Bioinformatics in R

Matthias Obst (originally developed by Ellika Faust), University of Gothenburg

11 November 2020

Objective

This is a 4-6 hours exercise and will introduce you to natural history research typically conducted with museum specimens. In this case it includes biogeographic analysis and phylogenetic inference - so you will learn how to plot maps and build phylogenetic trees. The tutorial below describes the “how”. Before and after the exercise you should read the three papers in the “literature” folder that will tell you about the “why”. Discuss you findings with your peers.

1 Getting started

Start with installation of RStudio and follow the associated file “Installation instructions”. Make sure you have a recent version of R. Preferably you should have the latest version 3.6.3, but anything above ≥ 3.2.1 should work. You can check your R version by typing:

```
R.version.string
## [1] "R version 3.6.1 (2019-07-05)"
```

1.1 R basics, workspace and working directory, Rstudio project

First we are going to start with going to an already existing tutorial created by the University of British Columbia. It gives some quick insights into R and the Rstudio working environment. Click on the link and read through the tutorial and follow along in Rstudio.

https://www.stat.ubc.ca/~jenny/STAT545A/block01_basicsWorkspaceWorkingDirProject.html#r-basics-workspace-and-working-directory-rstudio-projects

1.2 Packages

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used.

Install the following packages:

```
install.packages("swirl")
install.packages("ape")
install.packages("phangorn")
install.packages("seqinr")
install.packages("BiocManager")
BiocManager::install("msa")
install.packages("tidyverse")
install.packages("grDevices")
install.packages("maps")
```

Now you can load the packages

```
library(swirl)
library(msa)
library(ape)
library(phangorn)
library(seqinr)
library(tidyverse)
library(grDevices)
library(maps)
```

If you get an error when trying to load a package it's likely that the installation failed so try and install the package again. If it asks you to install an additional package install that too. If it still doesn't work and you can't figure out why, contact one of the teachers and ask for help.

If at some point you are unsure about the version of the package, you can check it using:

```
packageDescription("swirl", fields = "Version")
## [1] "2.4.5"
```

1.3 Getting help

There are several ways of getting information about R in general. The function `help.search` is used to look for help on any given topic. For instance:

```
help.search("alignment")
```

replies with a list of functions from different packages. To get help for a given function, use `help()` or `?` followed by the name of the function you are interested in.

```
help(readDNAStringSet)
?readDNAStringSet
```

will open up the help of the function. At the end of a manual page, an 'example' section often shows how to use a function.

If the function you are interested in is part of a package you haven't loaded into your working environment R will tell you

```
No documentation for 'package_name' in specified packages
and libraries: you could try '??package_name'
```

in which case you simply use `??`

```
??readDNAStringSet
```

1.4 Sample Data

R comes with a number of sample data sets that you can experiment with. Type `data( )` to see the available data sets. The results will depend on which packages you have loaded as many packages have their own sample data you can explore. Type `help(datasetname)` for details on a specific sample data set.

1.5 Tips, tricks and things to remember

- `alt` + `-` to create <- (on Mac it might be `option` + `-`)
- press tab for auto completion
- use up arrow to recycle through recent commands
- `help()` or `?`
- R iS cAsE SeNsAtIvE
- use "" to let R know what is a string of text like “this”
- if you want to save your code you can write everything inside a `script`
- to execute from a script press PC- `control` + `enter` Mac- `command` + `enter`

If you get an error do the following

1. Check if your spelling correct?
2. Ask your neighbour to check if your spelling is correct, 4 eyes are better than 2
3. Are you in the right directory? `getwd()`
4. Is the path to the input / output file correct?
5. Check the `help()` manual for your function
6. Copy the error message and google it, your unlikely to be the first with this error
7. Are you really sure the spelling is correct?
8. Ask one of the instructors for help.

2 Making maps

If you haven't already, now is a good time to start a new R project and a clean working directory where you can put all your files

open the folder associated to this tutorial called `data` and put it in your working directory. Inside your data folder you should have two files, one called HSCrabs_CO1.fas which contains 59 sequences from horseshoe crabs from 10 different locations. There is also a file called sample_info.csv with a table of information on all the sampling locations, listing their identification code, location, country and latitude and longitude. We will start with having a look at the sample_info.csv file. You can open it in excel or a text editor to have look ate the table before we load it into R.

Windows default text editor is Notepad, or Anteckningar in Swedish
Mac default text editor is TextEdit, or Textredigerare in Swedish (I think)

When working with samples from different areas it can be very helpful to plot it on a map to get a better understanding of what you are working with. To do this first load the sample_info.csv file in to R

```
info_df <- read.csv("data/sample_info.csv")
info_df
```

```
##       Code     Location  Country  Latitude Longitude
Species
## 1   CR_V1      Bac Lieu  Vietnam  9.265834 105.95895
C. rotundicauda
## 2   CR_T3       Bang Pu Thailand  6.900000 101.30000
C. rotundicauda
## 3   CR_T5        Phuket Thailand  7.850000  98.41667
C. rotundicauda
## 4  TG_T2         Ranong Thailand  9.789200  98.51580
T. gigas
## 5   TG_V1      Bac Lieu  Vietnam  9.265834 105.95895
```

```
T. gigas
## 6   TT_K1     Putoushan    China 28.132872 121.11760
T. tridendatus
## 7   TT_K2     Sinjahmen    China 29.073831 121.74975
T. tridendatus
## 8   LP_M2  Rio Lagartos   Mexico 21.603627 -88.14559
L. polyphemus
## 9    LP_F1      Florida      USA 28.864783 -80.83136
L. polyphemus
## 10   LP_W    Woods Hole      USA 41.516667 -70.70000
L. polyphemus
##    Sample.size  Ecology
## 1            4 Mangrove
## 2           10 Mangrove
## 3            8 Mangrove
## 4            6  Coastal
## 5            2  Coastal
## 6            4  Coastal
## 7            6  Coastal
## 8            6  Coastal
## 9            6  Coastal
## 10           7  Coastal
```

As you can see R easily reads in the file as a table and uses the first row as column names. This time it is what we want, but this is not always the case in which case you can use the argument `col.names = FALSE` If we want to access just one column there are many ways to do that but my favourite is to use the `$` symbol, why don't you try it out?

