Meeting 2: R and Ebola

Brooke Anderson

November 14, 2014

Last time, we read in our data

```
## If necessary, use setwd() to get to the right directory
ebola <- read.table("country_timeseries.csv", sep = ",",
                    header = TRUE)
ebola[1:3, 1:5]
```

```
##         Date Day Cases_Guinea Cases_Liberia Cases_SierraLeone
## 1  11/2/2014 225         1731            NA              4759
## 2 10/31/2014 222           NA          6525                NA
## 3 10/29/2014 220         1667            NA              5338
```

Today's plan

- Dataframes and vectors
- Subsetting
- Functions
- Example of a function– the plot function

Dataframes and vectors

Dataframes and vectors

A vector is a string of values:

Example 1: Start of the vector of the dates when Ebola cases were reported

```
##  [1] "11/2/2014"  "10/31/2014" "10/29/2014" "10/27/2014" "10/25/2014"
##  [6] "10/22/2014" "10/21/2014" "10/19/2014" "10/18/2014" "10/14/2014"
```

Example 2: Start of the vector of number of cases reported in Guinea

```
##  [1] 1731   NA 1667 1906   NA   NA 1553 1540   NA 1519
```

Dataframes and vectors

You can make a new vector using the concatenation vector, `c(...)`:

```
x <- c(1, 5, 7, 9, 10)
x
```

```
## [1]  1  5  7  9 10
```

```
class.names <- c("Taylor", "Maggie", "Mimi", "Brianna", "Jon")
class.names
```

```
## [1] "Taylor"  "Maggie"  "Mimi"    "Brianna" "Jon"
```

Dataframes and vectors

A dataframe is made up of a lot of vectors stuck together *(Notice how each column is a vector)*

```
##          Date Day Cases_Guinea Cases_Liberia Cases_SierraLeone
## 1   11/2/2014 225         1731            NA               4759
## 2  10/31/2014 222           NA          6525                 NA
## 3  10/29/2014 220         1667            NA               5338
## 4  10/27/2014 218         1906            NA               5235
## 5  10/25/2014 216           NA          6535                 NA
## 6  10/22/2014 214           NA            NA               3896
## 7  10/21/2014 213         1553            NA                 NA
## 8  10/19/2014 211         1540            NA               3706
## 9  10/18/2014 210           NA          4665                 NA
## 10 10/14/2014 206         1519            NA               3410
```

Dataframes and vectors

You can make a new dataframe using the function `data.frame()`:

```
class.data <- data.frame(name = class.names,
                         number = x)
class.data
```

```
##      name number
## 1  Taylor      1
## 2  Maggie      5
## 3    Mimi      7
## 4 Brianna      9
## 5     Jon     10
```

Subsetting

Subsetting

You can use indexing (`[...]`, `[..., ...]`) to subset from a vector or dataframe, like:

```
vector[locations]  ## Generic code
dataframe[row locations, column locations] ## Generic code
```

Subsetting

A vector has one dimension, so you index without a comma (i.e., in one dimension):

```
class.names[1]
```

```
## [1] "Taylor"
```

```
class.names[c(2, 3, 4)]  ## Equivalent: class.names[2:4]
```

```
## [1] "Maggie"  "Mimi"    "Brianna"
```

Subsetting

A dataframe has two dimensions (rows and columns), so you index with a column:

```
class.data[1,1]
```

```
## [1] Taylor
## Levels: Brianna Jon Maggie Mimi Taylor
```

```
class.data[1:3, 1:2]
```

```
##     name number
## 1 Taylor      1
## 2 Maggie      5
## 3   Mimi      7
```

Subsetting

To get all values in a dimension (row or column), leave that part of the index blank:

```
class.data[1, ]
```

```
##     name number
## 1 Taylor      1
```

```
class.data[ , 1]
```

```
## [1] Taylor  Maggie  Mimi    Brianna Jon
## Levels: Brianna Jon Maggie Mimi Taylor
```

Subsetting

For columns, you can use column names instead of location:

```
class.data[3:4, "number"]
```

```
## [1] 7 9
```

```
class.data[3:4, c("name", "number")]
```

```
##      name number
## 3    Mimi      7
## 4 Brianna      9
```

Subsetting

You can also pull a column (vector) from a dataframe using `$`, like:

```
dataframe$column.name  ## Generic code
```

For example, to get the column of `ebola` with cases from Guinea:

```
head(ebola$Cases_Guinea)
```

```
## [1] 1731   NA 1667 1906   NA   NA
```

Note: I've used `head` to look at just the start of the vector since the whole thing would be really long.

