Principles and Practice of Data Analysis

for Reproducible Research in R

Introduction to R

Heather Turner

Department of Statistics, University of Warwick

2016–09–26

Why R?

Free.

Open source.

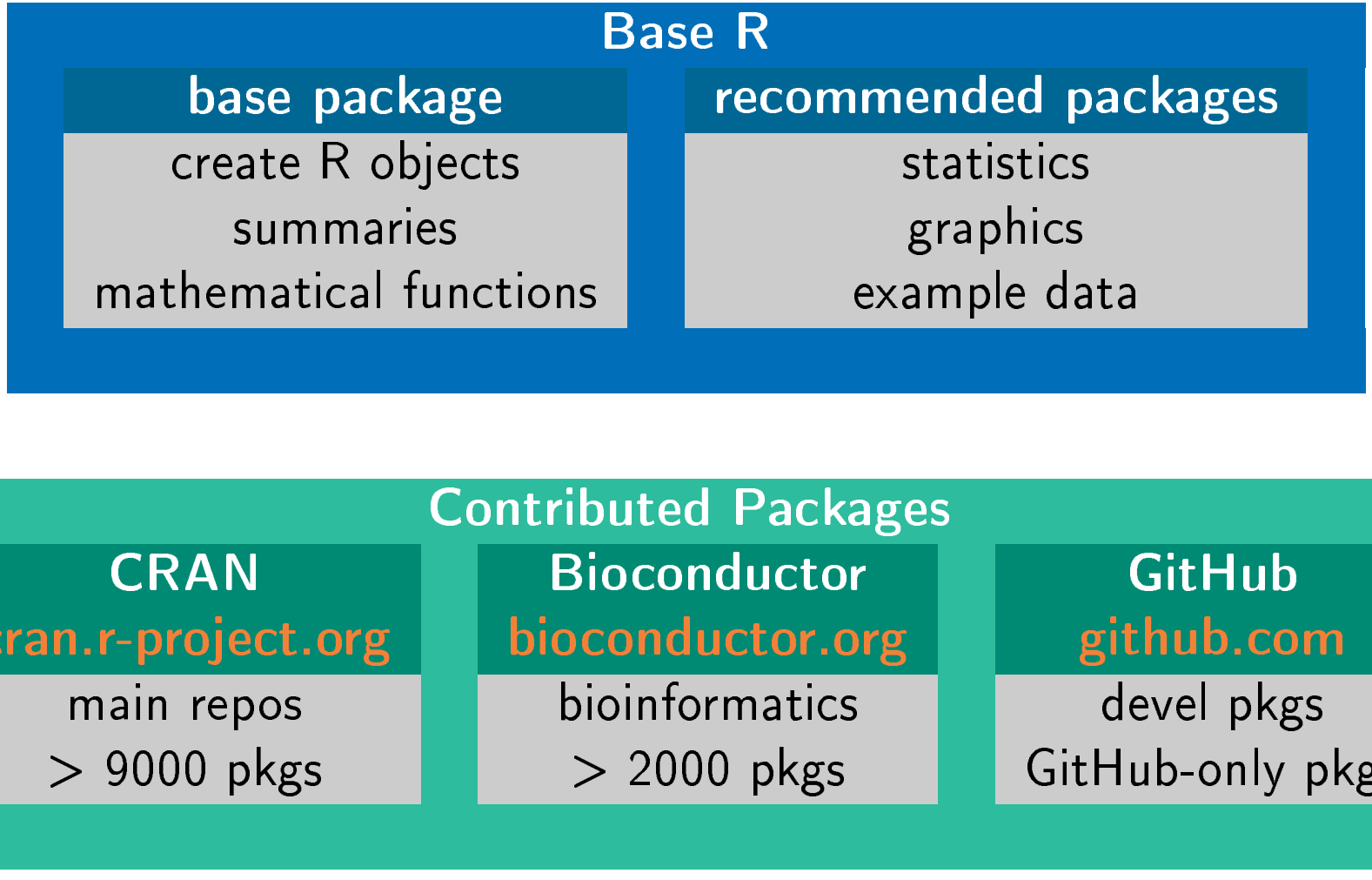
Software for data science:

- experiment/survey design
- data retrieval
- data wrangling
- data analysis
- reporting

A programming language, so we can

- use existing functions to code up our data science tasks
- write new functions for customised/novel tasks

