

Introduction to R

Matthew Suderman

Lecturer in Epigenetic Epidemiology



MRC Integrative
Epidemiology
Unit



Things you can do with R

Description	R tool
Remember data	Variables
Calculate	Maths functions
Store and manipulate sequences of values	Vectors and lists
Calculate with vectors	Vector operations
Generate data	Vector-generating functions
Store and manipulate matrices	Matrices
Calculate with matrices	Matrix functions
Store and manipulate datasets	Data frames
Handle missing data	NA
Save and load data	Files
Statistical analyses	Statistics functions
Visualise data	Plots
Make decisions	If/else statements
Create and apply recipes	Functions
Repeatedly apply recipes	Loops and apply functions
Share	Packages

Getting started

what, why, downloading, opening, using, rstudio, help, errors

Getting started: what

- R is a statistical programming language (based on S)
- R is open source - researchers develop packages to implement new statistical methods, plots or applications
- R runs on Windows, MacOS and UNIX



John Chambers



Dirk Eddelbuettel

Getting started: why

... rather than S, STATA, SPSS, etc

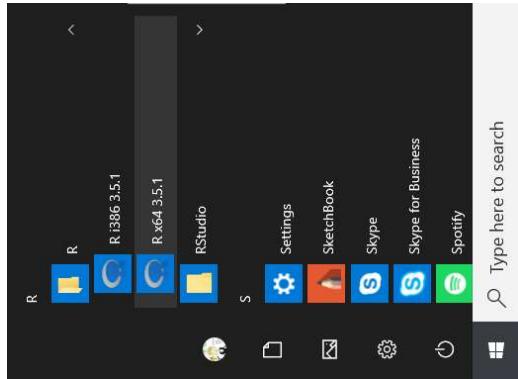
- R is free!
- R is flexible
- R is good at handling large datasets and multiple objects
- R has good plotting tools and packages for statistical analysis

Getting started: installing

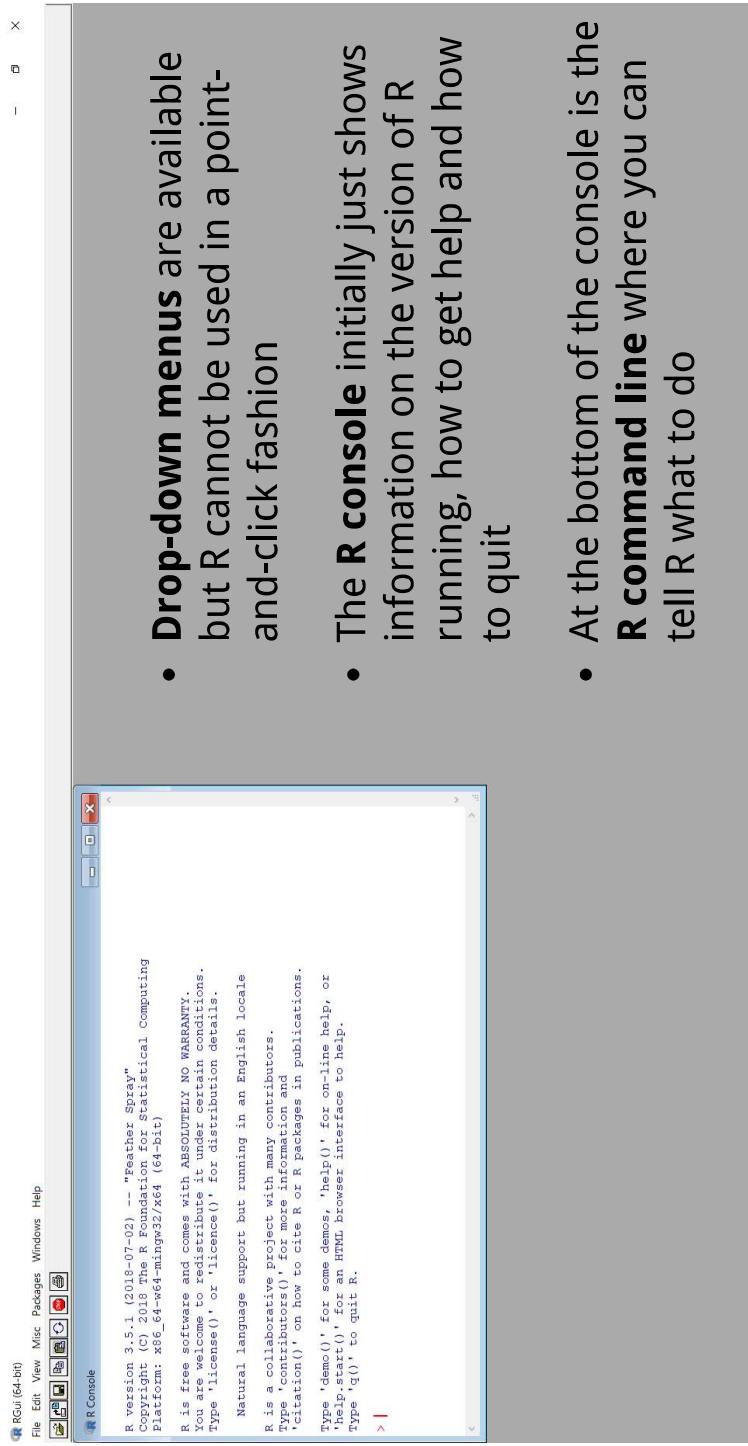
- Visit <https://www.r-project.org> and click "Download R"
- Choose your nearest CRAN mirror (such as <http://www.stats.bris.ac.uk/R>)
- Choose "Download R for [Windows/Mac/Linux]"
- Choose "base" for Windows and click on "Download R 3.5.2 for Windows"
- Choose "R-3.5.2.pkg" for Mac
- Once the .exe (Windows) or .pkg (Mac) have downloaded, run and install

Getting started: opening

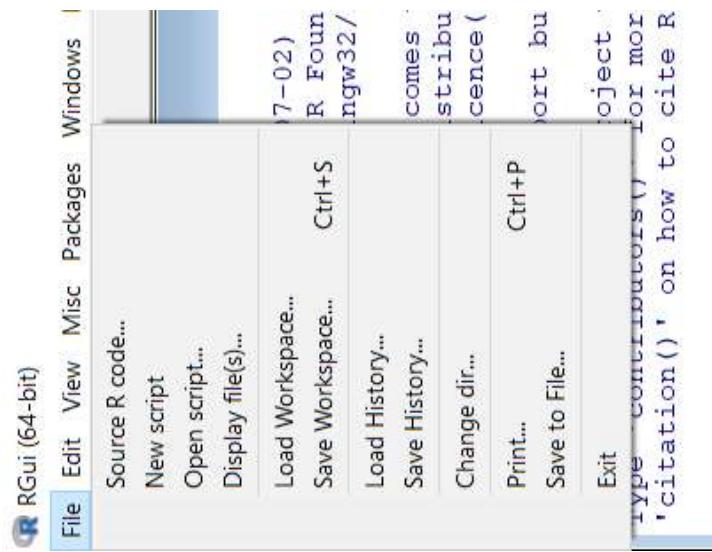
- Click "Start" | "All Programs" | "R" | "R x64 3.6.0"



