# Visualise trees in R

F. De Boever & D. Green

## Introduction

In this short tutorial we show how to use `ggtree` to quickly visualise and annotate trees in R. Although excelent documentation exists for this package, this tutorial contains some tips of what I find are the easiests ways to combine metadata and a tree.

## Let's keep going

```
# for general data maniuplarion
library(dplyr)

# for visualisation
library(ggplot2)
library(ggtree)

#for phylogenetics
library(ape)
library(phytools)
```

## Loading data files

load in the metadata, in this example we use the metadata file obtained in previous tutorial. This file is in a tab-delimited format, hence we can use `read.delim()` function from base R.

```
#load metadata file
metadata <- read.delim("~/Github/CCAP_course/main/examlpes/CCAP_SSURef_S
head(metadata)
```

Similartly, we load in the associated tree, obtained in earlier tutorials. A fundamental package in R to deal with phylogenies is ape, we can use a function from that package called

`read.tree()` to read trees.

```
#load tree
tree <- ape::read.tree("~/Github/CCAP_course/main/examlpes/nannochlorops
```

In this case we have an unrooted tree, which is the default output of most phylogenetic analsysis. Since we may not know the outgroup, a commonly used alternative is to use mid point routing. This can be achieved using `midpoint.root()` function from the `phytools` package

```
#mid point root
mid.point.rooted.tree <- midpoint.root(tree)
```

Next, we can order the tips in a way that is more presentable, branches are rotated (so topology remains the same), using a function called `ladderize()` from the `ape` package

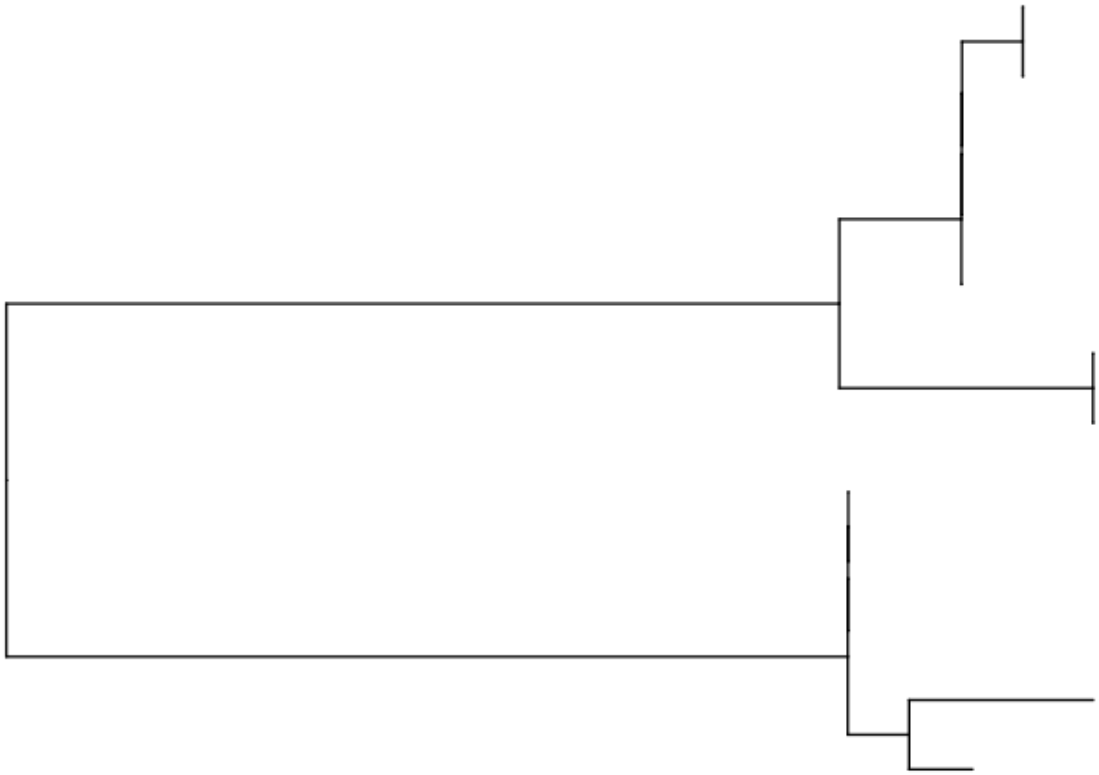
```
#ladderize
mid.point.rooted.tree <- ladderize(mid.point.rooted.tree)
```

