Introduction to R

Dr Heather Turner
Freelance/Department of Statistics, University of Warwick, UK

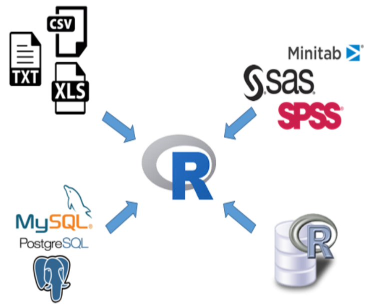
Dr Eralp Dogu
Department of Statistics, Mugla Sitki Kocman University, TR

2 October 2018

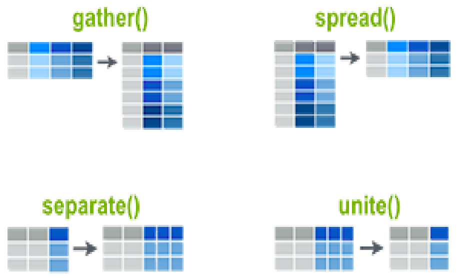
What can R do?

Data Management

Data Import

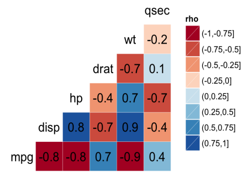


Reshaping

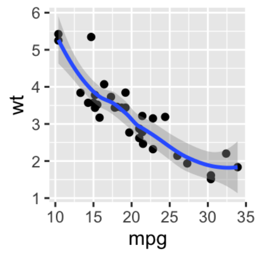


Data Analysis

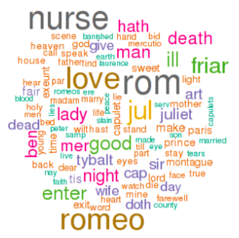
Explaratory Data Analysis



Statistical Modelling



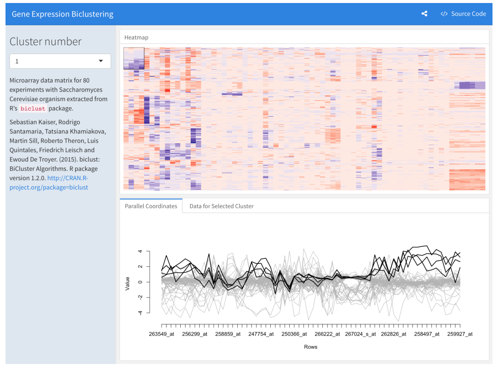
Advanced Statistics



What can R do?

Reporting

Dashboard


Gene Expression Biclustering

Source Code

Cluster number

1

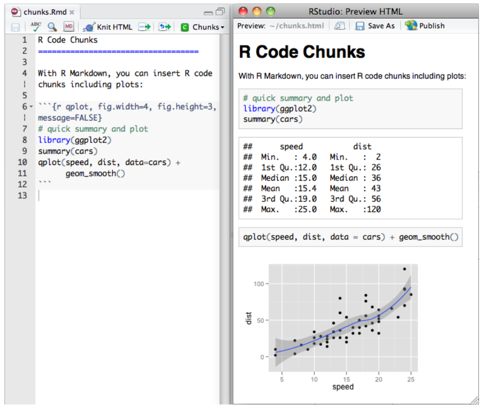
Microarray data matrix for 80 experiments with Saccharomyces Cerevisiae organism extracted from R's `biclust` package.

Sebastian Kaiser, Rodrigo Santamaria, Tatsiana Khamiakova, Martin Sill, Roberto Theron, Luis Quintales, Friedrich Leisch and Ewoud De Troyer. (2015). biclust: BiCluster Algorithms. R package version 1.2.0. http://CRAN.R-project.org/package=biclust

Heatmap

Parallel Coordinates | Data for Selected Cluster

Markdown


chunks.Rmd

Knit HTML | Chunks

~~~
R Code Chunks
=====================================

With R Markdown, you can insert R code
chunks including plots:

```{r qplot, fig.width=4, fig.height=3,
message=FALSE}
# quick summary and plot
library(ggplot2)
summary(cars)
qplot(speed, dist, data=cars) +
      geom_smooth()
```
~~~

RStudio: Preview HTML

Preview: ~/chunks.html | Save As | Publish

R Code Chunks

With R Markdown, you can insert R code chunks including plots:

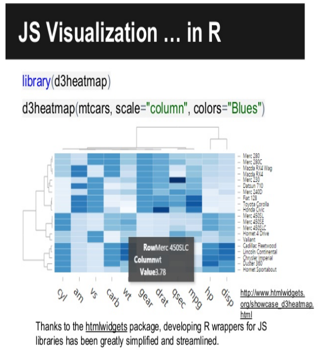
```
# quick summary and plot
library(ggplot2)
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2
##  1st Qu.:12.0   1st Qu.: 26
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```

```
qplot(speed, dist, data = cars) + geom_smooth()
```

Programming

Connect to other languages

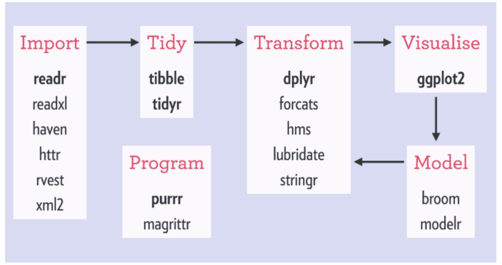

JS Visualization ... in R