The R Ecosystem

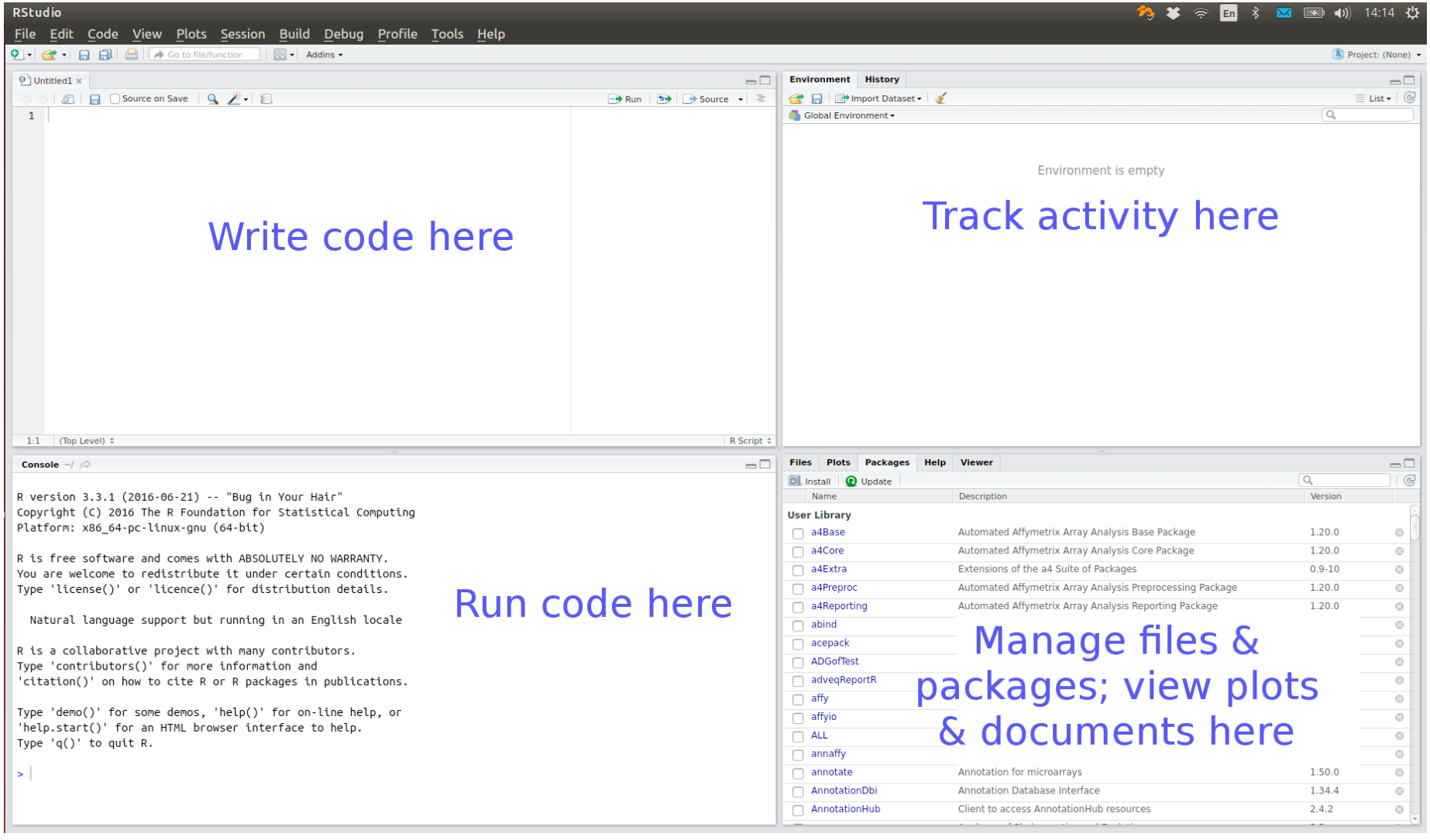


R Demo

We can type commands directly into the R console

```
3 + 4
?"+" #look up help for "+"
x <- 3 + 4 ; y <- log(x)
ls() # list of objects in the current workspace
rm(x)
data() # find out what standard data sets there are
plot(iris) # plot Fisher's iris data
```

