STAT1003

Introduction to Data Science

Solution: Lab 2

Table of Contents

Objectives

This Lab has two main objectives. The first is to introduce you to the notion of 'reproducible analyses' using the notebook mode of *RStudio*. This mode allows you to save all of your *R* code, and the resulting output, into a single file that you can consult later. Secondly, *R* has many tools for manipulating different kinds of data: numeric, character, and factors. In this Lab, you will be introduced to some simple *R* commands for understanding the structure of a dataset, for extracting subsets of data, and carrying out simple calculations.

Reproducible analyses

Up to now, you may have been using *R* within *RStudio* by simply typing commands into the console, as in the figure below, and then producing the output.

That's perfectly fine when you are quickly testing out code, or carrying out analyses that you don't necessarily want to keep or reproduce in the future. However, if you're carrying out a reasonably complex analysis that you want to document - for yourself or for others - then it makes sense to work in a way that allows you to combine your description of the analysis you're carrying out, the *R* code that you're writing, *and* the output of that code.

This section introduces your to **R Markdown**, a simple formatting syntax for writing integrated HTML, PDF, and MS Word documents from the same source file, known as an R Markdown (.Rmd) file. This worksheet, for example, has been produced using R Markdown.

In this unit, you will be doing all analyses - Labs, tests, and the project - using R Markdown. It is easy to learn, and you'll appreciate begin able to go back to something you've done a few weeks later and actually understand it.

Here are some advantages of using R Markdown (van Rij):

1. R code can be embedded in the document, so it is not necessary to keep the output and R script separately. Including the R code directly in a report provides structure to analyses.

2. The report text is written as normal text, so no knowledge of HTML coding is required.

3. The output is an HTML file that includes pictures, code blocks, R output, and text. No additional files are needed, everything is incorporated in the HTML file. If you want a publication-quality document, you can produce an MS Word document or a PDF file.

4. Integrated reports and documents enhance collaboration: It is easier to comment on an analysis when the R code, the R output, and the plots are available in the report.

The instructions below provide only a skeleton introduction. We will be learning about other aspects during this and subsequent Labs. For video tutorials on Rmarkdown, see the RStudio website; you can also find R Markdown cheatsheets there, or from the help menu (`Help -> Cheatsheets`). For a quick reference that will pop up in an RStudio window, see `Help -> Markdown Quick Reference`.

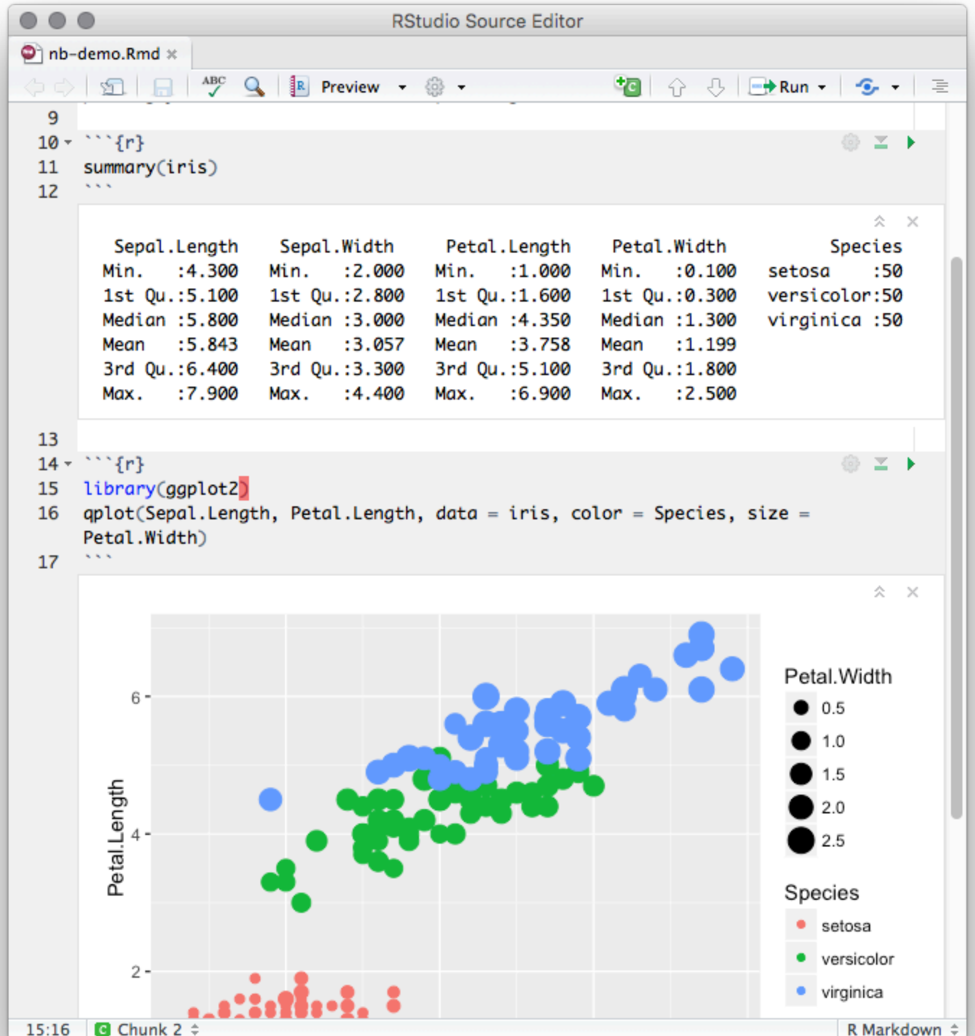
Starting a new R Markdown file (R Notebook mode)