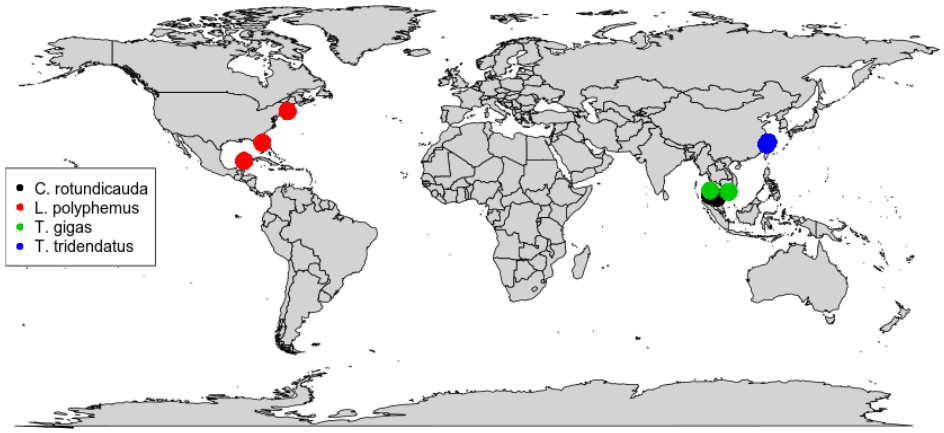
```
info_df$Species
```

Let's create some vectors that we can use for plotting

```
lon <- info_df$Longitude
lat <- info_df$Latitude
sp <- info_df$Species
code <- info_df$Code
```

Now it's time to make a map, in order to keep things simple we can simply use the `map()` function and plot a world map. But first we actually need to create another colour vector as the standard `palette` only has 8 colours and we have 10 different codes. We can use the `rainbow()` palette instead which comes with the package `grDevices`

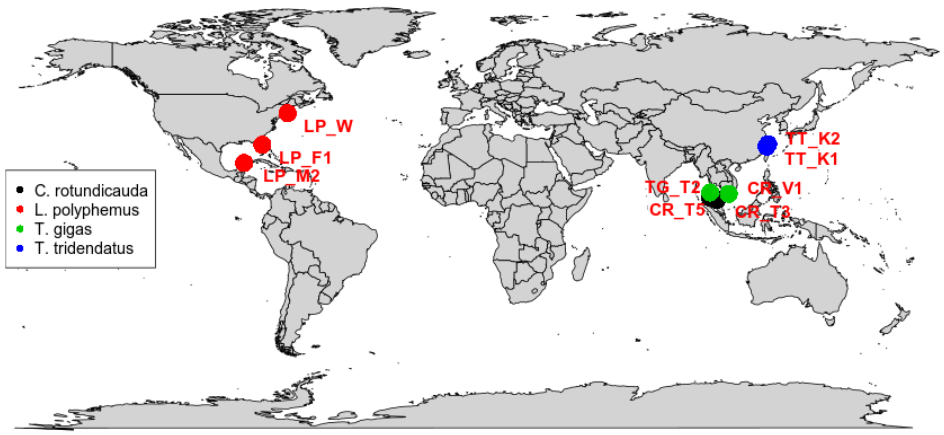
```
# map the world
map("world", fill=TRUE, col="lightgray", bg="white")
# add our points
points(x = lon, y = lat, col = sp, pch=16, cex = 3, )
#and add a legend
legend("left", legend = levels(sp), col = palette(),
pch = 16, cex = 1.2)
```



Now lets make the same map but also add the labels to the map. This can be a little cumbersome because our points are very close together and we don't want our labels to overlap.

```
# map the world
map("world", fill=TRUE, col="lightgray", bg="white", )
# add our points
points(x = lon, y = lat, col = sp, pch=16, cex = 3, )
#and add a legend
legend("left", legend = levels(sp), col = palette(),
pch = 16, cex = 1.2)
```

```
# make vectors of which labels you want placed where
right <- c(2,6,8,9,10)
left <- c(3)
left_up <- c(4)
right_up <- c(1,7)
#now add the labels
text(x = lon[right], y = lat[right], labels =
code[right], adj = c(-0.3,1.5), font = 2, cex = 1.3,
col = "red")
text(x = lon[left], y = lat[left], labels = code[left],
adj = c(1,1.5), font = 2, cex = 1.3, col = "red")
text(x = lon[right_up], y = lat[right_up], labels =
code[right_up], adj = c(-0.3,0), font = 2, cex = 1.3,
col = "red")
text(x = lon[left_up], y = lat[left_up], labels =
code[left_up], adj = c(1,0), font = 2, cex = 1.3, col =
"red")
```



2.0.1 Using ggplot

A very popular way of plotting is using something called ggplot, we wont go into any detail, but feel free to try it out

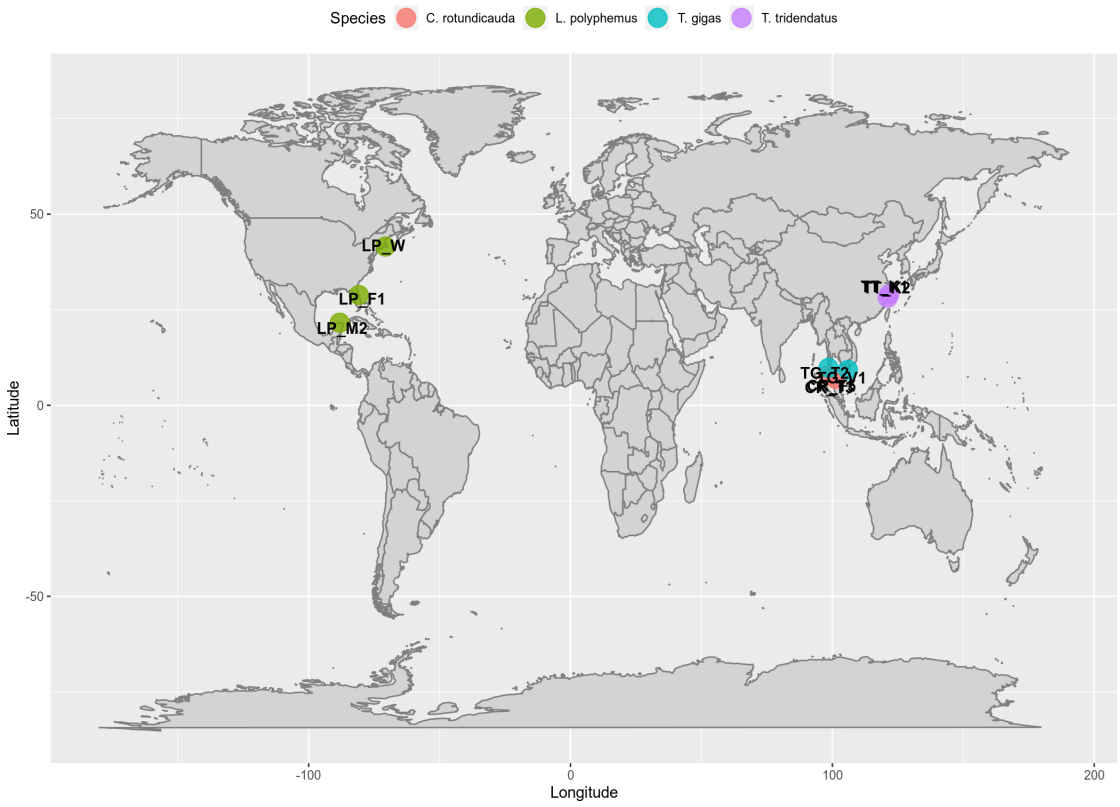
Extract the coordinates from all occurrence points and turn them into a spatial points object and load a world map using borders (gglpot2)

```
map_world <- borders(database = "world", colour =
```

```
"gray50", fill = "lightgray")
```

First we just call the function ggplot(), then tell it to plot our map_world object we created above, then add the data (info_df) using the geom_point() function, and finally we add a legend using the theme() command.

```
ggplot(data = info_df[-1,],  # Specify the data for geom_point()
               aes(x = Longitude,  # Specify the x axis as longitude
                   y = Latitude,
                   label = Code )) +
  map_world +
  geom_point(aes(colour = Species),  # Colour the points based on species name
             alpha = 0.8,  # Set point opacity to 80%
             size = 6) + # set size
  geom_text(fontface = "bold",position=position_jitter(width=3,height=3))+
  theme(legend.position = "top")
```



3 Alignment, BLAST and species diversity

Fasta files are text files used for nucleic acids and aminoacids. Sequences in Fasta start with a `>` symbol followed by a description (i.e., header). The next line contains the sequence itself. The following is an example of a DNA fasta file

To be able to compare these sequences we first need to align them.

