tax.id","id"
```

data formatting / setup

Overview

1. Basic Kraken data prep
   - filter to include only matches at family / genus / species taxonomic resolutions (not anything coarser, and no intermediate clades)
   - create new columns for `mix.ID` that matches the column in the sample (`pollen-mixes-proportions.csv`) data, and also `replicate.ID` for replicates within each mix
   - a couple of minor tweaks to make sure data are consistent etc.
2. Combine Kraken and sample metadata
3. Remove reads below contamination threshold
4. Format Kraken data for false negative and quantitative matching analyses
   - both of these analyses are about true positives (=false negatives) and straightforward to do the data formatting in the same step
5. Combine amplicon and empirical Kraken data: false negatives
6. Format Kraken data for false positive analysis
7. Combine amplicon and empirical Kraken data: false positives

Basic Kraken data prep

1. create new column for `mix.ID` that matches the column in the sample metadata (`pollen-mixes-proportions.csv`)

2. also create a new `replicate.ID` column (replicate samples of each `mix.ID`)
3. filter by `tax` column, including only species / genus / family levels

```
#Create mixID column
kraken2_rbind$mix.ID = substr(kraken2_rbind$sample.id, 19, 24)

# Want to remove the 6th character if mix.id does not contain the word "mix"
# or contains 2 dashes. Only want six characters if the mix number is
# double digits.
kraken2_rbind$mix.ID = ifelse(!grepl("mix", kraken2_rbind$mix.ID) | str_count(kraken2_rbind$mix.ID, patter

# replace dashes in 'mix.ID' with periods so that they match the 'mixes' data
# and remove underscores
kraken2_rbind$mix.ID = gsub("-", ".", kraken2_rbind$mix.ID)

# remove periods if mix.ID contains "mix" so it matches the mix data
kraken2_rbind$mix.ID = gsub("mix.","mix", kraken2_rbind$mix.ID)

# move the new column to be second in order
# (not at the end where it's hard to see)
kraken2_rbind = kraken2_rbind[, c(1, ncol(kraken2_rbind), 3:ncol(kraken2_rbind)-1)]

# Remove any rows that are identified to anything other than family, genus,
# or species (not higher tax or subgroups)
```

```
kraken2_rbind = kraken2_rbind[(kraken2_rbind$tax == "F" | kraken2_rbind$tax == "G" | kraken2_rbind$tax ==

# Remove leading white space in tax names
kraken2_rbind$id = trimws(kraken2_rbind$id, "left")

# Rename dataframe so it can run through script
krak = kraken2_rbind

# Create list of unique combinations of mix ID and rep ID to properly merge
# kraken with mixes data at later point
krak_uniqueid <- data.frame(krak$mix.ID)
krak_uniqueid <- unique.data.frame(krak_uniqueid)
```

combine Kraken & sample metadata

Create a new aggregated datasheet, based on sample metadata, but which matches sample data back to the Kraken data so we can run analyses about probability of matching (both qualitative and quantitative). In particular this is taking account of sample "replicates" (originally including some that are not true replicates, but rather forward vs. reverse reads—this is not the case in the later 2020 data); also different Illumina lanes in the Emory Genome Center data.

1. Merge the unique krak IDs with mixes to subset the sample data to only the samples in the kraken dataset
2. Create a dataset for analysis of false negatives by only including taxa that are present in sample data
   - for this one, do it separately at each taxonomic level that we are assessing, i.e. species, genus, family
3. Create a dataset for analysis of false positives by including all taxa in kraken data

use 'mix.ID' as the variable to combine by...

```
## Merge sample and replicate IDs with mixes data
krak_mixes <- merge(mixes,krak_uniqueid, by.x="mix.ID", by.y="krak.mix.ID")

# create separate datasets for family, genus, and species level,
# both including and excluding false positives
# excluding false positives (focus on true positives / false negatives):
# truepos.krak.family =  merge(krak_mixes, filter(krak, tax == "F"),
# by.x = c("mix.ID", "family"), by.y = c("mix.ID", "id"), all.x=T)
truepos.krak.genus =  merge(krak_mixes, filter(krak, tax == "G"), by.x = c("mix.ID", "genus"), by.y = c("m
truepos.krak.species =  merge(krak_mixes, filter(krak, tax == "S"), by.x = c("mix.ID", "species"), by.y =

# including false positives:
# all.krak.family =  merge(krak_mixes, filter(krak, tax == "F"),
# by.x = c("mix.ID", "family"), by.y = c("mix.ID", "id"), all.x = T, all.y = T)
all.krak.genus =  merge(krak_mixes, filter(krak, tax == "G"), by.x = c("mix.ID", "genus"), by.y = c("mix.I
all.krak.species =  merge(krak_mixes, filter(krak, tax == "S"), by.x = c("mix.ID", "species"), by.y = c("m
```

remove Kraken reads below the contamination threshold

1. remove reads below the isolation / PCR negative control thresholds
   a) ideally, would base these on negative controls *for a given Illumina run* (two separate runs here, at UGA and Emory Genome center; we will need to keep those separate); but unfortunately there are not negative controls for the Emory Genome Center data; use the UGA thresholds for all the data (definitely not ideal, but working with what we have)
   b) this entails removing rows with k-mer counts below the threshold
   c) in the Illumina QIIME amplicon data, we did this separately for (entire) taxa below the threshold level, and for reads, because of the way that the data were formatted in a taxon-by-sample matrix; the Kraken output is already formatted in a way that is closer to `tidy` so we only need to do this step once here.
2. remove negative control rows (once the above is completed) so they don't interfere with the analysis

```
# Establish threshold for maximum number of reads in the negative controls
maxy = max(krak$hits[krak$mix.ID=="negat"])

# Identify rows that fall below this threshold for false negative analysis
#truepos.indexy_family = which(truepos.krak.family$hits <= maxy)
truepos.indexy_genus = which(truepos.krak.genus$hits <= maxy)
truepos.indexy_species = which(truepos.krak.species$hits <= maxy)

# No rows for false negative analysis with read level below threshold
# (can continue with analysis)

# Identify rows that fall below thresshold for false positive analysis
#all.indexy_family = which(all.krak.family$hits <= maxy)
all.indexy_genus = which(all.krak.genus$hits <= maxy)
all.indexy_species = which(all.krak.species$hits <= maxy)

# Remove these rows
#all.krak.family = all.krak.family[-all.indexy_family,]
all.krak.genus = all.krak.genus[-all.indexy_genus,]
all.krak.species = all.krak.species[-all.indexy_species,]

# Remove negative control rows
#all.krak.family = filter(all.krak.family, mix.ID!="negat")
all.krak.genus = filter(all.krak.genus, mix.ID!="negat")
all.krak.species = filter(all.krak.species, mix.ID!="negat")

# Clean up
rm(maxy, all.indexy_genus, all.indexy_species, truepos.indexy_genus, truepos.indexy_species)
# rm(all.indexy_family,truepos.indexy_family)
```

Kraken data: format for false negative and quantitative analyses

(will do false positives in next step; the data formatting at this step is relatively straightforward; again both of these are focused on true positives so easy to do them in the same code chunk)

1. Convert NAs in true positives / false negatives to 0s
2. Convert the "percentage hits" to a proportion - which will be the response variable in the quantitative models
3. Create a qualitative variable that equals 1 if that taxon was detected in the analysis (i.e. if proportion is greater than 0), and 0 if not detected (i.e. if proportion = 0).

```
# create quantitative variable by simply dividing the percentage hits
# (perc.hit) by 100:
#truepos.krak.family$quant.family = truepos.krak.family$perc.hit/100
truepos.krak.genus$quant.genus = truepos.krak.genus$perc.hit/100
truepos.krak.species$quant.species = truepos.krak.species$perc.hit/100

# for taxa that were not detected, NAs are currently present.
# need to change this to 0 for quantitative variable before running analysis.
# truepos.krak.family$quant.family =
#   ifelse(is.na(truepos.krak.family$quant.family), 0,
#          truepos.krak.family$quant.family)
truepos.krak.genus$quant.genus = ifelse(is.na(truepos.krak.genus$quant.genus), 0, truepos.krak.genus$quant
truepos.krak.species$quant.species = ifelse(is.na(truepos.krak.species$quant.species), 0, truepos.krak.spe

#Create qualitative variable based on quantitative variable
#truepos.krak.family$qual.family =
# ifelse(truepos.krak.family$quant.family > 0, 1, 0)
```

```
truepos.krak.genus$qual.genus = ifelse(truepos.krak.genus$quant.genus > 0, 1, 0)
truepos.krak.species$qual.species = ifelse(truepos.krak.species$quant.species > 0, 1, 0)
```

combine amplicon and Kraken data for false negative analysis

Using the kraken data formatted for true positive/false negative analysis ("truepos.krak"), rbind with amplicon data.

```
# Add "K-" to beginning of sample number of kraken dataset
# and add variable "source" to indicate that data is from kraken dataset
# truepos.krak.family$sample <-
#   paste("K-", truepos.krak.family$krak.sample, sep="")
truepos.krak.genus$sample <-
  paste("K-", truepos.krak.genus$krak.sample, sep="")
truepos.krak.species$sample <-
  paste("K-", truepos.krak.species$krak.sample, sep="")

#Add "source" variable to indicate if data is WGS or amplicon
#truepos.krak.family$source = "krak"
truepos.krak.genus$source = "krak"
truepos.krak.species$source = "krak"
amp_all$source = "amp"

# Concatenate WGS and amplicon data using mapply so that
# mismatching variable names of the amplicon data will be ignored
# 'fn' appended at the end of the variable names signifies "false negatives"

# krakamp_rbc_family_fn <-
# as.data.frame(mapply(c, truepos.krak.family[,c("mix.ID", "family", "sample",
#  "qual.family", "source")],
# amp_all[,c("mix.ID","family","sample", "qual.family.rbcL", "source")]))

# krakamp_its_family_fn <-
# as.data.frame(mapply(c, truepos.krak.family[,c("mix.ID", "family", "sample",
# "qual.family", "source")],
# amp_all[,c("mix.ID","family","sample", "qual.family.ITS", "source")]))

krakamp_rbc_genus_fn <- as.data.frame(mapply(c, truepos.krak.genus[,c("mix.ID", "genus", "sample", "qual.g

  # make `qual.genus` numerical:
  krakamp_rbc_genus_fn$qual.genus = as.numeric(krakamp_rbc_genus_fn$qual.genus)

krakamp_its_genus_fn <- as.data.frame(mapply(c, truepos.krak.genus[,c("mix.ID", "genus", "sample", "qual.g

  # make `qual.genus` numerical:
  krakamp_its_genus_fn$qual.genus = as.numeric(krakamp_its_genus_fn$qual.genus)

krakamp_rbc_species_fn <- as.data.frame(mapply(c, truepos.krak.species[,c("mix.ID", "species", "sample", "

  # make `qual.species` numerical:
  krakamp_rbc_species_fn$qual.species = as.numeric(krakamp_rbc_species_fn$qual.species)

krakamp_its_species_fn <- as.data.frame(mapply(c, truepos.krak.species[,c("mix.ID", "species", "sample", "

  # make `qual.species` numerical:
  krakamp_its_species_fn$qual.species = as.numeric(krakamp_its_species_fn$qual.species)
```

format kraken data for false positive analysis

To analyze false positives in kraken data, need to have aggregated counts of the "true positive" and "false positive" reads by sample. These will be our "success" and "failure" numbers in a binomial mixed model.

1. If the merged sample data is N/A, this indicates the taxa is a false positive. Based on N/A values, create a variable indicating if taxa is true or false positive.
2. Remove taxa that are in sample data but not kraken data (false negatives) - not considered for this analysis
    - while perhaps counterintuitive, need to do this because in this analysis we are using counts of "true positive" reads vs. "false positive" reads, i.e. only using reads that exist, not reads that should have existed but do not. Those are dealt with in our false negative analysis.
3. Add "K" to the sample names to distinguish from the samples in the amplicon data
4. Aggregate the counts per sample ID by "true positive" and "false positive"

In addition :

1. removed *Zea mays* and mix5 samples (don't match with metadata; mix5 has *Zea* in it)
2. matched false positive data with metadata so that sample complexity analyses can be run down the line
3. one consideration is that since we are aggregating all of the reads for a particular sample into one row, we have to figure out how to deal with rarity / commonness of taxa within a sample.
    - to do that, we use the *minimum* % of any taxon within the sample as a measure of rarity

```
# For false positive analysis, create variable indicating if taxa if "false positive" or "true positive"
#all.krak.family$type <- ifelse(is.na(all.krak.family$question.1), "false_pos", "true_pos")
all.krak.genus$type <- ifelse(is.na(all.krak.genus$question.1), "false_pos", "true_pos")
all.krak.species$type <- ifelse(is.na(all.krak.species$question.1), "false_pos", "true_pos")

# If hits = N/A, indicates a false negative (in sample data but not in Kraken data).
# Remove because we are not considering for this analysis
# (for the future, probably easier with `filter` in `dplyr` package)
#all.krak.family <- all.krak.family[-which(is.na(all.krak.family$sample.id)),]
#all.krak.genus <- all.krak.genus[-which(is.na(all.krak.genus$sample.id)),]
#all.krak.species <- all.krak.species[-which(is.na(all.krak.species$sample.id)),]

#ABOVE COMMENTED OUT BECAUSE THERE WERE NO FALSE NEGATIVES AND CODE WAS CAUSING ISSUES

# Add "K_" to sample name
#all.krak.family$sample <- paste("K_",all.krak.family$sample, sep="")
all.krak.genus$sample <- paste("K_",all.krak.genus$sample, sep="")
all.krak.species$sample <- paste("K_",all.krak.species$sample, sep="")

# Aggregate counts by mix.ID, sample.ID, and type
#agg.krak.family <- all.krak.family %>%
          #select(mix.ID, family, sample, type, hits) %>%
          #group_by(mix.ID, sample, rep.ID, type) %>%
          #summarize(total_hits = sum(hits))
agg.krak.genus <- all.krak.genus %>%
          select(mix.ID, genus, sample, type, hits) %>%
          group_by(mix.ID, sample, type) %>%
          summarize(total_hits = sum(hits))
```

```
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
```

```
agg.krak.species <- all.krak.species %>%
          select(mix.ID, species, sample, type, hits) %>%
          group_by(mix.ID, sample, type) %>%
          summarize(total_hits = sum(hits))
```

```
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
```

```
# Convert to wide format with one column for true positive hits and one column for false positive hits
#agg.krak.family <- spread(agg.krak.family, key=type, value=total_hits)
agg.krak.genus <- spread(agg.krak.genus, key=type, value=total_hits)
agg.krak.species <- spread(agg.krak.species, key=type, value=total_hits)

# If positives are N/A, set to equal 0
#agg.krak.family$true_pos = ifelse(is.na(agg.krak.family$true_pos),0, agg.krak.family$true_pos)
agg.krak.genus$true_pos = ifelse(is.na(agg.krak.genus$true_pos),0, agg.krak.genus$true_pos)
agg.krak.species$true_pos = ifelse(is.na(agg.krak.species$true_pos),0, agg.krak.species$true_pos)
#agg.krak.family$false_pos = ifelse(is.na(agg.krak.family$false_pos),0, agg.krak.family$false_pos)
agg.krak.genus$false_pos = ifelse(is.na(agg.krak.genus$false_pos),0, agg.krak.genus$false_pos)
agg.krak.species$false_pos = ifelse(is.na(agg.krak.species$false_pos),0, agg.krak.species$false_pos)

# as set up currently, the false positive datasets ('agg.krak.family', 'agg.krak.genus', 'agg.krak.species
# do not have the needed components to replicate the analysis we used for false negatives
# specifically, we need to have data on the facets of samples complexity:
    # species richness, taxonomic relatedness, and rarity
# and we also need the data subset component ('sub' vs. 'all')

# first, remove Zea mays and mix5 (which had Zea in it):
#agg.krak.family = filter(agg.krak.family, mix.ID != "Z.may" & mix.ID != "mix5")
#agg.krak.genus = filter(agg.krak.genus, mix.ID != "Z.may" & mix.ID != "mix5")
#agg.krak.species = filter(agg.krak.species, mix.ID != "Z.may" & mix.ID != "mix5")

# now, add the needed metadata

# don't want the aggregated data to be repeated for each taxon in a mix
# and that is the way the 'mixes' dataframe is set up
# in addition, to assess rarity, different taxa have different proportions
# thus, summarize 'mixes' to have the *minimum* value of 'pollen.grain.proportion' for each mix

mixy = mixes
mixy = mixy %>% select(-family, -genus, -species) %>% group_by(mix.ID) %>% summarize_all(min)

# now put them together
# via a merge of the false positive datasets with the 'mixes' (sample metadata)
# (didn't use 'left_join' because of conversion of factors to character)
#agg.krak.family = merge(agg.krak.family, mixy, by = "mix.ID")
agg.krak.genus = merge(agg.krak.genus, mixy, by = "mix.ID")
agg.krak.species = merge(agg.krak.species, mixy, by = "mix.ID")

# clean up... or don't
# don't remove 'mixy' just yet as we'll employ it again below for the false-positive amplicon data
```

false positives: format amplicon data

This part is one of the more challenging components of the formatting. It is based on the six amplicon data files (one for each marker and taxonomic level), formatted as QIIME matrices such that taxa comprise the rows and samples comprise the columns.

we need to:

- `melt` each matrix into a single row for each taxon / mix / sample combination, reporting the number of reads for each
  - remove combos for which there were zero reads (not strictly necessary since we will later just add up all the true and false positives, but makes everything more compact)
  - also a good time to remove mix #5 and also *Zea* samples from the amplicon data for consistencey

- merge each melted dataframe with with the sample metadata
- classify reads (“hits”) into false positives and false negatives, on a replicate-by-replicate basis
- aggregate (sum) false positive and false negative reads for each sample and replicate
- merge sample metadata in with the amplicon data

```
#Create list of amplicon data frames
amp_data <- list(amp_its_genus, amp_its_species, amp_rbc_genus, amp_rbc_species)

# create a function for *melting* the amplicon QIIME matrices, with a row for each...
# ...taxon / mix / sample combination, reporting the number of reads for each
amp_convert <- lapply(amp_data, function(x){
  #convert first column name to "taxa"
  names(x)[1] <- "taxa"
  #convert from wide to long format
  x <- melt(x, id.vars="taxa", value.name="hits")
  #separate mix ID and sample ID
  x[,4:5] <- colsplit(x$variable, "_", c("mix.ID", "sample"))
  #remove variable "id"
  x <- x[,-2]
  # remove any rows for which the "hits" count == 0
  # there were no reads, so we don't need to include them
  # also a good point to remove Zea mays single-species samples
  # and also mix #5 which includes Zea
  x <- filter(x, hits!=0 & mix.ID != "Z.mays" & mix.ID != "mix5")
})
```

```
## Warning in melt(x, id.vars = "taxa", value.name = "hits"): The melt generic
## in data.table has been passed a data.frame and will attempt to redirect to the
## relevant reshape2 method; please note that reshape2 is deprecated, and this
## redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(x). In the next version, this warning will become
## an error.

## Warning in melt(x, id.vars = "taxa", value.name = "hits"): The melt generic
## in data.table has been passed a data.frame and will attempt to redirect to the
## relevant reshape2 method; please note that reshape2 is deprecated, and this
## redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(x). In the next version, this warning will become
## an error.

## Warning in melt(x, id.vars = "taxa", value.name = "hits"): The melt generic
## in data.table has been passed a data.frame and will attempt to redirect to the
## relevant reshape2 method; please note that reshape2 is deprecated, and this
## redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(x). In the next version, this warning will become
## an error.

## Warning in melt(x, id.vars = "taxa", value.name = "hits"): The melt generic
## in data.table has been passed a data.frame and will attempt to redirect to the
## relevant reshape2 method; please note that reshape2 is deprecated, and this
## redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(x). In the next version, this warning will become
## an error.
```

```
# apply the melting function to each of the 6 datasets
# convert them into data frames at the same time
# (in the future, better to do this with `lapply` or similar)
#amp_its_family <- as.data.frame(amp_convert[1])
amp_its_genus <- as.data.frame(amp_convert[1])
amp_its_species <- as.data.frame(amp_convert[2])
#amp_rbc_family <- as.data.frame(amp_convert[4])
amp_rbc_genus <- as.data.frame(amp_convert[3])
amp_rbc_species <- as.data.frame(amp_convert[4])

# merge with sample metadata (`mixes`) by mixID and sampleID
#amp_its_family_mix <- merge(amp_its_family, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "family"),
amp_its_genus_mix <- merge(amp_its_genus, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "genus"), all.
amp_its_species_mix <- merge(amp_its_species, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "species")
#amp_rbc_family_mix <- merge(amp_rbc_family, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "family"),
amp_rbc_genus_mix <- merge(amp_rbc_genus, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "genus"), all.
amp_rbc_species_mix <- merge(amp_rbc_species, mixes, by.x=c("mix.ID", "taxa"), by.y=c("mix.ID", "species")

#Rows with mixes variables that are "NA" are false positive taxa. Create a data frame that indicates if ro
#amp_its_family_mix$type <- ifelse(is.na(amp_its_family_mix$question.1), "false_pos", "true_pos")
amp_its_genus_mix$type <- ifelse(is.na(amp_its_genus_mix$question.1), "false_pos", "true_pos")
amp_its_species_mix$type <- ifelse(is.na(amp_its_species_mix$question.1), "false_pos", "true_pos")
#amp_rbc_family_mix$type <- ifelse(is.na(amp_rbc_family_mix$question.1), "false_pos", "true_pos")
amp_rbc_genus_mix$type <- ifelse(is.na(amp_rbc_genus_mix$question.1), "false_pos", "true_pos")
amp_rbc_species_mix$type <- ifelse(is.na(amp_rbc_species_mix$question.1), "false_pos", "true_pos")

#New list of mixed datasets
amp_mixed <- list(amp_its_genus_mix, amp_its_species_mix, amp_rbc_genus_mix, amp_rbc_species_mix)


#Apply function to list
amp_summed <- lapply(amp_mixed, function(x){
  #summarize number of hits by mix.ID, sample, and type
  x <- x %>%
          select(mix.ID, taxa, hits, sample, type) %>%
          group_by(mix.ID, sample, type) %>%
          summarize(total_hits = sum(as.numeric(hits)))
  #Convert to wide format with one column for true positive hits and one column for false positive hits
  #x <- reshape(x, idvar="sample", timevar="type", direction="wide")
})
```

```
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
```

```
#amp_its_family_summ <- as.data.frame(amp_summed[1])
amp_its_genus_summ <- as.data.frame(amp_summed[1])
amp_its_species_summ <- as.data.frame(amp_summed[2])
#amp_rbc_family_summ <- as.data.frame(amp_summed[4])
amp_rbc_genus_summ <- as.data.frame(amp_summed[3])
amp_rbc_species_summ <- as.data.frame(amp_summed[4])

# Add "A" to beginning of sample ID - couldn't get this to work in the lapply function
#amp_its_family_summ$sample <- paste("A_", amp_its_family_summ$sample, sep="")
amp_its_genus_summ$sample <- paste("A_", amp_its_genus_summ$sample, sep="")
amp_its_species_summ$sample <- paste("A_", amp_its_species_summ$sample, sep="")
```

```
#amp_rbc_family_summ$sample <- paste("A_", amp_rbc_family_summ$sample, sep="")
amp_rbc_genus_summ$sample <- paste("A_", amp_rbc_genus_summ$sample, sep="")
amp_rbc_species_summ$sample <- paste("A_", amp_rbc_species_summ$sample, sep="")