RStudio IDE


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1
Source on Save
Run
Source
Write code here
Environment
History
Import Dataset
Global Environment
Environment is empty
Track activity here
Console
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Run code here
Files
Plots
Packages
Help
Viewer
Install
Update
Name
Description
Version
User Library
a4Base
Automated Affymetrix Array Analysis Base Package
1.20.0
a4Core
Automated Affymetrix Array Analysis Core Package
1.20.0
a4Extra
Extensions of the a4 Suite of Packages
0.9-10
a4Preproc
Automated Affymetrix Array Analysis Preprocessing Package
1.20.0
a4Reporting
Automated Affymetrix Array Analysis Reporting Package
1.20.0
abind
acepack
ADGofTest
adveqReportR
affy
affyio
ALL
annaffy
annotate
Annotation for microarrays
1.50.0
AnnotationDbi
Annotation Database Interface
1.34.4
AnnotationHub
Client to access AnnotationHub resources
2.4.2
Manage files & packages; view plots & documents here

RStudio Features

Features provided by RStudio include:

- syntax highlighting, code completion, smart indentation
- interactively send code chunks from editor to R
- organise multiple scripts, help files, plots
- search code and help files

RStudio Shortcuts from the R Console

RStudio provides a few shortcuts to help write code in the R console

Up/Down	go back/forward through history one command at a time
Ctrl+Uparrow	review recent history and select command
Tab	view possible completions for part-written expression

Code completion is also provided in the source editor

R Scripts

Text files saved with a `.R` suffix are recognised as R code.

Code can be sent directly from the source editor as follows

current line	Ctrl+Enter or Run button
multiple lines	highlight lines, then run as above
whole file	Ctrl+Shirt+Enter or Source button. Can also check the Source on Save box.

R Studio Demo

```r
View(iris)
# showing code completion, running code, indentation
sum(3, 4)
summary(iris, maxsum = 2,
        digits = 2)
```

Vectors

A single number is a special case of a numeric vector. Vectors of length greater than one can be created using the concatenate function, `c`

```
x <- c(1, 3, 6)
```

The elements of the vector must be of the same type: common types are numeric, character and logical

```
y <- c("red", "yellow", "green")
z <- c(TRUE, FALSE)
```

Missing values (of any type) are represented by the symbol `NA`.

Data Frames

Data sets are stored in R as *data frames*. These are structured as a list of objects, typically vectors, of the same length

```
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
```

Here `Species` is a `factor`, a special data structure for categorial variables.

Numeric Summaries

Applying `summary` to a data frame will give a summary of each variable

```
summary(iris)
```

```
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

This is returned as the table of characters shown, so only suitable for printing.

Single Column

We can refer to a single column from the data frame by name.

Thus we can obtain summaries for a single variable as follows

```
head(iris$Sepal.Length, 4)
```

```
## [1] 5.1 4.9 4.7 4.6
```

```
fivenum(iris$Sepal.Length)
```

```
## [1] 4.3 5.1 5.8 6.4 7.9
```

```
quantile(iris$Sepal.Length, 0.25)
```

```
## 25%
## 5.1
```

Other useful summary functions for single variables include `min`, `mean` and `range`.

Contingency Tables

A one-way table can be created as follows

```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

To illustrate a two way table, we first create a new factor using the function `cut` to cut the `Petal.Width` variable into three levels

```
petalCat <- cut(iris$Petal.Width, breaks = c(0, 1, 2, 3))
table(iris$Species, petalCat)
```

```
##              petalCat
##               (0,1] (1,2] (2,3]
##   setosa         50     0     0
##   versicolor      7    43     0
##   virginica       0    27    23
```

Graphs

In RStudio, graphs are displayed in the Plots window. The plot is sized to fit the window and will be rescaled if the size of the window is changed.

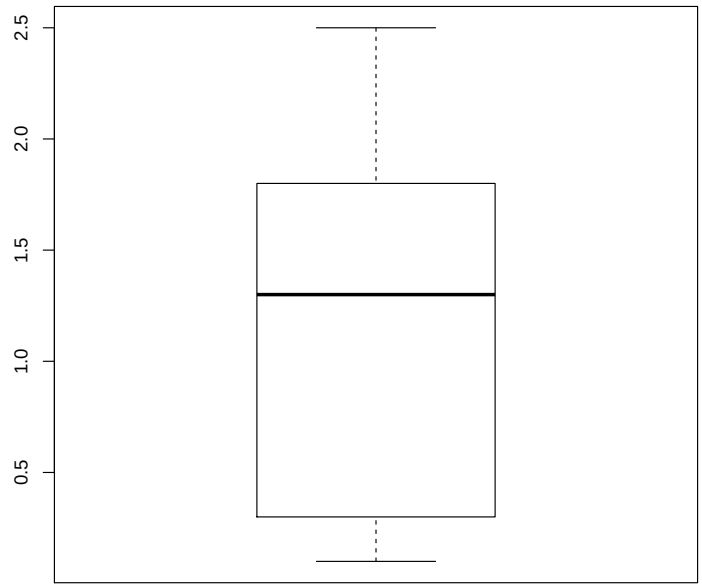
Back and forward arrows allow you to navigate through graphs that have been plotted.

