The Surgical Informatics Cookbook

Surgical Informatics, University of Edinburgh

2019-11-09

Contents


SURGICAL INFORMATI
Better surgical care through data and technology

Chapter 1

Introduction

1.1 How to contribute

(Steps 1. to 5. only need to be done once - to set-up.)

1. Connect your RStudio and GitHub using these instructions, only up to "Create new project" is necessary here (the repository/project already exists): https://www.datasurg.net/2015/07/13/rstudio-and-github/
2. Get your GitHub account added to the surgicalinformatics organisation on GitHub (ask Riinu/Ewen): https://github.com/SurgicalInformatics
3. Open the terminal on argonaut, go to your home folder (just `cd` will default to taking you home). If the prompt says `username@argonaut:~$` that means you're home - `~` means user's home.
4. In the terminal, do `git clone https://github.com/SurgicalInformatics/cookbook.git`
5. Look at the Files tab - a new folder called `cookbook` should have appeared at the bottom. Click on it and open the cookbook.Rproj file.
6. Add your thing by editing the appropriate .Rmd file - there's one for each chapter. In the `Build` pane (next to `Environment`) **click on More - Clean All** (if you don't do this you may be able to compile the book with code that won't work at a subsequent clean build which can be trickier to debug). Use the Build tab to Build your changes into a book.
7. If anyone has pushed since you cloned/last pulled (hopefully they've not been working on the exact same chapter): Make sure you **click on More - Clean All** (as above). Then Pull from the Git tab. This only cleans the output files - html and PDF, it will not touch the changes you've made in the .Rmd file.

8. Then Build Book again - this will include the new changes you pulled as well as your changes.

9. Git tab - commit everything, Push quickly before anyone else does or you'll have to go back to step 7. You can check for new pushed commits here: https://github.com/SurgicalInformatics/cookbook/commits/master Alternatively there's no harm in clicking the Pull button again - it should then say "Already up-to-date".

   Pro tip: instead of clicking on every single file in the Git tab, go to the terminal, `cd cookbook` to go to the project folder if still home, and do `git add -A` which is the same thing. Still need to Commit though!

10. Have fun!

1.2 Rules of posting

Rules of how to post here.

1.3 Indexing

1.3.1 Index

Bold index headings:
`\index{linear regression@\textbf{linear regression}}` (ticks in .Rmd file are excluded when actually using)

Sub-entries of bold headings:
`\index{linear regression@\textbf{linear regression}!diagnostics}`

Stand-alone entries:
`\index{linear regression}`

1.3.2 Chapter and section references

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter `\@ref(intro)` (ticks in .Rmd are excluded when actually using). If you do not manually label them, there will be automatic labels anyway, e.g., Chapter `\@ref(methods)`.

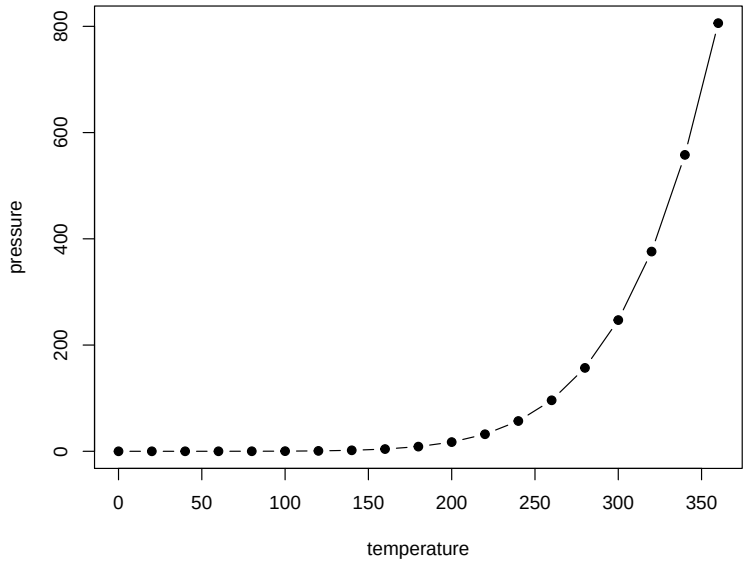


Figure 1.1: Here is a nice figure!

1.3.3 Figure and table references

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure `\@ref(fig:nice-fig)`. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table `\@ref(tab:nice-tab)`.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

1.3.4 Citations

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Table 1.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 2

Snippets

Random useful snippets that do not fit anywhere else.

2.1 Creating Reproducible R Examples to Share in the Group (binder, holepunch and docker)

When asking for help with R code having a reproducible example is crucial (some mock data that others can use along with your code to reproduce your error). Often this can be done easily with creation of a small tibble and posting of the code on slack but sometimes it requires more complex data or the error is due to something in the Linux system in which RStudio server is hosted. For example if the `cairo` package for Linux isn't installed then plots don't work. The `holepunch` package helps to reproduce examples like these (not suitable for projects with confidential data).

2.1.1 Create Basic Reproducible Examples

The three main parts of the reproducible example (reprex) in Surgical Informatics are 1. packages, 2. small dataset and 3. code. Other things like R version and Linux version can be assumed as we all use one of only a few servers.

If you have a small (and confidential) set of data in a tibble or data frame called `my_data` and want it to be easily copied run: `dput(droplevels(my_data))`. This will print out code in the console that can be copy-pasted to reproduce the data frame. Alternatively use the `tibble` or `tribble` functions to create it from scratch (this is preferable for simple datasets). Then copy in the packages

and finally the code (ideally the least amount possible to generate the error) and share with the group e.g.:

```
library(tidyverse)

# Output generated from dput(droplevels(my_data))
data = structure(list(a = c(1, 2, 3), b = c("a", "b", "c"), c = 10:12), .Names = c("a"
"b", "c"), row.names = c(NA, -3L), class = c("tbl_df", "tbl",
"data.frame"))

data %>%
  mutate(newvar = a /b)
```

```
## Error: Evaluation error: non-numeric argument to binary operator.
```

2.1.2 `holepunch` - Complex Reproducible Examples

From your project with data you are happy to make public make sure you are backed up to `git` and `GitHub`. See the relevant chapter on how to do this. Then run the following:

```
# Holepunch testing

remotes::install_github("karthik/holepunch")

library(holepunch)
write_compendium_description(package = "Title of my project",
                             description = "Rough description of project or issue")

write_dockerfile(maintainer = "SurgicalInformatics")

generate_badge()

build_binder()
```

The file will generate some text to copy into the top of a `README.md` file. It will look like:

```
<!-- badges: start -->
[![Launch Rstudio Binder](http://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/
<!-- badges: end -->
```

Now, whenever somebody clicks on the badge in the `README` on `GitHub` they will be taken to an RStudio server instance with all your files (excluding files listed in `.gitignore`), all the current versions of your package, all the current Linux packages and the current R version. They can then test your code in an near-identical environment to help identify the source of the error, their session

will time out after 10 minutes of inactivity or 12 hours since starting and will not save anything so should only be used for bug-testing or quick examples.