# Convert to wide format with one column for true positive hits and one column for false positive hits - c
#amp_its_family_reshape <- spread(amp_its_family_summ, key=type, value=total_hits)
amp_its_genus_reshape <- spread(amp_its_genus_summ, key=type, value=total_hits)
amp_its_species_reshape <- spread(amp_its_species_summ, key=type, value=total_hits)
#amp_rbc_family_reshape <- spread(amp_rbc_family_summ, key=type, value=total_hits)
amp_rbc_genus_reshape <- spread(amp_rbc_genus_summ, key=type, value=total_hits)
amp_rbc_species_reshape <- spread(amp_rbc_species_summ, key=type, value=total_hits)

#If positives are N/A, set to equal 0
#amp_its_family_reshape$true_pos = ifelse(is.na(amp_its_family_reshape$true_pos),0, amp_its_family_reshape
amp_its_genus_reshape$true_pos = ifelse(is.na(amp_its_genus_reshape$true_pos),0, amp_its_genus_reshape$tru
amp_its_species_reshape$true_pos = ifelse(is.na(amp_its_species_reshape$true_pos),0, amp_its_species_resha
#amp_rbc_family_reshape$true_pos = ifelse(is.na(amp_rbc_family_reshape$true_pos),0, amp_rbc_family_reshape
amp_rbc_genus_reshape$true_pos = ifelse(is.na(amp_rbc_genus_reshape$true_pos),0, amp_rbc_genus_reshape$tru
amp_rbc_species_reshape$true_pos = ifelse(is.na(amp_rbc_species_reshape$true_pos),0, amp_rbc_species_resha

#amp_its_family_reshape$false_pos = ifelse(is.na(amp_its_family_reshape$false_pos),0, amp_its_family_resha
amp_its_genus_reshape$false_pos = ifelse(is.na(amp_its_genus_reshape$false_pos),0, amp_its_genus_reshape$fa
amp_its_species_reshape$false_pos = ifelse(is.na(amp_its_species_reshape$false_pos),0, amp_its_species_resh
#amp_rbc_family_reshape$false_pos = ifelse(is.na(amp_rbc_family_reshape$false_pos),0, amp_rbc_family_resha
amp_rbc_genus_reshape$false_pos = ifelse(is.na(amp_rbc_genus_reshape$false_pos),0, amp_rbc_genus_reshape$fa
amp_rbc_species_reshape$false_pos = ifelse(is.na(amp_rbc_species_reshape$false_pos),0, amp_rbc_species_resh


# merge these data with the sample metadata
# use 'mixy' from above which is the sample metadata summarized by sample
# (not by taxon within sample, as for 'mixes')
#amp_its_family_reshape = merge(amp_its_family_reshape, mixy, by = "mix.ID")
amp_its_genus_reshape = merge(amp_its_genus_reshape, mixy, by = "mix.ID")
amp_its_species_reshape = merge(amp_its_species_reshape, mixy, by = "mix.ID")
#amp_rbc_family_reshape = merge(amp_rbc_family_reshape, mixy, by = "mix.ID")
amp_rbc_genus_reshape = merge(amp_rbc_genus_reshape, mixy, by = "mix.ID")
amp_rbc_species_reshape = merge(amp_rbc_species_reshape, mixy, by = "mix.ID")

# clean up
rm(mixy)
```

combine amplicon and Kraken data for false positive analysis

Combine the WGS with amplicon data by creating a dataframe showing the number of true positive and the number of false positive reads by source, mix ID, and sample ID. Straightforward. Updated Dec 2020 to also take out *Zea* (this had previously been done in a subsequent step).

```
# Add column "source" to amplicon and kraken data
#amp_its_family_reshape$source = "amp"
amp_its_genus_reshape$source = "amp"
amp_its_species_reshape$source = "amp"
#amp_rbc_family_reshape$source = "amp"
amp_rbc_genus_reshape$source = "amp"
amp_rbc_species_reshape$source = "amp"

#agg.krak.family$source = "krak"
```

```
agg.krak.genus$source = "krak"
agg.krak.species$source = "krak"

# Merge kraken and amplicon data
#krakamp_its_family = rbind(subset(amp_its_family_reshape, mix.ID %in% agg.krak.family$mix.ID), as.data.fr
krakamp_its_genus = rbind(subset(amp_its_genus_reshape, mix.ID %in% agg.krak.genus$mix.ID), as.data.frame(
krakamp_its_species = rbind(subset(amp_its_species_reshape, mix.ID %in% agg.krak.species$mix.ID), as.data.
#krakamp_rbc_family = rbind(subset(amp_rbc_family_reshape, mix.ID %in% agg.krak.family$mix.ID), as.data.fr
krakamp_rbc_genus = rbind(subset(amp_rbc_genus_reshape, mix.ID %in% agg.krak.genus$mix.ID), as.data.frame(
krakamp_rbc_species = rbind(subset(amp_rbc_species_reshape, mix.ID %in% agg.krak.species$mix.ID), as.data.

# Take out Zea
#truepos.krak.family = filter(truepos.krak.family, genus!= "Zea")
truepos.krak.genus = filter(truepos.krak.genus, genus!="Zea")
truepos.krak.species = filter(truepos.krak.species, genus!="Zea")
```

Analysis

1. false negatives
   - WGS data alone, as a function of sample complexity
   - WGS data compared to amplicon data
2. false positives
   - WGS data alone, as a function of sample complexity
   - WGS data compared to amplicon data
3. quantitative matching
   - WGS data alone (sample input compared to sample output)
   - WGS data compared to amplicon data

false negatives

To assess if species richness, relatedness, and pollen grain proportion have a significant effect on the ability of the empirical WGS to qualitatively detect the presence/absence of a species, use a binomial mixed effects model with species as a random effect and rep.ID nested in sample nested in mix.ID as a random effect.

For consistency, take out *Zea mays* and format data tables to have the same name as those in the mixed amplicon analysis.

sample complexity: false negatives

- the analysis approach here is based on binomial-errors mixed-effects models
- the **fixed effects** are sample complexity; we are assessing one facet of sample complexity per model
  - Question 1: **species richness** (1-9 species)
  - Question 2: **rarity** (actual proportion of grains this taxon has in a sample; from roughly half & half to < 1% of the rarer type)
  - Question 3: **taxonomic relatedness** (within genus to across broad clades in the seed plants)
- we separately assessed models that considered different **subsets of the data**:
  - just the mixes specifically set up for each of the questions above (smaller subset of the data)
  - all of the mixes (thus more data)
- we also ran separate models for matching at each of the **taxonomic levels:**
  - family, genus, and species
- the **random effects** are nested, with the highest level as `mix.ID` (i.e. the identity of the pollen mixture); then `sample` replicate within each mix and finally `rep.ID` (forward / reverse read within a sample, or Illumina lane for samples split across lanes)
  - we modeled these as random intercepts
  - the random effects were consistent and equivalent for all models
- there are 3 facets of sample complexity times 2 data subsets times 3 taxonomic levels = **18 total models**

  - set this up with a nested `for` loop; there are almost certainly cleaner and more elegant ways to do this (e.g. probably better to use `formula` objects and to vectorize)

'==========

- **update Dec 2020: NO ANALYSES POSSIBLE**
  - we found true positives for **all** samples in the shotgun data using the Kraken analysis pipeline
  - (the one exception was *Broussonetia papifera*, which apparently was a sequencing failure with very very low overall read rates; but even for that species, we did actually get a match, it is just that the read rates were so low that they did not pass our contamination threshold and were removed)
  - thus, statistical analyses are not possible since essentially 0% of samples had false negatives / 100% true positives
  - because of that, we set the below code chunk to `eval = F` (i.e., not run)

'==========

```
# set up the three factors by which we are running the models
taxon = c("species", "genus")
datasubset = c("sub", "all") # whether we are using the designated subset of data designed for the questio
question = c("spp.rich", "relatedness", "pollen.grain.proportion")

# calculate total number of models
total = length(taxon)*length(datasubset)*length(question)

# first set up a table for the results with a number of entries equal to the 'total' variable above (18):
results.table = data.frame(question = rep(NA,total), taxon  = rep(NA,total), data.subset  = rep(NA,total),

# keep track of which row of the table to record in:
tracker = 1

# EXAMPLE FORMULA:
# Krak.Q1.species.all = glmer(qual.species.rbcL ~ spp.rich + (1|mix.ID) + (1|species), family = binomial,

# 'for' loop:

for(q in 1:3) { # 'question': response variables for Q1 / Q2 / Q3
    for(k in 1:2){ # 'taxon': species, genus
      for(l in 1:2) { # 'datasubset': sub or all
        # first, name the analysis:
        namer = paste("Krak.Q", q, ".", taxon[k], ".", datasubset[l], sep = "")
        # second, set which taxonomic data to use:
        data.to.use = paste("truepos.krak.", taxon[k], sep = "")
        # third, set up the data subset
        subster = paste("data.sub = filter(", data.to.use, ", question.1 == ", q, " | question.2 == ", q,
        eval(parse(text = subster)) # probably not the most efficient thing ever...
        # fourth, set whether or not data subset is used (vs. all data)
        if(datasubset[l]=="sub") {data.to.use = "data.sub"} # i.e., doesn't change if all data are to be u
        # fifth, set up mixed-effects model:
        mixed = paste(namer, " = try(suppressWarnings(glmer(qual.", taxon[k],
          " ~ ",  question[q], " + (1|mix.ID) + (1|", taxon[k], "), family = binomial,
          data = ", data.to.use, ", control = glmerControl(optimizer=\"bobyqa\"))))", sep = "")
        # sixth, evaluate the mixed-effects model
        eval(parse(text = mixed))
            # # eighth, print summary of model [SKIP FOR NOW]
            # summarizer = paste("print(summary(", namer, "))", sep = "")
            # eval(parse(text = summarizer)) # print summary of the mixed-effects model

        ## extract p-value
```

```
        # (this would probably be easier using the 'broom.mixed' package?)
        # example: coef(summary(Q3.genus.ITS.all))[2,4]
        pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
        eval(parse(text = pvaller))

        # extract convergence failures
        converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
        converg = eval(parse(text = converger))
        converg.return = ifelse(length(converg)==1, "ERROR!!", "")

        # record results in table
        results.table[tracker,1] = question[q]
        results.table[tracker,2] = taxon[k]
        results.table[tracker,3] = datasubset[l]
        results.table[tracker,4] = namer
        results.table[tracker,5] = pval
        results.table[tracker,6] = nrow(eval(parse(text = data.to.use)))
        results.table[tracker,7] = converg.return

        # advance tracker
        tracker = tracker + 1
      }
    }
  }


# display results table
# note that the 'kable' function is part of the 'knitr' package and `kable_styling` is from the `kableExtr
kable(results.table) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

Amplicon vs. Shotgun comparison: false negatives

This analysis uses binomial mixed models to assess if using amplicon vs. WGS has an effect on the ability to qualitatively detect true positives vs. false negatives. This analysis is similar to the qualitative mixed models to assess the effect of species richness, pollen grain proportion, etc. on the ability to qualitatively detect the correct taxa, but instead using "source" as the (sole) fixed effect.

```
taxon = c("species", "genus")
marker = c("its", "rbc")


# first set up a table for the results:
rowz = length(taxon)*length(marker)

krakamp.fn.results.table = data.frame(taxon  = rep(NA,rowz), marker  = rep(NA,rowz), model.name = rep(NA,r

# keep track of which row of the table to record in:
tracker = 1

# # EXAMPLE FORMULA
# its.family = glmer(qual.family ~ source + (1|mix.ID/sample) + (1|family), family = binomial, data = krak

# response variables relating to each of the three questions (column names in data)
```

```
for(q in 1:2) { # 'marker': its or rbc
    for(k in 1:2){ # 'taxon': species or genus
        # first, name the analysis (i.e. name the result object):
        namer = paste(marker[q], ".", taxon[k], sep = "")

        # second, set which taxonomic data to use:
        data.to.use = paste("krakamp_", marker[q], "_", taxon[k],"_fn", sep = "")

        # third, set up mixed-effects model:
        mixed = paste(namer, " = suppressWarnings(glmer(qual.", taxon[k], " ~ source + (1|mix.ID/sample) +

        # fourth, evaluate the mixed-effects model
        eval(parse(text = mixed))
            #fifth, print summary of model [SKIP FOR NOW]
            #summarizer = paste("print(summary(", namer, "))", sep = "")
            #eval(parse(text = summarizer)) # print summary of the mixed-effects model

        # fifth, extract the results
        ## extract p-value
        # example: coef(summary(its.family))[2,4]
        pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
        eval(parse(text = pvaller))

        # extract convergence failures
        converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
        converg = eval(parse(text = converger))
        converg.return = ifelse(length(converg)==1, "ERROR!!", "")

        # record results in table
        krakamp.fn.results.table[tracker,1] = taxon[k]
        krakamp.fn.results.table[tracker,2] = marker[q]
        krakamp.fn.results.table[tracker,3] = namer
        krakamp.fn.results.table[tracker,4] = pval
        krakamp.fn.results.table[tracker,5] = nrow(eval(parse(text = data.to.use)))
        krakamp.fn.results.table[tracker,6] = converg.return

        # advance tracker
        tracker = tracker + 1
      }
    }
```

by marker (*rbcL* vs ITS2)—Amplicon vs. Shotgun false negatives

```
## boundary (singular) fit: see ?isSingular
## boundary (singular) fit: see ?isSingular
```

```
# display results table
# note that the 'kable' function is part of the 'knitr' package and `kable_styling` is from the `kableExtr
kable(krakamp.fn.results.table) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

taxon	marker	model.name	p.val	n	warning.msg
species	its	its.species	0.0001273	299	
genus	its	its.genus	0.0002684	299	
species	rbc	rbc.species	0.0000004	299	
genus	rbc	rbc.genus	0.0000000	299	ERROR!!

Results interpretation updated Dec 2020

- **take-home message:** shotgun / Kraken outperforms amplicon approach in terms of identifying true positives
- not surprising since the shotgun approach generated a true positive match for essentially every taxon in our samples
- this is true irrespective of amplicon marker (*rbcL* or ITS2) or taxonomic resolution (genus vs. species)

markers combined—Amplicon vs. Shotgun false negatives This is a repeat of the analysis above, but with ITS2 and *rbcL* combined

```
#Create new qualitative variable in amplicon dataset that equals 1 if ITS and/or rbcL equals 1, 0 if else

#amp_all$qual.family = ifelse(amp_all$qual.family.ITS == 1 | amp_all$qual.family.rbcL == 1, 1, 0)
amp_all$qual.genus = ifelse(amp_all$qual.genus.ITS == 1 | amp_all$qual.genus.rbcL == 1, 1, 0)
amp_all$qual.species = ifelse(amp_all$qual.species.ITS == 1 | amp_all$qual.species.rbcL == 1, 1, 0)

#Combine kraken and amplicon data
#krakamp_family_fn <- rbind(truepos.krak.family[,c("mix.ID", "family", "sample", "qual.family", "source")]

krakamp_genus_fn <- rbind(truepos.krak.genus[,c("mix.ID", "genus", "sample", "qual.genus", "source")], amp

krakamp_species_fn <- rbind(truepos.krak.species[,c("mix.ID", "species", "sample", "qual.species", "source


taxon = c("species", "genus")

# first set up a table for the results:
# I will set this up with 18 entries
krakamp.fn.results.combined = data.frame(taxon  = rep(NA,2), model.name = rep(NA,2), p.val  = rep(1.000001

# keep track of which row of the table to record in:
tracker = 1

# # EXAMPLE FORMULA
# its.family = glmer(qual.family ~ source + (1|mix.ID/sample) + (1|family), family = binomial, data = krak

# response variables relating to each of the three questions (column names in data)

    for(k in 1:2){ # 'taxon': species, genus
        # first, name the analysis:
        namer = taxon[k]

        # second, set which taxonomic data to use:
        data.to.use = paste("krakamp_", taxon[k],"_fn", sep = "")

        # third, set up mixed-effects model:
        mixed = paste(namer, " = suppressWarnings(glmer(qual.", taxon[k],
          " ~ source + (1|mix.ID/sample) + (1|", taxon[k], "), family = binomial,
          data = ", data.to.use, ", control = glmerControl(optimizer=\"bobyqa\")))", sep = "")
        # fourth, evaluate the mixed-effects model
        eval(parse(text = mixed))

        # fifth, extract model information
        ## extract p-value
        # example: coef(summary(its.family))[2,4]
        pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
        eval(parse(text = pvaller))
```

```
    # extract convergence failures
    converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
    converg = eval(parse(text = converger))
    converg.return = ifelse(length(converg)==1, "ERROR!!", "")

    # record results in table
    krakamp.fn.results.combined[tracker,1] = taxon[k]
    krakamp.fn.results.combined[tracker,2] = namer
    krakamp.fn.results.combined[tracker,3] = pval
    krakamp.fn.results.combined[tracker,4] = nrow(eval(parse(text = data.to.use)))
    krakamp.fn.results.combined[tracker,5] = converg.return

    # advance tracker
    tracker = tracker + 1
  }
```

```
## boundary (singular) fit: see ?isSingular
```

```
# display results table
# note that the 'kable' function is part of the 'knitr' package and `kable_styling` is from the `kableExtr
kable(krakamp.fn.results.combined) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

taxon	model.name	p.val	n	warning.msg
species	species	0.0027829	294	
genus	genus	0.4333710	294	

updated results Dec 2020

- **take-home message:** shotgun approach works better than the two amplicon markers combined, when focusing on species-level resolution; but there is no statistically significant difference when focusing on the genus level
- makes sense in light of the results above
- at the genus level, using both amplicon markers gets you true positives essentially all of the time, so it isn't surprising that there wasn't a difference

false positives

sample complexity: false positives

this analysis has the same structure as for the false negatives analysis of sample complexity, again examining 3 taxonomic levels (family / genus / species) for three dimensions of sample complexity (species richness / taxonomic relatedness / rarity).

- **update Dec 2020:** we had intended to also run this for each of two datasets (the subsample specifically designed for the sample complexity component of interest and the entire dataset), but did not do so, I (Berry) think that when we merged the forward and reverse reads we ended up with too few samples to do the subsets (even when accounting previously for the lack of independence of forward and reverse reads)
- take home is that all analyses are only from all of the data, not the subset

Some difference from the false negatives analysis:

- the response variable needs to change
  - formerly (for false negatives) it was a 0/1 column which had info on whether or not each taxon within a mixture was present or not
  - in this analysis, the response variable is two columns, the count of true positive reads and the count of false positive reads, which are bound together with `cbind`
- the random effect for taxon is removed, since data are not aggregated taxonomically (instead they are aggregated by sample, i.e. replicate within each mix)

- since the data are aggregated, for pollen grain proportion the *minimum* proportion (i.e. the proportion of the smallest quantity of pollen in any given mix) was used as the explanatory variable

```
# set up the three factors by which we are running the models
taxon = c("species", "genus")
# datasubset = c("sub", "all") # whether we are using the designated subset of data designed for the quest
# taking this out, Dec 2020, because data subsets are too small to be meaningful and may be throwing other
question = c("spp.rich", "relatedness", "pollen.grain.proportion")

# calculate total number of models
total = length(taxon)*length(question) # was previously also *length(datasubset)

# first set up a table for the results with a number of entries equal to the 'total' variable above (18):
results.table.falsepos = data.frame(question = rep(NA,total), taxon  = rep(NA,total), model.name = rep(NA,
# previously also included 'data.subset  = rep(NA,total)'

# keep track of which row of the table to record in:
tracker = 1

# EXAMPLE FORMULA:
# Krak.Q1.species.all = glmer(qual.species.rbcL ~ spp.rich + (1|mix.ID/sample) + (1|species), family = bin

# 'for' loop:

for(q in 1:3) { # 'question': response variables for Q1 / Q2 / Q3
  for(k in 1:2){ # 'taxon': species, genus
#    for(l in 1:2) { # 'datasubset': sub or all
      # first, name the analysis:
      namer = paste("Krak.falsepos.Q", q, ".", taxon[k], ".", sep = "") # had also included 'datasubset[l]
      # second, set which taxonomic data to use:
      data.to.use = paste("agg.krak.", taxon[k], sep = "")
      # third, set up the data subset
      subster = paste("data.sub = filter(", data.to.use, ", question.1 == ", q, " | question.2 == ", q, "
      eval(parse(text = subster)) # probably not the most efficient thing ever...
      # fourth, set whether or not data subset is used (vs. all data)
      # if(datasubset[l]=="sub") {data.to.use = "data.sub"} # i.e., doesn't change if all data are to be u
        # above taken out when datasubsets removed
      # fifth, set up mixed-effects model:
      mixed = paste(namer, " = suppressWarnings(glmer(cbind(false_pos, true_pos) ~ ",  question[q], " + (1
                    data = ", data.to.use, ", control = glmerControl(optimizer=\"bobyqa\", optCtrl = list(

      # NOTE (update Dec 2020): random effects specification changed from
      # (1|mix.ID/sample) to
      # (1|mix.ID) given that there is only one replicate per sample now
      # this is because of merging forward and reverse reads
      # this did not change anything at all results-wise (exact same *p*-values)

      # sixth, evaluate the mixed-effects model
      eval(parse(text = mixed))
      # # eighth, print summary of model [SKIP FOR NOW]
      # summarizer = paste("print(summary(", namer, "))", sep = "")
      # eval(parse(text = summarizer)) # print summary of the mixed-effects model

      ## extract p-value
      # (this would probably be easier using the 'broom.mixed' package?)
      # example: coef(summary(Q3.genus.ITS.all))[2,4]
      pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
```

```
      eval(parse(text = pvaller))

      # extract convergence failures
      converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
      converg = eval(parse(text = converger))
      converg.return = ifelse(length(converg)==1, "ERROR!!", "")

      # record results in table
      results.table.falsepos[tracker,1] = question[q]
      results.table.falsepos[tracker,2] = taxon[k]
      # results.table.falsepos[tracker,3] = datasubset[l]
      results.table.falsepos[tracker,3] = namer # was '[tracker, 4]'
      results.table.falsepos[tracker,4] = pval # was '[tracker, 5]' (etc for 6 and 7)
      results.table.falsepos[tracker,5] = nrow(eval(parse(text = data.to.use)))
      results.table.falsepos[tracker,6] = converg.return

      # advance tracker
      tracker = tracker + 1
    }
  }
# } # (was here for datasubset)

# display results table
# note that the 'kable' function is part of the 'knitr' package and `kable_styling` is from the `kableExtr
kable(results.table.falsepos) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

question	taxon	model.name	p.val	n	warning.msg
spp.rich	species	Krak.falsepos.Q1.species.	0.8957633	17	
spp.rich	genus	Krak.falsepos.Q1.genus.	0.3810926	17	
relatedness	species	Krak.falsepos.Q2.species.	0.5412559	17	
relatedness	genus	Krak.falsepos.Q2.genus.	0.0988828	17	
pollen.grain.proportion	species	Krak.falsepos.Q3.species.	0.6783320	17	
pollen.grain.proportion	genus	Krak.falsepos.Q3.genus.	0.0887988	17	

Results interpretation: updated Dec 2020

- **take-home message:** we do not see any statistically significant relationships between any of the sample complexity components (species richness, relatedness, or minimum pollen grain proportions) and increasing false positive reads, assessed at either the genus or the species level
  - two of the results were marginally significant (with marginality set at $0.05 < p < 0.10$)
  - I SUGGEST NOT DISCUSSING THESE MARGINAL RESULTS—not worth it relative to our more substantive results, and ultimately they are not statistically significant
  - both were for genus-level data
  - pollen grain relatedness: while marginal, if you look at figure J you can see that while the CIs are tiny, there doesn't really seem to be any *meaningful* difference among relatedness categories
  - pollen grain proportion: while marginal, again, tiny CIs and a pattern that doesn't seem particularly meaningful, with slightly higher false positive rates in the middle range of the minimum proportions (hard to interpret—we were expecting more false positives when minimum proportions got lower)
- again, we did not analyze the data based on the "subsample" of data designed for each question, instead we used all data, otherwise we were left with sample sizes e.g. of 4 and 2 for many of the questions
- in contrast to previous results, we did not get any warning messages / models seemed to fit well; this could be in part due to the fact that we did not have within-sample random effects due to the merging of the forward and reverse reads

Amplicon vs. Shotgun comparison, false positives

This analysis uses binomial mixed models to assess if using amplicon vs. WGS approaches return different proportions of false positive results (relative to true positives).

Amplicon markers considered separately (*rbcL* and ITS2) First we will split out the analysis by amplicon marker (*rbcL* vs. ITS2) and by taxonomic level of identification (family / genus / species); next we will combine both amplicons but continue to assess matches at all three taxonomic levels.