```
library(d3heatmap)
d3heatmap(mtcars, scale="column", colors="Blues")
```

http://www.htmlwidgets.org/showcase_d3heatmap.html

Thanks to the htmlwidgets package, developing R wrappers for JS libraries has been greatly simplified and streamlined.

Packages



The R Ecosystem

Base

base

create R objects
summaries
math functions

recommended

statistics
graphics
example data

Contributed Packages

CRAN
cran.r-project.org

main repos
~13000 pkgs

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS
bioconductor.org

bioinformatics
>1500 pkgs

GitHub
github.com

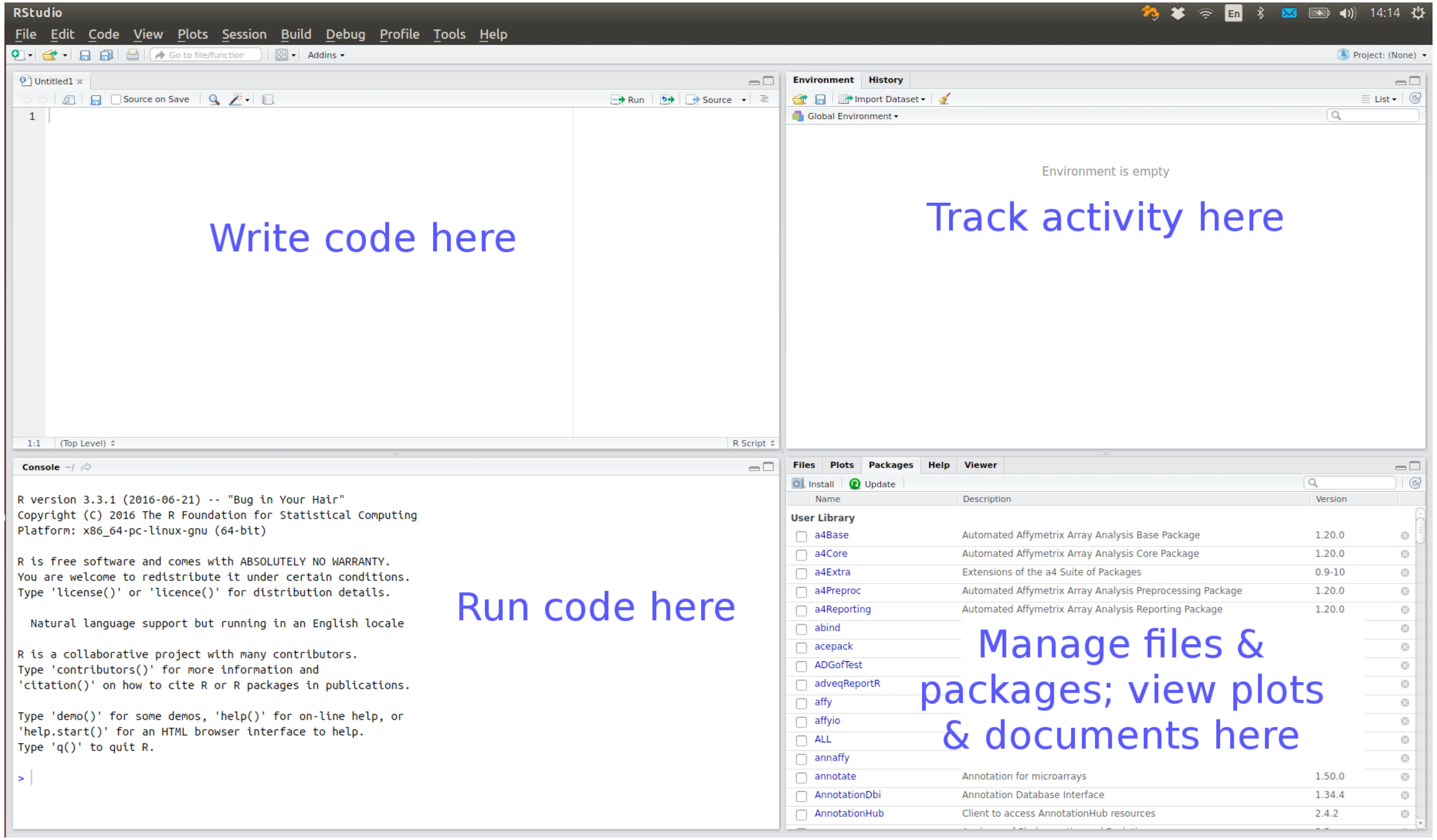
devel pkgs
GitHub-only pkgs

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



Environment is empty

```
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.

>
```

User Library

Name	Description	Version
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

R Studio 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

- `↑`/`↓` go back/forward through history one command at a time
- `Ctrl`/`⌘` + `↑` review recent history and select command
- `Tab` view possible completions for part-written expression

Code completion (using `Tab`) 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

- `Ctrl` / `⌘` + `↲` or Run run current line
  - run multiple lines by selecting first
- `Ctrl` / `⌘` + `Shift` + `↲` or Source. Run the script from start to finish.

R Studio Demo

```r
View(iris)
# showing code completion, running code, indentation
sum(3, 4)
summary(iris, maxsum = 2,
        digits = 2)
pbirthday(40, classes = 365, coincident = 3)
```

Why R Scipts

Writing an R script for an analysis has several advantages over a graphical user interface (GUI)

- it provides a *record* of the exact approach used in an analysis
- it enables the analysis to be easily *reproduced* and modified
- it allows greater control

Good Practices for R Scripts

Organising your R script well will help you and others understand and use it.

- Add comment or two at start to describe purpose of script
- Load required data and packages at the start
- Avoid hard-coding: define variables such as file paths early on
- Give functions and variable meaningful names
- use ### or `#---` to separate sections (in RStudio Code > Insert Section)

Your Turn!

In RStudio, open a new R Script.

Try typing some of the code from the R/R Studio demo and using some of the shortcuts to complete your commands and run them.

Data Structures

Data structures are the building blocks of code. In R there are four main types of structure:

- vectors and factors
- matrices and arrays
- lists
- data frames

The first and the last are sufficient to get started.

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

```
x <- 1:3
x
# [1] 1 2 3
x <- rep(1,3)
x
# [1] 1 1 1
y <- c("red", "yellow", "green")
y
# [1] "red"    "yellow" "green"
z <- c(TRUE, FALSE)
```

Data Frames

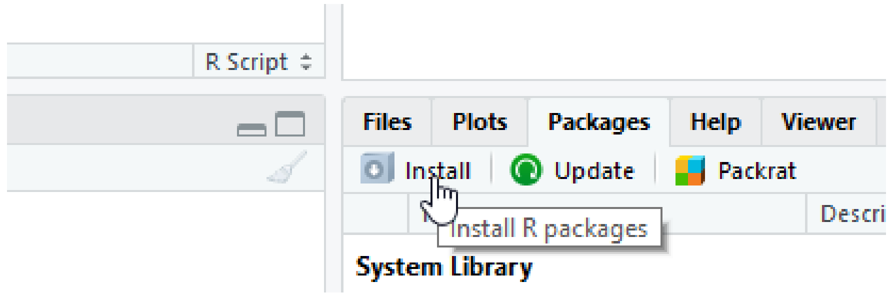
Data sets are stored in R as *data frames*. These are structured as a list of objects, typically vectors, of the same length

