Practical 3.3: Health Impact Analysis using R

Gavin Shaddick & Matthew Thomas, University of Bath, UK.

In this session we will introduce the `R` statistical software and demonstrate its functionality through two examples. The first of these examples repeats the health impacts calculations for outdoor air pollution that you performed earlier and the second takes you through a case study of the health impacts associated with landfills sites. The session contains the following:

- An introduction to `R`,
- Example: HIA associated with outdoor air pollution,
- Example: HIA associated with landfill sites.

An introduction to `R`

`R` is an integrated suite of software facilities for data manipulation, calculation and graphical display. It provides an environment within which almost all classical and modern statistical techniques have been implemented. A few of these are built into the base R environment, but many are supplied as packages. There are about 25 packages supplied with R and many thousands more are available through the CRAN family of Internet sites (cran.r-project.org)

`RStudio` is an integrated development environment (IDE) for `R`. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

Both `R` (cran.r-project.org) and `RStudio` (www.rstudio.com) are open source and can be downloaded free of charge. They are both available for Windows, Mac OSX and Linux.

Packages

Packages are collections of `R` functions and data. The area where packages are stored is called the library. `R` comes with a standard set of packages, for example `base` (the `R` base package).

To help others who wish to do similar analyses, developers often make these packages publicly available on CRAN (www.r-cran.org), a repository for `R` software. These packages will need to be downloaded and installed.

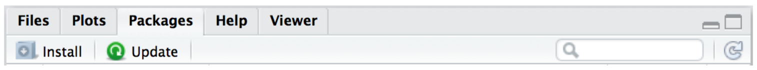
The `install.packages()` function will download and install the packages that we need. For example, the `foreign` package allows the user to read in data from other statistical analysis software such as Stata or SAS, which we can install.

```
install.packages("foreign")
```

Alternatively, we can install packages clicking the 'Install' button in the 'Packages' pane and searching for the required package.

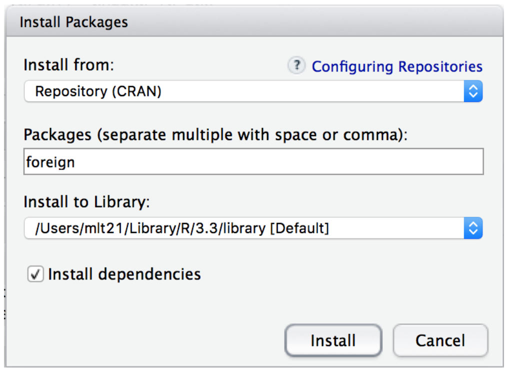
Once packages are downloaded and installed, we need to load them before using functions and/or data inside. We do this by using the `require()` function.

```
require(foreign)
```



R packages need to be loaded everytime a session in R is opened.

Packages are frequently updated. To update your packages, we use the `update.packages()`. Alternatively, we can update packages by clicking the ‘Update’ button in the Packages pane of RStudio.


Install Packages

Install from: Repository (CRAN)

Packages (separate multiple with space or comma): foreign

Install to Library: /Users/mlt21/Library/R/3.3/library [Default]

☑ Install dependencies

Getting Help with R

CRAN (the repository for R software and packages), requires all package authors to create, manuals with examples and help pages. The `help()` function and `?` operator in R provide access to help pages for all R functions and data sets in installed packages. Lets bring up the help pages for the sum function.

```
?sum
help(sum)
```

A help page will appear in the lower right pane in RStudio and will contain the following information (there maybe more):

- Description: purpose of the function
- Usage: an example of a typical implementation
- Arguments: a list of the arguments you can supply to the function and what each does
- Details: more detailed information about the function and its arguments
- Value: information about the likely output of a function (e.g. does the function return an integer, or a list, or a matrix, or something different)
- See Also: a list of useful related functions
- References: citations which can often be very useful
- Examples: example code

The basic components of data analysis using R

When working in `RStudio`, you write/edit code in the editor (upper left window). To run your code, you hit the ‘Run’ button, on a line of code and the output is displayed in the console (lower left window). If your code runs properly, the results will display in black. If there are errors in your code, warnings and errors will display in red.

Comments

R makes use of the # sign to add comments. Comments are useful so that you and others can understand what the R code does. It is good practice to always comment your code. Whenever R encounters the # operator, it will ignore everything printed after it in the current line so they don’t influence your results.

```
# Calculating 3+4
3 + 4
[1] 7
```

As you can see the line beginning with `#` has produced no results, and the other line has. In its most basic form, `R` can be used as a simple calculator, using some of the following operators.

Functions

`R` comes with many functions that are installed as part of the base package. All functions have names and take arguments in parentheses: `function(...)`. Here are some of the basic functions `R` knows

- `print()` – prints objects
- `sum()` – computes the sum of all elements entered
- `table()` – computes frequency tables
- `summary()` – computes a summary of inputted data
- `length()` – tells you the number of entries in a vector
- `round()` – rounds the input to the nearest decimal place.

```
# Printing 'Hello!'
print('Hello!')
[1] "Hello!"

# Sum 1,2 and 3
sum(1,2,3)
[1] 6

# Absolute value of 2 and -2
abs(2); abs(-2)
[1] 2
[1] 2
```

Storing numbers as variables

We can assign numbers (and many more things) to named variables so that we can use them again or use them as part of other calculations. To assign a number, for example, to a variable we `<-`.

```
# Assign the value 42 to x
x <- 42

# Print out the value of the object x
x
[1] 42

# Add 2 to x
x + 2
[1] 44

# Add 5 to x
x + 5
[1] 47
```

If a calculation is made, the results are printed but not stored, unless assigned to another object. So if we are interested in storing the results from our calculations to use in further ones later, they should be stored in other objects.

```
# Add 5 to x and reassigning to x
y <- x + 5

# Print out the value of the object x
y
[1] 47
```

When storing results of your calculations, be aware that assigning a result to a variable will overwrite what was already there.