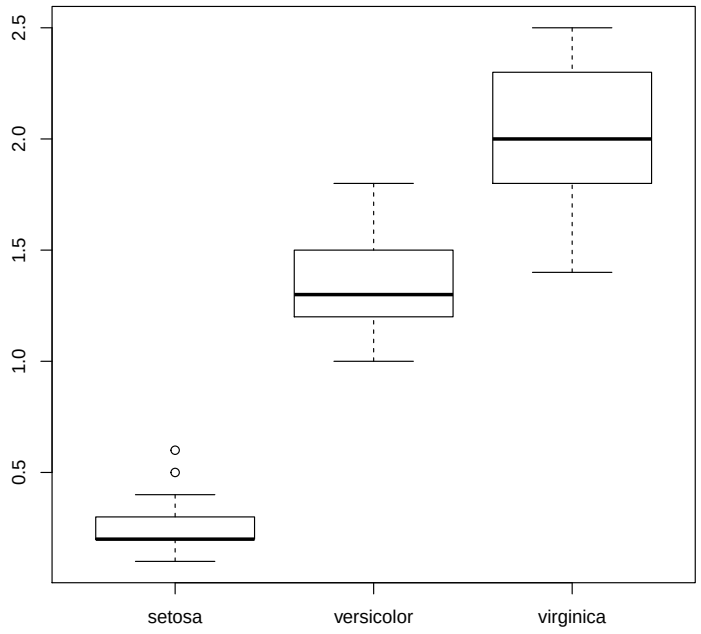
Graphs can be saved in various formats using the Export drop down menu, which also has an option to copy to the clipboard.

Here we consider "no-frills" plots, for quick exploratory plots.

Boxplots

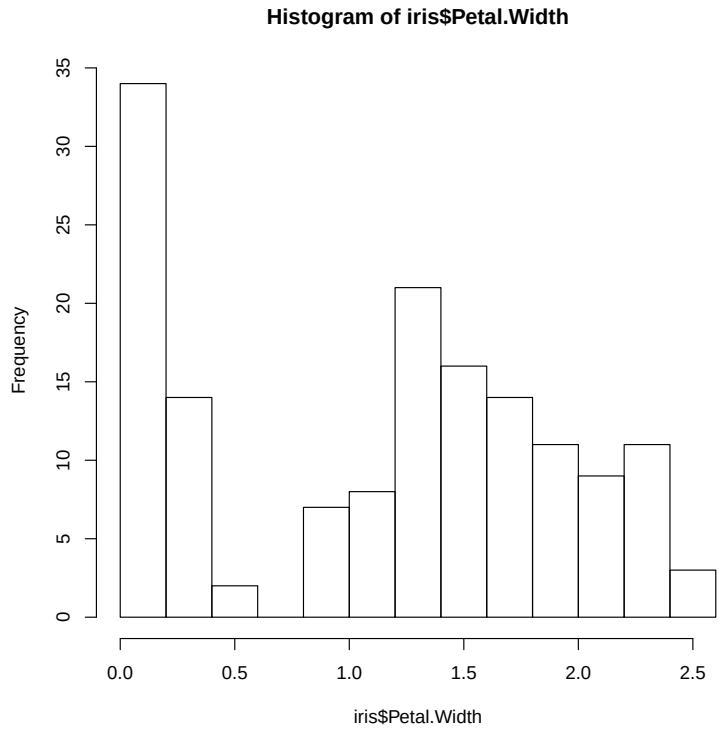
```
boxplot(iris$Petal.Width)
with(iris, boxplot(Petal.Width ~ Species))
```

