lab 0

Lab 0: Introduction to R

- Type 'demo()' for some demos, 'help()' for on-line help, or
- 'help.start()' for an HTML browser interface to help.
- Type 'q()' to quit R.

Basic Commands

Data frame

```
x <- 17.3
x
```

```
## [1] 17.3
```

```
7*x
```

```
## [1] 121.1
```

```
remove(x) # x disappears
```

`getwd()` - Prints your working directory.

setwd("the directory you want here")

Vectors (one dimensional arrays) are made like this:

```
v <- c( 1.1, 2.2, 17.3, -23, 4, 0)
v
```

```
## [1]   1.1   2.2  17.3 -23.0   4.0   0.0
```

```
v[1]
```

```
## [1] 1.1
```

```
v[2:5]
```

```
## [1]   2.2  17.3 -23.0   4.0
```

Logical (Boolean) vectors are useful.

Note that this is a Boolean vector, and TRUE and FALSE are printed (and entered) in capitals.

Logical vectors are very useful for selecting subsets of elements of a vector.

```
v
```

```
## [1]   1.1   2.2  17.3 -23.0   4.0   0.0
```

```
boolv <- v < 3
boolv
```

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

```
# This returns a vector of all elements of v that are less than 3.
v[boolv]
```

```
## [1]   1.1   2.2 -23.0   0.0
```

We can write an expression that gives a logical vector inside the subscript brackets:

```
v
```

```
## [1]   1.1   2.2  17.3 -23.0   4.0   0.0
```

```
v[ v < 3 ]
```

```
## [1]   1.1   2.2 -23.0   0.0
```

Lists:

Lists can contain elements of different types.

Giving names to elements of their lists. The advantage of doing this is that you can refer to elements of a list by name, without having to remember what position they are:

```
mylist <- list( alf=34, bert=c(2,3,4,5), 17.3, pain="wisdom tooth")
mylist
```

```
## $alf
## [1] 34
##
## $bert
## [1] 2 3 4 5
##
## [[3]]
## [1] 17.3
##
## $pain
## [1] "wisdom tooth"
```

```
mylist$alf
```

```
## [1] 34
```

```
mylist$pain
```

```
## [1] "wisdom tooth"
```

Alternatively, you can refer to them by position:

```
mylist[2]
```

```
## $bert
## [1] 2 3 4 5
```

```
class( mylist )
```

```
## [1] "list"
```

```
class( mylist[2] )
```

```
## [1] "list"
```

We see that mylist[2] is actually a one element list. To extract an individual element from a list by its index, we use

```
mylist[[2]]
```

```
## [1] 2 3 4 5
```

```
class( mylist[[2]] )
```

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

```
class( mylist $bert )
```

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

We can remove an element from a list by:

```
mylist$bert <- NULL
mylist
```

```
## $alf
## [1] 34
##
## [[2]]
## [1] 17.3
##
## $pain
## [1] "wisdom tooth"
```

```
mylist$bert
```

```
## NULL
```

Data Frames

A data frame is a class, designed to hold data in a convenient way. It is implemented as a list of vectors, which should be of equal lengths.

Classes in R are implemented as lists.

```
### Data Frames
mydf <- data.frame(row.names= c("alf","bert", "r2d2"),
                   sex=c("M","F","R"),
                   age=c(20,21,3) )
mydf
```

```
##      sex age
## alf    M  20
## bert   F  21
## r2d2   R   3
```

We can use a logical vector (which we can compute in any way we like) to select a subset of rows.

```
mydf[c(FALSE,TRUE,TRUE), ]
```

```
##      sex age
## bert   F  21
## r2d2   R   3
```

This selects the vector of ages.

```
mydf$age
```

```
## [1] 20 21  3
```

```
class( mydf)
```

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

```
class(mydf$sex)
```

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

A character object is used to represent string values in R. We can convert character into factor:

Each string (in this case "M", "F", or "R") is encoded as a number (the numbers 1, 2, and 3), and there is a separate (invisible) array that stores the strings.

```
class(as.factor(mydf$sex))
```

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

Functions

Write your own function "area" to calculate the area of a circle with a given radius:

```
# Functions
area <- function(radius) {
  pi * radius^2
}
area(10)
```

```
## [1] 314.1593
```

The Pipe: a tidy operator for functions.

