A Guided Tour of R

Who am I?

Dr Sean Murphy

- Post-doctoral research fellow at the University of Melbourne, Australia
- Focus on large and complex datasets (e.g. experience sampling, speed dating, social media data)
- Learned R as a PhD student

What is R?

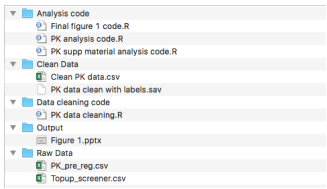
- Not (just) a statistics package
- A programming language designed for data wrangling, plotting, and statistics

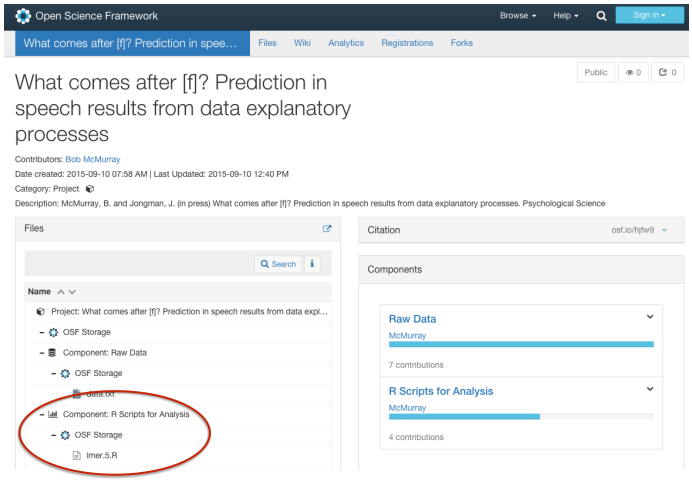
All statistical techniques under one roof

- Structural Equation Modelling
- Multi-level Modelling
- Longitudinal analysis
- Dyadic data
- Power Analysis
- Social Network Analysis
- Linguistic/text Analysis
- Bayesian Stats
- Simulations
- Neural Network Models

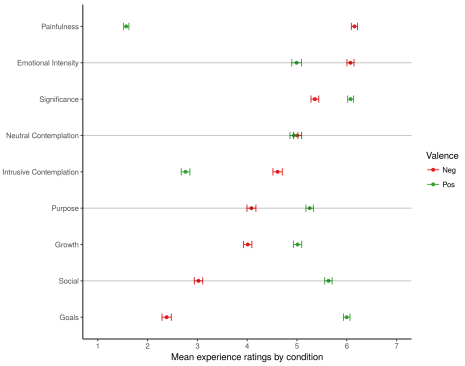
Reproducible research

- Basic principle: You (or someone else) should be able to get from your raw data to your final results and get the same results every time

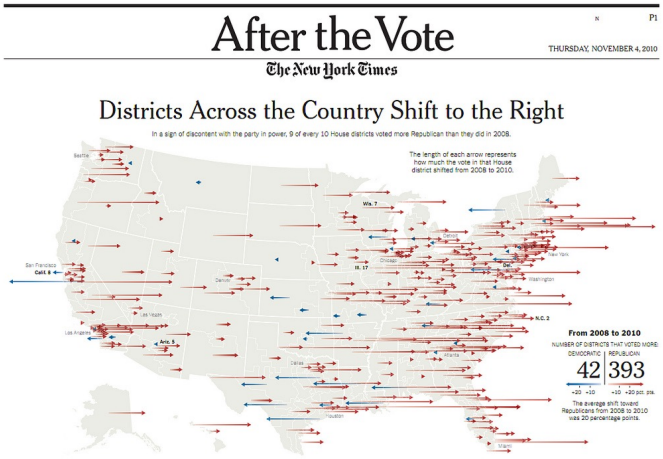


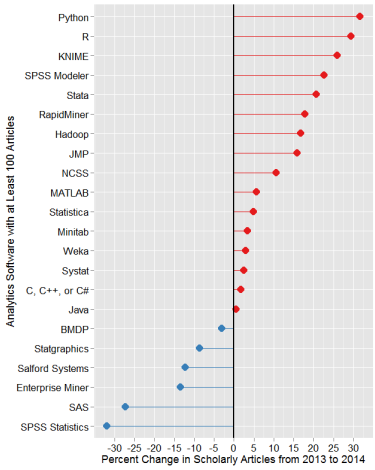


Excellent graphics



Excellent graphics





So what's the catch?

