MinSNP v1.1.0 User Manual

Author: Ludwig Hoon

**Comments added by Phil Giffard**

# Obtaining R programming environment

1. Go to <https://cran.r-project.org/mirrors.html>, select a suitable mirror and download the distributed binary depending on your operating system[[1]](#footnote-1)[[2]](#footnote-2).
2. Install the program.

**Comment:** I am using RStudio 1.0.153, downloaded on to my desktop from the CDU software centre. RStudio provides a semi-GUI interface for R that streamlines work greatly. However, it means that some tasks are done a little differently to how Ludwig states in the manual.

**Comment:** I run the program from my desktop hard-drive. I have it loaded in “C:/Users/pgiffard/documents”. (Note the use of forward slashes – this is in accordance with R syntax – Windows uses back slashes.) However, you can load it anywhere you like.

# Installing the package

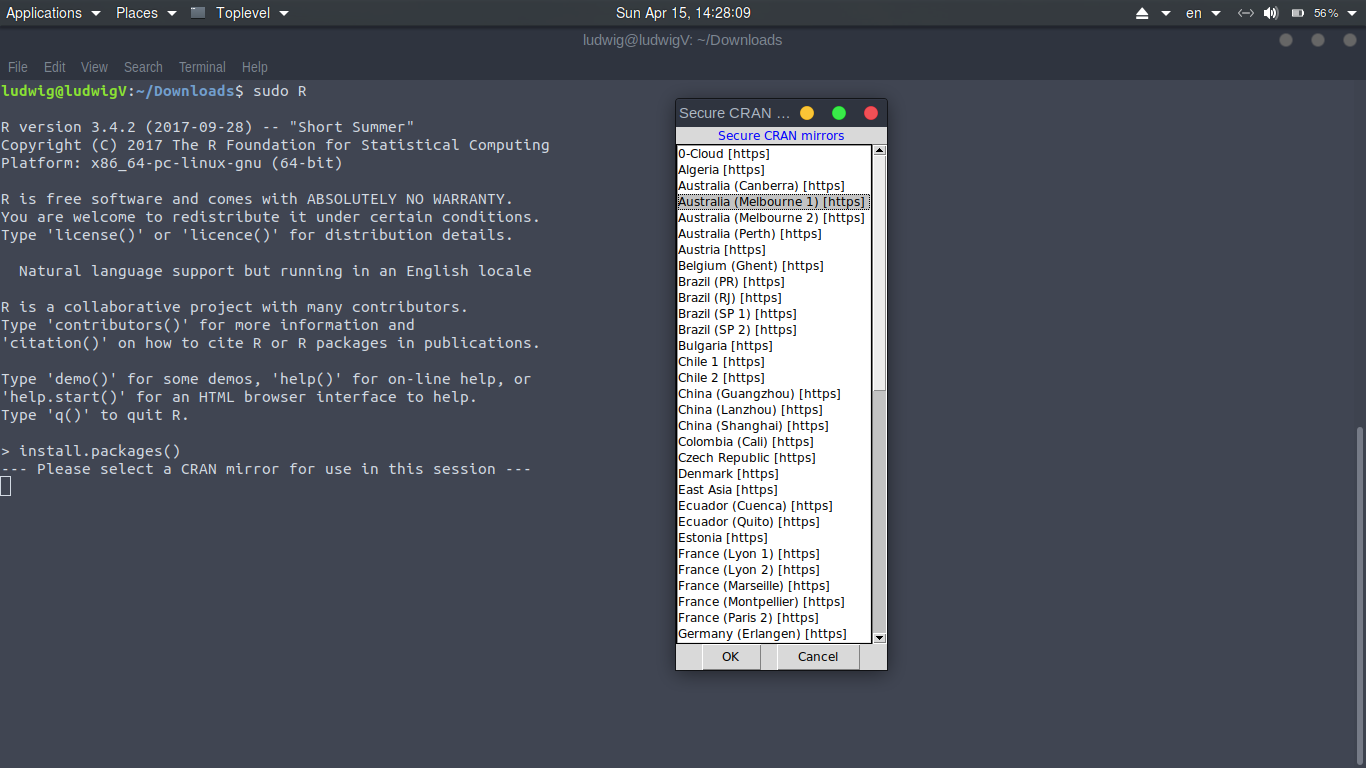
Step1: Getting the dependencies

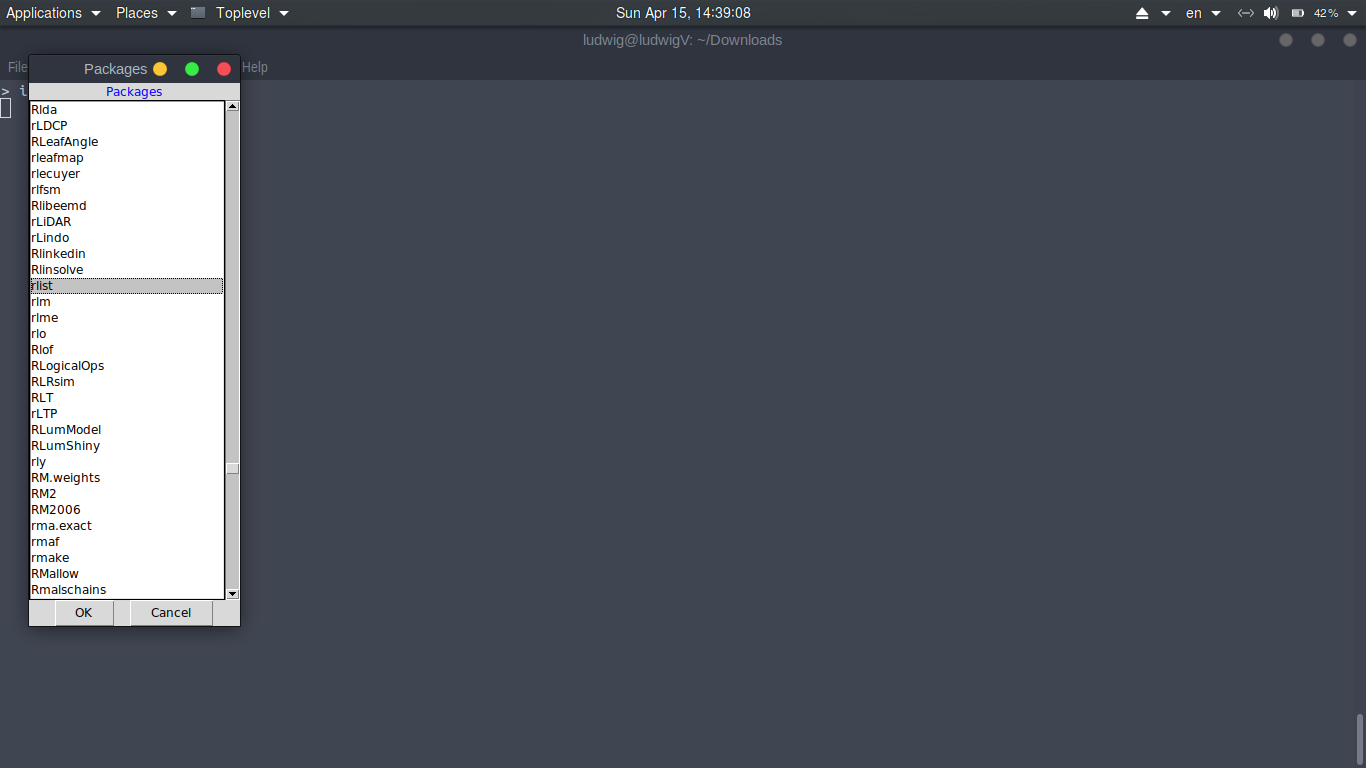
Start R[[3]](#footnote-3), use the command ***install.packages()***. When prompted, select any desire CRAN mirrors (see Illustration 1). You will be prompted by another similar windows about what packages to install (see Illustration 2). Select ***rlist*** and ***seqinr*** from the list. [[4]](#footnote-4)

