Introduction to R and Data Visualization

Forwards Teaching Team

@R_Forwards
https://forwards.github.io/

updated: 2020-03-14

Why R?

Reasons to learn R

Free and 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

Companies that use R

https://github.com/ThinkR-open/companies-using-r

- AirBnB
- Amazon Web Services
- BBC
- Booking.com
- Buzzfeed
- Ebay
- The Economist
- Facebook
- ...

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
bioconductor.org

bioinformatics
>1500 pkgs

GitHub
github.com

devel pkgs
GitHub-only pkgs

Using R

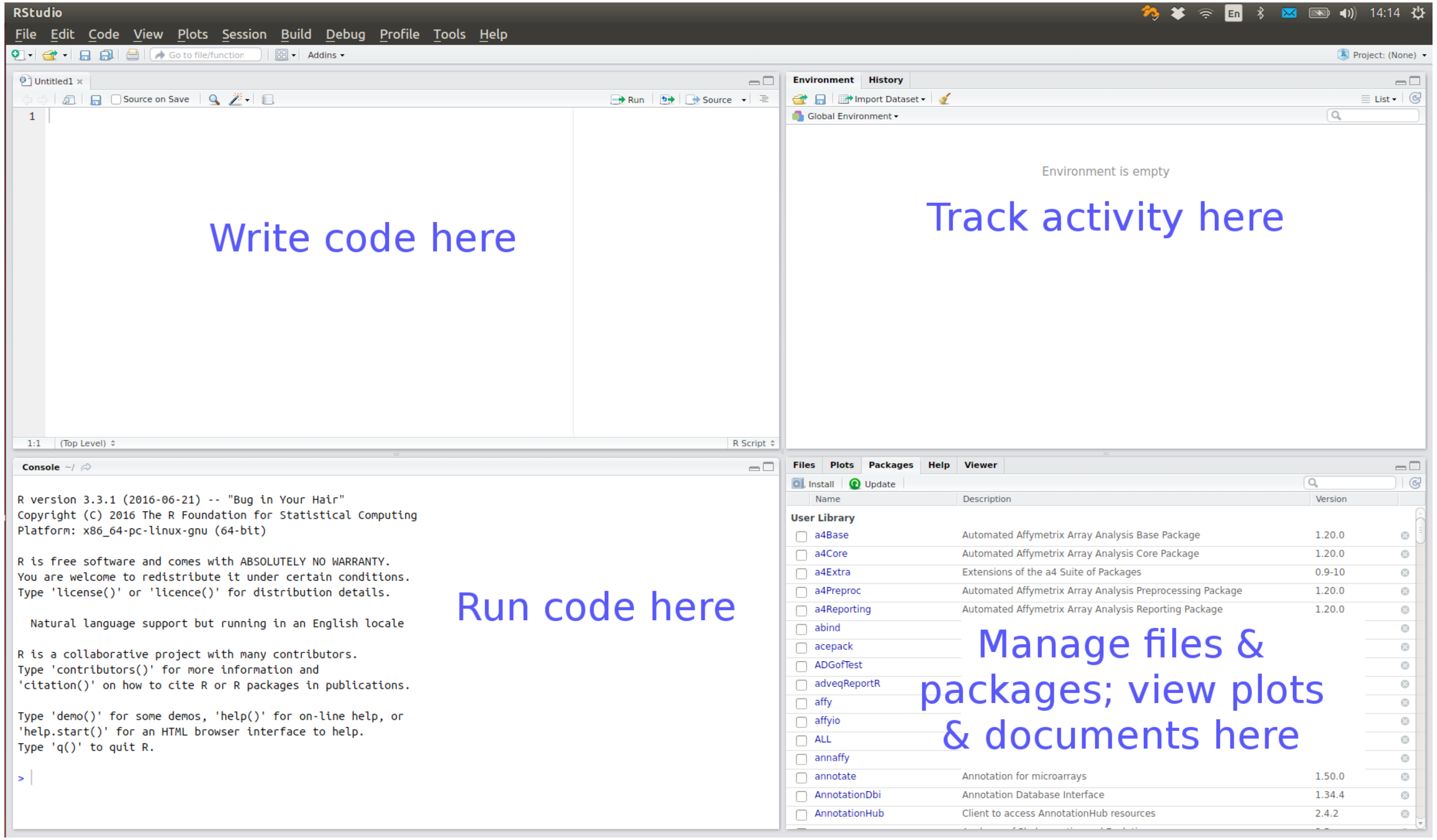
R Code-along

We can type commands directly into the R console

```r
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
```

Using RStudio

RStudio IDE



```
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:

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

Why R Scripts?

Writing an R script for an analysis has several advantages

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

R Scripts in RStudio

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

Code can be sent directly from the source editor as follows

- one function call: `Ctrl` / `⌘` + `↵` or Run
  - several separate function calls: select lines, then run as above
  - whole script: `Ctrl` / `⌘` + `Shift` + `↵` or Source

RStudio Shortcuts from the R Console

RStudio provides a few shortcuts to help run 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

RStudio Code-along

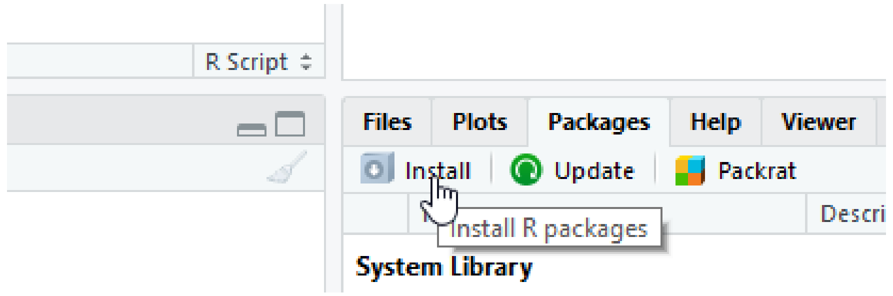
```r
View(iris)
# showing syntax highlighting, code completion, running code, indentation
sum(3, 4)
summary(iris, maxsum = 2,
        digits = 2)
```

Packages

Install Packages

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

CRAN packages can be installed from the Packages tab in RStudio



The corresponding code will be shown in the R console, e.g.

