Lab 3: Data manipulation

Joaquin Cavieres

1. Data Preparation

1.1. Data types

First to visualize your data, you have to get it into R. We will start using simulated data and manipulate them in some way.

Variables in R can be of different types. For example, we need to distinguish numbers from character strings and tables from simple lists of numbers. The function class helps us determine what type of object we have:

```
u <- 5
class(u)
```

```
## [1] "numeric"
```

A charatcter could be:

```
name <- "John"
class(name)
```

```
## [1] "character"
```

1.2. Data frames

Until now, the variables we have defined are just one number or a character, but this is not very useful for storing data. The most common way of storing a dataset in R is in a `data.frame`. So, what is a `data.frame`? A `data.frame`? could be described as a table with rows representing observations and the different variables reported for each observation defining the columns.

```
# install.packages("dslabs")
library(dslabs)
data(murders)
class(murders)
```

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

Rigth now we can use the function `str` to see what content the data:

```
str(murders)
```

```
## 'data.frame':    51 obs. of  5 variables:
##  $ state     : chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ abb       : chr  "AL" "AK" "AZ" "AR" ...
##  $ region    : Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...
##  $ population: num  4779736 710231 6392017 2915918 37253956 ...
##  $ total     : num  135 19 232 93 1257 ...
```

or, if we only want to see the firts rows of the data, we can use the function `head`

```
head(murders)
```

```
##        state abb region population total
## 1    Alabama  AL  South    4779736   135
## 2     Alaska  AK   West     710231    19
## 3    Arizona  AZ   West    6392017   232
## 4   Arkansas  AR  South    2915918    93
## 5 California  CA   West   37253956  1257
## 6   Colorado  CO   West    5029196    65
```

1.3. The accessor: $

For example, what happend if we want to acces to the variable "population" of the data "murders", how could we do it?

Well, we can do it using the symbol $.

```
murders$population
```

```
##  [1]  4779736   710231  6392017  2915918 37253956  5029196  3574097   897934
##  [9]   601723 19687653  9920000  1360301  1567582 12830632  6483802  3046355
## [17]  2853118  4339367  4533372  1328361  5773552  6547629  9883640  5303925
## [25]  2967297  5988927   989415  1826341  2700551  1316470  8791894  2059179
## [33] 19378102  9535483   672591 11536504  3751351  3831074 12702379  1052567
## [41]  4625364   814180  6346105 25145561  2763885   625741  8001024  6724540
## [49]  1852994  5686986   563626
```

To see what are the variables in the "murders" data we could use the function `names`:

```
names(murders)
```

```
## [1] "state"      "abb"        "region"     "population" "total"
```

in this way we get all the variables in the `data.frame`.

It is important to know that the order of the entries in `murders$population` preserves the order of the rows in our data set.

1.4. Vectors: numerics, characters, and logical

If we save the object `murders$population`, what do you think that is this object?

We call these types of objects vectors. A single number is technically a vector of length 1, but in general we use the term vectors to refer to objects with several entries. The function length tells you how many entries are in the vector:

```
pop <- murders$population
class(pop)
```

```
## [1] "numeric"
```

```
length(pop)
```

```
## [1] 51
```

As `pop` is a numeric vector, every element of it is a number. However, also there are "character" vectors, for example:

```
class(murders$state)
```

```
## [1] "character"
```

thus, all the entries of that type of vector are characters.

Another important type of vectors are "logical" vectors. These must be either `TRUE` or `FALSE`.

```
u <- 3 == 2
u
```

```
## [1] FALSE
```

```
class(u)
```

```
## [1] "logical"
```

Here the symbol == is a relational operator asking if 3 is equal to 2. In R, if you just use one =, you actually assign a variable, but if you use two == you test for equality.

1.5. Factors

In the "murders" dataset, we might expect the region to also be a character vector. However,

```
class(murders$region)
```

```
## [1] "factor"
```

that variable is a `factor`. Factors are useful for storing categorical data. This is more memory efficient than storing all the characters.

Suppose we want the levels of the region by the total number of murders rather than alphabetical order. If there are values associated with each level, we can use the reorder and specify a data summary to determine the order. The following code takes the sum of the total murders in each region, and reorders the factor following these sums.

```
region <- murders$region
value <- murders$total
region <- reorder(region, value, FUN = sum)
levels(region)
```

```
## [1] "Northeast"     "North Central" "West"          "South"
```

The new order is in agreement with the fact that the Northeast has the least murders and the South has the most.

1.6. Lists

Lists allow us to create any type of combination of different types of data frames. For example, you can create a list using the following function:

```
data_list <- list(name = "John",
             student_id = 50,
             grades = c(4, 4, 5, 3, 4),
             final_grade = 4)
data_list
```

```
## $name
## [1] "John"
##
## $student_id
## [1] 50
##
## $grades
## [1] 4 4 5 3 4
##
## $final_grade
## [1] 4
```

The list "data" includes a character, a number, a vector with five numbers, and a numeric value (mean).

As with data frames, you can extract the components of a list with the accessor $

```
data_list$student_id
```

```
## [1] 50
```

You could also encounter lists without variable names, for example:

```
data_list2 <- list("John", 50)
data_list2
```

```
## [[1]]
## [1] "John"
##
## [[2]]
## [1] 50
```

To acces to the variables in the list, you have to use brackets as follow:

```
data_list[[1]]
```

```
## [1] "John"
```

1.7. Matrices

Matrices are similar to data frames,that is, they have rows and columns. However, like numeric, character and logical vectors, entries in matrices have to be all the same type. For the above the data frames are more used becasue we can have characters, factors, and numbers in the same data structure.

Matrices have a major advantage over data frames wen we want to perform matrix algebra operations, a powerful type of mathematical technique. However, we will not see this part on this course.

A matrix can be defined using the `matrix` function. We also need to specify the number of rows and columns.

```
mat <- matrix(1:12, nrow = 4, ncol = 3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

To access to a matrixm we can specify the entries in a matrix using square brackets ([]). If you want the second row, third column, you use:

```
mat[2, 3]
```

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

If you want the entire second row, you leave the column spot empty:

```
mat[, 3]
```

```
## [1]  9 10 11 12
```

You can create a new matrix based on the previous, for example:

```
mat[, 2:3]
```

```
##      [,1] [,2]
## [1,]    5    9
## [2,]    6   10
## [3,]    7   11
## [4,]    8   12
```

or a subset of rows and columns:

```
mat[1:2, 2:3]
```

```
##      [,1] [,2]
## [1,]    5    9
## [2,]    6   10
```

A special transformation can be done converting our matrix to a data frame using the function `as.data.frame`:

```
as.data.frame(mat)
```

```
##   V1 V2 V3
## 1  1  5  9
## 2  2  6 10
## 3  3  7 11
## 4  4  8 12
```

Exercises

Using the "murders" dataset.

- How can we have a summary of the structure of our object (data)?
- How can we know the names of the columns on this data?
- Use the accessor $ to extract the state abbreviations and assign them to the object "**a**". What is the class of this object?
- Now use the square brackets to extract the state abbreviations and assign them to the object "**b**". Use the identical function to determine if "**a**" and "**b**" are the same.
- We saw that the region column stores a factor. With one line of code, use the function `levels` and `length` to determine the number of regions defined by this dataset.