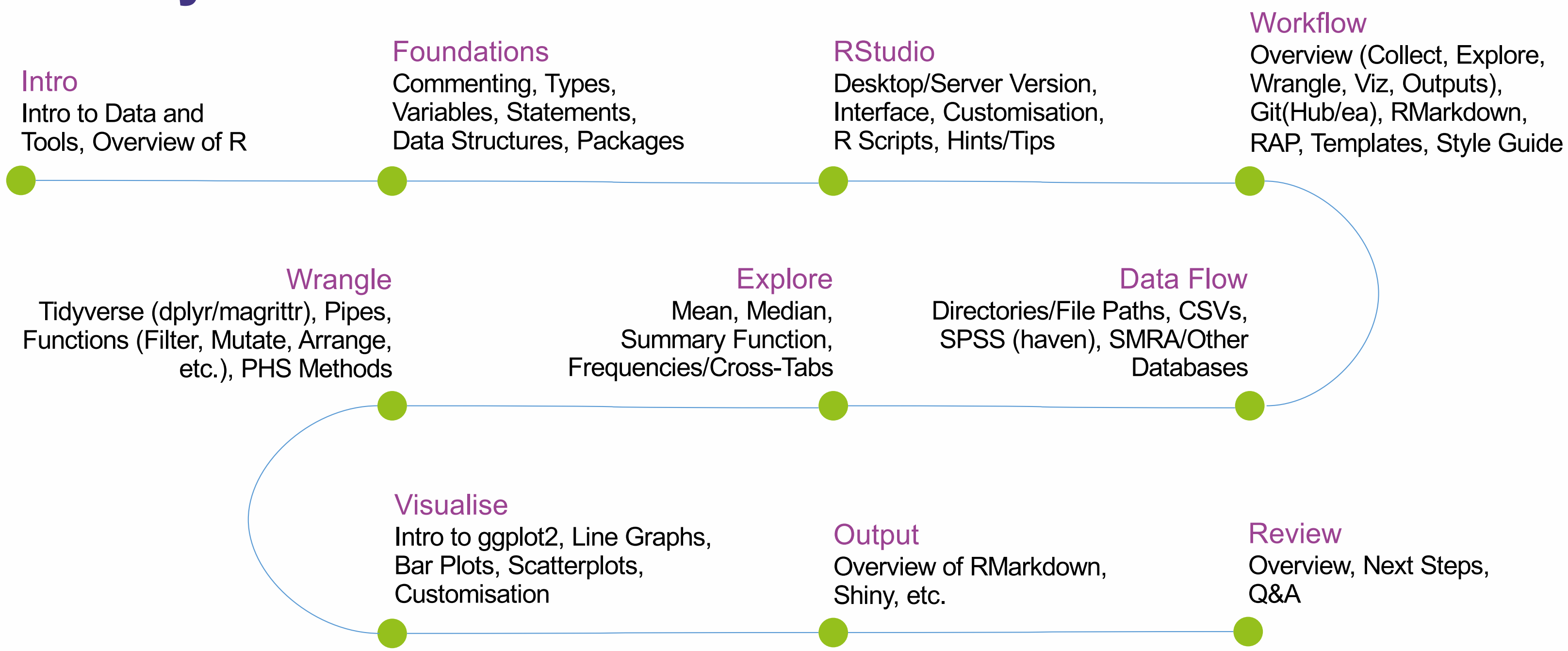
Introduction to R

Transforming Publishing
phs.transformingpublishing@phs.scot

Public Health Scotland

Pathway



Intro
Intro to Data and Tools, Overview of R

Foundations
Commenting, Types, Variables, Statements, Data Structures, Packages

RStudio
Desktop/Server Version, Interface, Customisation, R Scripts, Hints/Tips

Workflow
Overview (Collect, Explore, Wrangle, Viz, Outputs), Git(Hub/ea), RMarkdown, RAP, Templates, Style Guide

Data Flow
Directories/File Paths, CSVs, SPSS (haven), SMRA/Other Databases

Explore
Mean, Median, Summary Function, Frequencies/Cross-Tabs

Wrangle
Tidyverse (dplyr/magrittr), Pipes, Functions (Filter, Mutate, Arrange, etc.), PHS Methods

Visualise
Intro to ggplot2, Line Graphs, Bar Plots, Scatterplots, Customisation

Output
Overview of RMarkdown, Shiny, etc.