```
install.packages(c("dplyr", "ggplot2"))
```

Your Turn

Install the following packages that we will use later in the workshop

- readr
- readxl
- dplyr
- ggplot2

Using Packages

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

```
library(dplyr)
x <- c(1, 3, 6)
last(x)
```

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.

Working with Data in R

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
y <- c("red", "yellow", "green")
y
# [1] "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 1 1 1 1 1 ...
```

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

Creating a Data Frame and Extracting Values

```
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
```

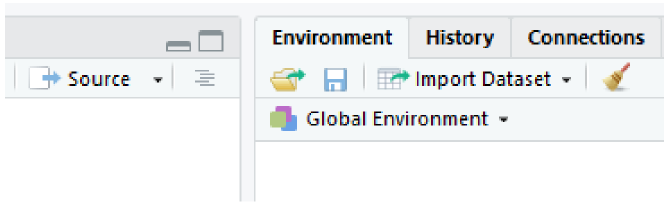
```
mydata$x
```

```
mydata[[1]] # or mydata[["x"]]
```

```
mydata[1, 2:3] # or mydata[1, c("x", "y")]
```

Data Input via Import Dataset

Using the Import Dataset button in RStudio



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

- `.txt` or `.csv` via `read_delim` or `read_csv` from **readr**.
- `.xlsx` via `read_excel` from **readxl**.
- `.sav`/`.por` , `.sas7bdat` and `.dta` via `read_spss`, `read_sas` and `read_dta` 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 of the workshop materials.

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

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` ``

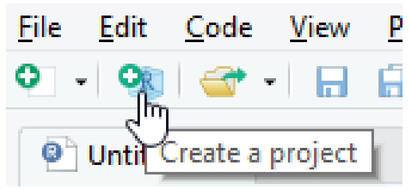
Workflow

R Studio Projects

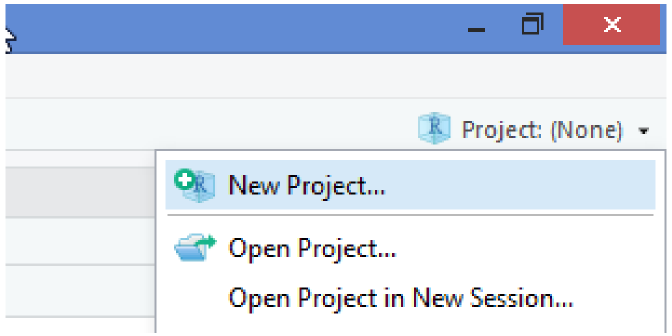
An Rstudio project is a context for work on a specific project

- automatically sets working directory to project folder
- has separate workspace and command history

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.



Your Turn

The `infant` directory in the workshop materials is an RStudio project. Open this folder and double-click on the `infant.Rproj` file to open it. *Hint* the file extension may be hidden on Windows, look for the icon that looks like a light blue box with a dark blue R on top.

From the Files tab, open the `cleaning.R` R script from the `scripts` folder.

Use the Import Dataset button to import the `infant.xlsx` file from the `data` folder.

Copy the code loading the **readxl** package to the "load packages" section of `cleaning.R`.

Copy the code importing the `infant.xlsx` file to the "import data" section of `cleaning.R`.

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:

- cleaning.R = cleaning the raw data
- analysis.R = analysing the tidy data

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

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
- Use an RStudio project to avoid using file paths that are specific to you
- Use `#---` to separate sections (in RStudio Code > Insert Section)
- Give objects and variables meaningful names

Data Wrangling

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

The **dplyr** package can be used for these tasks.

dplyr

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

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

We will look at the most commonly used functions today.

`select()`

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

Select named columns:

```r
library(dplyr)
infant <- select(infant, gestation, sex)
```

Select a sequence of columns, along with an individual column

```r
infant <- select(infant, pluralty:gestation, parity)
```

Drop selected columns

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

Your Turn

In the first R chunk of the `Cleaning data` section, use `select()` to remove the redundant variables `pluralty`, `outcome` and `sex`, as well as characteristics of the father `drace`, ... , `dwt` (Dad's race, ..., Dad's weight).

Overwrite the `infant` data frame with the result.

`filter()`

`filter()` selects rows of data by criteria

For example to keep records where the smoking status is not unknown (`smoke` is not equal to 9) and age is greater than 18:

```
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)`

Your Turn

The variable `gestation` gives the length of the pregnancy in days.

In the second R chunk of the `Cleaning data` section, use `filter()` to exclude extremely premature babies (gestation less than 28 weeks) AND extremely late babies (gestation more than 52 weeks).

HINT You can use `28 * 7` to give the number of days that is equal to 28 weeks.

Run the second filter in this chunk which has been completed for you - this exclude observations where the smoking status or race of the mother is unknown.

`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

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

The `recode_factor()` function from **dplyr** helps to converted numeric codes to a categorical variable (factor), e.g.

```
infant <- mutate(infant,
                 race = recode_factor(race,
                                      `6` = "Latino", `7` = "Black",
                                      `8` = "Asian", `9` = "Mixed",
                                      .default = "White"))
```

Your Turn

In the second R chunk of the `Cleaning data` section, update the variables `ht` and `smoke` as described below.

For `ht`, replace `999` with `NA`.

For `smoke`, recode the numeric codes as follows:

- 1 = currently
- 2 = until pregnancy
- 3 = used to
- 0 = never

Chaining

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

```
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.