# Basic visualisation

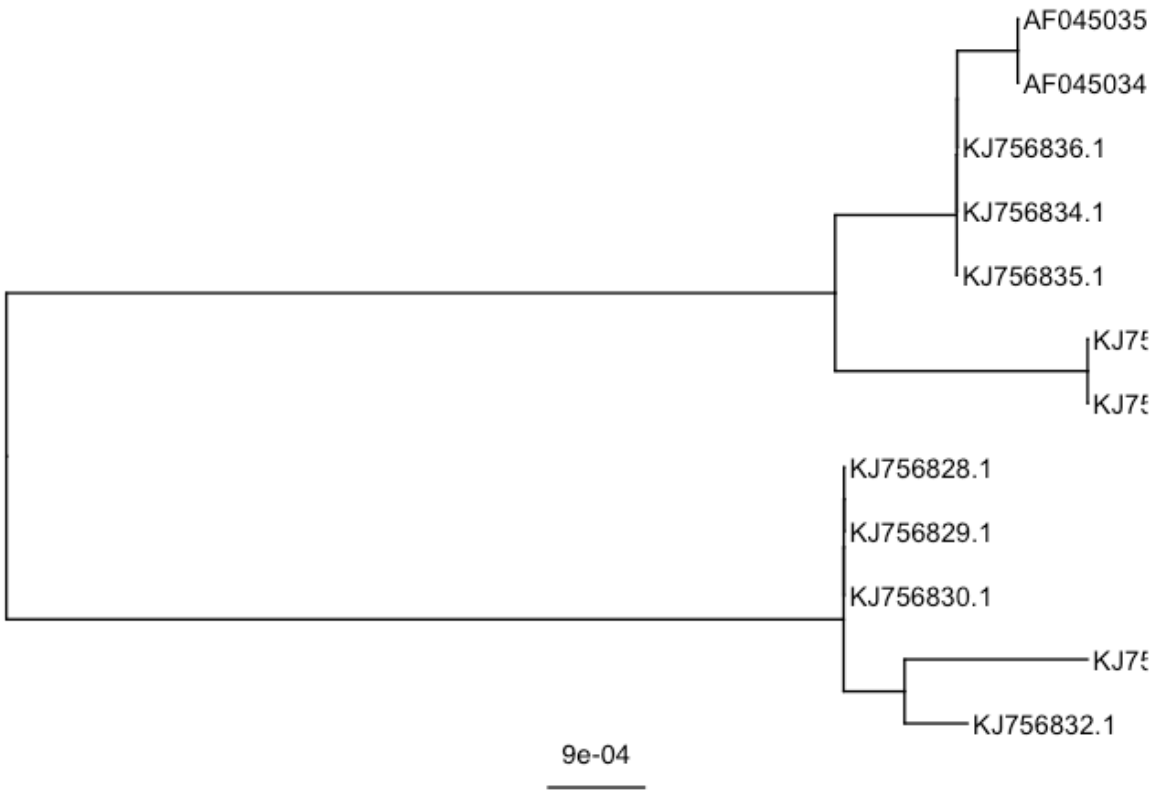
`ape` and `phytools` offer functions to visualise a phylogenetic tree using the generic `plot()` function. Yet a more powerful way to plot trees in R using the `ggplot2` gramar is via the ggtree package.

A simple visualisation is generated by

```
ggtree(mid.point.rooted.tree)
```



```
#add labels and a scale bar
ggtree(mid.point.rooted.tree) +
    geom_tiplab() +
    geom_treescale()
```

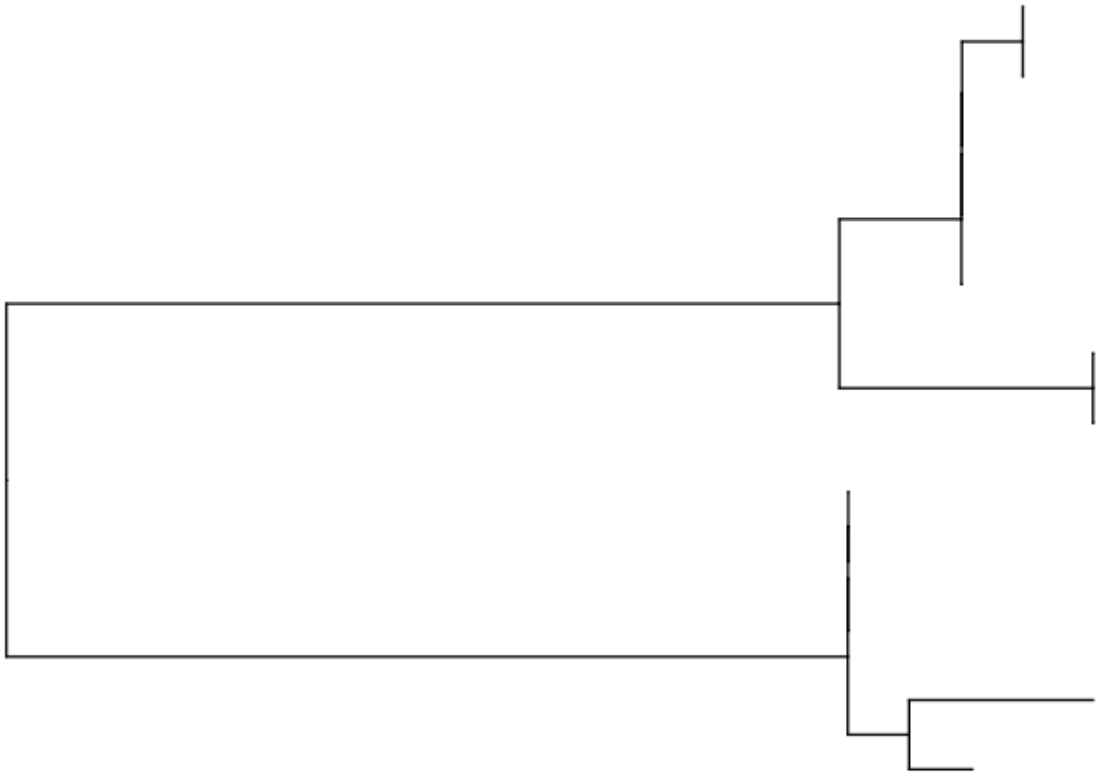



# Link metadata to the tree

In this tutorial we wish to annotate the tree further, this is where ggtree becomes really powerful. Ggtree offers several solutions to link metadata to a tree, among which the `groupOTU()` function. In find that the easiet solution is to first generate a ggtree object and merge it's data with the metadata using for example `dplyr`.

First, store a ggtree object in a variable called p

```
p <- ggtree(mid.point.rooted.tree)
p
```



at this stage you could consider changing the overall layout of the tree by specifying arguments within the `ggtree` function

```
# no branch lengths?
p.1 <- ggtree(mid.point.rooted.tree, branch.length='none')+geom_tiplab()
p.2 <- ggtree(mid.point.rooted.tree, layout='circular')+geom_tiplab()
p.3  <- ggtree(mid.point.rooted.tree, layout='fan', open.angle=180) + ge
```

Consider our first `ggtree` object stored in the variable p we can have a look at what is inside using the `str()` function. `str()` is very useful in general and should become one of your best friends in R

```
str(p)
```