As this is a free version of RStudio server there is a limit to what is supported and it shouldn't be used for computationally-intensive processes.

And, as mentioned: **No confidential data.**

2.2 Working with CHIs

Here are 4 functions for CHIs that could even be put in a small package. The Community Health Index (CHI) is a population register, which is used in Scotland for health care purposes. The CHI number uniquely identifies a person on the index.

2.2.1 `chi_dob()` - Extract date of birth from CHI

Note `cutoff_2000`. As CHI has only a two digit year, need to decide whether year is 1900s or 2000s. I don't think there is a formal way of determining this.

```
library(dplyr)
chi = c("1009701234", "1811431232", "1304496368")
# These CHIs are not real.
# The first is invalid, two and three are valid.

# Cut-off any thing before that number is considered 2000s
# i.e. at cutoff_2000 = 20, "18" is considered 2018, rather than 1918.
chi_dob = function(.data, cutoff_2000 = 20){
  .data %>%
    stringr::str_extract(".{6}") %>%
    lubridate::parse_date_time2("dmy", cutoff_2000 = cutoff_2000) %>%
    lubridate::as_date() # Make Date object, rather than POSIXct
}

chi_dob(chi)
```

```
## [1] "1970-09-10" "1943-11-18" "1949-04-13"
```

```
# From tibble
tibble(chi = chi) %>%
  mutate(
    dob = chi_dob(chi)
  )
```

```
## # A tibble: 3 x 2
##   chi        dob
```

```
##   <chr>      <date>
## 1 1009701234 1970-09-10
## 2 1811431232 1943-11-18
## 3 1304496368 1949-04-13
```

2.2.2 `chi_gender()` - Extract gender from CHI

Ninth digit is odd for men and even for women. A test for even is `x modulus 2 == 0`.

```
chi_gender = function(.data){
  .data %>%
    stringr::str_sub(9, 9) %>%
    as.numeric() %>%
    {ifelse(. %% 2 == 0, "Female", "Male")}
}

chi_gender(chi)
```

```
## [1] "Male"   "Male"   "Female"
```

```
# From tibble
tibble(chi = chi) %>%
  mutate(
    dob = chi_dob(chi),
    gender = chi_gender(chi)
  )
```

```
## # A tibble: 3 x 3
##   chi        dob        gender
##   <chr>      <date>     <chr>
## 1 1009701234 1970-09-10 Male
## 2 1811431232 1943-11-18 Male
## 3 1304496368 1949-04-13 Female
```

2.2.3 `chi_age()` - Extract age from CHI

Works for a single date or a vector of dates.

```
chi_age = function(.data, ref_date, cutoff_2000 = 20){
  dob = chi_dob(.data, cutoff_2000 = cutoff_2000)
  lubridate::interval(dob, ref_date) %>%
    as.numeric("years") %>%
    floor()
}
```

```
# Today
chi_age(chi, Sys.time())
```

```
## [1] 49 75 70
```

```
# Single date
library(lubridate)
chi_age(chi, dmy("11/09/2018"))
```

```
## [1] 48 74 69
```

```
# Vector
dates = dmy("11/09/2018",
            "09/05/2015",
            "10/03/2014")
chi_age(chi, dates)
```

```
## [1] 48 71 64
```

```
# From tibble
tibble(chi = chi) %>%
  mutate(
    dob = chi_dob(chi),
    gender = chi_gender(chi),
    age = chi_age(chi, Sys.time())
  )
```

```
## # A tibble: 3 x 4
##   chi        dob        gender   age
##   <chr>      <date>     <chr>  <dbl>
## 1 1009701234 1970-09-10 Male      49
## 2 1811431232 1943-11-18 Male      75
## 3 1304496368 1949-04-13 Female    70
```

2.2.4 `chi_valid()` - Logical test for valid CHI

The final digit of the CHI can be used to test that the number is correct via the modulus 11 algorithm.

```
chi_valid = function(.data){
  .data %>%
    stringr::str_split("", simplify = TRUE) %>%
    .[, -10] %>%              # Working with matrices hence brackets
    apply(1, as.numeric) %>%  # Convert from string
    {seq(10, 2) %*% .} %>%    # Multiply and sum step
    {. %% 11} %>%             # Modulus 11
    {11 - .} %>%              # Substract from 11
```

```
    dplyr::near(              # Compare result with 10th digit.
      {stringr::str_sub(chi, 10) %>% as.numeric()}
    ) %>%
    as.vector()
}

chi_valid(chi)
```

```
## [1] FALSE  TRUE  TRUE
```

```
# From tibble
tibble(chi = chi) %>%
  mutate(
    dob = chi_dob(chi),
    gender = chi_gender(chi),
    age = chi_age(chi, Sys.time()),
    chi_valid = chi_valid(chi)
  )
```

```
## # A tibble: 3 x 5
##   chi        dob        gender   age chi_valid
##   <chr>      <date>     <chr>  <dbl> <lgl>
## 1 1009701234 1970-09-10 Male      49 FALSE
## 2 1811431232 1943-11-18 Male      75 TRUE
## 3 1304496368 1949-04-13 Female    70 TRUE
```

2.3 Working with dates

2.3.1 Difference between two dates

I always forget how to do this neatly. I often want days as a numeric, not a lubridate type object.

```
library(lubridate)
date1 = dmy("12/03/2018", "14/05/2017")
date2 = dmy("11/09/2019", "11/04/2019")

interval(date1, date2) %>%
  as.numeric("days")
```

```
## [1] 548 697
```