```
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 1 1 1 1 1 ...
x <- 1:3
y <- c("red", "yellow", "green")
mydata <- data.frame(x, y)
mydata
#   x      y
# 1 1    red
# 2 2 yellow
# 3 3  green
ls()
# [1] "mydata" "x"      "y"      "z"
```

Install Packages

Most day-to-day work will require at least one contributed package.

CRAN packages can be installed from the Packages tab



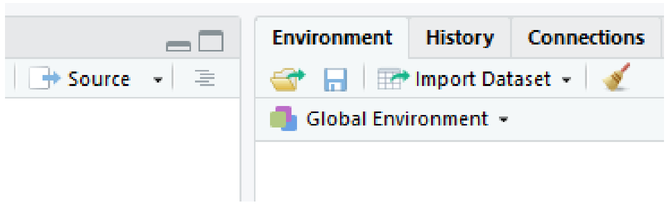
To use an installed package in your code, you must first load it from your package library

```r
install.packages("survival", dependencies = TRUE)
library(survival)
```

Sometimes an RStudio feature will require a contributed package. A pop-up will ask permission to install the package the first time, after that RStudio will load it automatically.

Data Input via Import Dataset

Using the Import Dataset dialog in RStudio



we can import files stored locally or online in the following formats:

- `.txt`/`.csv` via `read_delim`/`read_csv` from **readr**.
- `.xlsx` via `read_excel` from **readxl**.
- `.sav`/`.por` , `.sas7bdat` and `.dta` via `read_spss`, `read_sas` and `read_stata` respectively from **haven**.

Most of these functions also allow files to be compressed, e.g. as `.zip`.

Your Turn

Use the Import Dataset button, import a data set from the `Data Sets` folder on the Piazza page. RStudio will ask to install packages as required.

Try changing some of the import options to see how that changes the preview of the data and the import code.

Install the **skimr** package and use the `skim` function to summarise the data set.

Tibbles

The functions used by *Import Dataset* return data frames of class `"tbl_df"`, aka **tibbles**. The main differences are

	data.frame	**tibble**
Printing (default)	Whole table	10 rows; columns to fit Prints column type
Subsetting	`dat[, 1]`, `dat$X1`, `dat[[1]]` all return vector	`dat[,1]` returns tibble `dat$X1`, `dat[[1]]` return vector
Strings	Converted to factor (default)	Left as character
Variable names	Made *syntactically valid* e.g. `Full name` -> `Full.name`	Left as is use e.g. ``dat$`Full name` ``

Data Input via Code

The **rio** package provides a common interface to the functions used by *Import Dataset* as well as many others.

The data format is automatically recognised from the file extension. To read the data in as a tibble, we use the setclass argument.

```r
library(rio)
compsci <- import("compsci.csv", setclass = "tibble")
cyclist <- import("cyclist.xlsx", setclass = "tibble")
```

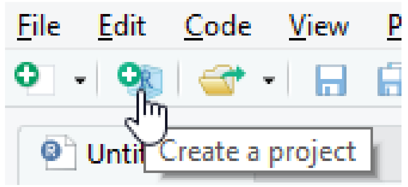
See `?rio` for the underlying functions used for each format and the corresponding optional arguments, e.g. the skip argument to `read_excel` to skip a certain number of rows.

R Studio Projects

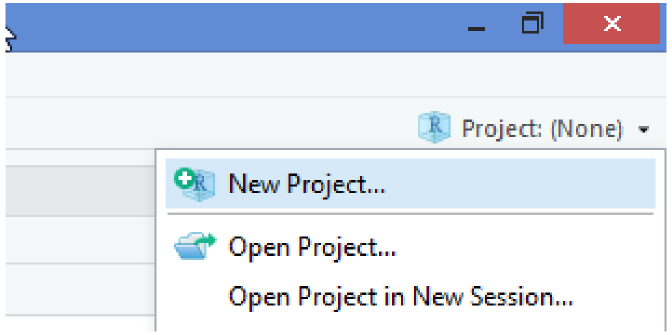
An Rstudio project is a context for work on a specific project

- automatically sets working directory to project root
- has separate workspace and command history
- works well with version control via git or svn

Create a project from a new/existing directory via the File Menu or the New Project button



Switch project, or open a different project in a new RStudio instance via the Project menu.



Project Structure

An R project structure helps you kickstart the project you undertake as fast as possible, using a standardized form.

A typical project can be organised into folders such as `data`, `scripts`, `results` and `reports`.

The scripts should be named to reflect the content/function:

- requirements.R = includes required packages
- functions.R = includes custom functions
- custom_theme.R = includes a custom theme to create graphs
- analysis.R = main analysis script

A README file can be used to describe the project and the files.

R Markdown Documents

R markdown documents (`.Rmd`) intersperse code chunks (R, Python, Julia, C++, SQL) with markdown text

YAML header

```
---
title: "Report"
output: html_document
---
```

Markdown text

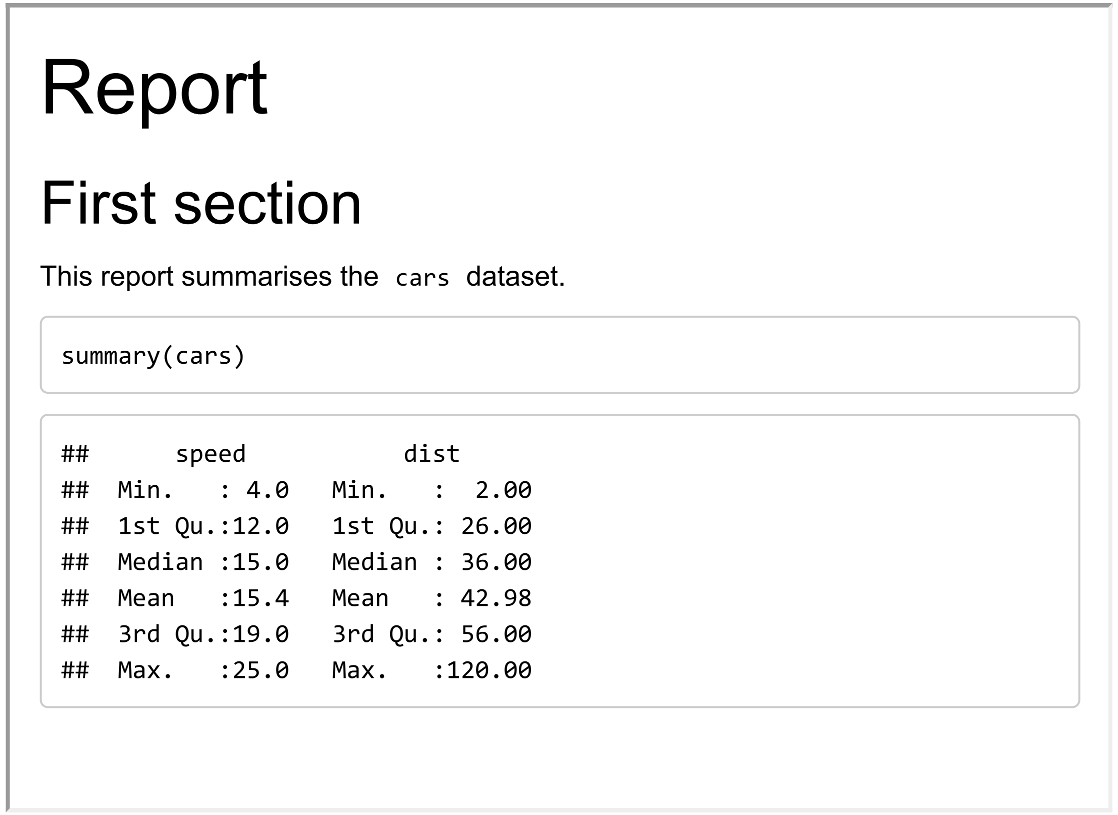
```
## First section