Getting started: navigating



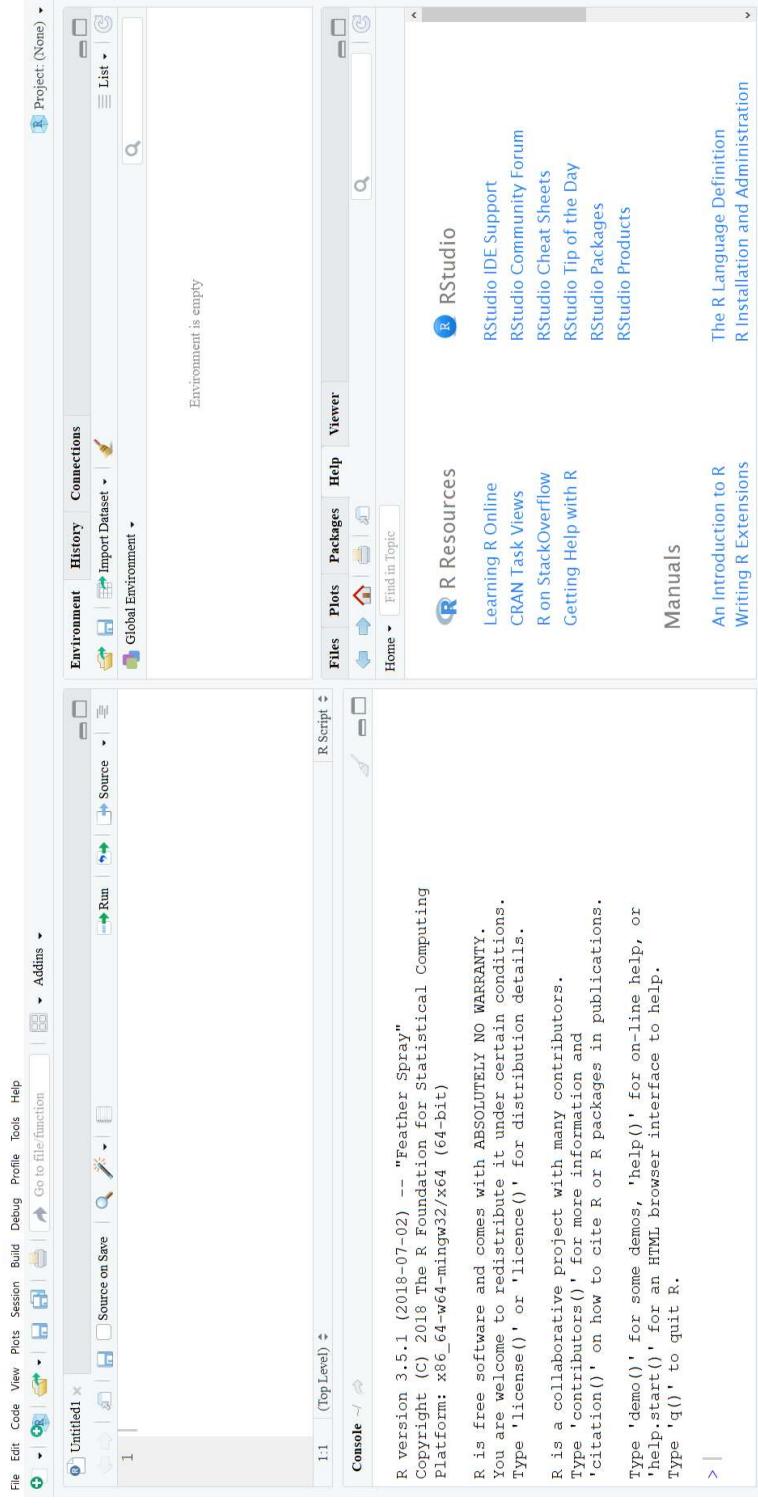
Getting started: menus



- A **script** is sequence of R commands. Creating or opening a script will open the script in a text editor.
- A **workspace** includes all the objects which are in R's memory at a given time. These can include data or results. To see what is in the current workspace type `ls()`.
- The **history** includes all the commands which are displayed during an R session

Getting started: RStudio

- RStudio makes R easier to use
- It includes a code editor, debugging & visualization tools
- <https://www.rstudio.com>



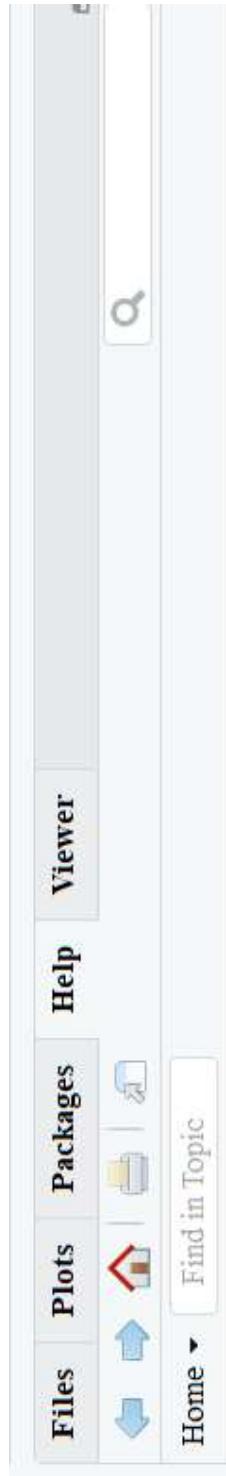
Getting started: RStudio script window



In the script window, you can **save a script** (disk image), **run selected commands** ("Run") and **run** the entire script ("Script").

Getting started: help!

- Use the command `help(topic)` or `?topic`
- In R this will open a web browser
- In RStudio this will open the Help tab



The screenshot shows the RStudio interface with the Help tab selected in the top navigation bar. The navigation bar also includes Files, Plots, Packages, and Viewer. Below the navigation bar are several icons: a blue downward arrow, a red upward arrow, a house icon, a blue square, and a magnifying glass. The main content area displays the R logo and the text "R Resources". Below this, there is a list of links:

- [Learning R Online](#)
- [CRAN Task Views](#)
- [R on StackOverflow](#)
- [Getting Help with R](#)

On the right side of the interface, there is a sidebar with the RStudio logo and the text "RStudio". Below this, there is a list of links:

- [RStudio IDE Support](#)
- [RStudio Community Forum](#)
- [RStudio Cheat Sheets](#)
- [RStudio Tip of the Day](#)
- [RStudio Packages](#)
- [RStudio Products](#)

Getting started: help(mean)

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Getting started: help(mean)

```
mean {base}
```

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

Default S3 method:

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole.

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

weightedmean, mean POSIXct, colMeans for row and column means.

Getting started: help(mean)

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

mean(x, ...)

Default S3 method:

mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

Value

If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

weightedmean, mean.POSIXct, colMeans for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Getting started: error messages

Suppose you type the following but variable 'a' has not been created.

```
> 10*a
```

Getting started: error messages

Suppose you type the following but variable 'a' has not been created.

```
> 10*a
```

```
Error: object 'a' not found
```

Getting started: error messages

Suppose you type the following but variable 'a' has not been created.

```
> 10*a
```

```
Error: object 'a' not found
```

R provides informative error messages

Getting started: error messages

Suppose you type the following but variable 'a' has not been created.

```
> 10*a
```

```
Error: object 'a' not found
```

R provides informative error messages



Getting started: www.r-project.org

Note:

- R Project | Search
- Help with R
- Manuals
- FAQs

The screenshot shows the official website for The R Project for Statistical Computing. At the top left is the R logo. To its right is a navigation bar with links: [Home], Download, CRAN, R Project, News, and Help With R. Below the navigation is a section titled "Getting Started" with a brief introduction to R and links for "About R", "Logo", "Contributors", "What's New?", "Reporting Bugs", "Conferences", "Search", "Get Involved", "Mailing Lists", "Developer Pages", and "R Blog". To the right of this is a "News" section with a link to "version 3.5.2 (Eggshell Igloo)" released on 2018-12-20. Further down are sections for "R Foundation" (Board, Members, Donors, Donate) and "Help With R" (Getting Help). On the far right, there is a "News via Twitter" section showing tweets from the R Foundation (@R_Foundation) and userRI 2019 (@useR2019_Conf) with their respective timestamps.

Getting started: forums

- Mailing list archive and forum <http://r.789695.n4.nabble.com>
- Stack Overflow <https://stackoverflow.com>
- Google!

Things you can do with R

*remember, calculate, store, generate, analyse, visualise, save, repeat, apply, decide,
share*

Remember data

A **variable** is a storage location for data with a name.

Store data ...

```
> x <- 3  
> y <- 4.2  
> name <- "abcdef"  
> skip <- TRUE
```

Query data type ...

```
> is.numeric(x)  
[1] TRUE  
  
> is.logical(name)  
[1] FALSE
```

Recall data ...

```
> x  
[1] 3  
> name  
[1] "abcdef"  
> nchar(name)  
[1] 6  
> z  
Error: object 'z' not found
```

Change data type ...

```
> as.character(x)  
[1] "3"  
  
> as.numeric("3.14")  
[1] 3.14  
  
> as.logical("abc")  
[1] NA  
  
> as.logical(x)
```

Calculate

```
> x + y + 0.8  
[1] 8
```

Calculate

```
> x + y + 0.8  
[1] 8
```

```
> 4*x/2  
[1] 6
```

Calculate

```
> x + y + 0.8  
[1] 8
```

```
> 4*x/2  
[1] 6
```

```
> log10(100)  
[1] 100  
> 2^x  
[1] 8  
> sin(pi/2)  
[1] 1  
> ln(exp(1))  
[1] 1  
> sqrt(x^2)  
[1] 3
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

```
> x[3] ## show the third
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

```
> x[3] ## show the third
```

```
[1] 13
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

```
> x[3] ## show the third
```

```
[1] 13
```

```
> length(x)
```

Store and manipulate sequences of values

A **vector** is a sequence of variables of the same type.

```
> x <- c(11,12,13,14,15,16,17,18,19,20)  
> x
```

```
[1] 11 12 13 14 15 16 17 18 19 20
```

```
> x[3] ## show the third
```

```
[1] 13
```

```
> length(x)
```

```
[1] 10
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

```
[1] 12 13 14 16 18 19 20
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

```
[1] 12 13 14 16 18 19 20
```

```
> x[x>13]
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

```
[1] 12 13 14 16 18 19 20
```

```
> x[x>13]
```

```
[1] 14 15 16 17 18 19 20
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

```
[1] 12 13 14 16 18 19 20
```

```
> x[x>13]
```

```
[1] 14 15 16 17 18 19 20
```

```
y <- c("Dog", "Cat")
z <- c(x[1:3], y)
z
```

Store and manipulate sequences of values (continued)

Vectors can be subset and combined.