```
taxon = c("species", "genus")
marker = c("its", "rbc")

# set up a data frame for the results:
# I will set this up with 6 rows (3 taxonomy levels x 2 markers)
rowz = length(taxon)*length(marker)
krakamp.results.table = data.frame(taxon  = rep(NA,rowz), marker  = rep(NA,rowz), p.val  = rep(1.000001,ro

# keep track of which row of the table to record in:
tracker = 1

# # EXAMPLE FORMULA
# glmer(cbind(true_pos,false_pos) ~ source + (1|mix.ID/sample), family = binomial, data = krakamp_its_spec


for(k in 1:2) { # 'taxon': species, genus, family
    for(l in 1:2){ # 'marker': ITS2 or rbcL
        # first, name the analysis:
        namer = paste(taxon[k], ".", marker[l], sep = "")

        # second, set which taxonomic data to use:
        data.to.use = paste("krakamp_", marker[l], "_", taxon[k], sep = "")

        # third, set up mixed-effects model:
        mixed = paste(namer, " = suppressWarnings(glmer(cbind(true_pos,false_pos) ~ source + (1|mix.ID/sam

        # fourth, evaluate the mixed-effects model
        eval(parse(text = mixed))

        ## extract p-value
        pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
        eval(parse(text = pvaller))

        # extract convergence failures
        converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
        converg = eval(parse(text = converger))
        converg.return = ifelse(length(converg)==1, "ERROR!!", "")

        # record results in table
        krakamp.results.table[tracker,1] = taxon[k]
        krakamp.results.table[tracker,2] = marker[l]
        krakamp.results.table[tracker,3] = pval
        krakamp.results.table[tracker,4] = nrow(eval(parse(text = data.to.use)))
        krakamp.results.table[tracker,5] = converg.return

        # advance tracker
        tracker = tracker + 1
      }
```

```
    }

# display results table
# note that the 'kable' function is part of the 'knitr' package, which I required at the very top of this
kable(krakamp.results.table) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

taxon	marker	p.val	n	warning.msg
species	its	0.9102740	54	
species	rbc	0.5067176	54	
genus	its	0.0000454	54	
genus	rbc	0.6917513	54	

updated results Dec 2020

- **take-home message:** somewhat surprisingly, with the new analysis these results have now changed and for the most part we don't see a difference between shotgun- and amplicon-based approaches in terms of false positives
- the one exception is for genus-level resolution, when shotgun data are compared to amplicon data from ITS2
- based on Figure 6, we are seeing statistically higher rates of false positives for the shotgun data relative to genus-level amplicon data using ITS2

amplicon markers combined for this analysis we need to format the data a little differently; the formatted amplicon data are in different data frames based on the marker (ITS2 vs. *rbcL*). Need to combine those two to generate a single amplicon data frame; this has to happen at each taxonomic level. While seemingly easy with `rbind`, we have to take the additional step of aggregating for each mix.id and sample.id the two amplicon markers (add the false positives and false negatives together). Still, this is very straightforward with `dplyr`.

```
# first combine the two amplicon datasets, aggregating counts for the same mix and sample IDs
# this has to happen at each taxonomic level
#ampkrak.family = rbind(amp_its_family_reshape, amp_rbc_family_reshape) %>% group_by(mix.ID, sample, rep.I

ampkrak.genus = rbind(amp_its_genus_reshape, amp_rbc_genus_reshape) %>% group_by(mix.ID, sample, source) %
```

```
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
```

```
ampkrak.species = rbind(amp_its_species_reshape, amp_rbc_species_reshape) %>% group_by(mix.ID, sample, sou
```

```
## `summarise()` regrouping output by 'mix.ID', 'sample' (override with `.groups` argument)
```

```
# then combine amplicon data with kraken data
#ampkrak.family = rbind(data.frame(ampkrak.family), data.frame(select(agg.krak.family, mix.ID, sample, rep
ampkrak.genus = rbind(data.frame(ampkrak.genus), data.frame(select(agg.krak.genus, mix.ID, sample, source,
ampkrak.species = rbind(data.frame(ampkrak.species), data.frame(select(agg.krak.species, mix.ID, sample, s

# set up a data frame for the results with 3 rows (by taxonomic level)
krakamp.results.table = data.frame(taxon  = rep(NA,2), p.val  = rep(1.000001,2), n  = rep(9999,2), warning

# keep track of which row of the table to record in:
tracker = 1

# # EXAMPLE FORMULA
# glmer(cbind(true_pos,false_pos) ~ source + (1|mix.ID/sample), family = binomial, data = krakamp_its_spec


for(k in 1:2) { # 'taxon': species, genus, family
        # first, name the analysis:
```

```
        namer = paste("krakamp.GLMM", taxon[k], sep = ".")

        # second, set which taxonomic data to use:
        data.to.use = paste("ampkrak", taxon[k], sep = ".")

        # third, set up mixed-effects model:
        mixed = paste(namer, " = suppressWarnings(glmer(cbind(true_pos,false_pos) ~ source + (1|mix.ID/sam

        # fourth, evaluate the mixed-effects model
        eval(parse(text = mixed))

        ## extract p-value
        pvaller = paste("pval <- coef(summary(", namer, "))[2,4]", sep = "")
        eval(parse(text = pvaller))

        # extract convergence failures
        converger = paste(namer, "@optinfo$conv$lme4$code", sep = "")
        converg = eval(parse(text = converger))
        converg.return = ifelse(length(converg)==1, "ERROR!!", "")

        # record results in table
        krakamp.results.table[tracker,1] = taxon[k]
        # krakamp.results.table[tracker,2] = marker[l]
        krakamp.results.table[tracker,2] = pval
        krakamp.results.table[tracker,3] = nrow(eval(parse(text = data.to.use)))
        krakamp.results.table[tracker,4] = converg.return

        # advance tracker
        tracker = tracker + 1
    }


# display results table
# note that the 'kable' function is part of the 'knitr' package, which I required at the very top of this
kable(krakamp.results.table) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

taxon	p.val	n	warning.msg
species	0.9694023	106	
genus	0.0993345	106	

*updated results Dec 2020**

- **take-home message:** consistently with the results for the amplicon markers split out, with the new analysis these results have now changed; we don't see a statistically significant difference between shotgun- and amplicon-based approaches in terms of false positives
- there is now a *marginally* significant result for genus-level resolution
- based on Figure 6, we are seeing statistically higher rates of false positives for the shotgun data relative to genus-level amplicon data

quantitative matching

To assess if pollen grain proportion is correlated to read proportion in the empirical WGS data, use mixed-effects models with mix.ID and species as random effects. Using the same dataset as for the false negatives analysis, i.e. data are formatted to have each taxon (truly present) within each sample representing a row.

There are two analyses:

1. WGS data only: quantitative matching
2. amplicon + WGS data: comparison on which gives us better quantitative matching

WGS data only: quantitative matching

Here the question is: do we see a statistical relationship between the proportion of pollen grains input into a sample and the proportion of read counts that are output? As described in the **Overview** section at the beginning, we do this with a linear mixed-effects model as we are looking for a linear relationship between the input pollen grain proportions, and the proportions of output sample reads. A logistic relationship, as would likely be predicted with a binomial-errors model, is not helpful to an end-user here even though the data are ultimately binomial in nature.

We tried specifying these models with intercept = 0. Unfortunately, we can't estimate an R^2 with a specification of zero intercept, at least with the `r2beta` function (not sure why). Either way, we ultimately didn't do this even though we could get the models to work.

```
# linear mixed-effects model

# ultimately want 3 analyses: {species, genus, and family}
# within each, we will do a separate analysis without the nested random effect of the 'replicate ID', and

# species
quant.species = lmer(quant.species ~ pollen.grain.proportion + (1|mix.ID) + (1|species), data = truepos.kr
#quant.species.nonest = lmer(quant.species ~ pollen.grain.proportion + (1|mix.ID) + (1|species), data = tr

# genus
quant.genus = lmer(quant.genus ~ pollen.grain.proportion + (1|mix.ID) + (1|genus), data = truepos.krak.gen
#quant.genus.nonest = lmer(quant.genus ~ pollen.grain.proportion + (1|mix.ID) + (1|genus), data = truepos.

#family
#quant.family = lmer(quant.family ~ pollen.grain.proportion + (1|mix.ID/krak.rep.ID) + (1|family), data =
#quant.family.nonest = lmer(quant.family ~ pollen.grain.proportion + (1|mix.ID) + (1|family), data = truep

#=======================================
# r-squared calculation at the family level
#r2.family = r2beta(quant.family)
#r2.family.nonest = r2beta(quant.family.nonest)

# r-squared calculation at the genus level
r2.genus = r2beta(quant.genus)
#r2.genus.nonest = r2beta(quant.genus.nonest)

# r-squared calculation at the species level
r2.species = r2beta(quant.species)
#r2.species.nonest = r2beta(quant.species.nonest)

#Merge the slope ("Estimate"), p-value ("Pr...t..") from the mixed models with the r-squared value for eac
#coefs_quant.family <- cbind((data.frame(coef(summary(quant.family)))["pollen.grain.proportion",c("Estimat
#coefs_quant.family.nonest <- cbind((data.frame(coef(summary(quant.family.nonest)))["pollen.grain.proporti

coefs_quant.genus <- cbind((data.frame(coef(summary(quant.genus)))["pollen.grain.proportion",c("Estimate",
#coefs_quant.genus.nonest <- cbind((data.frame(coef(summary(quant.genus.nonest)))["pollen.grain.proportion

coefs_quant.species <- cbind((data.frame(coef(summary(quant.species)))["pollen.grain.proportion",c("Estima
#coefs_quant.species.nonest <- cbind((data.frame(coef(summary(quant.species.nonest)))["pollen.grain.propor

#Rename row names
```

```
#row.names(coefs_quant.family) = "Family"
row.names(coefs_quant.genus) = "Genus"
row.names(coefs_quant.species) = "Species"

# row.names(coefs_quant.family.nonest) = "Family"
# row.names(coefs_quant.genus.nonest) = "Genus"
# row.names(coefs_quant.species.nonest) = "Species"

# rename column names
#colnames(coefs_quant.family) = c("Slope", "p-value", "R2")
colnames(coefs_quant.genus) = c("Slope", "p-value", "R2")
colnames(coefs_quant.species) = c("Slope", "p-value", "R2")

# colnames(coefs_quant.family.nonest) = c("Slope", "p-value", "R2")
# colnames(coefs_quant.genus.nonest) = c("Slope", "p-value", "R2")
# colnames(coefs_quant.species.nonest) = c("Slope", "p-value", "R2")

#Merge summary of coefficients into one dataset
coefs_summ <- rbind(coefs_quant.genus, coefs_quant.species)
# coefs_summ.nonest <- rbind(coefs_quant.family.nonest, coefs_quant.genus.nonest, coefs_quant.species.none

#Display table of summarized coefficients
kable(coefs_summ) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)
```

	Slope	p-value	R2
Genus	0.4559603	9e-07	0.6201248
Species	0.4550948	4e-07	0.6022033

```
# kable(coefs_summ.nonest) %>%
#   kable_styling(bootstrap_options = "striped", full_width = F)
```

Pollen grain proportion was highly significantly related to read proportion across all taxonomic levels, with relatively strong R^2 values of 0.60 and 0.62.

- this result stayed consistent with the updates in Dec 2020

Amplicon vs. WGS: quantitative matching

The basic idea here is to assess if WGS vs. amplicon data has better quantitative matching. This is at first glance more challenging than for the qualitative metrics, in which the directionality of the data tells the story completely: more true positives = good, and more false positives = bad. But for the quantitative analyses, having a higher proportion of matches than there are input proportion of pollen grains is bad, and having a lower proportion of matches than the input proportion of pollen grains is also bad.

We used two complementary approaches. The first compares the residuals of WGS vs. amplicon to the "true" model (slope = 1, intercept = 0). The second compares the R^2 values of unconstrained linear models between WGS and amplicon approaches.

FIRST A quantitative approach to this is to analyze the *residuals* of the input vs. output quantitative model, and see if there are differences in the *mean absolute values of the residuals* between WGS and amplicon approaches. Here, directionality has a meaning: smaller absolute residual values are better. There are at least two different approaches one could take: first is just to fit a linear model to the data (as we have done above); second is to fit a linear model with intercept = 0 and slope = 1, and use the residuals from that model.

The latter approach here seems better—ultimately we want the output proportion of sequence reads to match as closely to possible the input proportion of pollen grains (i.e. a linear relationship with intercept = 0 and slope = 1). To find the residuals of this relationship, we don't need to run a model at all; because the expected value of the

output proportion is the input proportion, we can just calculate the absolute value of (`input - output`) in a new column. Then we can use that residual value as the response variable in a new linear mixed-effects model, with the sole fixed effect being the data `source` (i.e. amplicon vs. WGS), but allowing us to use the random intercepts we specified in previous models (mix identity and taxonomic identity).

One thing to note about this approach is that we already know that the WGS data have more false positives, and that is likely to pull them further away from the "true" line with slope = 1 and intercept = 0.

SECOND Alternatively, another approach is to just directly compare the WGS and amplicon results in terms of R^2 values for the linear models. Higher R^2 values are better. This is somewhat qualitative, however: e.g. while $R^2 = 0.77$ is better than $R^2 = 0.70$, are those two values meaningfullly different? For example, it would be easy for a couple of values that were strongly off from the prediction to greatly alter the R^2 values. But if the results are strongly different between WGS and amplicon approaches, that will tell us something.

data prep for quant matching amplicon vs. WGS We need to do is to get the amplicon data formatted in a similar way to the WGS data for this analysis (with a percentage of hits for each taxon truly present in a sample, as well as the proportion of pollen grains of that taxon in that sample). For this, we will combine our original `amp_all` data with e.g. `truepos.krak.species` (and the corresponding files for genus and species)

```
# # first get a `rep.ID` variable into the `amp_all` dataset
# # so that there is a parallel with the kraken data
# amp_all$rep.ID = 1

# pull out only the relevant variables from the kraken data, for easier merging down the line:
#quant.krak.family = select(truepos.krak.family, mix.ID, family, pollen.grain.proportion, quant.family, so
quant.krak.genus = select(truepos.krak.genus, mix.ID, genus, pollen.grain.proportion, quant.genus, source)
quant.krak.species = select(truepos.krak.species, mix.ID, species, pollen.grain.proportion, quant.species,

# amplicon: pull out relevant variables, for easier `rbind` down the line
# rename `sample` to `rep.ID` since both represent sub-samples within a `mix.ID`
# ...but `sample` only meaningful for amplicon and `rep.ID` only meaningful for WGS
#quant.amp.family.its = select(amp_all, mix.ID, rep.ID = sample, family, pollen.grain.proportion, quant.fa
#quant.amp.family.rbcl = select(amp_all, mix.ID, rep.ID = sample, family, pollen.grain.proportion, quant.f
quant.amp.genus.its = select(amp_all, mix.ID, genus, pollen.grain.proportion, quant.genus = quant.genus.ITS
quant.amp.genus.rbcl = select(amp_all, mix.ID, genus, pollen.grain.proportion, quant.genus = quant.genus.rb
quant.amp.species.its = select(amp_all, mix.ID, species, pollen.grain.proportion, quant.species = quant.spe
quant.amp.species.rbcl = select(amp_all, mix.ID, species, pollen.grain.proportion, quant.species = quant.sp

# rbind 'em together:
#quant.krakamp.family.its = rbind(quant.amp.family.its, quant.krak.family)
#quant.krakamp.family.rbcl = rbind(quant.amp.family.rbcl, quant.krak.family)
quant.krakamp.genus.its = rbind(quant.amp.genus.its, quant.krak.genus)
quant.krakamp.genus.rbcl = rbind(quant.amp.genus.rbcl, quant.krak.genus)
quant.krakamp.species.its = rbind(quant.amp.species.its, quant.krak.species)
quant.krakamp.species.rbcl = rbind(quant.amp.species.rbcl, quant.krak.species)
```

residual-based analysis for quant matching amplicon vs. WGS

- calculate residuals (absolute value of difference between input pollen grain proportion and output proportion of sequence reads), as a new column
- then run a linear mixed-effects model, with `source` as the sole fixed effect

```
# calculate residuals
#quant.krakamp.family.its = quant.krakamp.family.its %>% mutate(resid = abs(pollen.grain.proportion - quan
#quant.krakamp.family.rbcl = quant.krakamp.family.rbcl %>% mutate(resid = abs(pollen.grain.proportion - qu
quant.krakamp.genus.its = quant.krakamp.genus.its %>% mutate(resid = abs(pollen.grain.proportion - quant.g
quant.krakamp.genus.rbcl = quant.krakamp.genus.rbcl %>% mutate(resid = abs(pollen.grain.proportion - quant
quant.krakamp.species.its = quant.krakamp.species.its %>% mutate(resid = abs(pollen.grain.proportion - qua
```

```
quant.krakamp.species.rbcl = quant.krakamp.species.rbcl %>% mutate(resid = abs(pollen.grain.proportion - q

# run models:
#quant.krakamp.family.its.lmer = lmer(resid ~ source + (1|mix.ID/rep.ID) + (1|family), data = quant.krakam
#quant.krakamp.family.rbcl.lmer = lmer(resid ~ source + (1|mix.ID/rep.ID) + (1|family), data = quant.kraka
quant.krakamp.genus.its.lmer = lmer(resid ~ source + (1|mix.ID) + (1|genus), data = quant.krakamp.genus.its
quant.krakamp.genus.rbcl.lmer = lmer(resid ~ source + (1|mix.ID) + (1|genus), data = quant.krakamp.genus.rb
quant.krakamp.species.its.lmer = lmer(resid ~ source + (1|mix.ID) + (1|species), data = quant.krakamp.speci
quant.krakamp.species.rbcl.lmer = lmer(resid ~ source + (1|mix.ID) + (1|species), data = quant.krakamp.spec

# build a table of results, using `broom.mixed`
#quant.krakamp.table = tidy(quant.krakamp.family.its.lmer)[2,c(3,4,8)]
#quant.krakamp.table = rbind(quant.krakamp.table, tidy(quant.krakamp.family.rbcl.lmer)[2,c(3,4,8)])
quant.krakamp.table = tidy(quant.krakamp.genus.its.lmer)[2,c(3,4,8)]
quant.krakamp.table = rbind(quant.krakamp.table, tidy(quant.krakamp.genus.rbcl.lmer)[2,c(3,4,8)])
quant.krakamp.table = rbind(quant.krakamp.table, tidy(quant.krakamp.species.its.lmer)[2,c(3,4,8)])
quant.krakamp.table = rbind(quant.krakamp.table, tidy(quant.krakamp.species.rbcl.lmer)[2,c(3,4,8)])

quant.krakamp.table$source = c("genus.its", "genus.rbcl", "species.its", "species.rbcl")

kable(quant.krakamp.table) %>% kable_styling(bootstrap_options = "striped", full_width = F)
```

term	estimate	p.value	source
sourcekrak	-0.0272715	0.1027510	genus.its
sourcekrak	-0.0608579	0.0252742	genus.rbcl
sourcekrak	-0.0683416	0.0026338	species.its
sourcekrak	-0.0226632	0.4738227	species.rbcl

*updated results Dec 2020**

- **take-home message:** in contrast to the previous results, we now see that for two of the four models, we have a statistically significant difference
- in those two models, amplicon data perform better / shotgun data perform worse
  - previously, amplicon outperformed shotgun for all four
  - directionality of trends is that amplicon does better, only significant for 2/4
- interestingly (weirdly?), the two statistically significant results are for different markers and different levels of taxonomic resolution (!?)—*rbcL* at the genus level, and ITS2 at the species level

linear-model based analysis for quant matching amplicon vs. WGS Run two separate linear mixed-effects models, one for WGS and one for amplicon. We already did the WGS ones above, so here we just have to do the ones for amplicon. Then we can ask two different things: 1) which has a better R^2, i.e. even if the slope is not 1 and the intercept is not zero, which one would give us a better relative estimate of pollen grain proportions; and 2) which one has a slope closer to one (which is really ultimately an analogue of the residuals analysis). While it would be sensible to constrain the intercept to be zero in both cases, again for the WGS I tried this and was not able to get R^2 estimates.