We'll be using a special mode known as an R notebook, which is an interactive version of an R Markdown file. In most instances, you will be starting with a pre-existing file containing Lab questions, but you can create a new notebook in RStudio with the menu command `File -> New File -> R Notebook`.

Once the file is open, you can start typing text below the header

Inserting R code in 'chunks'

Notebook chunks can be inserted quickly using the keyboard shortcut `Ctrl+Alt+I`, or via the `Add Chunk` command in the editor toolbar. Once you have a blank chunk, you can type one or several R commands, and then execute them by clicking on the green triangle (`Run Chunk`), or via the keyboard shortcut `Ctrl+Shift+Enter`. The output will appear below the command. R code that produces a figure is entered and executed in exactly the same way. See below.



Chunk input and output with figure

There are many ways of working using an R Notebook, but if you execute a series of chunks sequentially without error, you can then go on to produce, for example, an MS Word document:first, run all the chunks using the `Run All` command from the menu, and then 'knit' it to a Word document using the menu.

There is much more to be said, of course, and we will learn other aspects of R Markdown throughout the semester.

Gapminder data

The dataset we'll use is a subset of the demographic, health, employment, and economic data available for many countries on the Gapminder website. It is contained in the *R* package `gapminder`, that you *may* be able to install on the PC you're using in the computer lab. You can do so from either the *RStudio* menu (`Tools -> Install Packages`), or by invoking the following command in the console:

```
# Do not write this in a chunk! You don't want to install a package every
time
# you process an R Markdown file!
install.packages("gapminder", repos = "https://cran.curtin.edu.au")
```

Then, to load this package, invoke the command

```
library(gapminder)
```

Exercises

1. How do you find help on *R* commands?
2. How do you find help on *Rmarkdown*?

So that you can start practising *Rmarkdown* now, please structure your Lab document using appropriate *Rmarkdown* commands.

Getting to know the data

There is, in fact, quite a sophisticated set of packages for manipulating (or 'wrangling') data known as the tidyverse, but we'll be using a much simpler set of built-in commands in *R*. Keep in mind that there are many ways to get to the same result in *R*!

Imagine that you have been asked to explore the impact of demographic and economic variables over time on life expectancy in many different countries. One of the first activities of a data scientist is to get to know the data *before* trying to construct any statistical models. The questions you ask in order to get to know the data are specific to the dataset itself, but finding out basic characteristics of the dataset will be common to all analyses: how big, how many variables, which variables, and so on.

Furthermore, one question, or one plot, leads to another

Exercises

Keep in mind that there are several ways of doing the same thing in *R*. We discussed some of the alternatives during the Lab.

1. How big is the dataset?

```
# In RStudio, you can have a look at the 'Environment' tab, which will give
you
# information about objects in the workspace, but you could also use the
```

```r
# following function:

dim(gapminder)  # gives number of rows and columns of a data frame

[1] 1704    6
```

2. How many variables are in the dataset? What are they?

```r
# There are six, as we saw above. The function 'colnames' will give us the
# variable names:

colnames(gapminder)

[1] "country"   "continent" "year"      "lifeExp"   "pop"       "gdpPercap"
```

3. How would you examine the first and last few rows of the dataset?

```r
# You could look a them using the viewer from the 'Environment' tab, or you
could
# try this:

head(gapminder)

# A tibble: 6 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.

tail(gapminder)

# A tibble: 6 x 6
  country  continent  year lifeExp      pop gdpPercap
  <fct>    <fct>     <int>   <dbl>    <int>     <dbl>
1 Zimbabwe Africa     1982    60.4  7636524      789.
2 Zimbabwe Africa     1987    62.4  9216418      706.
3 Zimbabwe Africa     1992    60.4 10704340      693.
4 Zimbabwe Africa     1997    46.8 11404948      792.
5 Zimbabwe Africa     2002    40.0 11926563      672.
6 Zimbabwe Africa     2007    43.5 12311143      470.
```