Saving Data

Plain text files can be saved using the **readr** package:

- `write_csv()` to write a CSV file
- `write_delim()` to write a tab delimited file However, only the labels will be saved for factors.

To save the tibble as an object that can be reloaded into R, use `saveRDS()`

```
saveRDS(infant, "data/infant.rds")
```

The file path is relative to your project directory (where the .Rproj file is).

To reload the data, use

```
infant <- readRDS("data/infant.rds")
```

Your Turn

Run the code in the last chunk of the "Cleaning data" section, which sources an R script to clean the remaining variables.

Then run the code in the last chunk of `cleaning.R` to save the cleaned data.

In the Environment pane of RStudio, click on the broom to remove all objects from your current workspace.

Open the next script, `analysis.R` and run the code in the "Import data" section to reload the clean data.

Table Summaries

`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

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

```
# A tibble: 1 x 3
  `Mean mother's weight (kg)` `Sample size` `Total sample size`
                        <dbl>         <int>               <int>
1                        129.          1163                1198
```

Grouped Operations

`group_by()` sets grouping on a data frame. This is most useful for `summarise()`

```
infant %>%
    group_by(`Ethnic group` = race) %>%
    summarise(`Mean weight (kg)` = mean(wt, na.rm = TRUE)) %>%
    ungroup()
```

```
# A tibble: 5 x 2
  `Ethnic group` `Mean weight (kg)`
  <fct>                       <dbl>
1 Latino                       119.
2 Black                        138.
3 Asian                        110.
4 Mixed                        129.
5 White                        127.
```

It is good practice to `ungroup()` at the end to avoid affecting later analyses.

Your Turn

An infant is categorised as low weight if its birth weight is ≤ 2500 grams, regardless of gestation. The `bwt_cat` factor categories the birth weight into the following categories: (1500, 2000], (2000, 2500], (2500, 3000], (3000, 3500], and (3500, 5000].

In your `analysis.R` script, load the **dplyr** package in the "Load packages" section, so that you can use the functions from **dplyr**.

In the "Count by birth weight category" section, complete the pipeline to count the number of infants in each level of `bwt_cat`.

Plotting in R

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.

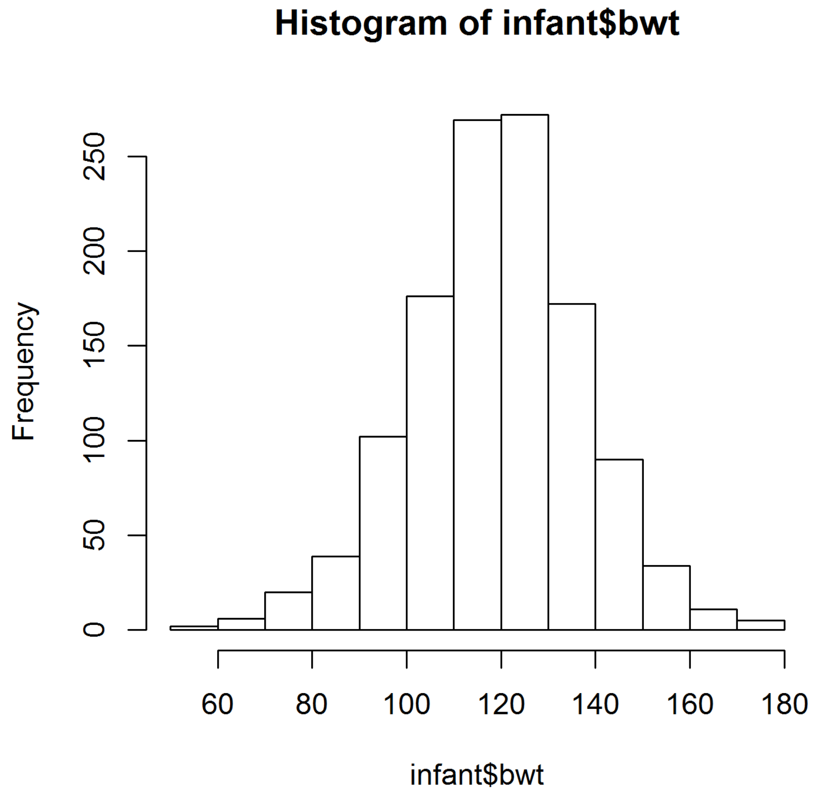
Graphs can be saved in various formats using the Export drop down menu, which also has an option to copy to the clipboard.