%>% is a popular new operator, increasingly used to clean big, messy data sets.

```
# The Pipe: a tidy operator for functions
library(tidyr) # %>% originally from library(magrittr)
```

```
## Warning: package 'tidyr' was built under R version 4.0.4
```

```
x <- rnorm(100, mean = 100) # # 100 random normal draws
x
```

```
##   [1] 100.12207 100.82628  99.89394  99.59631 100.74173 101.16057 101.65897
##   [8] 100.75900 101.09566  98.56440 100.21131  99.09919 101.99915  98.87499
##  [15] 100.23553  99.27103  99.83258 100.17959 100.19946 100.98613 101.08332
##  [22]  99.80345 100.88253 100.39561 100.72111 101.09945  98.85051  98.97064
##  [29] 100.07329 101.40312  99.44974  98.97942  98.78222  99.86791  99.39300
##  [36] 100.33768 100.62862 100.11383 100.64567 100.89663  99.75202  99.99010
##  [43] 100.72452  99.70293 100.17120  99.53835  99.26412  99.85895  99.14848
##  [50]  99.82973  99.45457 100.10971  99.97189 100.56118 100.11570 101.74572
##  [57]  98.51391 101.59291 100.52503 101.24490 102.42422 100.85188  98.44233
##  [64] 101.17285  97.81928 101.11966  99.21221 100.97990  99.49686 100.35234
##  [71]  99.11320 101.01440  99.72999 100.94707  99.41984  99.69289  98.93128
##  [78] 100.69749 100.33660  99.35111 100.84508 102.04413  96.45704  99.47794
##  [85]  99.37787 102.73111 101.37488  98.78182  99.79681  98.54563  99.04204
##  [92]  98.59119  98.90442 101.39445 101.19039  99.44872 100.70815  99.08084
##  [99]  99.61294  99.54147
```

```
x %>% range
```

```
## [1]  96.45704 102.73111
```

It is often used when several functions need to be applied in a row (where scale is a function to standardize a data).

```
x %>% scale %>% abs %>% range %>% round(1)
```

```
## [1] 0.0 3.5
```

```
y <- x %>% scale %>% abs %>% range %>% round(1)
y
```

```
## [1] 0.0 3.5
```

Loading data

```
Auto <- read.table("Auto.data")
dim(Auto)
```

```
## [1] 398   9
```

```
#fix(Auto)
head(Auto, 10)
```

```
##      V1        V2           V3         V4     V5           V6   V7     V8
## 1   mpg cylinders displacement horsepower weight acceleration year origin
## 2  18.0         8        307.0      130.0  3504.         12.0   70      1
## 3  15.0         8        350.0      165.0  3693.         11.5   70      1
## 4  18.0         8        318.0      150.0  3436.         11.0   70      1
## 5  16.0         8        304.0      150.0  3433.         12.0   70      1
## 6  17.0         8        302.0      140.0  3449.         10.5   70      1
## 7  15.0         8        429.0      198.0  4341.         10.0   70      1
## 8  14.0         8        454.0      220.0  4354.          9.0   70      1
## 9  14.0         8        440.0      215.0  4312.          8.5   70      1
## 10 14.0         8        455.0      225.0  4425.         10.0   70      1
##                           V9
## 1                       name
## 2  chevrolet chevelle malibu
## 3          buick skylark 320
## 4         plymouth satellite
## 5              amc rebel sst
## 6                ford torino
## 7           ford galaxie 500
## 8           chevrolet impala
## 9          plymouth fury iii
## 10          pontiac catalina
```

```
#Auto<-read.table("Auto.data", sep=",",header=T, na.strings="?", quote="")
Auto <- read.table("Auto.data", header=T, na.strings="?")
#fix(Auto)
head(Auto, 10)
```

```
##    mpg cylinders displacement horsepower weight acceleration year origin
## 1   18         8          307        130   3504         12.0   70      1
## 2   15         8          350        165   3693         11.5   70      1
## 3   18         8          318        150   3436         11.0   70      1
## 4   16         8          304        150   3433         12.0   70      1
## 5   17         8          302        140   3449         10.5   70      1
## 6   15         8          429        198   4341         10.0   70      1
## 7   14         8          454        220   4354          9.0   70      1
## 8   14         8          440        215   4312          8.5   70      1
## 9   14         8          455        225   4425         10.0   70      1
## 10  15         8          390        190   3850          8.5   70      1
##                         name
## 1  chevrolet chevelle malibu
## 2          buick skylark 320
## 3         plymouth satellite
## 4              amc rebel sst
## 5                ford torino
## 6           ford galaxie 500
## 7           chevrolet impala
## 8          plymouth fury iii
## 9           pontiac catalina
## 10        amc ambassador dpl
```

```
Auto<-na.omit(Auto)
dim(Auto)
```

```
## [1] 392   9
```

Once the data are loaded correctly, we can use names() to check the variable names.

```
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
## [6] "acceleration" "year"         "origin"       "name"
```

Writing scripts in R

```
### Writing scripts in R
if (2 > 0) {
  i <- 2
  print (i)
}
```

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

If the name of this file is `script.R`, you can run all these commands in one go as: source("script.R") Files used in this way are called scripts.

Saving plots in R

It is possible to save the graph using R codes as followings:

```
## Saving plots in R
jpeg("plot1.jpg")
cars<-c(1, 3, 6, 4, 9)
plot(cars, type="o", col="blue")
title(main="Autos", col.main="red", font.main=4)
dev.off()
```

```
## png
##   2
```