Human Genome Analysis Lab 2 : Basic R Object types and importing SNP data

- Learning objectives
- Overview
- Basic Data Objects
    - Vectors
    - Factors
    - Matrices
    - Data frames
    - Checking on object types
- Importing data
    - read.table()
    - From Excel
- Loading a truncuated 23andMe file
    - Getting Information on a Dataset
- Excercises
    - Exercise 1
    - Exercise 2
    - Exercise 3
    - Exercise 4
    - Exercise 5
    - Exercise 6
    - Exercise 7

Learning objectives

- Factors
- Data Matrics
- Data Frames
- Importing Data into R
- Working with 23andMe SNP data

Overview

The last lab we learned about data types in R and vectors, one of the most important object types in R. This session we will learn a few more object types and their importance for importing data from files

- vectors: ordered collection of numeric, character, complex and logical values.
- factors: special type vectors with grouping information of its components
- data frames: two dimensional structures with different data types
- matrices: two dimensional structures with data of same type
- arrays: multidimensional arrays of vectors (not covered today)
- lists: general form of vectors with different types of elements (not covered today)

We can think of matrices, arrays, lists and data frames as deviations from a vector. The deviations are related to the two characteristics order and homogeneity. Here are naming conventions that go with objects

- Object, row and column names should not start with a number.
- Avoid spaces in object, row and column names.
- Avoid special characters like '#' in object, row and column names.

Basic Data Objects

Vectors

Vectors are ordered collections of the same data type (numeric, character, complex, raw and logical values). Data types are also called atomic modes in R. In the last session we made a vector of numeric characters. You can assemble and combine vectors using the function "c" short for combine.

```
SNPs <- c("AA", "AA", "GG", "AG", "AG", "AA","AG", "AA", "AA", "AA", "AG")
SNPs
```

```
##  [1] "AA" "AA" "GG" "AG" "AG" "AA" "AG" "AA" "AA" "AA" "AG"
```

Factors

A Factor is a vector whose elements can take on one of a specific set of values. For example, "Sex" will usually take on only the values "M" or "F," whereas "Genotype" will generally have lots of possibilities. The set of values that the elements of a factor can take are called its levels. Factors encode categorical data.

```
SNPs_cat <- factor(SNPs)
SNPs_cat
```

```
##  [1] AA AA GG AG AG AA AG AA AA AA AG
## Levels: AA AG GG
```

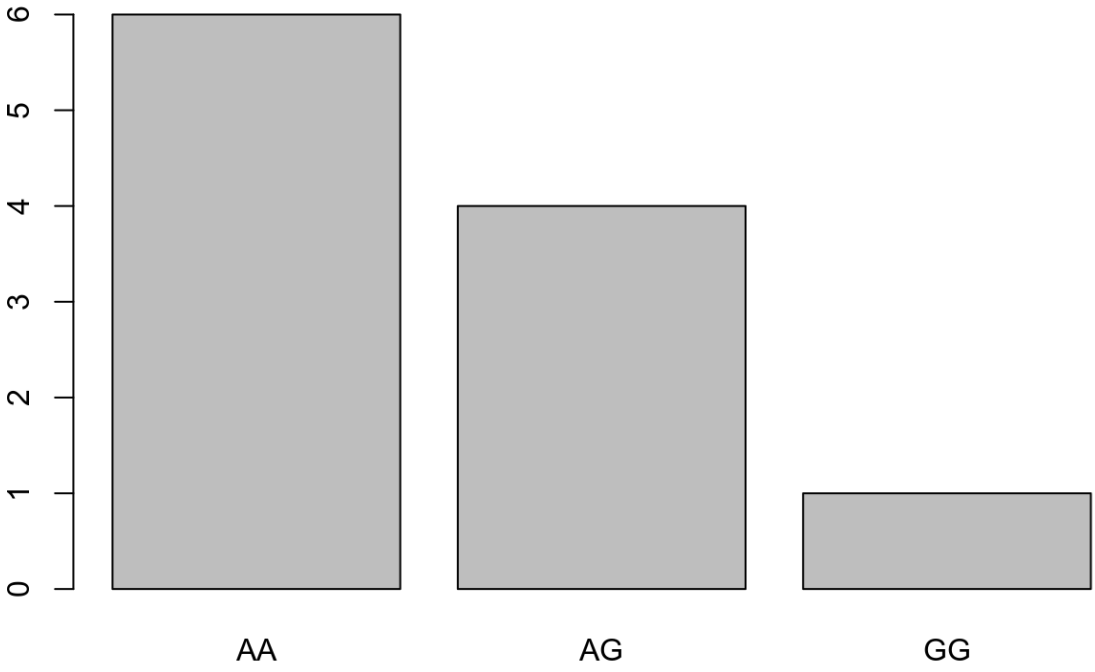
We can use the TABLE function to make a table of the factors

```
table(SNPs_cat)
```

```
## SNPs_cat
## AA AG GG
##  6  4  1
```