Vectors

A vector is a simple tool to store data. They contain a collection of numbers or text. You can create a vector using the concatenates function `c()`. This function takes a series of numbers or text, separated by commas, and puts them together into a vector.

```
# Storing 5 values in a vector and assigning to 'a'
a <- c(1, 2, 3, 4, 5)

# Storing 5 values in a vector and assigning to 'b'
b <- c(-2, -4, 6, 7, 8)
```

More information on the `c()` function can be seen on the help pages by typing `help(c)` into `R`

Elements from a vector can be extracted using '[]' with the corresponding element(s) to extract.

```
# Extracting the first element of a
a[1]
[1] 1

# Extracting the first and third element of a
a[c(1,3)]
[1] 1 3
```

Vectors can be used in arithmetic expressions, and the operations are performed element-wise. This means that if we want to add two vectors together, `R` will take the first element of the vectors and add them, then take the second elements of the vectors and add them and so on.

```
# Lets add a and b
a + b
[1] -1 -2  9 11 13
```

Similarly, if we want to multiply by two and add five to all elements of a vector, `R` will do this element-wise.

```
# Lets add a and b
2 * a + 5
[1]  7  9 11 13 15
```

Dataframes

Dataframes are very flexible objects for data analysis in `R`. Dataframes are two-dimensional, with an observation taking up a row, and a variable taking up a column, similar to a database or a spreadsheet. It can also be thought of as a collection of vectors concentrated into one object.

We can create dataframes using the `data.frame()` function. This function will take a list of vectors and concatenate them together into one object. You can also give specific variable names. More information and examples of the `data.frame()` function can be see by typing `data.frame` into `R`.

```
# Storing 5 values in a vector and assigning to 'a'
a <- c(1, 2, 3, 4, 5)

# Storing 5 values in a vector and assigning to 'b'
b <- c(-2, -4, 6, 7, 6)

# Creating a dataframe from 'a' and 'b'
mydata <- data.frame(a,b)

# Printing mydata
mydata
  a  b
1 1 -2
2 2 -4
3 3  6
4 4  7
5 5  6
```

Activity

- Create a vector called `c` which contains the elements 2, 3, 1, 6 and -1 and create a dataframe that contains variables `a`, `b` and `c`.

Datasets in `R`

`R` also has an extensive repository of example datasets which we can use. To call datasets to the workspace, we use the `data()` function. If entered without arguments, it will bring up a list of all datasets that currently available within `R`.

```
# List all the datasets stored in R
data()
```

Have a look at the `iris` dataset (Edgar Anderson's Iris Data, https://en.wikipedia.org/wiki/Iris_flower_data_set)

```
# Load the iris dataset in the workspace
data(iris)
```

Getting to know the structure of dataframes

Once a dataframe has been loaded into `R`, you should understand the data stored within. Initially, we can understand our dataset by finding the number of observations and variables in dataframes by using the `nrow()` and `ncol()` functions respectively.

```
# Viewing the structure of the iris dataset
nrow(iris)
[1] 150

# Viewing the first 5 rows of the iris dataset
ncol(iris)
[1] 5
```

A quick way of viewing the dataset to see the data are using the `names()`, `str()` and `head()` functions. The `names()` function will display the variable names within a dataframe. The `str()` function will display the structure of the dataset and the `head()` function will display the first 6 rows in the dataframe.

```
# Display the variable names
names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
[5] "Species"

# Viewing the structure of the iris dataset
str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

# View the first 6 rows of the iris dataset
head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

# View the last 6 rows of the iris dataset
tail(iris)
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Extracting and creating variables

Data within dataframes can be extracted using '[]'. As dataframes are two-dimensional objects, we need to specify two things, the row and/or the column, separated with a common for example [,]. Lets extract (i) the variable `Sepal.Width` from `iris`, (ii) the first row from of `iris` and (iii) 3rd row for variable `Petal.Length` from `iris`.

```
# Extracting the variable a from mydata
iris[,'Sepal.Width']
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
 [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
 [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
 [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
 [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
 [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
[103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
```

```
[120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
[137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0

# Extracting the row (or observation) from mydata
iris[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa

# Extracting the 3rd row for variable b from mydata
iris[3,'Petal.Length']
[1] 1.3
```

Alternatively, you can extract variables from dataframes we using the $ operator. We first specify the dataset then give the name of the variable that we want. Lets extract the variable `Sepal.Width` from our new dataframe `iris`.

```
# Extracting the variable a from mydata
iris$Sepal.Width
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
 [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
 [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
 [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
 [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
 [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
[103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
[120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
[137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

Creating a new variable within a dataframe is straightforward. Let's create a variable `Sepal.Width.Plus.Sepal.Length` within `iris` which is the sum of the variables `Sepal.Width` and `Petal.Length`. For this we extract the variables `Sepal.Width` and `Sepal.Length` from `iris`, sum them and assign the results to `Sepal.Width.Plus.Sepal.Length` in `iris`.

```
# Creating variable c by adding a and b together
iris$Sepal.Width.Plus.Sepal.Length <- iris$Sepal.Width + iris$Sepal.Length

# Printing the variable aplusb
iris$Sepal.Width.Plus.Sepal.Length
  [1]  8.6  7.9  7.9  7.7  8.6  9.3  8.0  8.4  7.3  8.0  9.1  8.2  7.8  7.3
 [15]  9.8 10.1  9.3  8.6  9.5  8.9  8.8  8.8  8.2  8.4  8.2  8.0  8.4  8.7
 [29]  8.6  7.9  7.9  8.8  9.3  9.7  8.0  8.2  9.0  8.5  7.4  8.5  8.5  6.8
 [43]  7.6  8.5  8.9  7.8  8.9  7.8  9.0  8.3 10.2  9.6 10.0  7.8  9.3  8.5
 [57]  9.6  7.3  9.5  7.9  7.0  8.9  8.2  9.0  8.5  9.8  8.6  8.5  8.4  8.1
 [71]  9.1  8.9  8.8  8.9  9.3  9.6  9.6  9.7  8.9  8.3  7.9  7.9  8.5  8.7
 [85]  8.4  9.4  9.8  8.6  8.6  8.0  8.1  9.1  8.4  7.3  8.3  8.7  8.6  9.1
 [99]  7.6  8.5  9.6  8.5 10.1  9.2  9.5 10.6  7.4 10.2  9.2 10.8  9.7  9.1
[113]  9.8  8.2  8.6  9.6  9.5 11.5 10.3  8.2 10.1  8.4 10.5  9.0 10.0 10.4
[127]  9.0  9.1  9.2 10.2 10.2 11.7  9.2  9.1  8.7 10.7  9.7  9.5  9.0 10.0
[141]  9.8 10.0  8.5 10.0 10.0  9.7  8.8  9.5  9.6  8.9
```

Activity

- Create a new variable called `Sepal.Width.Plus.Sepal.Length` in `iris` which multiplies `Sepal.Width` and `Petal.Length` together.