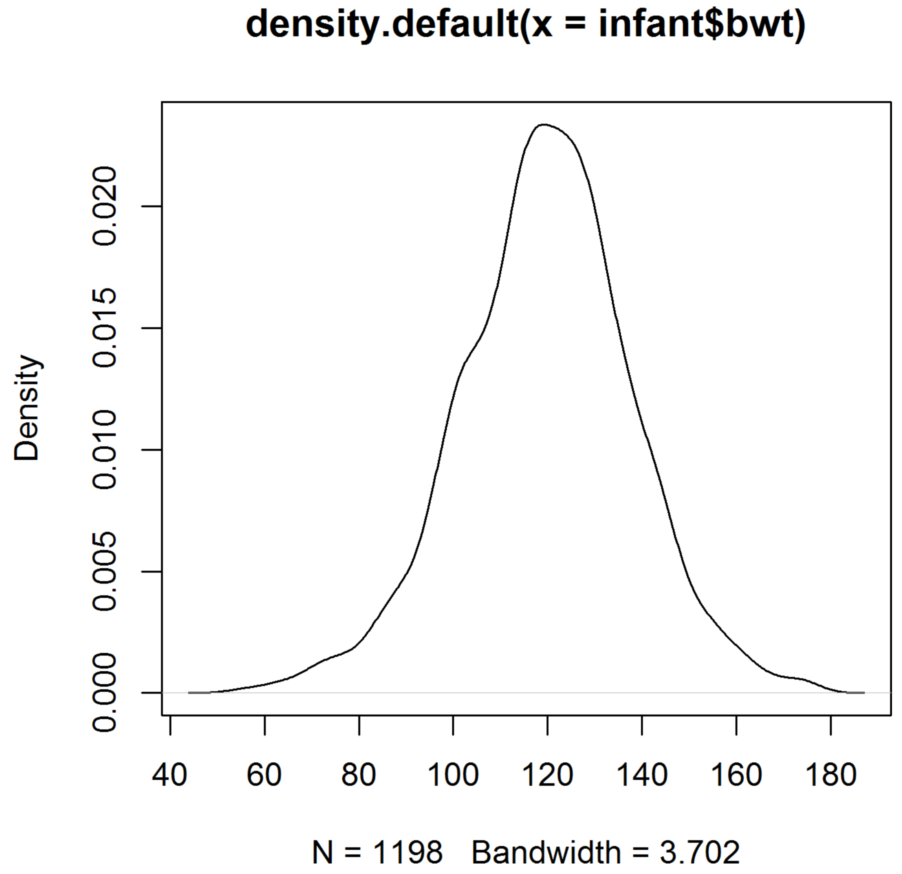
First we consider "no-frills" plots, for quick exploratory plots.

Base R Plots

Histogram/Density

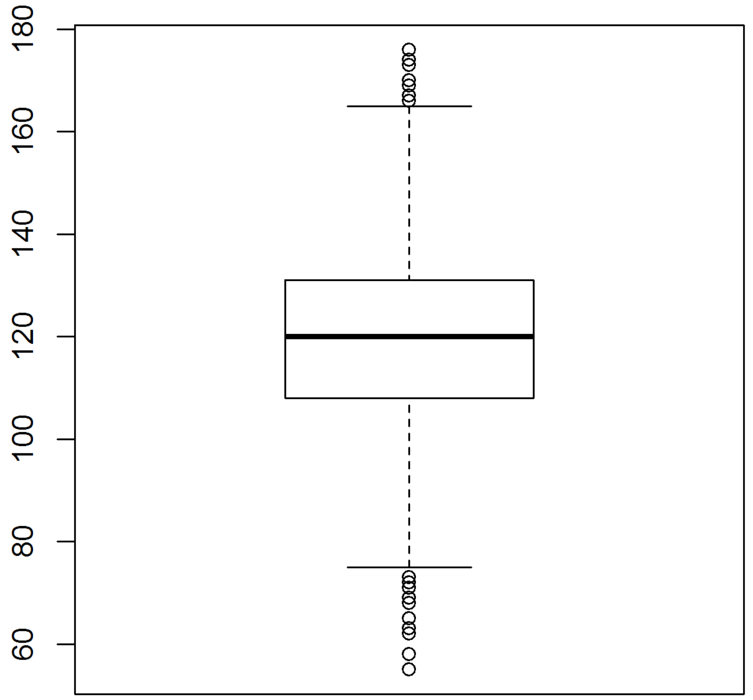
```
hist(infant$bwt)
plot(density(infant$bwt))
```