and make a plot of SNPs

```
plot(SNPs_cat)
```



If we had tried to make a plot of a vector of characters an error message would be returned.

Factors are actually stored as a list of integers, referring to the element number of the factor levels. In the following example, there are 6 levels (AA AC AG CC CT GG), which are represented as characters, and the numeric values of the factor comprise the integers 1-6, referring to the elements of the vector of levels. We can see these integers using

```
as.numeric(SNPs_cat)
```

```
##  [1] 1 1 3 2 2 1 2 1 1 1 2
```

Matrices

Matrices are two dimensional structures with data of same type and are often thought of as a numeric array of rows and columns. One of the easiest ways to create a matrix is to combine vectors of equal length using cbind(), meaning “column bind” OR rbind() to combine objects as rows

```
Day1 <- c(2,4,6,8)
Day2 <- c(3,6,9,12)
Day3 <- c(1,4,9,16)
A <- cbind(Day1,Day2,Day3)
A
```

```
##      Day1 Day2 Day3
## [1,]    2    3    1
## [2,]    4    6    4
## [3,]    6    9    9
## [4,]    8   12   16
```

```
Day1 <- c(2,4,6,8)
Day2 <- c(3,6,9,12)
Day3 <- c(1,4,9,16)
B <- rbind(Day1,Day2,Day3)
B
```

```
##      [,1] [,2] [,3] [,4]
## Day1    2    4    6    8
## Day2    3    6    9   12
## Day3    1    4    9   16
```

To add a row to a matrix we can use rbind or cbind with the vector representing the row and the matrix

```
Day4 <- c(5,10,11,20)
C <- rbind(B,Day4)
C
```

```
##      [,1] [,2] [,3] [,4]
## Day1    2    4    6    8
## Day2    3    6    9   12
## Day3    1    4    9   16
## Day4    5   10   11   20
```

As with vectors we can do calculations on the matix

```
A * 10
```

```
##      Day1 Day2 Day3
## [1,]   20   30   10
## [2,]   40   60   40
## [3,]   60   90   90
## [4,]   80  120  160
```

Matrices are stored in a 1 dimensional structure, so you can still access their elements with a single subscript:

```
A[1]
```

```
## [1] 2
```

```
A[12]
```

```
## [1] 16
```

They can also be accessed using row and column positions:

```
A[1,1]
```

```
## Day1
##    2
```

```
A[2,3]
```

```
## Day3
##    4
```

If we want to extract a submatrix consisting of the first and third column, leave the first positin the specifies the row blank.

```
A[ ,c(1,3)]
```

```
##      Day1 Day3
## [1,]    2    1
## [2,]    4    4
## [3,]    6    9
## [4,]    8   16
```

Extract a submatrix consisting of the second and fourth row. Leave the column (2nd) position blank.

```
A[c(2,4), ]
```

```
##      Day1 Day2 Day3
## [1,]    4    6    4
## [2,]    8   12   16
```

A matrix can be transposed using the function “t”"

```
t(A)
```

```
##      [,1] [,2] [,3] [,4]
## Day1    2    4    6    8
## Day2    3    6    9   12
## Day3    1    4    9   16
```