```
# run models:
#quant.amp.family.its.lmer = lmer(quant.family ~ pollen.grain.proportion + (1|mix.ID) + (1|family), data =
#quant.amp.family.rbcl.lmer = lmer(quant.family ~ pollen.grain.proportion + (1|mix.ID) + (1|family), data
quant.amp.genus.rbcl.lmer = lmer(quant.genus ~ pollen.grain.proportion + (1|mix.ID) + (1|genus), data = qu
quant.amp.genus.its.lmer = lmer(quant.genus ~ pollen.grain.proportion + (1|mix.ID) + (1|genus), data = quar
quant.amp.species.rbcl.lmer = lmer(quant.species ~ pollen.grain.proportion + (1|mix.ID) + (1|species), dat
quant.amp.species.its.lmer = lmer(quant.species ~ pollen.grain.proportion + (1|mix.ID) + (1|species), data

# r-squared calculations
#r2.family.its.amp = r2beta(quant.amp.family.its.lmer)
```

```
r2.genus.its.amp = r2beta(quant.amp.genus.its.lmer)
r2.species.its.amp = r2beta(quant.amp.species.its.lmer)
#r2.family.rbcl.amp = r2beta(quant.amp.family.rbcl.lmer)
r2.genus.rbcl.amp = r2beta(quant.amp.genus.rbcl.lmer)
r2.species.rbcl.amp = r2beta(quant.amp.species.rbcl.lmer)

# build a table of results, using `broom.mixed`
#quant.amp.table = tidy(quant.amp.family.its.lmer)[2,c(3,4,8)]
#quant.amp.table = rbind(quant.amp.table, tidy(quant.amp.family.rbcl.lmer)[2,c(3,4,8)])
quant.amp.table = tidy(quant.amp.genus.its.lmer)[2,c(3,4,8)]
quant.amp.table = rbind(quant.amp.table, tidy(quant.amp.genus.rbcl.lmer)[2,c(3,4,8)])
quant.amp.table = rbind(quant.amp.table, tidy(quant.amp.species.its.lmer)[2,c(3,4,8)])
quant.amp.table = rbind(quant.amp.table, tidy(quant.amp.species.rbcl.lmer)[2,c(3,4,8)])

# add in source column
quant.amp.table$source = c("genus.its", "genus.rbcl", "species.its", "species.rbcl")
quant.amp.table$rsquared = c(r2.genus.its.amp[2,6], r2.genus.rbcl.amp[2,6], r2.species.its.amp[2,6], r2.sp

# show off the table!
kable(quant.amp.table) %>% kable_styling(bootstrap_options = "striped", full_width = F)
```

term	estimate	p.value	source	rsquared
pollen.grain.proportion	0.4899757	0	genus.its	0.2562560
pollen.grain.proportion	0.5463068	0	genus.rbcl	0.3973826
pollen.grain.proportion	0.2421154	0	species.its	0.0897513
pollen.grain.proportion	0.3901673	0	species.rbcl	0.2959296

update Dec 2020: these results did not change at all

Comparing this table to the table from the WGS only quantitative matching, we can see that the R^2 values are are substantially higher for WGS / kraken as compared to the amplicon data. Consistently with the residual-based analysis, the slopes of the relationships for the amplicon data are much larger / much closer to one than for the WGS data.

This is consistent with:

1. WGS being more quantitative (higher R^2—better explanatory power in a linear relationship)
2. WGS having lower slopes, and thus higher residuals from a relationship where intercept = 0 and slope = 1, which could be driven in large part by the high rate of false positives in the WGS data (which greatly limit quantitative matching)

Figures

The numbering of figures in this section is not ultimately parallel to their numbering in the manuscript; we left it in the order they were developed because the naming in the code would have to change, which would be a substantial amount of work for relatively cosmetic purposes. Figures added later in the process are denoted with alphabetic rather than numeric names; e.g., X, Y, and Z for false negatives and I, J, and K for false positives.

Figure 1: true positives by taxon

This figure shows the mean proportion and binomial CI of correct (true positive) taxonomic matches by species and genus. Fig. 1a is by family, Fig. 1b is by genus, and Fig. 1c is by species.

The binomial CI is calculated using the binom.confint function, which requires a vector of the number of success (x) and a vector of the number of independent trials (n).

```r
#Create variable "presence" to get total number of independent tests
#truepos.krak.family$presence = 1
truepos.krak.genus$presence = 1
truepos.krak.species$presence = 1


#Total the number of taxa correctly identified (x) when grouped by taxa
#family_x = truepos.krak.family %>%
 # group_by(family) %>%
  #summarise(x = sum(qual.family))

genus_x = truepos.krak.genus %>%
  group_by(genus) %>%
  summarise(x = sum(qual.genus))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
species_x = truepos.krak.species %>%
  group_by(species) %>%
  summarise(x = sum(qual.species))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
#Get the total number of independent tests (n) when grouped by taxa
#family_n = truepos.krak.family %>%
 # group_by(family) %>%
#  summarise(n = sum(presence))

genus_n = truepos.krak.genus %>%
  group_by(genus) %>%
  summarise(n = sum(presence))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
species_n = truepos.krak.species %>%
  group_by(species) %>%
  summarise(n = sum(presence))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
#Calculate binomial confidence interval using x and n
#binomci_family = binom.confint(family_x$x, family_n$n, method="exact")
binomci_species = binom.confint(species_x$x, species_n$n, method="exact")
binomci_genus = binom.confint(genus_x$x, genus_n$n, method="exact")

#Add the taxa names, rename column
#binomci_family = cbind(family_x$family, binomci_family)
binomci_genus = cbind(genus_x$genus, binomci_genus)
binomci_species = cbind(species_x$species, binomci_species)

#colnames(binomci_family)[1] = "family"
colnames(binomci_genus)[1] = "genus"
colnames(binomci_species)[1] = "species"


#Figure 1A: Family level
#fig1a=binomci_family%>%
#  ggplot(aes(family,mean))+
#  geom_point(position=position_dodge(width=0.3), size=4, alpha=0.6)+
```

```
#  geom_errorbar(width=0.6, aes(family,ymin=lower, ymax=upper), alpha=0.6, position=position_dodge(width=0
#  xlab("family")+
#  ylab("proportion of correct matches")+
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw()
# fig1a = fig1a + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure 1B: Genus level
fig1b=binomci_genus%>%
  ggplot(aes(genus,mean))+
  geom_point(position=position_dodge(width=0.3), size=4, alpha=0.6)+
  geom_errorbar(width=0.6, aes(genus,ymin=lower, ymax=upper), alpha=0.6, position=position_dodge(width=0.3
  xlab("genus")+
  ylab("proportion of correct matches")+
  labs(title = NULL, subtitle = "B: Genus matches") +
  theme_bw()
fig1b = fig1b + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure 1C: Species level
fig1c=binomci_species%>%
  ggplot(aes(species,mean))+
  geom_point(position=position_dodge(width=0.3), size=4, alpha=0.6)+
  geom_errorbar(width=0.6, aes(species,ymin=lower, ymax=upper), alpha=0.6, position=position_dodge(width=0
  xlab("species")+
  ylab("proportion of correct matches")+
  labs(title = NULL, subtitle = "C: Species matches") +
  theme_bw()
fig1c = fig1c + theme(axis.text.x = element_text(angle = 45, hjust = 1))

  #Figure 1 panel

#suppressMessages(ggsave("WGS_fig1a.pdf", plot=fig1a, device="pdf", height=5, width=7, units="in"))
suppressMessages(ggsave("WGS_fig1b.pdf", plot=fig1b, device="pdf", height=5, width = 7, units="in"))
suppressMessages(ggsave("WGS_fig1c.pdf", plot=fig1c, device="pdf", height=5, width = 7, units="in"))
panel.fig1 = grid.arrange(#fig1a,
  fig1b,fig1c, ncol=1)
```

```
suppressMessages(ggsave("fig1_combined.jpg", plot=panel.fig1, device="jpeg", height=11, width = 8.5, units
```

Figure 2: true positives vs. species richness

This figure shows the proportion of correct taxonomic matches by species richness for *rbcL* and ITS2 at the species, genus, and family levels of matching. To correctly format the data, the mean and binomial CI of correct taxonomic matches needs to be summarized 3 times: once for sample, once for mix, and once for level of species richness. Otherwise samples with multiple species will get over-represented in terms of the overall means. The integers are tracked at each step and summarized with `dplyr`. The data are summarized with `dplyr` twice, once per mix and then once per sample. Then within each level of the factor of interest, the mean and binomial confidence intervals are calculated using `binom.confint` from the `binom` package.

```
#Summarize the agg dataset first by species richness, mix.ID, and sample. Also calculate pool size for eac

#CI.family.1 = truepos.krak.family %>%
#select(mix.ID, sample, spp.rich, qual.family, question.1, question.2, question.3) %>% group_by(spp.rich,
#summarize(pool.size = n(),
#  family = sum(qual.family),
# add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's figure 3 cod
#  question.1 = mean(question.1),
```

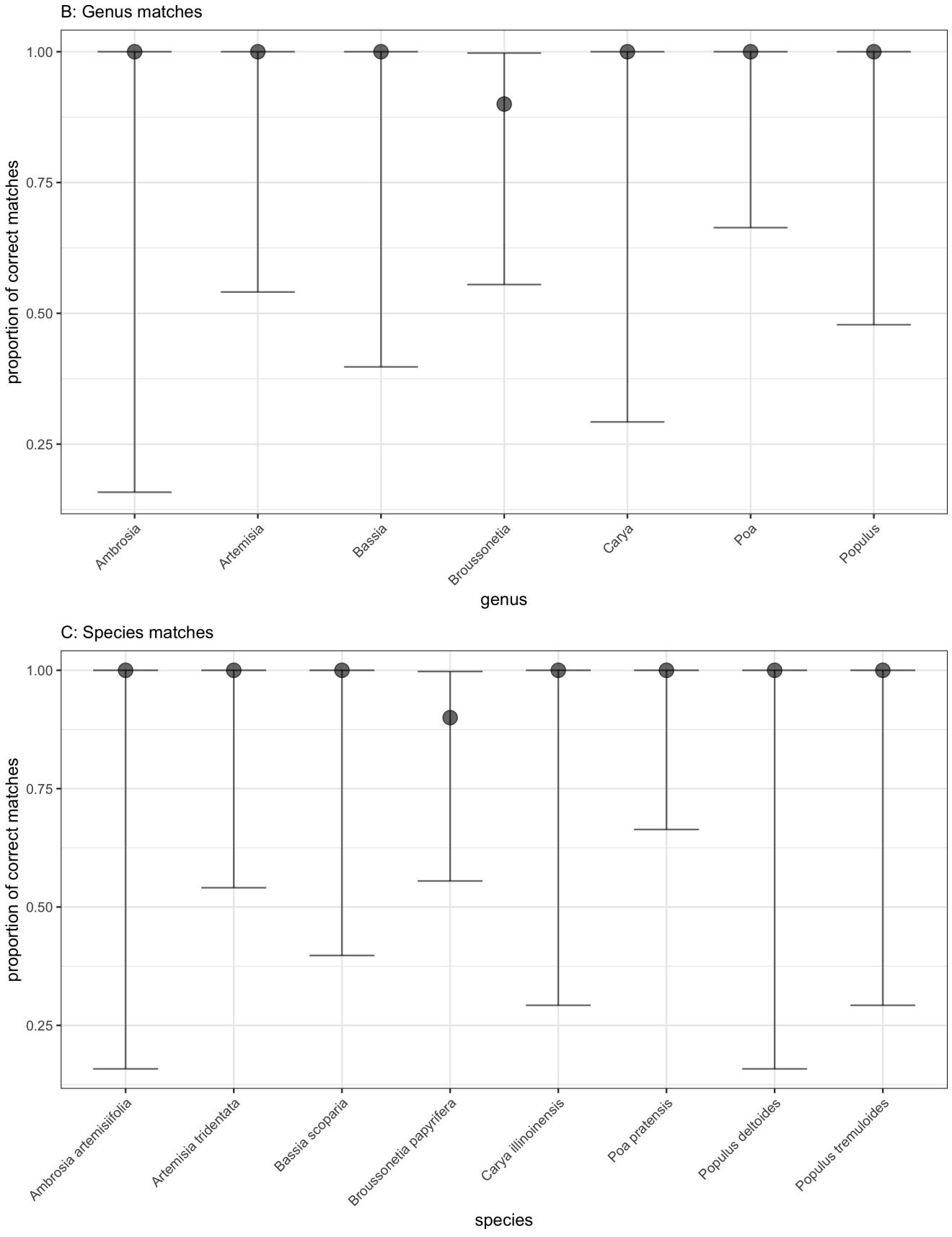


Figure 1: Figure 1

```
#  question.2 = mean(question.2),
#  question.3 = mean(question.3))

CI.genus.1 = truepos.krak.genus %>%
select(mix.ID, sample, spp.rich, qual.genus, question.1, question.2, question.3) %>% group_by(spp.rich, mi
summarize(pool.size = n(),
  genus = sum(qual.genus),
# add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's figure 3 cod
  question.1 = mean(question.1),
  question.2 = mean(question.2),
  question.3 = mean(question.3))
```

```
## `summarise()` regrouping output by 'spp.rich', 'mix.ID' (override with `.groups` argument)
```

```
CI.species.1 = truepos.krak.species %>%
select(mix.ID, sample, spp.rich, qual.species, question.1, question.2, question.3) %>% group_by(spp.rich, 
summarize(pool.size = n(),
  species = sum(qual.species),
# add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's figure 3 cod
  question.1 = mean(question.1),
  question.2 = mean(question.2),
  question.3 = mean(question.3))
```

```
## `summarise()` regrouping output by 'spp.rich', 'mix.ID' (override with `.groups` argument)
```

```
# second step: group_by(spp.rich, mix.ID, pool.size)

#CI.family.2 = CI.family.1 %>% group_by(spp.rich, mix.ID, pool.size) %>%
#  summarize(pool.num = n(),
#  family = sum(family),
#  question.1 = mean(question.1),
#  question.2 = mean(question.2),
#  question.3 = mean(question.3)) # %>%
  # multiply pool size & number to get total number of possibilities of matches
#  CI.family.2$n2 = CI.family.2$pool.size * CI.family.2$pool.num

CI.genus.2 = CI.genus.1 %>% group_by(spp.rich, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
  genus = sum(genus),
  question.1 = mean(question.1),
  question.2 = mean(question.2),
  question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'spp.rich', 'mix.ID' (override with `.groups` argument)
```

```
  # multiply pool size & number to get total number of possibilities of matches
  CI.genus.2$n2 = CI.genus.2$pool.size * CI.genus.2$pool.num

CI.species.2 = CI.species.1 %>% group_by(spp.rich, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
  species = sum(species),
  question.1 = mean(question.1),
  question.2 = mean(question.2),
  question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'spp.rich', 'mix.ID' (override with `.groups` argument)
```

```
  # multiply pool size & number to get total number of possibilities of matches
  CI.species.2$n2 = CI.species.2$pool.size * CI.species.2$pool.num
```

```
#Format variables to be factors.  Restrict to mixtures where questions are equal to 1. **Not sure why this

#dat.family <- CI.family.2
#dat.family$question.1=as.factor(dat.family$question.1)
#dat.family$question.2=as.factor(dat.family$question.2)
#dat.family$question.3=as.factor(dat.family$question.3)
#dat.family$spp.rich=as.factor(dat.family$spp.rich)
#dat.family=filter(dat.family,question.1=="1"|question.2=="1"|question.3=="1")
#fig2_family=dat.family%>%
#  select(spp.rich, mix.ID, n2, family)

dat.genus <- CI.genus.2
dat.genus$question.1=as.factor(dat.genus$question.1)
dat.genus$question.2=as.factor(dat.genus$question.2)
dat.genus$question.3=as.factor(dat.genus$question.3)
dat.genus$spp.rich=as.factor(dat.genus$spp.rich)
#dat.genus=filter(dat.genus,question.1=="1"|question.2=="1"|question.3=="1")
fig2_genus=dat.genus%>%
  select(spp.rich, mix.ID, n2, genus)

dat.species <- CI.species.2
dat.species$question.1=as.factor(dat.species$question.1)
dat.species$question.2=as.factor(dat.species$question.2)
dat.species$question.3=as.factor(dat.species$question.3)
dat.species$spp.rich=as.factor(dat.species$spp.rich)
#dat.species=filter(dat.species,question.1=="1"|question.2=="1"|question.3=="1")
fig2_species=dat.species%>%
  select(spp.rich, mix.ID, n2, species)

#Convert from wide to long

#fig2_long_family = melt(fig2_family, id.vars = c("spp.rich","mix.ID", "n2"))
fig2_long_genus = melt(fig2_genus, id.vars = c("spp.rich","mix.ID", "n2"))
```

```
## Warning in melt(fig2_genus, id.vars = c("spp.rich", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(fig2_genus). In the next version, this warning
## will become an error.
```

```
fig2_long_species = melt(fig2_species, id.vars = c("spp.rich","mix.ID", "n2"))
```

```
## Warning in melt(fig2_species, id.vars = c("spp.rich", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(fig2_species). In the next version, this warning
## will become an error.
```

```
#Subset to species richness

#spp.rich_family = fig2_long_family%>%
#  filter(substr(variable,1,7)=="family")%>%
#  rename(x=value, n=n2)
```

```
spp.rich_genus = fig2_long_genus%>%
  filter(substr(variable,1,7)=="genus")%>%
  rename(x=value, n=n2)
spp.rich_species = fig2_long_species%>%
  filter(substr(variable,1,7)=="species")%>%
  rename(x=value, n=n2)

#group by species richness and marker so that all mixtures with richness = 2 are grouped together

#spp.rich_family = spp.rich_family %>%
#  group_by(spp.rich, variable) %>%
#  summarize(n=mean(n),x=mean(x)) %>%
#  ungroup()

spp.rich_genus = spp.rich_genus %>%
  group_by(spp.rich, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'spp.rich' (override with `.groups` argument)
```

```
spp.rich_species = spp.rich_species %>%
  group_by(spp.rich, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'spp.rich' (override with `.groups` argument)
```

```
#Calculate mean and binomial CI

#bin.family = binom.confint(spp.rich_family$x, spp.rich_family$n,methods="exact") %>%
#  select(-method)

bin.genus = binom.confint(spp.rich_genus$x, spp.rich_genus$n,methods="exact") %>%
  select(-method)

bin.species = binom.confint(spp.rich_species$x, spp.rich_species$n,methods="exact") %>%
  select(-method)

#Merge mean and binomial CI with mixture info

#family.all = merge(bin.family, spp.rich_family, by = c("x", "n"))
#family.all = unique(family.all)

genus.all = merge(bin.genus, spp.rich_genus, by = c("x", "n"))
genus.all = unique(genus.all)

species.all = merge(bin.species, spp.rich_species, by = c("x", "n"))
species.all = unique(species.all)

#Figure 2A: Family level
#fig2a=family.all%>%
#  filter(variable=="family")%>%
#  ggplot()+
#  geom_point(aes(spp.rich,mean))+
#  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
#  xlab("species richness")+
```

```
#  ylab("proportion of correct matches")+
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches")+
#   theme_bw()

#Figure 2B: Genus level
fig2b=genus.all%>%
  filter(variable=="genus")%>%
  ggplot()+
  geom_point(aes(spp.rich,mean))+
  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
  xlab("species richness")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches")+
   theme_bw()

#Figure 2A: Species level
fig2c=species.all%>%
  filter(variable=="species")%>%
  ggplot()+
  geom_point(aes(spp.rich,mean))+
  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
  xlab("species richness")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches")+
   theme_bw()

#Create panel for Figure 2
fig2=grid.arrange(#fig2a,
  fig2b,fig2c, nrow=1)
```

```
suppressMessages(ggsave("fig2_combined.jpg", plot=fig2, device="jpeg", width=169, units="mm"))
```

Figure X: true positives vs. relatedness

```
#Summarize the agg dataset first by relatedness, mix.ID, and sample. Also calculate pool size for each sam

#CI.family.1 = truepos.krak.family %>%
#  select(mix.ID, sample, relatedness, qual.family, question.1, question.2, question.3) %>% group_by(relat
#  summarize(pool.size = n(),
#            family = sum(qual.family),
           # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
#            question.1 = mean(question.1),
#            question.2 = mean(question.2),
#            question.3 = mean(question.3))

CI.genus.1 = truepos.krak.genus %>%
  select(mix.ID, sample, relatedness, qual.genus, question.1, question.2, question.3) %>% group_by(relatedn
  summarize(pool.size = n(),
            genus = sum(qual.genus),
            # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3))
```

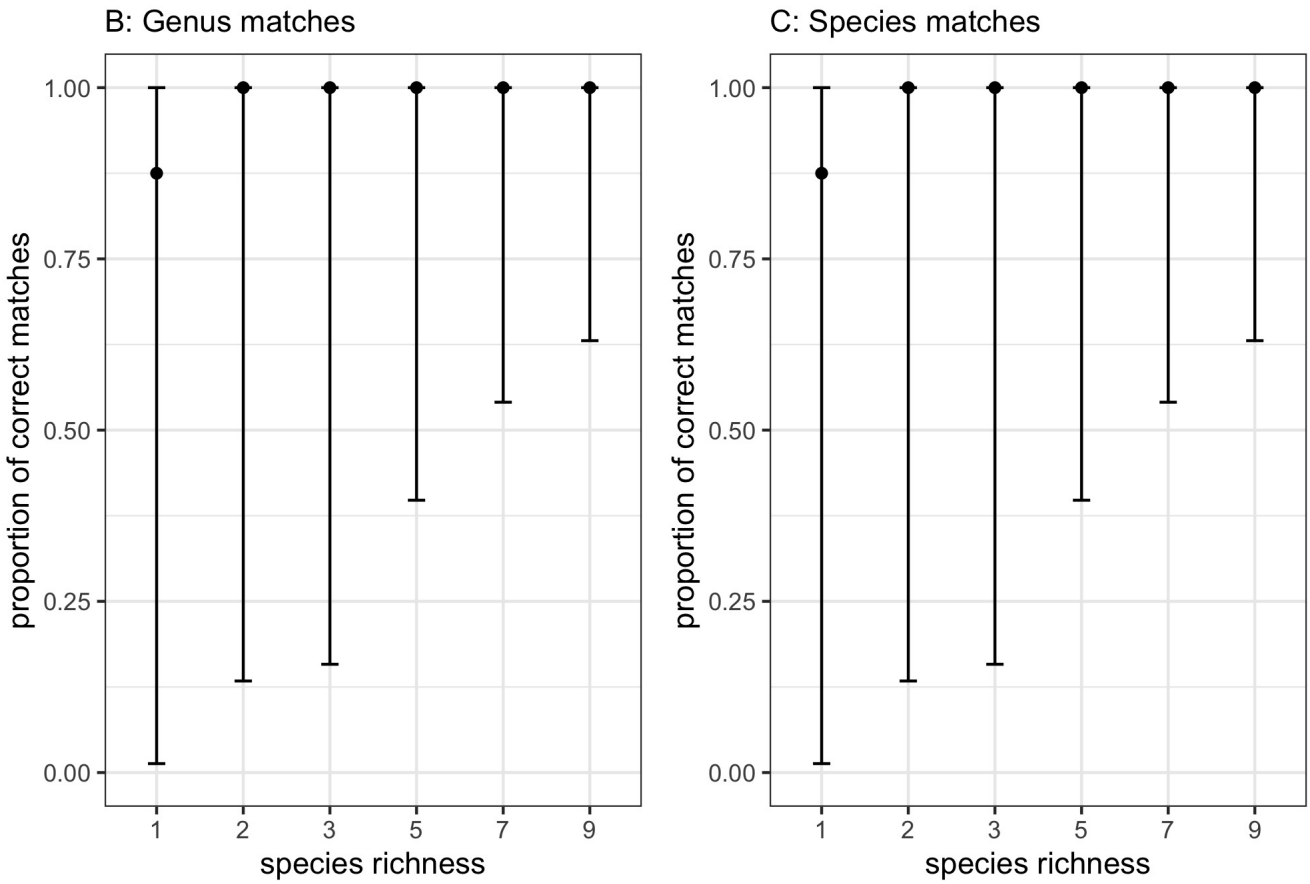


Figure 2: Figure 2

```
## `summarise()` regrouping output by 'relatedness', 'mix.ID' (override with `.groups` argument)
```

```
CI.species.1 = truepos.krak.species %>%
  select(mix.ID, sample, relatedness, qual.species, question.1, question.2, question.3) %>% group_by(relat
  summarize(pool.size = n(),
            species = sum(qual.species),
            # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3))
```

```
## `summarise()` regrouping output by 'relatedness', 'mix.ID' (override with `.groups` argument)
```

```
# second step: group_by(relatedness, mix.ID, pool.size)

#CI.family.2 = CI.family.1 %>% group_by(relatedness, mix.ID, pool.size) %>%
#  summarize(pool.num = n(),
#             family = sum(family),
#             question.1 = mean(question.1),
#             question.2 = mean(question.2),
#             question.3 = mean(question.3)) # %>%

# multiply pool size & number to get total number of possibilities of matches
CI.family.2$n2 = CI.family.2$pool.size * CI.family.2$pool.num
```

```
## Error in eval(expr, envir, enclos): object 'CI.family.2' not found
```

```
CI.genus.2 = CI.genus.1 %>% group_by(relatedness, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
            genus = sum(genus),
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'relatedness', 'mix.ID' (override with `.groups` argument)
```