Review
Overview, Next Steps, Q&A

Introduction – the why

R

- is a **programming language** widely used for *data analysis*, statistics, and *graphics;*

- **is open source**, available on all major operating systems;

- has the functionality to go from raw data to interactive reports and web apps;

- and it’s part of the **PHS analytical strategy.**

Foundations – the what

Building Blocks

- Basic Data Types – how is fundamental data, like numbers and text, represented?

  This is then the foundations of more complex, composite data types, e.g. tables.
- Variables – named storage to track "objects" across our program
- Statements – a complete line of code, made of expressions and operators
- Control Flow – branching (if statements) and iteration (loops)
- Functions – reusable code that *can* take inputs and give outputs

Basic Data Types

- **Character** (String) – e.g. `"Hello World!"`
- **Numeric** (Float/Real) – e.g. `123.5`
- **Logical** (Boolean) – e.g. `TRUE`

```
typeof("Hello World")
is.numeric(123.5)
print(typeof(TRUE))
```

```
> [1] "character"
> [1] TRUE
> [1] "logical"
```

Type Conversion

`as.<data_type>()`

- `as.character()` conversions tend to succeed without fault
- `as.numeric()` – TRUE and FALSE become 1 and 0, character types needs to be formatted correctly
- `as.logical()` – everything except 0 becomes TRUE for numeric conversions, character can be upper, lower, or proper case versions

```
as.character(123.5)
as.numeric("123.5")
as.logical("False")
```

```
> [1] "123.5"
> [1] 123.5
> [1] FALSE
```

Variables

- **Naming** – letters, numbers, dots `.` or underscores `_` are all okay. However, you **can't** start with an underscore or number, or a dot then a number. Any existing terminology is also reserved from being used as a variable. *Following style guidance is also important.*

- **Assignment** – variables are assigned mainly with `<-` but you may also see `=` being used.

```
# Good
Totals
sumOfPatientsUnder60

# Bad
60YearOldPatients
_template
TRUE
```

Operators

Precedence	Operator	Description
1	^	Exponentiation (right to left evaluation)
2	%%	Modulus
3	* /	Multiplication, Division
4	+ −	Addition, Subtraction
5	< > <= >= == !=	Comparison Operators (Less Than, More Than, Less Than or Equal To, More Than or Equal To, Equal To, Not Equal To)
6	!	Logical NOT
7	& &&	Logical AND
8	| ||	Logical OR

Knowledge Check

We’re going to use an app for a lot of our interactive work, especially today. (Hint: some questions have hint buttons).

scotland.shinyapps.io/phs-rtraining-intro/

- Foundations

Anatomy of a Program

```
# Example 1
hello_world <- "Hello World"
print(hello_world)
```

```
> [1] "Hello World"
```

- `# Example 1` – comment
- `hello_world` – variable
- `<-` – assignment operator (`alt` + –)
- `"Hello World"` – character (" or ')
- print() – function

Style Guide

- **Naming** – variables and filenames should have meaningful names in snake_case format, preferring all lower case.

- **Structure**
  - Space after a comma
  - No spaces before or after parenthesis
  - Comments to explain code and create sections within the code
  - Prefer `"` over `'` for characters