2.3.2 Lags

This is useful for calculating, for instance, the period off medications. Lags are much better than long to wide solutions for this.

```
library(tidyverse)
library(lubridate)
id = c(2, 2, 2, 2, 3, 5)
medication = c("aspirin", "aspirin", "aspirin", "tylenol", "lipitor", "advil")
start.date = c("05/01/2017", "05/30/2017", "07/15/2017", "05/01/2017", "05/06/2017", "05/28/2017"
stop.date = c("05/04/2017", "06/10/2017", "07/27/2017", "05/15/2017", "05/12/2017", "06/13/2017")
df = tibble(id, medication, start.date, stop.date)
df
```

```
## # A tibble: 6 x 4
##      id medication start.date stop.date
##   <dbl> <chr>      <chr>      <chr>
## 1     2 aspirin    05/01/2017 05/04/2017
## 2     2 aspirin    05/30/2017 06/10/2017
## 3     2 aspirin    07/15/2017 07/27/2017
## 4     2 tylenol    05/01/2017 05/15/2017
## 5     3 lipitor    05/06/2017 05/12/2017
## 6     5 advil      05/28/2017 06/13/2017
```

```
df %>%
  mutate_at(c("start.date", "stop.date"), lubridate::mdy) %>% # make a date
  arrange(id, medication, start.date) %>%
  group_by(id, medication) %>%
  mutate(
    start_date_diff = start.date - lag(start.date),
    medication_period = stop.date-start.date
  )
```

```
## # A tibble: 6 x 6
## # Groups:   id, medication [4]
##    id medication start.date stop.date  start_date_diff medication_period
##   <dbl> <chr>      <date>     <date>     <drtn>          <drtn>
## 1     2 aspirin    2017-05-01 2017-05-04 NA days         3 days
## 2     2 aspirin    2017-05-30 2017-06-10 29 days        11 days
## 3     2 aspirin    2017-07-15 2017-07-27 46 days        12 days
## 4     2 tylenol    2017-05-01 2017-05-15 NA days        14 days
## 5     3 lipitor    2017-05-06 2017-05-12 NA days         6 days
## 6     5 advil      2017-05-28 2017-06-13 NA days        16 days
```

2.3.3 Pulling out "change in status" data

If you have a number of episodes per patient, each with a status and a time, then you need to do this as a starting point for CPH analysis.

2.3.3.1 Example data

```
library(dplyr)
library(lubridate)
library(finalfit)
mydata = tibble(
  id = c(1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5),
  status = c(0,0,0,1,0,0,1,1,0,0,0,0,0,1,1,1,0,0,1,1),
  group = c(rep(0, 8), rep(1, 12)) %>% factor(),
  opdate = rep("2010/01/01", 20) %>% ymd(),
  status_date = c(
    "2010/02/01", "2010/03/01", "2010/04/01", "2010/05/01",
    "2010/02/02", "2010/03/02", "2010/04/02", "2010/05/02",
    "2010/02/03", "2010/03/03", "2010/04/03", "2010/05/03",
    "2010/02/04", "2010/03/04", "2010/04/04", "2010/05/04",
    "2010/02/05", "2010/03/05", "2010/04/05", "2010/05/05"
  ) %>% ymd()
)
mydata
```

```
## # A tibble: 20 x 5
##       id status group opdate     status_date
##    <dbl>  <dbl> <fct> <date>     <date>
##  1     1      0 0     2010-01-01 2010-02-01
##  2     1      0 0     2010-01-01 2010-03-01
##  3     1      0 0     2010-01-01 2010-04-01
##  4     1      1 0     2010-01-01 2010-05-01
##  5     2      0 0     2010-01-01 2010-02-02
##  6     2      0 0     2010-01-01 2010-03-02
##  7     2      1 0     2010-01-01 2010-04-02
##  8     2      1 0     2010-01-01 2010-05-02
##  9     3      0 1     2010-01-01 2010-02-03
## 10     3      0 1     2010-01-01 2010-03-03
## 11     3      0 1     2010-01-01 2010-04-03
## 12     3      0 1     2010-01-01 2010-05-03
## 13     4      0 1     2010-01-01 2010-02-04
## 14     4      1 1     2010-01-01 2010-03-04
## 15     4      1 1     2010-01-01 2010-04-04
## 16     4      1 1     2010-01-01 2010-05-04
## 17     5      0 1     2010-01-01 2010-02-05
```

```
## 18     5      0 1     2010-01-01 2010-03-05
## 19     5      1 1     2010-01-01 2010-04-05
## 20     5      1 1     2010-01-01 2010-05-05
```

2.3.3.2 Compute time from op date to current review

... if necessary

```
mydata = mydata %>%
  arrange(id, status_date) %>%
  mutate(
    time = interval(opdate, status_date) %>% as.numeric("days")
  )
mydata
```

```
## # A tibble: 20 x 6
##       id status group opdate     status_date  time
##    <dbl>  <dbl> <fct> <date>     <date>      <dbl>
##  1     1      0 0     2010-01-01 2010-02-01     31
##  2     1      0 0     2010-01-01 2010-03-01     59
##  3     1      0 0     2010-01-01 2010-04-01     90
##  4     1      1 0     2010-01-01 2010-05-01    120
##  5     2      0 0     2010-01-01 2010-02-02     32
##  6     2      0 0     2010-01-01 2010-03-02     60
##  7     2      1 0     2010-01-01 2010-04-02     91
##  8     2      1 0     2010-01-01 2010-05-02    121
##  9     3      0 1     2010-01-01 2010-02-03     33
## 10     3      0 1     2010-01-01 2010-03-03     61
## 11     3      0 1     2010-01-01 2010-04-03     92
## 12     3      0 1     2010-01-01 2010-05-03    122
## 13     4      0 1     2010-01-01 2010-02-04     34
## 14     4      1 1     2010-01-01 2010-03-04     62
## 15     4      1 1     2010-01-01 2010-04-04     93
## 16     4      1 1     2010-01-01 2010-05-04    123
## 17     5      0 1     2010-01-01 2010-02-05     35
## 18     5      0 1     2010-01-01 2010-03-05     63
## 19     5      1 1     2010-01-01 2010-04-05     94
## 20     5      1 1     2010-01-01 2010-05-05    124
```

2.3.3.3 Pull out “change of status”

```
mydata = mydata %>%
  group_by(id) %>%
  mutate(
```

```
    status_change = status - lag(status) == 1,                          # Mark TRUE if
    status_nochange = sum(status) == 0,                                 # Mark if no c
    status_nochange_keep = !duplicated(status_nochange, fromLast= TRUE) # Mark most re
  ) %>%
  filter(status_change | (status_nochange & status_nochange_keep)) %>%  # Filter out o
  select(-c(status_change, status_nochange, status_nochange_keep))      # Remove colum
mydata
```

```
## # A tibble: 5 x 6
## # Groups:   id [5]
##      id status group opdate     status_date  time
##   <dbl>  <dbl> <fct> <date>     <date>      <dbl>
## 1     1      1 0     2010-01-01 2010-05-01    120
## 2     2      1 0     2010-01-01 2010-04-02     91
## 3     3      0 1     2010-01-01 2010-05-03    122
## 4     4      1 1     2010-01-01 2010-03-04     62
## 5     5      1 1     2010-01-01 2010-04-05     94
```

2.3.3.4 Run CPH

```
mydata %>%
  finalfit("Surv(time, status)", "group")
```

```
##  Dependent: Surv(time, status)       all        HR (univariable)
## 1                       group 0 2 (100.0)                        -
## 2                           1 3 (100.0) 0.76 (0.10-5.51, p=0.786)
##          HR (multivariable)
## 1                          -
## 2 0.76 (0.10-5.51, p=0.786)
```