This report summarises the `cars` dataset.
```

R code chunk

~~~
```{r summary-cars}
summary(cars)
```
~~~

Rendering

The `.Rmd` file can be rendered to produce a document (HTML, PDF, docx) integrating the code output.



Report

First section

This report summarises the `cars` dataset.

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Your Turn

Copy the `infant` directory from Piazza to your laptop. Open `infant.Rproj`, which is an RStudio project file.

From the Files tab, open the `infant.Rmd` R markdown file. In the chunk labelled `import-data`, write some code that will import the `infant.xlsx` file and create a tibble named `infant`. Load any required packages in the `setup` chunk.

Run the code in the `import-data` chunk (the `setup` chunk is run automatically). Use `View()` in the console to inspect the result.

Data Pre-processing

The imported data are a subset of records from a US Child Health and Development Study, corresponding to live births of a single male foetus.

The data requires a lot of pre-processing

- converting numeric variables to factors
- converting coded values to missing values
- filtering rows and selecting columns

The **dplyr** package can be used to help with many of these steps

dplyr

The **dplyr** package provides the following key functions to operate on data frames

- `filter()`
- `arrange()`
- `select()` (and `rename()`)
- `distinct()`
- `mutate()` (and `transmute()`)
- `summarise()`

`filter()`

`filter()` selects rows of data by criteria

```r
library(dplyr)
infant <- filter(infant, smoke != 9 & age > 18)
```

Building block	R code
Binary comparisons	>, <, ==, <=, >=, `!=`
Logical operators	or `|` and &, not `!`
Value matching	e.g. `x %in% 6:9`
Missing indicator	e.g. `is.na(x)`

`select()`

`select()` selects variables from the data frame.

Select named columns:

```
select(infant, gestation, sex)
```

Select a sequence of columns, along with an individual column

```
select(infant, pluralty:gestation, parity)
```

Select all columns *except* specified columns

```
select(infant, -(id:outcome), -sex)
```

`mutate()`

`mutate()` computes new columns based on existing columns. Re-using an existing name replaces the old variable. For example we can convert the mother's weight from pounds to kilograms

```
mutate(infant,
       wt = ifelse(wt == 999, NA, wt),
       wt = wt * 0.4536)
```

To only keep the computed columns, use `transmute()` in place of `mutate()`.

Your Turn

In the `select-variables` chunk, use `select()` to remove the redundant variables `pluralty`, `outcome` and `sex`, as well a characteristics of the father `drace`, ... , `dwt`. Overwrite the `infant` data frame.

The variable `gestation` gives the length of the pregnancy in days. In the `filter-variables` chunk, filter the data to exclude extremely premature babies (gestation less than 28 weeks) and extremely late babies (gestation more than 52 weeks).

The value 999 has been used to code an unknown value in the `ht` variable. In the `mutate-variables` chunk, use `mutate` to update `ht`, replacing 999 with NA.

Chaining

We can use %>% to pipe the data from one step to the next

```r
infant <- infant %>%
    filter(smoke != 9 & age > 18) %>%
    select(-(id:outcome), -sex) %>%
    mutate(wt = ifelse(wt == 999, NA, wt),
           wt = wt * 0.4536)
```

Any function with data as the first argument can be added to the data pipeline.

`summarise()`

`summarise()` function is for computing single number summaries of variables, so typically comes at the end of a pipeline or in a separate step

```
stats <- summarise(infant,
                   `Mean mother's weight (kg)` = mean(wt, na.rm = TRUE),
                   `Sample size` = sum(!is.na(wt)),
                   `Total sample size` = n())
stats
```

```
# # A tibble: 1 x 3
#   `Mean mother's weight (kg)` `Sample size` `Total sample size`
#                         <dbl>         <int>               <int>
# 1                        58.4          1167                1203
```

In an R markdown document we can convert the result to a markdown table using `kable` from the **knitr** package

```
kable(stats, align = "ccc")
```

Grouped Operations

Grouping can be set on a data frame using `group_by`. This is most useful for `summarise()`

```r
infant <- infant %>% filter(!race %in% c(10, 99)) %>%
    mutate(race = recode(race, `6` = "Latino", `7` = "Black", `8` = "Asian",
                         `9` = "Mixed", .default = "White"))
res <- infant %>% group_by(`Ethnic group` = race) %>%
    summarise(`Mean weight (kg)` = mean(wt, na.rm = TRUE))
kable(res, align = "lc")
```

Ethnic group	Mean weight (kg)
Asian	49.84
Black	62.63
Latino	54.12
Mixed	58.46
White	57.86

Your Turn

`cut()` (in the **base** package) can be used to create a factor from a continuous variable, e.g.