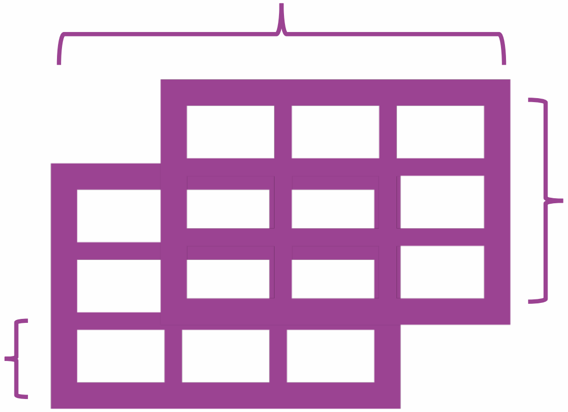
```
# Bad
pts <-c ( 'Al','Bert' )

# Good
patients <- c("Al", "Bert")
```

Foundations – the what

Data Structures

- Vectors
- Lists
- Factors
- Matrices
- Data Frame

- `str()` provides an overview and description of the data structure



Vectors

contain multiple objects of the *same* basic class

- Create: `c(...)` or `vector(<type>, <length>)`
- Access: `<vector>[<index>]`

```
c("a", "c", "f", "b")[1]
c(2, 5, 1, "abc")[3:4]
vector("logical", 4)
```

```
> [1] "a"
> [1] "1" "abc"
> [1] FALSE FALSE FALSE FALSE
```

Knowledge Check

scotland.shinyapps.io/phs-rtraining-intro/

- Data Structures - Vectors

Lists

are a special type of vector that can contain objects of *different* classes, including other lists

- Create: `list(...)`

- Sub-list: `<list>[<index>]`

- Access: `<list>[[<index>]]`

```
list("abc", 4, FALSE)[1:2]
list(list(2, 3), "abc")[[2]]
```

```
> [[1]]
> [1] "abc"
> [[2]]
> [1] 4
```

```
> [1] "abc"
```

Naming Elements

can be done on vectors and lists during or after creation.

- At creation:
  `c("<name>", = <item>)` or `list("<name>", = <item>)`

- After:
  `names(<object>) <- c("<name>")`

```
x <- list("Ch" = "a", "Nm" = 2)
names(x) <- c("Char", "Num")
x$Char
```

```
> [1] "a"
```

Factors

are used to represent categorical data, with both ordered and unordered variations.

- Create: `factor(c(...), ordered = <TRUE/FALSE>)`
- Levels: `factor(c(...), levels = c(...))`

```
x <- factor(c("M", "F", "M"),
     levels = c("F", "M"))
x
```

```
> [1] M F M
Levels: F M
```

Knowledge Check

scotland.shinyapps.io/phs-rtraining-intro/

- Data Structures - Factors

Matrices

expand our dimensions with a `nrow` and `ncol` arguments, constructed column-wise

- Create: `matrix(<data>, nrow = <int>, ncol = <int>)`

- Access: `<matrix>[<row>, <col>]`

```
x <- matrix(1:6, 2, 3)
x
x[2, 3]
```

```
>         [,1][,2][,3]
> [1,]       1   3   5
> [2,]       2   4   6
```

```
> [1] 6
```

Data Frames

are used to store tabular data, each column contains one variable, each row contains an observation

- Create: `data.frame("<name>" = <element(s)>)`

- Subset: `[]`

- Access: `[[]]` or `$`

```
data.frame(name = c("Harry",
                    "Sarah"),
           score = c(62, 91))
```

```
>       name  score
>  1  Harry      62
>  2  Sarah      91
```

Knowledge Check

scotland.shinyapps.io/phs-rtraining-intro/

- Data Structures - Data Frames

Tibbles

data frames that attempt to make our lives a bit easier. First, load the package: `library(tibble)`

- **Create**: `tibble("<name>" = <element(s)>)` or coerce an existing data frame with `as_tibble(<object>)`

- **Subset and Access**: `[]` `[[]]` or `$`

```
tibble(name = c("Harry",
                "Sarah"),
       score = c(62, 91))
```

```
>        name   score
>   1   Harry      62
>   2   Sarah      91
```

Foundations – the what

Anatomy of a Function

Functions allow us to bundle code for reuse, taking inputs, doing something and, optionally, providing outputs.

```
<name> <- function(<inputs…>){
    <code>
    return(<outputs…>)
}
```

```
mult_2 <- function(x){
    x <- x * 2
    return(x)
}
```

```
mult_2(4)
```

```
> [1] 8
```

Packages

are used to expand the functionality of R with more functions

- Install: `install.packages("<package>")`

- Load: `library(<package>)` or `require(<package>)` if loading packages as part of functions as it returns a logical value and a warning if the package isn't installed.

```
install.packages("tidyverse")
library(tidyverse)
```

- output varies by package, warnings (not errors) are normal. An example would be where a function 'masks' that of another R function.