- No drop-down menus, everything in code
- Fluency takes time and practice
- BUT you can get a long way by learning the basics

What we're going to cover

1. The basics of R and the RStudio interface
2. Wrangling our data
3. Visualising our data
4. Analysing our data

What we're going to cover

1. The basics of R and the RStudio interface
2. Wrangling our data
3. Visualising our data
4. Analysing our data

Functions

- function_name(arg1 = val1, arg2 = val2, …)
- Take inputs (called *arguments*) and perform some *function* based on these arguments, returning an *output*

Functions

- function_name(arg1 = val1, arg2 = val2, …)
- seq(from = 1, to = 10, by = 2)
- seq(1, 10, 2)

Packages

Packages

- Need to be separately installed (once)
  - install.packages('tidyverse')
- Then loaded (every time you restart R)
  - library(tidyverse)

Reading in data

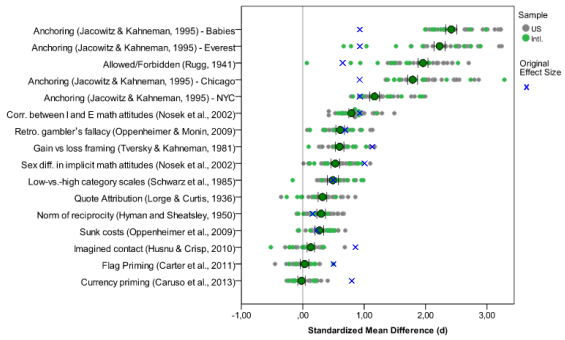
- There is no dropdown menu to read in your data to R!
- Instead, we use functions like read_csv or read_sav to load data from files into R objects

Reading in data

- For csv data
  - mydat <- read_csv('data file name.csv')
- For SPSS data (requires haven)
  - mydat <- read_sav('data file name.sav')
- For excel data (requires readxl)
  - mydat <- read_excel('data file name.xls')

1. The basics of R and the RStudio interface
2. Wrangling our data
3. Visualising our data
4. Analysing our data

Manylabs dataset



Examining your data

- Look at the first few rows and variables:
  - head(data, n = 6)
- Look at the last few rows and variables:
  - tail(data, n = 6)
- Check the variable names:
  - colnames(data)
- See all the variables spread out:
  - glimpse(data)
- See some basic descriptives for each variables
  - summary(data)
- Look at your data directly:
  - View(data)

Data wrangling

- Examine specific variables
- Sort data by values
- View summary statistics
- Remove participants who fail manipulation checks
- Remove extra variables
- Mean-centre variables
- Calculate scales
- Reverse-score variables

dplyr: data wrangling in R

- select (select variables of your choice)
- filter (choose rows based on a criterion)
- arrange (sort data according to criteria)
- mutate (calculate/create new variables)
- summarise (calculate a summary from variables)

id	sex	height	weight
1	Male	165	87
2	Male	176	80
3	Male	182	67
4	Male	170	75
5	Male	190	73
6	Female	165	50
7	Female	135	46
8	Female	150	55
9	Female	153	64
10	Female	142	60

select

- Select columns from a data frame
- Example:
  - “Take my data, and give me just the height and weight columns”
  - select(data, height, weight)

select

- “Select the first two columns”
  - select(data, 1:2)
- “Select variables that end with ‘eight’”
  - select(data, ends_with(‘eight’))
- “Select everything except the ‘id’ variable”
  - select(data, -id)

select

- “Select the first two columns”
  - select(data, id:sex)
- “Select variables that end with ‘eight’”
  - select(data, ends_with(‘eight’), -weight)

filter

- Choose rows based on a specific criterion
- Example:
  - “Take my data, and give me only rows with male participants”
  - filter(data, sex == “Male”)

filter

- “Keep only participants who are above 170cm and more than 60kg”
- filter(data, height > 170 & weight > 60)
- “Keep only women, or men who are shorter than 170cm”
- filter(data, sex == ‘Female’ | height < 170)

missing data (NA)

```
NA > 5
#> [1] NA
10 == NA
#> [1] NA
NA + 10
#> [1] NA
NA / 2
#> [1] NA
```

```
NA == NA
#> [1] NA
```