Once the sequences have been aligned we can look for mutations, conserved regions, variable regions, among other things. We can also use these sequences for taxonomic classification. Particularly, the cytochrome c gene is useful when looking for the identity of metazoans.

Go to your `data` folder associated to this tutorial and open HSCrabs_CO1.fas with a text editor (Do not try and use excel for this file)

1. Can you see where one sequence ends and another starts?
2. In this case each sequence is from a different individual, what is the first individual called?
3. Each individual has identifier code following this pattern speciesID_locationID_SpecimentID. Open the .csv to check species, location and country of the first and last individual in the fasta file.

3.1 Multiple sequence alignment

Now lets start working with the sequences! Time to load the fasta file into R. This code assumes that inside your working directory there is a folder named `data` and inside that folder we have all our files with unchanged names. You can choose to organise it differently but then you will have to adjust the paths which tells R where to look (eg “data/data/HSCrabs_CO1.fas”).

```
# this simply saves the path to the file as a object
my_sequence_file <- "data/HSCrabs_CO1.fas"
#this reads in the fasta file using the assigned path above
my_sequences <- readDNAStringSet(my_sequence_file)
#this prints it to your console
my_sequences
```

```
##   A DNAStringSet instance of length 59
##      width seq
names
##  [1]   598 TGGAACTGC-
CCTCAGAATTTTA...ATCCCGCAGGAGGTGGTGATCC TT_K1_1
##  [2]   598 TGGAACTGC-
CCTCAGAATTTTA...ATCCCGCAGGAGGTGGTGATCC TT_K1_2
##  [3]   598 TGGAACTGC-CCTCAGAATTTTA...ATCCCGCAGGAGG-
GG-GATCC TT_K1_3
##  [4]   598 TGGAACTGC-
CCTCAGAATTTTA...ATCCCGCAGGAGGTGG-GATCC TT_K1_4
##  [5]   598
----------------------...ATCCCGCAGGAGGTGGTGATCC
TT_K2_1
##  ...   ... ...
## [55]   598
TAGGGACAGCTCTCAGAATCTTA...ACCCTGCAGGAGGGGGTGACCC LP_W_3
## [56]   598
TAGGGACAGCTCTCAGAATCTTA...ACCCTGCAGGAGGGGGTGACCC LP_W_4
## [57]   598
```

```
TAGGGACAGCTCTCAGAATCTTA...ACCCTGCAGGAGGGGGTGACCC LP_W_5
## [58]   598
TAGGGACAGCTCTCAGAATCTTA...ACCCTGCAGGAGGGGGTGACCC LP_W_6
## [59]   540
TAATGCCAAAGATATAGGAACTT...CTTTATTTGGATGAGCTGTTAC LP_W_7
```

Now it is time for the alignment. We will use a package called `msa` which allows you to align multiple sequences with many different alignment and substitution algorithms. For now we won't go into much detail here and simply use the default alignment and substitution matrix.

```
#Does the alignment
my_first_alignment <- msa(my_sequences)
## use default substitution matrix
# prints the alignment to the console
my_first_alignment
## CLUSTAL 2.1
##
## Call:
##    msa(my_sequences)
##
## MsaDNAMultipleAlignment with 59 rows and 657 columns
##      aln
names
##  [1]
--------------------------...TTGACCCTGCAGGAGGGGGTGACCC
TG_T2_1
##  [2]
--------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_1
##  [3]
--------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_3
##  [4]
--------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_4
##  [5]
```

```
-------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_5
##  [6]
-------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_6
##  [7]
-------------------------...TTGACCCTGCAGGAGGGGGTGACCC
LP_F1_2
##  [8]
-------------------------...------------------------
LP_M2_4
##  [9]
-------------------------...TTGACCCTGCAGGAGGGGGGACCCA
LP_M2_5
##  ... ...
## [52]
-------------------------...TTGACCCAGCAGGCGGTGGTGACCC
TG_T2_4
## [53]
-------------------------...TTGACC------------------
TG_T2_5
## [54]
-------------------------...TTGACCCAGCAGGCGGTGGTGACCC
TG_T2_2
## [55]
-------------------------...TTGACCCAGCAGGCGGTGGTGACCC
TG_T2_3
## [56]
-------------------------...TTGACCCAGCAGGCGGTGGTGACCC
TG_V1_1
## [57]
-------------------------...TTGACCCAGCAGGCGGTGGTGACCC
TG_V1_2
## [58]
-------------------------...TTGACCCAGNAGGNGGGGGTGACCC
TG_T2_6
```

```
## [59]
TAATGCCAAAGATATAGGAACTTTA...------------------------
LP_W_7
##  Con ------------------------...TTGACCC?GCAGG?GG?
GGTGATCC Consensus
```

The default printing function shortens the alignment for the sake of compacting the output. The `print()` function provided by the `msa` package provides some ways for customizing the output, such as, showing the entire alignment split over multiple blocks of sub-sequences:

```
print(my_first_alignment, show="complete")
```

A lot of times it's easier to look at alignments if you colour code the differente bases. We can convert your alignment to a "phyDat" object for use in phangorn, the major package for tree building in R.

```
dna_phy <- msaConvert(my_first_alignment,
"phangorn::phyDat")
image(dna_phy)
```

Each individual is here plotted on a row and the sequences are colour

coded. As you can see are the sequences of different lengths, and they are not all aligned from the beginning of the first base but according to the best “match”.

4. There is one sequence which is quite short, but aligns pretty well, which one is that?
5. There is one sequence which seems to be quite different from the others, which one?

This could be potential contamination, lets investigate it using BLAST.

3.2 Manual BLAST

- Open the fasta file in your text editor
- Locate the odd individual
- Go to BLAST on the NCBI website and click nucleotide BLAST.

BLAST will compare the sequence you are interested into, against a database with many sequences and show you the most similar results (homologous sequences). This is useful when trying to figure out which organisms you have sequenced. We will use the nucleotide-nucleotide BLAST (blastn) program. This program compares DNA sequences against a specific DNA database.

- From your text editor, copy the the odd individuals sequence and paste it in **`blastn`** where it says **`Enter accession number(s), gi(s), or FASTA sequence(s)`**

- Copy and paste the header of the sequence in **`blastn`** where it says **`Job Title`**

- In the **`Database`** option, select **`Standard databases`**

- In **`Program Selection`**, select **`Highly similar sequences (megablast)`**
- Expand **`Algorithm parameters`**
- In **`General Parameters`**, select 10 for **`Max target sequences`**
- Click BLAST below

BLAST will return now a table showing similar sequences to your submission.

