Part 2 - a glimpse into tidyverse world

Data frames and tibbles

Data frames are similar to tables. They are very handy for storing and processing observations, as each column can hold data of different type. Let's read a table with results of an experiment. Tibbles are a tidyverse extended version of data.frames and they use their own, simpler syntax. We will start by loading a library tidyverse - a bundle with many data processing libraries.

```
library( tidyverse )
```

We'll read input data directly from a webpage. We will use the data about immune cells (collected by flow cytometry) from COVID patients: https://www.immunophenotype.org/ :

```
covid_data <- read_csv( "https://www.immunophenotype.org/wp-content/uploads/2020/07/2020-06-30flow_sero

#How big is this dataset? ###
covid_data

#print a useful summary
glimpse( covid_data )

nrow( covid_data )
ncol( covid_data )
```

Accessing data in a tibble

You can use the same method as for data frames:

```
covid_data[ 1:3, c( 1, 4, 55 ) ]

#what type is data in the columns Clinical_sample and `B_01/CD45_01 |freq` ?
covid_data[ 1:3, "Clinical_sample" ]

#But you can also access specific column with the operator $ ( as for list ). #type covid_data$ and hit

#Additional_point
#Have a look on column names of covid_data. Some of them contain 'forbidden' characters - spaces, "|".
```

Operator %>%

In R we often perform whole series of operations: for example subset a table by rows and columns, compute an average of some columns, add a number to this average. Operator %>% (pipe) helps to make it more legible by using an output of the function before pipe as input of the function after pipe.

```
b <- 1:10


b %>%    #take b
 .[ 1:4 ] %>%   # take first 4 elements of b
 mean( . ) %>%  #compute a mean
 + 3 %>%   #add 3
paste( "result is", . ) %>% #add text
 print( )


#This is the same as:
print( paste ( "result is", mean( b[ 1:4 ] ) + 3 ) )

#what will be the result of this operation?
a <- 4

a %>%
 sqrt( ) %>%
 .^2
```

Pipes are especially helpful for filtering and summarizing data.

Selecting columns and filtering rows in tibbles

One can select columns and filter rows for specific values with functions select( ) and filter( ). For filtering helpful are: == , >, >= , <, <= , != , %in%. Select let's you also easily change column order and give a column a new name when selecting

```
#prepare a smaller table with columns "T_01/CD45_01 |freq" and "Memory_B_01/B_01 |freq"
covid_data %>%
 select( "Clinical_sample", "T_01/CD45_01 |freq", "Memory_B_01/B_01 |freq" )

#Now changing column order
covid_data %>%
 select( "T_01/CD45_01 |freq", "Memory_B_01/B_01 |freq", "Clinical_sample" )

#With tibbles, quotes around well formed names are unnecessary. Names with spaces, / and other bad char
covid_data %>%
 select( Clinical_sample, `T_01/CD45_01 |freq`, `Memory_B_01/B_01 |freq` )


#Find all samples with less than 10% of T cells in CD45 cells
covid_data %>%
 select( Clinical_sample, `T_01/CD45_01 |freq` ) %>%
 filter( `T_01/CD45_01 |freq` < 0.1 )


#Find frequencies of T and B cells for sample "p028n01" ( this is 'patient 28, sample 1' )
covid_data %>%
 select( Clinical_sample, `T_01/CD45_01 |freq`, `Memory_B_01/B_01 |freq` ) %>%
 filter( Clinical_sample == "p028n01" )
```

```
#Now you. How many samples has patient p001? Add missing parts of the code
#covid_data
# filter( )
```

```
#Merge sample information with clinical information, joining two tables by patient ids. From the big ta


patient_info <- read_csv( "data/20-07-25patients_metadatav2.csv" )

columns_to_take_from_big_data <- colnames( covid_data ) %>%grep( pattern = "Median|Ratio|freq", value =

covid_data_annotated <- covid_data %>%
 select( Clinical_sample, patient_id, columns_to_take_from_big_data ) %>%
 left_join( patient_info, ., by = c( "patient_id" ) )

write_csv( covid_data_annotated, file = "data/covid_data_annotated.csv" )
```

```
covid_data_annotated <- read_csv( "data/covid_data_annotated.csv" )

colnames( covid_data_annotated )

#Select a subset of columns, giving them handy names.

#columns to choose: "Time_03/Cells_03/Singlets1_03/Singlets2_03/Live_03/CD45p_03/T_03/CD3p_gdn_03/CD4_0
# "Time_03/Cells_03/Singlets1_03/Singlets2_03/Live_03/CD45p_03/T_03/CD3p_gdn_03/CD8_03 | Count_back"


covid_data_annotated_small <- covid_data_annotated %>%
 select( Clinical_sample, patient_id, sex, age, status = class_dss,
   CD4 = "Time_03/Cells_03/Singlets1_03/Singlets2_03/Live_03/CD45p_03/T_03/CD3p_gdn_03/CD4_03 | Count_ba
   CD8 = "Time_03/Cells_03/Singlets1_03/Singlets2_03/Live_03/CD45p_03/T_03/CD3p_gdn_03/CD8_03 | Count_ba


#Add a column Tcell_freq by dividing values in
```