```
cut(c(0.12, 1.37, 0.4, 2.3), breaks = c(0, 1, 2, 3))
# [1] (0,1] (1,2] (0,1] (2,3]
# Levels: (0,1] (1,2] (2,3]
```

where `(0, 1]` is the set of numbers > 0 and ≤ 1, etc.

The infant birth weight `bwt` is given in ounces. In the `table-bwt` chunk use `mutate()` to create a new factor called `Birth weight (g)`, by converting `bwt` to grams through multiplication by 28.35 and then converting the result to a factor with the following categories: (1500, 2000], (2000, 2500], (2500, 3000], (3000, 3500], and (3500, 5000].

An infant is categorised as low weight if its birth weight is ≤ 2500 grams, regardless of gestation. Use `group_by()` to group the data by the new weight factor, then pipe to `summarise()` to count the number of infants in each weight category.

Plots

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.

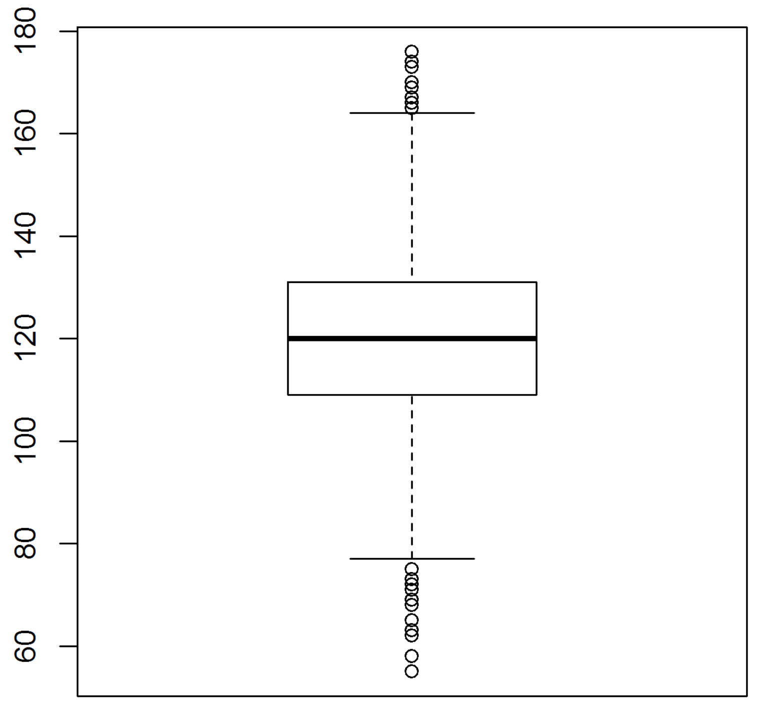
Back and forward arrows allow you to navigate through graphs that have been plotted.

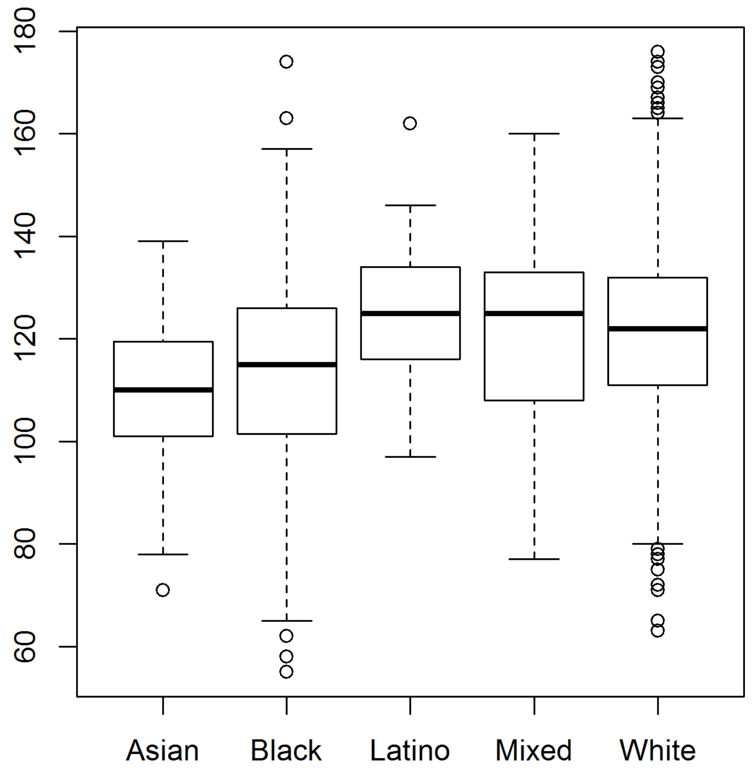
Graphs can be saved in various formats using the Export drop down menu, which also has an option to copy to the clipboard.

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

Boxplots

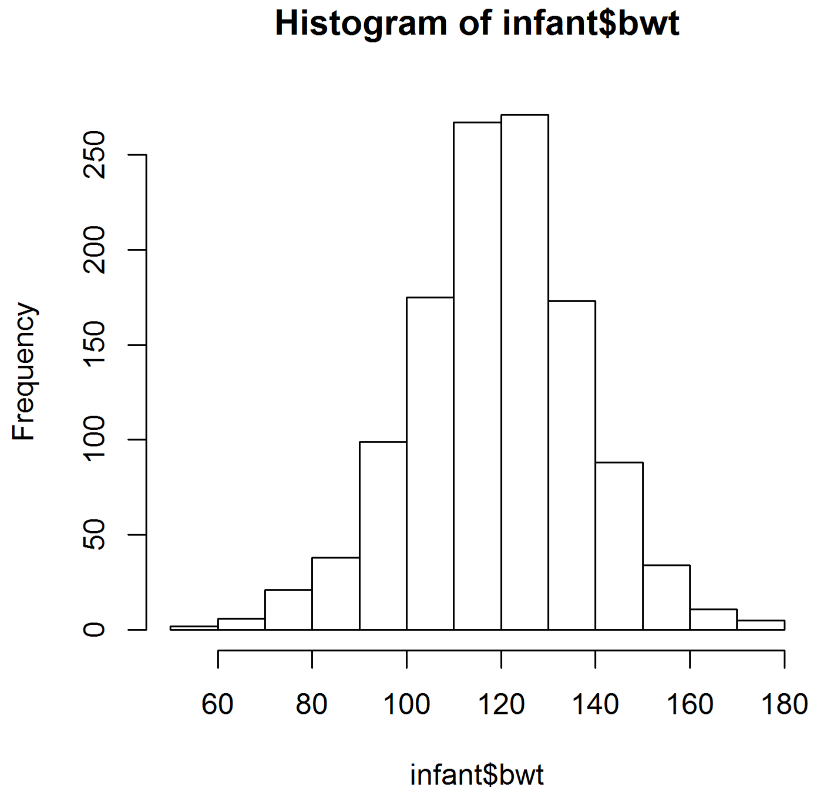
```
boxplot(infant$bwt)
with(infant, boxplot(bwt ~ race))
```

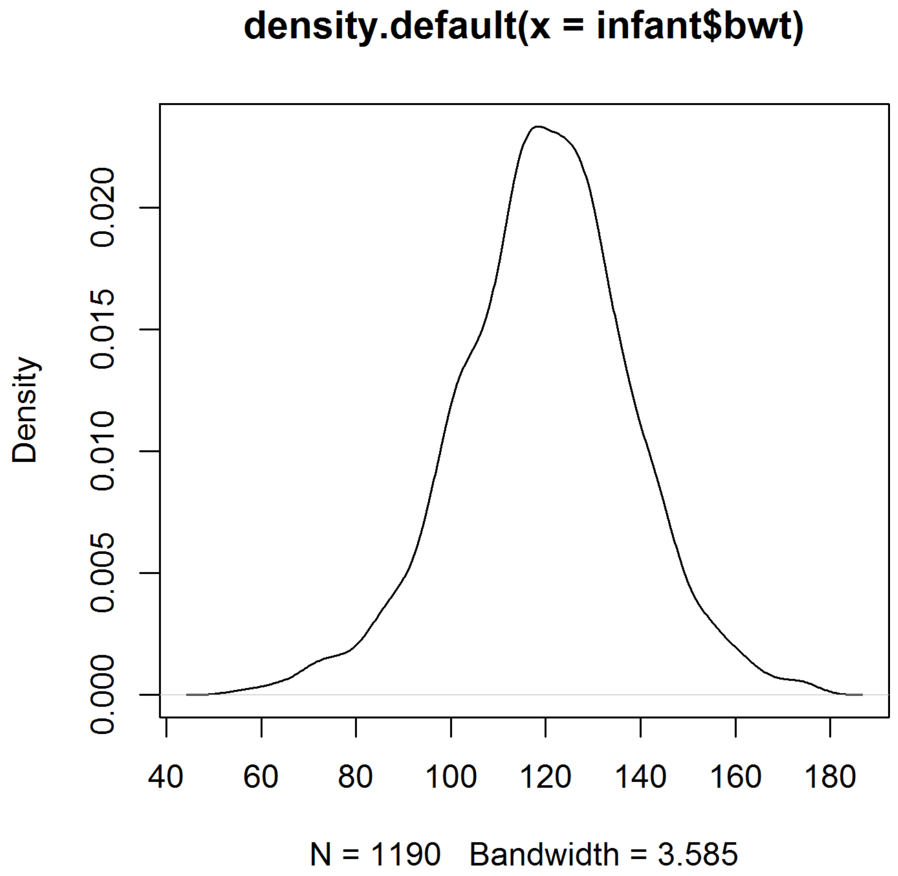




Histogram/Density