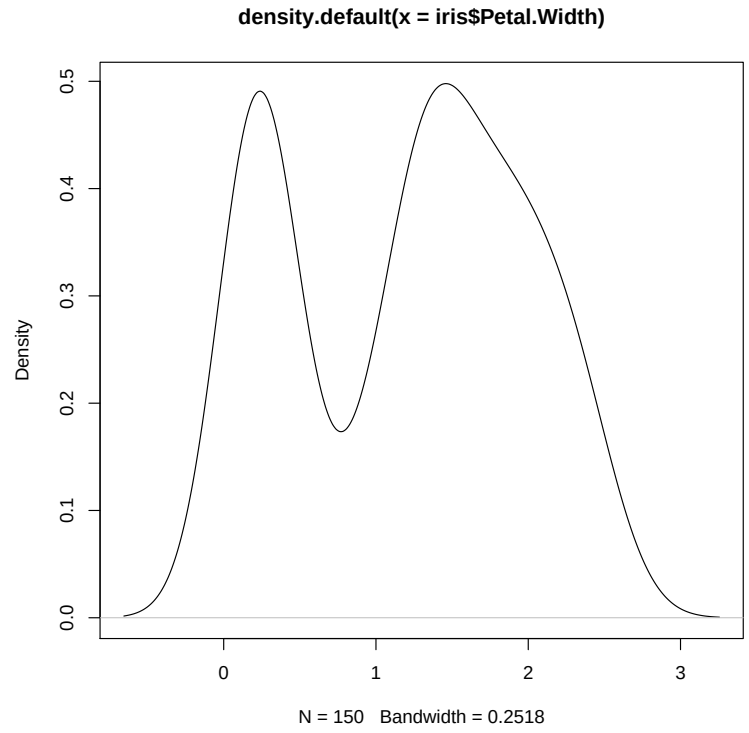




Histogram/Density

```
hist(iris$Petal.Width)
plot(density(iris$Petal.Width))
```

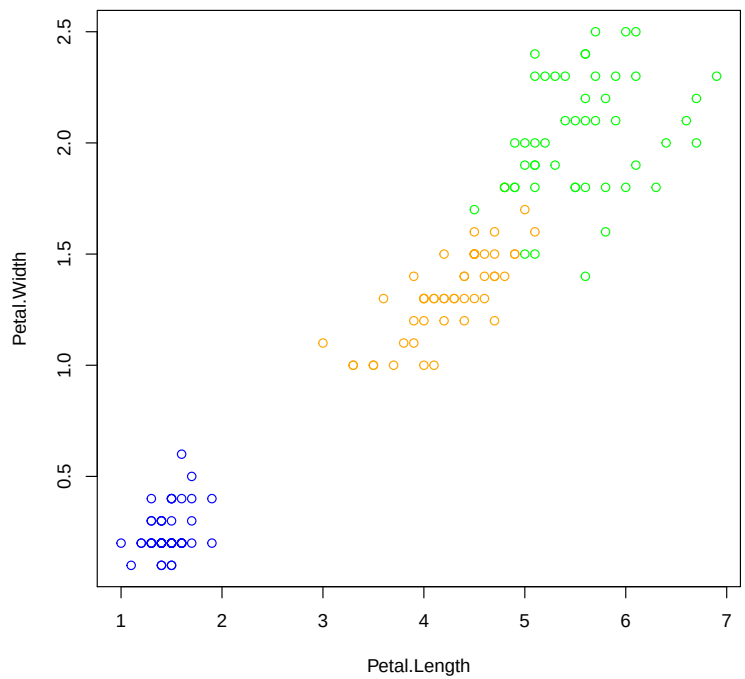




Scatterplots

Here we use the underlying numeric representation of the `Species` factor to index a colour vector for colouring the data points.

```
mycol <- c("blue", "orange", "green")
with(iris, plot(Petal.Width ~ Petal.Length,
                col = mycol[Species]))
```