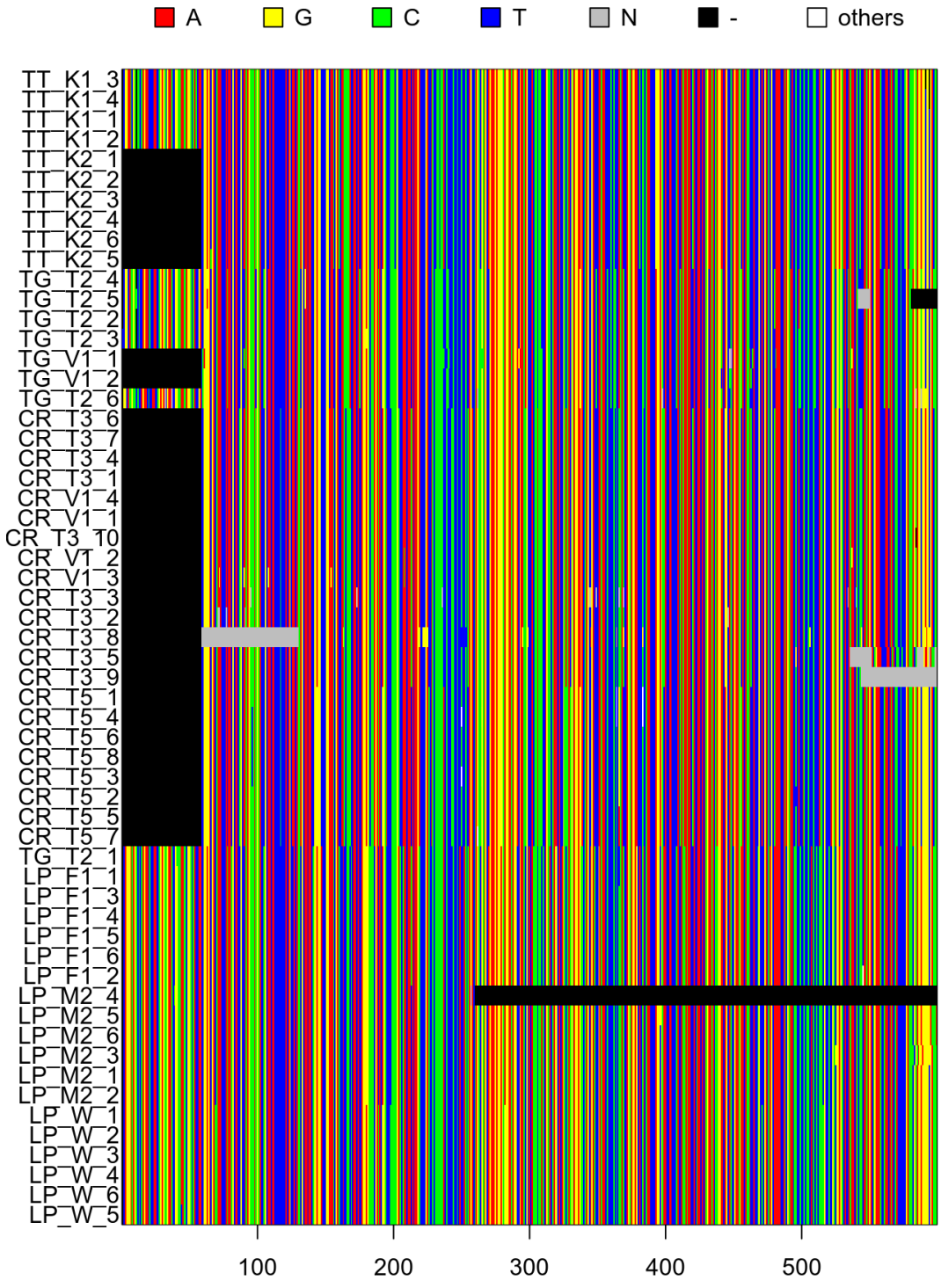
6. What is the top alignment?
7. What is the common name of top alignment?

It looks like we got some contamination. We should remove the bad sequence and do a new alignment.

Simply remove everything from “>” to the end of of the bad sequence and save. If you save it with a new name remember to update the path accordingly `my_sequence_file <- "data/HSCrabs_CO1.fas"`

8. After that you can load in the new file and redo the alignment, how does it look now?

```
## use default substitution matrix
```


A
G
C
T
N
-
others
TT_K1_3
TT_K1_4
TT_K1_1
TT_K1_2
TT_K2_1
TT_K2_2
TT_K2_3
TT_K2_4
TT_K2_6
TT_K2_5
TG_T2_4
TG_T2_5
TG_T2_2
TG_T2_3
TG_V1_1
TG_V1_2
TG_T2_6
CR_T3_6
CR_T3_7
CR_T3_4
CR_T3_1
CR_V1_4
CR_V1_1
CR_T3_10
CR_V1_2
CR_V1_3
CR_T3_3
CR_T3_2
CR_T3_8
CR_T3_5
CR_T3_9
CR_T5_1
CR_T5_4
CR_T5_6
CR_T5_8
CR_T5_3
CR_T5_2
CR_T5_5
CR_T5_7
TG_T2_1
LP_F1_1
LP_F1_3
LP_F1_4
LP_F1_5
LP_F1_6
LP_F1_2
LP_M2_4
LP_M2_5
LP_M2_6
LP_M2_3
LP_M2_1
LP_M2_2
LP_W_1
LP_W_2
LP_W_3
LP_W_4
LP_W_6
LP_W_5
100
200
300
400
500

3.3 Species diversity

But how much much difference is there really? If we look `dna_phy` we can see that there are "58 sequences with 599 character and 210 different site patterns. This means that in the overall alignments there are 210 bases where the sequences differ.

We can investigate if our species have different levels of diversity like this

```
# first we extract the individuals names from or phyDat
object
name <- colnames(as.data.frame(dna_phy))
# Then we extract only the species code, which is the
first two characters
species_code <- substr(name, 1, 2)

# Using this we can subset the phyDat file by species
TT <- subset(dna_phy, which(species_code == "TT"))
CR <- subset(dna_phy, which(species_code == "CR"))
TG <- subset(dna_phy, which(species_code == "TG"))
LP <- subset(dna_phy, which(species_code == "LP"))

# and finally we recalculate the number of
substitutions for each species

TT_phy <- phyDat(TT, type = "DNA", levels = NULL)
CR_phy <- phyDat(CR, type = "DNA", levels = NULL)
TG_phy <- phyDat(TG, type = "DNA", levels = NULL)
LP_phy <- phyDat(LP, type = "DNA", levels = NULL)
```

