

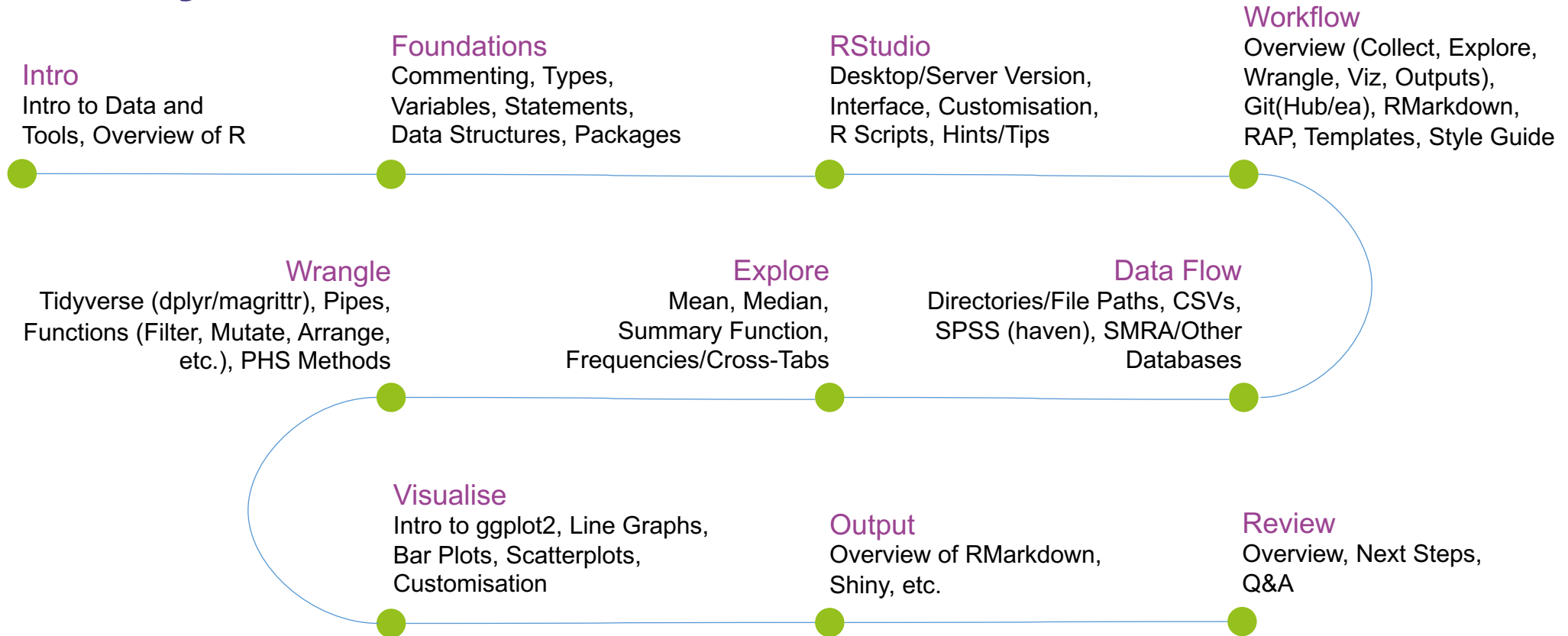
# Introduction to R

v2.0

Transforming Publishing

[phs.transformingpubilshing@phs.scot](mailto:phs.transformingpubilshing@phs.scot)