4. What is the class of each of the variables? In other words, what types of variables are they?

```r
# The function 'str` (short for structure) is very useful! It'll tell you
# variable types: numeric, either continuous or integer, categorical
variables,
# or factors, etc.
```

```r
str(gapminder)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   1704 obs. of  6 variables:
 $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3
...
 $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
 $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
 $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372
12881816 13867957 16317921 22227415 ...
 $ gdpPercap: num  779 821 853 836 740 ...
```

5. How many countries are represented?

```r
# From above, there are 142 countries. Or else we could work out the number
of
# unique elements of the vector gapminder$country:

length(unique(gapminder$country))
```

```
[1] 142
```

6. How many continents are represented?

```r
# Same as above

length(unique(gapminder$continent))
```

```
[1] 5
```

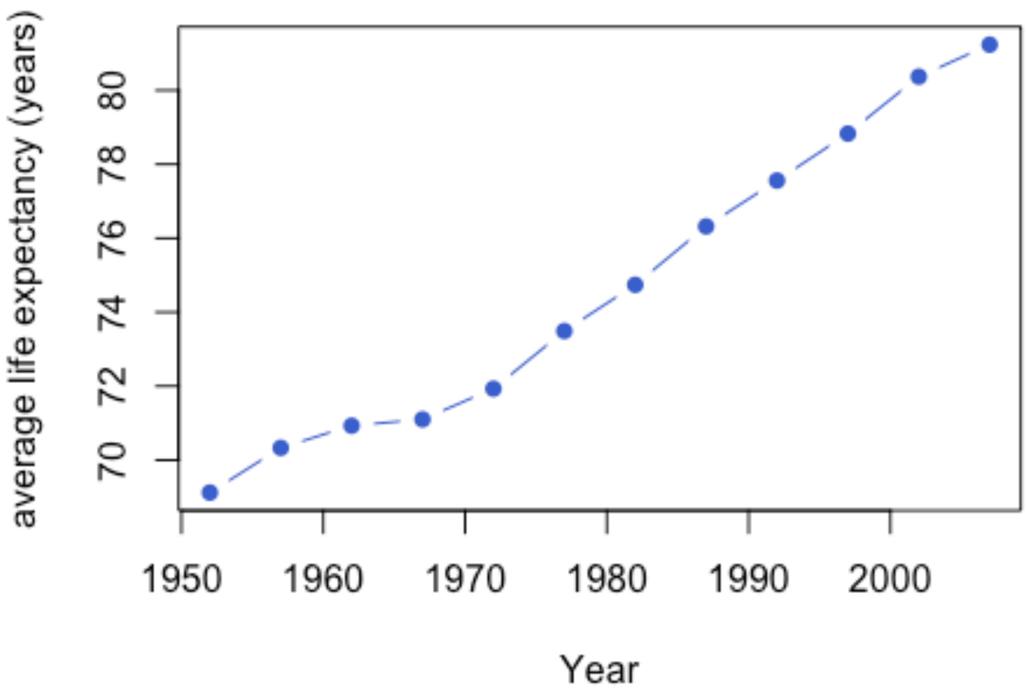
7. Create a new dataset containing only the data for Australia.

```r
# It is essential to know how to subset a larger data structure. The subset
# command is very useful!  Note that the value of the argument 'subset' has
to be
# a logical vector

Oz <- subset(gapminder, subset = (country == "Australia"))
Oz
```

```
# A tibble: 12 x 6
   country   continent  year lifeExp      pop gdpPercap
   <fct>     <fct>     <int>   <dbl>    <int>     <dbl>
 1 Australia Oceania    1952    69.1  8691212    10040.
 2 Australia Oceania    1957    70.3  9712569    10950.
 3 Australia Oceania    1962    70.9 10794968    12217.
 4 Australia Oceania    1967    71.1 11872264    14526.
 5 Australia Oceania    1972    71.9 13177000    16789.
 6 Australia Oceania    1977    73.5 14074100    18334.
 7 Australia Oceania    1982    74.7 15184200    19477.
 8 Australia Oceania    1987    76.3 16257249    21889.
 9 Australia Oceania    1992    77.6 17481977    23425.
```

```
10 Australia Oceania    1997    78.8 18565243    26998.
11 Australia Oceania    2002    80.4 19546792    30688.
12 Australia Oceania    2007    81.2 20434176    34435.
```

```
# For Australia, here's a plot of the change in average life expectancy over
# time. Note the syntax of the plot statement, and some of the additional
# arguments.

plot(lifeExp ~ year, data = Oz, xlab = "Year", ylab = "average life
expectancy (years)",
    pch = 16, type = "b", col = "royalblue3")
```



