S3.0: Recap last time

1. Generate a matrix filled with random numbers and dimensions of your own choosing.
2. Calculate the mean of the rows of that matrix.
3. Create a new matrix, which is identical to the previous matrix, except that you divide each element by the mean of its corresponding row.
4. Now calculate the row-means again, they should all equal 1 now.
5. In the previous exercises you normalized the matrix by row. Try to do it per column now. (This involves a "trick".)

S3.1: Data frames & import/export data

1. Create the following dataframes:

```
df1 <- data.frame(gene = c('rgma', 'prr18', 'thrap3b',
'cdkn1ba', 'oaz1a'), cluster = c(1,3,2,2,5), replicate =
c(1,1,1,2,2))

df2 <- data.frame(gene = c('zeb1a', 'prickle2b', 'cdc34b',
'mpped2', 'krtcap2', 'arl4ab', 'xbp1', 'admp'), cluster =
c(4,4,3,4,1,2,4,2), replicate = c(3,3,3,4,4,5,5,5))

df3 <- data.frame(organism = c('zebrafish', 'zebafish',
'mouse', 'zebrafish', 'celegans'), expression = c(3.12,
5.50, 0.31, 1.00, 90.31), sgRNA = c(T,T,F,F,F))
```

   a) Explore the output of `rbind(df1,df2)`.
   b) Explore the output of `cbind(df1, df3)`.
   c) Why does `rbind(df1, df3)` give an error?
   d) From the output of `cbind(df1, df3)`, select those rows that are assigned to zebrafish.

2. With the following code:

```
dfa <- data.frame(cell1 = runif(10), cell2 = runif(10),
cell3 = runif(10), cell4 = runif(10))
row.names(dfa) <- c('g01', 'g02', 'g03', 'g04', 'g05',
'g06', 'g07', 'g08', 'g09', 'g10')
```

We are creating a gene expression table, where each column is a sample and each row a gene. Numbers (now created using random numbers) are supposed to be normalized gene expression values.

   a) Select the expression values generated for cell 1

b) Calculate the mean mean of all the expression values detected for cell 3
c) Using the function `apply` obtain the total expression (sum) detected in each cell
d) Again, using the function `apply`, calculate the mean expression for each gene among the different cells.

3. **Working with metadata.** In the repository GSE81608, scRNAseq data from several cells isolated from the pancreas is stored. We downloaded the metadata from this repository and saved it in the file "metadata_pancreas_scRNAseq.tsv". Import this file to your R session using the following code:

```
mdf <- read.table('metadata_pancreas_scRNAseq.tsv', sep = '\t', header = TRUE, stringsAsFactors = FALSE)
```

If you type `head(mdf)` or `tail(mdf)`, the first or last few rows of the imported data frame will be printed on screen.

Now, answer the following questions using R commands (you might need a combination of commands to answer some of the questions).

a. Use the command `dim(mdf)` to answer how many columns and how many rows are in this file.
b. Which are the names of each column?
c. Which are the names of each row?
d. Which different cell types are annotated in this metadata? (Hint: you might want to use the command `unique`. This command takes as an input a vector, and provides as an output a vector with the unique elements of the input).
e. Which is the most abundant cell type? And which is the least abundant?
f. How many different ethnic groups are reported in this metadata?
g. Create a new data frame that contains only cells whose gender is female (F) and whose condition is T2D. How many cells are there in total? How many of these cells are from the 'beta' subtype?
h. Which is the age span of the donors? (that is, how old are the oldest and the youngest donors?)
i. You can use the order command to sort the rows in a dataframe according to a particular criteria. Explore the output data frame of these lines of code:

```
sdf <- mdf[order(mdf$age),]
```

and

```
sdf <- mdf[order(mdf$cell.subtype),]
```

j. Create a vector cell id's (srr column) from the PP subtype, detected in Female donors from ethnicity C that are younger than 60.
k. From these cells, how many come from a diabetic donor? And how many do not?

4. **Loading data, a simple example.**
   a. Go to https://www.timeanddate.com/weather/netherlands/amsterdam/historic and scroll down. There is a table of historic weather data. Copy some of these lines, open TextEdit, notepad or some other text editor, and paste the data there. Now

try to edit the data a bit such that it can be read in by the R function "read.table()".

b. Now go to Rstudio, and try to read in the data. (You might need to go back to [a] and make some changes to the file.)
c. Can you create two corresponding vectors, one with the time and one with the temperature at that time?