```
> x[c(1, 5, 7)]
```

```
[1] 11 15 17
```

```
> x[-c(1, 5, 7)]
```

```
[1] 12 13 14 16 18 19 20
```

```
> x[x>13]
```

```
[1] 14 15 16 17 18 19 20
```

```
y <- c("Dog", "Cat")
z <- c(x[1:3], y)
z
```

Calculate with vectors

$> x+1$

Calculate with vectors

> x+1

```
[1] 12 13 14 15 16 17 18 19 20 21
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

```
[1] 33 36 39 42 45 48 51 54 57 60
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

```
[1] 33 36 39 42 45 48 51 54 57 60
```

```
> x/2
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

```
[1] 33 36 39 42 45 48 51 54 57 60
```

```
> x/2
```

```
[1] 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

```
[1] 33 36 39 42 45 48 51 54 57 60
```

```
> x/2
```

```
[1] 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
```

```
> log10(x)
```

Calculate with vectors

```
> x+1
```

```
[1] 12 13 14 15 16 17 18 19 20 21
```

```
> 3*x
```

```
[1] 33 36 39 42 45 48 51 54 57 60
```

```
> x/2
```

```
[1] 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
```

```
> log10(x)
```

```
[1] 1.041393 1.079181 1.113943 1.146128 1.176091 1.204120 1.230449 1.255273  
[9] 1.278754 1.301030
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

```
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

```
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
> x*y
```

Calculate with vectors (continued)

```
> y <- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

```
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
> x*y
```

```
[1] 132 156 182 210 240 272 306 342 380 420
```

Calculate with vectors (continued)

```
> y<- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

```
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
> x*y
```

```
[1] 132 156 182 210 240 272 306 342 380 420
```

```
> x/(y-1)
```

Calculate with vectors (continued)

```
> y<- x + 1  
> x + y
```

```
[1] 23 25 27 29 31 33 35 37 39 41
```

```
> x-y
```

```
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

```
> x*y
```

```
[1] 132 156 182 210 240 272 306 342 380 420
```

```
> x/(y-1)
```

```
[1] 1 1 1 1 1 1 1 1 1
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50  
[12] 55 60 65 70 75 80 85 90 95 100
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50  
[12] 55 60 65 70 75 80 85 90 95 100
```

```
> length(a)
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50  
[12] 55 60 65 70 75 80 85 90 95 100
```

```
> length(a)
```

```
[1] 21
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50  
[12] 55 60 65 70 75 80 85 90 95 100
```

```
> length(a)
```

```
[1] 21
```

```
> a <- seq(0, 100, length=5)  
> a
```

Generate data: seq()

```
> a <- seq(0, 100, by=5)  
> a
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50  
[12] 55 60 65 70 75 80 85 90 95 100
```

```
> length(a)
```

```
[1] 21
```

```
> a <- seq(0, 100, length=5)  
> a
```

```
[1] 0 25 50 75 100
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

```
[1] 1 1 1 1 1
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

```
[1] 1 1 1 1 1
```

```
> b <- rep(1, 5)  
> b
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

```
[1] 1 1 1 1 1
```

```
> b <- rep(1, 5)  
> b
```

```
[1] 1 1 1 1 1
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

```
[1] 1 1 1 1 1
```

```
> b <- rep(1, 5)  
> b
```

```
[1] 1 1 1 1 1
```

```
> rep(c(1,2,3),4)
```

Generate data: rep()

```
> b <- c(1, 1, 1, 1, 1)  
> b
```

```
[1] 1 1 1 1 1
```

```
> b <- rep(1, 5)  
> b
```

```
[1] 1 1 1 1 1
```

```
> rep(c(1,2,3),4)
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```

Store and manipulate matrices

A **matrix** is a set of elements laid out in rows and columns.

```
> x <- rep(1, 5)
> y <- c("A","B","A","B","C")
> z <- matrix(c(x,y), nrow=5)
> z
```

Store and manipulate matrices

A **matrix** is a set of elements laid out in rows and columns.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> z <- matrix(c(x, y), nrow=5)
> z
```

```
[1][,2]
[1] "1" "A"
[2] "1" "B"
[3] "1" "A"
[4] "1" "B"
[5] "1" "C"
```

Store and manipulate matrices

A **matrix** is a set of elements laid out in rows and columns.

```
> x <- rep(1, 5)
> y <- c("A","B","A","B","C")
> z <- matrix(c(x,y), nrow=5)
> z
```

```
[1][,2]
[1] "1" "A"
[2] "1" "B"
[3] "1" "A"
[4] "1" "B"
[5] "1" "C"
```

```
> dim(z) ## dimensions
```

Store and manipulate matrices

A **matrix** is a set of elements laid out in rows and columns.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> z <- matrix(c(x, y), nrow=5)
> z
```

```
[1] [2]
[1] "1" "A"
[2] "1" "B"
[3] "1" "A"
[4] "1" "B"
[5] "1" "C"
```

```
> dim(z) ## dimensions
```

```
[1] 5 2
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

```
[1] [2] [3] [4] [5]  
[1,] "1" "1" "1" "1" "1"  
[2,] "A" "B" "A" "B" "C"
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

```
[1] [2] [3] [4] [5]  
[1,] "1" "1" "1" "1" "1"  
[2,] "A" "B" "A" "B" "C"
```

```
> cbind(0:4, z)
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

```
[1] [2] [3] [4] [5]  
[1,] "1" "1" "1" "1" "1"  
[2,] "A" "B" "A" "B" "C"
```

```
> cbind(0:4, z)
```

```
[1,] "0" "1" "A"  
[2,] "1" "1" "B"  
[3,] "2" "1" "A"  
[4,] "3" "1" "B"  
[5,] "4" "1" "C"
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

```
[1] [2] [3] [4] [5]  
[1,] "1" "1" "1" "1" "1"  
[2,] "A" "B" "A" "B" "C"
```

```
> cbind(0:4, z)
```

```
[1,] "0" "1" "A"  
[2,] "1" "1" "B"  
[3,] "2" "1" "A"  
[4,] "3" "1" "B"  
[5,] "4" "1" "C"
```

```
> rbind(c("0", "X"), z)
```

Store and manipulate matrices (continued)

```
> t(z) ## transpose
```

```
[1] [2] [3] [4] [5]  
[1,] "1" "1" "1" "1" "1"  
[2,] "A" "B" "A" "B" "C"
```

```
> cbind(0:4, z)
```

```
[1,] "0" "1" "A"  
[2,] "1" "1" "B"  
[3,] "2" "1" "A"  
[4,] "3" "1" "B"  
[5,] "4" "1" "C"
```

```
> rbind(c("0", "X"), z)
```

```
[1,] "0" "X"  
[2,] "1" "A"  
[3,] "1" "B"  
[4,] "2" "C"  
[5,] "3" "D"
```

Store and manipulate matrices (continued)

$> z[1]$

Store and manipulate matrices (continued)

```
> z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

```
[1] "A"
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

```
[1] "A"
```

```
>z[2,]
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

```
[1] "A"
```

```
>z[2,]
```

```
[1] "1" "B"
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

```
[1] "A"
```

```
>z[2,]
```

```
[1] "1" "B"
```

```
>z[c(1,3),2]
```

Store and manipulate matrices (continued)

```
>z[1]
```

```
[1] "1" "1" "1" "1" "1" "1"
```

```
>z[1,2]
```

```
[1] "A"
```

```
>z[2,]
```

```
[1] "1" "B"
```

```
>z[c(1,3),2]
```

```
[1] "A" "A"
```

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

x	y
r1	"A"
r2	"B"
r3	"A"
r4	"B"
r5	"C"

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

x	y
r1 "1" "A"	
r2 "1" "B"	
r3 "1" "A"	
r4 "1" "B"	
r5 "1" "C"	

```
> z[1,2]
" A "
```

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

x	y
r1	"A"
r2	"B"
r3	"A"
r4	"B"
r5	"C"

```
> z[1,2]
"A"
```

```
> z["x","c2"]
"A"
```

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

```
x  y
r1 "1" "A"
r2 "1" "B"
r3 "1" "A"
r4 "1" "B"
r5 "1" "C"
```

```
> z[1,2]
" A "
```

```
> z["x","c2"]
" A "
```

```
> z[,y]
```

Store and manipulate matrices (continued)

```
> colnames(z) <- c("x", "y")
> rownames(z) <- c("r1", "r2", "r3", "r4", "r5")
> z
```

```
x  y
r1 "1" "A"
r2 "1" "B"
r3 "1" "A"
r4 "1" "B"
r5 "1" "C"
```

```
> z[1,2]
" A "
```

```
> z["x","c2"]
" A "
```

```
> z[,y]
```

Calculate with matrices

```
> m <- rbind(c(1,2,3),  
+ c(3,2,1),  
+ c(5,6,4))  
  