Subsetting dataframes

We can use subsetting to easily access specific data from dataframes. Logical operators are crucial for subsetting data. When `R` evaluates statements containing logical operators it will return either `TRUE` or `FALSE`.

Here are some of the logical operators in `R`

- `<` - less than
- `<=` - less than or equal to
- `>` - greater than
- `>=` - greater than or equal to
- `==` - equal
- `!=` - not equal
- `&` = and
- `|` - or

```
# Checking whether 1 == 2
1 == 2
[1] FALSE

# Checking whether1 == 1
1 == 1
[1] TRUE
```

Suppose we are interested in extracting the rows of the dataframe `iris` where the variable `Sepal.Width` is greater than 3. To subset data, we use the `subset()` function.

```
# Subsetting iris where `Sepal.Width` is less than 3
subset(iris, Sepal.Width > 4)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
16          5.7         4.4          1.5         0.4  setosa
33          5.2         4.1          1.5         0.1  setosa
34          5.5         4.2          1.4         0.2  setosa
   Sepal.Width.Plus.Sepal.Length
16                          10.1
33                           9.3
34                           9.7
# And assign it to a new dataframe
BigIris <- subset(iris, Sepal.Width > 3)
```

Activities

- Use subsetting to extract the rows of `iris` where the variable `Sepal.Width` is equal to 2.9.
- Use subsetting to extract the rows of `iris` where the variable `Sepal.Width` is less than 2.9 and `Sepal.Length` greater than 3.

Example: HIA associated with outdoor air pollution

We now demonstrate how using dataframes can allow us to perform HIA very efficiently. Following the example on air pollution you have already seen, we will be calculating the annual number of deaths attributable to $PM_{2.5}$.

Preliminaries

We wish to estimate the annual number of deaths attributable to $PM_{2.5}$ air pollution. In order to do this, we need

- a relative risk (RR),
- the population at risk for the areas of interest,
- the overall mortality rate (OMR), and
- a baseline value for air pollution (for which there is no associated increase in risk).

In this example, we use a RR of 1.06 per $10 \mu g m^{-3}$, the population at risk is 1 million and the OMR is 80 per 10000. We first enter this information into `R` by assigning these values to a series of variables.

```r
# Relative Risk
RR <- 1.06

# Size of population
Population <- 1000000

# Unit for the Relative Risk
RR_unit <- 10

# Overall mortalilty count, used for calulating the overall mortality rate
OMR_count <- 80

# Denonmmator (population at risk), used for calulating the overall mortality rate.
OMR_pop <- 10000

# Mortality rate
OMR = OMR_count/OMR_pop
OMR
[1] 0.008

# Baseline value of PM2.5 for which there is no increased risk
baseline <- 5

# Population attributable fraction
#PAF = (Proportion of population exposed*(RR-1))/(Proportion of population exposed*(RR-1)+1).
#In this case the proportion of the population exposed is one.

PAF = (RR-1)/RR
PAF
[1] 0.05660377
```

In this example, we will calculate the attributable deaths for increments of 10, however the following code is general and will work for any increments.

```r
# PM2.5 categories
PM2.5.cats <- c(5,15,25,35,45,55,65,75,85,95,105)

# Create a dataframe containing the PM2.5 categoriess
Impacts <- data.frame(PM2.5.cats)
```

Calculating Risks

We now calculate the increases in risk for each category of $PM_{2.5}$. For each category, we find the increase in risk compared to the baseline.

For the second category, with $PM_{2.5} = 15$, the risk will be 1.06 (the original RR) as this is $10\mu gm^{-3}$ (one unit) greater than the baseline.

For the next category, $PM_{2.5}$ is $10\mu\text{gm}^{-3}$ higher than the previous category (one unit in terms of the RR) and so the risk in that category again be increased by a factor of 1.06 (on that of the previous category). In this case, the relative risk (with respect to baseline) is therefore `1.06 * 1.06 = 1.1236`.

For the next category, $PM_{2.5} = 25$ which is again $10\mu\text{gm}^{-3}$ (one unit in terms of the RR) higher, and so the relative risk is 1.06 multiplies by the previous value, i.e. `1.06 * 1.1236 =  1.191016`.

We can calculate the relative risks for each category (relative to baseline) in `R`. For each category, we find the number of units from baseline and repeatedly multiple the RR by this number. This is equivalent to raising the RR to the power of (Category-Baseline)/Units, e.g.

$$\text{RR}^{\left(\frac{\text{Category-Baseline}}{\text{Units}}\right)}$$

We add another column to the Impacts dataframe containing these values.

```
# Calculating Relative Risks
Impacts$RR <- RR^((Impacts$PM2.5.cats - baseline)/RR_unit)
```

Once we have the RR for each pollution level, we can calculate the rate for each category. This is found by applying the risks to the overall rate. Again, we add these numbers to the Impacts dataframe as an additional column.

```
# Calculating the rates in each category
Impacts$Rate <- Impacts$RR * OMR

# Add the PAFs for each category
Impacts$PAF <- Impacts$RR * (Impacts$RR-1)/Impacts$RR

# Add the nuimber of (expected) deaths  per year for each category
Impacts$Deaths.Per.Year <- Impacts$Rate * Population
```

For each category, we need to calculate the extra deaths (with reference to the overall rate). The number of deaths for the reference category is the first number in the `Deaths.Per.Year` column.

```
# The number of deaths
Impacts$Deaths.Per.Year[1]
[1] 8000

# We can then calculate the excess numbers of deaths for each category
Impacts$Extra.Deaths.Per.Year <- Impacts$Deaths.Per.Year - Impacts$Deaths.Per.Year[1]
```