**Comment:** What I understand is that *rlist* and *seqinr* are pre-existing R applications that are necessary for Minimum SNPs to run, so they have to be installed – which means made available to run. I have had trouble using the *install.packages* command in RStudio. The RStudio way of doing it seems to be to use the GUI functions in the bottom right hand section of the RStudio screen, where packages are listed. **Here, the “install” button is your friend.** If you press that, you will get a choice of searching for packages in the CRAN repository, or searching for them on your own computer/network. For rlist and seqinr, your only choice is the CRAN repository. For the actual software, which has the package name minSNP, you can install from either your own computer/computer network or from the CRAN repository. I have noticed, that there is a tickable box for “install dependencies”. In theory this should install *rlist* and *seqinr* when you install minSNP, but I am not sure if that works, especially when your install minSNP from your own computer/network – so you may have to install them separately.

The bottom line is that you need to have *minSNP,* *rlist* and *seqinr* included in the list of packages in the bottom right screen. Once they are, I think that ticking the box activates them – it is the same as using the “library” command.

**Comment:** the screen shots below are not from RStudio.

  
Illustration 1: slecting mirror for installation

  
Illustration 2: Installing package

Step 2: Obtaining minSNP packages, download minSNP\_1.0.0.tar.gz from <https://github.com/ludwigHoon/PRT452-BINFO-> .

**Comment:** the current version is actually minSNP\_1.1.0.tar.gz

Step 3: In R, use ***setwd()*** command to change the directory to where minSNP\_1.0.0.tar.gz has been downloaded, e.g. ***setwd(“C:/Users/Admin/Downloads”)***.

**Comment:** “Step 3” above possibly includes a minor error of fact by Ludwig, but it does not really matter in practice. I am informed that using the “install\_packages” command or GUI equivalent results in the package being loaded into the directory where the R software itself is located (I think with other program files), whereas the working directory is only for input and output files – so setting the working directory makes the commands simpler because you don’t have to specify complex paths in your commands to run analyses. This does not affect/is not affected by where the actual software +

packages are located – but as I said that makes no difference in practice.

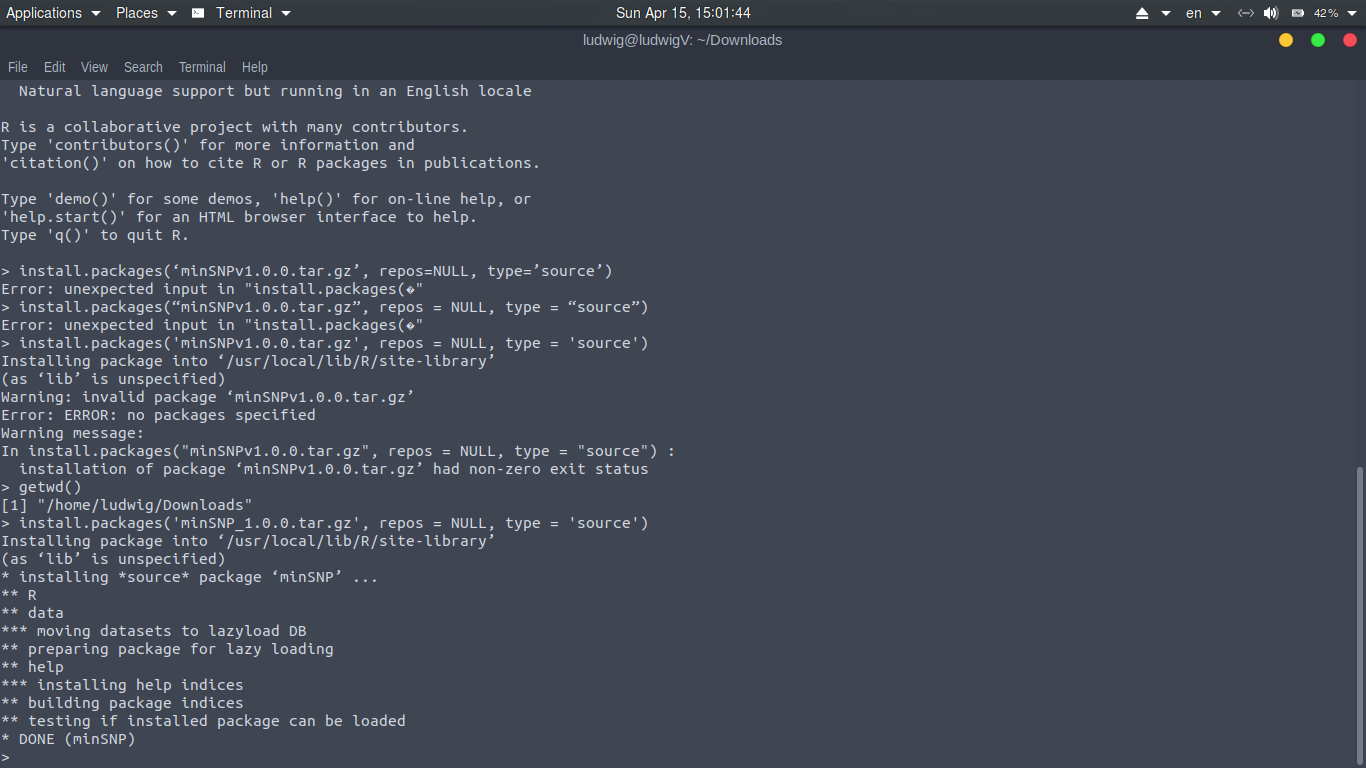
I have been using this script to set my working directory:

**setwd(“C:/Users/pgiffard/documents”)**

As at August 23 I have also set up the “minSNPs working files” directory on the M drive, and define this as the working directory using this script:

**setwd("M:/Units/infectious-diseases/minimum\_SNPs\_rewrite/minSNPs working files")**

Step 4: Install the package using the command ***install.packages(“minSNP\_1.0.0.tar.gz”, repos = NULL, type = “source”)*** (see Illustration 3)

  
Illustration 3: Installing minSNP package

**Comment:** Package installation in RStudio is covered above. Because I am using the GUI commands, I don’t actually use the command in step 4 above. Apparently setting repos to NULL and type to “source” is necessary (I have no clue what this means). After playing around with syntax, I managed to get this to work using two different command lines:

repos = NULL

type=“source”

Therefore I suggest running these commands after installing the *rlist* and *seqinr* packages – so you may not have to tick these – but I would imagine that it does not matter if you do.

Step 5: Verify package has been successfully installed. To verify run command ***library(minSNP)***. If the library has been successfully installed, there should be no output. If you see the following output: ***Error in library(minSNP) : there is no package called ‘minSNP’***, then the installation has either failed or has not been completed.

**Comment:** I believe that this basically switches the software on. Ticking the box in the package list is the same as running the library command. You can do either. I think that doing this with minSNP automatically activates the *seqinr* and *rlist* packages, so you may not have to tick these as well – but it probably does not matter if you do.

\*The package has only been tested in both Windows & Ubuntu environment.

# Using MinSNP

## Reading the fasta file

1. Import minSNP library into the workspace, using ***library(minSNP).***

**Comment**: this is covered in the comment above – it activates the software.

1. Read the fasta file and assign it to a variable, e.g. ***Chlaymydia<-readFasta("/home/ludwig/Downloads/Chlamydia\_1.txt")****.*

**Comment:** This is where you load the file from which you want to mine resolution optimised SNP sets. **Note, the syntax above is confusing simply because Ludwig has used an italics font which makes it almost impossible to see where the inverted commas are (this actually caused me significant drama).** The correct syntax is that the double inverted commas are around the path, then brackets are outside the inverted commas.

The file must have the following characteristics.

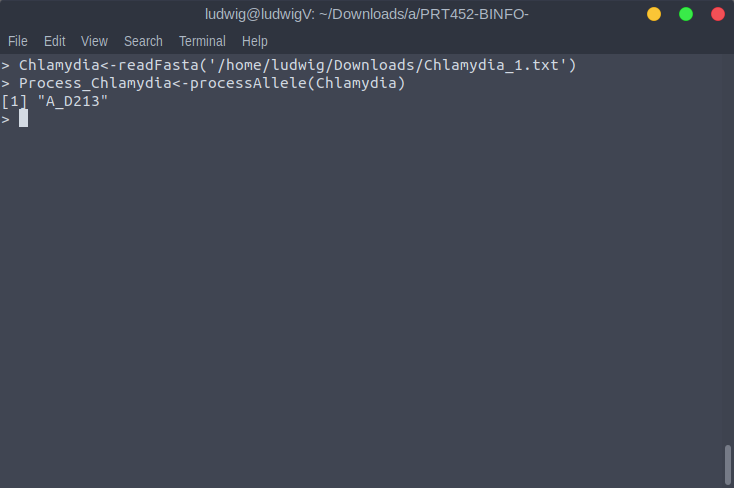
* 1. Multiple sequences in FASTA format in a single file (you cannot load multiple files)
  2. Txt format
  3. Non-standard bases are allowed.
  4. Indels are allowed, denoted by dashes.
  5. Aligned, so that all positions in the sequences are orthologous. This implies that all the sequences have to be the same length – or, to be exact, if they are not, then the truncated sequences have to be lengthened with dashes, and/or indels have to be indicated. I am not sure what happens if sequences of different length are included – but it is possible to remove non-standard length sequences prior to analyses.

**This script works for me:**

**chlamydiaSNPs<-readFasta ("Chlamydia\_Frankenstein\_SNPs\_orig.txt")**

I read this out of the directory that I had set to the working directory, which is why I did not have to include a path. If you do need to include a path, then use forward slashes. Don’t forget the filename extension. The file is a genome-wide orthologous SNP alignment from Chlamydia trachomatis. The name “chlamydiaSNPs” is arbitrary.

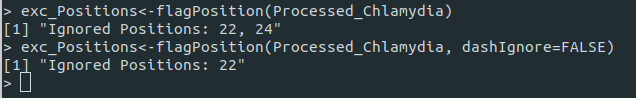
1. Check the alleles and flag any that is excluded from operation due to differences in length. ***Processed\_Chlamydia<-processAllele(Chlamydia)***. Excluded Allele’s name will be printed out, if there’s any. E.g. A\_D213 is excluded from operation (see Illustration 4). The processed\_Chlamydia is now ready for operation.

  
Illustration 4: Exclude allele

**Comment: This script works for me:**

**Processed\_chlamydiaSNPs<-processAllele(chlamydiaSNPs)**

1. Check if there are any invalid character in the sequence. If there is any character that is not a G, A, T or C, then the position of that character is ignored. The character does not need to be in all the alleles – even if it is just one, then the position will be ignored. **Exc\_Positions<- flagPosition(Processed\_Chlamydia*).*** The excluded positions will be printed in the console (see illustration below).

  
Illustration 5: How to flag positions

**Comment:** I think what Ludwig has written is slightly misleading, because this procedure does not delete these positions from the alignment or tell all analyses to ignore these positions. What this does is creates a variable called exc\_Positions that contains all the alignment positions where at least one of the sequences has a base that is not G, A, T or C. To be used, this variable then has to be included in an analysis command with the “excluded” function which actually tells the program which positions to exclude, with syntax like this ***excluded=exc\_Positions*.** In other words, excluding positions with non-standard bases is a two-stage procedure, with the first stage identifying such positions, and the next stage including these as excluded positions in an analysis. I think that the “exc\_Positions” variable name is actually arbitrary – you can call it anything.

**This function is the bomb for dealing with “N’s”, of which there can be many in orthologous SNP matrices.**

This script works for me:

**exc\_Positions<-flagPosition(Processed\_chlamydiaSNPs)**

This actually shows the result in truncated form in the coding screen (bottom left) – a large number of positions are identified.

If “dashignore=FALSE” is added to the command, then dashes (indels) are not identified as non-standard characters, so can included in analysis as informative, even when all other non-standard characters trigger ignoring of that position.