# Pathway



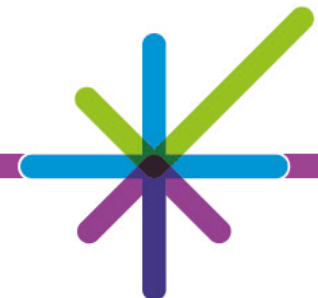
## **Introduction – the why**

# Data



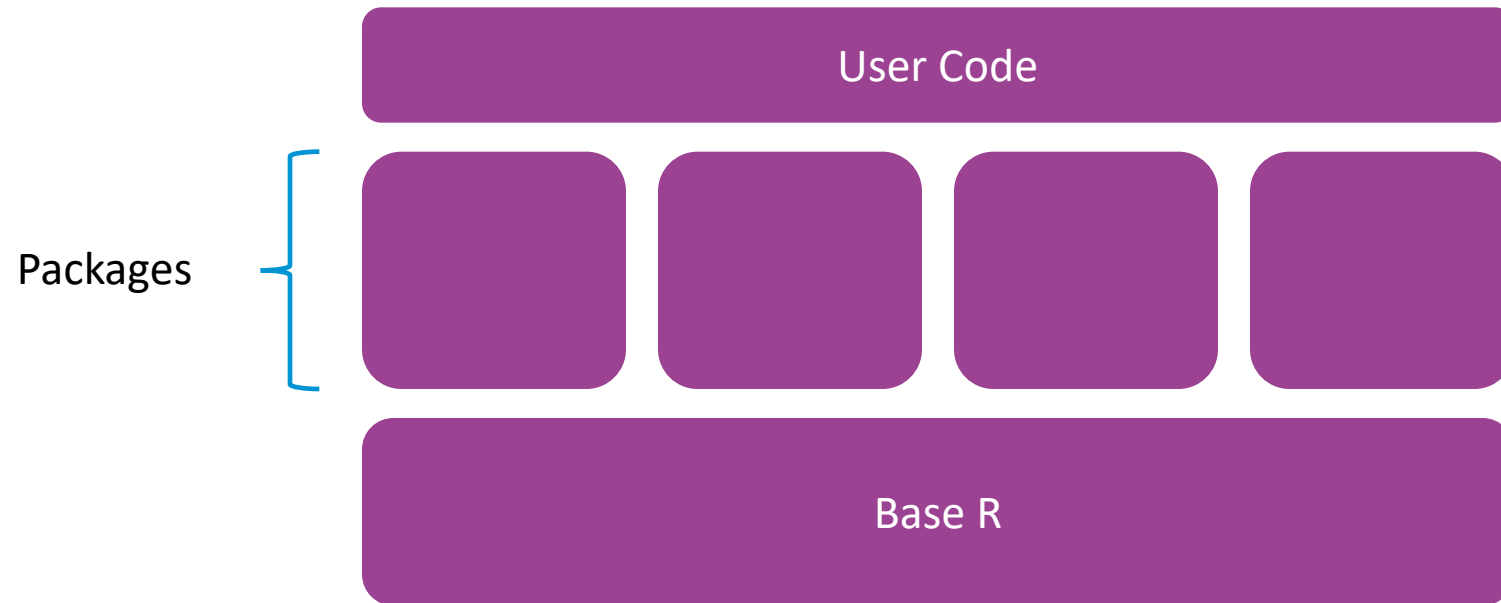
# R

- 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.





# R Structure



# Input

```
# 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





# Basic Data Types

- Character – e.g. "Hello World!"
- Numeric (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"
```