For each category, we then want to calculate the number of deaths gained. These are the difference between the values in each category. We can find these using the `diff()` function. This will produce a set of differences for which the length is one less than the number of rows in our Impacts dataframe. We need to add a zero to this to ensure that they line up when we add them as another column.

```
# Calculate the number of deaths gained
diff(Impacts$Extra.Deaths.Per.Year)
 [1] 480.0000 508.8000 539.3280 571.6877 605.9889 642.3483 680.8892
 [8] 721.7425 765.0471 810.9499
```

```
# We can now add these gains to the main Impacts dataframe
Impacts$Gain <- c(0,diff(Impacts$Extra.Deaths.Per.Year))

# Show the results
Impacts
   PM2.5.cats       RR        Rate       PAF Deaths.Per.Year
1           5 1.000000 0.008000000 0.0000000        8000.000
2          15 1.060000 0.008480000 0.0600000        8480.000
3          25 1.123600 0.008988800 0.1236000        8988.800
4          35 1.191016 0.009528128 0.1910160        9528.128
5          45 1.262477 0.010099816 0.2624770       10099.816
6          55 1.338226 0.010705805 0.3382256       10705.805
7          65 1.418519 0.011348153 0.4185191       11348.153
8          75 1.503630 0.012029042 0.5036303       12029.042
9          85 1.593848 0.012750785 0.5938481       12750.785
10         95 1.689479 0.013515832 0.6894790       13515.832
11        105 1.790848 0.014326782 0.7908477       14326.782
   Extra.Deaths.Per.Year     Gain
1                  0.000   0.0000
2                480.000 480.0000
3                988.800 508.8000
4               1528.128 539.3280
5               2099.816 571.6877
6               2705.805 605.9889
7               3348.153 642.3483
8               4029.042 680.8892
9               4750.785 721.7425
10              5515.832 765.0471
11              6326.782 810.9499
```

Example: HIA associated with landfill sites

This example follows the analysis associated with the 'Waste and human health: evidence and needs' study.

Preliminaries

For this analysis, we need the `ggplot2` package which will produce graphics in `R`. Make sure that this package is downloaded and installed in `R`. We use the `require()` function to load them into the current `R` workspace.

```
# Loading required packages
require(ggplot2)
```

Calculating the health impacts associated with landfills

Details of contaminants or pollutants for a range of sources from the 28 member states of the European Union (plus Iceland, Liechtenstein, Norway and Switzerland) are registered on the European Pollutant and Transfer Register (E-PRTR) which is run by the European Environment Agency (EEA). The information registered here includes details from Landfill sites. The full dataset of E-PRTR is hosted by the European Environment Agency (EEA) in its Data Service.

From the database, we have extracted information from three files that are stored in the Data folder:

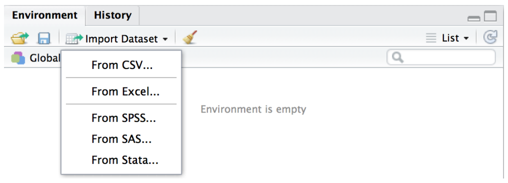
- `POLLUTANTRELEASE.csv` - CSV file containing all the contaminant and pollutant releases on the E-PRTR website,
- `FACILITYREPORT.csv` - CSV file containing all the information about the Facilities that report pollutant,
- `POLLUTANTRELEASEANDTRANSFERREPORT.csv` - CSV file containing extra information about the Pollutants and the reporting (including the year of reporting).

A CSV is a simple file format used to store tabular data, such as a spreadsheet. Files in CSV format can be written from programs such as Microsoft Excel. Importing data from 'comma separated values' (.csv) files is straightforward.

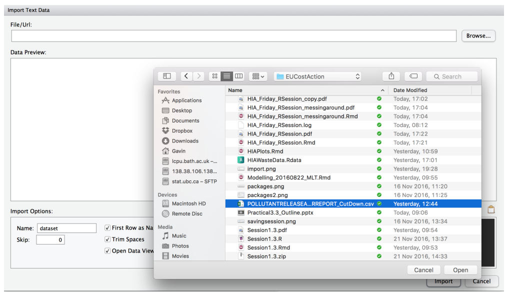
Importing a dataset

Using `RStudio` we can import datasets using the 'Import Dataset' option in the 'Environment' window. You can choose whether you want to import a .csv file, an Excel file or files from SPSS, SAS or stata. You can import many more filetypes by installing an appropriate package and writing code to read in the files (see the next section).

Try the following with the file `POLLUTANTRELEASEANDTRANSFERREPORT_CutDown.csv`. Note that this is a cut down and tidied up version of the third of the files listed above. It is used here as an example of how to import data into `R` as the original files are quite large and can take a while to load (as well as being a little messy in places)



After clicking on browse, you can find the file you require.



Import Options:

Name: dataset
Skip: 0
✓ First Row as Na
✓ Trim Spaces
✓ Open Data View

Name	Date Modified
HIA_Friday_RSession_copy.pdf	Today, 17:02
HIA_Friday_RSession_messingaround.pdf	Today, 17:04
HIA_Friday_RSession_messingaround.Rmd	Today, 17:04
HIA_Friday_RSession.log	Today, 08:12
HIA_Friday_RSession.pdf	Today, 17:22
HIA_Friday_RSession.Rmd	Today, 17:21
HIAPlots.Rmd	Yesterday, 10:59
HIAWasteData.Rdata	Yesterday, 17:01
import.png	Yesterday, 19:28
Modelling_20160822_MLT.Rmd	Yesterday, 09:55
packages.png	16 Nov 2016, 11:20
packages2.png	16 Nov 2016, 11:25
POLLUTANTRELEASEA...RREPORT_CutDown.csv	Yesterday, 12:44
Practical3.3_Outline.pptx	Today, 09:06
savingsession.png	16 Nov 2016, 13:34
Session1.3.pdf	Yesterday, 09:54
Session1.3.R	21 Nov 2016, 13:37
Session1.3.Rmd	Yesterday, 09:53
Session1.3.zip	21 Nov 2016, 14:33

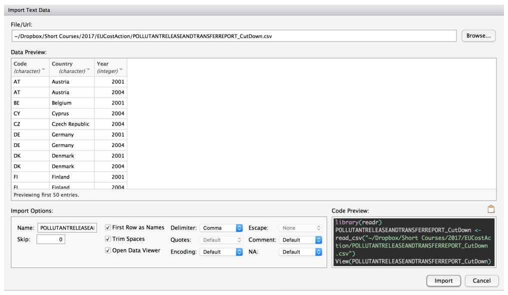
You will then see a preview of your data together with an example of the code that `R` will use to read in the data. You can choose what the dataset will be called when it has been imported and set other options. These options can be very useful if a dataset isn't quite as clearly formatted, as is the case with the original files. For example, you can decide to skip a certain number of rows at the beginning of the file which is often useful or to use different text delimiters (things that indicate the gap between columns in th data,).