```
hist(infant$bwt)
plot(density(infant$bwt))
```





Scatterplots

```r
infant <- infant %>%
    mutate(gestation = ifelse(gestation == 999, NA, gestation))
plot(bwt ~ gestation, data = infant)
# plot(infant$gestation, infant$bwt); plot(x = infant$gestation, y = infant$bwt)
```

The two observations with very low gestation look suspect, so we will exclude them from the rest of the analysis

```r
infant <- infant %>%
    filter(gestation > 200)
```

ggplot2

ggplot2 is a package that produces plots based on the *grammar of graphics* (Leland Wilkinson), the idea that you can build every graph from the same components

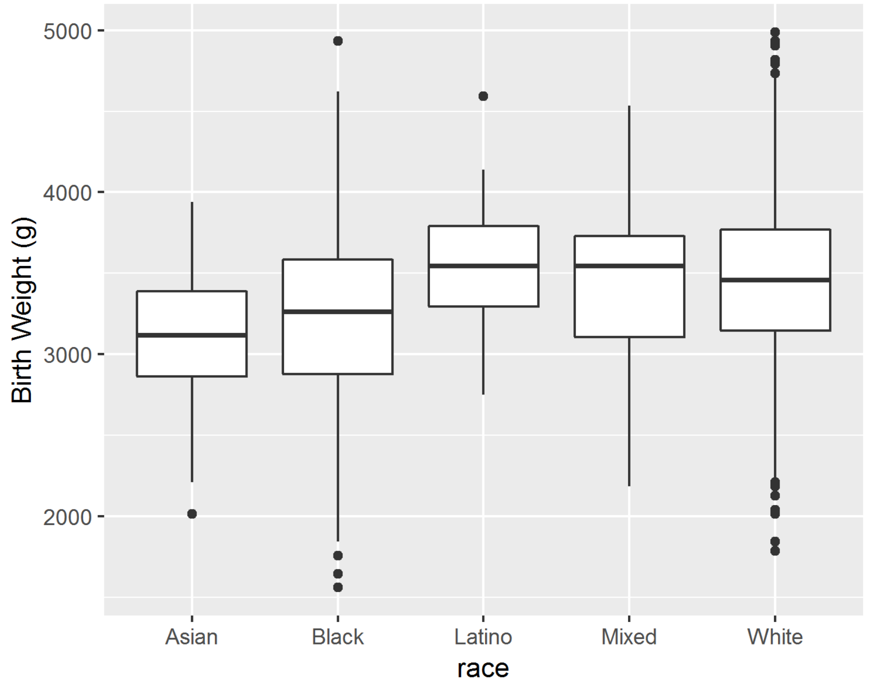
- a data set
- a co-ordinate system
- and *geoms* visual marks that represent data points

To display values, you map variables in the data to visual properties of the geom (*aesthetics*), like size, colour, x-axis and y-axis.

It provides a unified system for graphics, simplifies tasks such as adding legends, and is fully customisable.

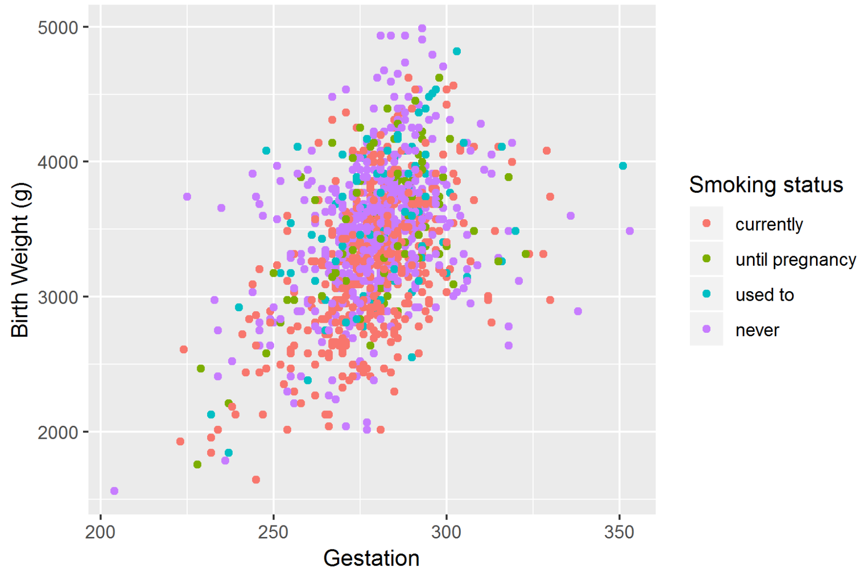
ggplot2 boxplots