Chapter 3

Data manipulation

3.1 Collapse multiple "no" and "yes" options

Common to have to do this in globalsurg projects

```
library(dplyr)
mydata = tibble(
  ssi.factor = c("No", "Yes, no treatment/wound opened only (CD 1)",
                 "Yes, antibiotics only (CD 2)", "Yes, return to operating theatre (CD 3)",
                 "Yes, requiring critical care admission (CD 4)",
                 "Yes, resulting in death (CD 5)",
                 "Unknown") %>%
    factor(),
  mri.factor = c("No, not available", "No, not indicated",
                 "No, indicated and facilities available, but patient not able to pay",
                 "Yes", "Unknown", "Unknown", "Unknown") %>%
    factor()
)

# Two functions make this work
fct_collapse_yn = function(.f){
  .f %>%
    forcats::fct_relabel(~ gsub("^No.*", "No", .)) %>%
    forcats::fct_relabel(~ gsub("^Yes.*", "Yes", .))
}

is.yn = function(.data){
  .f = is.factor(.data)
  .yn = .data %>%
    levels() %>%
```

```
    grepl("No|Yes", .) %>%
    any()
  all(.f, .yn)
}

# Raw variable
mydata %>%
  pull(ssi.factor) %>%
  levels()
```

```
## [1] "No"
## [2] "Unknown"
## [3] "Yes, antibiotics only (CD 2)"
## [4] "Yes, no treatment/wound opened only (CD 1)"
## [5] "Yes, requiring critical care admission (CD 4)"
## [6] "Yes, resulting in death (CD 5)"
## [7] "Yes, return to operating theatre (CD 3)"
```

```
# Collapse to _yn version
mydata %>%
  mutate_if(is.yn, list(yn = fct_collapse_yn)) %>%
  pull(ssi.factor_yn) %>%
  levels()
```

```
## [1] "No"      "Unknown" "Yes"
```

Chapter 4

Machine learning

4.1 Deep learning

4.1.1 Pulling images from REDCap directly to argodeep

4.1.1.1 Original file names

```
library(REDCapR)
uri = "https://redcap.cir.ed.ac.uk/api/"
token = "" # API token here
record_list = 1:318
field_list = c("photo", "photo_2", "photo_3", "photo_4")
event_list = c("wound_concerns_arm_2", "questionnaire_1_arm_2",
               "questionnaire_2_arm_2", "questionnaire_3_arm_2")
directory = "wound_raw" # destination directory must exist already


for(record in record_list){
  for(field in field_list){
    for(event in event_list){
      result =
        tryCatch({      # suppress breaking error when no image in slot
          redcap_download_file_oneshot(
            record        = record,
            field         = field,
            redcap_uri    = uri,
            token         = token,
            event         = event,
            overwrite     = TRUE,
```

```
            directory     = directory
          )
        }, error=function(e){})
    }
  }
}
```

4.1.1.2 Named from REDCap record ID and event

```
library(REDCapR)
uri = "https://redcap.cir.ed.ac.uk/api/"
token = "" # API token here
record_list = 1:318
field_list = c("photo", "photo_2", "photo_3", "photo_4")
event_list = c("wound_concerns_arm_2", "questionnaire_1_arm_2",
               "questionnaire_2_arm_2", "questionnaire_3_arm_2")
directory = "wound_named" # destination directory must exist already

for(record in record_list){
  for(field in field_list){
    for(event in event_list){
      file_name = paste0(record, "_", field, "_", event, ".jpg")
      result =
        tryCatch({
          redcap_download_file_oneshot(
            record        = record,
            field         = field,
            redcap_uri    = uri,
            token         = token,
            event         = event,
            overwrite     = TRUE,
            directory     = directory,
            file_name     = file_name
          )
        }, error=function(e){})
    }
  }
}
```

Chapter 5

Data Transfer and Eddie

Often it is necessary to transfer data into and out of RStudio server. This can be done from personal laptops (as long as you have permission for the data!), university supported desktops, Eddie and a number of other devices or servers.

5.1 Uploading and Downloading Data the Easy Way

Often the GUI in RStudio is sufficient. In the `Files` pane click upload to move data from your current computer into RStudio server. To download select the file and then click `More > Export`. As always this should be done only when appropriate.

Sometimes this isn't possible either with large files or files stored remotely on other servers such as Eddie.

5.2 Alternative Methods when the Easy Way won't work

5.2.1 What is Eddie?

Eddie Mark 3 is the third iteration of the University's compute cluster and is available to all University of Edinburgh researchers. It consists of some 7000 Intel® Xeon® cores with up to 3 TB of memory available per compute node.

The surgical informatics group has a shared folder in the Eddie cluster which can be accessed to store data and perform analyses which might be too large

or complex for RStudio server. Every task in Eddie is controlled through the command line interface - those familiar with Linux/Unix will be familiar with this.

5.3 Using the Command Line

Sometimes the command line is the only possible way to copy data to and from RStudio server. Argonaut and Argosafe are `SSH`-enabled (Secure SHell) meaning that you can securely copy data to and from them using the command line editor on another device. It is also possible to use the RStudio terminal to copy to another device or server that is `SSH`-enabled although this isn't currently recommended due to issues with RStudio server's websockets (when you type in the terminal you have to do so very slowly for characters to appear in the right order or you have to use a script - described below).

First, if you don't have a command line editor or SSH client installed (often the case for earlier Windows versions although there is talk of a native client becoming default) then you will need to install one. For working on a Windows device generally PuTTY is the recommended SSH client (allows you connect to other servers) and a reasonable command line editor to work with files on the local device is GitBash.

5.3.1 Downloads and Setup - Eddie Example

1. Make sure you have the University VPN downloaded for your own computer to access Eddie if needed. The link is at: https://www.ed.ac.uk/information-services/computing/desktop-personal/vpn/vpn-service-using. It is the Cisco Connect Any Client which logs into the VPN (the password should be different to your EASE password).

2. Download the PuTTY terminal software: https://www.putty.org/.

Open up PuTTY and you will see a configuration screen. On this screen make sure to enter `eddie3.ecdf.ed.ac.uk` into the box and tick SSH as your method for connection. The PuTTY terminal will launch (assuming you are connected to the University VPN already) and ask `login as` at which point you should enter your EASE username. You will then be asked for your EASE password and you should now see that you are logged into Eddie3.

If you are logging in from RStudio Server or another terminal software you should enter on the command line:

```
ssh <UUN>@eddie3.ecdf.ed.ac.uk
```

is your university username for EASE. You will then be asked for your password.