Source: R for Data Science, Hadley Wickham

missing data (NA)

```
# Let x be Mary's age. We don't know how old she is.
x <- NA

# Let y be John's age. We don't know how old he is.
y <- NA

# Are John and Mary the same age?
x == y
#> [1] NA
# We don't know!
```

Source: R for Data Science, Hadley Wickham

filter

- To test if data is missing, we use the is.na() function
- “Show me rows where sex if missing”
- filter(data, is.na(sex))

pipes (%>%)

- Allow you to combine multiple verb operations
- We use %>% (the ‘pipe’ operator) at the end of each line to tell R we’re not done yet
- Can be read as “and then”

pipes (%>%)

- Imagine we want to select only male participants, but then also drop the gender column. We could do this in two steps, saving the data each time:

data_men <- filter(data, sex = “Male”)

data_men <- select(data_men, -sex)

pipes (%>%)

- To do this all in one step, we could wrap one function inside the other:

```
select(filter(data, sex = "Male"), -sex)
```

- But piping allows us to unpack this and make it easier to read:

```
data %>%
    filter(sex == "Male") %>%
    select(-sex)
```

arrange

- Arranges rows in ascending order based on a criterion
- Example:
  - "Sort my data from lowest to highest weight"
  - arrange(data, weight)

arrange

- "Sort my data from highest to lowest weight"
  - arrange(data, desc(weight))
- "Sort my data by weight, breaking ties with height"
  - arrange(data, weight, height)

mutate

- Calculate new variables in a dataset
- Example:
  - "Take my data, and create a BMI variable by dividing weight by height squared"
  - mutate(data, bmi = weight/(height^2))

summarise

- Calculates a summary statistic (e.g. mean) from your data.
- Example:
  - "Find the mean height in my data"
  - summarise(data, height_mean = mean(height))

group_by

- Groups data in a dataset by a specific variable. After this, calculations will be done by-group.
- Example:
  - "Group my data by sex"
  - group_by(data, sex)

summarise

- Summarise is most useful when combined with group_by to give summaries by group
  - "Find the mean height by sex in my data"
  - data %>%
    group_by(sex) %>%
    summarise(height_mean = mean(height))

summarise

- We can compute multiple summary statistics
  - "Find the mean, median and sd of height by gender in my data"
  - data %>%
    group_by(sex) %>%
    summarise(h_mean = mean(height),
    h_median = median(height),
    h_sd = sd(height))

Data wrangling review

- Five fundamental data wrangling verbs
  - filter (select rows)
  - select (select variables of your choice)
  - arrange (sort data according to criteria)
  - mutate (calculate/create new variables)
  - summarise (calculate a summary from variables)
- Can all by connected through pipes %>%
- mutate and summarise can perform grouped operations with group_by

Saving your data

- Once you're finished working with you're data, you'll want to save it to a file, as objects in your environment generally won't stick around after you close R
- write_csv(data, "my clean data.csv")

1. The basics of R and the RStudio interface
2. Wrangling our data
3. Visualising our data
4. Analysing our data

Plotting in R

- We're going to learn how to create graphics in R using ggplot2
- Based on the concept of a 'grammar of graphics'

The ggplot system

"In brief… a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinates system."

- Hadley Wickham (ggplot2 creator)

The ggplot system

- Moves away from a defined set of graphs (e.g. 'scatterplot', 'bar plot')
- Instead, breaks graphics down to their basic components, and allows you to build them up layer by layer

Building a plot

1. First, we define the dataset we're using
2. Next, we specify aesthetics (color, size, position) that will map our data to a geom
3. Next, we choose what geom (geometric shape) we want to use to represent our data