```r
library(ggplot2)
ggplot(infant, aes(y = bwt * 28.35, x = race)) +
    geom_boxplot() +
    ylab("Birth Weight (g)")
```



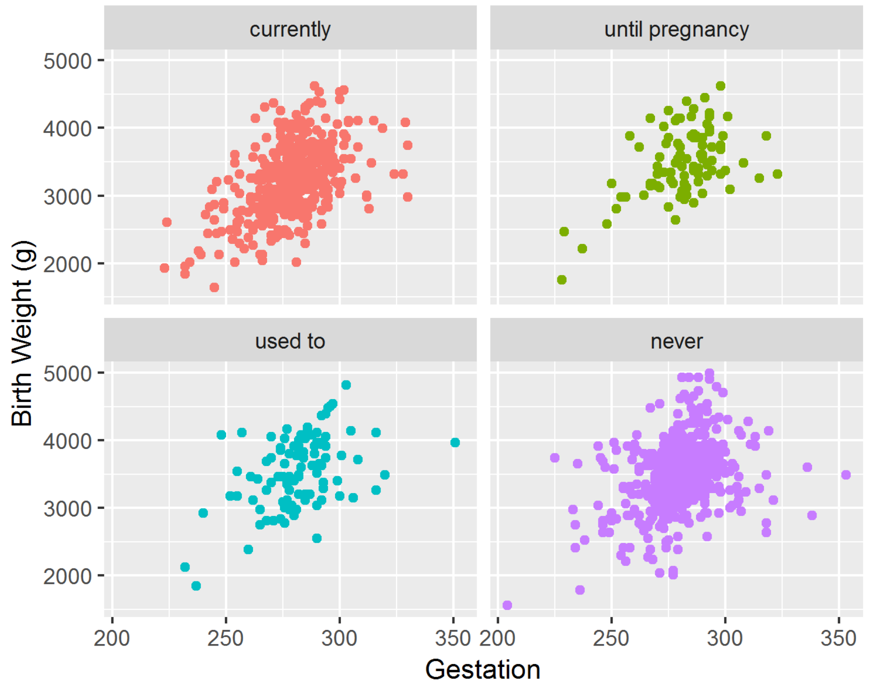
ggplot2 with Colour

```
infant <- mutate(infant, smoke = recode_factor(smoke,
                                               `1` = "currently", `2`= "until pregnancy",
                                               `3` = "used to", `0` = "never"))
p <- ggplot(infant, aes(y = bwt * 28.35, x = gestation, color = smoke)) +
    geom_point() + labs(x = "Gestation", y = "Birth Weight (g)", color = "Smoking status")
p
```



ggplot2 Facetting

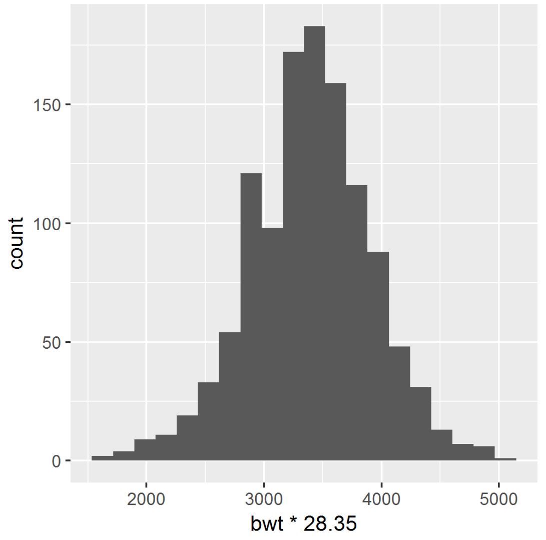
```
p + facet_wrap(~ smoke) + guides(color = FALSE)
```

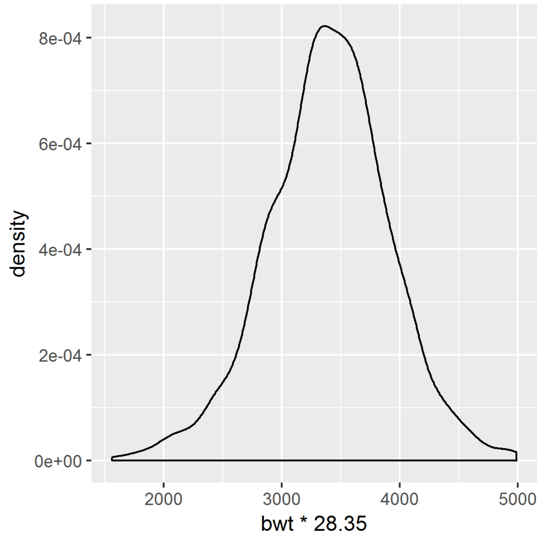


Plots in R Markdown

The chunk options enable you to arrange plots (caption used as alt text in HTML slides)

