Reference

Introduction to R and RStudio

- Use the escape key to cancel incomplete commands or running code (Ctrl+C) if you're using R from the shell.
- Basic arithmetic operations follow standard order of precedence:
  - Brackets: `(`, `)`
  - Exponents: `^` or `**`
  - Divide: `/`
  - Multiply: `*`
  - Add: `+`
  - Subtract: `-`
- Scientific notation is available, e.g: `2e-3`
- Anything to the right of a `#` is a comment, R will ignore this!
- Functions are denoted by `function_name()`. Expressions inside the brackets are evaluated before being passed to the function, and functions can be nested.
- Mathematical functions: `exp`, `sin`, `log`, `log10`, `log2` etc.
- Comparison operators: `<`, `<=`, `>`, `>=`, `==`, `!=`
- Use `all.equal` to compare numbers!
- `<-` is the assignment operator. Anything to the right is evaluate, then stored in a variable named to the left.
- `ls` lists all variables and functions you've created
- `rm` can be used to remove them
- When assigning values to function arguments, you *must* use `=`.

Project management with RStudio

- To create a new project, go to File -> New Project
- Install the `packrat` package to create self-contained projects

- `install.packages` to install packages from CRAN
- `library` to load a package into R
- `packrat::status` to check whether all packages referenced in your scripts have been installed.

Seeking help

- To access help for a function type `?function_name` or `help(function_name)`
- Use quotes for special operators e.g. `?"+"`
- Use fuzzy search if you can't remember a name `??search_term`
- CRAN task views are a good starting point.
- Stack Overflow is a good place to get help with your code.
  - `dput` will dump data you are working from so others can load it easily.
  - `sessionInfo()` will give details of your setup that others may need for debugging.

Data structures

Individual values in R must be one of 5 **data types**, multiple values can be grouped in **data structures**.

Data types

- `typeof(object)` gives information about an items data type.
- There are 5 main data types:
  - `numeric` real (decimal) numbers
  - `integer` whole numbers only
  - `character` text
  - `complex` complex numbers
  - `logical` TRUE or FALSE values

  Special types:

  - `NA` missing values
  - `NaN` "not a number" for undefined values (e.g. `0/0`).
  - `Inf`, `-Inf` infinity.

- `NULL` a data structure that doesn't exist

`NA` can occur in any atomic vector. `NaN`, and `Inf` can only occur in complex, integer or numeric type vectors. Atomic vectors are the building blocks for all other data structures. A `NULL` value will occur in place of an entire data structure (but can occur as list elements).

Basic data structures in R:

- atomic `vector` (can only contain one type)
- `list` (containers for other objects)
- `data.frame` two dimensional objects whose columns can contain different types of data
- `matrix` two dimensional objects that can contain only one type of data.
- `factor` vectors that contain predefined categorical data.
- `array` multi-dimensional objects that can only contain one type of data

Remember that matrices are really atomic vectors underneath the hood, and that data.frames are really lists underneath the hood (this explains some of the weirder behaviour of R).

Vectors

- `vector` All items in a vector must be the same type.
- Items can be converted from one type to another using *coercion*.
- The concatenate function 'c()' will append items to a vector.
- `seq(from=0, to=1, by=1)` will create a sequence of numbers.
- Items in a vector can be named using the `names()` function.

Factors

- `factor` Factors are a data structure designed to store categorical data.
- `levels` shows the valid values that can be stored in a vector of type factor.

Lists

- `list` Lists are a data structure designed to store data of different types.

Matrices

- `matrix` Matrices are a data structure designed to store 2-dimensional data.

Data Frames

- `data.frame` is a key data structure. It is a `list` of `vectors`.
- `cbind` will add a column (vector) to a data.frame.
- `rbind` will add a row (list) to a data.frame.

Useful functions for querying data structures:

- `str` structure, prints out a summary of the whole data structure
- `typeof` tells you the type inside an atomic vector
- `class` what is the data structure?
- `head` print the first `n` elements (rows for two-dimensional objects)
- `tail` print the last `n` elements (rows for two-dimensional objects)
- `rownames`, `colnames`, `dimnames` retrieve or modify the row names and column names of an object.
- `names` retrieve or modify the names of an atomic vector or list (or columns of a data.frame).
- `length` get the number of elements in an atomic vector
- `nrow`, `ncol`, `dim` get the dimensions of a n-dimensional object (Won't work on atomic vectors or lists).

Exploring Data Frames

- `read.csv` to read in data in a regular structure
  - `sep` argument to specify the separator
    - `","` for comma separated
    - `"\t"` for tab separated
  - Other arguments:
    - `header=TRUE` if there is a header row

Subsetting data

- Elements can be accessed by:
  - Index
  - Name
  - Logical vectors
  - `[` single square brackets:
  - *extract* single elements or *subset* vectors
  - e.g. `x[1]` extracts the first item from vector x.
  - *extract* single elements of a list. The returned value will be another `list()`.
  - *extract* columns from a data.frame
- `[` with two arguments to:
  - *extract* rows and/or columns of
    - matrices
    - data.frames
    - e.g. `x[1,2]` will extract the value in row 1, column 2.
    - e.g. `x[2,:]` will extract the entire second column of values.
- `[[` double square brackets to extract items from lists.
- `$` to access columns or list elements by name
- negative indices skip elements

Control flow

- Use `if` condition to start a conditional statement, `else if` condition to provide additional tests, and `else` to provide a default
- The bodies of the branches of conditional statements must be indented.
- Use `==` to test for equality.
- `X && Y` is only true if both X and Y are `TRUE`.
- `X || Y` is true if either X or Y, or both, are `TRUE`.

- Zero is considered `FALSE`; all other numbers are considered `TRUE`
- Nest loops to operate on multi-dimensional data.

Vectorization

- Most functions and operations apply to each element of a vector
- `*` applies element-wise to matrices
- `%*%` for true matrix multiplication
- `any()` will return `TRUE` if any element of a vector is `TRUE`
- `all()` will return `TRUE` if *all* elements of a vector are `TRUE`

Functions explained

- `?"function"`
- Put code whose parameters change frequently in a function, then call it with different parameter values to customize its behavior.
- The last line of a function is returned, or you can use `return` explicitly
- Any code written in the body of the function will preferably look for variables defined inside the function.
- Document Why, then What, then lastly How (if the code isn't self explanatory)

Writing data

- `write.table` to write out objects in regular format
- set `quote=FALSE` so that text isn't wrapped in `"` marks

Producing reports with knitr

- Value of reproducible reports
- Basics of Markdown
- R code chunks
- Chunk options

- Inline R code
- Other output formats

Best practices for writing good code

- Program defensively, i.e., assume that errors are going to arise, and write code to detect them when they do.
- Write tests before writing code in order to help determine exactly what that code is supposed to do.
- Know what code is supposed to do before trying to debug it.
- Make it fail every time.
- Make it fail fast.
- Change one thing at a time, and for a reason.
- Keep track of what you've done.
- Be humble