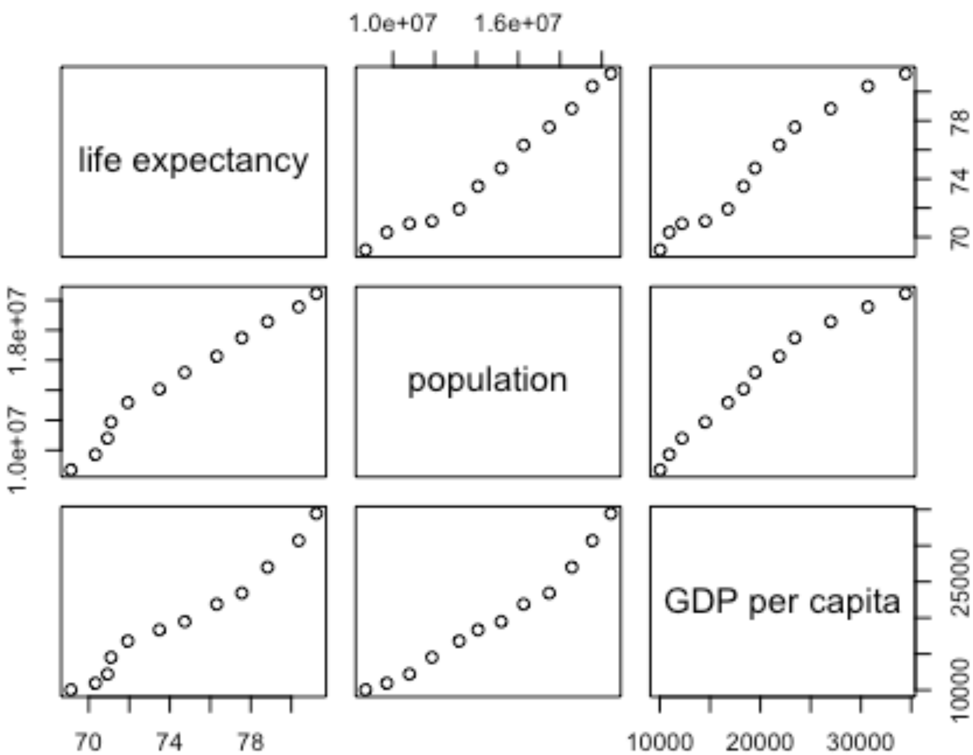
8. Calculate the increase in average life expectancy for Australians over the time period in the dataset.

```
# Clearly, the data is organized in time order, so we need only subtract the
# first from the last element of the vector Oz$lifeExp. (Try it!) More
generally,
# however, if the rows weren't in time order, you could do something like
this
# (again, work it out):

Oz$lifeExp[which.max(Oz$year)] - Oz$lifeExp[which.min(Oz$year)]
```

```
[1] 12.115
```

9. Produce pairwise scatterplots of the three numerical variables for Australia.

```
# We need to extract the variables lifeExp, pop, and gdpPercap from the data
for
# Australia, and then use the function 'plot'. Lot's of ways to do it; here's
one
# (what would happen if I didn't add the 'labels' argument?)

plot(subset(Oz, select = c(lifeExp, pop, gdpPercap)), labels = c("life
expectancy",
    "population", "GDP per capita"))
```



10. What is the increase in life expectancy for Burundi?

```
# You could do the same thing we did for Australia

Burundi <- subset(gapminder, subset = (country == "Burundi"))
Burundi$lifeExp[which.max(Burundi$year)] -
Burundi$lifeExp[which.min(Burundi$year)]

[1] 10.549
```

11. In which rows of the dataset does Swaziland occur?

```r
which(gapminder$country == "Burundi")

 [1] 205 206 207 208 209 210 211 212 213 214 215 216
```

12. How would you create a dataset that contains only the data for 2007 for all countries?

```r
# Again, we could use the function 'subset':
Data2007 <- subset(gapminder, subset = (year == 2007))
```

13. Which country had the lowest life expectancy in 2007? Which had the highest?

```r
# Here, we simply find the row corresponding to the min/max life expectancy, and
# then use that index to select and display the corresponding row of the data
# object for 2007
Data2007[which.min(Data2007$lifeExp), ]

# A tibble: 1 x 6
  country   continent  year lifeExp     pop gdpPercap
  <fct>     <fct>     <int>   <dbl>   <int>     <dbl>
1 Swaziland Africa     2007    39.6 1133066     4513.

Data2007[which.max(Data2007$lifeExp), ]

# A tibble: 1 x 6
  country continent  year lifeExp       pop gdpPercap
  <fct>   <fct>     <int>   <dbl>     <int>     <dbl>
1 Japan   Asia       2007    82.6 127467972    31656.
```