More information on Eddie basics can also be found at: https://www.wiki.ed.ac.uk/display/ResearchServices/Quickstart.

5.3.2 Copy and Paste with PuTTY

Like many other command line interfaces PuTTY can be made to work more easily with copy and paste. This is done simply through highlighting and clicking and not with traditional `ctrl-c` and `ctrl-v` commands like typical word processors.

To copy text from PuTTY: Highlight the text. That's it! No need to click anything or type / press anything, highlighting is enough. You can then paste the text elsewhere.

To copy text into PuTTY: Once you've copied text (either by highlighting in PuTTY itself or by using `ctrl-c` in another programme, just right-click. The text will appear at the command line. If you copy several lines separated by `\newline` then PuTTY will run each line up until the last one copied and leave the last line at the command line (if you highlight large sections of text in PuTTY and right-click it will try to run all of them).

5.3.3 Closing a Session in PuTTY

To close a session use `ctrl-d`.

5.3.4 Eddie File Structure

Once you have logged into Eddie there are several directories (folders/places) where you can store and manipulate files. Moving between these directories is usually done using the `cd` command. The same idea is applicable to your own machine

When you first log in you will default to your home directory. In order to see the "path" to that directory enter the command:

```
pwd
```

The terminal should print out something like:

```
/home/<UUN>
```

To get back to this directory at any point enter one of the following (they are both equivalent):

```
cd /home/<UUN>
```

or

```
cd ~
```

When inside your home folder to see any of the files or subdirectories in your home directory enter:

```
ls -a
```

The `-a` argument to ls shows hidden files which begin with a `.` such as `.Renviron` if you have created this. There are several other arguments which can be passed to ls such as `-l` which will should the permissions for the file or subdirectory. When using the `-l` argument the files start with `-` and are followed by 7 characters or `-` which explain whether different groups of people can write, read or execute the files. The first three are for the file owner, the next three for the group and the next three for any else with access to the directory. For example the following printed after running `ls -l` would indicate that the owner could read, write and execute a file, the group could read and execute it and that others could only read it:

```
-rwxr-xr-- ... ... ... ... my_file.txt
```

5.4 Copying Data from Eddie into RStudio server

5.4.1 Method 1: Using PuTTY (or another terminal/shell connected to Eddie)

To copy data from another server or device into RStudio Server use the following code in the command line editor (e.g. PuTTY, GitBash etc.):

```
scp <file_path_on_other_device_or_server>new_file.txt <RStudio_Server_Username>@argonau
```

You will be asked for your RStudio server (Argonaut etc.) password. If you are unsure of the file path enter `pwd` when you are working in the directory with the file before copying and pasting.

If you are not sure of the path to the directory in which you wish to copy the data to or from then use `getwd()` in RStudio when inside the project you wish to copy to (you many need to add a subdirectory folder to the `getwd()` output if you are using subdirectories in your RStudio projects).

The `scp` command will work with Argonaut and Argosafe as the ability to allow SSH connections has been activated. This may need to be established separately for other RStudio servers within the department.

5.4.2 Method 2: Using the RStudio Server Terminal

To use the RStudio server terminal to copy data in and out of Eddie do not SSH into Eddie as described above but instead use the `scp` command. If you are using RStudio's terminal for accessing Eddie instead of PuTTY or something equivalent then you will need to either temporarily disconnect from Eddie (`ctrl-d`) or open a new terminal if using RStudio server Pro which allows multiple terminals.

In the terminal enter the following command to copy a single file from Eddie into RStudio server from a project subdirectory in the SurgInfGrp directory:

```
scp <UUN>@eddie3.ecdf.ed.ac.uk:/exports/cmvm/eddie/edmed/groups/SurgInfGrp/<project_subdirectory>

# Or adapt to copy entire directory contents (drop the * to copy the directory/folder as well as
scp -r <UUN>@eddie3.ecdf.ed.ac.uk:/exports/cmvm/eddie/edmed/groups/SurgInfGrp/<project_subdirecto
```

On entering the command to the terminal you will be asked to enter your EASE password (type this slowly if using RStudio Server Pro prior to web sockets issue being fixed). In order to move data in the other direction simply change the order of the file paths.

5.5 The Surgical Informatics Group Folder (On Eddie)

To get to the Surgical Informatics shared group directory enter:

```
cd /exports/cmvm/eddie/edmed/groups/SurgInfGrp
```

If you don't have access then Riinu or Ewen can provide this as they have admin rights.

This directory should be the place to store any group projects or apps or other files which are somewhat (but not very) large. The group space has 200GB of memory allocated by default which is much greater than your home directory (10GB) but much less than the group space on the datastore which is up to 1TB. The group space should be used to store bulk files which can be staged into Eddie for an active session but will not be permanently stored in Eddie.

5.5.1 The Scratch Space

Your own personal scratch space is where you should work on active projects during a session after staging them in from the datastore and/or shared Surgical Informatics group directory. The scratch space has 2TB of storage per user but this is cleaned up after one month. This means large datasets can be analysed

here but not stored in the long-term. To find your own personal scratch space enter:

```
cd /exports/eddie/scratch/<UUN>
```

Finally there is also a temporary directory ($TMPDIR) which is only present and accessible whilst a job is running in Eddie. This has 1TB of available storage.

5.5.2 Other Spaces

Occasionally group members may have access to other directories to share / collaborate on projects. These are most often found at the path: `cd /exports/<COLLEGE>/eddie/<SCHOOL>/groups/<GROUP>` although for IGMM it may be: `cd /exports/igmm/eddie/<GROUP>`.

5.5.3 Running Eddie from RStudio Server

It is possible to log in to Eddie from RStudio server and use Shell Scripts stored in RStudio to perform tasks in Eddie. This may be helpful if you plan to modify shell scripts a lot and want to have the benefit of the RStudio interface. `ctrl-alt-enter` sends data to the terminal in the same way that it would the console without the `alt`.

To connect initially enter:

```
ssh <UUN>@eddie3.ecdf.ed.ac.uk
```

You will then be asked for your password which has to be typed (currently slowly and carefully!) into the terminal. Afterwards you can send any commands from a script using `ctrl-alt-enter`. This will not work with `Rmd` or `notebooks`.

5.5.4 Modules (Applications) in Eddie

Eddie has several modules (applications) which can be run such as R, python, cuda, java, intel, fastqc etc. etc. The best way to see which modules you have available is to run `module avail`. To see which modules are loaded is to run `module list`.

To load in a new module to use run the following:

```
module load <module>

# For example:
module load R/3.5.3
```

```
# Note that the default R version is 3.3.2 (as of 13th June 2019) and is loaded using:
module load R
```

Once you have loaded modules that you will need for your analysis you may need to create new files or establish a library of packages for those modules. There are a default set of packages available for R when loaded from either the main applications library or from other installations on the IGMM paths but these are read-only meaning that for any customisation and installation of new packages, a separate library must be maintained and that the R options must be amended to point to this library. This can be done by creating a personal `.Rprofile` file in your home directory in Eddie which points to the Surgical Informatics Group Rlibrary.