When you are ready press 'Import'.

Importing a datafile using code

To read a CSV file into `R` we use the `read.csv()` function.

```r
# Read in a .csv file
mydata <- read.csv(file='<filepath_and_filename>.csv')
```



File/Url:
~/Dropbox/Short Courses/2017/EUCostAction/POLLUTANTRELEASEANDTRANSFERREPORT_CutDown.csv

Data Preview:

Code (character)	Country (character)	Year (integer)
AT	Austria	2001
AT	Austria	2004
BE	Belgium	2001
CY	Cyprus	2004
CZ	Czech Republic	2004
DE	Germany	2001
DE	Germany	2004
DK	Denmark	2001
DK	Denmark	2004
FI	Finland	2001
FI	Finland	2004

Previewing first 50 entries.

Import Options:

Name: POLLUTANTRELEASEAI
Skip: 0
✓ First Row as Names
✓ Trim Spaces
✓ Open Data Viewer
Delimiter: Comma
Quotes: Default
Encoding: Default
Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(readr)
POLLUTANTRELEASEANDTRANSFERREPORT_CutDown <-
read_csv("~/Dropbox/Short Courses/2017/EUCostAc
tion/POLLUTANTRELEASEANDTRANSFERREPORT_CutDown
.csv")
View(POLLUTANTRELEASEANDTRANSFERREPORT_CutDown)
```

Note that we should put in the full path to the file when using `read.csv`. Or we can set a **working directory**. The function `getwd()` will tell you the filepath of the current working directory of the `R` process.

```
getwd()
[1] "/Users/Gavin/Dropbox/ShortCourses/2017/EUCostAction"
```

The function `setwd()` is used to set the working directory to your chosen location.

```
setwd("Chosen_Directory_Path")
```

Or you can use the files Pane in ‘RStudio’ and choose the working directory from the ‘Session’ menu (Session->Set Working Directory->Files Pane Location). Once we have set the working directory we don’t have to repeatedly type the path to the folder.

Returning to reading in the `.csv` file. Let’s read in an example. We will use a reduced version of the third of the files listed above (note that the first two files are very large and may take a while to load).

```
# Calculate the number of deaths gained
mydata <- read.csv(file='POLLUTANTRELEASEANDTRANSFERREPORT_CutDown.csv')

# Lets view the first few rows
head(mydata)
  Code        Country Year
1   AT        Austria 2001
2   AT        Austria 2004
3   BE        Belgium 2001
4   CY         Cyprus 2004
5   CZ Czech Republic 2004
6   DE        Germany 2001
```

Information on landfill sites

Taken together, the three datasets contain information about type contaminant and pollutant releases on the E-PRTR website. We have extracted the relevant information on landfill waste which is coded on the E-PRTR website as `5 (d)`. The relevant information has been combined and made into a single dataset that we will use for this example. This datasets also has information on populations within buffers around the landfills and country specific birthrates. The dataset is in `R` format and is called `HIAWasteData`. You can load this dataset into workspace using `load()` function.

```
# Loading in landfill  data
load('HIAWasteData.RData')
```

We can have a quick look at the data using the `head()` function which will show us the first six rows of the dataset. The `tail()` function would show us the last six rows.

```
# Viewing the top six rows of the landfill  data
head(HIAWasteData)
  CountryName FacilityID FacilityReportID
1     Austria     229289          1103758
2     Austria       5966          1103786
3     Austria     182114          1103814
4     Austria       5881          1103732
5     Austria      52021          1103776
6     Austria     229289          1103758
                                 FacilityName                City
1 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
2           Abfallbeseitigungsverband Westtirol              Roppen
3                       PORR Umwelttechnik GmbH               Steyr
4                       Stadtwerke Judenburg AG         Gasselsdorf
5                     Hettegger Entsorgung GmbH Sankt Veit im Pongau
6 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
       Lat     Long Buffer Population   BirthRate PercLowBirthWeight
1 47.98172 13.62269    3km   12569.02 0.009564917                6.8
2 47.23392 10.82542    2km    1226.93 0.009564917                6.8
3 48.12614 14.39919    4km    3343.56 0.009564917                6.8
4 47.20506 14.62006    4km    7182.40 0.009564917                6.8
5 47.32281 13.18033    3km    8804.15 0.009564917                6.8
6 47.98172 13.62269    2km    5495.74 0.009564917                6.8
       CARate  RespRate
1 0.002596748 0.0180708
2 0.002596748 0.0180708
3 0.002596748 0.0180708
4 0.002596748 0.0180708
5 0.002596748 0.0180708
6 0.002596748 0.0180708
```

We can see that this dataset contains the following variable:

- `CountryName` - Name of country
- `FacilityID` - ID of the landfill site
- `FacilityReportID` - ID of the corresponding pollution release information
- `FacilityName` - Name of landfill site
- `City` - City
- `Lat` - Latitude of landfill site
- `Long` - Longitude of landfill site
- `Buffer` - Distance around the landfill site for populations
- `Population` - Population within designated buffer
- `BirthRate` - Country specific birthrate
- `PercLowBirthWeight` - Country specific percentage of all births that are low birth rate
- `CARate` - Country specific rate of congenital abnormality
- `RespRate` - Country specific rate of respiratory disease

We also need information on the health outcomes, for example relative risks and life expectancies. This information is in a `R` dataset called `Health`. You can load this dataset into workspace using the comment `load('Health.RData')`.