```
# multiply pool size & number to get total number of possibilities of matches
CI.genus.2$n2 = CI.genus.2$pool.size * CI.genus.2$pool.num

CI.species.2 = CI.species.1 %>% group_by(relatedness, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
            species = sum(species),
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'relatedness', 'mix.ID' (override with `.groups` argument)
```

```
# multiply pool size & number to get total number of possibilities of matches
CI.species.2$n2 = CI.species.2$pool.size * CI.species.2$pool.num

#Format variables to be factors.  Restrict to mixtures where questions are equal to 1. **Not sure why this

#dat.family <- CI.family.2
#dat.family$question.1=as.factor(dat.family$question.1)
#dat.family$question.2=as.factor(dat.family$question.2)
#dat.family$question.3=as.factor(dat.family$question.3)
#dat.family$relatedness=as.factor(dat.family$relatedness)
```

```
#dat.family=filter(dat.family,question.1=="1"|question.2=="1"|question.3=="1")
#figX_family=dat.family%>%
#  select(relatedness, mix.ID, n2, family)

dat.genus <- CI.genus.2
dat.genus$question.1=as.factor(dat.genus$question.1)
dat.genus$question.2=as.factor(dat.genus$question.2)
dat.genus$question.3=as.factor(dat.genus$question.3)
dat.genus$relatedness=as.factor(dat.genus$relatedness)
#dat.genus=filter(dat.genus,question.1=="1"|question.2=="1"|question.3=="1")
figX_genus=dat.genus%>%
  select(relatedness, mix.ID, n2, genus)

dat.species <- CI.species.2
dat.species$question.1=as.factor(dat.species$question.1)
dat.species$question.2=as.factor(dat.species$question.2)
dat.species$question.3=as.factor(dat.species$question.3)
dat.species$relatedness=as.factor(dat.species$relatedness)
#dat.species=filter(dat.species,question.1=="1"|question.2=="1"|question.3=="1")
figX_species=dat.species%>%
  select(relatedness, mix.ID, n2, species)

#Convert from wide to long
#figX_long_family = melt(figX_family, id.vars = c("relatedness","mix.ID", "n2"))
figX_long_genus = melt(figX_genus, id.vars = c("relatedness","mix.ID", "n2"))
```

```
## Warning in melt(figX_genus, id.vars = c("relatedness", "mix.ID", "n2")):
## The melt generic in data.table has been passed a grouped_df and will attempt
## to redirect to the relevant reshape2 method; please note that reshape2 is
## deprecated, and this redirection is now deprecated as well. To continue using
## melt methods from reshape2 while both libraries are attached, e.g. melt.list,
## you can prepend the namespace like reshape2::melt(figX_genus). In the next
## version, this warning will become an error.
```

```
figX_long_species = melt(figX_species, id.vars = c("relatedness","mix.ID", "n2"))
```

```
## Warning in melt(figX_species, id.vars = c("relatedness", "mix.ID", "n2")):
## The melt generic in data.table has been passed a grouped_df and will attempt
## to redirect to the relevant reshape2 method; please note that reshape2 is
## deprecated, and this redirection is now deprecated as well. To continue using
## melt methods from reshape2 while both libraries are attached, e.g. melt.list,
## you can prepend the namespace like reshape2::melt(figX_species). In the next
## version, this warning will become an error.
```

```
#Subset to relatedness
#relatedness_family = figX_long_family%>%
#  filter(substr(variable,1,7)=="family")%>%
#  rename(x=value, n=n2)
relatedness_genus = figX_long_genus%>%
  filter(substr(variable,1,7)=="genus")%>%
  rename(x=value, n=n2)
relatedness_species = figX_long_species%>%
  filter(substr(variable,1,7)=="species")%>%
  rename(x=value, n=n2)

#group by relatedness and marker
#relatedness_family = relatedness_family %>%
```

```
#  group_by(relatedness, variable) %>%
#  summarize(n=mean(n),x=mean(x)) %>%
#  ungroup()

relatedness_genus = relatedness_genus %>%
  group_by(relatedness, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'relatedness' (override with `.groups` argument)
```

```
relatedness_species = relatedness_species %>%
  group_by(relatedness, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'relatedness' (override with `.groups` argument)
```

```
#Calculate mean and binomial CI
#bin.family = binom.confint(relatedness_family$x, relatedness_family$n,methods="exact") %>%
  #select(-method)

#bin.genus = binom.confint(relatedness_genus$x, relatedness_genus$n,methods="exact") %>%
  #select(-method)

#bin.species = binom.confint(relatedness_species$x, relatedness_species$n,methods="exact") %>%
  #select(-method)

#Merge mean and binomial CI with mixture info
#family.all = merge(bin.family, relatedness_family, by = c("x", "n"))
#family.all = unique(family.all)

genus.all = merge(bin.genus, relatedness_genus, by = c("x", "n"))
genus.all = unique(genus.all)

species.all = merge(bin.species, relatedness_species, by = c("x", "n"))
species.all = unique(species.all)

#Figure XA: Family level
#figXa=family.all%>%
#  filter(variable=="family")%>%
#  ggplot()+
#  geom_point(aes(relatedness,mean))+
#  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
#  xlab("relatedness")+
#  ylab("proportion of correct matches")+
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches") +
#  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
#  theme_bw()
# figXa = figXa + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure XB: Genus level
figXb=genus.all%>%
  filter(variable=="genus")%>%
  ggplot()+
  geom_point(aes(relatedness,mean))+
```

```
  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
  xlab("relatedness")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches")+
  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
  theme_bw()
figXb = figXb + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure XA: Species level
figXc=species.all%>%
  filter(variable=="species")%>%
  ggplot()+
  geom_point(aes(relatedness,mean))+
  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
  xlab("relatedness")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches") +
  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
  theme_bw()
figXc = figXc + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Create panel for Figure X
figX=grid.arrange(#figXa,
  figXb,figXc, nrow=1)
```

```
suppressMessages(ggsave("figX_combined.jpg", plot=figX, device="jpeg", width=169, units="mm"))
```

Figure Y: true positives vs. rarity (binned)

The first step here is to create a categorical `rarity` variable based on binning `pollen.grain.proportion` into four equal bins (1-25%; 26-50%; 51-75%; 76-100%). We do that using the `cut` function. By creating a categorical variable, we can produce a figure that is highly parallel to those for the other two facets of sample complexity (species richness and relatedness)

```
# this figure mirrors the other two questions (species richness and rarity) but
# rarity (input pollen grain proportion). Since rarity is continuous and the others are
# categorical we will 'bin' the rarity metrics into 4 bins:
# 1-25%; 26-50%; 51-75%; 76-100%

# use the `cut` function to create the `rarity` variable by binning `pollen.grain.proportion`:
#truepos.krak.family$rarity = cut(truepos.krak.family$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0.
#      labels = c("<25%", "25-50%", "50-75%", ">75%"))
truepos.krak.genus$rarity = cut(truepos.krak.genus$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0.75,
      labels = c("<25%", "25-50%", "50-75%", ">75%"))
truepos.krak.species$rarity = cut(truepos.krak.species$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0
      labels = c("<25%", "25-50%", "50-75%", ">75%"))

#Summarize the agg dataset first by rarity, mix.ID, and sample. Also calculate pool size for each sample (

#CI.family.1 = truepos.krak.family %>%
#  select(mix.ID, sample, rarity, qual.family, question.1, question.2, question.3) %>% group_by(rarity, mi
#  summarize(pool.size = n(),
#            family = sum(qual.family),
           # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
#            question.1 = mean(question.1),
```

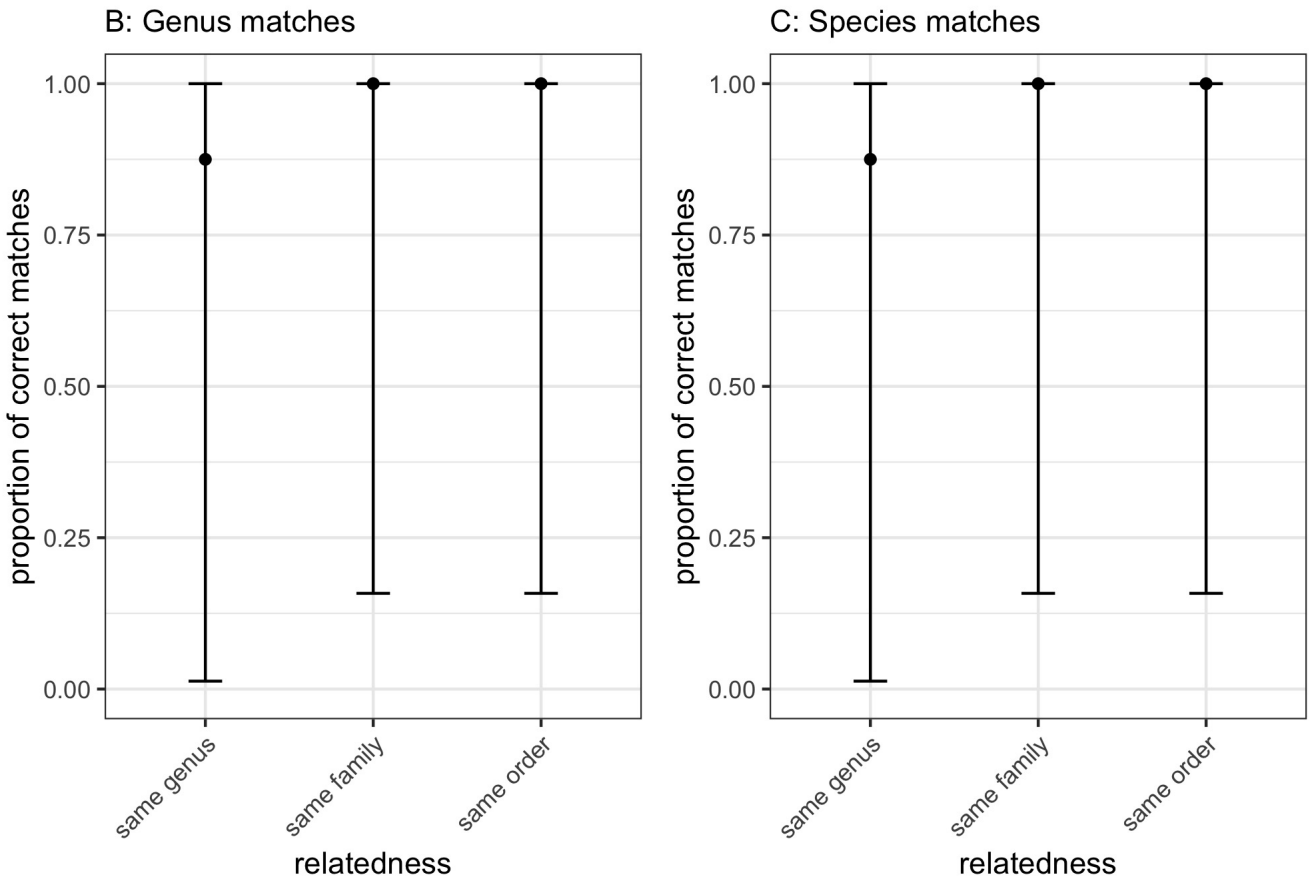


Figure 3: Figure X

```
#            question.2 = mean(question.2),
#            question.3 = mean(question.3))

CI.genus.1 = truepos.krak.genus %>%
  select(mix.ID, sample, rarity, qual.genus, question.1, question.2, question.3) %>% group_by(rarity, mix.
  summarize(pool.size = n(),
            genus = sum(qual.genus),
            # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3))
```

```
## `summarise()` regrouping output by 'rarity', 'mix.ID' (override with `.groups` argument)
```

```
CI.species.1 = truepos.krak.species %>%
  select(mix.ID, sample, rarity, qual.species, question.1, question.2, question.3) %>% group_by(rarity, mi
  summarize(pool.size = n(),
            species = sum(qual.species),
            # add question.1, question.2, question.3 because those equal to 1 are filtered out in Jamie's
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3))
```

```
## `summarise()` regrouping output by 'rarity', 'mix.ID' (override with `.groups` argument)
```

```
# second step: group_by(rarity, mix.ID, pool.size)

#CI.family.2 = CI.family.1 %>% group_by(rarity, mix.ID, pool.size) %>%
#  summarize(pool.num = n(),
#            family = sum(family),
#            question.1 = mean(question.1),
#            question.2 = mean(question.2),
#            question.3 = mean(question.3)) # %>%

# multiply pool size & number to get total number of possibilities of matches
#CI.family.2$n2 = CI.family.2$pool.size * CI.family.2$pool.num

CI.genus.2 = CI.genus.1 %>% group_by(rarity, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
            genus = sum(genus),
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'rarity', 'mix.ID' (override with `.groups` argument)
```

```
# multiply pool size & number to get total number of possibilities of matches
CI.genus.2$n2 = CI.genus.2$pool.size * CI.genus.2$pool.num

CI.species.2 = CI.species.1 %>% group_by(rarity, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
            species = sum(species),
            question.1 = mean(question.1),
            question.2 = mean(question.2),
            question.3 = mean(question.3)) # %>%
```

```
## `summarise()` regrouping output by 'rarity', 'mix.ID' (override with `.groups` argument)
```

```
# multiply pool size & number to get total number of possibilities of matches
CI.species.2$n2 = CI.species.2$pool.size * CI.species.2$pool.num

#Format variables to be factors.  Restrict to mixtures where questions are equal to 1. **Not sure why this

#dat.family <- CI.family.2
#dat.family$question.1=as.factor(dat.family$question.1)
#dat.family$question.2=as.factor(dat.family$question.2)
#dat.family$question.3=as.factor(dat.family$question.3)
#dat.family$rarity=as.factor(dat.family$rarity)
#dat.family=filter(dat.family,question.1=="1"|question.2=="1"|question.3=="1")
#figY_family=dat.family%>%
#  select(rarity, mix.ID, n2, family)

dat.genus <- CI.genus.2
dat.genus$question.1=as.factor(dat.genus$question.1)
dat.genus$question.2=as.factor(dat.genus$question.2)
dat.genus$question.3=as.factor(dat.genus$question.3)
dat.genus$rarity=as.factor(dat.genus$rarity)
#dat.genus=filter(dat.genus,question.1=="1"|question.2=="1"|question.3=="1")
figY_genus=dat.genus%>%
  select(rarity, mix.ID, n2, genus)

dat.species <- CI.species.2
dat.species$question.1=as.factor(dat.species$question.1)
dat.species$question.2=as.factor(dat.species$question.2)
dat.species$question.3=as.factor(dat.species$question.3)
dat.species$rarity=as.factor(dat.species$rarity)
#dat.species=filter(dat.species,question.1=="1"|question.2=="1"|question.3=="1")
figY_species=dat.species%>%
  select(rarity, mix.ID, n2, species)

#Convert from wide to long

#figY_long_family = melt(figY_family, id.vars = c("rarity","mix.ID", "n2"))
figY_long_genus = melt(figY_genus, id.vars = c("rarity","mix.ID", "n2"))
```

```
## Warning in melt(figY_genus, id.vars = c("rarity", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(figY_genus). In the next version, this warning
## will become an error.
```

```
figY_long_species = melt(figY_species, id.vars = c("rarity","mix.ID", "n2"))
```

```
## Warning in melt(figY_species, id.vars = c("rarity", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(figY_species). In the next version, this warning
## will become an error.
```

```
#Subset to rarity
```

```
#rarity_family = figY_long_family%>%
#  filter(substr(variable,1,7)=="family")%>%
#  rename(x=value, n=n2)
rarity_genus = figY_long_genus%>%
  filter(substr(variable,1,7)=="genus")%>%
  rename(x=value, n=n2)
rarity_species = figY_long_species%>%
  filter(substr(variable,1,7)=="species")%>%
  rename(x=value, n=n2)

#group by rarity and marker so that all mixtures with richness = 2 are grouped together

#rarity_family = rarity_family %>%
#  group_by(rarity, variable) %>%
#  summarize(n=mean(n),x=mean(x)) %>%
#  ungroup()

rarity_genus = rarity_genus %>%
  group_by(rarity, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'rarity' (override with `.groups` argument)
```

```
rarity_species = rarity_species %>%
  group_by(rarity, variable) %>%
  summarize(n=mean(n),x=mean(x)) %>%
  ungroup()
```

```
## `summarise()` regrouping output by 'rarity' (override with `.groups` argument)
```

```
#Calculate mean and binomial CI

#bin.family = binom.confint(rarity_family$x, rarity_family$n,methods="exact") %>%
#  select(-method)

bin.genus = binom.confint(rarity_genus$x, rarity_genus$n,methods="exact") %>%
  select(-method)

bin.species = binom.confint(rarity_species$x, rarity_species$n,methods="exact") %>%
  select(-method)

#Merge mean and binomial CI with mixture info

#family.all = merge(bin.family, rarity_family, by = c("x", "n"))
#family.all = unique(family.all)

genus.all = merge(bin.genus, rarity_genus, by = c("x", "n"))
genus.all = unique(genus.all)

species.all = merge(bin.species, rarity_species, by = c("x", "n"))
species.all = unique(species.all)

#Figure YA: Family level
#figYa=family.all%>%
#  filter(variable=="family")%>%
#  ggplot()+
```

```
#  geom_point(aes(rarity,mean))+
#  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2) +
#  xlab("rarity") +
#  ylab("proportion of correct matches") +
#  ylim(0,1) +
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw()
#figYa = figYa + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure YB: Genus level
figYb=genus.all%>%
  filter(variable=="genus")%>%
  ggplot()+
  geom_point(aes(rarity,mean))+
  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2)+
  xlab("rarity")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches")+
  theme_bw()
figYb = figYb + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure YC: Species level
figYc=species.all%>%
  filter(variable=="species")%>%
  ggplot()+
  geom_point(aes(rarity,mean))+
  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2)+
  xlab("rarity")+
  ylab("proportion of correct matches") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches")+
  theme_bw()
figYc = figYc + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Create panel for Figure Y
figY=grid.arrange(#figYa,
  figYb,figYc, nrow=1)
```

```
suppressMessages(ggsave("figY_combined.jpg", plot=figY, device="jpeg", width=169, units="mm"))
```

Figure S1: true positives combined

Here we are putting together a figure comprised of panels we have already built in the sections for Figs 2, X, and Y. This figure has 9 panels, with a row for each of the 3 facets of sample complexity and a column for each of the 3 levels of taxonomic matching. Very straightforward.

```
# re-label panels:
#figXa = figXa + labs(title = NULL, subtitle = "D: Family matches")
figXb = figXb + labs(title = NULL, subtitle = "E: Genus matches")
figXc = figXc + labs(title = NULL, subtitle = "F: Species matches")
#figYa = figYa + labs(title = NULL, subtitle = "H: Family matches")
figYb = figYb + labs(title = NULL, subtitle = "I: Genus matches")
figYc = figYc + labs(title = NULL, subtitle = "J: Species matches")

#Create panel for Figure S1
figS1 = grid.arrange(#fig2a,
```

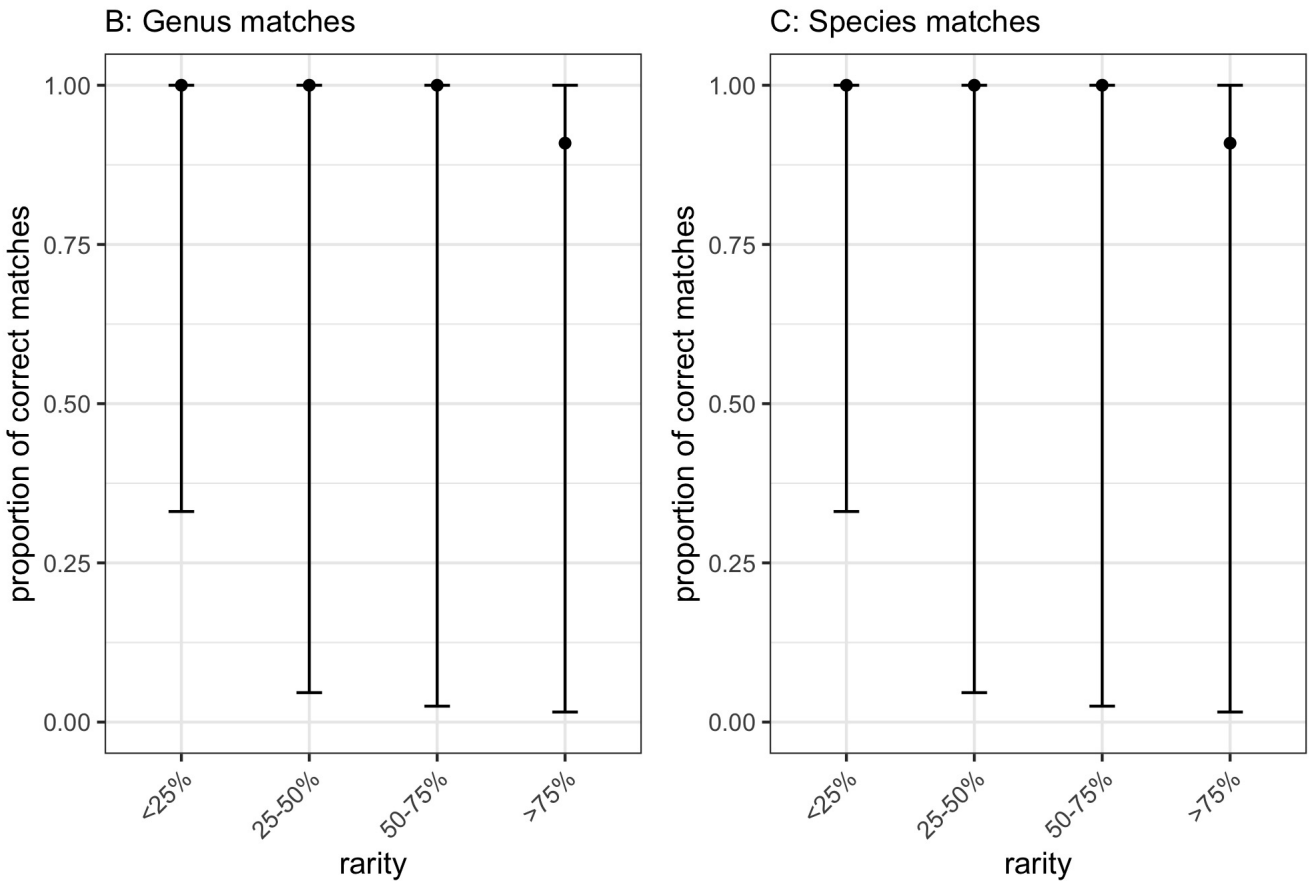


Figure 4: Figure Y

```
  fig2b, fig2c, #figXa,
  figXb, figXc, #figYa,
  figYb, figYc, nrow=3)
```

```
suppressMessages(ggsave("figS1_combined.jpg", plot=figS1, device="jpeg", width=169, height = 250, units="m
```

Figure Z: true positives by all 3 complexity variables; species level only

Here we are putting together a figure comprised of panels we have already built in the sections for Figs 2, X, and Y. One relatively minor tweak is altering the figure header for each panel. Currently they all say "species" because they were pooled into figures with varying levels of taxonomic matching. We need to instead change the headers to reflect the explanatory variable (element of sample complexity) that each panel is focused on.

This code chunk comes *after* the Supplemental Figure so that we change those headers only after the previous headers are used as they are supposed to be (in the figure where each panel needs to be identified by its level of taxonomic matching).

A second minor change is that because the x-axis labels are different for each panel, with some of them being longer, the panels themselves end up being different heights. To get around this, setting the orientation of this plot as vertical rather than horizontal. This may also work better in a journal format with the figure being easier to compare to the text to the left of it.