It is not advised to create a separate library in your own Eddie space as this will quickly use up your disc quota. Theoretically a separate installation of a package could be stored in your own space if working on developer edition or github branch / fork of a package.

5.5.5 Working with R in Eddie

5.5.5.1 .Rprofile File

When you load up R an `.Rprofile` file is sourced and anything contained within the file will be run for the current R session. This is particularly useful in Eddie because the defaults are not very helpful:

- The R package library is the Eddie default library which cannot be added to
- The CRAN mirror is not specified so every session will ask you to select a new CRAN mirror to install packages in a temporary library

The `.Rprofile` file also has other options which can be customised to improve how your current R session will run on Eddie. There are a number of possible customisations but be careful as not all are helpful if you end up collaborating with other research teams, some of these customisations such as `stringsAsFactors=FALSE` by default could be used in a project-specific `.Rprofile` file with caution but if you are working with another research group who rely on `read.csv` and have scripts established which assume that all strings are factors then having that customisation in your home directory may generate bugs when collaborating on projects.

R will automatically look for a `.Rprofile` file when R is started during each Eddie session and will look for this file in three places with an order of preference. The first place is the current working directory of a project so an `.Rprofile` file could be stored here to generate very specific customisations for a project if necessary. The next place it will look is in your own home directory (`/home/<UUN>/`

or `/~/`), this is where you should create a `.Rprofile` file which will be your default for all sessions, it is recommended to use this sparingly but that certain key features are used such as setting up your main library location as the Surgical Informatics Group and setting up a default CRAN mirror e.g. the RStudio CRAN mirror. Finally, if there is not a `.Rprofile` file here then R will attempt to source a file in the R home directory (found in R using `.R.home()`) and if there is no file here then no customisation will occur. Some servers also have a `.Rprofile.site` file although this is not currently present in Eddie.

Important: Always leave the final line of an `.Rprofile` file blank.

A possible `.Rprofile` file for using in Eddie as part of the Surgical Informatics Group is:

```
## Rprofile template


## Stop being asked for CRAN mirror every time
options("repos" = c(CRAN = "http://cran.rstudio.com/"))


## Change the default editor to nano
options(editor="nano")

## Change the terminal prompt to make it clear R is loaded
options(prompt="R > ")

## Prevent default saving of workspace image (similar to recommended RStudio server se
q <- function(save="no", ...) {
        quit(save=save, ...)
}


## Clever code to allow tab-completion of package names used in library()
utils::rc.settings(ipck=TRUE)


## Add some colour to the console if colourout is available
if(Sys.Getenv("TERM") == "xterm-256color")
        library("colorout")

## Create a new envisible environment which can be used to create new functions
## Benefit of this is that all new functions here are hidden in environment and not rm
.env <- new.env()


## Set library path to make sure packages are loaded from SurgInfGrp
```

```
.libPaths("/exports/cmvm/eddie/edmed/groups/SurgInfGrp/R/Rlibrary")


## If working on a particular project e.g. the gwas_pipeline from IGMM best to create a new libra
## Just copy the .Rprofile file from your own home directory into the project working directory a


## Attach all the variables created
attach(.env)


## Remember that an .Rprofile file always silently ignores the last line so don't forget to leave
```

The above configuration should generate minimal portability issues when working on other projects.

There are a few explanations of the benefits and side effects of having a `.Rprofile` file as well as ways to temporarily mask them for specific projects which will be shared with other collaborators in this blog post: https://www.r-bloggers.com/fun-with-Rprofile-and-customizing-r-startup/.

Should you wish to have an entire directory of startup files (e.g. one file for CRAN, one file for library, one file for custom function etc. etc.) so that segments of the customisation can be shared quickly without modifying / dropping all of the other elements of the `.Rprofile` file then this is described here: https://cran.r-project.org/web/packages/startup/vignettes/startup-intro.html. This relies on the `startup` package. Other options such as having secure directories with GitHub tokens and other content which is protected from access by other Eddie users is also discussed there.

5.5.5.2 Creating an .Rprofile File

When you first use Eddie there will not be any `.Rprofile` file generated by default and a blank file will need to be created. This can be done by entering the following into the terminal after loading R. To load R enter `module load R/3.5.3` (or the currently available R versions seen on `module avail`) into the terminal followed by `R`. This will start an active R session. Then enter the following code (please do not change the path to the shared group folder! If you need to change the path to a project be sure to include the subdirectory otherwise R will use your `.Rprofile` file for all SurgInfGrp users which won't be popular):

```
file.create("~/.Rprofile")
```

This will create a blank `.Rprofile` file in your home directory on Eddie which should be edited to customise the R configuration.

In order to edit the file it is suggested that you use the nano Unix-based editor. Firstly quit the current R session as that has not yet been configured to use the editor. Enter `q()` as you would normally do when closing R and enter `n` if asked about saving workspace image.

Navigagte to your home directory and using `ls -a` you should see the `.Rprofile` file which is currently blank.

5.5.5.3 .Renviron File

In addition to changing the `.Rprofile` file you will most likely need to change the `.Renviron` file. The default for this is probably located in the R home directory which may be `/exports/applications/apps/SL7/R/3.5.3/lib64/R/etc/`. The `.Renviron` file is searched for in the same order by R as the `.Rprofile` file with R first aiming to find the file in the project directory and if not able to find it there looking in the user home directory and finally looking to the R home directory. To copy the `Renviron` file in the R home directory into a `.Renviron` file in your home directory enter the following code in the terminal:

```
# Path will need edited if future installations of R replace 3.5.3
cp /exports/applications/apps/SL7/R/3.5.3/lib64/R/etc/Renviron ~/.Renviron
```