Although the natrue of a ggtree object seems complex, the data that is visualised is stored under `p$data` You can have a look at the first rows by using the `head()` function

```
head(p$data)
```

```
## # A tibble: 6 x 9
##   parent  node branch.length label      isTip       x     y  branch a
##    <int> <int>         <dbl> <chr>      <lgl>   <dbl> <dbl>   <dbl> <
## 1     16     1    0.00000209 KJ756830.1 TRUE  0.00766     3 0.00766
## 2     17     2    0.00000219 KJ756829.1 TRUE  0.00766     4 0.00766
## 3     17     3    0.00000219 KJ756828.1 TRUE  0.00766     5 0.00766
## 4     15     4    0.000577   KJ756832.1 TRUE  0.00879     1 0.00850
## 5     15     5    0.00168    KJ756831.1 TRUE  0.00989     2 0.00905
## 6     21     6    0.00000219 KJ756836.1 TRUE  0.00870    10 0.00870
```

Now that we know where the data is stored within a ggtree object, we can start merging it with the metadata. Note, one can use any metadata, as long as you have a column in there that is identical to the tip labels in the tree. In case of the provided metadata file, tip labels are stored in a column names accession_version. We can use `left_join()` from `dplyr` to merge the metedata with the data stored in the ggtree object like so:

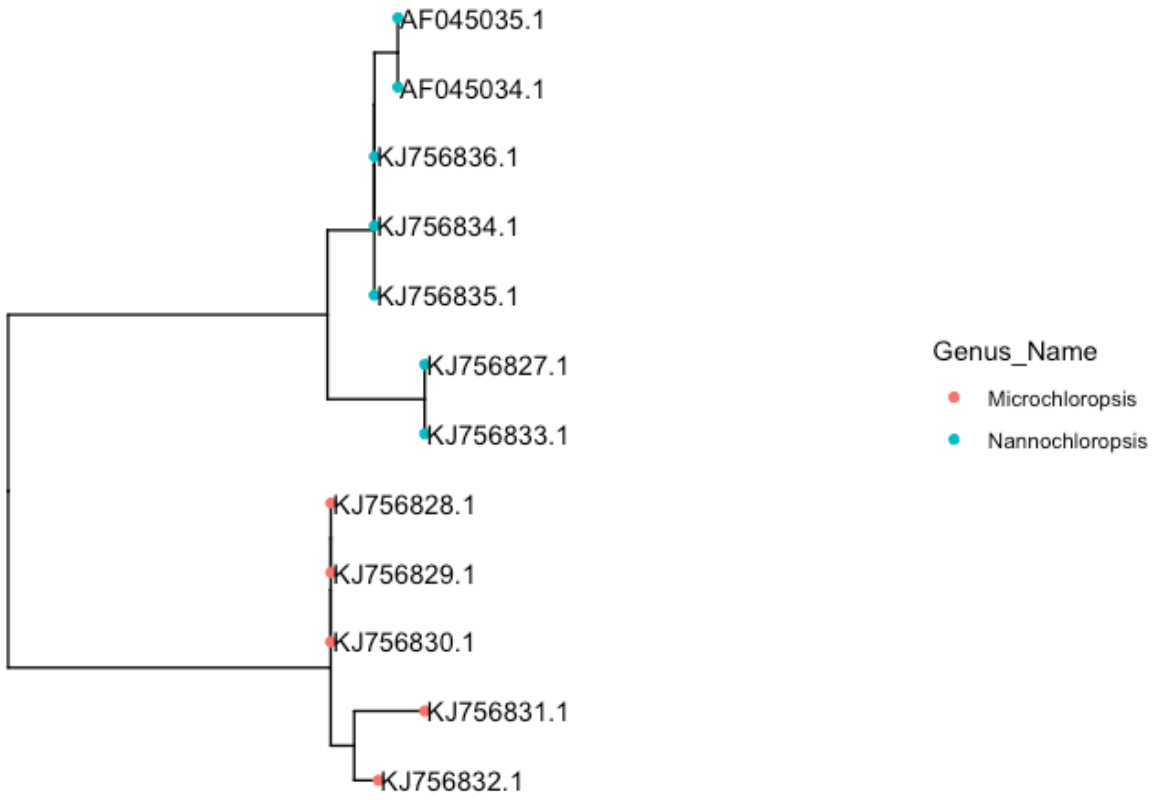
```
p$data <- p$data %>% dplyr::left_join(metadata, by=c('label'='accession_

# it is always good to have a look if things work
head(p$data)
```

```
## # A tibble: 6 x 38
##   parent  node branch.length label  isTip       x     y  branch angle
##    <int> <int>         <dbl> <chr>  <lgl>   <dbl> <dbl>   <dbl> <dbl>
## 1     16     1    0.00000209 KJ756… TRUE  0.00766     3 0.00766    90
## 2     17     2    0.00000219 KJ756… TRUE  0.00766     4 0.00766   120
## 3     17     3    0.00000219 KJ756… TRUE  0.00766     5 0.00766   150
## 4     15     4    0.000577   KJ756… TRUE  0.00879     1 0.00850    30
## 5     15     5    0.00168    KJ756… TRUE  0.00989     2 0.00905    60
## 6     21     6    0.00000219 KJ756… TRUE  0.00870    10 0.00870   300
## # … with 28 more variables: length <int>, moltype <fct>, create_date
## #   definition <fct>, source <fct>, organism <fct>, CCAP_RefSeq <fct>
## #   SeqSource <fct>, strainName <fct>, qSpecies_No_Letter <fct>,
## #   Genus_No <int>, Species_No <int>, Species_No_Letter <lgl>,
## #   Genus_Name <fct>, Species.SVF_name <fct>, Isolator <fct>,
## #   Isolation_Year <fct>, Orig_Desig <fct>, Equiv_Strains <fct>,
## #   Other_Desigs <fct>, Type_Culture <fct>, Pathogen <lgl>, Environme
## #   Geog_Location <fct>, Curator <fct>, Taxonomy <fct>, Archived <fct
## #   strainID <fct>
```

Now that we merged the data, we can access those columns straight from the ggtree object. for example we can color the tip points using the function `geom_tippoint` and assign