```
# change panel headers
fig2c = fig2c + labs(title = NULL, subtitle = "A: Species Richness")
figXc = figXc + labs(title = NULL, subtitle = "B: Relatedness")
figYc = figYc + labs(title = NULL, subtitle = "C: Rarity")

#Create panel for Figure Z
figZ = grid.arrange(fig2c,figXc,figYc, nrow=3)
```

```
suppressMessages(ggsave("figZ_combined.jpg", plot=figZ, device="jpeg", width=(169/3), height = 250, units=
```

Figure 3: true positives vs. rarity by taxon

don't use this figure—instead use fig Y

This figure shows the relationship between the proportion of pollen grains belonging to a particular taxon with the probability of detection (presence/absence) of that taxon in the sequencing reads. Each color belongs to a particular taxon, and each taxon has its own trendline as determined by logistic regression.

```
#fig3_sub_family=truepos.krak.family%>%
#  select(mix.ID, family, spp.rich, pollen.grain.proportion, qual.family)

#fig3_sub_family$spp.rich=as.factor(fig3_sub_family$spp.rich)

fig3_sub_genus=truepos.krak.genus%>%
  select(mix.ID, genus, spp.rich, pollen.grain.proportion, qual.genus)

fig3_sub_genus$spp.rich=as.factor(fig3_sub_genus$spp.rich)

fig3_sub_species=truepos.krak.species%>%
  select(mix.ID, species, spp.rich, pollen.grain.proportion, qual.species)

fig3_sub_genus$spp.rich=as.factor(fig3_sub_genus$spp.rich)

#Figure 3a: Family level
#fig3a=fig3_sub_family%>%
#  group_by(family)%>%
#  ggplot(aes(pollen.grain.proportion, qual.family, color = family, shape = family))+
```

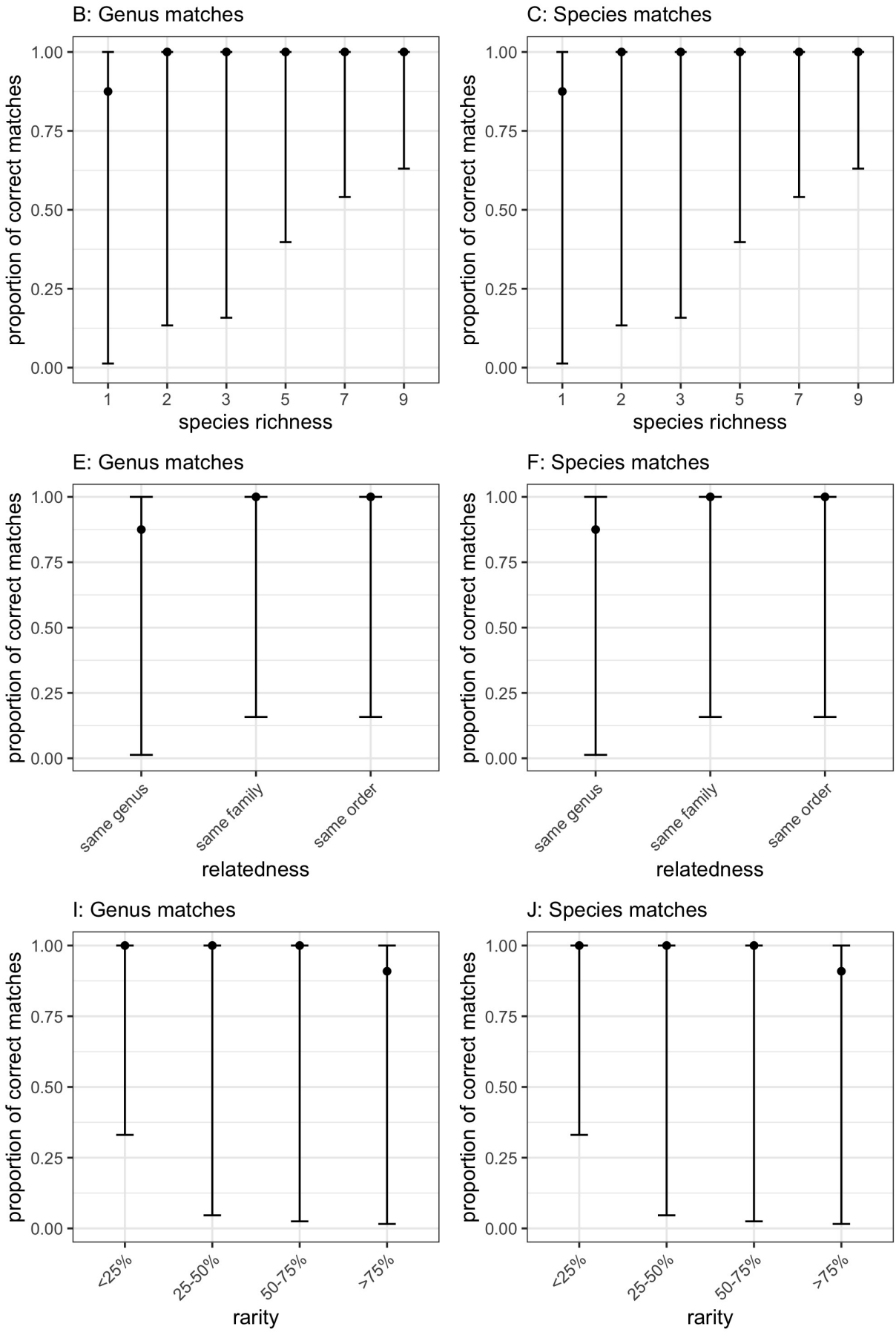


Figure 5: Figure S1

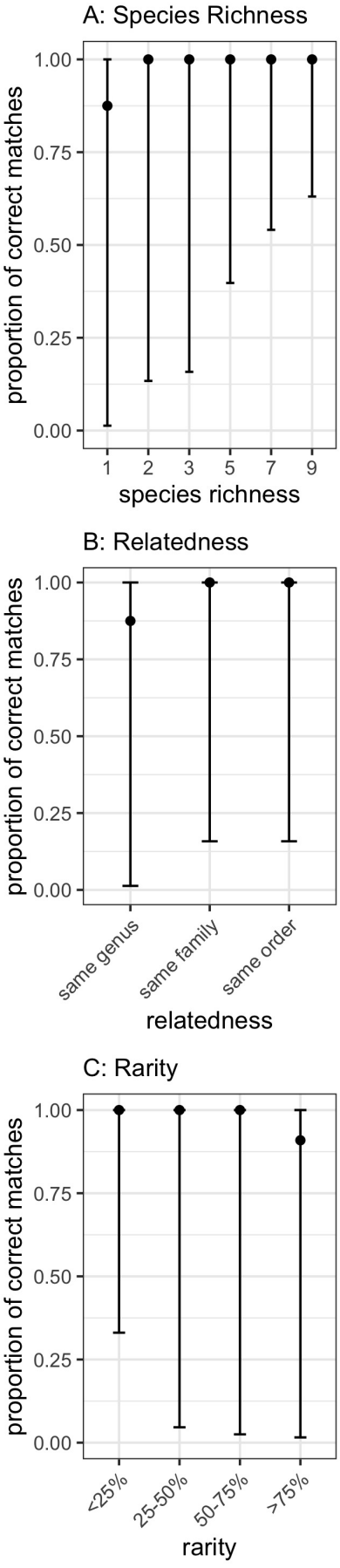


Figure 6: Figure Z

```
#  geom_jitter( size=4, alpha=0.4, height=0.05 )+
#    geom_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE, alpha=0.2) +
#  ggtitle("A")+
#  xlab("proportion of pollen grains in the sample") +
#  ylab("probability of detection") +
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw() +
#  theme(legend.key = element_rect(size = 5),
#        legend.key.size = unit(1.5, 'lines'))

#Figure 3b: Genus level
fig3b=fig3_sub_genus%>%
  group_by(genus)%>%
  ggplot(aes(pollen.grain.proportion, qual.genus, color = genus, shape = genus))+
  geom_jitter( size=4, alpha=0.4, height=0.05 )+
    geom_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE, alpha=0.2) +
  ggtitle("B")+
  xlab("proportion of pollen grains in the sample") +
  ylab("probability of detection") +
  labs(title = NULL, subtitle = "B: Genus matches") +
  theme_bw() +
  theme(legend.key = element_rect(size = 5),
        legend.key.size = unit(1.5, 'lines'))

#Figure 3c: Species level
fig3c=fig3_sub_species%>%
  group_by(species)%>%
  ggplot(aes(pollen.grain.proportion, qual.species, color = species, shape = species))+
  geom_jitter( size=4, alpha=0.4, height=0.05 )+
    geom_smooth(method = "glm", method.args = list(family = "binomial"), se = FALSE, alpha=0.2) +
  ggtitle("C")+
  xlab("proportion of pollen grains  in the sample") +
  ylab("probability of detection") +
  labs(title = NULL, subtitle = "C: Species matches")+
  theme_bw() +
  theme(legend.key = element_rect(size = 5),
        legend.key.size = unit(1.5, 'lines'))

#Create panel for figure 3
fig3=grid.arrange(#fig3a,
  fig3b,fig3c, nrow=1)
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.

## Warning: Removed 5 rows containing missing values (geom_point).

## `geom_smooth()` using formula 'y ~ x'

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
#suppressMessages(ggsave("WGS_fig3a.pdf", plot=fig3a, device="pdf", width=7, height=5, units="in"))
suppressMessages(ggsave("WGS_fig3b.pdf", plot=fig3b, device="pdf", width=7, height=5, units="in"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
suppressMessages(ggsave("WGS_fig3c.pdf", plot=fig3c, device="pdf", width=7, height=5, units="in"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
panel.fig3 = grid.arrange(#fig3a,
  fig3b, fig3c, ncol=1) #  heights = rep(50,3)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```

```
suppressMessages(ggsave("fig3_combined.jpg", plot=panel.fig3, device="jpeg", height=8.5, width = 11, units=
```

AGAIN, DO NOT USE THIS FIGURE

Figure 4: quantitative matching by taxon

This figure shows the quantitative relationship between the proportion of pollen grains and the proportion of reads matching to a particular taxa. There are plots for the species, genus, and family level.

Fortunately, these figures do not require formatting of the dataset beyond what has already been done. The code is fairly self-explanatory.

While it may seem non-sensical that all or nearly all of the points are falling out below the "true" line with slope = 1 and intercept = 0, that is because there are many false positive reads.

```
#Figure 4A: Family level
#fig4a=truepos.krak.family%>%
#  ggplot()+
#  geom_point(aes(pollen.grain.proportion, quant.family, color=family, shape=family), size=4, alpha=0.4, p
#  xlab("proportion of pollen grains in sample")+
#  ylab("proportion of reads")+
```

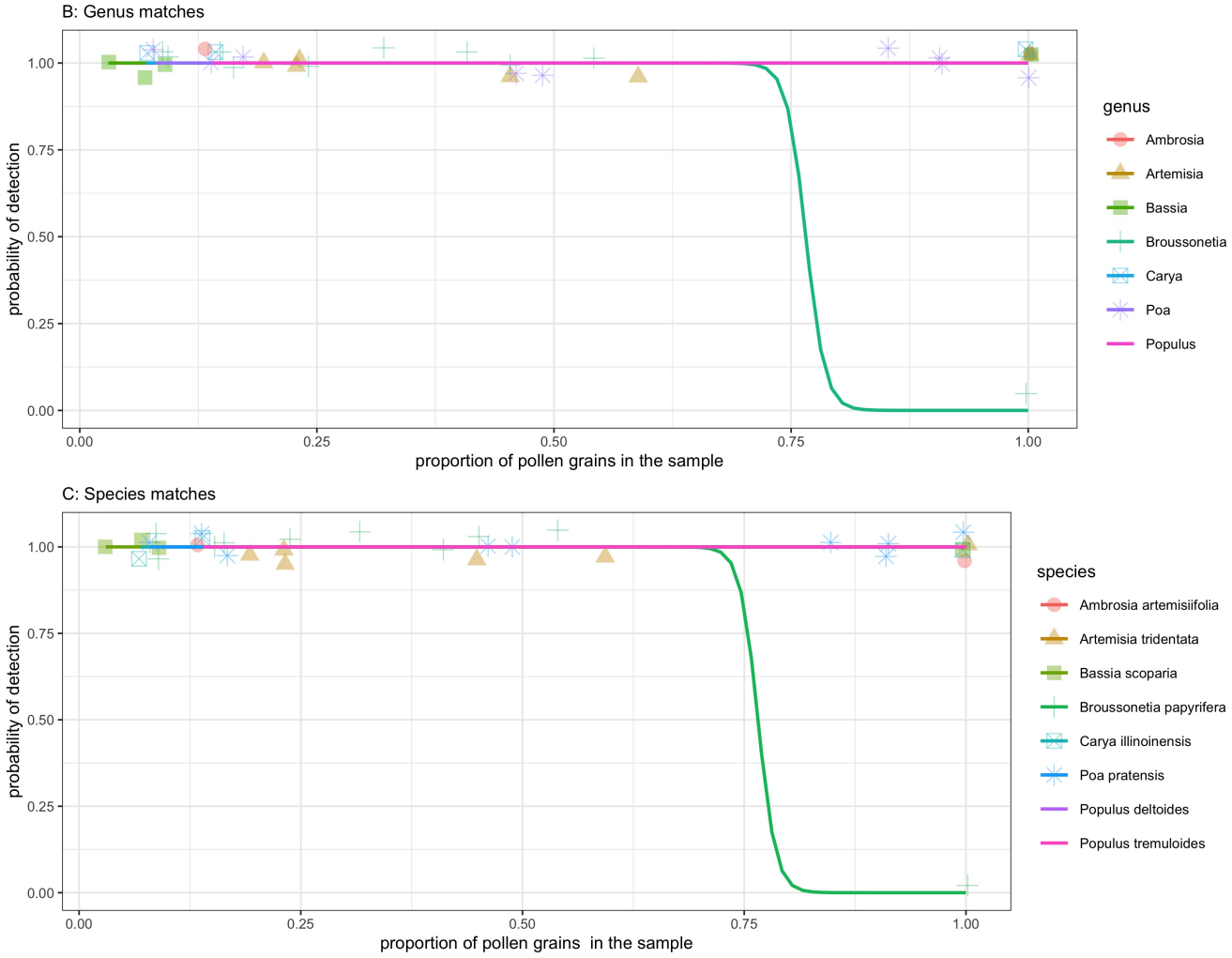


Figure 7: Figure 3

```
#  geom_abline(size=0.2, alpha=1) + #line w/ slope 1
#  labs(title = NULL, subtitle = "A: Family matches") +
#  scale_shape_manual(values=c(15,16,17,18,0,1,2,5,6)) +
#  theme_bw()

#Figure 4B: Genus level
fig4b=truepos.krak.genus%>%
  ggplot()+
  geom_point(aes(pollen.grain.proportion, quant.genus, color=genus, shape=genus), size=4, alpha=0.4, posit
  xlab("proportion of pollen grains in sample") +
  ylab("proportion of reads") +
  geom_abline(size=0.2, alpha=1) + #line w/ slope 1
  labs(title = NULL, subtitle = "B: Genus matches") +
  scale_shape_manual(values=c(15,16,17,18,0,1,2,5,6)) +
  theme_bw()

#Figure 4C: Species level
fig4c=truepos.krak.species%>%
  ggplot() +
  geom_point(aes(pollen.grain.proportion, quant.species, color=species, shape=species), size=4, alpha=0.4,
  xlab("proportion of pollen grains in sample")+
  ylab("proportion of reads") +
  geom_abline(size=0.2, alpha=1) + #line w/ slope 1
  labs(title = NULL, subtitle = "C: Species matches") +
  scale_shape_manual(values=c(15,16,17,18,0,1,2,5,6)) +
  theme_bw()

#Figure 4 Panel
#ggsave("WGS_fig4a.pdf", fig4a, device="pdf", width=7, height=5, units="in")
ggsave("WGS_fig4b.pdf", fig4b, device="pdf", width=7, height=5, units="in")
ggsave("WGS_fig4c.pdf", fig4c, device="pdf", width=7, height=5, units="in")

# save panel
panel.fig4 = grid.arrange(#fig4a,
  fig4b, fig4c,ncol=1) #  heights = rep(50,3)
```

```
suppressMessages(ggsave("fig4_combined.jpg", plot=panel.fig4, device="jpeg", height=11, width = 11, units=
```

##figure 5: true positives by source

This figure shows the proportion of correct taxonomic matchess by source (amplicon vs. WGS) at the species, genus, and family level. To correctly format the data, the mean and binomial CI of correct taxonomic matches needs to be summarized 3 times: once for sample, once for mix, and once for source. Otherwise samples with multiple species will get over-represented in terms of the overall means. The integers are tracked at each step and summarized with `dplyr`. The data is summarized with `dplyr` twice, once per mix and then once per sample. Then within each level of the factor of interest, the mean and binomial confidence intervals are calculated using `binom.confint`.

```
#Summarize the agg dataset first by source, mix.ID, and sample. Also calculate pool size for each sample (

#CI.krakamp.family.fn.1 = krakamp_family_fn %>%
#select(mix.ID, sample, qual.family, source) %>% group_by(source, mix.ID, sample) %>%
#summarize(pool.size = n(),
#  family = sum(as.numeric(qual.family)))

CI.krakamp.genus.fn.1 = krakamp_genus_fn %>%
select(mix.ID, sample, qual.genus, source) %>% group_by(source, mix.ID, sample) %>%
summarize(pool.size = n(),
```