9. How many different site patterns do the different species have?
10. which species has the most diversity?

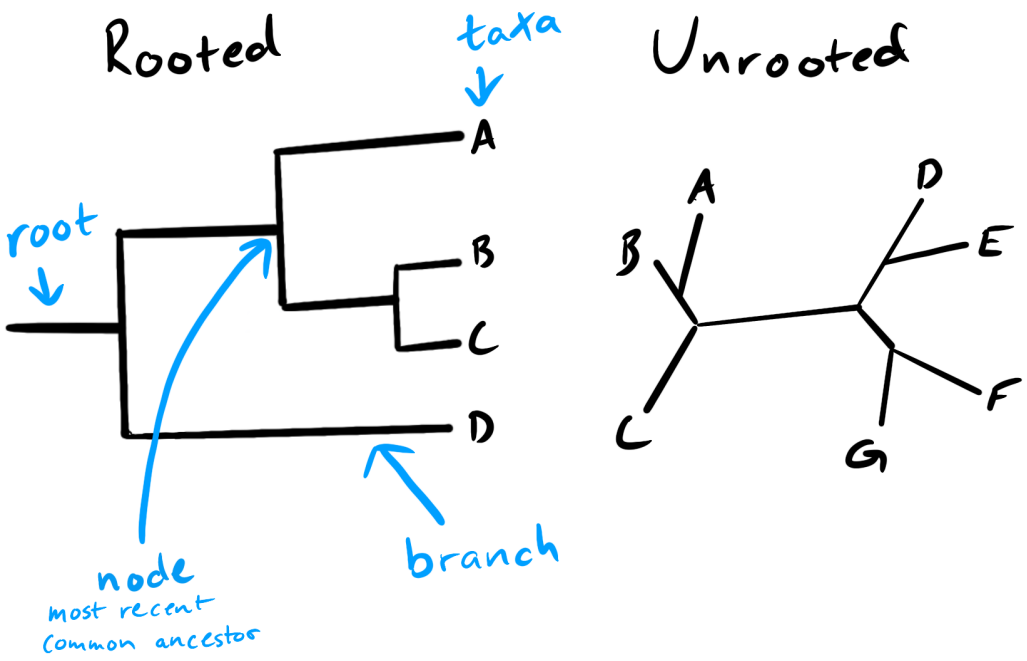
4 Phylogenetic trees

Phylogenetic trees are depictions of the estimated evolutionary

relationships between taxa - these can be organisms, species, strains or even genes. The taxa are on the tips of the trees, and the internal nodes of the tree represent their hypothetical ancestors. Nodes and taxa are connected by branches/edges. Groups of taxa that share a unique common ancestor are considered a clade.

In a rooted phylogenetic tree, each node with descendants represents the inferred most recent common ancestor of those descendants, and the edge lengths in some trees may be interpreted as time estimates.

Unrooted trees illustrate only the relatedness of the leaf nodes and do not require the ancestral root to be known or inferred.



4.1 Compute pairwise distances

Phylogenetic trees can be modelled from sequence data using a variety of approaches. Here, we will begin with a ‘neighbour joining’ algorithm, which sequentially joins taxa based on their sequence

similarity. We have to begin the process by calculating the pairwise ‘distance’ among all of the sequences. Reducing sequence alignments into a matrix of pairwise distances allows for the rapid estimation of phylogenetic trees (at the cost of the additional information those sequences contain).

A distance matrix describes the differences between each pair of sequences, often ranging from 0-1, with 1 being the most different.

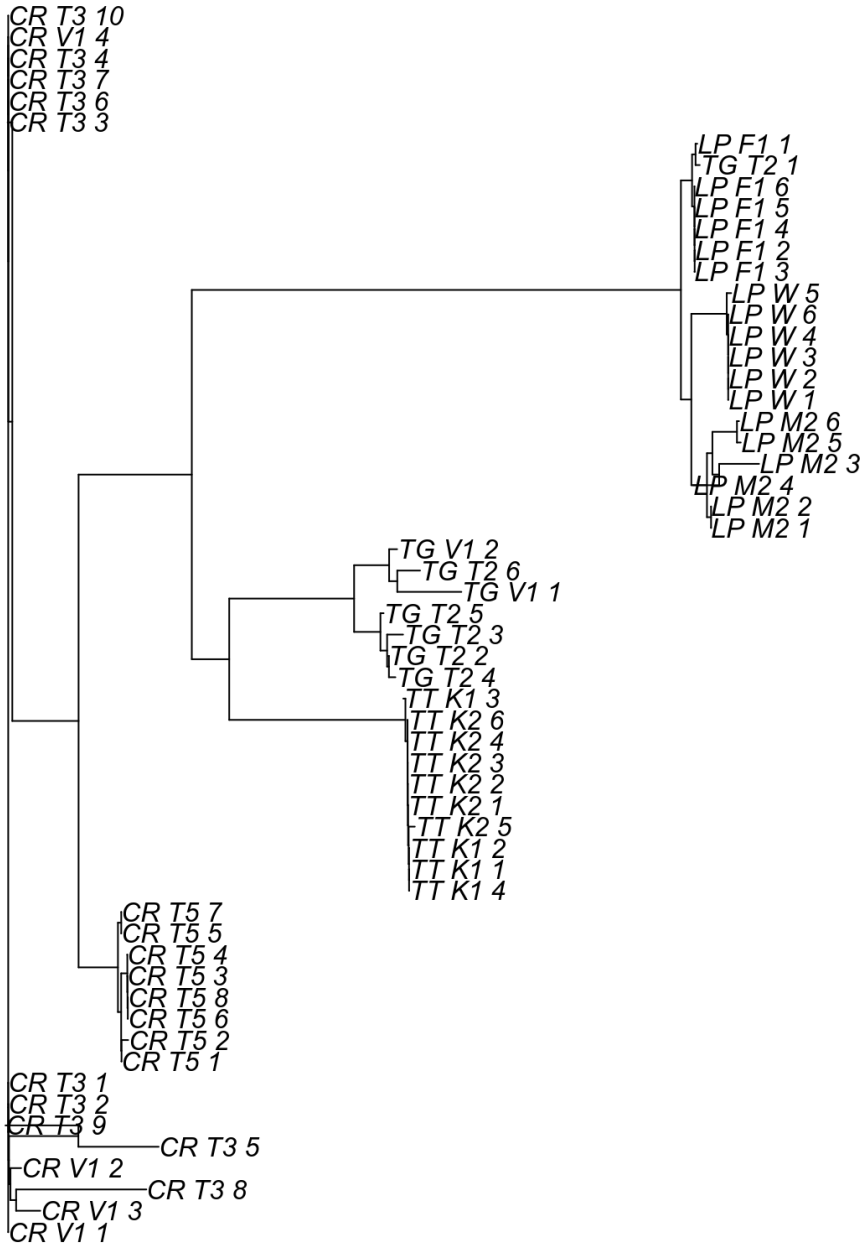
```
dna_dist <- dist.ml(dna_phy)
```

11. which two pairs of sequences have the largest distance, aka difference?

4.2 Building and plotting phylogenetic trees

Once you have a distance matrix we can construct and plot a neighbour joining tree from the distance matrix using the function `NJ()`

```
#make tree
my_NJ  <- NJ(dna_dist)
#plot tree
plot.phylo(my_NJ, main = "My tree")
```



Great we have our first tree!

