Statistical Programming
Week 2

THE UNIVERSITY *of* EDINBURGH

Gordon Ross

School of Mathematics and Maxwell Institute

Data frames

THE UNIVERSITY
of EDINBURGH

- `Data frames` are the de facto data structure for most tabular data, and what we use for statistics and plotting.
- A data frame is a collection of vectors of identical lengths. Each vector represents a column, and each vector can be of a different data type (e.g., characters, integers, factors). The `str()` function is useful to inspect the data types of the columns.
- A data frame can be created by hand, but most commonly they are generated by the functions `read.csv()` or `read.table()`, that is, when importing spreadsheets from your hard drive (or the web).

Data frames

THE UNIVERSITY *of* EDINBURGH

- By default, when building or importing a data frame, the columns that contain characters (i.e., text) are coerced (=converted) into `factors`. Depending on what you want to do with the data, you may want to keep these columns as character. To do so, `read.csv()` and `read.table()` have an argument called `stringsAsFactors` which can be set to `FALSE`:

```
{r}
some_data <- read.csv("path/to/file",
                      stringsAsFactors=FALSE)
```

Inspecting Data frames

THE UNIVERSITY *of* EDINBURGH

Size

- `dim()`: returns a vector with the number of rows in the
- `nrow()/ncol()`: returns the number of rows/columns

Content

- `head()/tail()`: shows the first/last six rows

Names

- `names()`: returns the column names
- `rownames()`: returns the row names

Inspecting Data frames

- You can also create a data frame manually with the function `data.frame`. This function can also take the argument `stringsAsFactors` . Compare the output of these examples, and compare the difference between when the data are being read as character, and when they are being read as factor.

```{r}
df <- data.frame(
animal=c("dog", "cat",
         "sea cucumber",
         "sea urchin"),
feel=c("furry", "furry",
       "squishy", "spiny"),
weight=c(45, 8, 1.1, 0.8)
)
```

Inspecting data frames

THE UNIVERSITY *of* EDINBURGH

```
{r}
> df
'data.frame': 4 obs. of  3 variables:
$ animal: Factor w/ 4 levels "cat","dog","sea cucumber",.
          : 2 1 3 4
$ feel  : Factor w/ 3 levels "furry","spiny",..: 1 1 3 2
$ weight: num  45 8 1.1 0.8


{r}
> str(df)
        animal    feel weight
1          dog   furry   45.0
2          cat   furry    8.0
3 sea cucumber squishy    1.1
4   sea urchin   spiny    0.8
```

apply(),mapply() and efficient loops

THE UNIVERSITY of EDINBURGH

Avoiding for loops is important for speeding up code.
This can be conveniently performed in many cases by correctly implementing the function apply()

```{r}
    muX <- NULL
    for(i in 1:dim(X)[2]){
      muX[i] <- mean(X[,i])
    }
```

Operation can be performed much faster using the apply function

```{r}
apply(X,2,mean)                    ## compute column means
apply(X,1,mean)                    ## compute rowmeans
apply(X,1,function_name,args)      ## apply a function to
apply(X,1,quantile,0.025)          ## all row vectors
apply(X,1,quantile,0.975)
```

Loops

THE UNIVERSITY of EDINBURGH

Loops are extremely powerful when repetitive/recursive tasks are required.

```
{r}
## for loop: basic syntax
for (variable in sequence){
    ## Do something
}

## while loop: basic syntax
while (condition){
    ## Do something
}
```

Loops

THE UNIVERSITY *of* EDINBURGH

Example: compute factorial of a number

```{r}
factorial <- function(i){
    val <- 1
    for(j in 2:i){
        val <- val*j
    }
    return(val)
}
```

Example: simulate n random walks each of length p

```{r}
  n <- 1000
  p <- 100
  Y <- matrix(nrow=n, ncol=p)
  for(i in 1:n) {
    epsilon  <- rnorm(p)
```

Functions

THE UNIVERSITY
of EDINBURGH

```
{r}
    function_name <- function(arg1,arg2,...){
    <body> (logical statements)
    <body> (expression evaluation)
    <body> (argument manipulation)
    <body> (data handling)
    <body> (statistical functions: mean, var, sd)
    }
```

```{r}
 hypotenuse <- function(side1,side2)
 {
     side3 <- sqrt(side1^2 + side2^2)
     return(side3)
 }

 a <- 3
 b <- 4
 c <- hypotenuse(a,b)
```

```{r}
 hypotenuse <- function(side1,side2)
 {
     side3 <- sqrt(side1^2 + side2^2)
     plot(c(side1,0),c(0,side2),axes=FALSE)
     arrows(0,0,side1,0,length=0)
     arrows(0,0,0,side2,length=0)
     arrows(0,side2,side1,0,length=0)
 }

 a <- 1
 b <- 10
 c <- hypotenuse(a,b)
```

Functions

THE UNIVERSITY
of EDINBURGH

```{r}
conf_interval <- function(X){
    mu    <- mean(X)
    quant <- qt(0.975,df=(n-1))
    low   <- c(mu - quant*sqrt(var(X)))
    upp   <- c(mu + quant*sqrt(var(X)))
    return(c(low,upp))
}
conf_interval(my_data)
```

Data semantics

THE UNIVERSITY
of EDINBURGH

A dataset is a collection of values (quantitative or qualitative). Values are organised in two ways:

EVERY VALUE BELONGS TO A VARIABLE AND AN OBSERVATION!

VARIABLE: contains all values that measure the same underlying attribute (like height, temperature, duration) across units.

OBSERVATION: contains all values measured on the same unit (like a person, or a day, or a race) across attributes.

Messy data

THE UNIVERSITY of EDINBURGH

- Most statistical data come in tabular form;
- that is, most statistical datasets are data frames made up **rows** and **columns**.
- Unfortunately, most datasets come in **messy** form: for examplex
  - column headers are values, not names;
  - multiple variables stored in one column;
  - variables are stored in both rows and columns;
  - multiple types in one table.
- it is said that 80% of data analysis is spent on cleaning and preparing data!!

```
{r}
ID   First Name Surname Telephone Number
123  Pooja      Patel   555-861-2025, 192-122-1111
45   Zhang      San     (555) 403-1659 Ext. 53; 182-929
789  John       Doe     555-808-9633
```

Tidy data definition

THE UNIVERSITY *of* EDINBURGH

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In **tidy data**:

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

This is Codd's 3rd normal form, but with the constraints framed in statistical language, and the focus put on a single dataset rather than the many connected datasets common in relational databases. **Messy data** is any other arrangement of the data.

Tidy data


```
{r}
ID  First Name Surname Telephone Number
123 Pooja      Patel   555-861-2025
123 Pooja      Patel   192-122-1111
456 Zhang      San     182-929
456 Zhang      San     (555) 403-1659 Ext. 53
789 John       Doe     555-808-9633
```