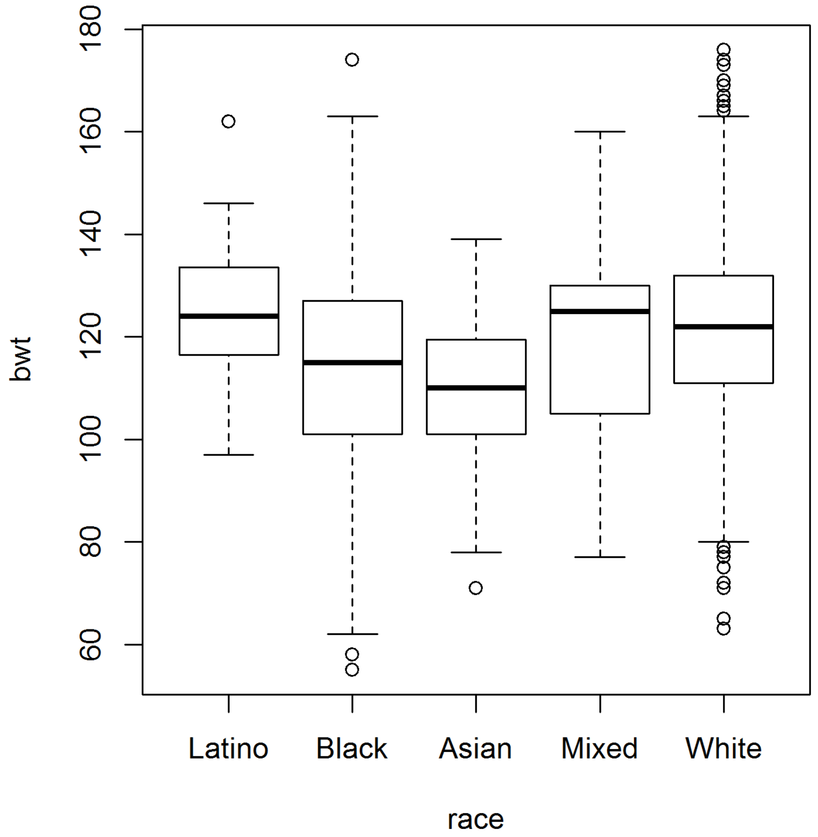




Boxplots

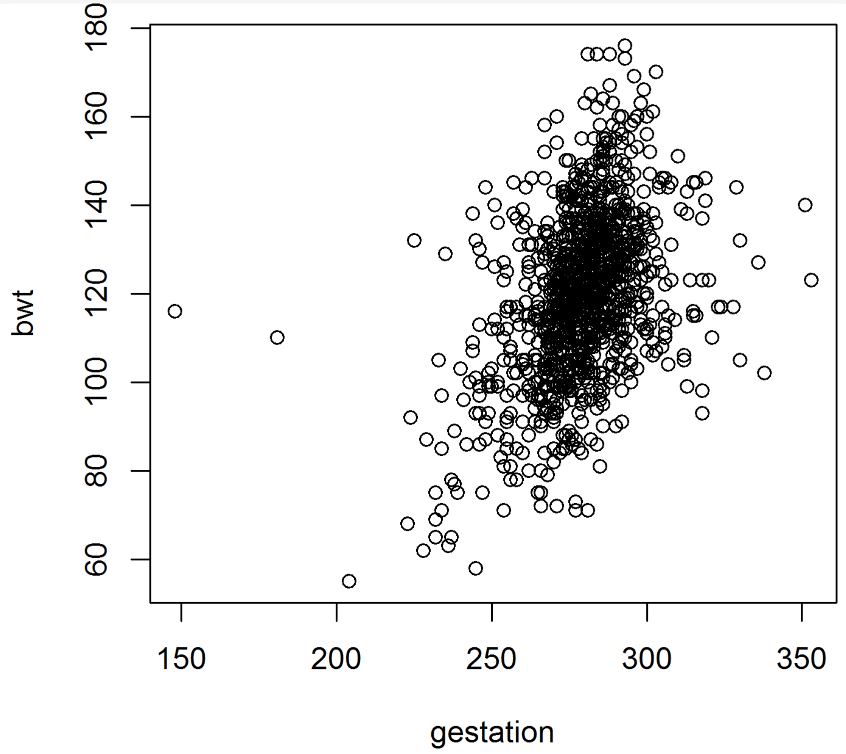
```
boxplot(infant$bwt)
boxplot(bwt ~ race, data = infant)
```





Scatterplots

```
plot(bwt ~ wt, data = infant)
# plot(infant$wt, infant$bwt); plot(x = infant$wt, y = infant$bwt)
```



Your Turn

In the section "Explore association of bwt with gestation", plot `bwt` on the y axis against `gestation` on the x axis.

You should see two outliers, where the gestation is very low (less than 180 days, i.e. 6 months), but the birth weight is average.

These two observations look suspect, it is likely they have been recorded incorrectly, so we will exclude them from the rest of the analysis. Run the code in the "Filter outliers" section to exclude them.

ggplot2 Plots

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

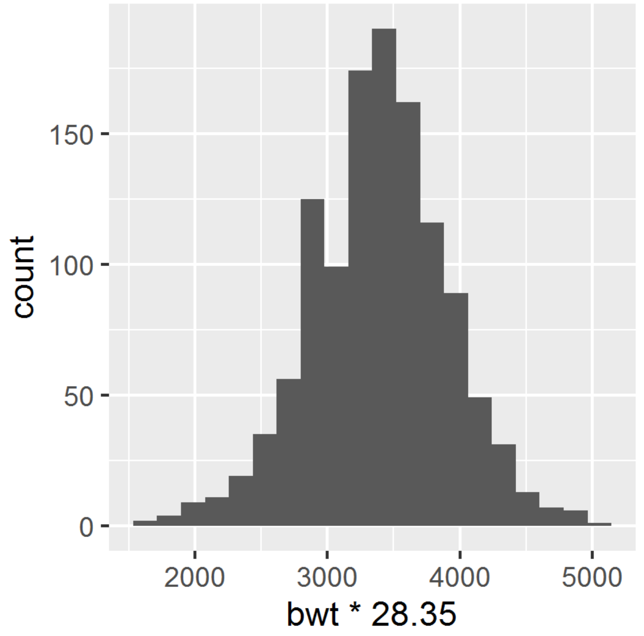
- a data set
- a co-ordinate system
- and *geoms* visual marks that represent data points

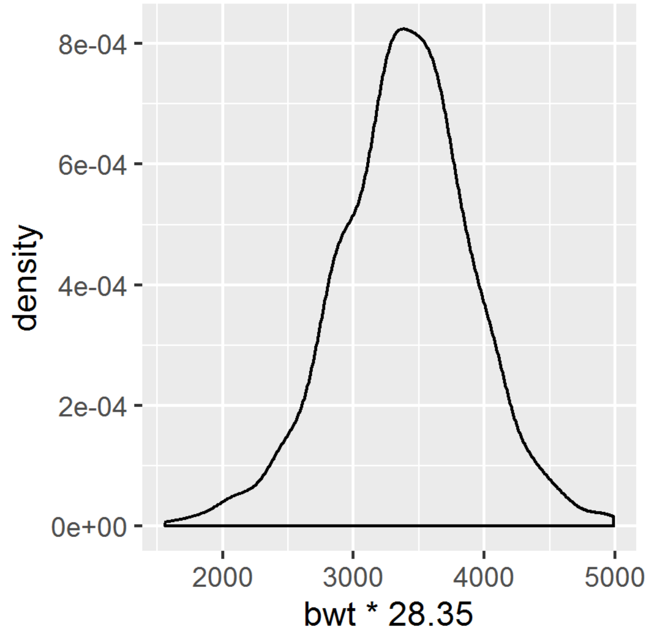
To display values, you map variables in the data to visual properties of the geom (*aesthetics*), like size, colour, x-axis and y-axis.