The tree can still be a little hard to read so why don't we try some different ways of plotting Lets investigate our tree object `my_NJ`

```
my_NJ
##
## Phylogenetic tree with 58 tips and 56 internal
nodes.
##
## Tip labels:
##  TT_K1_3, TT_K1_4, TT_K1_1, TT_K1_2, TT_K2_1,
TT_K2_2, ...
##
## Unrooted; includes branch lengths.
```

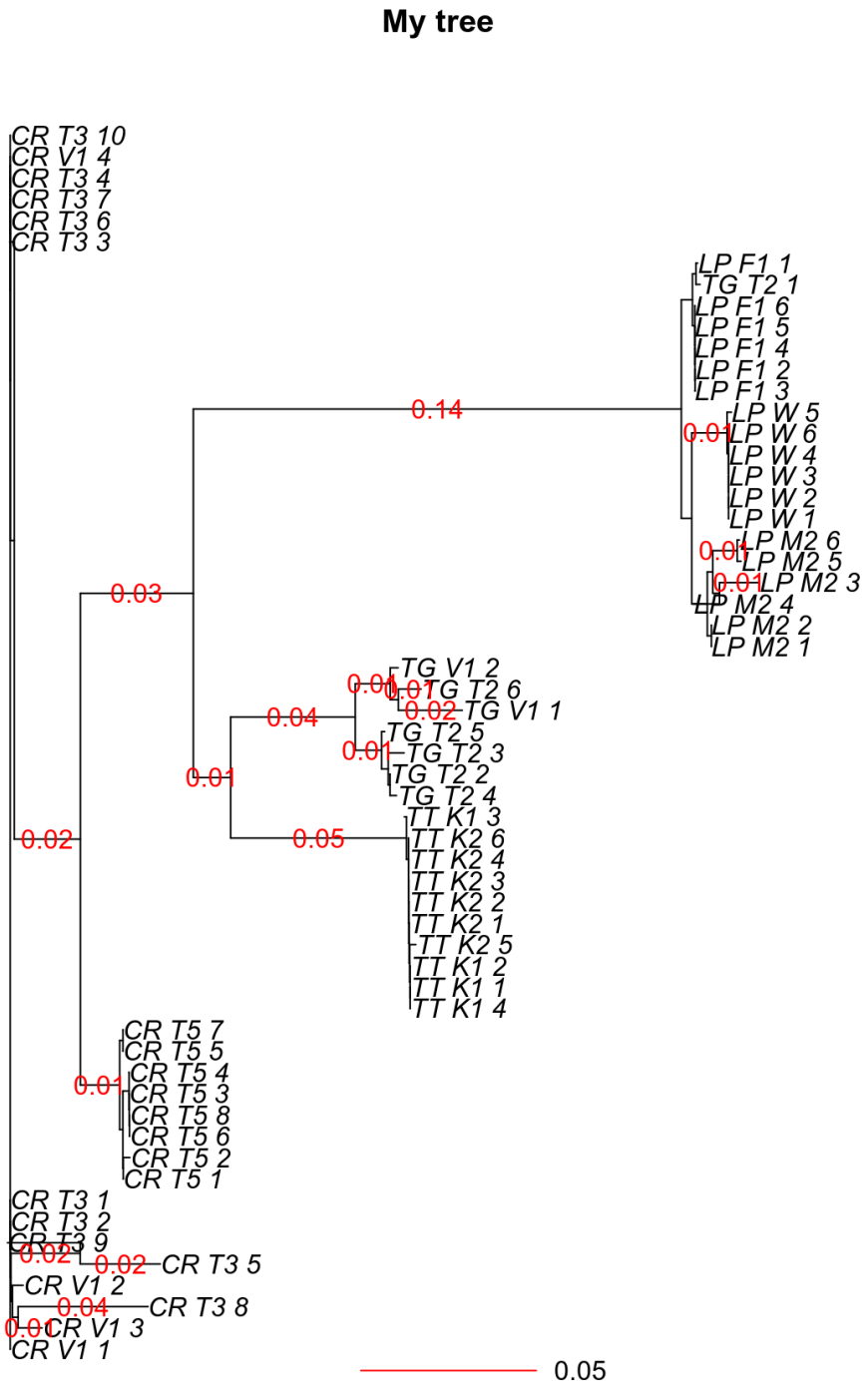
This give us some basic info about our tree, like how many nodes and tips there are, if its rooted or not and lastly if it includes branch lengths (also called edge lengths). A branch length is a number associated with each branch and may represent time or it may represent a measure of expected genetic distance, in our case it is the genetic distance. You can take a look at branch length by typing `my_NJ$edge.length` which will show you a numeric vector of the lengths of the branches. We can also plot these values on the tree, but before we do that let's round the values to only use 2 decimals

```
annot <- round(my_NJ$edge.length,2)
annot
##   [1]  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00
##  [13]  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00
##  [25] -0.01  0.01  0.00  0.01  0.00  0.00  0.00
0.00  0.00  0.01  0.00  0.00
##  [37]  0.02  0.01  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.01  0.01
##  [49]  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00
##  [61]  0.00  0.00  0.00  0.00  0.00  0.00  0.05
```

```
0.04  0.01  0.14  0.00  0.00
##  [73]  0.00  0.00  0.00  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00
##  [85]  0.01  0.03  0.02  0.00  0.00  0.00  0.00
0.00  0.00  0.00  0.00  0.00
##  [97]  0.00  0.00  0.02 -0.02  0.01  0.04  0.00
0.00  0.00  0.02  0.00  0.00
## [109]  0.00  0.00  0.00  0.00  0.00
```

Now we can plot the tree again and add the branch lengths

```
# plot tree
plot.phylo(my_NJ, main = "My tree")
# add all branch length that are abov 0
edgelabels(annot[annot>0], which(annot>0), frame="n",
col = "red")
# add scale of the branch lengths
add.scale.bar(0.1,0, lcol = "red")
```



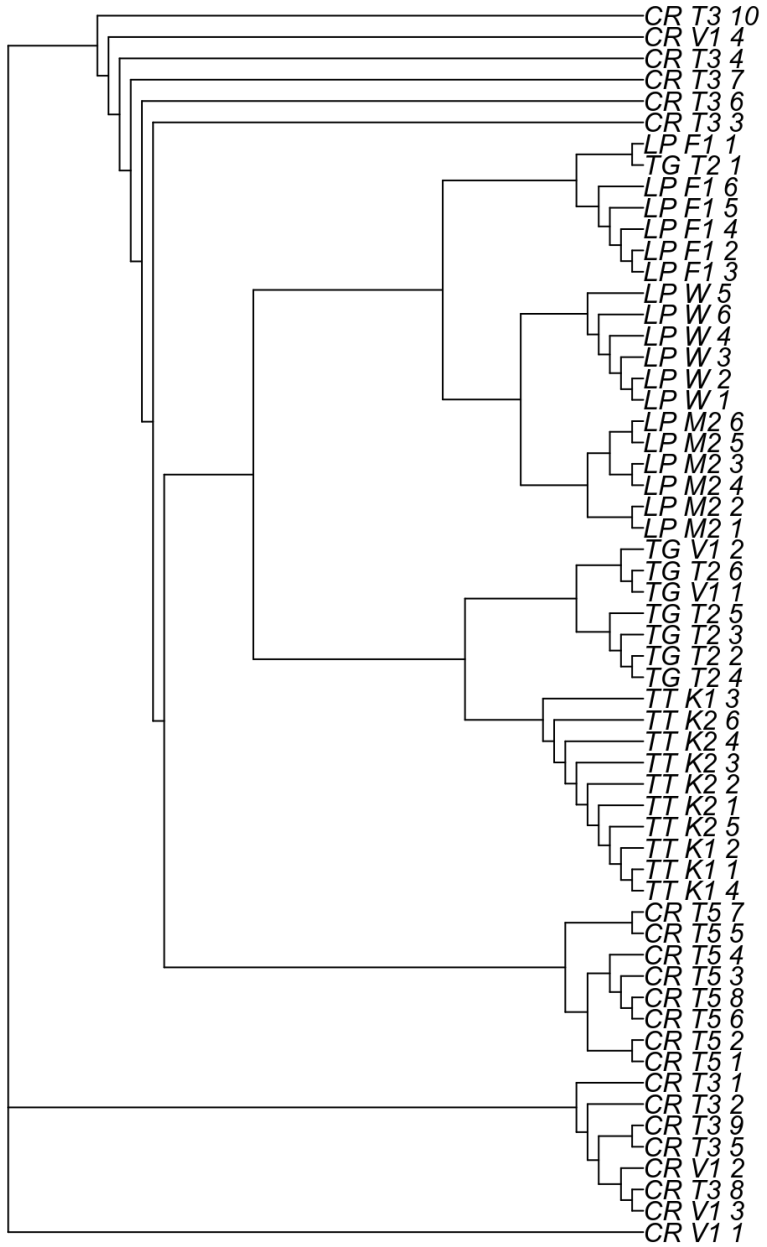
12. Which two individuals have the largest evolutionary