Note: all of this might involve some fiddling, and the exercise is also intended to illustrate that saving/loading files can come very precise.

5. **Working with gene expression tables.** For the same repository GSE81608, we got the gene expression table, which can be found in the file "merge_exon_uniqueCounts.coutb.tsv.gz". Use the following code to import this file as a data frame:

```
edf <- read.table('expression_pancreas_scRNAseq.tsv.gz',
sep = '\t', row.names = 1, header = TRUE)
```

Now that by adding the parameter "row.names = 1" we are asking R to assign the first column of the file to the row names of the data frame. Note that we do this because we know that this file is made of a table where the first row contains the cell ID, and the first column contains the gene.

a. How many cells and how many genes are found in this data frame?
b. Notice that now the output of the `head(edf)` command is not as nice as in the previous exercise. You can now explore how this dataframe looks like by typing: `edf[1:10, 1:5]`.
c. Which is the cell with the highest number of counts?
d. Which is the gene with the lowest number of counts? If there is more than one, how many genes have the same lowest number of counts?
e. Create a new data frame that contains only cells with more than 500000 reads. How many cells survive this filtering?
f. In the new filtered dataframe, are there genes with a total of 0 detected transcripts? If so, filter them out.
g. Now, we will normalize the data. This means that we will "manipulate" the counts per gene per cell so that all cells have the same total number of reads. A common approach is to divide the each gene count in each cell by the total number of counts in that cell. Let's go step by step:
    i. Create a new vector that contains the total number of counts per cell in the filtered data set (where cells are sorted in the same order than the data set).
    ii. Create a new data frame that is the transpose of the filtered matrix. That means that in the new data frame, columns become rows and rows become columns. This can be achieved by using the command `t(df)`, where `df` is the data frame that you want to transpose.
    iii. Now, divide the transpose data frame by the vector created in step (i).
    iv. Transpose the resulting data frame. This is your normalized data frame.

h. Compute the total number of reads per cell in your new normalized data frame. They should be all equal to 1. If not, that means that we did something wrong.
i. It is common practice to normalize the total counts per cell to 10000. This way, the computer does not need to work with very small numbers (which usually leads to numerical errors). To do so, multiply the normalized data frame by 10000 and assign it to a new variable.
j. Using the metadata information from the previous exercise, create a new data frame that contains the normalized count table for only beta cells and another one that contains the normalized count table for only alpha cells.
k. Which gene has the highest mean expression in alpha cells?
l. Which 10 genes have the highest mean expression in beta cells?
m. Using the `grep` command, find the row.name that contains the pattern '`_INS_`' in its name.

6. **Working with bed files.** A bed file is a file made of at least three columns; the first column is the chromosome ID, the second is the start of a genomic feature (exon, intron, promoter, CpG island…), the third is the end of the same genomic feature, and the rest of the columns might contain other information for the same feature. Of course, there might be variations to this format. We will now have a look at the file entitled "SRR1248457_methylationdata.cov.gz" with the following line of code:

```
df <- read.table('SRR1248457_methylationdata.cov.gz', sep = '\t', header = FALSE)
```

This file contains methylation data from a mouse embryonic stem cell treated with 2i. Each line contains information about the methylation status of a single detected CpG: position (chromosome, start and end, methylated counts, unmethylated counts, and methylation fraction.

a. How many columns are in the imported data frame?
b. Using colnames, rename the columns to: `chrom, start, end, frac ,meth, unmeth` (with this order, please).
c. Are columns start and end identical? Why?
d. Columns `meth` and `unmeth` contain the number of counts that a CpG. Create a new column in the data frame containing the total number of counts for each detected CpG.
e. In which chromosome is found the CpG that got more counts? And in which position of the chromosome? Which is the methylation fraction?
f. Using the `unique(df$chrom)` command, you can create a list of all chromosomes. Write a `for` loop that prints out the total number of CpG's detected in each chromosome.
g. Select all CpG in chromosome 1 that have a methylation fraction not equal to 0 or 100. What is the mean methylation fraction of these remaining CpG's? And what is the total number of counts?

7. **Save excel documents.** Find out how to save the CpG's from chromosome 1 that have more than 3 reads (previous exercise) as a data frame in an excel document.