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

ggplot2 histogram/density plot

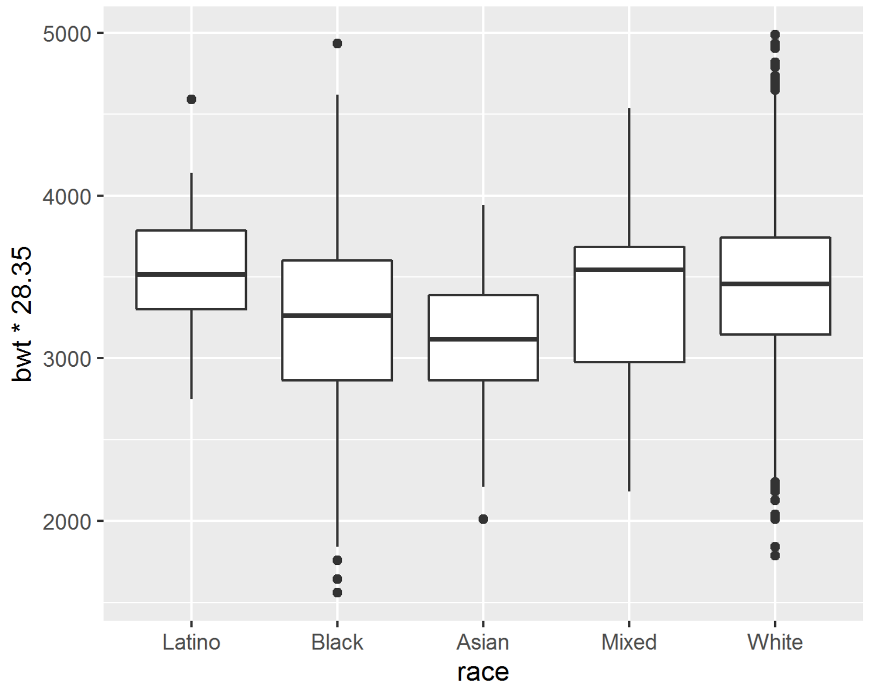
```r
library(ggplot2)

ggplot(infant, aes(x = bwt * 28.35)) +
  geom_histogram(bins = 20)

ggplot(infant, aes(x = bwt * 28.35)) +
  geom_density()
```





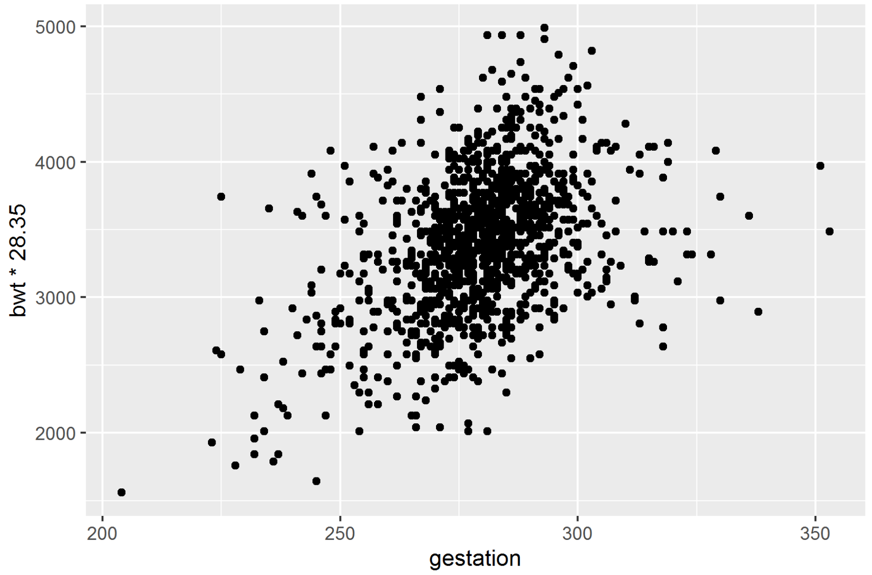
ggplot2 boxplots

```
ggplot(infant, aes(y = bwt * 28.35, x = race)) +
    geom_boxplot()
```



ggplot2 scatterplot

```
ggplot(infant, aes(y = bwt * 28.35, x = gestation)) +
    geom_point()
```



Your Turn

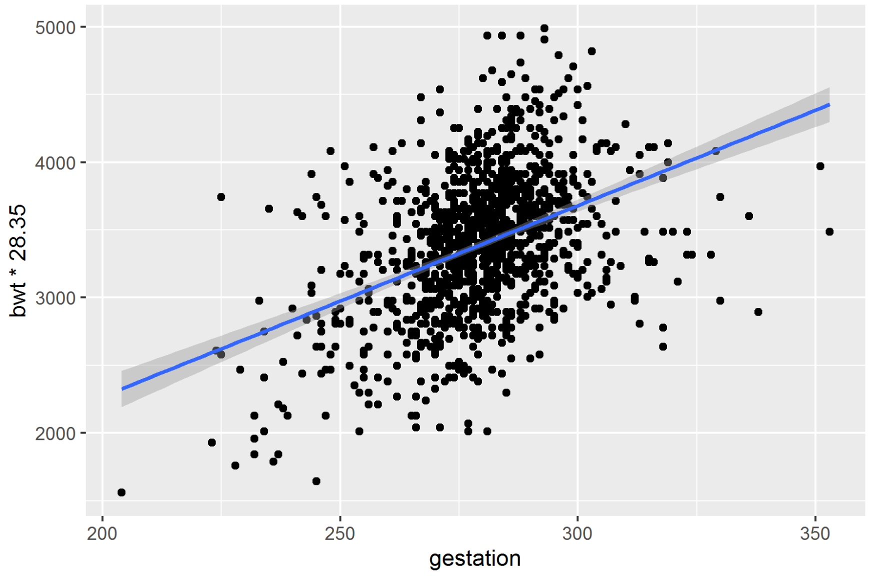
Load the **ggplot2** package in the "Load packages" section, so that you can use the functions from **ggplot2**.

In the "Explore association of birth weight with mother's age" section, use `ggplot()` to create a scatterplot of birth weight in grams on the y-axis (`bwt * 28.35`) against mother’s age (`age`) on the x axis.