Adding new columns

New columns get added by the function mutate( )

```
#Ad a new column: with ratio of CD4 cells to CD8 cells
covid_data_annotated_small <- covid_data_annotated_small %>%
 mutate( CD4_CD8_ratio = CD4/CD8 )
```

Summarising

Function summarise( ) gives one number summaries

```
#summarise: what summarie
covid_data_annotated_small %>%
 summarise( mean_CD4 = mean( CD4 ),
   mean_CD8 = mean( CD8 ),
   median_CD4 = median( CD4 ),
   median_CD8 = median( CD8 ),
   max_ratio = max( CD4_CD8_ratio ),
   min_ratio = min( CD4_CD8_ratio ),
   N = n( ), individuals = n_distinct( patient_id ) )

#This did not work as intended - many functions require special dealing with missing values ( NA ). For

covid_data_annotated_small %>%
 summarise( mean_CD4 = mean( CD4, na.rm = T ),
   mean_CD8 = mean( CD8, na.rm = T ),
   median_CD4 = median( CD4, na.rm = T ),
   median_CD8 = median( CD8, na.rm = T ),
   max_ratio = max( CD4_CD8_ratio, na.rm = T ),
   min_ratio = min( CD4_CD8_ratio, na.rm = T ),
   N = n( ), individuals = n_distinct( patient_id ) )
```

Grouping

Grouping data by a column value permits to perform operations per group. You can group by more than 1 column. To remove grouping: ungroup(). To treat each row separately use rowwise()

```
#Group by sex and perform the same calculations as before
 covid_data_annotated_small %>%
 group_by( sex ) %>%
 summarise( mean_CD4 = mean( CD4, na.rm = T ),
   mean_CD8 = mean( CD8, na.rm = T ),
   median_CD4 = median( CD4, na.rm = T ),
   median_CD8 = median( CD8, na.rm = T ),
   max_ratio = max( CD4_CD8_ratio, na.rm = T ),
   min_ratio = min( CD4_CD8_ratio, na.rm = T ),
   N = n( ), individuals = n_distinct( patient_id ) )

#Now the same, but group by sex and disease status
covid_data_annotated_small %>%
 group_by( sex, status ) %>%
 summarise( mean_CD4 = mean( CD4, na.rm = T ),
   mean_CD8 = mean( CD8, na.rm = T ),
   median_CD4 = median( CD4, na.rm = T ),
   median_CD8 = median( CD8, na.rm = T ),
   max_ratio = max( CD4_CD8_ratio, na.rm = T ),
   min_ratio = min( CD4_CD8_ratio, na.rm = T ),
   N = n( ), individuals = n_distinct( patient_id ) )

#Now you: fill in the code. Count number of samples per individual ( group by individual and use n( ) )

covid_data_annotated_small
 group_by( sex, status ) %>%
 summarise( N = )

# What was the biggest number of samples per individual?
covid_data_annotated_small
 group_by( sex, status ) %>%
 summarise( N = ) %>%
 summarise ( max_no_of_samples = )
```

ggplot

The package ggplot2 and its many related packages allows to get beautiful graphics, change your plots quickly and also fine-tune details. It is based on the concept of layers- you can add many charts from one dataset to a plot. Each layer is prepared with the function geom_XXX, where XXX is type of plot. Information from your input data is mapped to different aesthetics: for example CD4 number is presented on y axis, age group on x, sex as colour. Parts of the ggplot( ) are added with + .

```
covid_data_annotated_small %>%
 ggplot( ) +
 geom_point( aes( x = age, y = CD4, col = sex ) )

#Now the same, but plotting as boxplots
covid_data_annotated_small %>%
 ggplot( ) +
 geom_boxplot( aes( x = age, y = CD4, fill = sex ) )


#Now you. Plot CD8s by status
```

lapply()

lapply() applies a function on each elemment of a list separately HERE I AM NOT SURE WHETHER TO NOT SWITCH TO map() or even a for() loop

```
my_list <- list( 1:3, 4:8, 0:2 )
lapply( list( 1:3, 4:8, 0:2 ), max )  #computes max of each list element and returns a list with result

#You can use a ready-made function or make your own:
add_flower <- function( word ){
 paste( word, "flower" )
}

lapply( 1:3, add_flower )
```