Data frames

Data frames are two dimensional structures with different data types. The data.frame() function can combine vectors and/or factors into a single data frame.

```
Gene1 <- c(2,4,6,8)
Gene2 <- c(3,6,9,12)
Gene3 <- c(1,4,9,16)
Gene <- c("Day 1", "Day 2","Day 3", "Day 4")
RNAseq <- data.frame(Gene1, Gene2, Gene3, row.names = Gene)
RNAseq
```

```
##       Gene1 Gene2 Gene3
## Day 1     2     3     1
## Day 2     4     6     4
## Day 3     6     9     9
## Day 4     8    12    16
```

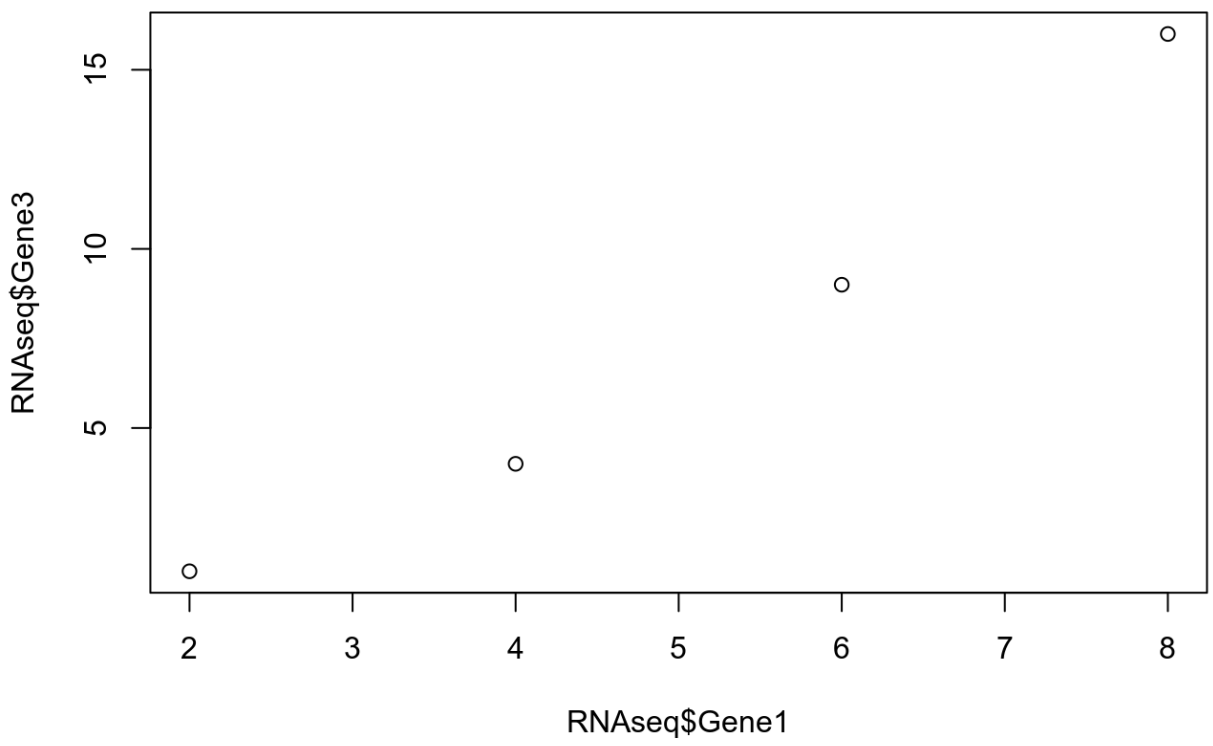
To work with data in dataframes we use the names of the data frame and the name of the vector (column). For example

```
RNAseq$Gene3
```

```
## [1]  1  4  9 16
```

To make an x-y plot of the data

```
plot(RNAseq$Gene1,RNAseq$Gene3)
```



Note what happens if we plot Day vs Gene3

```
plot(RNAseq$Day,RNAseq$Gene3)
```

You will get a message “Error in xy.coords(x, y, xlabel, ylabel, log) : ‘x’ and ‘y’ lengths differ”