Control Flow - if statements

Package: `dplyr`

Load package, `library(dplyr)`

- `if_else(<condition>, <true>, <false>)`

```
library(dplyr)
x <- 5
if_else(x > 10, TRUE, FALSE)
```

```
> [1] FALSE
```

Knowledge Check

scotland.shinyapps.io/phs-rtraining-intro/

- Control Flow - If

Control Flow - case statements

Package: `dplyr`

Load package, `library(dplyr)`

- `case_when(<condition> ~ <result>)`

```
library(dplyr)
x <- c(1, 2, 3, 4, 5)
case_when(x < 3 ~ "LT3",
          x %% 2 == 0 ~ "Even")
```

```
> [1] "LT3" "LT3" NA "Even" NA
```

Knowledge Check

scotland.shinyapps.io/phs-rtraining-intro/

- Control Flow - Case

Iteration – for loop

allowing us to do the same thing repeatedly with different inputs.

- for(<value> in <sequence>) {<statement>}

```
files <- list.files(pattern = ".csv")
all_files <- list()
for(i in seq_along(files)) {
    all_files[[i]] <-
              read.csv(files[i])
}
```

Iteration – loop with purrr

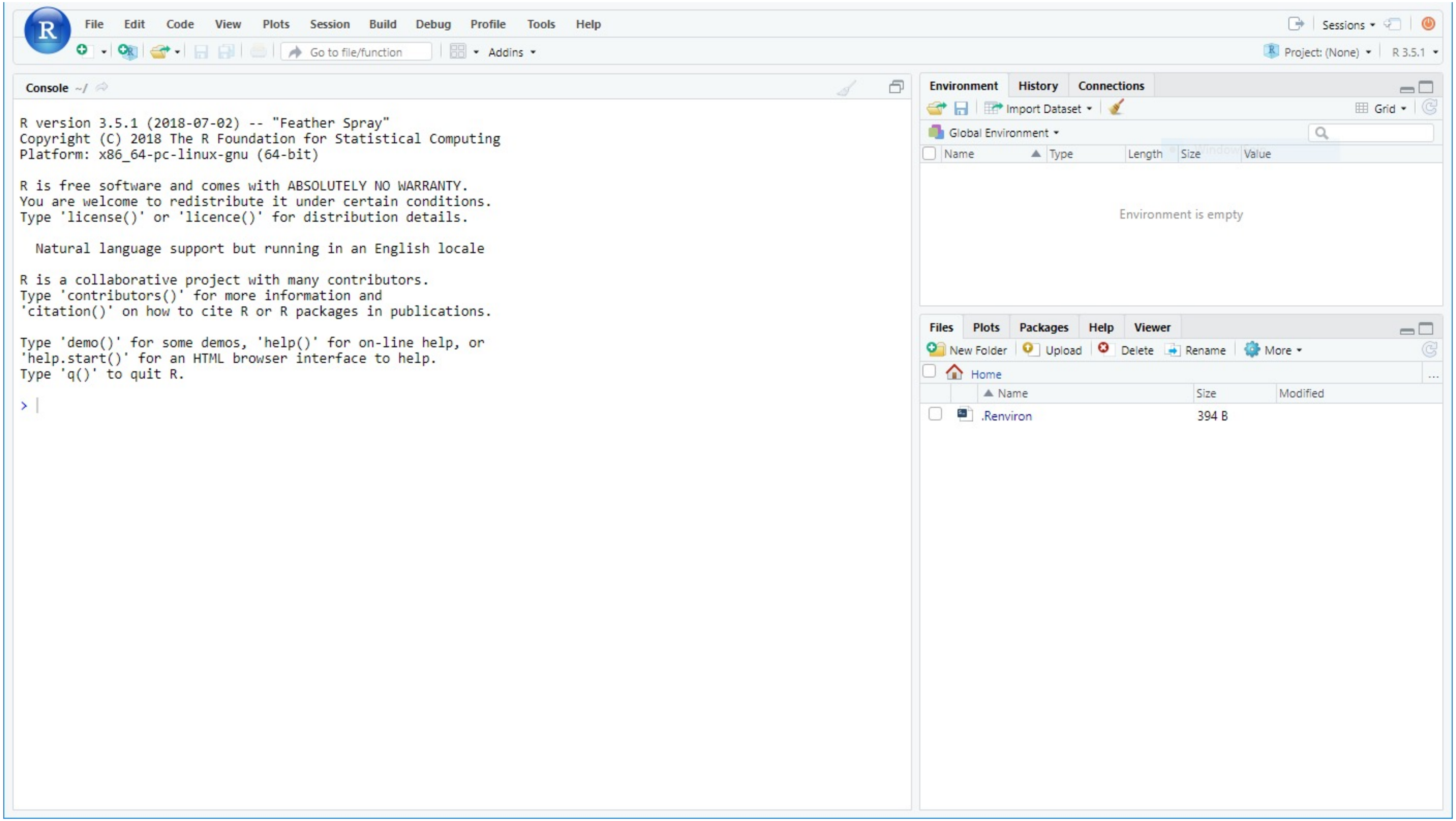