Gapminder

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.801	8425333	779.4453145
Afghanistan	Asia	1957	30.332	9240934	820.8530296
Afghanistan	Asia	1962	31.997	10267083	853.10071
Afghanistan	Asia	1967	34.02	11537966	836.1971382
Afghanistan	Asia	1972	36.088	13079460	739.9811058
Afghanistan	Asia	1977	38.438	14880372	786.11336
Afghanistan	Asia	1982	39.854	12881816	978.0114388
Afghanistan	Asia	1987	40.822	13867957	852.3959448
Afghanistan	Asia	1992	41.674	16317921	649.3413952
Afghanistan	Asia	1997	41.763	22227415	635.341351
Afghanistan	Asia	2002	42.129	25268405	726.7340548
Afghanistan	Asia	2007	43.828	31889923	974.5803384
Albania	Europe	1952	55.23	1282697	1601.056136
Albania	Europe	1957	59.28	1476505	1942.284244

Building a plot

- First, we define the dataset we're using:

  ggplot(data = gapminder)

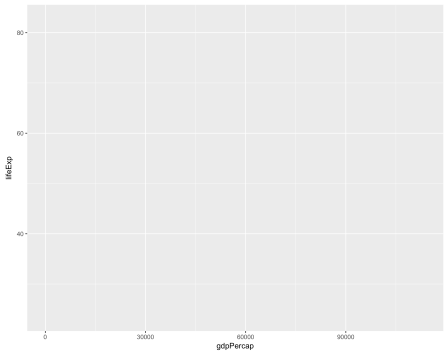
ggplot(data = gapminder)

Building a plot

- Then, we specify our aesthetics

```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp))
```

```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp))
```

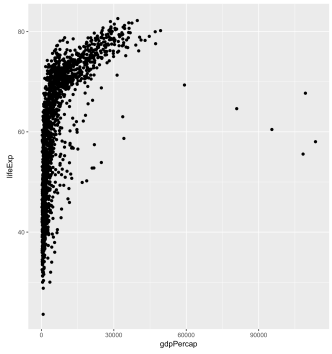


Building a plot

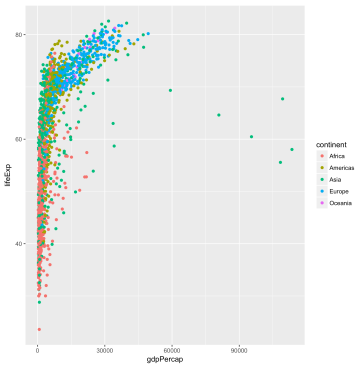
- Finally, we add a geom

```
ggplot(data = gapminder, aes(x = gdpPercap, y = lifeExp)) +
    geom_point()
```

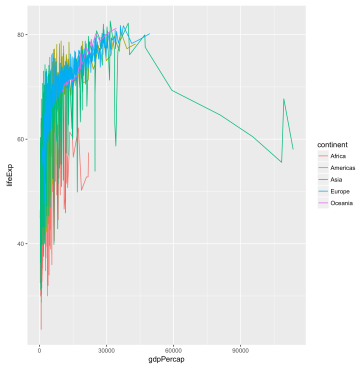
```
ggplot(data = gapminder,
    aes(x = gdpPercap, y = lifeExp)) +
    geom_point()
```



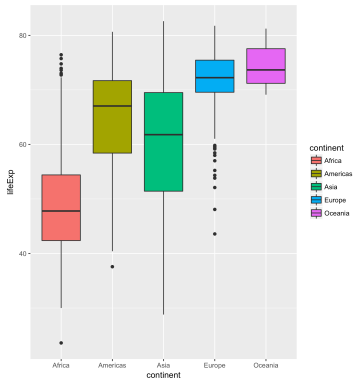

```
ggplot(data = gapminder,
    aes(x = gdpPercap, y = lifeExp, color = continent) +
    geom_point()
```




```
ggplot(data = gapminder,
    aes(x = gdpPercap, y = lifeExp, color = continent) +
    geom_line()
```



```
ggplot(data = gapminder,
       aes(x = continent, y = lifeExp, fill = continent) +
       geom_boxplot()
```



geoms

- geom_point
- geom_line
- geom_boxplot
- geom_violin
- geom_density
- geom_bar
- geom_histogram
- geom_smooth
- geom_col
- geom_errorbar