> m  
[1,] 1 2 3  
[2,] 3 2 1  
[3,] 5 6 4
```

Calculate with matrices

```
> m <- rbind(c(1,2,3),  
+               c(3,2,1),  
+               c(5,6,4))
```

```
> m
```

```
[1,] 1 2 3  
[2,] 3 2 1  
[3,] 5 6 4
```

```
> m + 1
```

```
[1,] 2 3 4  
[2,] 4 3 2  
[3,] 6 7 5
```

Calculate with matrices

```
> m <- rbind(c(1,2,3),  
+               c(3,2,1),  
+               c(5,6,4))
```

```
> m
```

```
[1,] 1 2 3  
[2,] 3 2 1  
[3,] 5 6 4
```

```
> m + 1
```

```
[1,] 2 3 4  
[2,] 4 3 2  
[3,] 6 7 5
```

```
> m %*% solve(m) ## multiple m by its inverse
```

```
[1,] [2,] [3]  
[1,] 1 -2.220446e-16 2.220446e-16  
[2,] 0 1.000000e+00 2.220446e-16  
[3,] 0 0.000000e+00 1.000000e+00
```

Store and manipulate datasets: lists

A **list** is like a vector, but it can contain elements of different types.

```
> y <- diag(2)
> w <- list(x=1, y=y, z="abc")
```

Store and manipulate datasets: lists

A **list** is like a vector, but it can contain elements of different types.

```
> y <- diag(2)
> w <- list(x=1, y=y, z="abc")
```

```
w$y
```

Store and manipulate datasets: lists

A **list** is like a vector, but it can contain elements of different types.

```
> y <- diag(2)
> w <- list(x=1, y=y, z="abc")
```

```
w$y
```

```
[1] [2]
[1,] 1 0
[2,] 0 1
```

```
names(r)
```

Store and manipulate datasets: lists

A **list** is like a vector, but it can contain elements of different types.

```
> y <- diag(2)
> w <- list(x=1, y=y, z="abc")
```

```
w$y
```

```
[1] [2]
[1,] 1 0
[2,] 0 1
```

```
names(r)
```

```
[1] "x" "y" "z"
```

Store and manipulate datasets: data frames

A **data frame** is like a matrix but the columns can be different types. You could think of it as a list of vectors each of the same length.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> d <- data.frame(a=x, b=y)
```

Store and manipulate datasets: data frames

A **data frame** is like a matrix but the columns can be different types. You could think of it as a list of vectors each of the same length.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> d <- data.frame(a=x, b=y)
```

```
> d[2,]
  a   b
2 1 B
```

Store and manipulate datasets: data frames

A **data frame** is like a matrix but the columns can be different types. You could think of it as a list of vectors each of the same length.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> d <- data.frame(a=x, b=y)
```

```
> d[2,]
  a   b
2 1 B
```

```
> d$a
[1] 1 1 1 1 1
```

Store and manipulate datasets: data frames

A **data frame** is like a matrix but the columns can be different types. You could think of it as a list of vectors each of the same length.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> d <- data.frame(a=x, b=y)
```

```
> d[2,]
  a   b
2 1   B
```

```
> d$a
[1] 1 1 1 1 1
```

```
> class(d$a)
[1] "numeric"
```

Store and manipulate datasets: data frames

A **data frame** is like a matrix but the columns can be different types. You could think of it as a list of vectors each of the same length.

```
> x <- rep(1, 5)
> y <- c("A", "B", "A", "B", "C")
> d <- data.frame(a=x, b=y)
```

```
> d[2,]
  a   b
2 1   B
```

```
> d$a
[1] 1 1 1 1 1
```

```
> class(d$a)
[1] "numeric"
```

```
> class(d$b)
[1] "character"
```

`stop("Time for a break")`

`We will resume after Sys.sleep(10*60).`



Handle missing data

```
> y <- c("A","B","A","B","C")
> y[2] <- 2
> y[3] <- NA
> y
[1] "A" "2" NA "A" "B" "C"
```

Handle missing data

```
> y <- c("A","B","A","B","C")
> y[2] <- 2
> y[3] <- NA
> y
[1] "A" "2" NA "A" "B" "C"
```

```
> is.na(y)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
```

Handle missing data

```
> y <- c("A","B","A","B","C")
> y[2] <- 2
> y[3] <- NA
> y
[1] "A" "2" NA "A" "B" "C"
```

```
> is.na(y)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
```

```
> is.na(y[3])
[1] TRUE
```

Handle missing data

```
> y <- c("A","B","A","B","C")
> y[2] <- 2
> y[3] <- NA
> y
[1] "A" "2" NA "A" "B" "C"
```

```
> is.na(y)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
```

```
> is.na(y[3])
[1] TRUE
```

```
> na.omit(y)
[1] "A" "2" "A" "B" "C"
```

Save and load data: working directory

To open files, you will need to tell R where the files are. To do this, it helps to know where R will start looking. This is called the 'working directory'.

```
> getwd()  
[1] "C:/"
```

Save and load data: working directory

To open files, you will need to tell R where the files are. To do this, it helps to know where R will start looking. This is called the 'working directory'.

```
> getwd()  
[1] "C:/"
```

You can change this using setwd.

```
> setwd("O:/Documents")
```

The working directory can also be chosen using the Session menu in RStudio.

Save and load data: reading csv files

Suppose I have a spreadsheet stored in CSV format.

```
"id","age","sex","diet","bmi"  
1,32,"M",0,25.7957231474661  
2,35,"M",0,28.8952139451377  
3,41,"M",0,29.9258448186199  
4,29,"M",0,27.3383500990741  
5,33.5,"M",1,28.2469210985821  
6,33.2,"M",1,27.0473176810354  
7,32.9,"M",1,30.3031786852156  
8,32.6,"F",1,28.5621419729205  
9,32.3,"F",1,28.05344654591  
10,32,"F",1,28.8864676350323  
...
```

Save and load data: reading CSV files

Suppose I have a spreadsheet stored in CSV format.

read.csv will read csv files and save them as data frames.

```
"id","age","sex","diet","bmi"  
1,32,"M",0,25.7957231474661  
2,35,"M",0,28.8952139451377  
3,41,"M",0,29.9258448186199  
4,29,"M",0,27.3383500990741  
5,33.5,"M",1,28.2469210985821  
6,33.2,"M",1,27.0473176810354  
7,32.9,"M",1,30.3031786852156  
8,32.6,"F",1,28.5621419729205  
9,32.3,"F",1,28.05344654591  
10,32,"F",1,28.8864676350323
```

```
> dat <- read.csv("bmi.csv")
```

```
> dat[1:5,]  
   id age sex diet  bmi  
1  1 32.0 M  0 25.79572  
2  2 35.0 M  0 28.89521  
3  3 41.0 M  0 29.92584  
4  4 29.0 M  0 27.33835  
5  5 33.5 M  1 28.24692
```

```
> dat$bmi[1:5]  
[1] 25.79572 28.89521 29.92584 27.33835 28.22  
> dat[5,]  
   id age sex diet  bmi  
5  5 33.5 M  1 28.24692
```

...

Save and load data: writing csv files

`write.csv` saves a data frame as a csv file.

```
> dat.corrected <- dat  
> dat.corrected$sex[5] <- "F" ## make correction  
> write.csv(dat.corrected, "bmi-corrected.csv", row.names=F, quote=F)
```

`write.table` is similar but allows the user to change the column separator character.
Here we separate columns by a semicolon rather than a comma.

```
> write.table(dat, "bmi-corrected.csv", row.names=F, quote=F, sep=";")
```

Save and load data: writing csv files

write.csv saves a data frame as a csv file.

```
> dat.corrected <- dat  
> dat.corrected$sex[5] <- "F" ## make correction  
> write.csv(dat.corrected, "bmi-corrected.csv", row.names=F, quote=F)
```

write.table is similar but allows the user to change the column separator character.
Here we separate columns by a semicolon rather than a comma.