If you want to plot Day vs Gene3 then you have to add Day as a column with numeric values and not as the row names

There are many different ways of adding columns to a data frame.

```
RNAseq$Gene4 <- c(5, 10, 15, 20)
RNAseq
```

```
##       Gene1 Gene2 Gene3 Gene4
## Day 1     2     3     1     5
## Day 2     4     6     4    10
## Day 3     6     9     9    15
## Day 4     8    12    16    20
```

and

```
RNAseq[,"Gene5"] <- c(1, 2, 3, 3)
RNAseq
```

```
##       Gene1 Gene2 Gene3 Gene4 Gene5
## Day 1     2     3     1     5     1
## Day 2     4     6     4    10     2
## Day 3     6     9     9    15     3
## Day 4     8    12    16    20     3
```

To add a row use rbind

```
RNAseq["Day 4",] <- rbind(10, 14, 20, 22, 3)
```

Checking on object types

Sometimes it is confusing as to what type of type of data is in a object. You can use the str() function

```
x = 1
str(x)
```

```
##  num 1
```

Object with a number(num)

```
a = "ATGCCCTGA"
str(a)
```

```
##  chr "ATGCCCTGA"
```

Object with a character

```
str(SNPs)
```

```
##  chr [1:11] "AA" "AA" "GG" "AG" "AG" "AA" "AG" "AA" "AA" "AA" "AG"
```

This reveals a vector of characters (chr).

```
SNPs <- c("AA", "AA", "GG", "AG", "AG", "AA","AG", "AA", "AA", "AA", "AG")
str(SNPs_cat)
```

```
##  Factor w/ 3 levels "AA","AG","GG": 1 1 3 2 2 1 2 1 1 1 ...
```

This reveals a factor with 3 levels

```
Day1 <- c(2,4,6,8)
Day2 <- c(3,6,9,12)
Day3 <- c(1,4,9,16)
B <- rbind(Day1,Day2,Day3)
str (B)
```

```
##  num [1:3, 1:4] 2 3 1 4 6 4 6 9 9 8 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:3] "Day1" "Day2" "Day3"
##   ..$ : NULL
```

This shows a matrix of numbers (num) with the columns having character names (chr)

```
Gene1 <- c(2,4,6,8)
Gene2 <- c(3,6,9,12)
Gene3 <- c(1,4,9,16)
Gene <- c("Day 1", "Day 2","Day 3", "Day 4")
RNAseq <- data.frame(Gene1, Gene2, Gene3, row.names = Gene)
str(RNAseq)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ Gene1: num  2 4 6 8
##  $ Gene2: num  3 6 9 12
##  $ Gene3: num  1 4 9 16
```

A dataframe of numbers (num)

The Environment window in the top right corner of RStudio will also display the value types.

Importing data

read.table()

Most of our class data will be in files that we import into R. A common way to store data is in CSV (comman-separated values) and tab separated file. In these file formats the columns are separated by either a comman or tab.

The function read.table() is used to load these files into R. This function reads a file in table format and creates a data frame from it. By default, read.table uses '#' as a comment character, and if this is encountered (except in quoted strings) the rest of the line is ignored. Lines containing only white space and a comment are treated as blank lines. The important fields in read.table are the file name, header and separator.

```
read.table("file.csv", header = TRUE, sep = ",")
read.table("file.txt", header = TRUE, sep = "\t")
```

Often the header line has entries only for the columns and not for the row labels, so one field shorter than the remaining lines. (If R sees this, it sets header = TRUE.) If the first column is row names specific this using row.names = 1

```
read.table("file.dat", header = TRUE, , sep = "\t", row.names = 1)
```

From Excel

One of the best ways to read an Excel file is to export it to a comma delimited file and import it using the read.table() method above. Alternatively you can use the xlsx package to access Excel files. You

will first need to install the xlsx package. This may not not work on some operating systems (e.g. on my laptop running Ubuntu), but if it works for you, it is ok with me if you use this method.