Package: `purrr`

Load: `library(purrr)`

- `map(<object>, <function>)`

```
library(purrr)
files <- list.files(pattern = “.csv”)
all_files <- map(files, read_csv)
```

RStudio



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 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.

> 
```

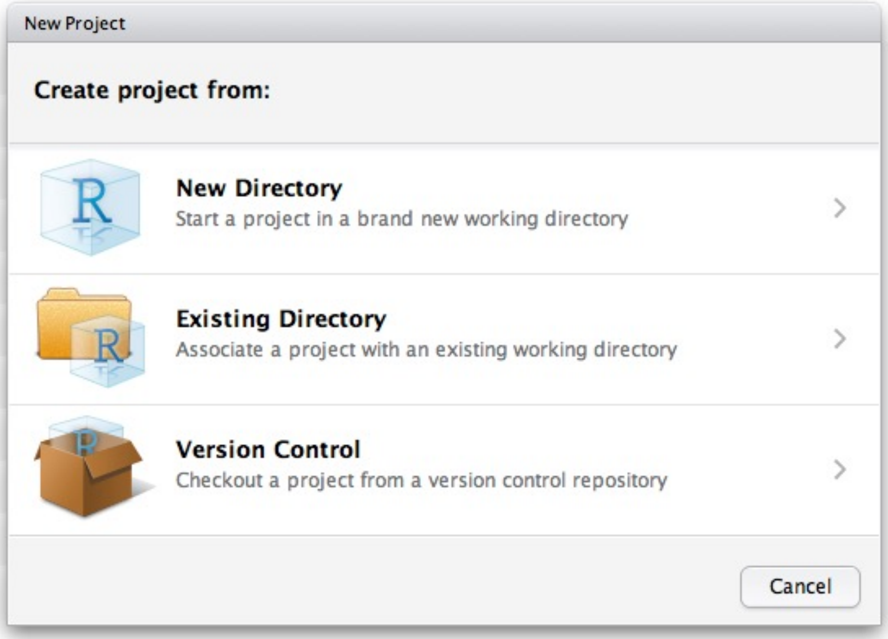
R Projects

keeps work separate, giving a project its own working directory, workspace, and history.

Opening an .Rproj file will:

- Start a new R session
- Load project specifics and settings
- The project directory is set as the current working directory.

- **Create:** available in the Projects menu or global toolbar.



New Project

Create project from:

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Data Flow

Working Directory

- Current: `getwd()`
- Set new: `setwd(<filepath>)`
- We can also use the `here` package, with `here()`

RStudio also provides options through the user interface for navigating files and directories.