## Percent Mode

**Comment:** ok it is time for your first analysis. This is a percent mode analysis, which means it finds find SNPs diagnostic for a particular sequence in the alignment.

1. Use the following function, ***result=branch.percent(Processed\_Chlamydia, ‘A\_D213’, 3, excluded=exc\_positions, numRes=3)***, where the first argument is a processed allele and the second argument is the name of the allele to be compared, the third being how many positions are joined together, excluded are the positions that are excluded from calculation and numRes is the number of result to obtain.
2. To see the residual, use the function ***percent.residual(Processed\_Chlamydia, ‘A\_D213’, c(2))***. It will returns the alleles which have been falsely identified as A\_D213 if we use position 2.
3. Another method to present the result is to use ***present.percent(Processed\_Chlamydia, ‘A\_D213’, result)***. Using this method will return the residual as well as the result together. See illustration 8.
4. To output the result to csv, use the function ***output.percent(result, ‘A\_D213’, Processed\_Chlamydia, ‘filename’)***. The output csv (filename.csv) can be found in the current directory.[[5]](#footnote-5)

Comment: the above steps are actually quite idiot-proof.

This script works for me:

**result=branch.percent(Processed\_chlamydiaSNPs, 'L3\_404', 3, excluded=exc\_Positions, numRes=2)**

This is asking the program to find two sets of SNPs each of size of three SNPs, diagnostic for sequence L3\_404 in the alignment. It generates the variable with name “result” with the following contents: (I think the name “result” might be arbitrary). If you want to see the result, just type “result” (without the inverted commas) and hit return. This is what you get.

$`result 1`result

$`result 1`[[1]]

$`result 1`[[1]]$position

[1] 522

$`result 1`[[1]]$percent

[1] 100

**Comment:** The above says that just a single position – position 522 (result set 1, position 1) – unambiguously identifies sequence L3\_404

$`result 1`[[2]]

$`result 1`[[2]]$position

[1] 522 1

$`result 1`[[2]]$percent

[1] 100

**Comment:** the above refers to result set 1, the first two SNPs (positions 522 and 1). Obviously it is still 100%, because you can’t get higher than the 100% that you get from position 522 by itself. A minor issue is that the output in the two lines above does not make it absolutely clear it is looking at the resolving power of two SNPs – the two in the double square brackets means “the first two SNPs/positions in the set”, not “the second SNP in the set by itself”.

That second two SNPs are alignment positions 1 and 2 is a consequence of the fact that position 522 gives 100%. This the maximum possible resolving power than that, so all other positions in the alignment give the same performance of adding zero extra resolving power, so the program picked the first two it came to – positions 1 and 2.

$`result 1`[[3]]

$`result 1`[[3]]$position

[1] 522 1 2

$`result 1`[[3]]$percent

[1] 100

**Comment:** The second result below show that position 768 by itself also unambiguously identifies the target sequence.

$`result 2`

$`result 2`[[1]]

$`result 2`[[1]]$position

[1] 768

$`result 2`[[1]]$percent

[1] 100

$`result 2`[[2]]

$`result 2`[[2]]$position

[1] 768 1

$`result 2`[[2]]$percent

[1] 100

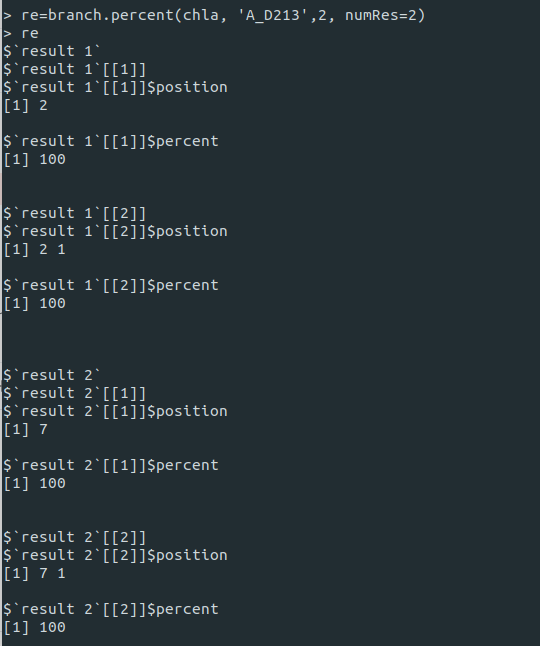
$`result 2`[[3]]

$`result 2`[[3]]$position

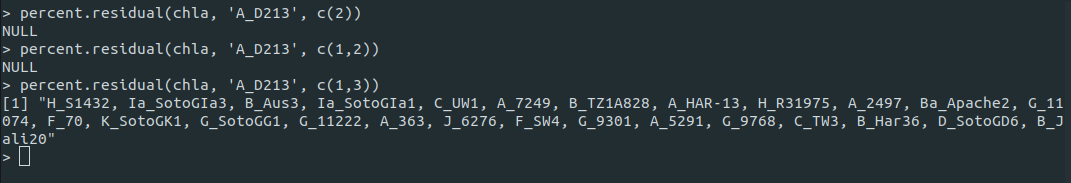
[1] 768 1 2

$`result 2`[[3]]$percent

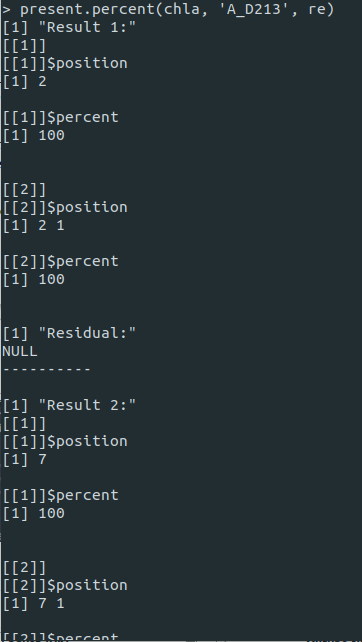
[1] 100

  
Illustration 6: Sample percent mode result

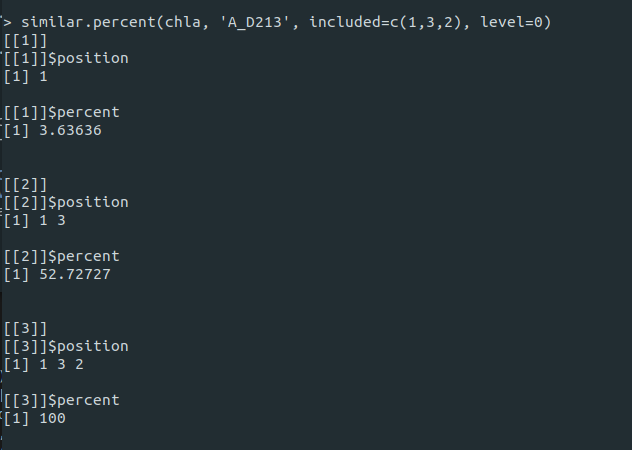
\*More values can specified with c(1, 2,3 ....), e.g. ***percent.residual(chla, 'A\_D213', c(1, 2))*** will give alleles falsely classified as A\_D213 when position 1 and 2 are used. Since 2 completely differentiate A\_D213, that returns no result. An instance where the function returns result is shown by third query in illustration 7, e.g. ***percent.residual(chla, ‘A\_D213’, c(1,3))***.