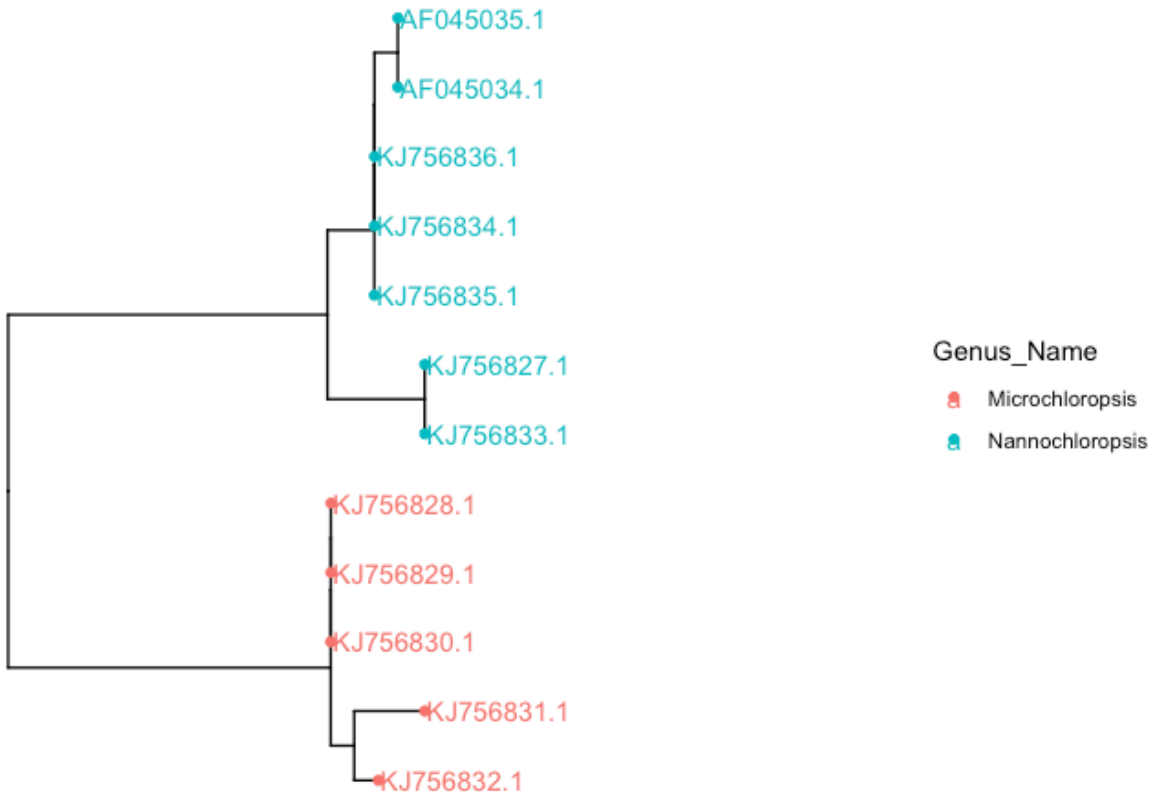
`color=Genus_Name` to color the tips according to genus.

```
# color points
p + geom_tippoint(aes(color=Genus_Name)) +
    xlim(c(0,0.02)) +
    geom_tiplab()
```




we could also color the tip labels if you'd like

```
# color tip labels
p + geom_tippoint(aes(color=Genus_Name)) +
    xlim(c(0,0.02)) +
    geom_tiplab(aes(color=Genus_Name))
```




# Changing the tip labels

Tip labels in our tree correspond to the accession numbers of the sequences that were used. But, what if we want to change the tip labels to something more sensible? A common pattern in literature is that people combine both the accession number and the organisms name. Let's do the same here.

Before messing things up, lets first store out treeobject with another name

```
p1 <- p
```

in the data of ggtree object called p1 (p1$data), we see a column called isTip denoting whether the data corresponds to a tip or an internal node in the tree. We can use this column to subset the data. For example the below line will return only the tips, and ignore internal nodes in the tree.

```
p1$data[p1$data$isTip==TRUE,]
```