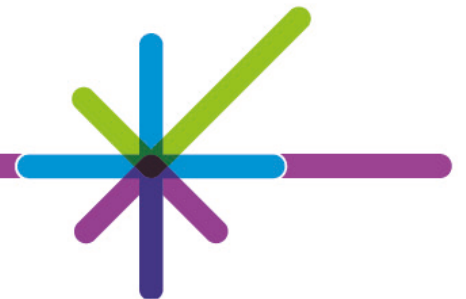
# Type Conversion

```
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

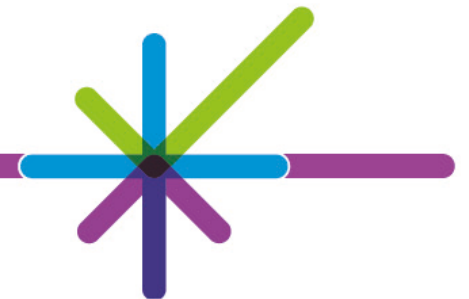
```
as.character(123.5)  
as.numeric("123.5")  
as.logical("False")
```

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



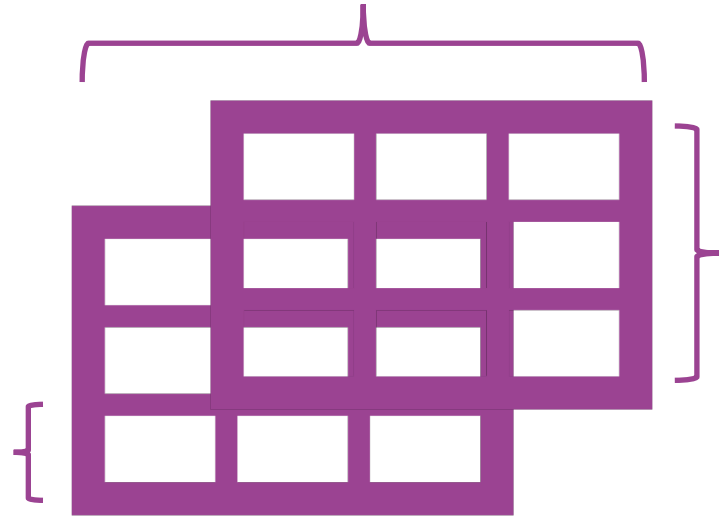
# Operators

| Precedence | Operator                       | Description   |
|------------|--------------------------------|---|
| 1          | $\wedge$                       | Exponentiation (right to left evaluation)   |
| 2          | $\% \%$                        | Modulus   |
| 3          | $*$ $/$                        | Multiplication, Division  |
| 4          | $+$ $-$                        | Addition, Subtraction   |
| 5          | $<$ $>$ $<=$ $>=$<br>$==$ $!=$ | 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  |



# Data Structures

- Vectors
- Lists
- Factors
- Matrices
- Data Frame
- `str()` provides an overview and description of the data structure



# Vectors

- contain objects of the *same* basic class
- Create: `c(...)` or `vector(<type>, <length>)`
- Access: `<vector>[<index>]`

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

```
> [1] FALSE FALSE FALSE FALSE
> [1] "a"
> [1] "5"
```



# 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 List Elements

- can be done during or after creation.

- At creation:

```
list("<name>", = <item>)
```

- After:

```
names(<list>) <-  
c("<name>")
```

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

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



# Factors

- are used to represent categorical data

- Create: `factor(c(...))`

- Levels: `factor(c(...),  
levels = c(...))`

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

```
x
```

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





# Matrices

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

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

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

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

```
> [1] 6
```

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



# 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
```



# Packages

- are used to expand the functionality of R with additional functions

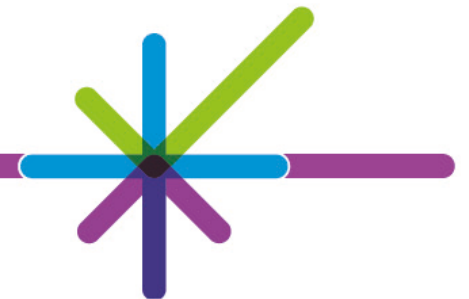
- **Install:**

```
install.packages("<package>")
```

- **Load:** `library(<package>)` or `require(<package>)`

```
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.*



**RStudio**

Console ~/

```
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.