```
> write.table(dat, "bmi-corrected.csv", row.names=F, quote=F, sep=";")
```

Note: There is similarly a function read.table()
just like read.csv() but more flexible.

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761
```

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761
```

```
> median(dat$bmi)    ## median  
[1] 27.58069
```

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761  
  
> median(dat$bmi)    ## median  
[1] 27.58069  
  
> sd(dat$bmi)        ## standard deviation  
[1] 1.47353
```

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761
```

```
> median(dat$bmi)    ## median  
[1] 27.58069
```

```
> sd(dat$bmi)        ## standard deviation  
[1] 1.47353
```

```
> min(dat$bmi)       ## min  
[1] 25.34356
```

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761  
  
> median(dat$bmi)    ## median  
[1] 27.58069  
  
> sd(dat$bmi)        ## standard deviation  
[1] 1.47353  
  
> min(dat$bmi)       ## min  
[1] 25.34356  
  
> quantile(dat$bmi, probs=0.25) ## first quartile  
25%  
26.60918
```

Statistical analyses: basic summaries

```
> mean(dat$bmi)      ## mean  
[1] 27.60761  
  
> median(dat$bmi)    ## median  
[1] 27.58069  
  
> sd(dat$bmi)        ## standard deviation  
[1] 1.47353  
  
> min(dat$bmi)       ## min  
[1] 25.34356  
  
> quantile(dat$bmi, probs=0.25) ## first quartile  
25%  
26.60918  
  
> table(dat$sex)     ## frequencies  
F M  
13 7
```

Statistical analyses: dataset summaries

The summary function can be used to summarize single variables

```
> summary(dat$bmi)
   Min. 1st Qu. Median Mean 3rd Qu. Max.
25.34 26.61 27.58 27.61 28.64 30.30
```

Statistical analyses: dataset summaries

The summary function can be used to summarize single variables

```
> summary(dat$bmi)
Min. 1st Qu. Median Mean 3rd Qu. Max.
25.34 26.61 27.58 27.61 28.64 30.30
```

or entire datasets.

```
> summary(dat)
   id    age     sex   diet    bmi
Min. :1.00  Min. :29.00  Length:20  Min. :0.00  Min. :25.34
1st Qu.:5.75  1st Qu.:30.12  Class :character  1st Qu.:0.00  1st Qu.:26.61
Median :10.50  Median :31.55  Mode :character  Median :1.00  Median :27.58
Mean  :10.50  Mean  :31.85  NA's :1          Mean  :0.55  Mean  :27.61
3rd Qu.:15.25 3rd Qu.:32.67
Max. :20.00  Max. :41.00
```

Statistical analyses: evaluating associations

There is a fairly strong association between BMI and age.

```
> cor(dat$bmi, dat$age)  
[1] 0.5705649
```

Here is the association.

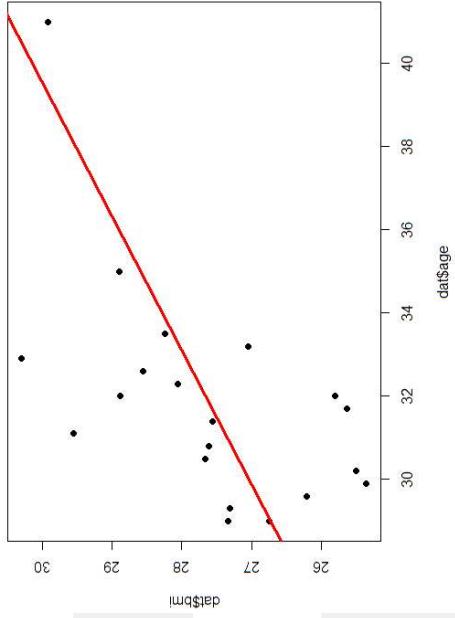
```
> fit <- lm(bmi ~ age, data = dat)  
> fit  
Call:  
lm(formula = bmi ~ age, data = dat)  
  
Coefficients:  
(Intercept)      age  
17.6770        0.3118
```

Statistical analyses: evaluating associations

There is a fairly strong association between BMI and age.

```
> cor(dat$bmi, dat$age)  
[1] 0.5705649
```

Here is the association.



```
> fit <- lm(bmi~age, data=dat)  
> fit  
Call:
```

```
lm(formula = bmi ~ age, data = dat)
```

Coefficients:
(Intercept) age
17.6770 0.3118

The **regression line** in the plot runs from (29, 17.6770 + 0.3118*29)=(29, 26.7192) to (41, 17.6770 + 0.3118*41)= (41, 30.4608).

Statistical analyses: summarizing regression model fits

```
> summary(fit)

Call:
lm(formula = bmi ~ age, data = dat)

Residuals:
    Min     1Q Median     3Q    Max 
-1.9334 -0.7760  0.2152  0.5293 2.3682 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 17.6770   3.3805  5.229 5.67e-05 ***
age         0.3118   0.1058  2.948  0.00861 **  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.243 on 18 degrees of freedom
Multiple R-squared:  0.3255, Adjusted R-squared:  0.2881 
F-statistic: 8.688 on 1 and 18 DF, p-value: 0.008612
```

Statistical analyses: fitting multiple variable models

```
> fit <- lm(bmi ~ age + sex + diet, data=dat)
```

Statistical analyses: fitting multiple variable models

```
> fit <- lm(bmi ~ age + sex + diet, data=dat)
```

```
> summary(fit)
```

Call:

lm(formula = bmi ~ age + sex + diet, data = dat)

Residuals:

	Min	1Q	Median	3Q	Max
-2.2978	-0.6210	0.0106	0.8208	1.7891	

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept)	18.4784	3.7464	4.932	0.00015	***
age	0.2643	0.1228	2.152	0.04700	*
sexM	0.3512	0.6866	0.512	0.61596	
diet	1.0671	0.5532	1.929	0.07168	.

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.188 on 16 degrees of freedom

Multiple R-squared: 0.4528, Adjusted R-squared: 0.3502

F-statistic: 4.413 on 3 and 16 DF, p-value: 0.01921

Statistical analyses: retrieving model fit elements

```
> coef(fit)
(Intercept)      age     sexM     diet
18.4783563  0.2643460  0.3512052  1.0671192
```

Statistical analyses: retrieving model fit elements

```
> coef(fit)
(Intercept)      age     sexM     diet
18.4783563  0.2643460  0.3512052  1.0671192
```

```
> coef(summary(fit))
Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.4783563 3.7463529 4.9323587 0.0001500352
age          0.2643460 0.1228288 2.1521491 0.0469973143
sexM         0.3512052 0.6865686 0.5115369 0.6159621270
diet         1.0671192 0.5532367 1.9288657 0.0716810178
```

Statistical analyses: retrieving model fit elements

```
> coef(fit)
(Intercept)      age      sexM     diet
18.4783563  0.2643460  0.3512052  1.0671192
```

```
> coef(summary(fit))
Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.4783563 3.7463529 4.9323587 0.0001500352
age          0.2643460 0.1228288 2.1521491 0.0469973143
sexM         0.3512052 0.6865686 0.5115369 0.6159621270
diet         1.0671192 0.5532367 1.9288657 0.0716810178
```

```
> coef(summary(fit))["age","Estimate"]
[1] 0.264346
```

Statistical analyses: retrieving model fit elements

```
> coef(fit)
(Intercept)      age      sexM     diet
18.4783563  0.2643460  0.3512052  1.0671192
```

```
> coef(summary(fit))
Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.4783563 3.7463529 4.9323587 0.0001500352
age          0.2643460 0.1228288 2.1521491 0.0469973143
sexM         0.3512052 0.6865686 0.5115369 0.6159621270
diet         1.0671192 0.5532367 1.9288657 0.0716810178
```

```
> coef(summary(fit))["age","Estimate"]
[1] 0.264346
```

```
> names(summary(fit))
[1] "call"        "terms"       "residuals"
[4] "coefficients" "aliased"    "sigma"
[7] "df"           "r.squared"   "adj.r.squared"
[10] "fstatistic"  "cov.unscaled"
```

Statistical analyses: retrieving model fit elements

```
> coef(fit)
(Intercept)      age      sexM     diet
18.4783563  0.2643460  0.3512052  1.0671192
```

```
> coef(summary(fit))
Estimate Std. Error t value Pr(>|t|)
(Intercept) 18.4783563 3.7463529 4.9323587 0.0001500352
age          0.2643460 0.1228288 2.1521491 0.0469973143
sexM         0.3512052 0.6865686 0.5115369 0.6159621270
diet         1.0671192 0.5532367 1.9288657 0.0716810178
```

```
> coef(summary(fit))["age","Estimate"]
[1] 0.264346
```

```
> names(summary(fit))
[1] "call"        "terms"       "residuals"
[4] "coefficients" "aliased"    "sigma"
[7] "df"          "r.squared"   "adj.r.squared"
[10] "fstatistic"  "cov.unscaled"
```