14. What is the mean GDP per capita in each country?

```r
# Hint: use the function 'tapply' on the vector gapminder$gdpPercap
MeanGDP <- tapply(gapminder$gdpPercap, gapminder$country, mean)
head(MeanGDP)

Afghanistan     Albania     Algeria      Angola   Argentina   Australia
   802.6746   3255.3666   4426.0260   3607.1005   8955.5538  19980.5956
```

15. Do all countries have data in the same range of years?

```r
# Hint: This one's a bit messy, but you could use tapply on the vector
# gapminder$year along with gapminder$country as the factor argument and the
# function range, and then scan down the results to see whether the ranges are
# the same. There is no doubt a more sophisticated way.

YearRange <- tapply(gapminder$year, gapminder$country, range)
head(YearRange)  # type our the whole vector

$Afghanistan
[1] 1952 2007

$Albania
[1] 1952 2007
```

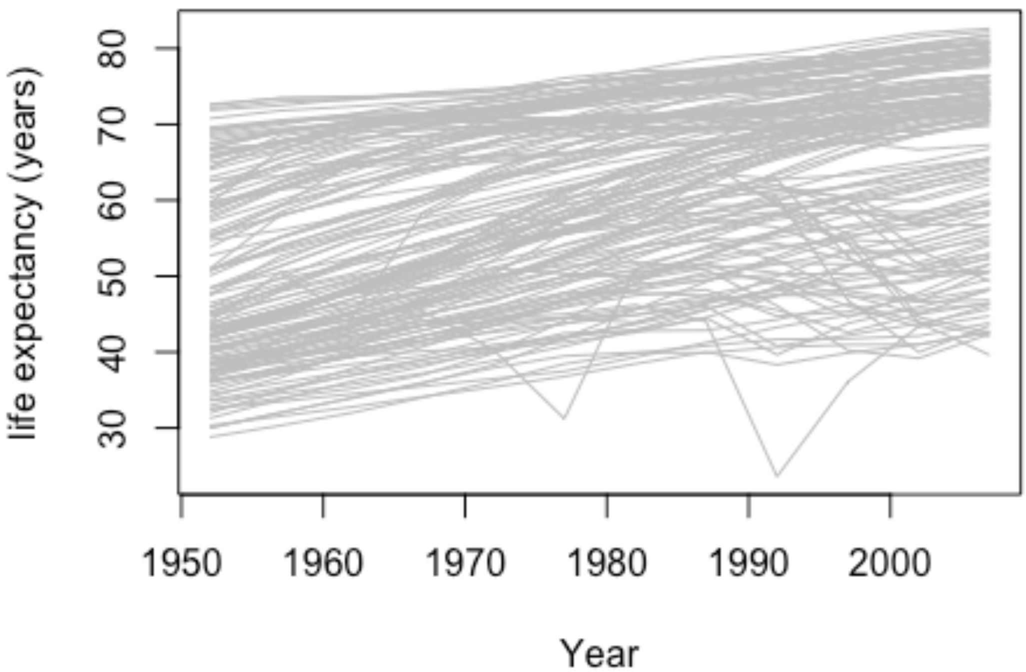
```
$Algeria
[1] 1952 2007

$Angola
[1] 1952 2007

$Argentina
[1] 1952 2007

$Australia
[1] 1952 2007
```