> |
```

Environment History Connections

Global Environment

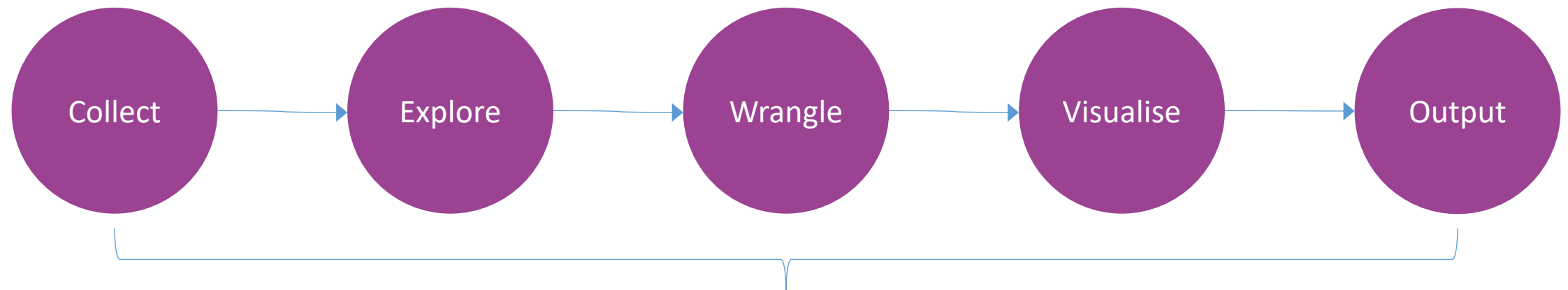
| Name                 | Type | Length | Size | Value |
|----------------------|------|--------|------|-------|
| Environment is empty |      |        |      |       |

Files Plots Packages Help Viewer

New Folder Upload Delete Rename More

| Name      | Size  | Modified |
|-----------|-------|----------|
| .Renviron | 394 B |          |

**Workflow**



Process

Foundations



**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 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)
```



# 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.scot/dataset/\[...\].csv")
View(hospital_codes)
```



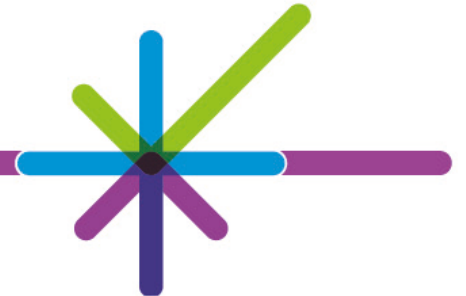
# Database (SMRA)

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



# RDS

Package: readr

Load package, `library(readr)`

## Read

- `read_rds(<filepath>)`

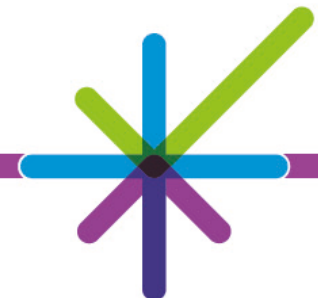
## Write

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

```
library(readr)

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

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



# 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
- `summary()` returns all summary statistics based on a given array (usually from a data frame)
- We access columns using `[[]]` or `$`

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

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



# 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$HospitalCode, borders$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

```
library(dplyr)
```

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

- |                           |                            |
|---------------------------|----------------------------|
| • <code>filter()</code>   | • <code>summarise()</code> |
| • <code>mutate()</code>   | • <code>count()</code>     |
| • <code>arrange()</code>  | • <code>rename()</code>    |
| • <code>select()</code>   | • <code>recode()</code>    |
| • <code>group_by()</code> |                            |



# 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)

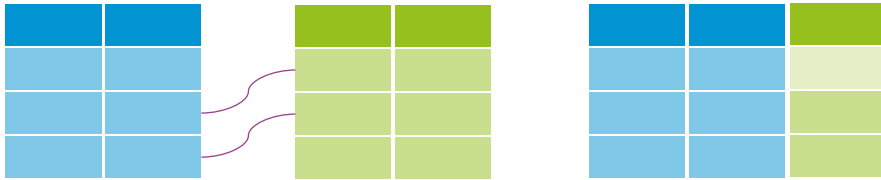
```
# merge baby data  
baby5 <- read_csv("data/Baby5.csv")  
baby6 <- read_csv("data/Baby6.csv")  
baby_joined <-  
  left_join(baby5, baby6, by =  
    c("FAMILYID", "DOB"))
```