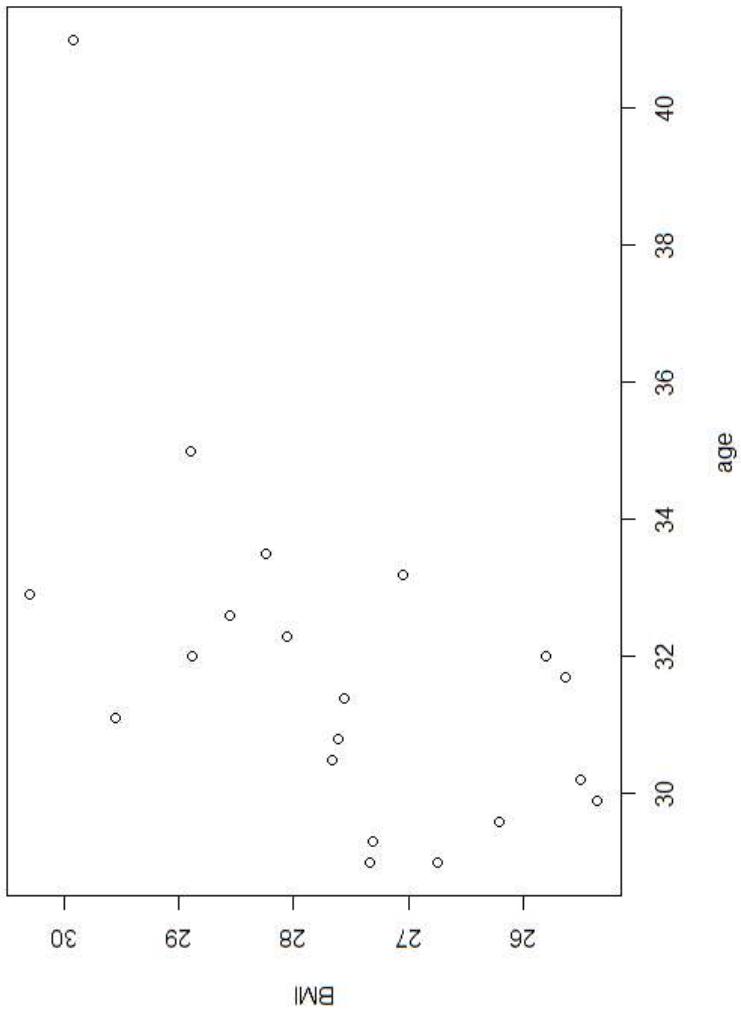
Visualise: scatterplot

```
plot(dat$age, dat$bmi,  
main="age vs BMI",  
xlab="age",  
ylab="BMI")
```

Visualise: scatterplot

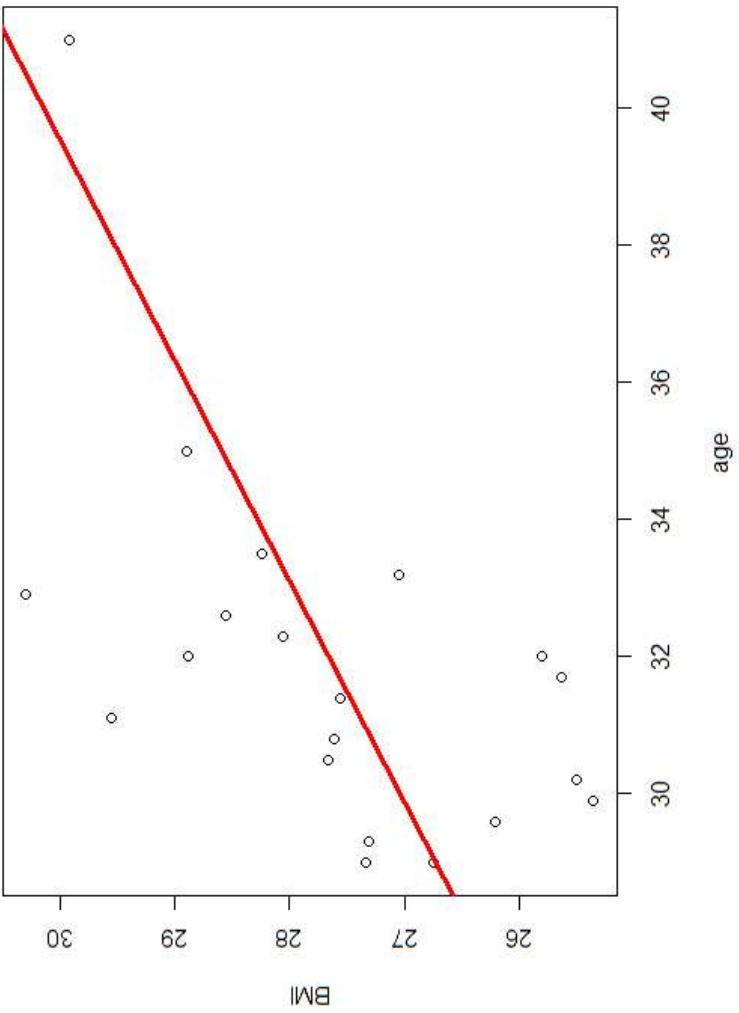
```
plot(dat$age, dat$bmi,  
main="age vs BMI",  
xlab="age",  
ylab="BMI")
```

age vs BMI



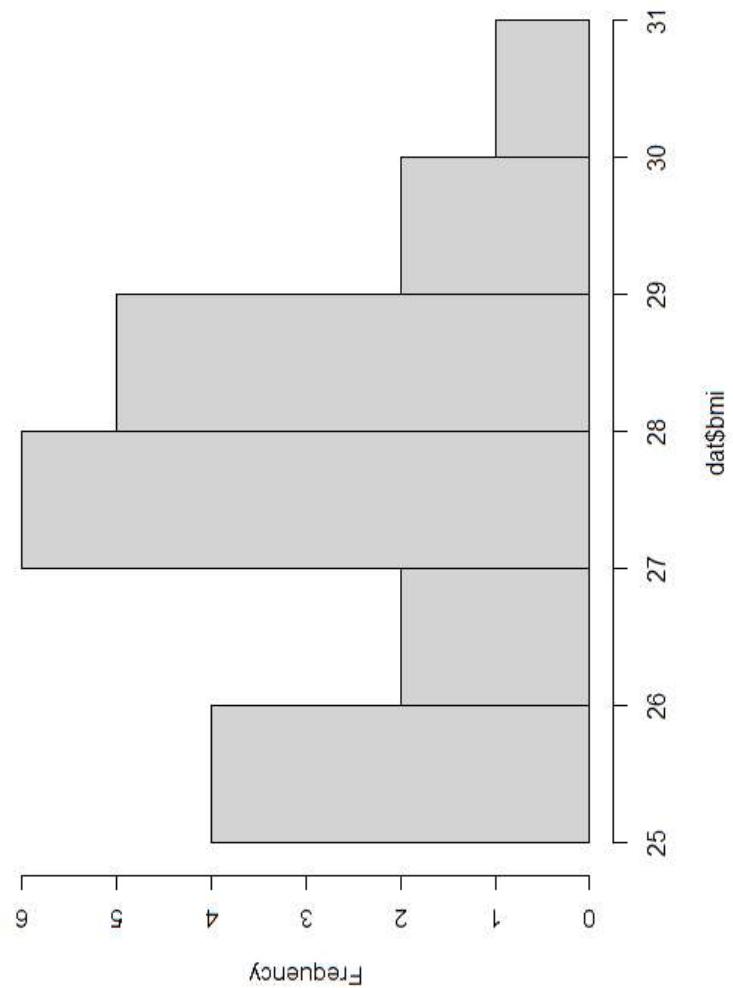
Visualise: scatterplot with regression line

```
plot(dat$age, dat$bmi,  
main="age vs BMI",  
xlab="age",  
ylab="BMI")  
fit <- lm(bmi ~ age,  
data=dat)  
abline(fit, col="red", lwd=3)
```



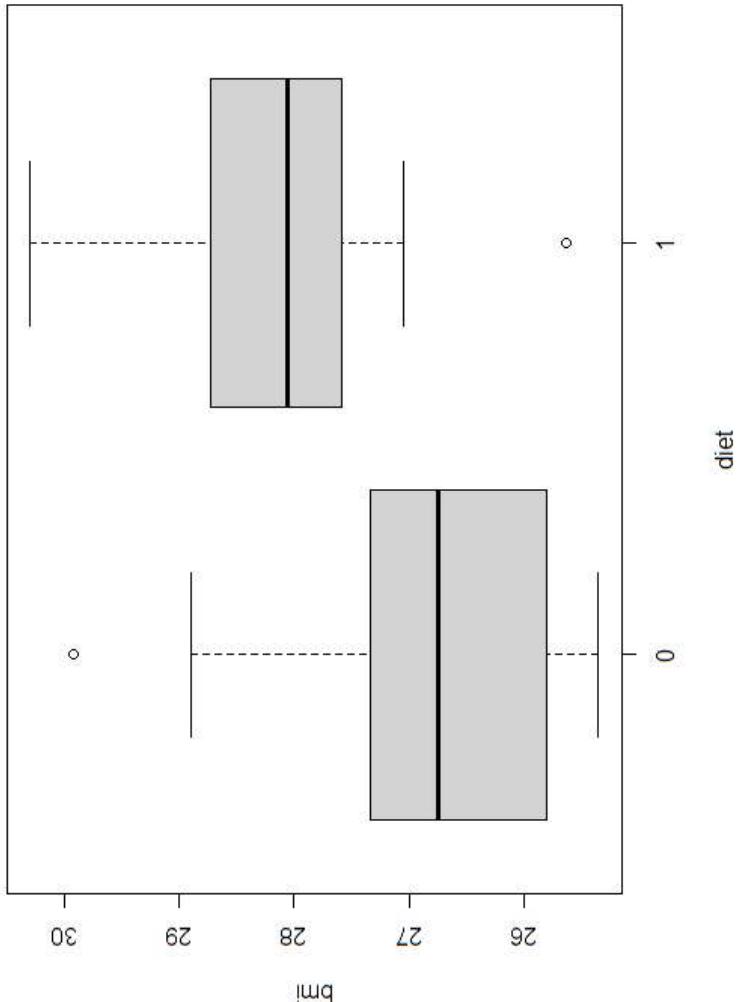
Visualise: histogram

```
hist(dat$bmi, main="BMI")
```