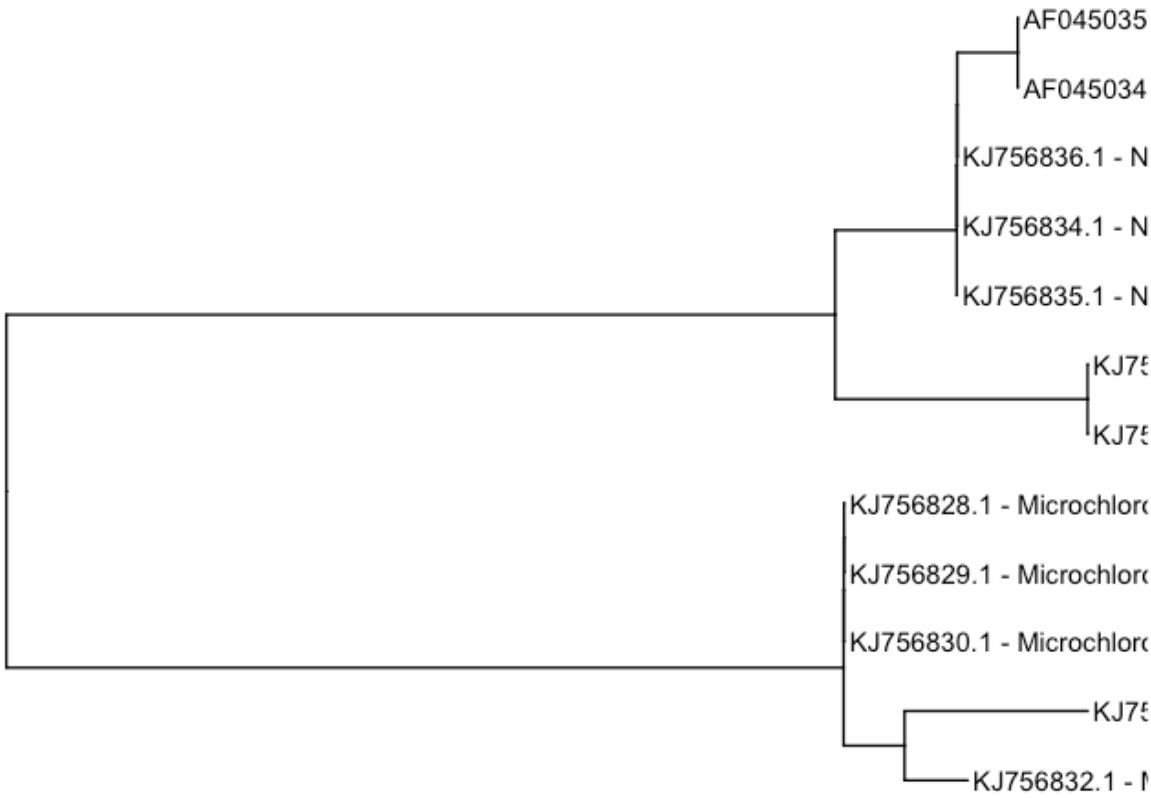
```
## # A tibble: 12 x 38
##    parent  node branch.length label  isTip       x     y  branch angl
##     <int> <int>         <dbl> <chr>  <lgl>   <dbl> <dbl>   <dbl> <dbl
##  1     16     1    0.00000209 KJ756… TRUE  0.00766     3 0.00766    9
##  2     17     2    0.00000219 KJ756… TRUE  0.00766     4 0.00766   12
##  3     17     3    0.00000219 KJ756… TRUE  0.00766     5 0.00766   15
##  4     15     4    0.000577   KJ756… TRUE  0.00879     1 0.00850    3
##  5     15     5    0.00168    KJ756… TRUE  0.00989     2 0.00905    6
##  6     21     6    0.00000219 KJ756… TRUE  0.00870    10 0.00870   30
##  7     19     7    0.00000208 KJ756… TRUE  0.00869     8 0.00869   24
##  8     23     8    0.00000207 KJ756… TRUE  0.00989     6 0.00989   18
##  9     23     9    0.00000208 KJ756… TRUE  0.00989     7 0.00989   21
## 10     20    10    0.00000219 KJ756… TRUE  0.00869     9 0.00869   27
## 11     22    11    0.00000219 AF045… TRUE  0.00925    11 0.00925   33
## 12     22    12    0.00000219 AF045… TRUE  0.00925    12 0.00925   36
## # … with 28 more variables: length <int>, moltype <fct>, create_date
## #   definition <fct>, source <fct>, organism <fct>, CCAP_RefSeq <fct>
## #   SeqSource <fct>, strainName <fct>, qSpecies_No_Letter <fct>,
## #   Genus_No <int>, Species_No <int>, Species_No_Letter <lgl>,
## #   Genus_Name <fct>, Species.SVF_name <fct>, Isolator <fct>,
## #   Isolation_Year <fct>, Orig_Desig <fct>, Equiv_Strains <fct>,
## #   Other_Desigs <fct>, Type_Culture <fct>, Pathogen <lgl>, Environme
## #   Geog_Location <fct>, Curator <fct>, Taxonomy <fct>, Archived <fct
## #   strainID <fct>
```

Using this rationale we can change the labels of the tips, by subsetting using `isTip=TRUE`, and changing the label collumn to a combined version of both label and strain name. we can use paste0 which will in this case concatenate two columns, `label` and `strainName`

```
p1$data[p1 $data$isTip==TRUE,]$label <- paste0(p1$data[p$data$isTip==TRU
```

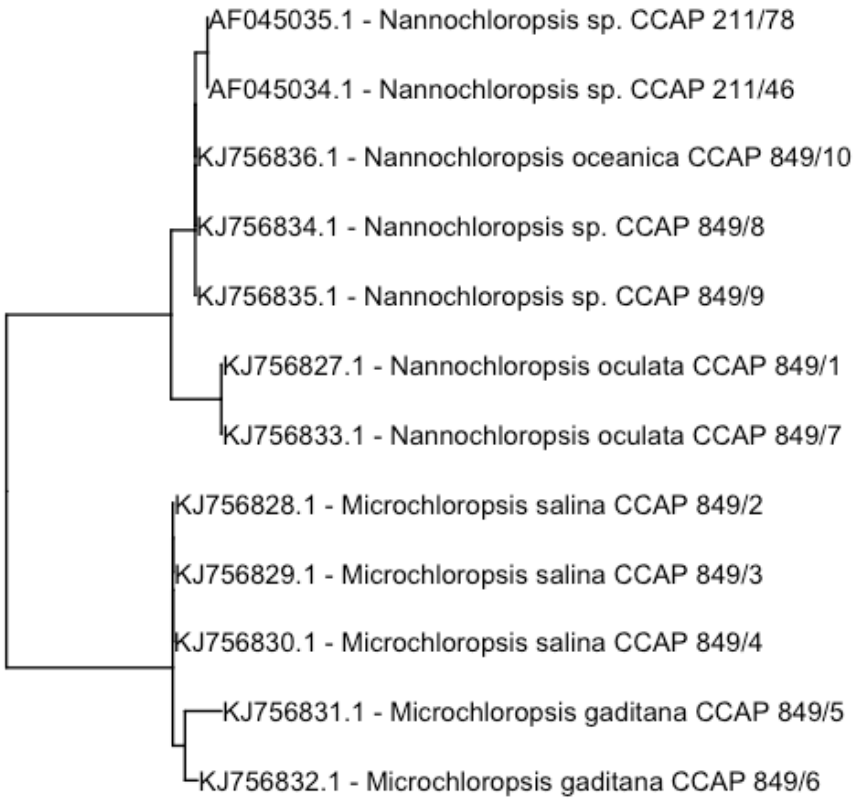
Let's have a look if this works.

```
p1 + geom_tiplab()
```