```
setwd("/home/learnr01/intro_R")
getwd()
here()
```

```
> [1] "/home/learnr01/intro_R"
> [1] "/home/learnr01/intro_R"
```

Read CSV

Package: `readr`

1. Load package, `library(readr)`
2. `read_csv(<filepath>)`
3. Check output

```
library(readr)
borders_csv <-
     read_csv("data/Borders.csv")
View(borders_csv)
```

RDS

Package: `readr`

Load package, `library(readr)`

Read

- `read_rds(<filepath>)`

Write

- `write_rds(<object>, <filepath>)`

```
library(readr)

borders_RDS <-
     read_rds("data/borders.rds")

write_rds(borders,
     "data/borders.rds")
```

Read SPSS

Package: `haven`

1. Load package, `library(haven)`
2. `read_sav(<filepath>)`
3. Check output

```
library(haven)
borders_spss <-
     read_sav("data/Borders.sav")
View(borders_spss)
```

Read Web

The packages/functions used will vary depending on the structure of the data. This example uses a CSV so the process to follow is the same as before.

Package: `readr`

1. Load package, `library(readr)`
2. `read_csv(<filepath>)`
3. Check output

```
library(readr)
hospital_codes <- read_csv("
     https://www.opendata.nhs.sco
t/dataset/[...].csv")
View(hospital_codes)
```

Open Data – CKAN API

Package: `ckanr`

1. Load package, `library(ckanr)`
2. Set up connection and resource ID
3. `dplyr::tbl(<src>, <res>) %>%`
   `as_tibble()`
4. Check output

```
library(ckanr)
ckan <-
     src_ckan("https://www.openda
     ta.nhs.scot")
res_id <- ”<ID>"
resource <- dplyr::tbl(src =
ckan$con, from = res_id) %>%
   as_tibble()
```

Database (SMRA)

Package: `odbc`

1. Load package, `library(odbc)`
2. Connect:
   ```
   smra_connection <- dbConnect(drv = odbc(),
               dsn = "SMRA",
               uid = .rs.askForPassword("SMRA Username:"),
               pwd = .rs.askForPassword("SMRA Password:"))
   ```
3. Extract:
   ```
   smr01 <- dbGetQuery(smra_connection,
               paste("<sql>"))
   ```
4. Check output

Write CSV

Package: `readr`

1. Load package, `library(readr)`
2. `write_csv(<object>, <filepath>)`
3. Check output

```
library(readr)
write_csv(borders,
          "data/borders.csv")
```

- the write functions expect a dataframe as the object.

Explore

Mean/Median & Summary

- `mean()` and `median()` are passed arrays of values (usually from a data frame) to return the mean and median value

```
mean(borders[["LengthOfStay"]])
summary(borders$LengthOfStay)
```

- `summary()` returns all summary statistics based on a given array (usually from a data frame)

```
> [1] 4.297008
>   Min. 1st Qu. Median   Mean  3rd Qu.   Max
  0.000   0.000   1.000  4.297   4.000   458
```

- We access columns using `[[]]` or `$`

Frequencies & Crosstabs

- Frequency: `table(<df>$<col>)`
- Crosstab: `table(<df>$<col1>, <df>$<col2>)`
- To add column and row totals, the function `addmargins()` can be used

```
addmargins(table(borders$Hospita
lCode, borders$Sex))
```

```
            1      2     3     Sum
A210H       1      0     0       1
B102H      56    100     0     156
B103H      50    108     0     158
...
Sum     11947  13340     2   25289
```

Exercise 1

1. Read in "`Borders.csv`" (giving the data frame an appropriate name)
2. What are the **mean, median, and max** values from the `LengthOfStay` variable? Can you do this in one step?
3. Produce a **frequency table** to check the `sex` variable, save this as an object with an appropriate name
4. Export the frequency table as a **csv** file. You'll need to use `as.data.frame()`

Wrangle

Tidyverse

is a suite of packages for data exploration, manipulation, and visualisation; it’s best practice to utilise these where possible.

- functions have a consistent format, i.e. `function(data, task)`

- gives us the package `dplyr`

dplyr

is a grammar of data manipulation, providing a set of "*verbs*" to help solve most data manipulation challenges