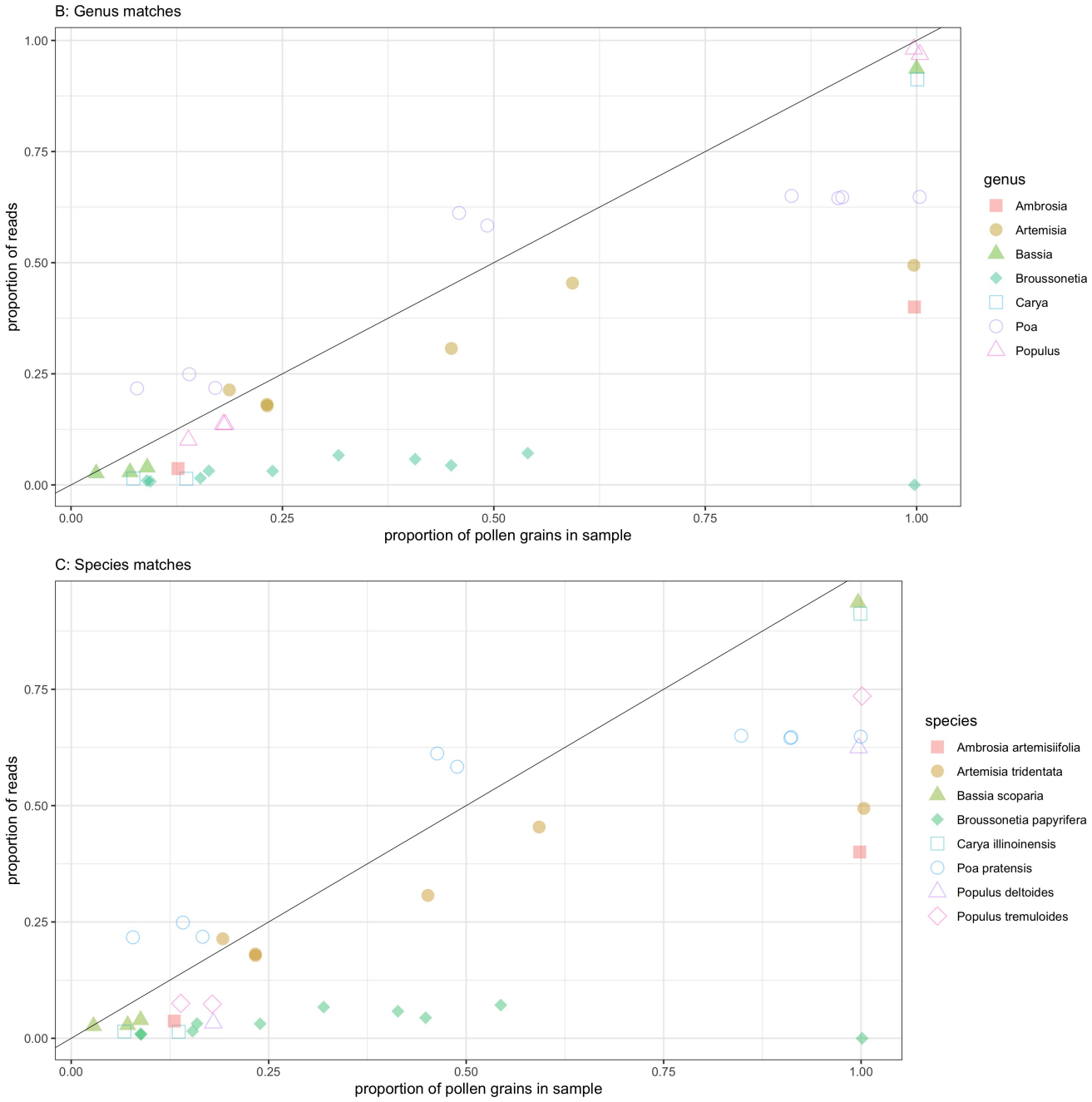


Figure 8: Figure 4

```
  genus = sum(as.numeric(qual.genus)))
```

```
## `summarise()` regrouping output by 'source', 'mix.ID' (override with `.groups` argument)
```

```
CI.krakamp.species.fn.1 = krakamp_species_fn %>%
select(mix.ID, sample, qual.species, source) %>% group_by(source, mix.ID, sample) %>%
summarize(pool.size = n(),
  species = sum(as.numeric(qual.species)))
```

```
## `summarise()` regrouping output by 'source', 'mix.ID' (override with `.groups` argument)
```

```
# second step: group_by(source, mix.ID, pool.size)

#CI.krakamp.family.fn.2 = CI.krakamp.family.fn.1 %>% group_by(source, mix.ID, pool.size) %>%
#  summarize(pool.num = n(),
#  family = sum(family))
  # multiply pool size & number to get total number of possibilities of matches
#  CI.krakamp.family.fn.2$n2 = CI.krakamp.family.fn.2$pool.size * CI.krakamp.family.fn.2$pool.num
  #format source to be factor
#  CI.krakamp.family.fn.2$source=as.factor(CI.krakamp.family.fn.2$source)

#  fig5_family=CI.krakamp.family.fn.2%>%
#    select(source, mix.ID, n2, family)

CI.krakamp.genus.fn.2 = CI.krakamp.genus.fn.1 %>% group_by(source, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
  genus = sum(genus))
```

```
## `summarise()` regrouping output by 'source', 'mix.ID' (override with `.groups` argument)
```

```
  # multiply pool size & number to get total number of possibilities of matches
  CI.krakamp.genus.fn.2$n2 = CI.krakamp.genus.fn.2$pool.size * CI.krakamp.genus.fn.2$pool.num
  #format source to be factor
  CI.krakamp.genus.fn.2$source=as.factor(CI.krakamp.genus.fn.2$source)

  fig5_genus=CI.krakamp.genus.fn.2%>%
    select(source, mix.ID, n2, genus)

CI.krakamp.species.fn.2 = CI.krakamp.species.fn.1 %>% group_by(source, mix.ID, pool.size) %>%
  summarize(pool.num = n(),
  species = sum(species))
```

```
## `summarise()` regrouping output by 'source', 'mix.ID' (override with `.groups` argument)
```

```
  # multiply pool size & number to get total number of possibilities of matches
  CI.krakamp.species.fn.2$n2 = CI.krakamp.species.fn.2$pool.size * CI.krakamp.species.fn.2$pool.num
  #format source to be factor
  CI.krakamp.species.fn.2$source=as.factor(CI.krakamp.species.fn.2$source)

  fig5_species=CI.krakamp.species.fn.2%>%
    select(source, mix.ID, n2, species)

  #Convert from wide to long
 # fig5_long_family = melt(fig5_family, id.vars = c("source","mix.ID", "n2"))
#  fig5_long_family = fig5_long_family%>%
#  rename(x=value, n=n2)

  fig5_long_genus = melt(fig5_genus, id.vars = c("source","mix.ID", "n2"))
```

```
## Warning in melt(fig5_genus, id.vars = c("source", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(fig5_genus). In the next version, this warning
## will become an error.
```

```
  fig5_long_genus = fig5_long_genus%>%
  rename(x=value, n=n2)

  fig5_long_species = melt(fig5_species, id.vars = c("source","mix.ID", "n2"))
```

```
## Warning in melt(fig5_species, id.vars = c("source", "mix.ID", "n2")): The melt
## generic in data.table has been passed a grouped_df and will attempt to redirect
## to the relevant reshape2 method; please note that reshape2 is deprecated, and
## this redirection is now deprecated as well. To continue using melt methods from
## reshape2 while both libraries are attached, e.g. melt.list, you can prepend the
## namespace like reshape2::melt(fig5_species). In the next version, this warning
## will become an error.
```

```
  fig5_long_species = fig5_long_species%>%
  rename(x=value, n=n2)

  #group by source
#  fig5_long_family = fig5_long_family %>%
#    group_by(source) %>%
#    summarize(n=mean(n),x=mean(x)) %>%
#    ungroup()

  fig5_long_genus = fig5_long_genus %>%
    group_by(source) %>%
    summarize(n=mean(n),x=mean(x)) %>%
    ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
  fig5_long_species = fig5_long_species %>%
    group_by(source) %>%
    summarize(n=mean(n),x=mean(x)) %>%
    ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
  #Calculate mean and binomial CI

 # fig5_bin.family = binom.confint(fig5_long_family$x, fig5_long_family$n,methods="exact") %>%
#    select(-method)

  fig5_bin.genus = binom.confint(fig5_long_genus$x, fig5_long_genus$n,methods="exact") %>%
    select(-method)

  fig5_bin.species = binom.confint(fig5_long_species$x, fig5_long_species$n,methods="exact") %>%
    select(-method)

  #merge datasets

#  fig5_family.all = merge(fig5_bin.family, fig5_long_family, by = c("x", "n"))
#  fig5_family.all = unique(fig5_family.all)
```

```
  fig5_genus.all = merge(fig5_bin.genus, fig5_long_genus, by = c("x", "n"))
  fig5_genus.all = unique(fig5_genus.all)

  fig5_species.all = merge(fig5_bin.species, fig5_long_species, by = c("x", "n"))
  fig5_species.all = unique(fig5_species.all)

#Figure 5A: Family level
#fig5a=fig5_family.all%>%
#  ggplot()+
#  geom_point(aes(source,mean))+
#  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
#  xlab("source")+
#  ylab("proportion of correct matches")+
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw()

#Figure 5B: Genus level
fig5b=fig5_genus.all%>%
  ggplot()+
  geom_point(aes(source,mean))+
  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
  xlab("source")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches") +
  theme_bw()

#Figure 5C: Species level
fig5c=fig5_species.all%>%
  ggplot()+
  geom_point(aes(source,mean))+
  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
  xlab("source")+
  ylab("proportion of correct matches")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches") +
  theme_bw()


#Create panel for Figure 5
fig5=grid.arrange(#fig5a,
  fig5b,fig5c, nrow=1)
```

```
suppressMessages(ggsave("fig5_combined.jpg", plot=fig5, device="jpeg", width=169, units="mm"))
```

Figure 6: false negatives by source

This figure shows the proportion of hits that were false positives. The "false positive" vs. "false negative" variables created previously are converted to proportion of correct matches. The mean and binomial CI of false positive counts are aggregated once by sample, once by mix, and once by source. Data summary and confidence intervals are calculated as in figure 5.

```
#Create variable called "falsepos" that is equivalent to "type"

#all.krak.family$falsepos = all.krak.family$type
```

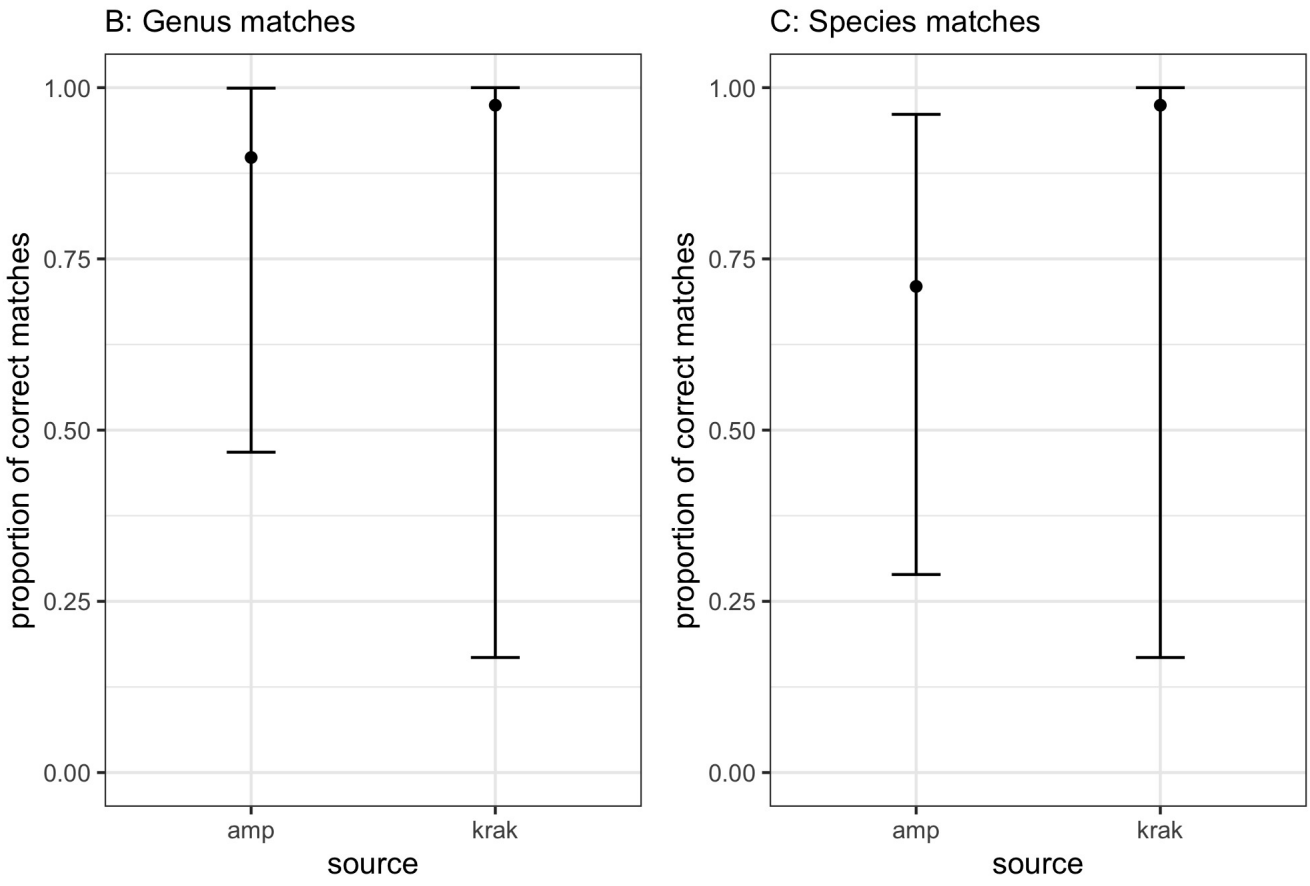


Figure 9: Figure 5

```
all.krak.genus$falsepos = all.krak.genus$type
all.krak.species$falsepos = all.krak.species$type

#Merge ITS and rbcL data
#amp_family_allmix = merge(amp_its_family_mix, amp_rbc_family_mix, by=c("mix.ID","taxa","sample"), all.x=T
amp_genus_allmix = merge(amp_its_genus_mix, amp_rbc_genus_mix, by=c("mix.ID","taxa","sample"), all.x=T, al
amp_species_allmix = merge(amp_its_species_mix, amp_rbc_species_mix, by=c("mix.ID","taxa","sample"), all.x

#Create hits variable that sums the hits for rbcL and ITS
#amp_family_allmix$hits = rowSums(amp_family_allmix[,c("hits.x","hits.y")], na.rm=T)
amp_genus_allmix$hits = rowSums(amp_genus_allmix[,c("hits.x","hits.y")], na.rm=T)
amp_species_allmix$hits = rowSums(amp_species_allmix[,c("hits.x","hits.y")], na.rm=T)

#Create false positive variable
#amp_family_allmix$falsepos = ifelse(!is.na(amp_family_allmix$type.x),amp_family_allmix$type.x,amp_family_
amp_genus_allmix$falsepos = ifelse(!is.na(amp_genus_allmix$type.x),amp_genus_allmix$type.x,amp_genus_allmi
amp_species_allmix$falsepos = ifelse(!is.na(amp_species_allmix$type.x),amp_species_allmix$type.x,amp_speci

#Rename variable "taxa"
#setnames(amp_family_allmix, "taxa", "family")
setnames(amp_genus_allmix, "taxa","genus")
setnames(amp_species_allmix, "taxa", "species")

#all.krak.family$source = "K"
#amp_family_allmix$source = "A"
all.krak.genus$source = "K"
amp_genus_allmix$source = "A"
all.krak.species$source = "K"
amp_species_allmix$source = "A"

#Merge amplicon and kraken data
#krakamp_family_fp <- rbind(all.krak.family[,c("mix.ID", "family", "sample", "falsepos", "source", "hits")
krakamp_genus_fp <- rbind(all.krak.genus[,c("mix.ID", "genus", "sample", "falsepos", "source", "hits")], a
krakamp_species_fp <- rbind(all.krak.species[,c("mix.ID", "species", "sample", "falsepos", "source", "hits

#Summarize the agg dataset first by source and mix ID

#CI.krakamp.family.fp = krakamp_family_fp %>%
#select(mix.ID, falsepos, source, hits) %>% group_by(source, mix.ID) %>%
#summarize(total.hits = sum(hits),
#          fp.hits = sum(hits[falsepos=="false_pos"]))

CI.krakamp.genus.fp = krakamp_genus_fp %>%
select(mix.ID, falsepos, source, hits) %>% group_by(source, mix.ID) %>%
summarize(total.hits = sum(hits),
          fp.hits = sum(hits[falsepos=="false_pos"]))
```

```
## `summarise()` regrouping output by 'source' (override with `.groups` argument)
```

```
CI.krakamp.species.fp = krakamp_species_fp %>%
select(mix.ID, falsepos, source, hits) %>% group_by(source, mix.ID) %>%
summarize(total.hits = sum(hits),
          fp.hits = sum(hits[falsepos=="false_pos"]))
```

```
## `summarise()` regrouping output by 'source' (override with `.groups` argument)
```

```
#Convert from wide to long
#fig6_long_family = melt(CI.krakamp.family.fp, id.vars = c("source","mix.ID", "total.hits"))
#  fig6_long_family = fig6_long_family%>%
#  rename(x=value, n=total.hits)

fig6_long_genus = melt(CI.krakamp.genus.fp, id.vars = c("source","mix.ID", "total.hits"))
```

```
## Warning in melt(CI.krakamp.genus.fp, id.vars = c("source", "mix.ID",
## "total.hits")): The melt generic in data.table has been passed a grouped_df
## and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as
## well. To continue using melt methods from reshape2 while both libraries
## are attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(CI.krakamp.genus.fp). In the next version, this warning will
## become an error.
```

```
  fig6_long_genus = fig6_long_genus%>%
  rename(x=value, n=total.hits)

fig6_long_species = melt(CI.krakamp.species.fp, id.vars = c("source","mix.ID", "total.hits"))
```

```
## Warning in melt(CI.krakamp.species.fp, id.vars = c("source", "mix.ID",
## "total.hits")): The melt generic in data.table has been passed a grouped_df
## and will attempt to redirect to the relevant reshape2 method; please note
## that reshape2 is deprecated, and this redirection is now deprecated as
## well. To continue using melt methods from reshape2 while both libraries
## are attached, e.g. melt.list, you can prepend the namespace like
## reshape2::melt(CI.krakamp.species.fp). In the next version, this warning will
## become an error.
```

```
  fig6_long_species = fig6_long_species%>%
  rename(x=value, n=total.hits)

#group by source
# fig6_long_family = fig6_long_family %>%
#    group_by(source) %>%
#    summarize(n=mean(n),x=mean(x)) %>%
#    ungroup()

fig6_long_genus = fig6_long_genus %>%
    group_by(source) %>%
    summarize(n=mean(n),x=mean(x)) %>%
    ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

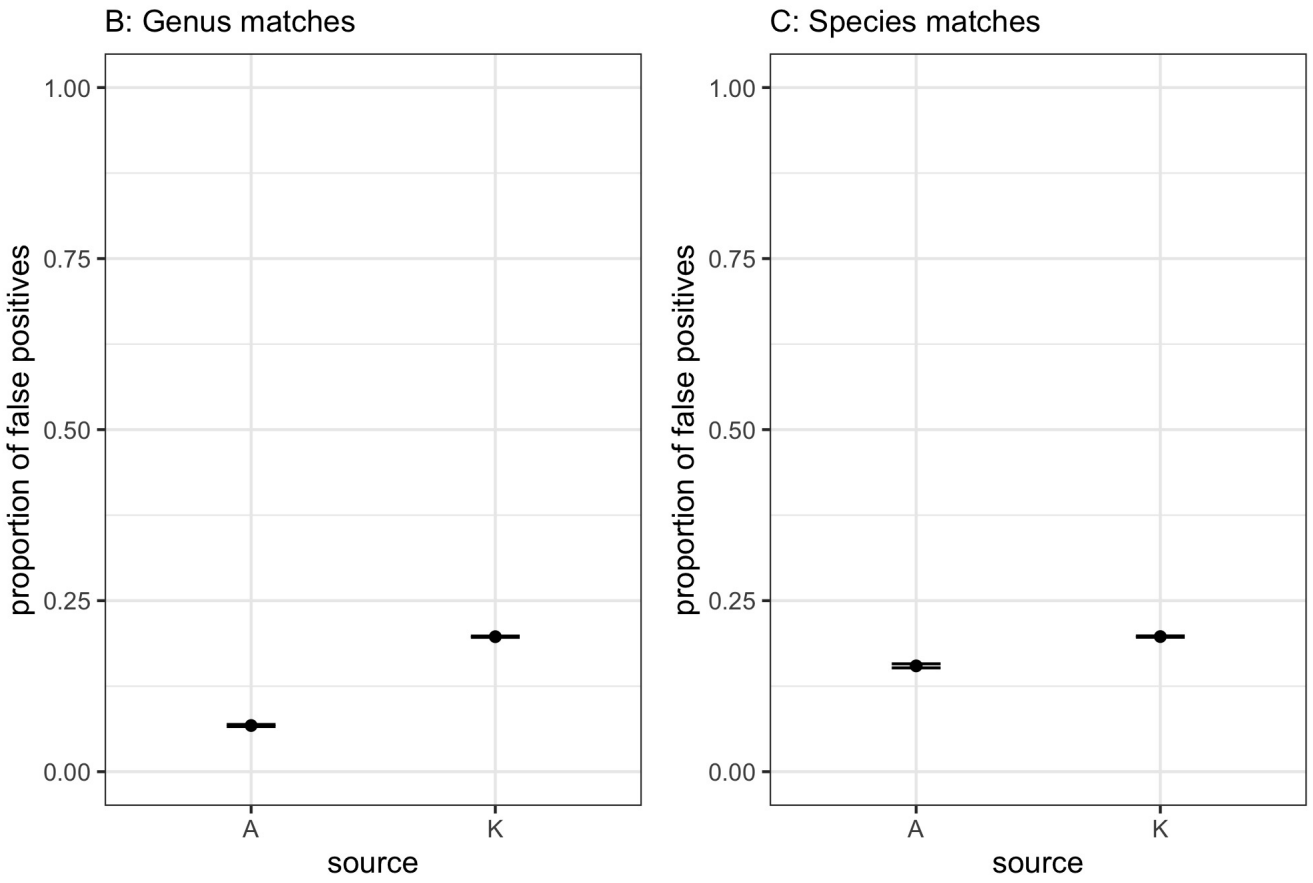
```
fig6_long_species = fig6_long_species %>%
    group_by(source) %>%
    summarize(n=mean(n),x=mean(x)) %>%
    ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
 #Calculate mean and binomial CI

#fig6_bin.family = binom.confint(fig6_long_family$x, fig6_long_family$n,methods="exact") %>%
#    select(-method)

fig6_bin.genus = binom.confint(fig6_long_genus$x, fig6_long_genus$n,methods="exact") %>%
```

```
    select(-method)

fig6_bin.species = binom.confint(fig6_long_species$x, fig6_long_species$n,methods="exact") %>%
    select(-method)

  #merge datasets

#  fig6_family.all = merge(fig6_bin.family, fig6_long_family, by = c("x", "n"))
#  fig6_family.all = unique(fig6_family.all)

  fig6_genus.all = merge(fig6_bin.genus, fig6_long_genus, by = c("x", "n"))
  fig6_genus.all = unique(fig6_genus.all)

  fig6_species.all = merge(fig6_bin.species, fig6_long_species, by = c("x", "n"))
  fig6_species.all = unique(fig6_species.all)

#Figure 6A: Family level
#fig6a=fig6_family.all%>%
#  ggplot()+
#  geom_point(aes(source,mean))+
#  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
#  xlab("source")+
#  ylab("proportion of false positives")+
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw()

#Figure 6B: Genus level
fig6b=fig6_genus.all%>%
  ggplot()+
  geom_point(aes(source,mean))+
  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
  xlab("source")+
  ylab("proportion of false positives")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches") +
  theme_bw()

#Figure 6C: Species level
fig6c=fig6_species.all%>%
  ggplot()+
  geom_point(aes(source,mean))+
  geom_errorbar(aes(source,ymin=lower, ymax=upper), width=0.2)+
  xlab("source")+
  ylab("proportion of false positives")+
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches") +
  theme_bw()


#Create panel for Figure 6
fig6=grid.arrange(#fig6a,
  fig6b,fig6c, nrow=1)
```

```
suppressMessages(ggsave("fig6_combined.jpg", plot=fig6, device="jpeg", width=169, units="mm"))
```



Note that the confidence intervals here are tiny, relative to the false negatives. That is because of the large counts of true and false positive reads in these data, as compared with a single count (1) per sample if a taxon was found that was truly present in that sample, in the false negatives analysis. In binomial data, it is very intuitive that as counts increase confidence intervals go down.

data prep for false positives by sample complexity

approach:

- use the data we put together for fig 6 (`CI.krakamp.family.fp`)
- subset to include only the kraken data (i.e. not the amplicon data)
- merge these data with the `mixes` metadata
- for rarity, create new `rarity` column based on binning `pollen.grain.proportion`

then follow the basic approach for e.g. Fig 2:

- will need to alter how the `x` and `n` are calculated relative to the true positive / false negative analysis
- here we have basically already calculated them, but need to **add** up hits of false positives and false negatives to summarize within each level (e.g. for a given level of species richness)—there are already steps to do the summarizing, just need to make sure it happens via addition
- then run `binom.confint` (from the `binomial` package) to get a mean and CI for each level
- then plot

```
# data prep
# family
#falsepos.plot.fam = CI.krakamp.family.fp %>% filter(source == "K")
#falsepos.plot.fam = merge(falsepos.plot.fam, mixes, by = "mix.ID", all.y = F)
#falsepos.plot.fam$rarity = cut(falsepos.plot.fam$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0.75,

# genus
```

```
falsepos.plot.gen = CI.krakamp.genus.fp %>% filter(source == "K")
falsepos.plot.gen = merge(falsepos.plot.gen, mixes, by = "mix.ID", all.y = F)
falsepos.plot.gen$rarity = cut(falsepos.plot.gen$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0.75, 1

# species
falsepos.plot.spp = CI.krakamp.species.fp %>% filter(source == "K")
falsepos.plot.spp = merge(falsepos.plot.spp, mixes, by = "mix.ID", all.y = F)
falsepos.plot.spp$rarity = cut(falsepos.plot.spp$pollen.grain.proportion, breaks = c(0, 0.25, 0.5, 0.75, 1
```

Figure I: false positives vs. species richness

```
# summarize for each level of species richness:
# family:
#falsepos.rich.plot.fam = falsepos.plot.fam %>% group_by(spp.rich) %>% summarize(total.hits = sum(total.hi
# genus
falsepos.rich.plot.gen = falsepos.plot.gen %>% group_by(spp.rich) %>% summarize(total.hits = sum(total.hit
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# species
falsepos.rich.plot.spp = falsepos.plot.spp %>% group_by(spp.rich) %>% summarize(total.hits = sum(total.hit
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# convert species richness to factor for better plotting:
#falsepos.rich.plot.fam$spp.rich = factor(falsepos.rich.plot.fam$spp.rich)
falsepos.rich.plot.gen$spp.rich = factor(falsepos.rich.plot.gen$spp.rich)
falsepos.rich.plot.spp$spp.rich = factor(falsepos.rich.plot.spp$spp.rich)

# binomial confidence intervals (and mean)
# family:
#falsepos.rich.plot.fam = data.frame(falsepos.rich.plot.fam, binom.confint(falsepos.rich.plot.fam$fp.hits,
# genus:
falsepos.rich.plot.gen = data.frame(falsepos.rich.plot.gen, binom.confint(falsepos.rich.plot.gen$fp.hits, 
# species:
falsepos.rich.plot.spp = data.frame(falsepos.rich.plot.spp, binom.confint(falsepos.rich.plot.spp$fp.hits, 

#Figure IA: Family level
#figIa = falsepos.rich.plot.fam %>%
#  ggplot()+
#  geom_point(aes(spp.rich,mean))+
#  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
#  xlab("species richness") +
#  ylab("proportion of false positives") +
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches") +
#  theme_bw()

#Figure IB: Genus level
figIb = falsepos.rich.plot.gen %>%
  ggplot()+
  geom_point(aes(spp.rich,mean))+
  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
  xlab("species richness") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches")+
```

```
  theme_bw()

#Figure Ic: Species level
figIc = falsepos.rich.plot.spp %>%
  ggplot()+
  geom_point(aes(spp.rich,mean))+
  geom_errorbar(aes(spp.rich,ymin=lower, ymax=upper), width=0.2)+
  xlab("species richness") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches") +
  theme_bw()

#Create panel for Figure I
figI = grid.arrange(#figIa,
  figIb,figIc, nrow=1)
```