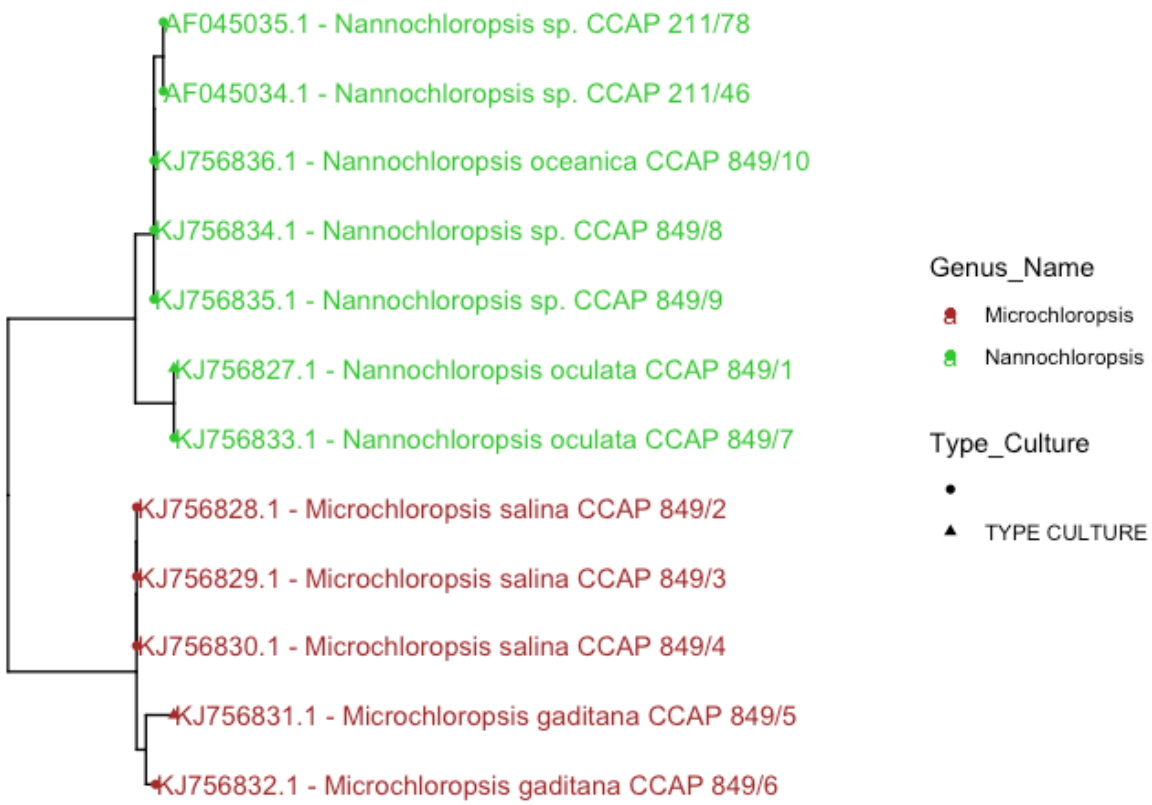

Oh sh*t, we can not read the lables? A solution is to play with the limits of the x-axis using xlim(). Depending on your three and the maximum branchlengths within, you may have to increase or decrease the maximum value. In this case, since the sequences are not too dissimilar we set the xlim to range from 0 to 0.05

```
p1 + geom_tiplab()+ xlim(c(0,0.05))
```




We can further beautify it by assigning colors. One of the nice things about ggtree is that it can be used together with ggplot2 functions such as scale*color*manual() to manually change the colors.

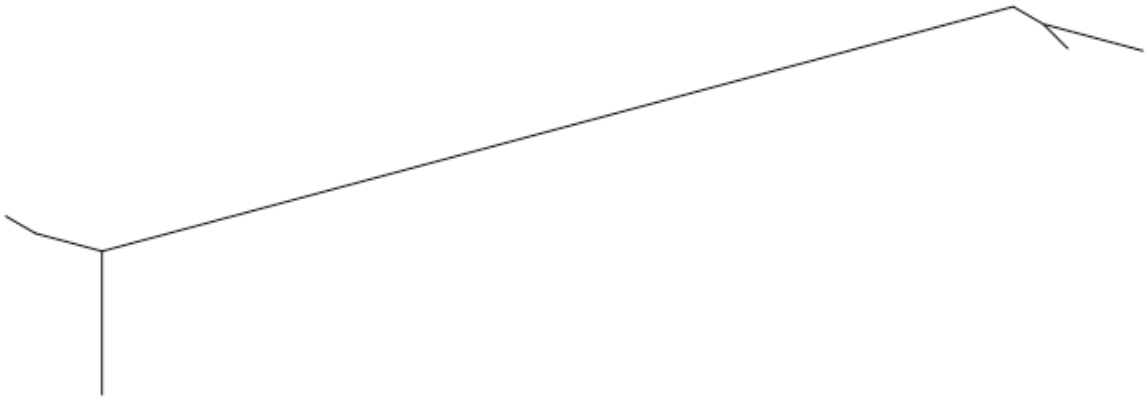
```
p1 + geom_tippoint(aes(color=Genus_Name, shape= Type_Culture)) +
    xlim(c(0,0.05)) +
    geom_tiplab(aes(color=Genus_Name))+
    scale_color_manual(values=c('brown','limegreen'))
```




# What about unrooted trees?

Although ggtree offers a so-called unrooted tree layout, it is not that great! we can rely on oldschool R using the ape/phytools plot function, which does a much better job

```
plot(tree, type = "unrooted",no.margin = TRUE,label.offset = 0.5,pch=21,
```



## Putting data next to it?

Let me create some random data, just as an example, if you have real data, merge it with the ggtree object using `dplyr` or alike, as shown before. In my case, I will make som random data, and assign it to a new column inside the ggtree object called p1

```
n_tips <- length(tree$tip.label)

dummy.data <- data.frame(
  cbind('ID' = tree$tip.label,
      'data1'=sample(1:100,n_tips)
      ),stringsAsFactors=FALSE
  )
str(dummy.data)
```

```
## 'data.frame':    12 obs. of  2 variables:
##  $ ID   : chr  "KJ756836.1" "KJ756835.1" "KJ756833.1" "KJ756827.1" ..
##  $ data1: chr  "15" "63" "82" "78" ...
```

```
dummy.data$data1 <- as.numeric(dummy.data$data1)
```