Aesthetics

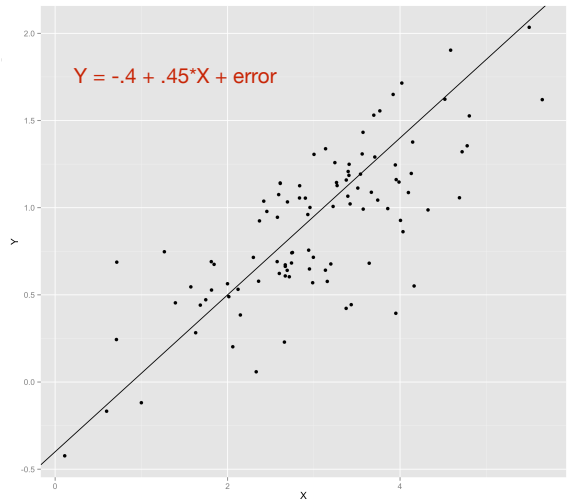
- x, y
- group
- color
- fill
- shape
- size
- alpha
- linetype

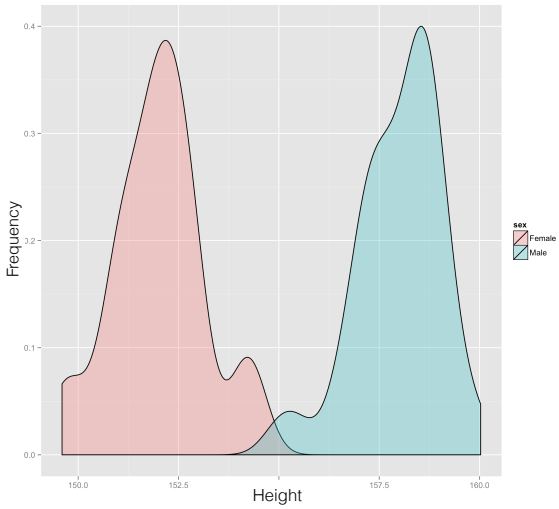
1. The basics of R and the RStudio interface
2. Wrangling our data
3. Visualising our data
4. Analysing our data

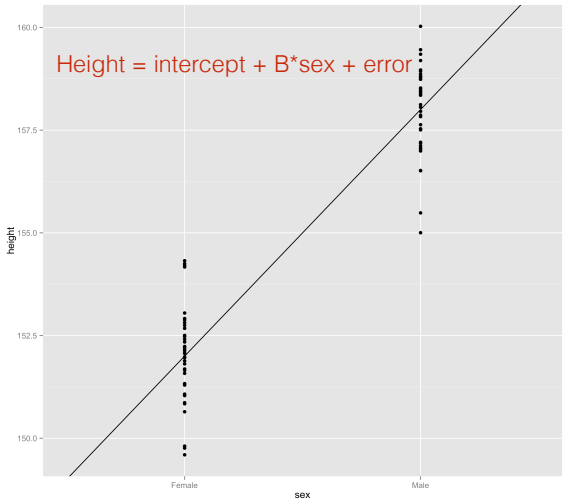
Statistics in R

The General Linear Model

- Y = a + BX + e
- $Y = B_0 + B_1X + e$
- Y = Intercept + Slope*(predictor) + error







Linear models in R

- The syntax for a linear model in R looks like this:
  - y ~ 1 + x
- Or, equivalently:
  - y ~ x
- R uses this same formula for pretty much all analyses falling under the general linear model

Different analyses in R

(Assume X and Z are continuous variables, V is a two-level categorical and W a three-level categorical variable)

- Simple regression
  lm(Y ~ X)
- Multiple regression
  lm(Y ~ X + Z)
- T-Test
  lm(Y ~ V)
- One-way ANOVA
  lm(Y ~ W)
- One-way ANCOVA
  lm(Y ~ W + X)
- Regression with interaction
  lm(Y ~ X*Z)
- Two-way anova
  lm(Y ~ V*W)
- Moderation
  lm(Y ~ V*X) or lm(Y ~ W*X)

- Fitting a three-way ANOVA, with our experimental condition (treat) predicting our dependent variable (depression):

  lm(depression ~ treat, data = treatdata)

- Fitting a three-way ANCOVA, with our experimental condition (treat) predicting our dependent variable (depression), controlling for age:

  lm(depression ~ treat + age, data = treatdata)

- Fitting a regression model, where extraversion, social support (socsup), and their interaction, predict depression:

  lm(depression ~ extraversion * socsup, data = treatdata)

How R handles analysis

- Because R treats so many different analyses by fitting them to the general linear model, the lm() function doesn't show a lot of output by default.
- Instead, we generally analyse data in two steps. We fit a model with a function like lm(), then use other functions to view the results depending on what we're interested in.