16. Construct a dataset consisting of only the life expectancy over time for all countries.

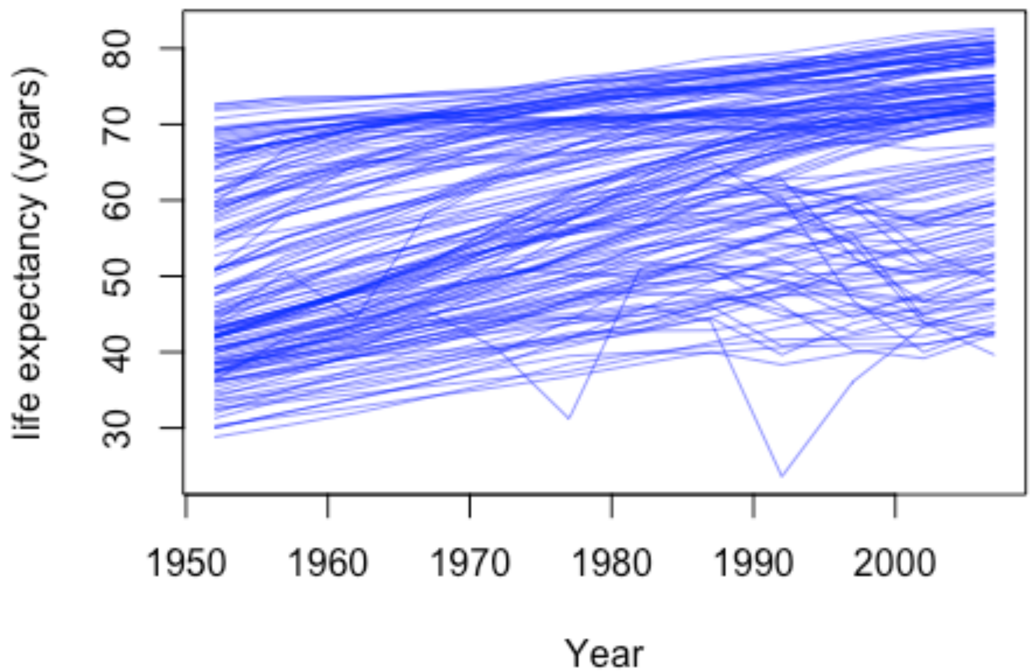
```
# Hint: Extract the data for life expectancy, and then form a matrix; see the
# help file for 'matrix'.  Make sure you give the matrix row and column names
# ('rownames', 'colnames')

LifeExp <- gapminder$lifeExp
LifeExp <- matrix(LifeExp, nrow = 12, byrow = FALSE)
colnames(LifeExp) <- unique(gapminder$country)
rownames(LifeExp) <- unique(gapminder$year)
LifeExp[1:5, 1:6]

     Afghanistan Albania Algeria Angola Argentina Australia
1952      28.801   55.23  43.077 30.015    62.485     69.12
1957      30.332   59.28  45.685 31.999    64.399     70.33
1962      31.997   64.82  48.303 34.000    65.142     70.93
1967      34.020   66.22  51.407 35.985    65.634     71.10
1972      36.088   67.69  54.518 37.928    67.065     71.93
```

17. Plot a time series plot of the life expectancy against time for all countries on the same plot.

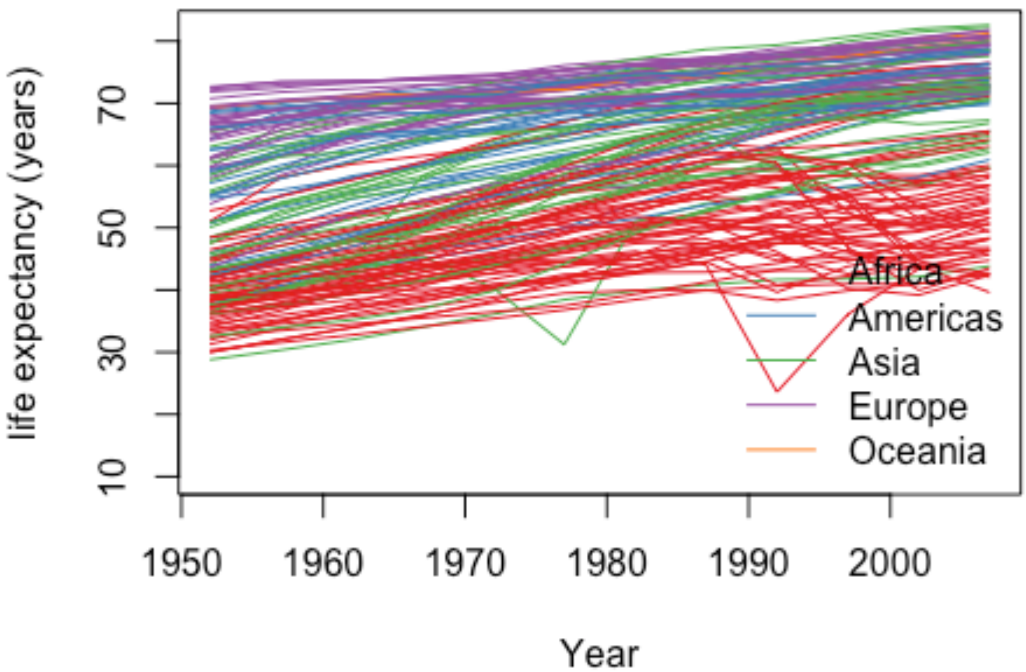
```
# Hint: See the function 'matplot'. Depending on how you transformed the
matrix
# you might have to transpose it first using 't()`. Don't forget to add axis
# labels.

matplot(rownames(LifeExp), LifeExp, xlab = "Year", ylab = "life expectancy
(years)",
    type = "l", lty = 1, col = "grey")
```