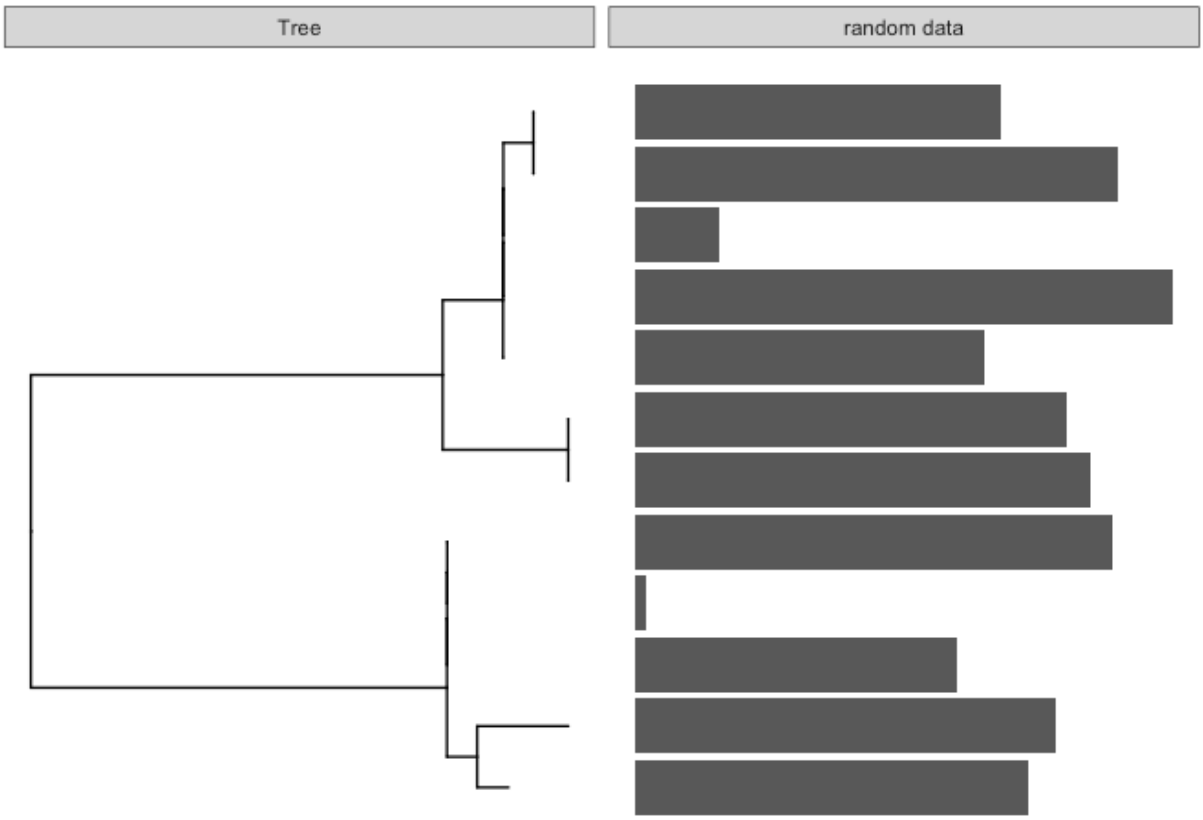
the function facet_plot is made to put different so-called geoms (a `ggplot2` thing) next to the tree. It requires the ggtree objet in our case called p, a name for the panel, the data to visualise and the mapping aesthetics. For example if we want to visualise a simple bar char next to the tree we could run somthing like:

```
library(ggstance)
```

```
##
## Attaching package: 'ggstance'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     geom_errorbarh, GeomErrorbarh
```

```
facet_plot(p, panel = 'random data', data = dummy.data,
               geom = geom_barh,
               mapping = aes(x = data1),
               stat='identity' )
```