  
Illustration 7: Percent.residual sample

The percentage for these positions of interest can be checked using ***similar.percent(chla, 'A\_D213', included=c(1,3,2), level=0)***, \*note that the order in which the included positions are given will affected the cumulative percentage, 1, 3, 2 will give percentage of 1, 1& 3 combined and 1, 3, 2 combined.

  
Illustration 8: Using present.percent

pwercent mode

  
Illustration 9: Similar.percent sample

## Comment: How to find SNPs diagnostic for groups of variants (groups of interest).

This is outlined in full here by Phil because it encompasses a written command that Ludwig has not yet fully included in his version of the manual.

The way this is done is by piggy-backing on the “branch.percent” and “excluded” commands. This minimised the extra coding that was necessary to make this work. The logic is that if all positions that are variable within the group of interest are excluded from the analysis, then using the percent mode to find SNPs diagnostic for any member of the group of interest will identify SNPs that are optimised to discriminate the group of interest from all other variants, without dividing up the group of interest (which is undesirable). While this might not appear to be a very efficient use of genetic variation – if the members of the group of interest are not very closely related then a very large number of positions will be excluded – it relies on the fact that genome-wide orthologous SNP matrices encompass massive amounts of SNP variation information, so you can afford to lose a lot. Also, this provides SNPs that on the basis of known variation will never give a false negative for the group of interest, which is arguably ideal for surveillance.

**The key new command is “distinct”.** This identifies positions that are variable in a group of interest. This information is assigned to a variable, that can then be used in an “excluded” command in a branch.percent analysis.

The syntax is as follows:

* 1. The identities of the group of interest are defined to a variable, here named “interests”.

Here is script that works, with the brackets and inverted commas inclosing the group of interest.

**interests=c('B\_TZ1A828', 'B\_Har36', 'B\_Jali20', 'Ba\_Apache2')**

* 1. The positions that exhibit variation in the group of interest are assigned to a variable, here called “exc\_distinct\_positions”, using the wonderful “distinct” command.

**exc\_distinct\_positions=distinct(****Processed\_chlamydiaSNPs, interests)**

* 1. A percent mode analysis, is performed using one of the group of interest, selected at random, as a target. The script below shows how multiple sets of excluded positions can be included in a single command. The exclusions are the “exc\_flag\_positions” which are positions with non-standard bases, as well as the “exc\_distinct\_positions”. Note this required first re-running the flag positions command to put the positions with non-standard characters into a more descriptive variable name. The two lines of script are:

**exc\_flag\_Positions<-flagPosition(Processed\_chlamydiaSNPs)**

**result=branch.percent(Processed\_chlamydiaSNPs, 'B\_Har36', 2, excluded=c(exc\_distinct\_positions, exc\_flag­\_Positions), numRes=1)**

**This takes several minutes to run. The result is:**

$`result 1`

$`result 1`[[1]]

$`result 1`[[1]]$position

$`result 1`[[1]]$position[[1]]

[1] 1773

$`result 1`[[1]]$percent

[1] 92.1875

$`result 1`[[2]]

$`result 1`[[2]]$position

$`result 1`[[2]]$position[[1]]

[1] 1773

$`result 1`[[2]]$position[[2]]

[1] 24

$`result 1`[[2]]$percent

[1] 95.3125

The command only asked for a single SNP set with two members. The set is positions 1773 and 24, which gives a resolving power of 95.3%. Note that with analyses like this you cannot achieve 100% because it is impossible to separate your target allele from the other members of the group of interest – which is the whole idea of course.

The command to specifically see how well this worked is:

**percent.residual(Processed\_chlamydiaSNPs, 'B\_Har36', c(1773,24))**

The result from that is: [1] "B\_TZ1A828, Ba\_Apache2, B\_Jali20".

**This shows that the analysis has worked perfectly** in that the only alleles that are not discriminated from B\_Har36 by positions 1773 and 25 are alleles B\_TZ1A828, Ba\_Apache2, B\_Jali20 – which are exactly the other members of the group of interest.

This script will output the result to a csv file in the working directory, with the file name:

SNPs\_BBa\_classic\_ocular

**output.percent(result, 'B\_Har36',Processed\_chlamydiaSNPs, 'SNPs\_BBa\_classic\_ocular')**

This is cut and pasted from the CSV file opened in Excel



## Using D mode

1. Use the following function, ***result=branch.simpson(Processed\_Chlamydia, excluded=Exc\_Positions, level=1, numRes=3)***. The first argument for the function is the processed allele, the second argument is the number of positions considered and the third argument is the number of result to be returned.[[6]](#footnote-6)
2. Use the following function to view the result, ***present.simpson(Processed\_Chlamydia, result)***. The result shows how the genotype are grouped if the allele at position is used. It also shows the position of the allele in Description and the simpson index in Index. (see Illustration below)
3. To output the result to a csv file, use the function ***output.simpson(result, Processed\_Chlamydia, ‘filename’)***, note that the final argument is optional (if the filename is not supplied, then the output will be in a csv will be named with system date time. The output csv can then be found in the directory where R is started. Alternatively, it can be specified, e.g. ***output.simpson(result, Processed\_Chlamydia, ‘/home/ludwig/something’).*** The csv file (something.csv) can then be found in the directory *home/*ludwig.

**Comment:** the above is more or less idiot-proof.

This script worked for me. It provided three sets of three SNPs.

**result=branch.simpson(Processed\_chlamydiaSNPs, excluded=exc\_Positions, level=3, numRes=3)**

This put it into a CSV file named “D\_test”:

**output.simpson(result, Processed\_chlamydiaSNPs, 'D\_test')**

This script is **THE BOMB** – it provides complete information on how the different SNP profiles divide the sequence variants.