Now you try…

Try to get the following vectors from the dataset:

- Date
- The ten most recent counts of cases in the US
- The earliest twenty counts of deaths in Liberia
- A dataframe of the first five observations of date, cases in Mali and deaths in Mali

Hint: Try using `colnames(ebola)` *to find out the names of all the columns in* `ebola`*. Also, use* `dim(ebola)` *to find out the dimensions of the dataframe so you can get the index numbers right for the latest ten data points.*

Functions

Functions

In general, functions in R take the following structure:

```
function.name(required information, options)  ## Generic code
```

The result of the function will be output to your R session, unless you choose to save the output in an object:

```
new.object <- function.name(required information, options)  ## Generic code
```

Functions

Examples of this structure:

```
head(ebola)
head(ebola, n = 3)
ebola <- read.table("country_timeseries.csv", sep = ",",
                    header = TRUE)
```

Find out more about a function by using `?function.name`. This will take you to the help page for the function, where you can find out all the possible arguments for the function, required and optional.

Example of a function

The `plot` function

The plot function has two required arguments: the x coordinates of points in the plot, and the y coordinates of points in the plot. The generic structure is:

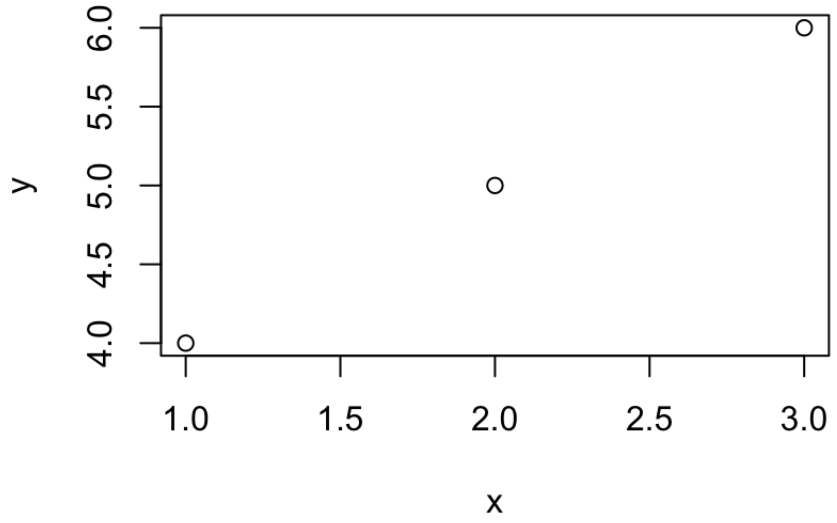
```
plot(x = x coordinates, y = y coordinate)  ## Generic code
```

As long as you put the x coordinates first and the y coordinates second, you can leave out the `x =` and `y =`:

```
plot(x coordinates, y coordinate)  ## Generic code
```

The `plot` function

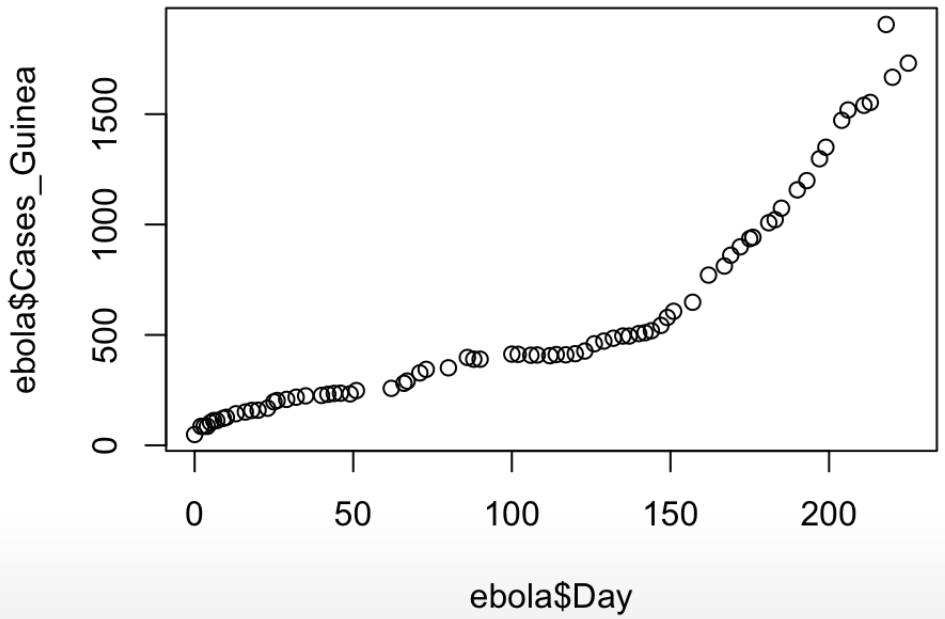
```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
plot(x, y)
```