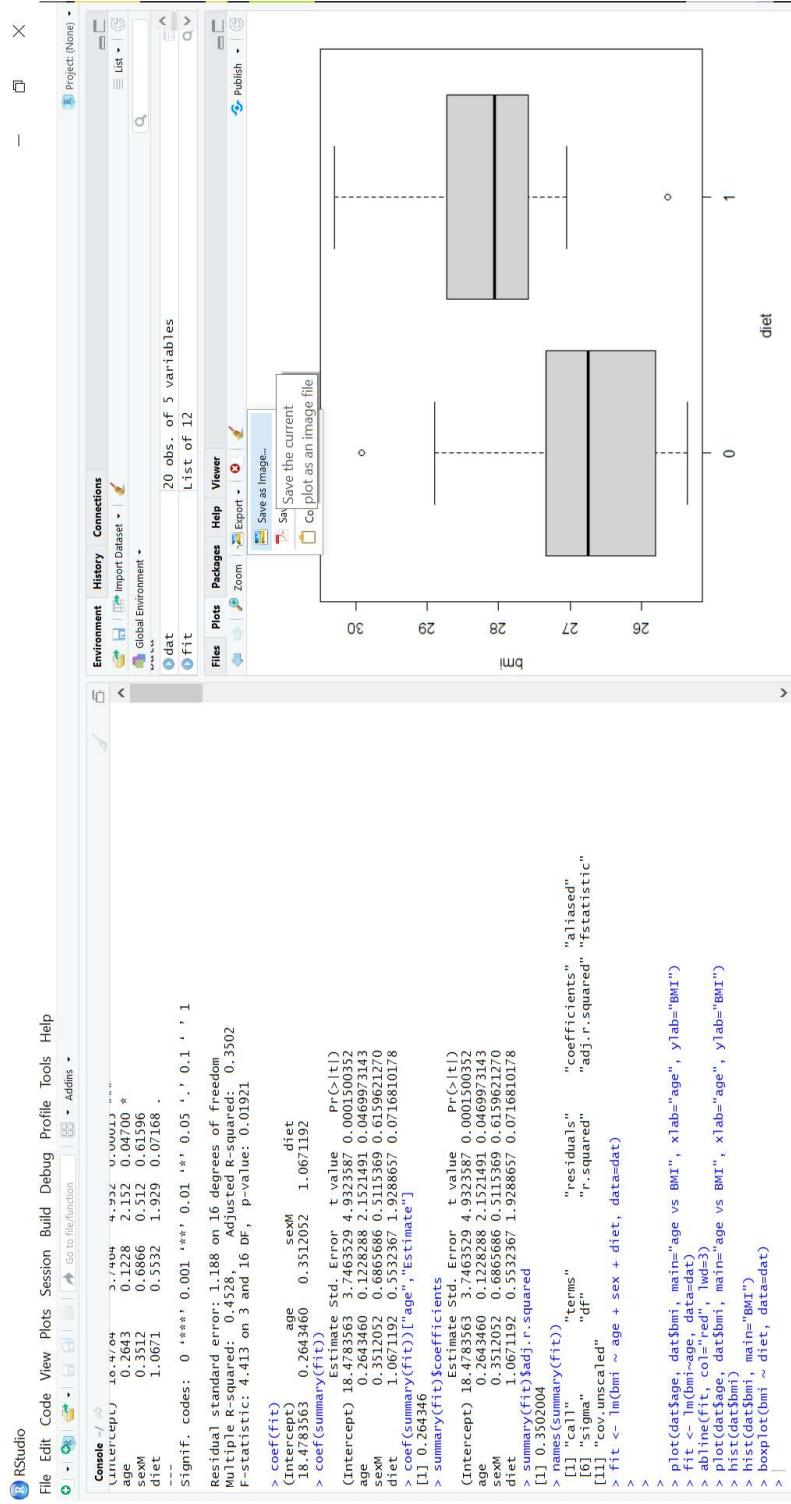


Visualise: boxplot

```
boxplot(bmi ~ diet, data=dat)
```



Visualise: saving



Visualise: saving *like a total maniac*

```
> png(filename="boxplot.png", width=10, height=10, units="cm", res=500)
> boxplot(bmi~diet, data=dat)
> dev.off()
```

Decisions: ask questions

> **x <- 11:15**

Decisions: ask questions

```
> x <- 11:15
```

```
> x[3] == 13          ## 13 equals 13  
[1] TRUE
```

Decisions: ask questions

```
> x <- 11:15  
  
> x[3] == 13          ## 13 equals 13  
[1] TRUE  
  
> x[3] != 12          ## 13 not equal to 12
```

Decisions: ask questions

```
> x <- 11:15  
  
> x[3] == 13      ## 13 equals 13  
[1] TRUE  
  
> x[3] != 12     ## 13 not equal to 12  
  
> x[3] < x[4]    ## 13 less than 14
```

Decisions: ask questions

```
> x <- 11:15  
  
> x[3] == 13          ## 13 equals 13  
[1] TRUE  
  
> x[3] != 12         ## 13 not equal to 12  
  
> x[3] < x[4]        ## 13 less than 14  
  
> x[1] <= x[2] & !x[2] > x[3]  ## 11 <= 12 and 12 not greater than 13
```

Decisions: ask questions

```
> x <- 11:15  
  
> x[3] == 13          ## 13 equals 13  
[1] TRUE  
  
> x[3] != 12         ## 13 not equal to 12  
  
> x[3] < x[4]        ## 13 less than 14  
  
> x[1] <= x[2] & !x[2] > x[3]  ## 11 <= 12 and 12 not greater than 13  
  
> x[1] < x[2] | x[2] > x[3]  ## 11 < 12 or 12 > 13
```

Decisions: ask questions

```
> x <- 11:15

> x[3] == 13          ## 13 equals 13
[1] TRUE

> x[3] != 12         ## 13 not equal to 12

> x[3] < x[4]        ## 13 less than 14

> x[1] <= x[2] & !x[2] > x[3]  ## 11 <= 12 and 12 not greater than 13

> x[1] < x[2] | x[2] > x[3]  ## 11 < 12 or 12 > 13

> x < 12
[1] TRUE FALSE FALSE FALSE
```

Decisions: ask questions

```
> x <- 11:15

> x[3] == 13          ## 13 equals 13
[1] TRUE

> x[3] != 12         ## 13 not equal to 12

> x[3] < x[4]        ## 13 less than 14

> x[1] <= x[2] & !x[2] > x[3]  ## 11 <= 12 and 12 not greater than 13
## 11 < 12 or 12 > 13

> x < 12
[1] TRUE FALSE FALSE FALSE

> all(x >= 11)      ## each value in x >= 11
[1] TRUE
```

Decisions: alternatives

```
if(score > 85) {  
  grade <- "A"  
} else {  
  grade <- "F"  
}
```

Decisions: alternatives

```
if(score > 85) {  
  grade <- "A"  
} else {  
  grade <- "F"  
}
```

```
grade <- ifelse(score > 85, "A", "F")
```

Decisions: alternatives

```
if(score > 85) {  
  grade <- "A"  
} else {  
  grade <- "F"  
}
```

```
grade <- ifelse(score > 85, "A", "F")
```

```
if(score > 85) {  
  grade <- "A"  
} else if(score > 75) {  
  grade <- "B"  
} else {  
  grade <- "F"  
}
```

Decisions: alternatives

```
if(score > 85) {  
  grade <- "A"  
} else {  
  grade <- "F"  
}
```

```
grade <- ifelse(score > 85, "A", "F")
```

```
if(score > 85) {  
  grade <- "A"  
} else if(score > 75) {  
  grade <- "B"  
} else {  
  grade <- "F"  
}
```

```
grade <- ifelse(score > 85, "A", ifelse(score > 75, "B", "F"))
```

Recipes: create and apply

Functions consist of a sequence of commands applied to a set of variables that return some output.

R provides many functions such as `help`, `cor`, `median`.

Recipes: create and apply

Functions consist of a sequence of commands applied to a set of variables that return some output.

R provides many functions such as help, cor, median.