The following is a reasonable starting point for the `.Renviron` file in your own profile:

```
### etc/Renviron.  Generated from Renviron.in by configure.
###
### ${R_HOME}/etc/Renviron
###
### Record R system environment variables.

R_PLATFORM=${R_PLATFORM-'x86_64-pc-linux-gnu'}
## Default printer paper size: first record if user set R_PAPERSIZE
R_PAPERSIZE_USER=${R_PAPERSIZE}
R_PAPERSIZE=${R_PAPERSIZE-'a4'}
## Default print command
R_PRINTCMD=${R_PRINTCMD-''}
# for Rd2pdf, reference manual
R_RD4PDF=${R_RD4PDF-'times,hyper'}
## used for options("texi2dvi")
R_TEXI2DVICMD=${R_TEXI2DVICMD-${TEXI2DVI-'texi2dvi'}}
## used by untar and installing grDevices
R_GZIPCMD=${R_GZIPCMD-'/usr/bin/gzip'}
## Default zip/unzip commands
R_UNZIPCMD=${R_UNZIPCMD-'/usr/bin/unzip'}
R_ZIPCMD=${R_ZIPCMD-'/usr/bin/zip'}
R_BZIPCMD=${R_BZIPCMD-'/usr/bin/bzip2'}
```

```
## Default browser
R_BROWSER=${R_BROWSER-'/usr/bin/xdg-open'}
## Default pager
PAGER=${PAGER-'/usr/bin/less'}
## Default PDF viewer
R_PDFVIEWER=${R_PDFVIEWER-'/usr/bin/xdg-open'}
## Used by libtool
LN_S='ln -s'
MAKE=${MAKE-'make'}
## Prefer a POSIX-compliant sed on e.g. Solaris
SED=${SED-'/usr/bin/sed'}
## Prefer a tar that can automagically read compressed archives
TAR=${TAR-'/usr/bin/gtar'}

## System and compiler types.
R_SYSTEM_ABI='linux,gcc,gxx,gfortran,?'


## Change the default R libraries and directories
R_DOC_DIR=/exports/applications/apps/SL7/R/3.5.3/lib64/R/doc
R_INCLUDE_DIR=/exports/applications/apps/SL7/R/3.5.3/lib64/R/include
R_SHARE_DIR=/exports/applications/apps/SL7/R/3.5.3/lib64/R/share
RBIN=/exports/applications/apps/SL7/R/3.5.3/bin
RDIR=/exports/applications/apps/SL7/R/3.5.3/3.3.3
R_LIBS=/exports/cmvm/eddie/edmed/groups/SurgInfGrp/R/Rlibrary
R_LIBS_SITE=/exports/cmvm/eddie/edmed/groups/SurgInfGrp/R/Rlibrary
R_LIBS_USER=/exports/cmvm/eddie/edmed/groups/SurgInfGrp/R/Rlibrary




### Local Variables: ***
### mode: sh ***
### sh-indentation: 2 ***
```

The `.Renviron` file can be copied to specific projects when specific R files / directories (or files / directories for other languages) are needed to run a particular analysis. For example, to use the GCC compiler from IGMM the following could be added to the file:

```
## Change the Clang variables
CC=igmm/compilers/gcc/5.5.0 -fopenmp
CXX=igmm/compilers/gcc/5.5.0 -fopenmp
```

The `.Rprofile` files and `.Renviron` files should be as generic as possible in your own profile so as to avoid issues when sharing scripts. They can be tai-

lored for specific projects when needed in the local working directory. Typically this should be to alter the paths to directories etc. and not to create bespoke functions which should be done in scripts.

5.5.6 Specific R Package Versions

When working with other teams it is often necessary to work with particular R package versions. One effective way to ensure you do this is using the remotes package:

```
# Install the GenABEL package from CRAN but as no longer supporte need the 1.8-0 versi
remotes::install_version("GenABEL", version = "1.8-0")
```

5.5.7 General Eddie Etiquette

Don't overwrite or change other users files without permission and set up your own files to prevent others doing that if appropriate. Please don't leave configuration files like `.Renviron` files in the shared space as this can play havoc with other users R tasks in Eddie - if you need a particular configuration (e.g. a default package library for a project) then best to leave the config files in the directory for that project (or in your own home directory).

5.6 Unix Tutorials

The University Digital Skills Framework offers a three-part tutorial covering Unix which is free. If you use a lot of Eddie etc. this may be helpful:https://www.digitalskills.ed.ac.uk/all-resources/?wdt_column_filter%5B1%5D=Classroom

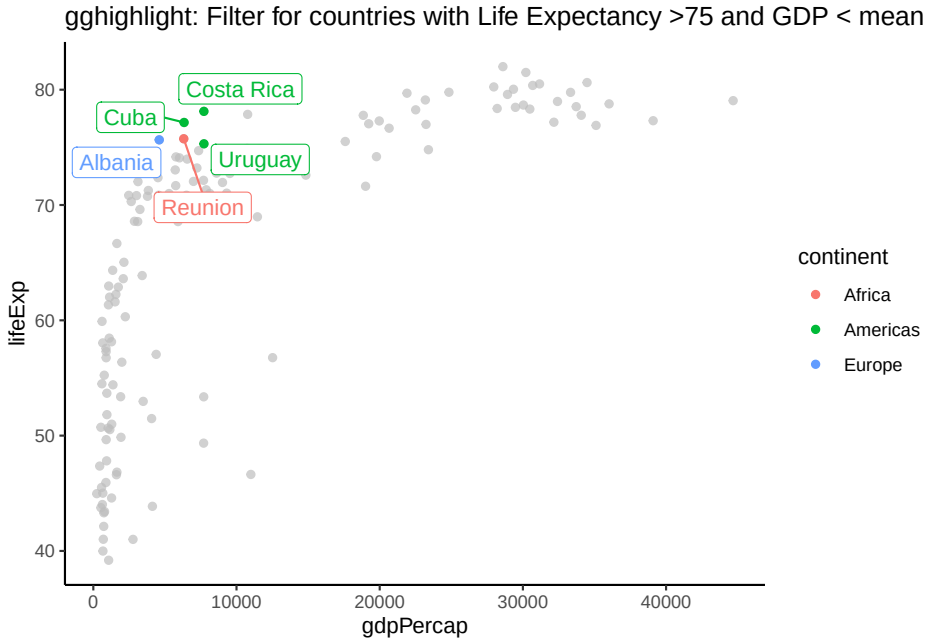
Chapter 6

Plotting

6.1 GGHighlight Example

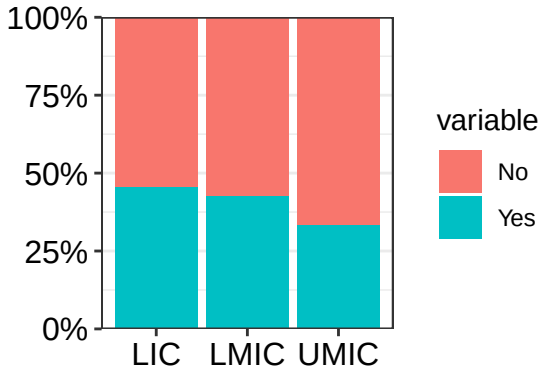
Plotting with gghighlight is pretty awesome allowing you to filter on any variable. It seems that gghighlight overwrites any 'colour' variable you put in the main aes. To get round this and have labels, save as a plot and add geom_label_repel separately.

```
library(gghighlight)
library(ggrepel)

mydata=gapminder


plot = mydata %>%
  filter(year == "2002") %>%
  ggplot(aes(x = gdpPercap, y = lifeExp, colour=continent)) +
  geom_point()+
  gghighlight(lifeExp > 75 & gdpPercap < mean(gdpPercap), label_key = country, use_direct_label =
  theme_classic()+
  labs(title= "gghighlight: Filter for countries with Life Expectancy >75 and GDP < mean" )

plot + geom_label_repel(aes(label= country), show.legend = FALSE) #only needed if you use  use_di
```