```
# Loading in Landfill site data and have a look at the first rows
load('Health.RData')
head(Health)
      Plant ExpoBuffer ExpoIndex              Outcome Risk RiskLow RiskUpp
```

```
1 Landfills          2      Dist Congenital Anomalies 1.02   1.010   1.030
2 Landfills          2      Dist Respiratory Diseases 1.09   1.000   1.190
3 Landfills          2      Dist Annoyance from Odour 5.40      NA      NA
4 Landfills          2      Dist     Low Birth Weight 1.06   1.052   1.062
     DW    L
1 0.170 79.6
2 0.080  1.0
3 0.030  1.0
4 0.106 79.6
```

We can see that this dataset contains the following variable:

- `Plant` - Type of site
- `ExpoBuffer` - Exposure buffer (defined by distance)
- `ExpoIndex` - How the buffer is defined (in this case by distance)
- `Outcome` - Health outcome
- `Risk` - Relative risk
- `RiskLow` - Lower end of 95% confidence interval around the relative risk
- `RiskUpp` - Upper end of 95% confidence interval around the relative risk
- `DW` - Disability weight
- `L` - Disease duration

We will work through an example of calculating the health impacts associated with low birth weight. First, we need to calculate the size of the population that may experience low birth weight, e.g. the number of new borns. This will be the population multiplied by the birth rate.

```r
# Calculating the population  exposed. Those who may experience
# low birth weight i.e. all new borns
HIAWasteData$PopExposed <- HIAWasteData$Population * HIAWasteData$BirthRate
```

We also need to extract the figures required to calculate the health impacts for this particular outcome from the `Health` dataset; the relative risk, disability weight and disease duration.

```r
# Extracting relative risk, disability weight and disease duration
# for the disease 'Low Birth Weight'
LBW <- subset(Health, Outcome == 'Low Birth Weight')
```

The number of attributable cases (AC) is the produce of the attributable fraction, incidence rate and population exposed.

$$AC = AF * Rate_{exp} * Pop_{exp}$$

where

- AF - The attributable fraction
- $Rate_{exp}$ - Incidence rate of health outcome
- Pop_{exp} - Population exposed

We first extract the relative risk, `RR`, associated of low birth weight associated with proximity to landfill sites and then use this to calculate the number of attributable cases.

```r
# Extracting Relative Risk
RR <- LBW$Risk
RR
[1] 1.06

# Calculating the number of Attributable cases
HIAWasteData$AC.LBW <- ((RR - 1)/RR) * (HIAWasteData$PercLowBirthWeight/100) *
```

```
    HIAWasteData$PopExposed

head(HIAWasteData)
  CountryName FacilityID FacilityReportID
1     Austria     229289          1103758
2     Austria       5966          1103786
3     Austria     182114          1103814
4     Austria       5881          1103732
5     Austria      52021          1103776
6     Austria     229289          1103758
                                   FacilityName                City
1 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
2           Abfallbeseitigungsverband Westtirol              Roppen
3                        PORR Umwelttechnik GmbH               Steyr
4                        Stadtwerke Judenburg AG         Gasselsdorf
5                      Hettegger Entsorgung GmbH Sankt Veit im Pongau
6 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
       Lat     Long Buffer Population   BirthRate PercLowBirthWeight
1 47.98172 13.62269    3km   12569.02 0.009564917                6.8
2 47.23392 10.82542    2km    1226.93 0.009564917                6.8
3 48.12614 14.39919    4km    3343.56 0.009564917                6.8
4 47.20506 14.62006    4km    7182.40 0.009564917                6.8
5 47.32281 13.18033    3km    8804.15 0.009564917                6.8
6 47.98172 13.62269    2km    5495.74 0.009564917                6.8
       CARate  RespRate PopExposed     AC.LBW
1 0.002596748 0.0180708  120.22163 0.46273987
2 0.002596748 0.0180708   11.73548 0.04517054
3 0.002596748 0.0180708   31.98087 0.12309619
4 0.002596748 0.0180708   68.69906 0.26442657
5 0.002596748 0.0180708   84.21096 0.32413277
6 0.002596748 0.0180708   52.56630 0.20233065
```

Note, we could have used `LBW$Risk` directly in this equation, rather than first assigning the relative risk to `RR`.

Now we use the disability weights and disease duration to obtained the disability adjusted life years (DALYS) for low birth weight associated with living close to landfills.

$$DALY = AC * DW * L$$

where

- AC - Attributable cases
- DW - Disability weight
- L - Disease duration

First, we extract the disability weight and disease duration from the `Health` dataset.

```
# Extracting disability weight
DW <- LBW$DW
DW
[1] 0.106

# Extracting disease duration
L  <- LBW$L
```

```
L
[1] 79.6
```

These are multiplied together with the number of attributable cases from the `HIAWasteData` dataset that you have just calculated.

```
# Calculating DALYs
HIAWasteData$DALY.LBW <- HIAWasteData$AC.LBW * DW * L
head(HIAWasteData)
  CountryName FacilityID FacilityReportID
1     Austria     229289          1103758
2     Austria       5966          1103786
3     Austria     182114          1103814
4     Austria       5881          1103732
5     Austria      52021          1103776
6     Austria     229289          1103758
                                 FacilityName                City
1 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
2            Abfallbeseitigungsverband Westtirol              Roppen
3                        PORR Umwelttechnik GmbH               Steyr
4                        Stadtwerke Judenburg AG         Gasselsdorf
5                      Hettegger Entsorgung GmbH Sankt Veit im Pongau
6 Wasserreinhaltungsverband Lenzing - Lenzing AG             Obereck
       Lat     Long Buffer Population   BirthRate PercLowBirthWeight
1 47.98172 13.62269    3km   12569.02 0.009564917                6.8
2 47.23392 10.82542    2km    1226.93 0.009564917                6.8
3 48.12614 14.39919    4km    3343.56 0.009564917                6.8
4 47.20506 14.62006    4km    7182.40 0.009564917                6.8
5 47.32281 13.18033    3km    8804.15 0.009564917                6.8
6 47.98172 13.62269    2km    5495.74 0.009564917                6.8
       CARate  RespRate PopExposed     AC.LBW DALY.LBW
1 0.002596748 0.0180708  120.22163 0.46273987 3.904414
2 0.002596748 0.0180708   11.73548 0.04517054 0.381131
3 0.002596748 0.0180708   31.98087 0.12309619 1.038636
4 0.002596748 0.0180708   68.69906 0.26442657 2.231126
5 0.002596748 0.0180708   84.21096 0.32413277 2.734903
6 0.002596748 0.0180708   52.56630 0.20233065 1.707185
```

We might be interested in finding the total number of attributable cases and DALYs associated with 0-2 km of landfill sites. To do this we need to extract the data relating to the `2km` buffers. We will do this by subsetting the dataframe based on the `2km' entries in the`Buffer‘ column.