~~~
```{r, fig.show = "hold", fig.align = "center", fig.width = 4, fig.height = 4,
 ¬ fig.cap = "Left: ggplot histogram. Right: ggplot density curve", out.width = "30%"}
ggplot(infant, aes(x = bwt * 28.35)) + geom_histogram(bins = 20)
ggplot(infant, aes(x = bwt * 28.35)) + geom_density()
```
~~~





Your Turn

Inspect the association between birth weight (`bwt`) and mother's age (`age`) with a ggplot. Use `geom_smooth(method = "lm")` to add a linear smooth with confidence interval.

Does the association depend on smoking status? (Hint: assign one or more aesthetics to `smoke`).

Simple Linear Modelling

The `lm()` function can be used to fit linear models, e.g. a simple linear model of `bwt` vs `gestation`

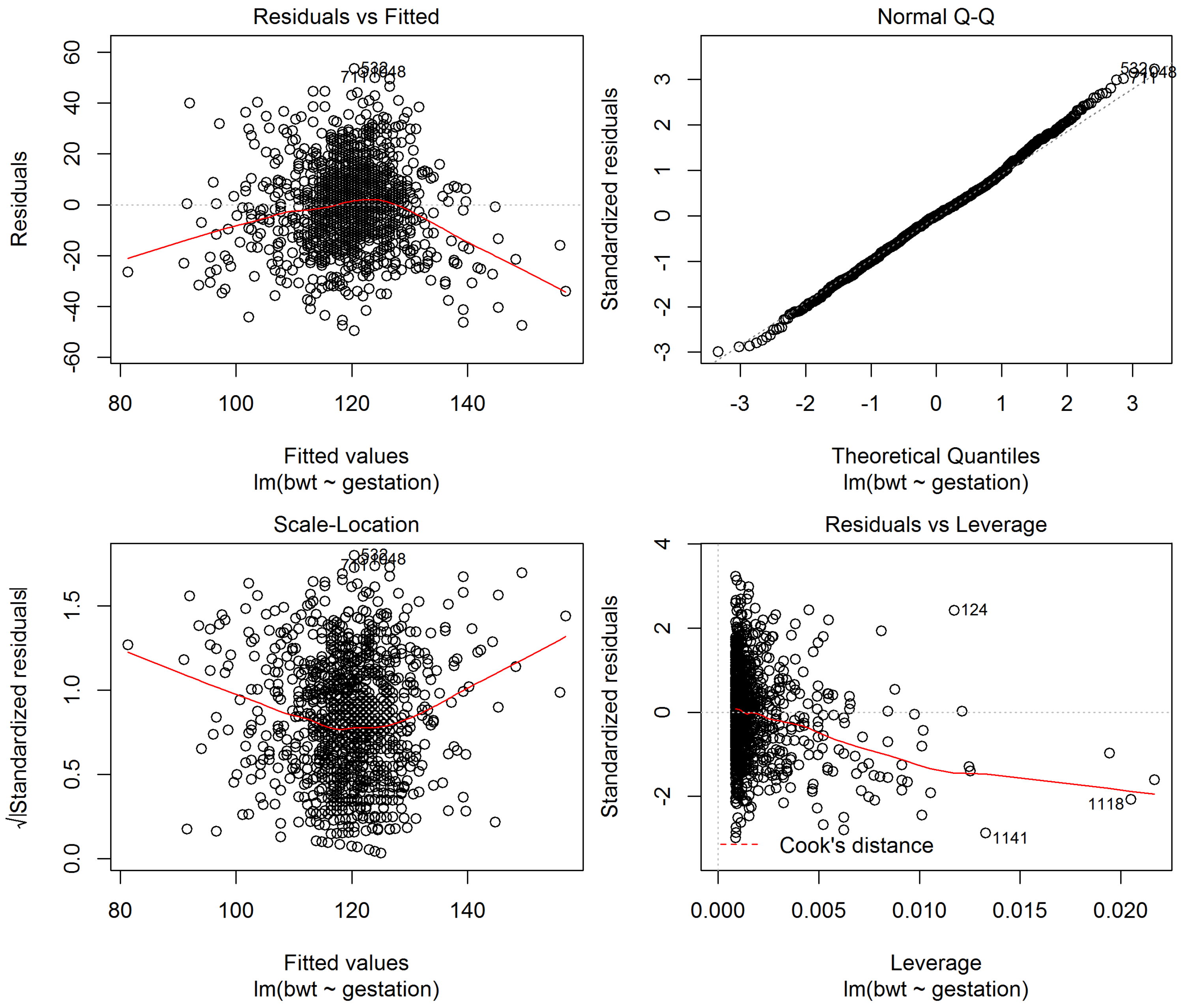
```
model1 <- lm(bwt ~ gestation, data = infant)
model1
```

```
#
# Call:
# lm(formula = bwt ~ gestation, data = infant)
#
# Coefficients:
# (Intercept)    gestation
#     -22.171        0.507
```

Diagnostic plots can be created by plotting the model object

```
plot(model1)
```

(result on next slide).


Residuals vs Fitted
Residuals
Fitted values
lm(bwt ~ gestation)
532
711
1648
Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(bwt ~ gestation)
Scale-Location
√|Standardized residuals|
Fitted values
lm(bwt ~ gestation)
Residuals vs Leverage
Standardized residuals
Leverage
lm(bwt ~ gestation)
124
1118
1141
Cook's distance

Standard summary

```
summary(model1)
```

```
#
# Call:
# lm(formula = bwt ~ gestation, data = infant)
#
# Residuals:
#    Min     1Q Median     3Q    Max
# -49.42 -10.98   0.09  10.05  53.58
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) -22.1707     8.8608    -2.5    0.012 *
# gestation     0.5074     0.0316    16.0   <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 16.6 on 1173 degrees of freedom
# Multiple R-squared:  0.18,    Adjusted R-squared:  0.179
# F-statistic:  257 on 1 and 1173 DF,  p-value: <2e-16
```

broom package

`tidy()` and `glance()` from **broom** extract coefficient and model statistics

```r
library(broom)
tidy(model1)
```

```
# # A tibble: 2 x 5
#   term        estimate std.error statistic  p.value
#   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
# 1 (Intercept)  -22.2      8.86       -2.50 1.25e- 2
# 2 gestation      0.507    0.0316     16.0  1.80e-52
```

```r
glance(model1)
```

```
# # A tibble: 1 x 11
#   r.squared adj.r.squared sigma statistic  p.value    df logLik
# *     <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl>
# 1     0.180         0.179  16.6      257. 1.80e-52     2 -4965.
# # ... with 4 more variables: AIC <dbl>, BIC <dbl>, deviance <dbl>,
# #   df.residual <int>
```

`augment()` can be used to augment the original data with fitted values etc.

Multiple Regression

Additional terms can be added to the model via +

```
y ~ a + b + c
```

We can use `update` to update a fitted model, changing any aspect of the model call. We can update the formula by adding or subtracting terms.

```
update(fit, . ~ . + d - c)
```

All linear models have an intercept by default, it can be removed with `-1`.

`anova()` can applied to a single model to assess the significance of each term or nested models to assess the significance of additional terms.

Your Turn

Use `lm()` to model birth weight against both gestation and the key maternal characteristics weight (`wt`), height (`ht`), parity and race. Use `anova()` to inspect the results.

Now add the smoking variables: smoking status (`smoke`), time since quitting (`time`) and number of cigarettes smoked per day (`number`). Inspect the results - are all the smoking variables significant?

Use `anova(fit1, fit2)` to produce compare the two models.

Learning more/getting support

RStudio cheatsheets (rmarkdown, dplyr, ggplot2) https://www.rstudio.com/resources/cheatsheets/)

R for Data Science (data handling, basic programming and modelling, R markdown) http://r4ds.had.co.nz

RStudio Community (friendly forum focused on "tidyverse" packages, including dplyr, ggplot2) https://community.rstudio.com/

Going further

Quick-R (basic "how to"s from data input to advanced statistics)
http://www.statmethods.net/

Task views http://cran.r-project.org/web/views/ provide an overview of R’s support for different topics, e.g. Bayesian inference, survival analysis, ...

http://www.rseek.org/ – search the web, online manuals, task views, mailing lists and functions.

Package vignettes, many books, e.g. on https://bookdown.org/.