Multiple geoms

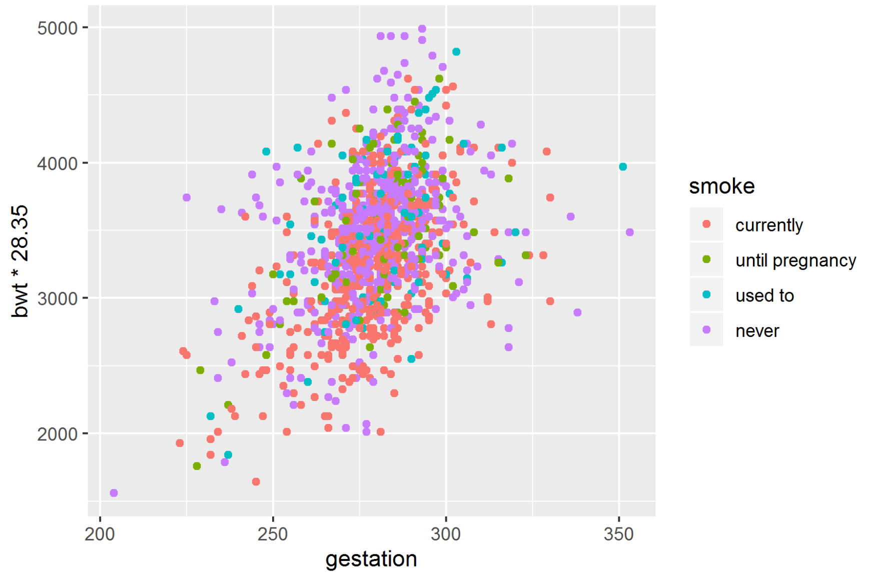
We can add more than one geom to a plot, e.g. we can add a linear smooth with confidence interval as follows

```
ggplot(infant, aes(y = bwt * 28.35, x = gestation)) +
    geom_point() +
    geom_smooth(method = "lm")
```



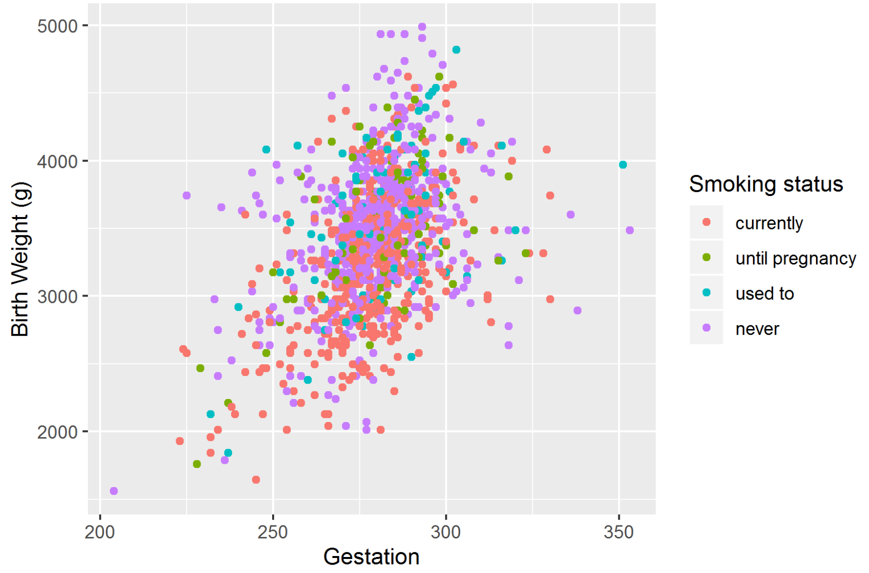
Colour by group

```r
p <- ggplot(infant, aes(y = bwt * 28.35, x = gestation, color = smoke)) +
    geom_point()
p
```



Customising labels

```
ggplot(infant, aes(y = bwt * 28.35, x = gestation, color = smoke)) +
   geom_point() +
   labs(x = "Gestation", y = "Birth Weight (g)", color = "Smoking status")
```

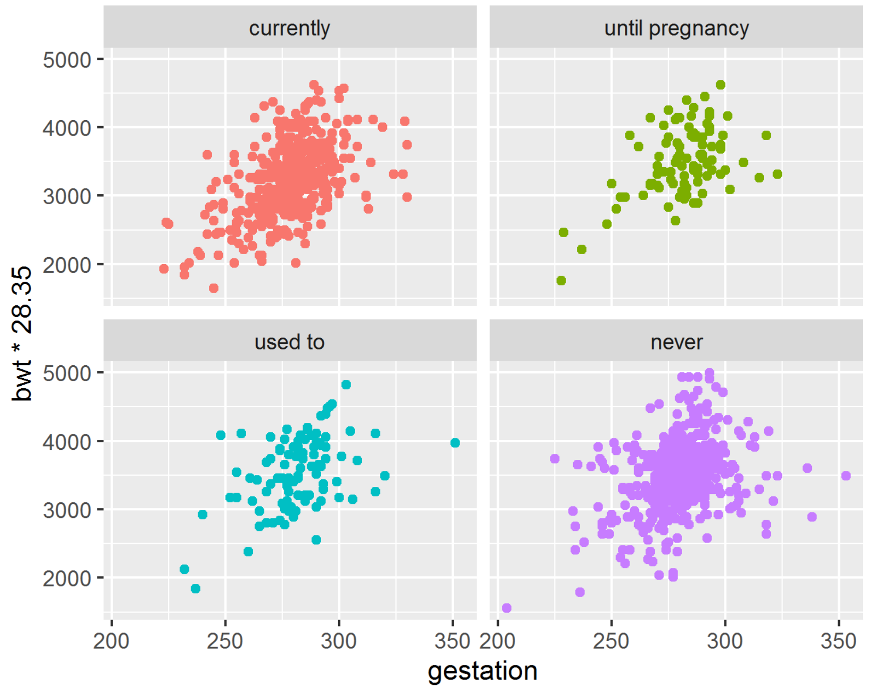


Or equivalently:

```
p + labs(x = "Gestation", y = "Birth Weight (g)", color = "Smoking status")
```

ggplot2 Facetting

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



Your Turn

Use `geom_smooth(method = "lm")` to add a linear smooth with confidence interval to your plot from the last exercise.