The `plot` function

Now that you know how to pull out two vectors you want from the ebola dataset, you can plot them:

```
plot(ebola$Day, ebola$Cases_Guinea)
```



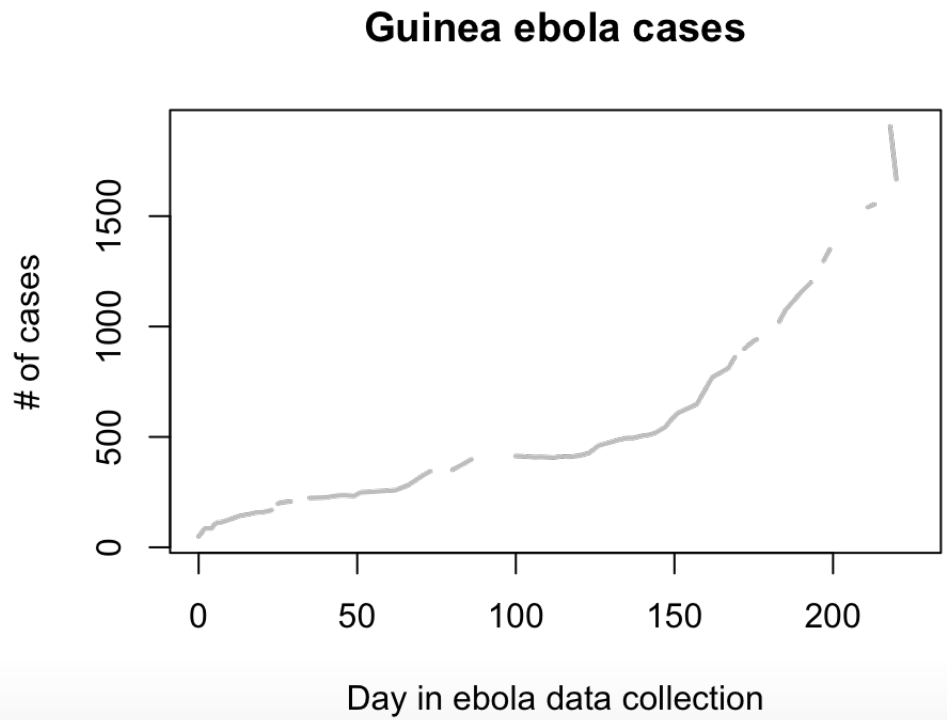
The `plot` function

The plot function also has *many* optional arguments (check `?plot`, `?plot.default`). For example,

- `type`: What do you want to plot? Points (`"p"`)? Lines (`"l"`)?
- `main`: Give a title to your plot (`"My title"`)
- `xlab`, `ylab`: Give nicer labels to your x- and y-axes (`xlab = "Day in Ebola data collection"`)
- `xlim`, `ylim`: Specify the range of your x- and y- axes (`xlim = c(0, 100)`)

The `plot` function

```
plot(ebola$Day, ebola$Cases_Guinea, main = "Guinea ebola cases",
     xlab = "Day in ebola data collection", ylab = "# of cases",
     type = "l", lwd = 2, col = "gray")
```



Now you try…

Try plotting:

- Deaths in Liberia by day
- Mortality rate in Liberia by day
- Deaths in Liberia by date

Experiment with options like `type`; `col`, `pch` and `cex` (when you're plotting points, `type = "p"`), `lwd` (when you're plotting lines, `type = "l"`), `main`, `sub`, `xlim`, and `ylim`.

Hint: Try using `colnames(ebola)` to find out the names of all the columns in `ebola`.