distance between them?

13. How large is this distance?

If we are not interested in the evolutionary distance between the sequences and only want to see the topology (branching order) of the tree, we can choose to look at our tree without using branch lengths like this

```
plot.phylo(my_NJ, main = "My tree without branch
length", use.edge.length = FALSE)
```



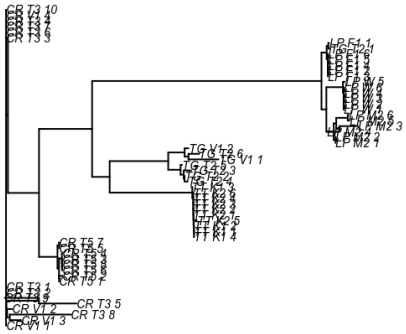
Now this is a bit easier to read as the topology became more clear,

but we do loose a lot of information when we don't include branch length.

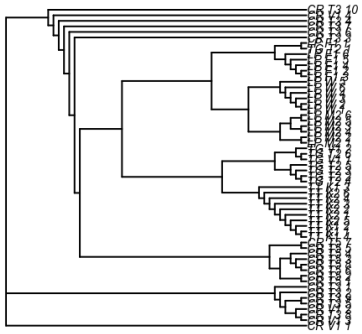
Let's plot a couple of more trees

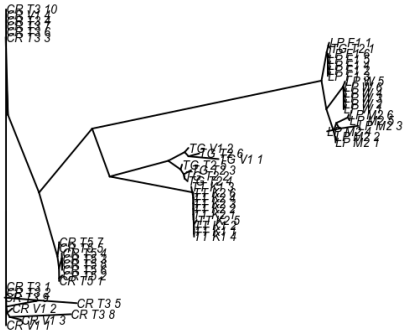
```
# This only tells R that we want to make 6 plots in 3
rows and 2 columns
layout(matrix(1:6, 3, 2))

plot.phylo(my_NJ, main = "With branch lengths")
plot.phylo(my_NJ, type = "c")
plot.phylo(my_NJ, type = "u")
plot.phylo(my_NJ, use.edge.length = FALSE, main =
"Without branch lengths")
plot.phylo(my_NJ, type = "c", use.edge.length = FALSE)
plot.phylo(my_NJ, type = "u", use.edge.length = FALSE)
```

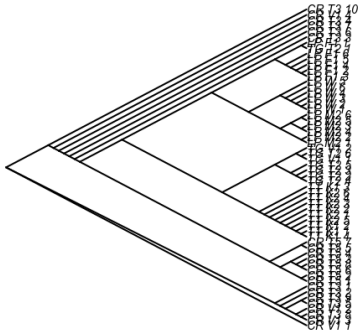
With branch lengths

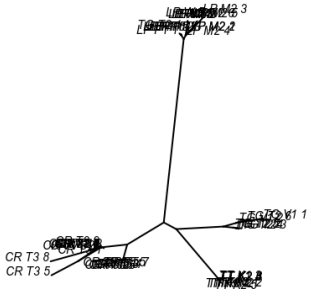


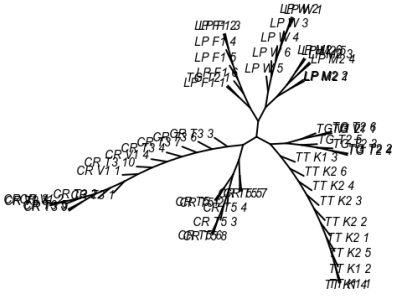
Without branch lengths











```
# this tells R that now we only want to plot on thing
at the time again
```