```
# Extracting all data within a 2km buffer of a landfill
HIAWasteData.2km <- subset(HIAWasteData, Buffer == '2km')
```

We then sum the `AC.LBW` and `DALY.LBW` columns and assign the results to two new variables, `ACs.2km` and `DALYs.2km` respectively.

```
# Adding all ACs and DALYs within a 2km buffer
ACs.2km <- sum(HIAWasteData.2km$AC.LBW)
DALYs.2km <- sum(HIAWasteData.2km$DALY.LBW)

# Printing ACs and DALYs at 0-2km buffer
ACs.2km
[1] 283.5344
DALYs.2km
```

```
[1] 2392.35
```

We then repeat this for the 0-3km and 0-4 km buffers.

```
# Extracting all data within 0-3km buffer of a landfill
HIAWasteData.3km <- subset(HIAWasteData, Buffer == '3km')

# Adding all ACs and DALYs within 0-3km buffer
ACs.3km <- sum(HIAWasteData.3km$AC)
DALYs.3km <- sum(HIAWasteData.3km$DALY)

# Extracting all data within 0-4km buffer of a landfill
HIAWasteData.4km <- subset(HIAWasteData, Buffer == '4km')

# Adding all ACs and DALYs within 0-4km buffer
ACs.4km <- sum(HIAWasteData.4km$AC)
DALYs.4km <- sum(HIAWasteData.4km$DALY)
```

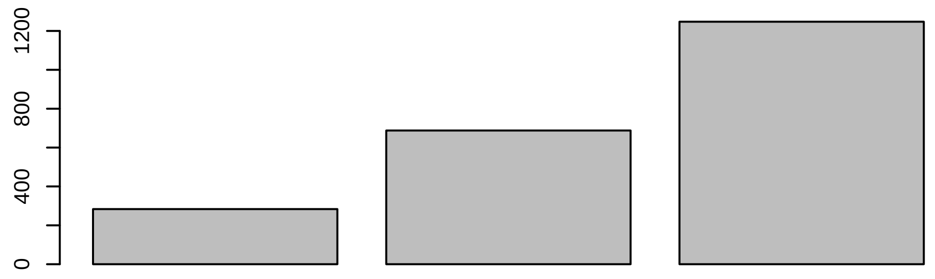
Having obtained the number of attributable cases and DALYs for each of the buffers, we can put the results into a table.

```
# Putting all results into a table
tab1 <- data.frame(Buffer = c('0-2km','0-3km','0-4km'),
                   ACs    = c(ACs.2km,ACs.3km,ACs.4km),
                   DALYs  = c(DALYs.2km,DALYs.3km, DALYs.4km))

# Printing tab1
tab1
  Buffer       ACs    DALYs
1  0-2km  283.5344  2392.35
2  0-3km  687.8271  5803.61
3  0-4km 1247.3829 10524.92
```

We might also want to present this information graphical in the form of a barchart. For this we are going to use the ggplot2 package. Using the geom_bar() argument tells R that you want the result to be a barchart. Note, that the ggplot() function produces nice graphics although its syntax is a bit harder to learn that the original R command for a barchart.

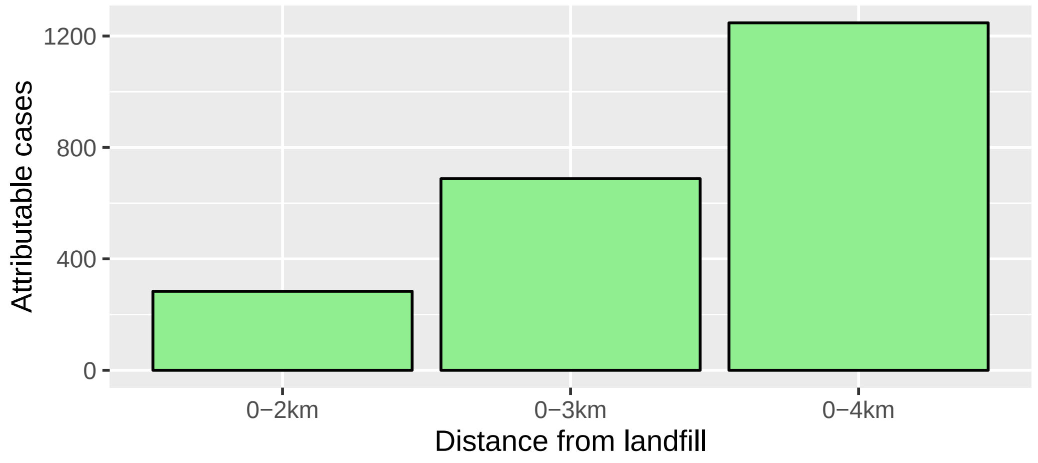
Here is an example of using the barplot() command to show attributable cases.

```
barplot(tab1$AC)
```

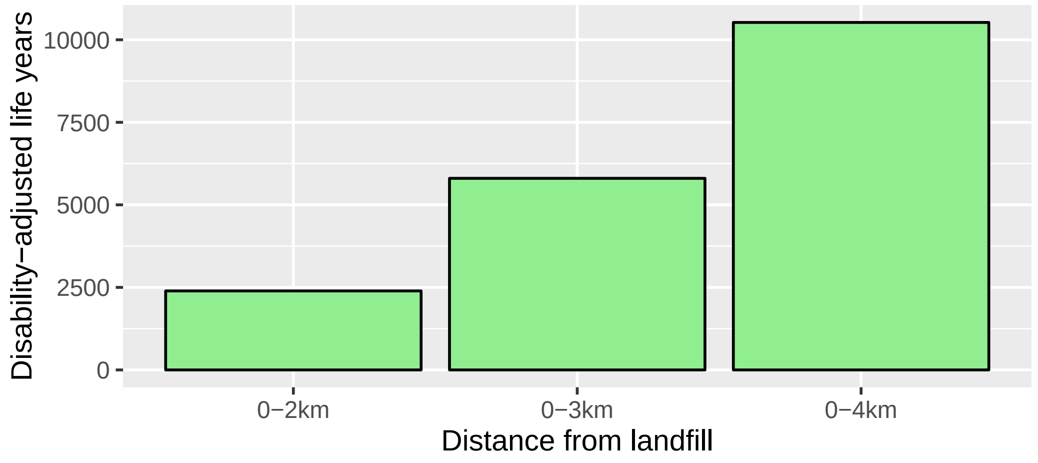


And here is the ggplot2 version and also one for DALYs.