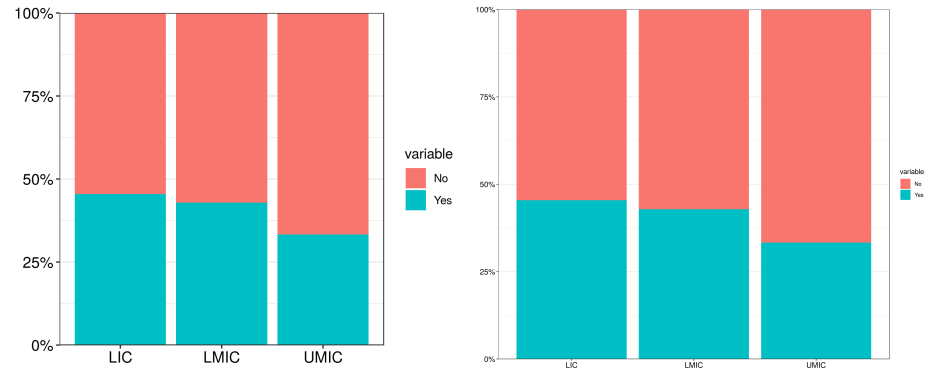
6.2 Axis font size

```
# OPTION 1: theme(axis.text = element_text(size = 12, colour = "black"))
# OPTION 2: width and height arguments of ggsave()

library(tidyverse)
library(scales)

# made-up example data
mydata = tibble(group    = c("UMIC", "LMIC", "LIC") %>% rep(each = 2),
                value    = 1:6,
                variable = c("Yes", "No") %>% rep(3))

mydata %>%
  ggplot(aes(x = group, y = value, fill = variable)) +
  geom_col(position = "fill") +
  scale_y_continuous(labels = percent, expand = c(0, 0)) +
  theme_bw() +
  # OPTION 1: change font with theme()
  theme(axis.text  = element_text(size = 12, colour = "black"),
        axis.title = element_blank())
```




```
# OPTION 2: play around with export size. Since PDF will always have max resolution anyway
# but changing width and height modifies text size
mywidth  = 5
myheight = 4
#ggsave("barplot_5x4.pdf", width = mywidth, height = myheight)

mywidth  = 10
myheight = 8
#ggsave("barplot_10x8.pdf", width = mywidth, height = myheight)
```

Same plot 5x4 inches vs 10x8 inches:



Chapter 7

Genomics

7.1 Single Cell Analysis

7.1.1 Minimising the size of a Seurat Object

Single cell analyses are ofetn collated in Seurat objects which can be huge. We reduced the size of our Seurat object by pulling the relevant sections using the code below reducing the object to <20% of its original size. This works by only including the sparse matrices. This is not a workable example, but the PBMC dataset could be used if necessary.

```
#library(Seurat)
#library(dplyr)


#Load in the data

pathtoglobaldata <-"mac_shiny/data/Global_sc_object"
object_global<-readRDS(pathtoglobaldata)


# Get sparse assay data
sizetest_global<-GetAssayData(object = object_global, slot = "data") # counts don't work, scaled
object_global_small<- CreateSeuratObject(sizetest_global)

# Add in classifications.

object_global_small@reductions$tsne<-object_global@reductions$tsne # reduction embeddings for ts
object_global_small$Phenotype<- object_global$Phenotype
```

```
object_global_small@meta.data$final_classification<- object_global@meta.data$final_clas
object_global_small$final_classificaiton<- object_global$final_classification #cell cl
Idents(object = object_global_small) <- "final_classificaiton" #set forever


#same
saveRDS(object_global_small,"mac_shiny/data/global_object_sml")
```

Chapter 8

Programming in rlang

8.1 rlang

8.1.1 What is rlang?

rlang is a low-level programming API for R which the tidyverse uses (meaning it speaks to R in as R like way as possible, rather than a 'high-level' - high level is more user orientated and interpretable). It enables you to extend what the tidyverse can do and adapt it for your own uses. It's particularly good to use if you're doing lots of more 'programming' type R work, for example, building a package, making a complex shiny app or writing functions. It might also be handy if you're doing lots of big data manipulation and want to manipulate different datasets in the same way, for example.

In this chapter we'll discuss some uses of it.

8.1.2 Dynamic calling of variables using dplyr

In this example, say we have a tibble of variables, but we want to apply dynamic changes to it (so we feed R a variable, that can change, either using another function like `purr::map` or in a ShinyApp). In this instance, specifying each variable and each different possible consequence using different logical functions would take forever and be very clunky. So we can use rlang to simply put a dynamic variable/object through the same function.

We'll use an example where we want to summarise by different outcomes in a dynamic way.

```
library(tidyverse)
#Make data - here nonsense numbers on deaths per 100k from guns say
```

```
example_data = tibble(countries = c('UK', 'USA', 'Pakistan', 'Mexico', 'Ireland', 'Esto
                      region = c('Europe', 'Americas', 'Asia', 'Americas', 'Europe', 'E
                      Death_from_guns = c(1, 200, 150, 450, 3, 3.5),
                      Death_from_smoking = c(100, 300, 140, 150, 120, 300))


#example function for summarising using dynamic variables and bang bangs
#note metric must be numeric
summarise_feature = function(df, col_var, ...){
  require(tidyverse)

  wurly_curly = function(.){#The wurly_curly function makes things nicer
  require(rlang)
  !!quo_name(enquo(.))}

  summary_nm_sum <- paste0("metric_", wurly_curly(col_var))#The new LHS variable must b

  df %>%
    group_by(...) %>%
    summarise(!!summary_nm_sum := sum({{col_var}}))
}


#output new variable
example_data %>%
  summarise_feature(Death_from_guns, region)

#How to dynamically change names using mutate and rlang
#Bang bang variables into mutate/tidyverse(so they can be dynamically changed)

#Select metric
metric = 'metric_Death_from_guns' #example but can be any named column

metric_mutate = paste0('prefix_', metric) #some reason doesn't take these within the m

example_data %>%
  summarise_feature(Death_from_guns, region) %>%
  mutate(!!metric_mutate := !!rlang::sym(metric) * 2)#apply arbitary multiplication by
```

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.