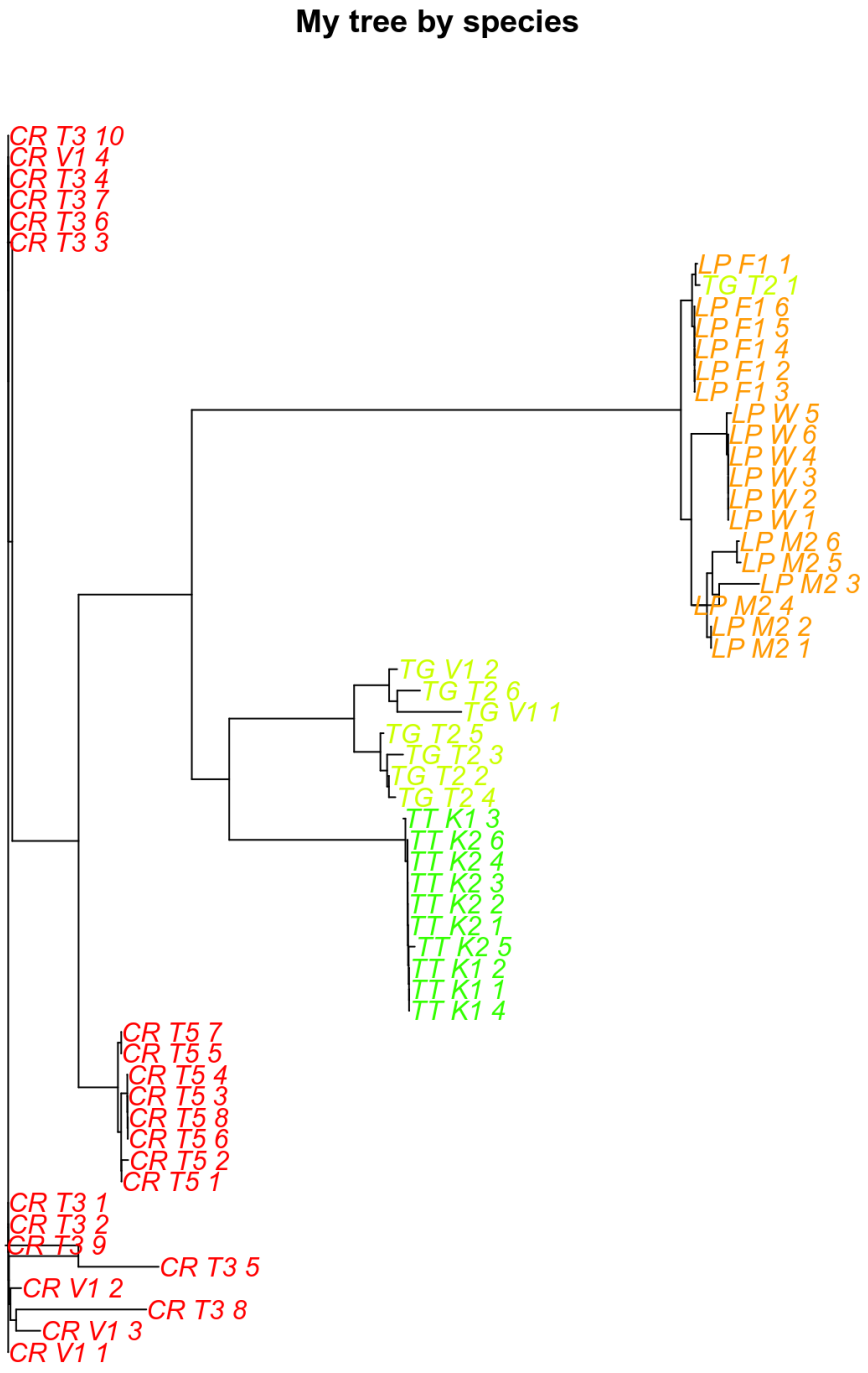
```
layout(1)
```

OK now let's add a bit of colour to make our tree easier to understand If we look inside our tree object we can find the our individual names under `my_NJ$tip.label` and we can use that to extract location labels to use for colouring in the graph like this.

```
#splits names by "_" and makes it into a data frame
ID_df <- t(data.frame(strsplit(my_NJ$tip.label,
split="_")))
# takes the first column and makes it into a vector of
colour
species_col <- rainbow(10)[as.factor(ID_df[,1])]
#takes the second column and makes into a vector of
colours
location_col <- rainbow(10)[as.factor(ID_df[,2])]
```

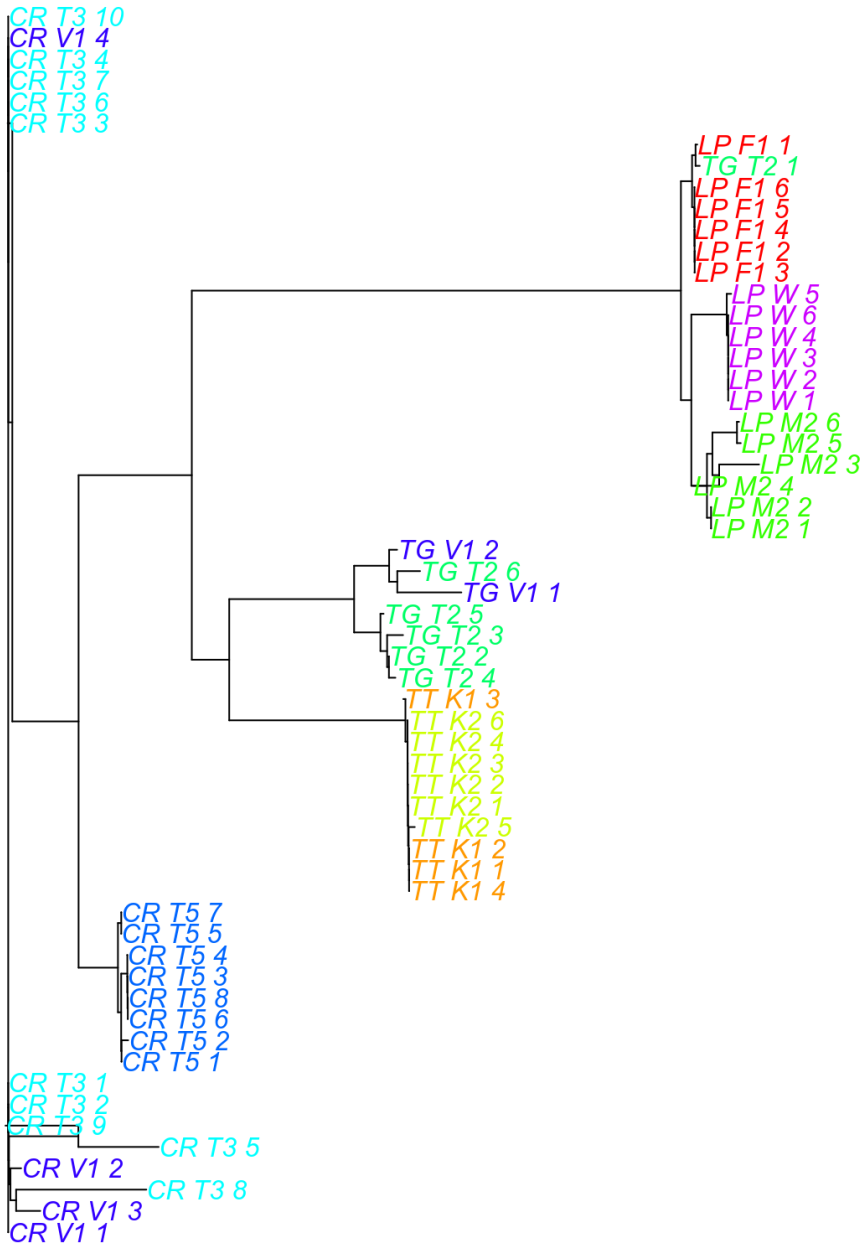
now we can choose to plot the tree according to the species

```
plot.phylo(my_NJ, main = "My tree by species",
           tip.col = species_col)
```



or according to location

```
plot.phylo(my_NJ, main = "My tree by species",
           tip.col = location_col)
```



14. Try some different plotting options and decide on your favourite tree which you think best shows the relationship between the different individuals and save it.

```
# Step 1: Call the png command to start the plot
png(file = "my_best_plot.png",   # The directory you
want to save the file in
    width = 400, # The width of the plot in inches
    height = 800) # The height of the plot in inches

# Step 2: Create the plot with R code
plot.phylo()

# Step 3: Run dev.off() to create the file!
dev.off()
```

4.3 Exporting Trees

Finally, unless you're just analyzing relationships among a very limited number of taxa, the system viewer in R is probably going to be insufficient for viewing and saving your tree. The function write.tree() allows you to export the output of your analyses in Newick format, useful for further tinkering in programs such as FigTree.

```
write.tree(my_NJ, file="my_tree.tre")
```

5 Discussion questions

Discuss the following questions in groups of 4 students during the Friday seminar

Question 15: You calculate the sequence variation within each species. Which species had the highest and lowest diversity. If you look into the ecology of the

four species (explained in the introduction section of Vestbo et al paper), can you find an explanation of the patterns you observed?

Question 16: Plot the geographical positions of all branches of the phylogenetic tree on the map (you can do this manually). Do you see any genetic breaks, i.e. geographical areas that are genetically very distant?

Question 17: What is the average branch length dividing the 2 populations of Carcinoscorpius rotundicauda? If you assume a mutation rate of 18%, when did these 2 populations separate, and which geological event could have caused this separation?