```
# Creating a bar chart for attributable cases
ggplot(tab1, aes(x=Buffer, y=ACs)) +
  geom_bar(stat = "identity", fill="lightgreen", colour='black') +
  ylab('Attributable cases')+
  xlab('Distance from landfill')
```



```r
# Creating a bar chart for disability-adjusted life year
ggplot(tab1, aes(x=Buffer, y=DALYs)) +
  geom_bar(stat = "identity", fill="lightgreen", colour='black') +
  ylab('Disability-adjusted life years')+
  xlab('Distance from landfill')
```

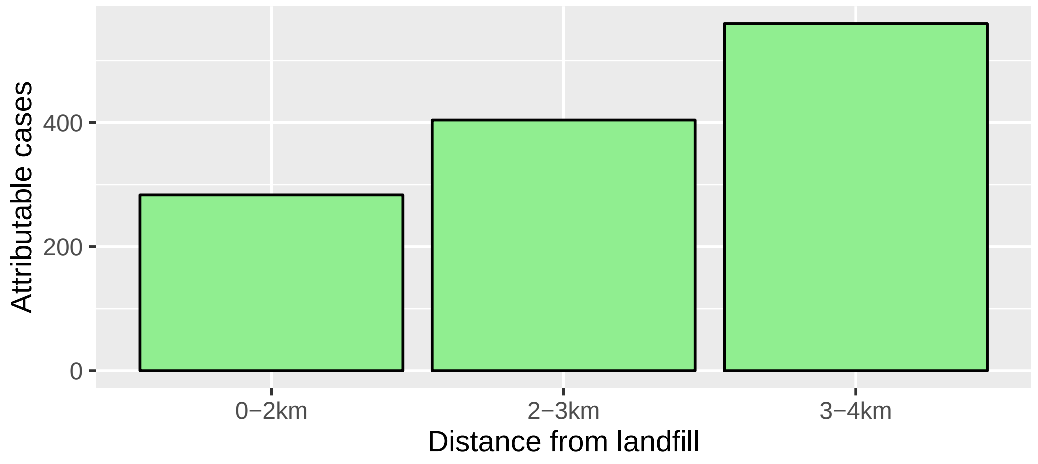


Note that the buffers 0-2km, 0-3km and 0-4km are overlapping and we might want to present the results for separate distance bands, e.g. 0-2km, 2-3km, 3-4km. We will do this by finding the differences between the rows of the `tab1` table and we put the result into a new table, `tab2`.

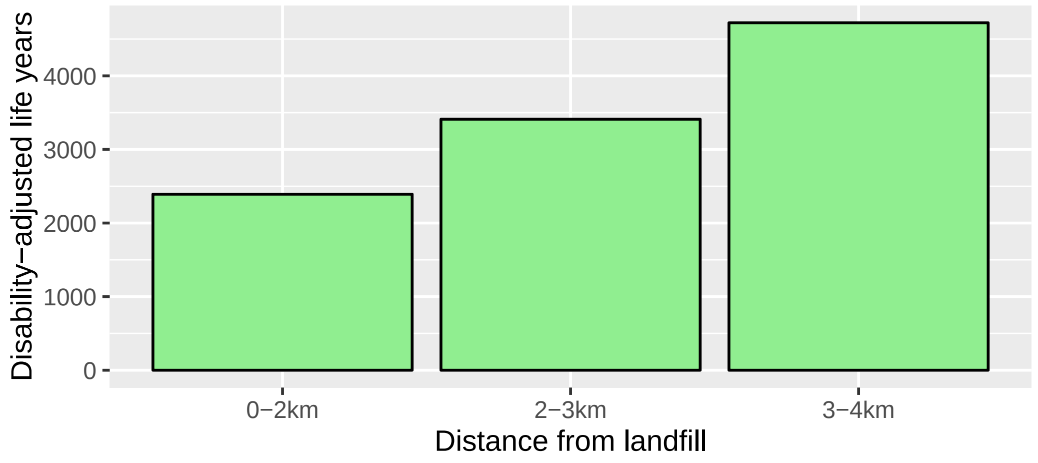
```r
# Putting all results into a table with different buffers
tab2 <- data.frame(Buffer = c('0-2km','2-3km','3-4km'),
                   ACs    = c(ACs.2km,   ACs.3km - ACs.2km,     ACs.4km - ACs.3km),
                   DALYs  = c(DALYs.2km, DALYs.3km - DALYs.2km, DALYs.4km - DALYs.3km))

# Printing tab2
tab2
  Buffer      ACs    DALYs
1  0-2km 283.5344 2392.350
2  2-3km 404.2927 3411.260
3  3-4km 559.5557 4721.308
```

We can now present this information in barcharts as before by simply replacing `tab1` with `tab2` in the `ggplot` code from before.

```r
# Creating a bar chart for attributable cases
ggplot(tab2, aes(x=Buffer, y=ACs)) +
  geom_bar(stat = "identity", fill="lightgreen", colour='black') +
  ylab('Attributable cases')+
  xlab('Distance from landfill')
```



```r
ggplot(tab2, aes(x=Buffer, y=DALYs)) +
  geom_bar(stat = "identity", fill="lightgreen", colour='black') +
  ylab('Disability-adjusted life years')+
  xlab('Distance from landfill')
```



Activities

- Repeat this analysis for 'Congenital Abnormalities'.
- Repeat this analysis for 'Respiratory Diseases'. Note that you should alter the population exposed as respiratory diseases affect more than just newborns.

Closing your R session

When closing down R, you will be asked whether you want to save your R workspace. Your R workspace contains all the data and plots that you have created. At the end of an R session, you can save the current workspace and then everything will automatically reload when you reopen R.