How R handles analysis

So we might start by fitting a model to our toy dataset from before. We do this by assigning the output of lm() to a variable. Here we predict height from sex.

model <- lm(height ~ sex, data = data)

Viewing results

- If we want anova-style output:
  - anova(model)
- If we want to look at regression-style output:
  - summary(model)

Reading anova output

anova(model):

```
Analysis of Variance Table

Response: height
          Df  Sum Sq Mean Sq F value    Pr(>F)
sex        1  7095.7  7095.7  18.358 0.0001203 ***
Residuals 38 14687.4   386.5
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Reading regression output

summary(model):

```
Call:
lm(formula = height ~ sex, data = heightdata)

Residuals:
    Min      1Q  Median      3Q     Max
-38.134 -13.784   0.241  10.537  45.998

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  152.397      4.396  34.667  < 2e-16 ***
sexmale       26.638      6.217   4.285  0.00012 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 19.66 on 38 degrees of freedom
Multiple R-squared:  0.3257,    Adjusted R-squared:  0.308
F-statistic: 18.36 on 1 and 38 DF,  p-value: 0.0001203
```

```
Call:
lm(formula = lifeExp ~ continent, data = gapminder)

Residuals:
     Min       1Q   Median       3Q      Max
-31.2639  -5.4537   0.3297   6.0480  27.5767

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       48.8653     0.3696  132.22   <2e-16 ***
continentAmericas 15.7934     0.6486   24.35   <2e-16 ***
continentAsia     11.1996     0.5931   18.88   <2e-16 ***
continentEurope   23.0384     0.6110   37.70   <2e-16 ***
continentOceania  25.4609     1.9204   13.26   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.232 on 1699 degrees of freedom
Multiple R-squared:  0.4904,    Adjusted R-squared:  0.4892
F-statistic: 408.7 on 4 and 1699 DF,  p-value: < 2.2e-16
```

Same model, different analyses

- Because many functions in R use this common linear model formula, it doesn't take a lot to extend your knowledge to new analyses
- Both multilevel models and structural equation models in R use the same general syntax

- Fitting a multilevel (mixed effects regression) model, where extraversion predicts depression, controlling for clustering within schools:

  lmer(depression ~ extraversion + (1|school), data = treatdata)

How to learn new things in R

(by yourself)

Our learning process so far

- I teach a variety of new tools and (try to) explain how they work and when to use them
- You take those tools and practice with them, (hopefully) adding them to your repertoire

Extending things

But what happens when you know what you want to do, but you don't know what function(s) in R will do what you want?

The process of learning in R

1. Encounter new problem
2. Think about the logical steps you'd need to take to solve it - i.e. what needs to happen
3. Assuming you don't know how to make those things happen, you search online sources and find the answer
4. That answer becomes part of your repertoire

Where to look?

- http://stackoverflow.com/questions/tagged/r
- http://www.statmethods.net/
- https://www.r-bloggers.com/
- Or google it!

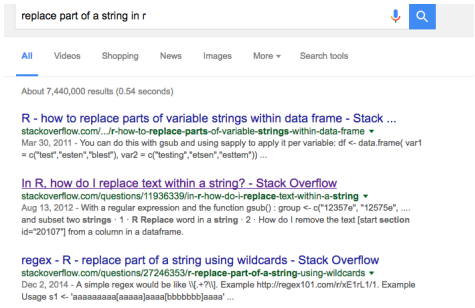
Case study

- I have some messy data where the string 'age' occurs before the numerical age in my age variable.

```
> age
[1] "age24" "age56" "age87" "age23" "age43" "age23"
```

- I need to know how to remove part of a string ('age') from a variable in R

Case study



Where to from here?

- Practice, practice, practice!
- Learn by doing - start with a project, and work your way through each step, learning as you go
- Google is your friend!