The first row should contain variable/column names.

```
# read in the first worksheet from the workbook myexcel.xlsx
# first row contains variable names
library(xlsx)
mydata <- read.xlsx("c:/myexcel.xlsx", 1)

# read in the worksheet named mysheet
mydata <- read.xlsx("c:/myexcel.xlsx", sheetName = "mysheet")
```

Loading a truncuated 23andMe file

Down load the 23andMe truncated file (23andMe_example_cat25.txt) from the Moodle site. Open it up with your text editor (Notepad++, Text Wrangler, etc). The file contains important background information that is demarked with #. There are 4 columns for the rsid number, chromosome number, chromosome position and the genotype.

The rsid number is used by researchers to refer to specific SNPs. It stands for Reference SNP cluster ID. When researchers identify a SNP, they send the report, which includes the sequence immediately surrounding the SNP, to the dbSNP database - http://www.ncbi.nlm.nih.gov/SNP/ (http://www.ncbi.nlm.nih.gov/SNP/) at the National Center for Biotechnology Information. Genome-wide association studies linking SNPs to traits or conditions usually report their results by rsid. The rsid numbers for SNPs in Health and Traits articles can be found in the technical report section.

```
SNP_table <- read.table("23andMe_example_cat25.txt", header = TRUE, sep = "\t")
SNP_table
```

```
##          rsid chromosome position genotype
## 1   rs4477212          1    82154       AA
## 2   rs3094315          1   752566       AA
## 3   rs3131972          1   752721       GG
## 4  rs12124819          1   776546       AG
## 5  rs11240777          1   798959       AG
## 6   rs6681049          1   800007       CC
## 7   rs4970383          1   838555       AC
## 8   rs4475691          1   846808       CT
## 9   rs7537756          1   854250       AG
## 10 rs13302982          1   861808       GG
```

Getting Information on a Dataset

List the variables in the data set

```
names(SNP_table)
```

```
## [1] "rsid"       "chromosome" "position"   "genotype"
```

List the structure of the data set

```
str(SNP_table)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ rsid      : Factor w/ 10 levels "rs11240777","rs12124819",..: 7 4 5 2 1 9
8 6 10 3
##  $ chromosome: int  1 1 1 1 1 1 1 1 1 1
##  $ position  : int  82154 752566 752721 776546 798959 800007 838555 846808 8
54250 861808
##  $ genotype  : Factor w/ 6 levels "AA","AC","AG",..: 1 1 6 3 3 4 2 5 3 6
```

List levels of factor genotype

```
levels(SNP_table$genotype)
```

```
## [1] "AA" "AC" "AG" "CC" "CT" "GG"
```

Dimensions of an object

```
dim(SNP_table)
```

```
## [1] 10  4
```

Class of an object (numeric, matrix, data frame, etc)

```
class(SNP_table)
```

```
## [1] "data.frame"
```

Print mydata - warming do not try to do this on large data sets such as the full 23andME file

```
SNP_table
```

```
##          rsid chromosome position genotype
## 1   rs4477212          1    82154       AA
## 2   rs3094315          1   752566       AA
## 3   rs3131972          1   752721       GG
## 4  rs12124819          1   776546       AG
## 5  rs11240777          1   798959       AG
## 6   rs6681049          1   800007       CC
## 7   rs4970383          1   838555       AC
## 8   rs4475691          1   846808       CT
## 9   rs7537756          1   854250       AG
## 10 rs13302982          1   861808       GG
```

Print first 5 rows of the data set

```
head(SNP_table, n=10)
```

```
##          rsid chromosome position genotype
## 1   rs4477212          1    82154       AA
## 2   rs3094315          1   752566       AA
## 3   rs3131972          1   752721       GG
## 4  rs12124819          1   776546       AG
## 5  rs11240777          1   798959       AG
## 6   rs6681049          1   800007       CC
## 7   rs4970383          1   838555       AC
## 8   rs4475691          1   846808       CT
## 9   rs7537756          1   854250       AG
## 10 rs13302982          1   861808       GG
```

Print last 5 rows of the data set

```
tail(SNP_table, n=5)
```

```
##          rsid chromosome position genotype
## 6   rs6681049          1   800007       CC
## 7   rs4970383          1   838555       AC
## 8   rs4475691          1   846808       CT
## 9   rs7537756          1   854250       AG
## 10 rs13302982          1   861808       GG
```