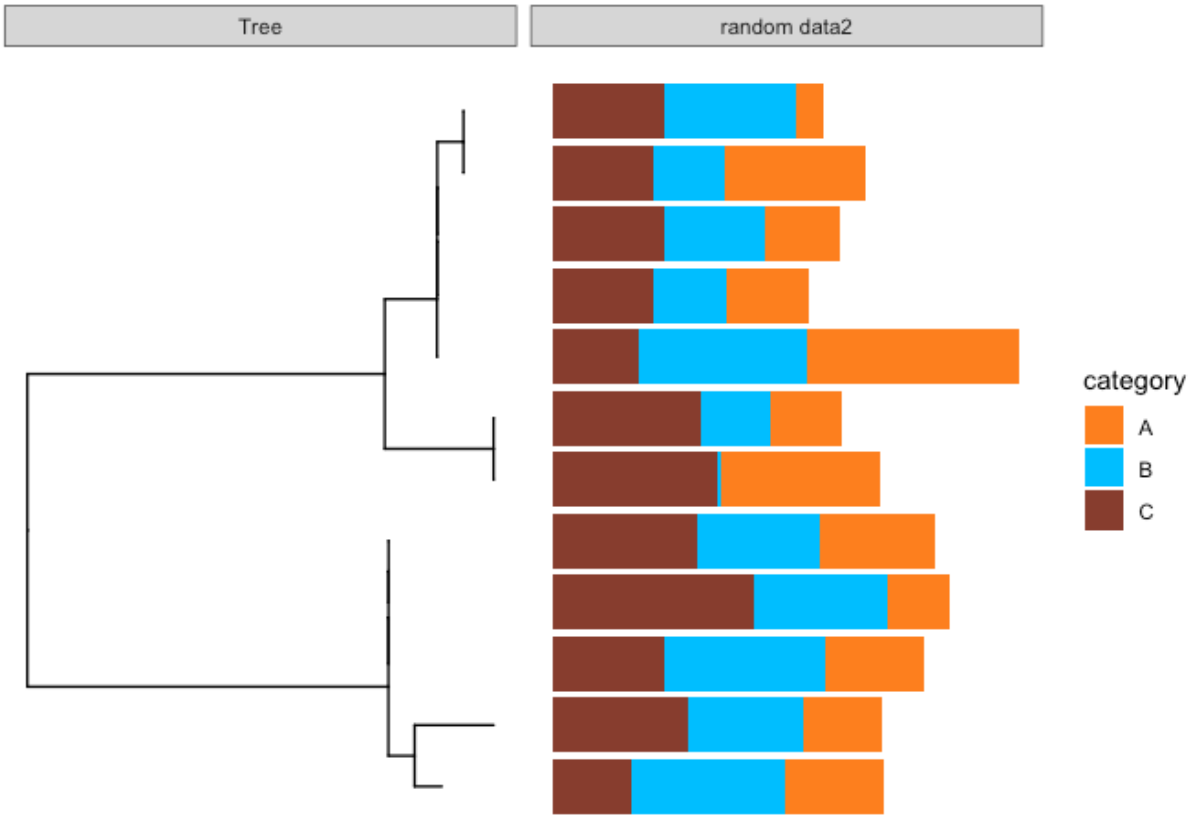

To take it a bit further, we can also visualise stacked bar charts. First, again I create some random data, with an additional column called category, this collumn will be used to color the

bars.

```
dummy.data.2 <- data.frame(id = rep(tree$tip.label, each=3),
                    value = abs(rnorm(n_tips*3, mean=100, sd=50)),
                    category = rep(LETTERS[1:3], n_tips))
str(dummy.data.2)
```

```
## 'data.frame':    36 obs. of  3 variables:
##  $ id      : Factor w/ 12 levels "AF045034.1","AF045035.1",..: 12 12
##  $ value   : num  77.3 103.6 114 219.6 173.1 ...
##  $ category: Factor w/ 3 levels "A","B","C": 1 2 3 1 2 3 1 2 3 1 ...
```

```
facet_plot(p, panel = 'random data2', data = dummy.data.2,
                geom = geom_barh,
                mapping = aes(x = value, fill=category),
                stat='identity') +
                scale_fill_manual(values=c('chocolate1','deepskyblue','c
```



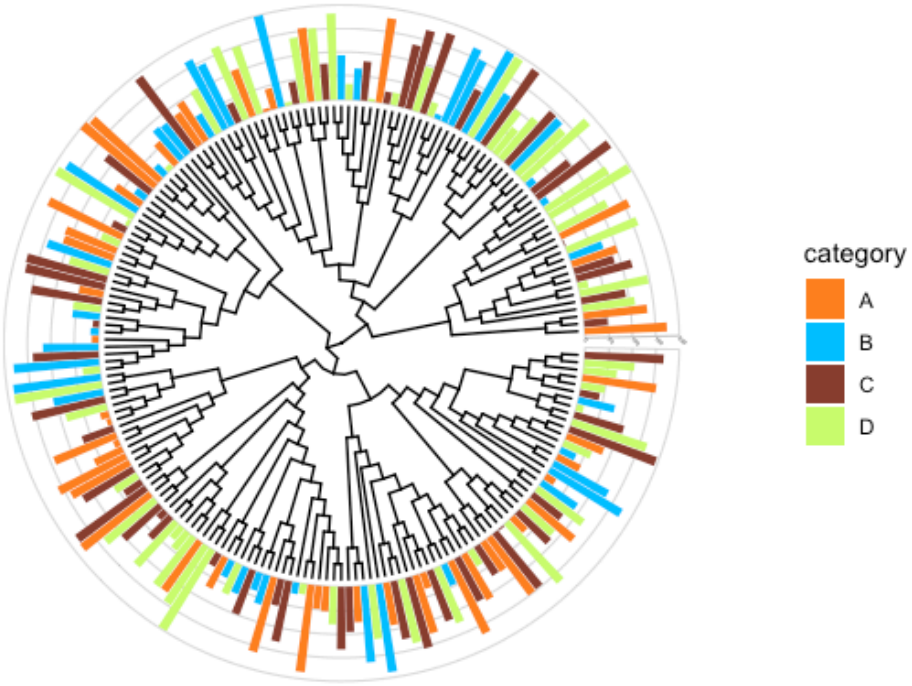

Many other alternatives exist, and I strongly recommend to look into the ggtree handbook, it is a great resource.

# introduciton to ggtreeExtra

A very cool recent addition to the ggtree series is a package called `ggtreeExtra`. This is outside the scope of this tutorial, but here you will have learned important basics that can be taken to the next level with ggtreeExtra!

For example sake, below is code to generate a random tree with 200 tips, and some code to show how easy it is to annotate the tree. for more informaiton on ggtreeExtra please refer to the online ggtreeExtra documentation.

```r
#Generate a random tree, midpoint root it and ladderize
tree.random <- rtree(200)
tree.random.mid <- midpoint.root(tree.random)
tree.random.mid <- ladderize(tree.random)

#assign the tree to a ggtree object
p.2 <- ggtree(tree.random.mid, layout='circular', branch.length='none')

#Creating dummy data
n_tips<- length(tree.random$tip.label)
dummy.data.3 <- data.frame(
  cbind('ID' = tree.random$tip.label,
      'data1'=sample(1:200,n_tips),
      'category' = sample(LETTERS[1:4],n_tips,replace=TRUE)
      ),stringsAsFactors=FALSE
  )
dummy.data.3$data1 <- as.numeric(dummy.data.3$data1)


#visualise and annotate with ggtreeextra
p.2 +
      ggtreeExtra::geom_fruit(
          data=dummy.data.3,
          geom=geom_bar,
          mapping=aes(y=ID, x=data1, fill=category),
          pwidth=0.4,
          stat="identity",
          orientation="y", # the orientation of axis.
          axis.params=list(
                          axis="x", # add axis text of the layer.
                          text.angle=-45, # the text size of axis.
                          hjust=0  # adust the horizontal position of te
                      ),
          grid.params=list() # add the grid line of the external bar plo
      ) +
      scale_fill_manual(values=c('chocolate1','deepskyblue','coral4','da
```


category
A
B
C
D