```
suppressMessages(ggsave("figI_combined.jpg", plot=figI, device="jpeg", width=169, units="mm"))
```

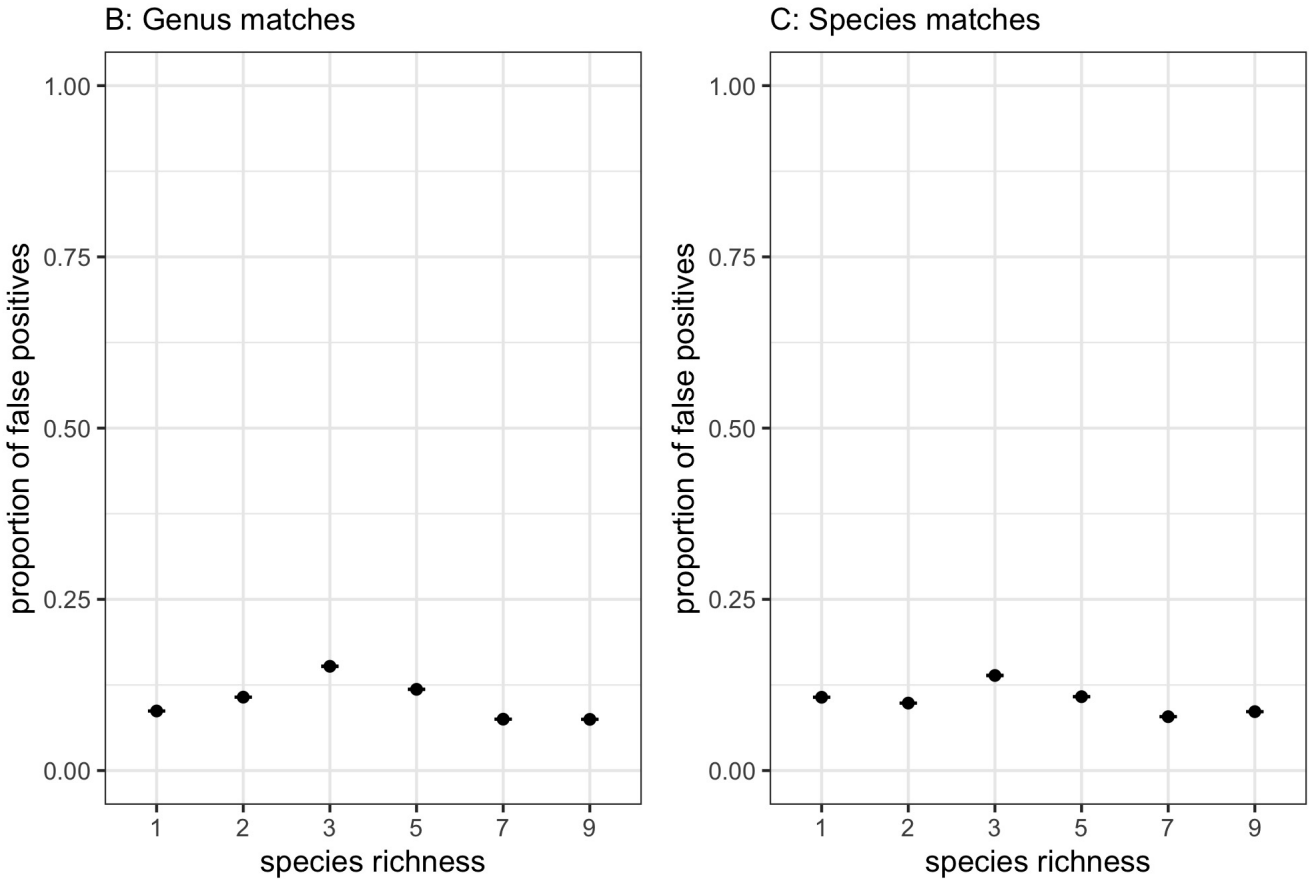


Figure 10: Figure I

Figure J: false positives vs. relatedness

```
# summarize for each level of relatedness:
# family:
```

```
#falsepos.relat.plot.fam = falsepos.plot.fam %>% group_by(relatedness) %>% summarize(total.hits = sum(tota
# genus
falsepos.relat.plot.gen = falsepos.plot.gen %>% group_by(relatedness) %>% summarize(total.hits = sum(total
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# species
falsepos.relat.plot.spp = falsepos.plot.spp %>% group_by(relatedness) %>% summarize(total.hits = sum(total
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# convert relatedness to factor for better plotting:
#falsepos.relat.plot.fam$relatedness = factor(falsepos.relat.plot.fam$relatedness)
falsepos.relat.plot.gen$relatedness = factor(falsepos.relat.plot.gen$relatedness)
falsepos.relat.plot.spp$relatedness = factor(falsepos.relat.plot.spp$relatedness)

# binomial confidence intervals (and mean)
# family:
#falsepos.relat.plot.fam = data.frame(falsepos.relat.plot.fam, binom.confint(falsepos.relat.plot.fam$fp.hi
# genus:
falsepos.relat.plot.gen = data.frame(falsepos.relat.plot.gen, binom.confint(falsepos.relat.plot.gen$fp.hit
# species:
falsepos.relat.plot.spp = data.frame(falsepos.relat.plot.spp, binom.confint(falsepos.relat.plot.spp$fp.hit


#Figure JA: Family level
#figJa = falsepos.relat.plot.fam %>%
#  ggplot()+
#  geom_point(aes(relatedness,mean))+
#  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
#  xlab("relatedness") +
#  ylab("proportion of false positives") +
#  ylim(0,1)+
  # scale_x_discrete(labels=paste(unique(falsepos.relat.plot.fam$relatedness))) +
#  labs(title = NULL, subtitle = "A: Family matches")+
#  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
#  theme_bw()
#figJa = figJa + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure JB: Genus level
figJb = falsepos.relat.plot.gen %>%
  ggplot()+
  geom_point(aes(relatedness,mean))+
  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
  xlab("relatedness") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches") +
  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
  theme_bw()
figJb = figJb + theme(axis.text.x = element_text(angle = 45, hjust = 1))


#Figure Jc: Species level
figJc = falsepos.relat.plot.spp %>%
  ggplot()+
  geom_point(aes(relatedness,mean))+
```

```
  geom_errorbar(aes(relatedness,ymin=lower, ymax=upper), width=0.2)+
  xlab("relatedness") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches")+
  scale_x_discrete(labels=c("same genus", "same family", "same order", "dif. orders")) +
  theme_bw()
figJc = figJc + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Create panel for Figure J
figJ = grid.arrange(#figJa,
  figJb,figJc, nrow=1)
```

```
suppressMessages(ggsave("figJ_combined.jpg", plot=figJ, device="jpeg", width=169, units="mm"))
```

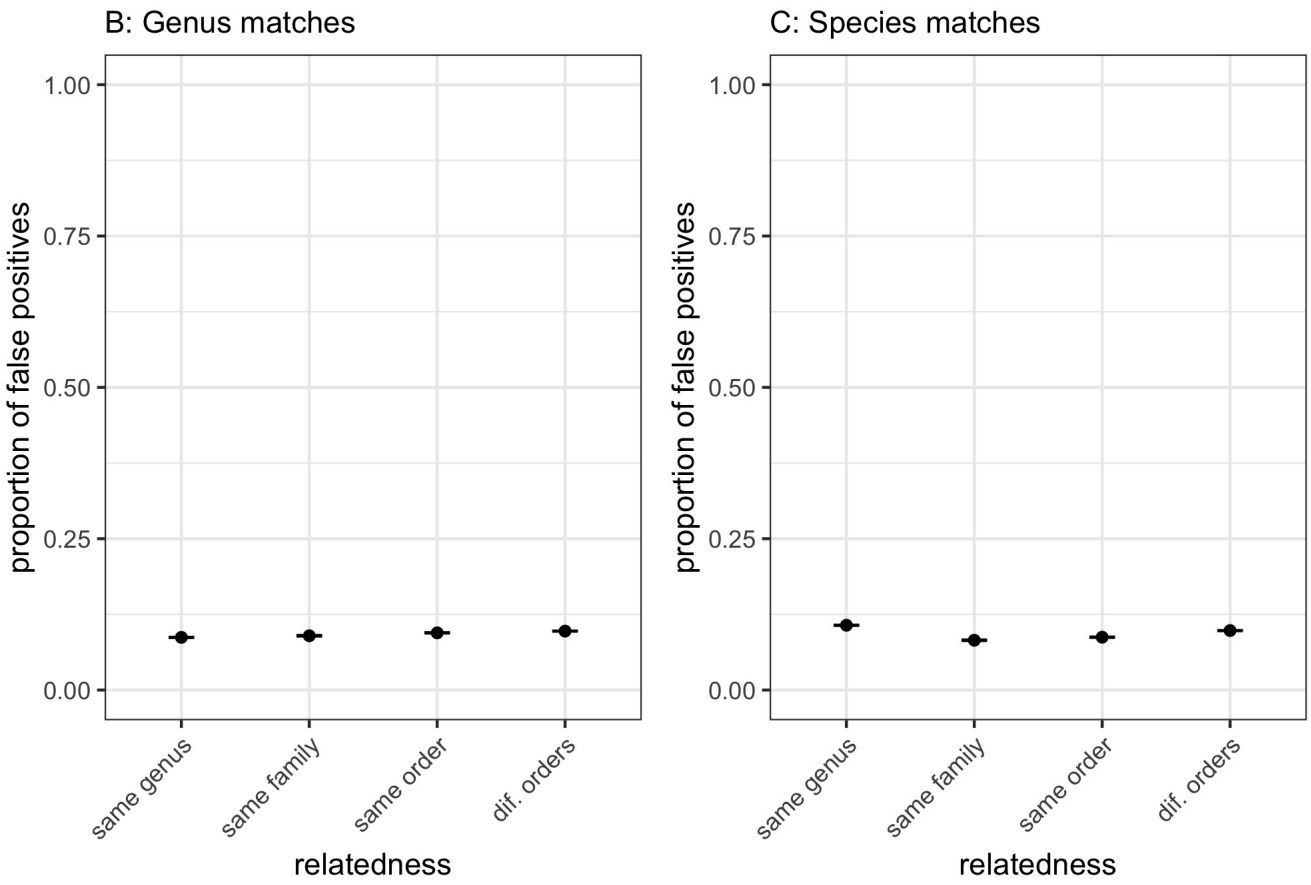


Figure 11: Figure J

Again, the confidence intervals are tiny as explained in the previous figure. We do not remark upon these more in subsequent figures.

Figure K: false positives vs. rarity (binned)

```
# summarize for each level of rarity:
# family:
#falsepos.rarity.plot.fam = falsepos.plot.fam %>% group_by(rarity) %>% summarize(total.hits = sum(total.hi
```

```
# genus
falsepos.rarity.plot.gen = falsepos.plot.gen %>% group_by(rarity) %>% summarize(total.hits = sum(total.hit
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# species
falsepos.rarity.plot.spp = falsepos.plot.spp %>% group_by(rarity) %>% summarize(total.hits = sum(total.hit
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# convert rarity to factor for better plotting:
#falsepos.rarity.plot.fam$rarity = factor(falsepos.rarity.plot.fam$rarity)
falsepos.rarity.plot.gen$rarity = factor(falsepos.rarity.plot.gen$rarity)
falsepos.rarity.plot.spp$rarity = factor(falsepos.rarity.plot.spp$rarity)

# binomial confidence intervals (and mean)
# family:
#falsepos.rarity.plot.fam = data.frame(falsepos.rarity.plot.fam, binom.confint(falsepos.rarity.plot.fam$fp
# genus:
falsepos.rarity.plot.gen = data.frame(falsepos.rarity.plot.gen, binom.confint(falsepos.rarity.plot.gen$fp.
# species:
falsepos.rarity.plot.spp = data.frame(falsepos.rarity.plot.spp, binom.confint(falsepos.rarity.plot.spp$fp.



#Figure KA: Family level
#figKa = falsepos.rarity.plot.fam %>%
#  ggplot()+
#  geom_point(aes(rarity,mean))+
#  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2)+
#  xlab("minimum rarity") +
#  ylab("proportion of false positives") +
#  ylim(0,1)+
#  labs(title = NULL, subtitle = "A: Family matches")+
#  theme_bw()
# figKa = figKa + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure KB: Genus level
figKb = falsepos.rarity.plot.gen %>%
  ggplot()+
  geom_point(aes(rarity,mean))+
  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2)+
  xlab("minimum rarity") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "B: Genus matches")+
  theme_bw()
figKb = figKb + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Figure Kc: Species level
figKc = falsepos.rarity.plot.spp %>%
  ggplot()+
  geom_point(aes(rarity,mean))+
  geom_errorbar(aes(rarity,ymin=lower, ymax=upper), width=0.2)+
  xlab("minimum rarity") +
  ylab("proportion of false positives") +
  ylim(0,1)+
  labs(title = NULL, subtitle = "C: Species matches")+
  theme_bw()
```

```
figKc = figKc + theme(axis.text.x = element_text(angle = 45, hjust = 1))

#Create panel for Figure K
figK = grid.arrange(#figKa,
  figKb,figKc, nrow=1)
```

```
suppressMessages(ggsave("figK_combined.jpg", plot=figK, device="jpeg", width=169, units="mm"))
```

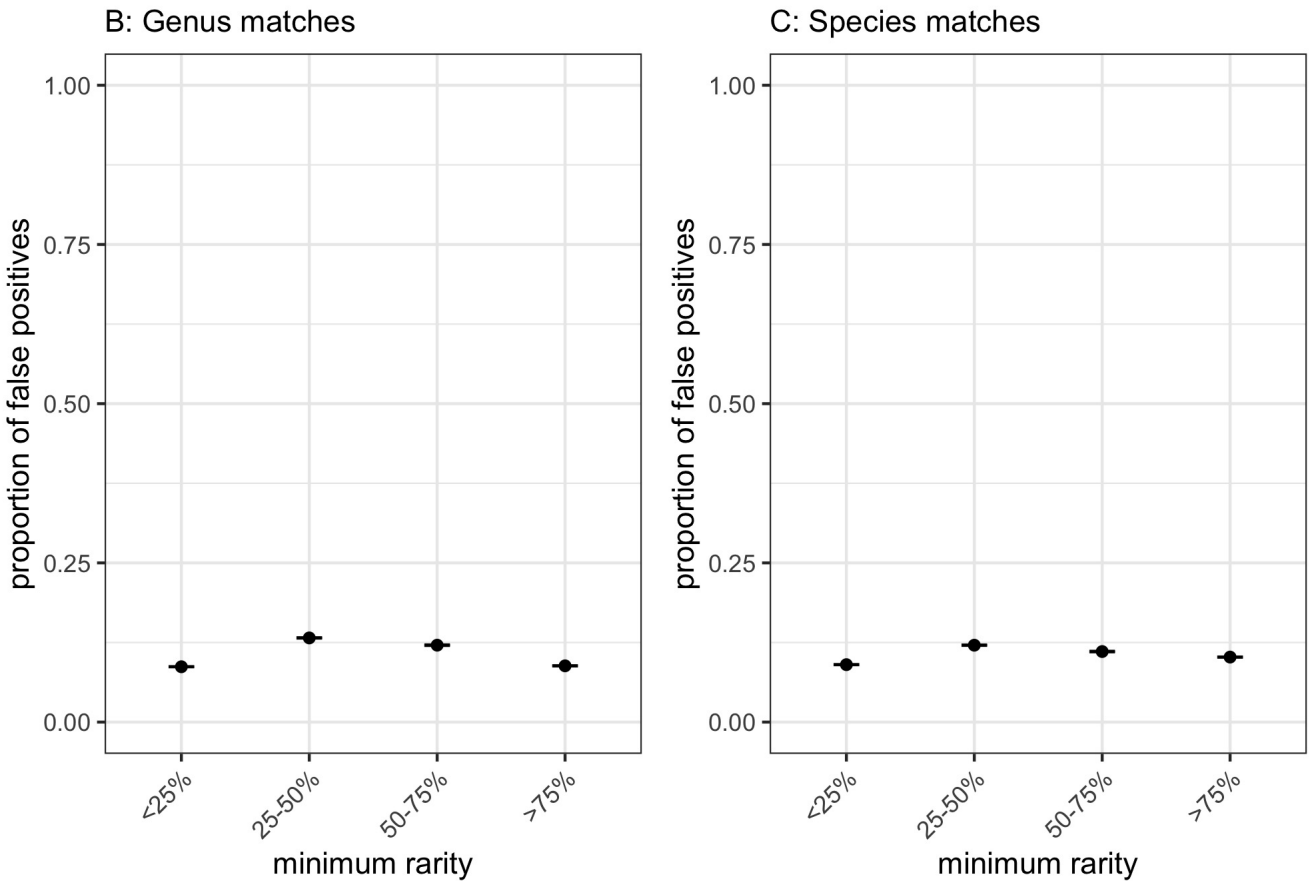


Figure 12: Figure K

Figure S2: false positives combined

Here we are putting together a figure comprised of panels we have already built in the sections for Figs I, J, and K. This figure has 9 panels, with a row for each of the 3 facets of sample complexity and a column for each of the 3 levels of taxonomic matching. Very straightforward.

```
# re-label panels:
#figJa = figJa + labs(title = NULL, subtitle = "D: Family matches")
figJb = figJb + labs(title = NULL, subtitle = "E: Genus matches")
figJc = figJc + labs(title = NULL, subtitle = "F: Species matches")
#figKa = figKa + labs(title = NULL, subtitle = "H: Family matches")
figKb = figKb + labs(title = NULL, subtitle = "I: Genus matches")
figKc = figKc + labs(title = NULL, subtitle = "J: Species matches")

#Create panel for Figure S2
figS2 = grid.arrange(#figIa,
```

```
  figIb, figIc, #figJa,
  figJb, figJc, #figKa,
  figKb, figKc, nrow=3)
```

```
suppressMessages(ggsave("figS2_combined.jpg", plot=figS2, device="jpeg", width=169, height = 250, units="m
```

Session Information

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] broom.mixed_0.2.6 gridExtra_2.3     ggplot2_3.3.2     binom_1.1-1
##  [5] data.table_1.13.4 tibble_3.0.4      tidyr_1.1.2       knitr_1.30
##  [9] r2glmm_0.1.2      lmerTest_3.1-3    lme4_1.1-26       Matrix_1.2-18
## [13] reshape2_1.4.4    stringr_1.4.0     dplyr_1.0.2       kableExtra_1.3.1
##
## loaded via a namespace (and not attached):
##  [1] statmod_1.4.35      TMB_1.7.18          tidyselect_1.1.0
##  [4] xfun_0.19           purrr_0.3.4         splines_4.0.3
##  [7] lattice_0.20-41     colorspace_2.0-0    vctrs_0.3.6
## [10] generics_0.1.0      htmltools_0.5.0     viridisLite_0.3.0
## [13] mgcv_1.8-33         yaml_2.2.1          rlang_0.4.9
## [16] pillar_1.4.7        nloptr_1.2.2.2      glue_1.4.2
## [19] withr_2.3.0         lifecycle_0.2.0     plyr_1.8.6
## [22] munsell_0.5.0       gtable_0.3.0        rvest_0.3.6
## [25] coda_0.19-4         evaluate_0.14       labeling_0.4.2
## [28] broom_0.7.3         Rcpp_1.0.5          backports_1.2.1
## [31] scales_1.1.1        webshot_0.5.2       farver_2.0.3
## [34] digest_0.6.27       stringi_1.5.3       numDeriv_2016.8-1.1
## [37] grid_4.0.3          tools_4.0.3         magrittr_2.0.1
## [40] crayon_1.3.4        pkgconfig_2.0.3     ellipsis_0.3.1
## [43] MASS_7.3-53         xml2_1.3.2          minqa_1.2.4
## [46] rmarkdown_2.6       httr_1.4.2          rstudioapi_0.13
## [49] R6_2.5.0            boot_1.3-25         nlme_3.1-149
## [52] compiler_4.0.3
```

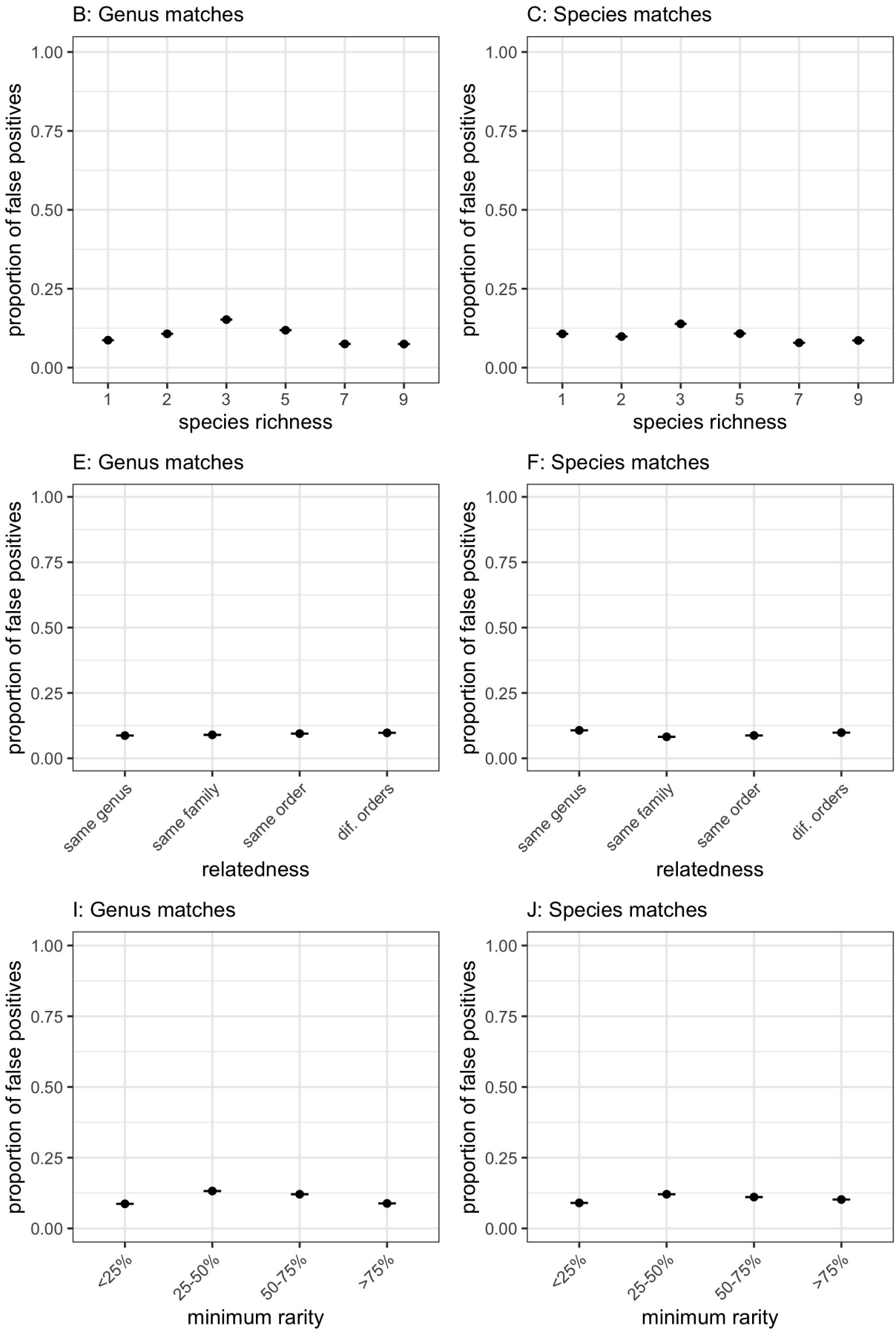


Figure 13: Figure S2