**present.simpson(Processed\_chlamydiaSNPs, result)**

**e.g. for SNP set 1 (positions 1952, 11508 and 16210):**

[[1]]

[[1]]$ccc

[1] "A\_D213" "C\_UW1" "C\_TW3"

[[1]]$cac

[1] "H\_S1432" "H\_R31975"

[[1]]$cag

[1] "Ia\_SotonIa3" "Ia\_SotonIa1" "K\_SotonK1" "J\_6276"

[[1]]$tac

[1] "B\_Aus3" "B\_Aus36" "B\_Aus5" "B\_Aus2" "B\_Aus4"

[[1]]$tta

[1] "F\_SW5" "Ba\_Aus25" "Ds\_2923" "F\_70" "Ba\_Aus28" "F\_SW4" "F\_SotonF3" "D\_SotonD1"

[[1]]$gca

[1] "L2b\_CV204" "L2\_LST" "L2b\_UCH-1" "L2b\_795" "L1\_224" "L2b\_C1" "L2b\_8200" "L2b\_C2" "L2\_434"

[10] "L1\_115" "L2b\_UCH-2"

[[1]]$tca

[1] "L1\_440" "L1\_SA16"

[[1]]$ccg

[1] "A\_7249" "A\_HAR-13" "A\_2497" "A\_363" "A\_5291"

[[1]]$tcg

[1] "B\_TZ1A828" "B\_Jali20"

[[1]]$gta

[1] "E\_150" "E\_SotonE4" "E\_SW2" "E\_SW3" "E\_SotonE8"

[[1]]$gaa

[1] "E\_Bour" "E\_11023"

[[1]]$cta

[1] "C\_Aus10" "C\_Aus8" "C\_Aus9" "C\_Aus33" "C\_Aus30"

[[1]]$tcc

[1] "Ba\_Apache2" "B\_Har36"

[[1]]$cca

[1] "L3\_404"

[[1]]$tag

[1] "D\_UW-3" "D\_SotonD6" "D\_SotonD5"

[[1]]$gag

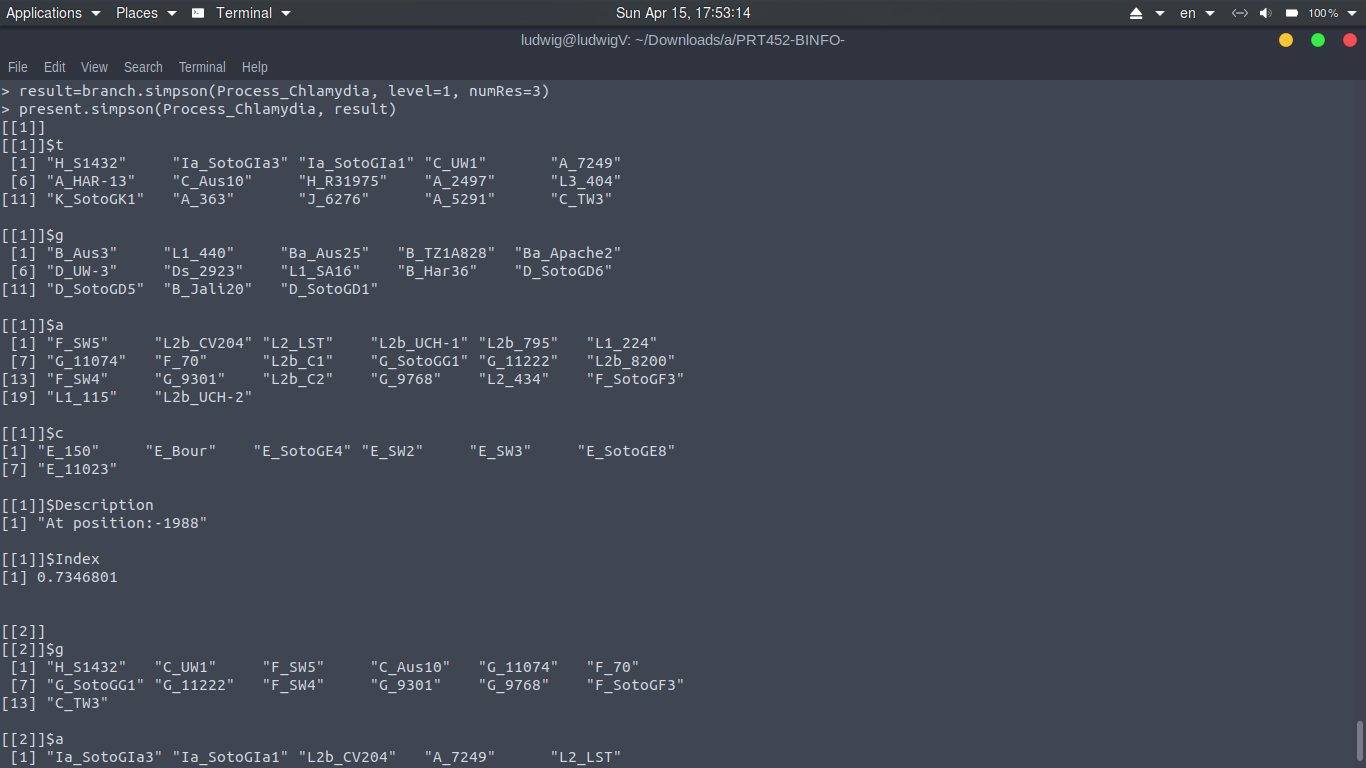
[1] "G\_11074" "G\_SotonG1" "G\_11222" "G\_9301" "G\_9768"

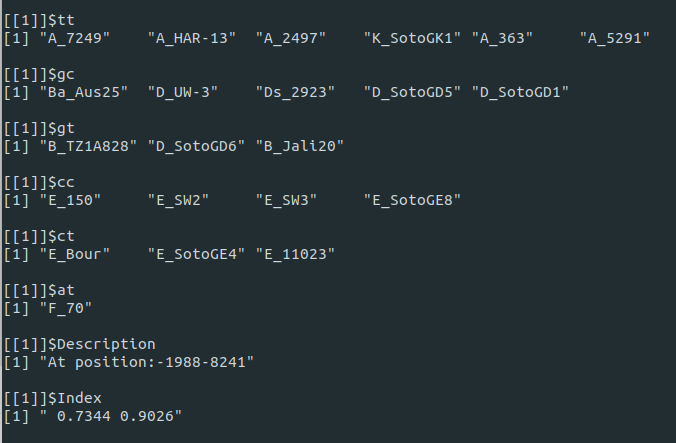
[[1]]$Description

[1] "At position:-1952-11508-16210"

[[1]]$Index

[1] " 0.676 0.8952 0.9279"