# Join Types

`left_join()`

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



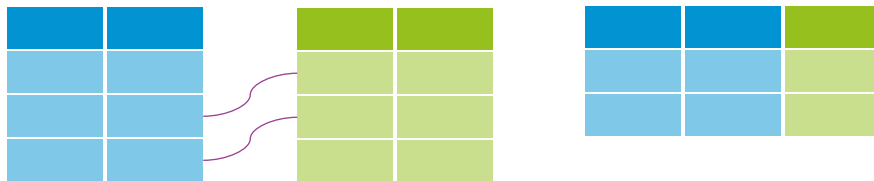
`right_join()`

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



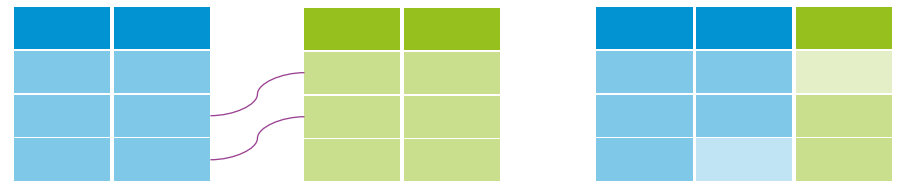
`inner_join()`

rows of matched fields from data  
sets



`full_join()`

all rows, na for non-matched  
fields

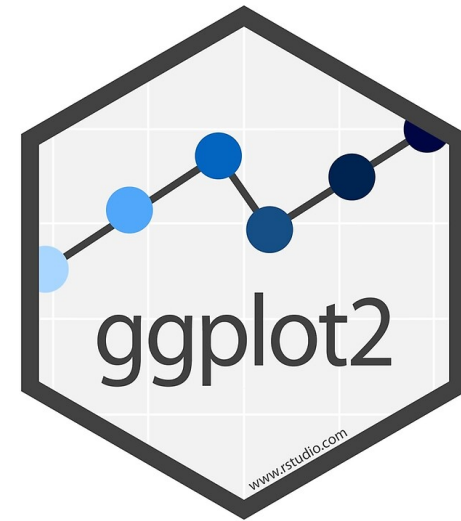


**Visualise**

# ggplot2

```
library(ggplot2)
```

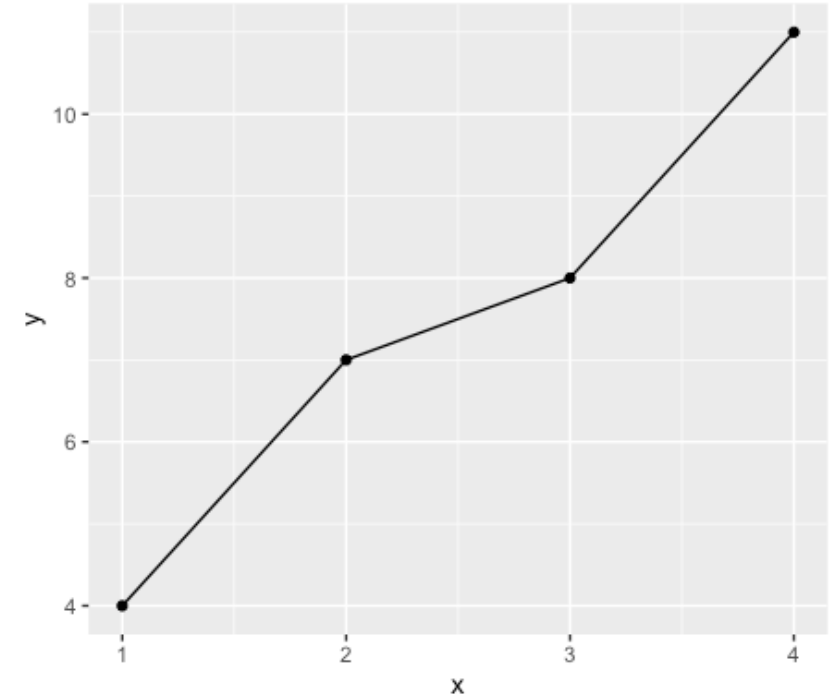
- is a system for creating graphics to visualise data
- Begin with a call to `ggplot()`, supplying the data and aesthetic mappings using `aes()`
- Then add layers, scales, coords, and facets with `+`