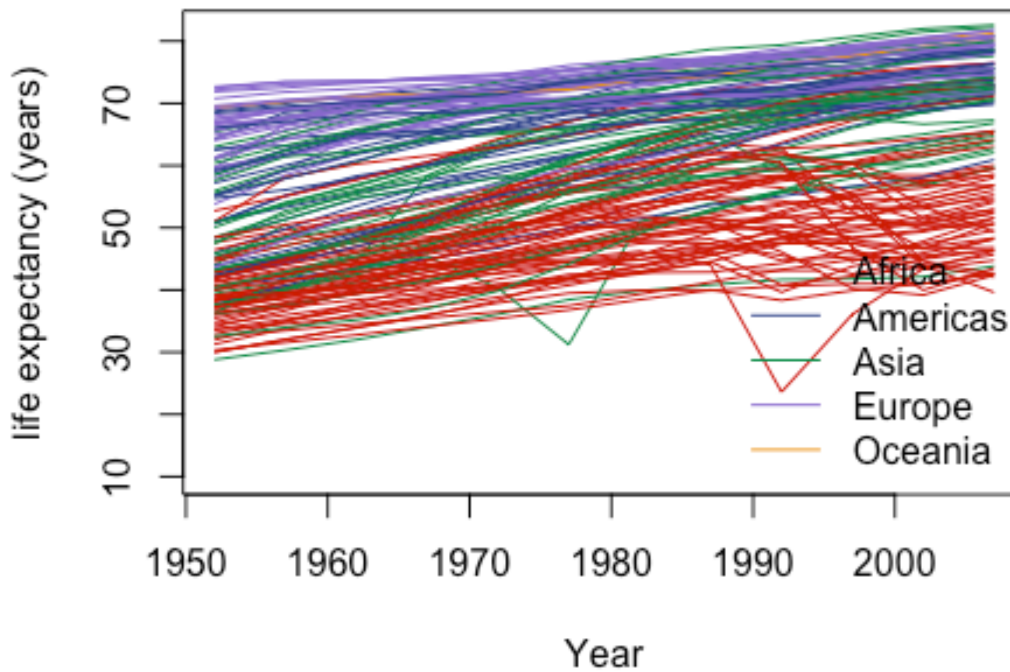

```
# With transparent colour
matplot(rownames(LifeExp), LifeExp, xlab = "Year", ylab = "life expectancy
(years)",
    type = "l", lty = 1, col = rgb(0, 0, 1, 0.5))
```




```
# If you have the RColorBrewer package installed, you could colour the
countries
# in each continent with a different colour. In the code below, ContinentCols
is
# just a big long vector containing a unique colour for each of the five
# continents.
library(RColorBrewer)
ContinentCols <- brewer.pal(5, "Set1")[tapply(gapminder$continent,
gapminder$country,
    function(x) {
        unique(x)
    })]
matplot(rownames(LifeExp), LifeExp, xlab = "Year", ylab = "life expectancy
(years)",
    type = "l", lty = 1, col = ContinentCols, ylim = c(10, 82))
legend("bottomright", col = brewer.pal(5, "Set1"), legend =
levels(gapminder$continent),
    lty = 1, bty = "n")
```