The object type of each column can be determine during the import process. For more information of modifying the data and object type

```
help(read.table)
```

For example for some purposes when we have loaded the complete SNP file it would be nice to have the chromosomes considered as factors rather than integers.

```
SNP_table$chromosome <- as.factor(SNP_table$chromosome)
str(SNP_table)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ rsid      : Factor w/ 10 levels "rs11240777","rs12124819",..: 7 4 5 2 1 9
8 6 10 3
##  $ chromosome: Factor w/ 1 level "1": 1 1 1 1 1 1 1 1 1 1
##  $ position  : int  82154 752566 752721 776546 798959 800007 838555 846808 8
54250 861808
##  $ genotype  : Factor w/ 6 levels "AA","AC","AG",..: 1 1 6 3 3 4 2 5 3 6
```

and back

```
SNP_table$chromosome <- as.integer(SNP_table$chromosome)
str(SNP_table)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ rsid      : Factor w/ 10 levels "rs11240777","rs12124819",..: 7 4 5 2 1 9
8 6 10 3
##  $ chromosome: int  1 1 1 1 1 1 1 1 1 1
##  $ position  : int  82154 752566 752721 776546 798959 800007 838555 846808 8
54250 861808
##  $ genotype  : Factor w/ 6 levels "AA","AC","AG",..: 1 1 6 3 3 4 2 5 3 6
```

Select a subset of the data with the genotype AG and make a summary table

```
SNP_table_AG <- subset(SNP_table, genotype == 'AG')
SNP_table_AG
```

```
##         rsid chromosome position genotype
## 4 rs12124819          1   776546       AG
## 5 rs11240777          1   798959       AG
## 9  rs7537756          1   854250       AG
```

```
table(SNP_table_AG$chromosome)
```

```
##
## 1
## 3
```

Select a subset of a chromosome by position

```
subset(SNP_table, position > 700000 & position < 800000)
```

```
##         rsid chromosome position genotype
## 2  rs3094315          1   752566       AA
## 3  rs3131972          1   752721       GG
## 4 rs12124819          1   776546       AG
## 5 rs11240777          1   798959       AG
```

Excercises

There are many ways to do the below excercises, but try to do them simply by using the commands in the above examples. In later sessions we will focus on graphing. For now do not worry that your graphs do not have titles, x or y labels, units and other things. Please submit a .R file with comments denoting the examples and exercises.

Exercise 1

Add, subtract, multipy and divide the following two vectors (1,3,6,9,12) and (1,0,1,0,1)

Exercise 2

Create 3 different vectors from (0,1,2,3), (“aa”,“bb”,“cc”,“dd”) and (“aa”,1,“bb”,2). Use str() to determine what data types each vector holds.

Exercise 3

Create a matrix of the data: genotype 1 (“AA”, “AA”, “AG”, “GG”, “GG”), genotype 2 (“AA”, “AA”, “GG”, “GG”, “GG”). Display the matrix. Use the table function (as in the above examples) to show the total number of each genotype.

Exercise 4

Create a dataframe of the following experiment in samples were collected every 2 minutes starting at t = 0. treatment 1 (0,1,2,3,4), treatment 2 (0,2,4,6,8), treatment 3 (0,3,6,9,12). Display the dataframe. Plot treatment 3 vs. time (you will need to load time as a column rather than a row name)

Exercise 5

Following the example above with the truncated file use read.table to import the full SNP file 23andME_complete.txt. (This is a large file and may take several minutes to load into R)

What object type is chromosome? Why is it different from the above SNP_table example with the truncated file?

Exercise 6

Make a table with the total number of each genotype. There may be unsual genotypes. 23andMe reports a very small number of deletions and insertions coded as D DD DI I II. The double dash – represents an uncertain (not reported) call at this position.

Exercise 7

Determine which chromosome(s) the single letter genotype A is found on (e.g which chromosomes have only one copy of DNA)? Hint: Use subset() to make a table with just the genotype A.