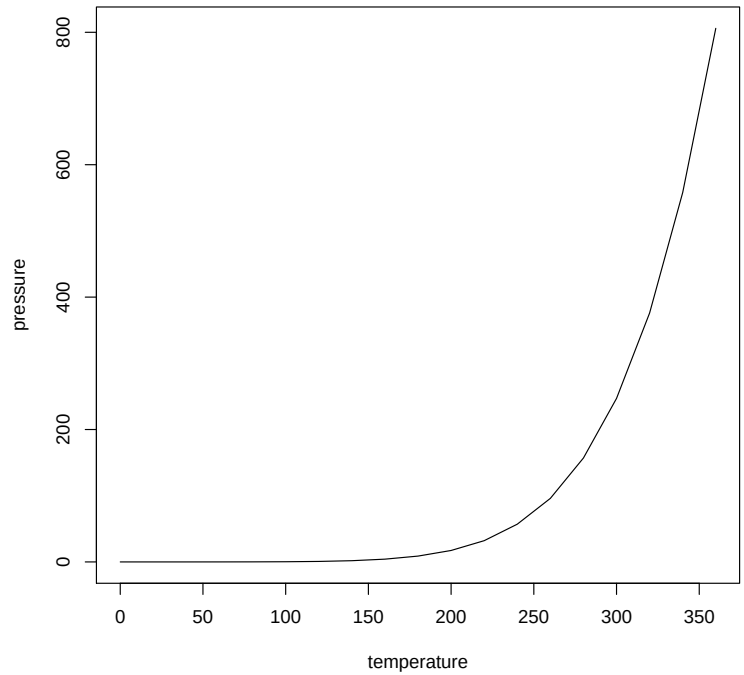


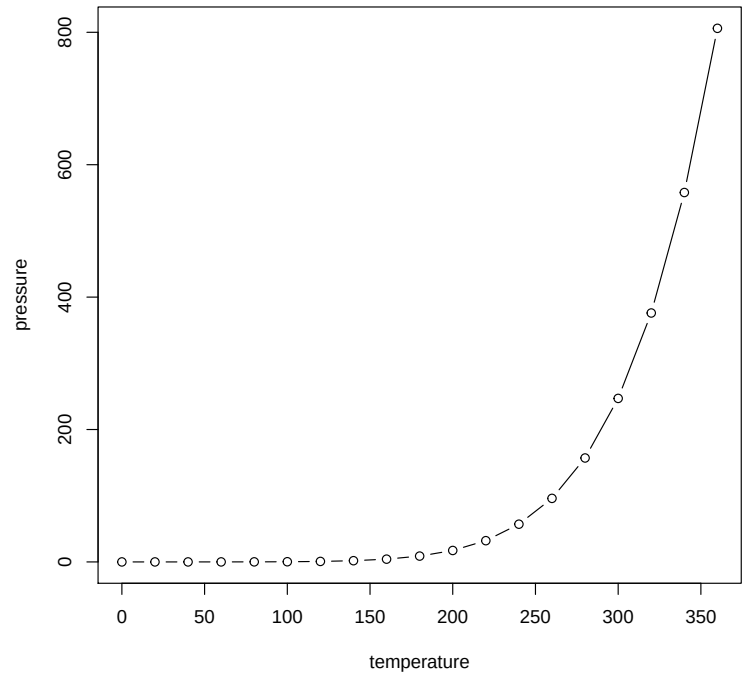
See Colors in R for a display of all named colours in R.

Line chart

By default the `type` argument is `"p"` (points), use `"l"` for lines or `"b"` for both.

```
with(pressure, plot(pressure ~ temperature, type = "l"))
with(pressure, plot(pressure ~ temperature, type = "b"))
```





Packages

At installation, a library will have been created with the base and recommend packages.

A selection of these packages are loaded by default at the start of an R session. These packages are then on the *search path* for R to find functions

```
search()
```

```
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

If we try to use a function in a package that is not loaded, we get an error

```
truehist(iris$Species)


## Error in eval(expr, envir, enclos):  could not find function
"truehist"
```

Packages

Using the help search in RStudio, or `??` we can discover if the function is in an installed package. If so, we simply need to load the package first

```
library(MASS)
truehist(iris$Species)
```

If it's not in an installed package, we can try searching on Rdocumentation.org or RSeek.org.

Installing Packages

CRAN packages can be installed from the RStudio Packages tab.

For Bioconductor packages, we source the `biocLite` function from the Bioconductor website, then run e.g.

```
source("https://bioconductor.org/biocLite.R")
biocLite("limma", "edgeR")
```

`biocLite` must be sourced in each session a Bioconductor package is to be installed. It ensures the Bioconductor and R versions are compatible.

GitHub packages are built from source, so require additional tools (see "Install R" notes). Once setup, we can use `install_github` from the **devtools** package, e.g.

```
library(devtools)
install_github("hadley/vctrs")
```

It may be necessary to install other packages the package depends on from GitHub/Bioconductor first.

Finding Out About Packages

Often we find out about packages from a book, paper or tutorial.

To find a package that implements a particular method, we can search on Rdocumentation.org or RSeek.org.

CRAN Task Views and BiocViews give overviews of packages relating to a particular application or type of method.

Navigating a Package

Help for a package can be found by clicking on the *Packages* tab. This lists the functions/topics with help pages in that package.

In addition the following may be available

`DESCRIPTION` Metadata for the package, including a short `Description` field

`vignettes` User guides or extended documentation on specific topics

`demos` Use e.g. `demo(package = "stats")` to see available demos

`NEWS` News of changes in each package release

`pkg-package` Overall help file for `pkg`