Users can create their own functions for repetitive tasks, e.g. a function for the distance of a point from 0.

```
euclidean.norm <- function(x) {  
  res <- sqrt(sum(x^2))  
  return(res)  
}
```

Recipes: create and apply

Functions consist of a sequence of commands applied to a set of variables that return some output.

R provides many functions such as help, cor, median.

Users can create their own functions for repetitive tasks, e.g. a function for the distance of a point from 0.

```
euclidean.norm <- function(x) {  
  res <- sqrt(sum(x^2))  
  return(res)  
}
```

```
> euclidean.norm(c(3,4))  
[1] 5
```

```
> sqrt(3*3 + 4*4)  
[1] 5
```

```
> euclidean.norm(c(2,3,6))  
[1] 7
```

Recipes: repeat

Given a dataset giving a scores for a group of people:

```
> people <- c("bill",
  "jane",
  "bob",
  "felicia",
  "carl",
  "apple")
> scores <- c(90, 80,
  60, 95,
  75, 99)
> stats <- data.frame(person=people, score=scores)
> stats
   person score
1    bill   90
2   jane   80
3    bob   60
4 felicia   95
5   carl   75
6  apple   99
```

Recipes: repeat

Given a dataset giving a scores for a group of people:

Step 1. Create a function for determining a high score.

```
> people <- c("bill",
  "jane",
  "bob",
  "felicia",
  "carl",
  "apple")
> scores <- c(90, 80,
  60, 95,
  75, 99)
> stats <- data.frame(person=people, score=score)
> stats
  person score
  1    bill   90
  2    jane   80
  3    bob    60
  4 felicia   95
  5    carl   75
  6   apple   99
```

```
is.high <- function(s) {
  ## something really complicated
  ## that takes into account grade
  ## barriers and homework completed
  ## and the name of their first pet
  ## ... well okay, we'll just say
  ## scores above 90 ...
  s > 90
}

> is.high(65)
[1] FALSE
> is.high(94)
[1] TRUE
> is.high(stats$score[2])
[1] FALSE
```

Recipes: repeat (cont)

Step 2. Construct the list of people by applying `is.high()` to each score. Here are 3 equivalent ways:

Option 1. While-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)
i <- 1
while (i <= n) {
  has.high.score[i] <- is.high(stats$score[i])
  i <- i + 1
}
high.scorers <- stats$person[has.high.score]
```

Recipes: repeat (cont)

Step 2. Construct the list of people by applying `is.high()` to each score. Here are 3 equivalent ways:

Option 1. While-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)
i <- 1
while (i <= n) {
  has.high.score[i] <- is.high(stats$score[i])
  i <- i + 1
}
high.scorers <- stats$person[has.high.score]
```

Option 2. For-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)

for (i in 1:n) {
  has.high.score[i] <- is.high(stats$score[i])
}

high.scorers <- stats$person[has.high.score]
```

Recipes: repeat (cont)

Step 2. Construct the list of people by applying `is.high()` to each score. Here are 3 equivalent ways:

Option 1. While-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)
i <- 1
while (i <= n) {
  has.high.score[i] <- is.high(stats$score[i])
  i <- i + 1
}
high.scorers <- stats$person[has.high.score]
```

Option 2. For-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)

for (i in 1:n) {
  has.high.score[i] <- is.high(stats$score[i])
}

high.scorers <- stats$person[has.high.score]
```

Option 3. `apply()`

```
has.high.score <- sapply(stats$score, is.high)
high.scorers <- stats$person[has.high.score]
```

Recipes: repeat (cont)

Step 2. Construct the list of people by applying `is.high()` to each score. Here are 3 equivalent ways:

Option 1. While-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)
i <- 1
while (i <= n) {
  has.high.score[i] <- is.high(stats$score[i])
  i <- i + 1
}
high.scorers <- stats$person[has.high.score]
```

Option 2. For-loop

```
n <- nrow(stats)
has.high.score <- rep(F,n)

for (i in 1:n) {
  has.high.score[i] <- is.high(stats$score[i])
}

high.scorers <- stats$person[has.high.score]
```

Option 3. `apply()`

```
has.high.score <- sapply(stats$score, is.high)
high.scorers <- stats$person[has.high.score]
```

Share: loading packages

An **R package** is a collection of functions and/or data sets created for use by other R users. A **package** can be loaded using library.

```
> library(Hmisc)
```

Attaching package: ‘Hmisc’

Share: loading packages

An **R package** is a collection of functions and/or data sets created for use by other R users. A package can be loaded using library.

```
> library(Hmisc)  
Attaching package: 'Hmisc'
```

After loading, functions and data provided by the package can be used.

```
> describe(c(1,1,2,1,3,5,1,3,5,2,3,4))  
   n missing distinct Info  
 12    0      5  0.944  
Mean   Gmd  
2.583  1.742
```

lowest : 1 2 3 4 5, highest: 1 2 3 4 5

Value	1	2	3	4
Frequency	4	2	3	1
Proportion	0.3333	0.167	0.250	0.083

Value	5
Frequency	2
Proportion	0.167

Share: installing packages

If the package has not been installed, you'll see an error message like this.

```
> library(Hmisc)
Error in library(Hmisc) : there is no package called 'Hmisc'
```

Share: installing packages

If the package has not been installed, you'll see an error message like this.

```
> library(Hmisc)
Error in library(Hmisc) : there is no package called 'Hmisc'
```

The package can be installed using `install.packages`.

```
> install.packages("Hmisc")
```

Installing package into ‘C:/Users/dude/Documents/R/win-library/4.0’
(as ‘lib’ is unspecified)
also installing the dependencies ‘png’, ‘jpeg’, ‘Formula’,
‘latticeExtra’, ‘gridExtra’, ‘htmlTable’, ‘viridis’

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/png_0.1-7.zip'
Content type 'application/zip' length 336667 bytes (328 KB)
downloaded 328 KB

...

package ‘Hmisc’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in

Share: package help

The `help` function can be used to open the documentation for an R package, e.g. `help(package="Hmisc")`

The screenshot shows the R Help Viewer interface. At the top, there's a menu bar with 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu is a toolbar with icons for file operations like Open, Save, Print, and Help. A search bar is present. The main area has tabs for 'R Harrel Miscellaneous' and 'Find in Topic'. The title 'Harrell Miscellaneous' is displayed above the content area, which features a large blue 'R' logo. Below the title, the text reads 'Documentation for package 'Hmisc' version 4.1-1'. It lists several sections: 'DESCRIPTION file.', 'Package NEWS.', 'Help Pages', and 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z misc'. A sidebar on the right contains links for 'Indexes of Absolute Prediction Error for Linear Models', 'Add Marginal Observations', 'Character String Editing and Miscellaneous Character Handling Functions', and 'Check if All Elements in Character Vector are Numeric'.

Search the package list on CRAN
<https://cran.r-project.org/web/packages/Hmisc/>

The screenshot shows the CRAN package page for 'Hmisc'. The page header includes the package name, version (4.4-1), and a brief description: 'Contains many functions useful for data analysis, high-level graphics, utility operations, advanced table making, variable clustering, character string manipulation, etc'. Below the description, it lists dependencies: 'lattice', 'survival (>= 3.1-6)', 'Formula', 'ggplot2 (>= 2.2)', 'methods', 'latticeExtra', 'cluster', 'part', 'onet', 'foreign', 'grid', 'acepack', 'chron', 'rms', 'mice', 'tables', 'knitr', 'plotly (>= 4.5.6)', 'lang', 'I'. It also shows the date of publication (2020-08-10), author (Frank E Harrell Jr), and maintainer (Frank E Harrell Jr <fharrell@ufl.edu>). License information is provided under 'GPL_2 | GPL_3 [expanded from: GPL (>= 2)]'. The URL is 'https://bbiosstat.org/R/Hmisc/'. The page indicates that 'NeedsCompilation: yes', 'Materials: README NEWS ChangeLog', 'In views: Bayesian, ClinicalTrials, Econometrics, MissingData, Multivari', and 'CRAN checks: Hmisc results'. The 'Downloads:' section includes links to the reference manual ('Hmisc.pdf'), package source ('Hmisc_4.4-1.tar.gz'), and Windows binaries ('r-devel: Hmisc_4.4-1.zip, r-release: Hmisc_4.4-1.zip, r-oldrel: H').

Share: learn R with "swirl"

swirl (<http://swirlstats.com/students.html>) allows you to learn R within R itself.



```
> install.packages("swirl")
> library(swirl)
| Hi! Type swirl() when you are ready to begin.
```

Share: creating packages for others

If you have a set of R functions and/or a dataset that you think others might like to use, create a package and put them in it!

Here is a good place to get started:

<https://support.rstudio.com/hc/en-us/articles/200486488-Developing-Packages-with-RStudio>

Acknowledgements

Slides were based on work by these beautiful minds.



Harriet Mills

Andrew Simpkin

James Staley

National University of
Ireland

UCB