```
library(dplyr)
```

- `filter()`
- `mutate()`
- `arrange()`
- `select()`
- `group_by()`
- `summarise()`
- `count()`
- `rename()`
- `recode()`

Pipe Operator

- `%>%` is used to link functions together, passing the previous to the next
- Using the pipe operator makes R code more readable and prevents extensive parenthesis building up with multiple function calls
- Readable as "and then"
- Shortcut: (`ctrl` + `shift` + `M`)

```
arrange(filter(borders,
     HospitalCode == "B102H"), Dateofbirth)
```

```
borders %>%
     filter(HospitalCode == "B102H") %>%
     arrange(Dateofbirth)
```

Filter

```
filter(<data>, <logical
expression>)
```

- picks cases based on their values

```
# all cases with E12 specialty
borders %>%
     filter(Specialty == "E12")

# B120H cases more than 10 days
borders %>%
     filter(HospitalCode ==
               "B120H" &
             LengthOfStay > 10)
```

Mutate

```
mutate(<data>, <new_col> =
<expression>)
```

- adds new variables that are functions of existing variables

```
# length of stay divided by 2
borders %>%
    mutate(los_div2 =
         LengthOfStay / 2)
```

Arrange

```
arrange(<data>,
<variables>)
```

- changes the ordering of rows
- `desc()` to sort in descending order

```
# sort by Hospital Code
borders %>%
     arrange(HospitalCode)
```

Select

```
select(<data>,
<expression>)
```

- picks variables based on their names

- prepend "`-`" to a variable to remove

```
# remove Postcode
borders %>%
      select(-Postcode)
```

Exercise 2

1. Read in "`Borders.csv`" (giving the data frame an appropriate name)
2. Which patients had a `LengthOfStay` of between 2 and 10 days?
3. Which of these patients were under `Specialty` E12 or C8?
4. Remove all columns other than `URI`, `Specialty`, and `LengthOfStay`
5. Complete all the above using pipes and write this to a CSV ordered by `LengthOfStay`

Group By

```
group_by(<data>,
<col_name>)
```

- groups variables to perform operations
- This doesn’t visibly affect the data, but we can see the output shows the grouping. We can then perform other operations on the groups.

```
# sort by Hospital Code
borders %>%
     group_by(HospitalCode)
```

```
> ...
# Groups: HospitalCode [48]
...
```

Summarise

```
summarise(<data>, <name> =
<expression>)
```

- reduced multiple values down to a single summary

```
# avg length of stay by hospital
borders %>%
    group_by(HospitalCode) %>%
    summarise(mean_los =
        mean(LengthOfStay))
```

Count

```
count(<data>, <variables>)
```

- useful for running frequencies, this calls group_by() and produces counts for a specified column

- sort by descending order using `sort = TRUE` as an argument

```
# counts of specialty
borders %>%
     count(Specialty, sort = TRUE)
```

Exercise 3

1. Read in "`Borders.csv`" (giving the data frame an appropriate name)

2. What is the earliest admission date by specialty?

3. What is the latest discharge date by specialty?

4. What are the number of admissions per hospital, per specialty?

Rename

```
rename(<data>, <new_name> =
<existing_name>)
```

- renaming specific columns in a data frame

```
# rename Date of Birth column
borders %>%
    rename(date_of_birth =
        Dateofbirth)
```

Recode

```
mutate(<col> = recode(<col>,
<existing_code> =
<new_code>))
```

- for changing values within a column
- works best when used with `mutate()`