Use `labs` to customise the x and y axis labels to say "Birth Weight (g)" and "Mother's Age",

Update the aesthetics (inside the call to `aes()`) to colour the points by smoking status (`smoke`). Update the call to `labs()` to make the legend title "Smoking status".

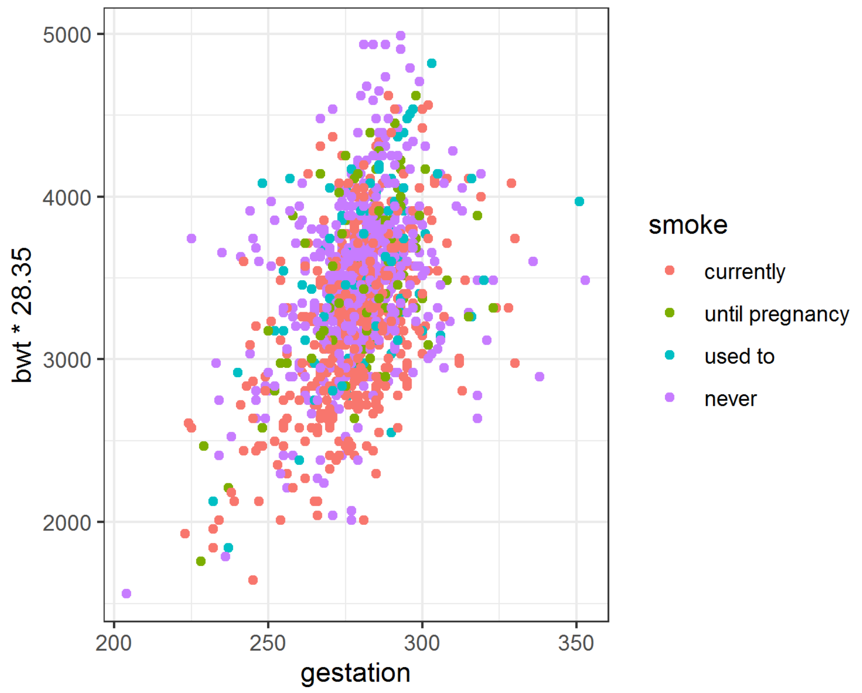
Look at the help file for `labs()` to find out how to add a title to the plot, then add the title "Birth Weight by Mother's Age".

Try mapping `smoke` to different aesthetics, e.g. `shape`, `size`, `alpha`. What happens if you map to more than one aesthetic?

Themes

Themes control the display of all non-data elements of the plot. There are some default themes, e.g.

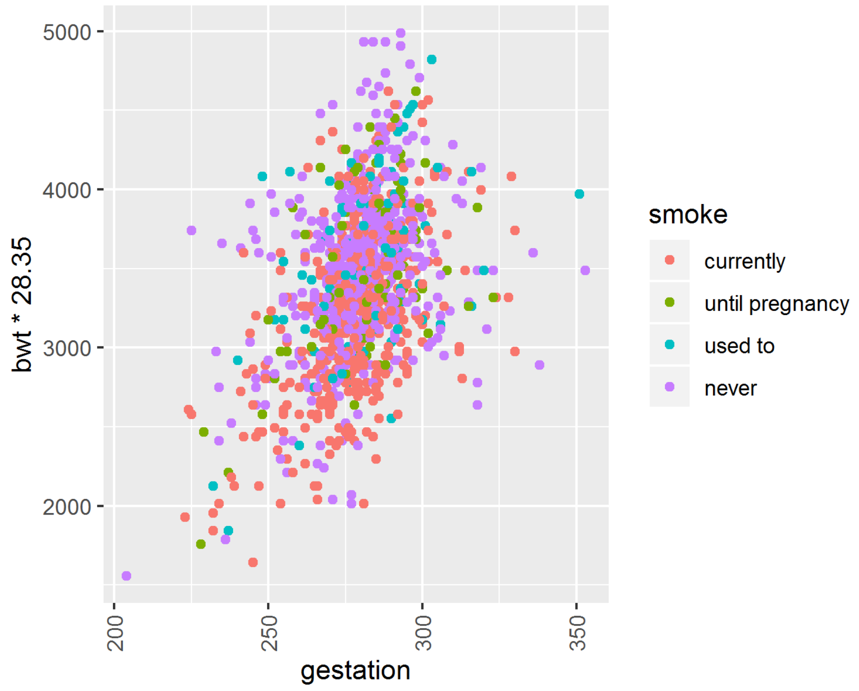
```
p + theme_bw()
```



Themes

We can also control individual elements of the theme, e.g.

```r
p + theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



Saving Plots

Once we have a plot we are happy with we can save it using `ggsave()`

```
ggsave("bwt_vs_gestation.png", width = 4, height = 4, units = "in")
```

We can also save from the plot window, but using `ggsave()` makes it easier to save the plot in the same format if we edit the plot later.

Your Turn

(If time allows) Go to https://ggplot2.tidyverse.org/reference/index.html and scroll to the Themes section, to see the complete themes that are available. Try out different themes.

When you are finished, save your plot with `ggsave()`.

Learning more/getting support

Package vignettes (dplyr)

Package website, e.g. https://ggplot2.tidyverse.org/

RStudio cheatsheets (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.

Many books, e.g. on https://bookdown.org/.

License