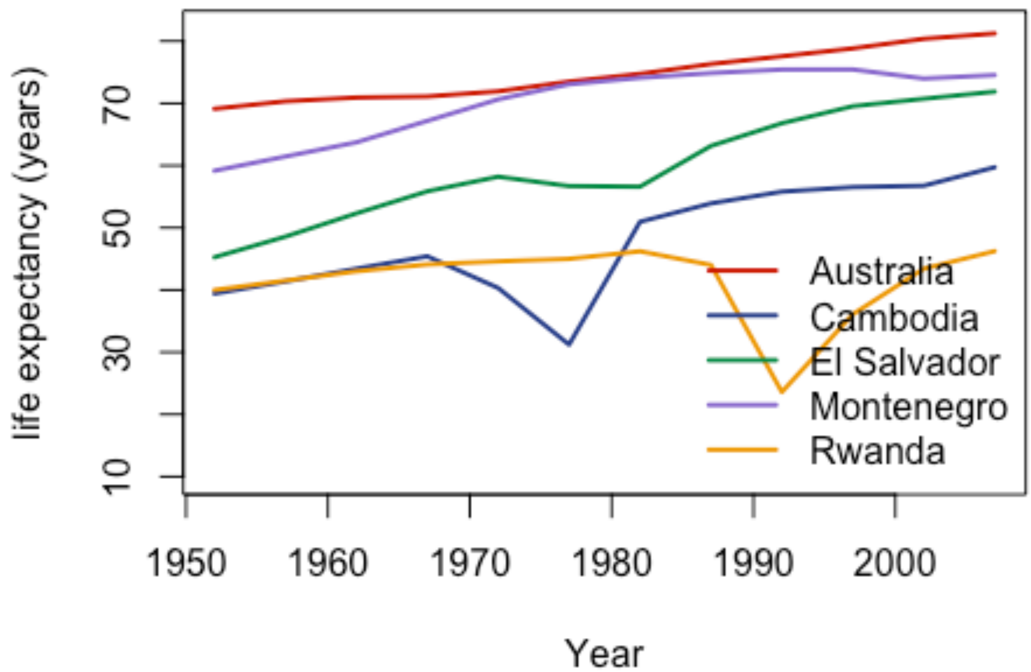

```
# In the code above, I used the function brewer.pal() to get five nice-
looking
# colours, but there is no reason why you couldn't define your own set of
five
# colours. For lots of colours, see
# www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
FiveCols <- c("red3", "royalblue4", "springgreen4", "mediumpurple3",
"orange2")
ContinentCols <- FiveCols[tapply(gapminder$continent, gapminder$country,
function(x) {
    unique(x)
})]
matplot(rownames(LifeExp), LifeExp, xlab = "Year", ylab = "life expectancy
(years)",
    type = "l", lty = 1, col = ContinentCols, ylim = c(10, 82))
legend("bottomright", col = FiveCols, legend = levels(gapminder$continent),
lty = 1,
    bty = "n")
```



18. **Challenging** Using base R commands, can you identify the country in each continent that experienced the sharpest 5-year drop in life expectancy, or if there was no drop, then the smallest increase? What was that drop/increase? For these countries, plot life expectancy over time.

```
## Let's do this in steps

# 1. We already have a data frame with all life expectancies - LifeExp The
# function diff() calculates differences between neighbouring observations in
a
# vector, and we can apply() this function to each of the columns of LifeExp,
# e.g.,
LifeExpDiff <- apply(LifeExp, 2, diff)  # 2 refers to columns

# 2. Now let's find the largest (e.g., largest negative number) drop in life
# expectancy for each country using the function min().
LifeExpDiffMin <- apply(LifeExpDiff, 2, min)
head(LifeExpDiffMin)

Afghanistan     Albania     Algeria      Angola   Argentina   Australia
      0.089      -0.419       1.307      -0.036       0.492       0.170

# 3. We now need to identify which countries in each continent had the
largest
```

```
# drop. First we need a vector of continents to which each country belongs.
We
# can't just use gapminder$country, because that would be too long. There are
# lots of ways of doing this, but here's just one:
Continents <- tapply(gapminder$continent, gapminder$country, function(x) {
    levels(gapminder$continent)[unique(x)]
})

# 4. We can now identify which country in each continent has the smallest
(most
# negative) change in life expectancy
LifeExpChange <- tapply(LifeExpDiffMin, Continents, min)
LifeExpChange

  Africa Americas     Asia   Europe  Oceania 
 -20.421   -1.511   -9.097   -1.464    0.170 

# 5. Countries given by
MaxDiffCountries <- LifeExpDiffMin[LifeExpDiffMin %in% LifeExpChange]
MaxDiffCountries

  Australia    Cambodia El Salvador  Montenegro      Rwanda 
      0.170      -9.097      -1.511      -1.464     -20.421 

matplot(rownames(LifeExp), LifeExp[, names(MaxDiffCountries)], type = "l",
lty = 1,
    lwd = 2, xlab = "Year", ylab = "life expectancy (years)", col = FiveCols,
ylim = c(10,
        82))
legend("bottomright", col = FiveCols, legend = names(MaxDiffCountries), lty =
1,
    lwd = 2, bty = "n")
```




```
# If you have a look at the help file for the gapminder dataset (?gapminder),
# you'll see some code for doing the same thing but using commands from the
# tidyverse package. I'm not sure that they're necessarily more transparent!

require(tidyverse)

gapminder %>% group_by(continent, country) %>% select(country, year,
continent, lifeExp) %>%
    mutate(le_delta = lifeExp - lag(lifeExp)) %>% summarize(worst_le_delta =
min(le_delta,
    na.rm = TRUE)) %>% filter(min_rank(worst_le_delta) < 2) %>%
arrange(worst_le_delta)

# A tibble: 5 x 3
# Groups:   continent [5]
  continent country     worst_le_delta
  <fct>     <fct>                <dbl>
1 Africa    Rwanda             -20.4
2 Asia      Cambodia            -9.10
3 Americas  El Salvador         -1.51
4 Europe    Montenegro          -1.46
5 Oceania   Australia            0.170
```