```
# change hospital code
borders %>%
    mutate(HospitalCode =
        recode(HospitalCode,
            "B120V" = "B120H"))
```

Exercise 4

1. Select the `URI`, `Specialty`, and `Dateofbirth` columns from the borders data and save to a new data frame.
2. Arrange this new data in ascending order by `Specialty` and check the results.
3. Extract the records with a missing `Dateofbirth` (hint: `?filter`)
4. Finally, recode `Specialty` "A1" to be "General Medicine"

Joining Tables

```
<type>_join(<data1>,
<data2>, by =
<common_variable>)
```

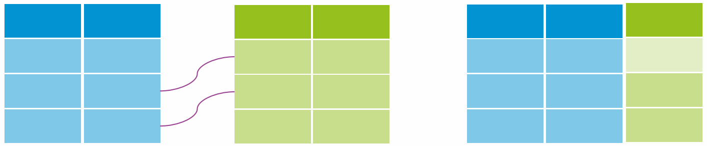
- for merging data by matching together using common variable(s)

```
# merge baby data
baby5 <- read_csv("data/Baby5.csv")
baby6 <- read_csv("data/Baby6.csv")
baby_joined <-
     left_join(baby5, baby6, by =
          c("FAMILYID", "DOB"))
```

Join Types

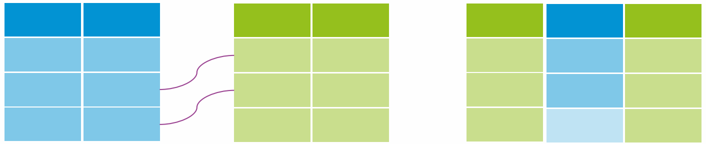
left_join()

all rows from the ‘left’, any matches from the ‘right’



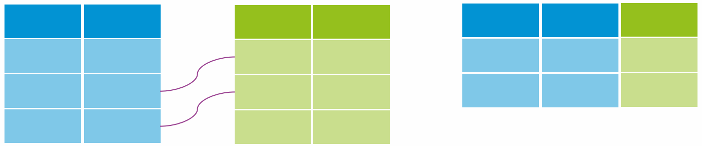
right_join()

all rows from the ’right’, any matches from the ’left’



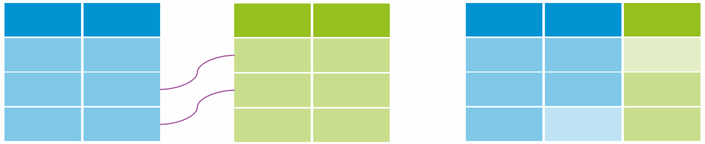
inner_join()

rows of matched fields from data sets



full_join()

all rows, na for non-matched fields



General Skills

Debugging

1. **Review warnings/errors** – these can appear cryptic but use Google and some will become familiar. Checking the functions could help – `?<function>`
2. **Narrow the problem** – step through the code, isolating the issue.
3. **Google/Stack Overflow** – this can be specific to the bug or more general to the problem you're trying to solve.
4. **Pair up** – sometimes a fresh pair of eyes makes the difference. Post a message on the R User Group Technical Queries Teams channel

Continuous Learning

- Data Science Knowledge Base – (People Development Hub) is the place for all content related to Data Science learning:
  - **Review, Follow, and Contribute to Guidance** – guidance is for sharing best practice, maintaining security, and improving efficiency.
  - **Expand your skills** – take another course to build your R skills or on related technologies (e.g. Git).
  - **Keep up to date** – our infrastructure is improving, we support knowledge sharing events, and so much more!

Review

Project

scotland.shinyapps.io/phs-rtraining-intro/

- Day 2 Project – Handwashing

Feel free to follow along for the project on the app or build a script on RStudio.

Next Steps

- Homework project & day 3
- Embed your new knowledge and skills!
- Expand your knowledge and skills with related technologies (e.g. git)
- Take R Further - look at other training opportunities (`phsmethods`)

Getting Help

- Vignettes (Help) / `?<function>`

- Google / Stack Overflow

  tag queries "[r] & [tidyverse]"

- R User Group Teams – Technical Queries

- Transforming Publishing