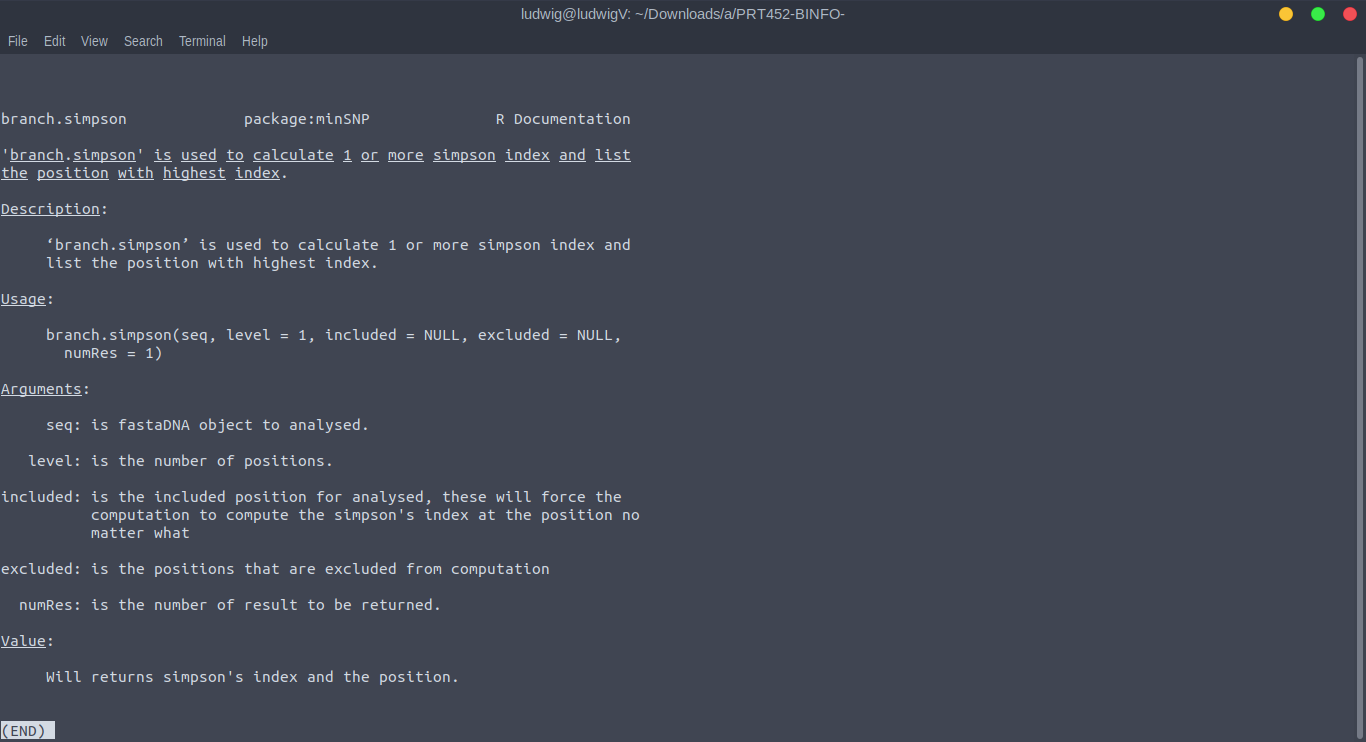
  
Illustration 10: D mode

  
Illustration 11: Sample d Mode result

The result shows how the sequences are typed, the positions used, the index and also the cumulative index.

# Functions & descriptions

* Presented here are the usual use-case. It is also possible to include and exclude certain SNPs when calculating simpson index. To include or exclude certain position of allele when calculating simpson index, use the following function ***branch.simpson(Processed\_Chlamydia, included = c(1,2,3), excluded=c(4,5,6), level=1, numRes=3)***. In the example, position 1, 2, 3 are included and position 4, 5, 6 are excluded.
* ***flagPosition(proSeq, dashIgnore=TRUE)***, by default, flagPosition will see ‘-’ as a position to be ignored. It can be set otherwise, by adding the adding argument ***dashIgnore=FALSE*** (see Illustration 5).
* If you need more help with the functions, use ***help()*** function in R, e.g. ***help(branch.simpson)*** will show the documentation of the function. (see Illustration below)
* All the functions that a normal user will interact with are all listed in this document.
* For developers, please refer to <https://github.com/ludwigHoon/PRT452-BINFO-/blob/master/minSNP/man/documents/minSNP.pdf>

  
Illustration 12: Using help (branch.simpson)

## HRM Module

### Prerequisites

1. Loci information in a csv file in the following format

*filename, allele\_length, fragment\_name, fragment\_start, fragment\_stop*

Where each row is a region for analysis,

* filename is the name for the alleles file of the locus (see 2)
* allele\_length is the length of the allele at the locus, any allele that has different length will be ignored
* fragment\_name is the name of the region under analysis
* fragment\_start is where the region under analysis begins
* fragment\_stop is where the region under analysis ends

1. Alleles files in fasta format, these can be obtained from <https://pubmlst.org/>. These files need to be individial alleles for each locus, concatenated sequence is not supported at the moment.
2. ST profiles information in a csv file in the following format

*ST, arcC, aroE, glpF, gmk, pta, tpi, yqiL, clonal\_complex*

This file can be obtained from <https://pubmlst.org/>.

1. Changes in the curve in a csv file in the following format

*fragment\_name, type, replace\_with, if\_match\_fragment, if\_match\_curve*

Where each row is a changes that need to be made before further processing,

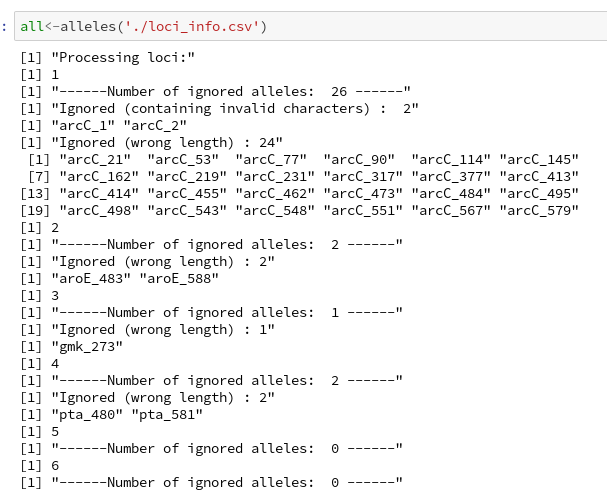
* fragment\_name
* type
* replace\_with
* if\_match\_fragment
* if\_match\_curve

Examples:

* tpi241/243,1,42,GCG...GG,\_ *(This changes tpi241/243 from 43 to 42 for those STs that have a sequence here that matches that for ST59)*
* tpi36,2,65,\_,64 *(This changes tpi36 from 64 to 65 for all sequences as these could not be discriminated)*
* tpi36,1,67,TGC...GGG,\_ *( This changes tpi36 to 67 for those STs that have a sequence here that matches that for ST93)*
* *aroE88/155,1,23.5,ATT...TGA,\_ (This changes aroE to 23.5 for those STs that have a sequence here that matches that for ST59)*
* aroE88/155,1,24.5,ATT...TGA,\_ *(This changes aroE to 24.5 for those STs that have a sequence here that matches that for ST39 and ST121)*

### Steps

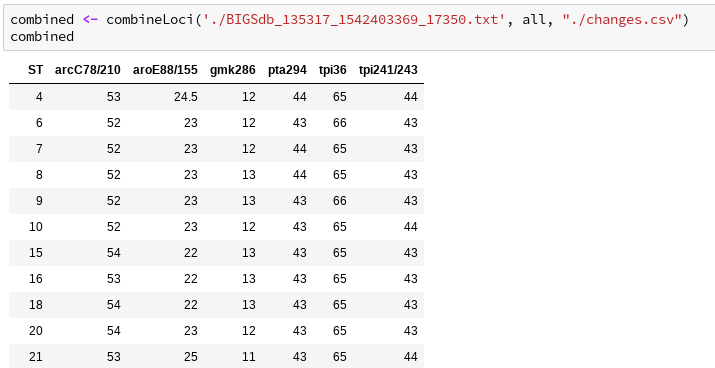
1. Load the loci information file



Total number of alleles ignored from analysis are shown, their name as well as the reasons, i.e. wrong length or invalid characters. (Valid characters are A, T, C, G, and ‘-’ )

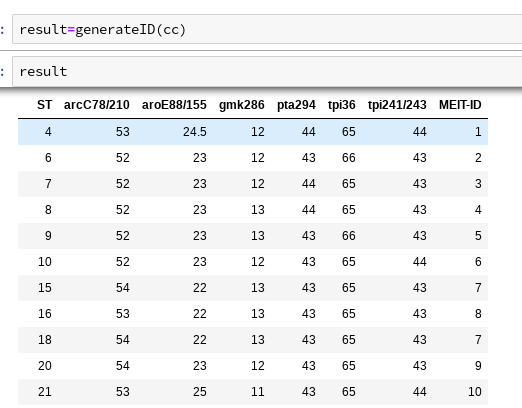
Comment: The result shows the allele that has been dropped due to incorrect length and containing invalid characters. Some fragments in arcC were modified to demonstrate the functionalities. The file provided in pubmlst does not contain invalid characters.

1. Load the ST profiles and combine the allele information, ST profiles from pubmlst is tab-delimited, & the function on handle tab-delimited file. The file containing the changes to be made to curve number should be included together into the function. If no changes are to be made to the cuve number, the third argument can be left empty, i.e. *combined <- combineLoci(“./BIGSdb\_135317\_154203369\_17350.txt”, all).*



Comment: The result shows the curve number for sequences at different allele. Sequences will be dropped if the curve number for any of its alleles is not available. This can be due to previous step where thefragment is dropped. In this example, it can be seen that ST 1, 2, 3, 4, … are dropped. A log is generated which show a list of modified curved number. More on that later.

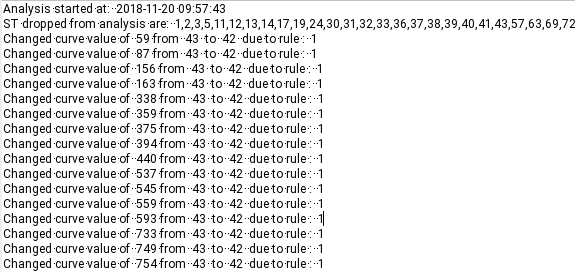
1. Generate MelT-ID



1. (Optional) Write result to csv file, *write.csv(result, "result.csv"),* the result will be saved as a csv file in the directory specified earlier.

Log file

Comment: The log file shows the date and time when the analysis started (i.e. when the function is called, this is to differentiate between different log), ST dropped from analysis and also a list of changed curve number and the relevant rule that caused the change, e.g.



This can be used to traced back the changes that were made.

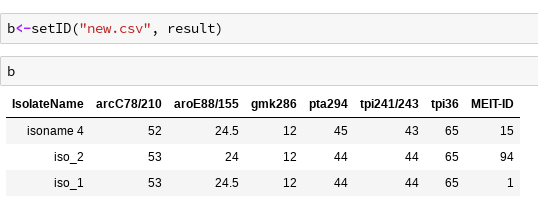
### Comparing result & append Melt-ID

Prerequisite:

* A csv file with the following format:
  + First row should consist of the column names, separated by comma and should be the same as those in the result
  + The order of the column does not matter, but must otherwise be the same, spelling, symbols, etc.
  + All the columns / fragments that exist in the result dataframe must also exist in the csv
  + An extra column called IsolateName can be present.
  + E.g. (For this case study)

IsolateName, arcC78/210,aroE88/155,gmk286,pta294,tpi241/243,tpi36  
isolate 1, 53,24.5,12,44,44,65

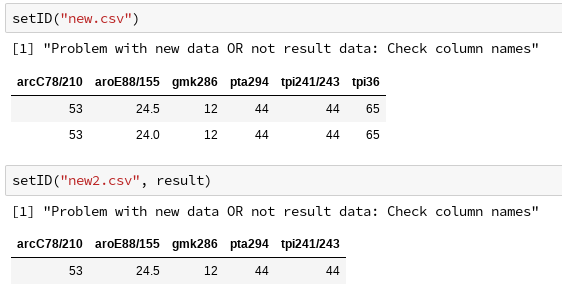
Step: Use the function setID(“csv\_file\_name”, result)



will output the Melt-ID at the end.

Comment: The function matches each row of data with the result and identify the MelT-ID with the same curve numbers. The comparison will not start if the name of the columns do not match.

If result is not specified, not columns are presence, or columns have different names, a warning will be shown and the original data frame from the csv file without Melt-ID appended will be returned.



1. For windows, any of the following distributions (base, contrib, old-contrib, Rtools) are fine, but base distribution is recommended. [↑](#footnote-ref-1)
2. For Ubuntu, run ***sudo apt-get install r-base***. [↑](#footnote-ref-2)
3. If you are using Ubuntu, start it with admin privilege, i.e. ***sudo R***. [↑](#footnote-ref-3)
4. If you are using Ubuntu, run the following command before install rlist, ***sudo apt-get install r-cran-xml***. [↑](#footnote-ref-4)
5. Filename is optional, check step 3 in Simpson analysis for more details. [↑](#footnote-ref-5)
6. Note that this operation will takes some time depending on the performance of the machine and the length of the Fasta file, as well as the level of search. [↑](#footnote-ref-6)