# Line Graph

```
# Define data
data <- data.frame(
  x = c(1, 3, 2, 4),
  y = c(4, 8, 7, 11))

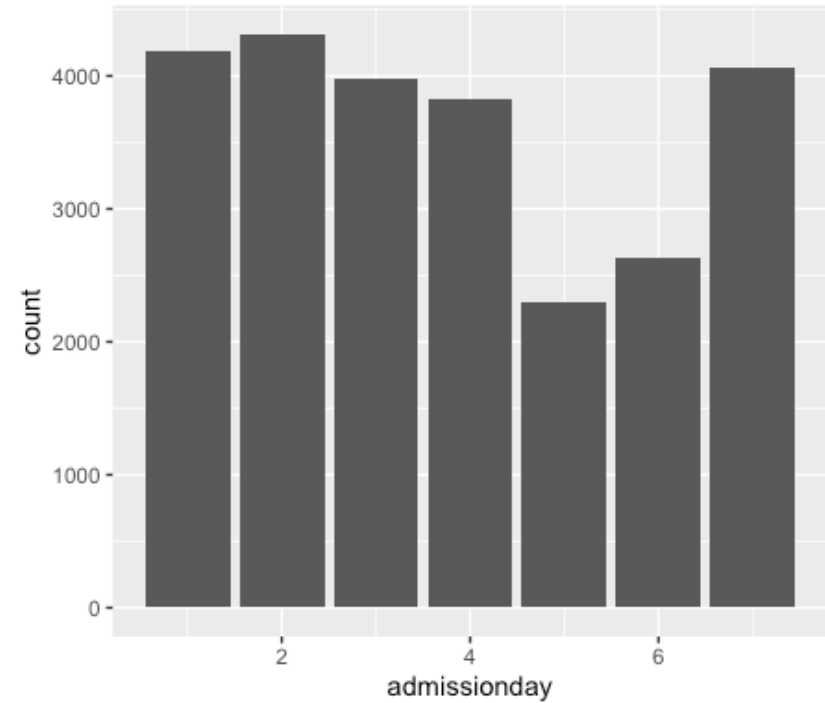
# Plot the data
line_plot <- ggplot(data,
  aes(x, y)) +
  geom_line() +
  geom_point()
```





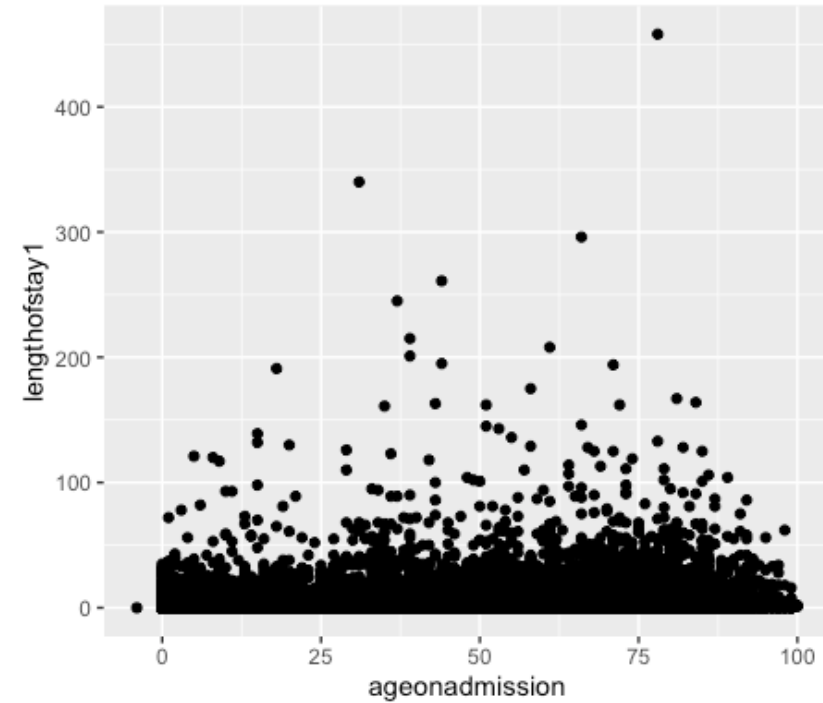
# Bar Plot

```
# Define data  
data <- read_csv("data/Borders.csv")  
  
# Plot the data  
bar_plot <- ggplot(data) +  
  geom_bar(aes(x =  
    admissionday))
```



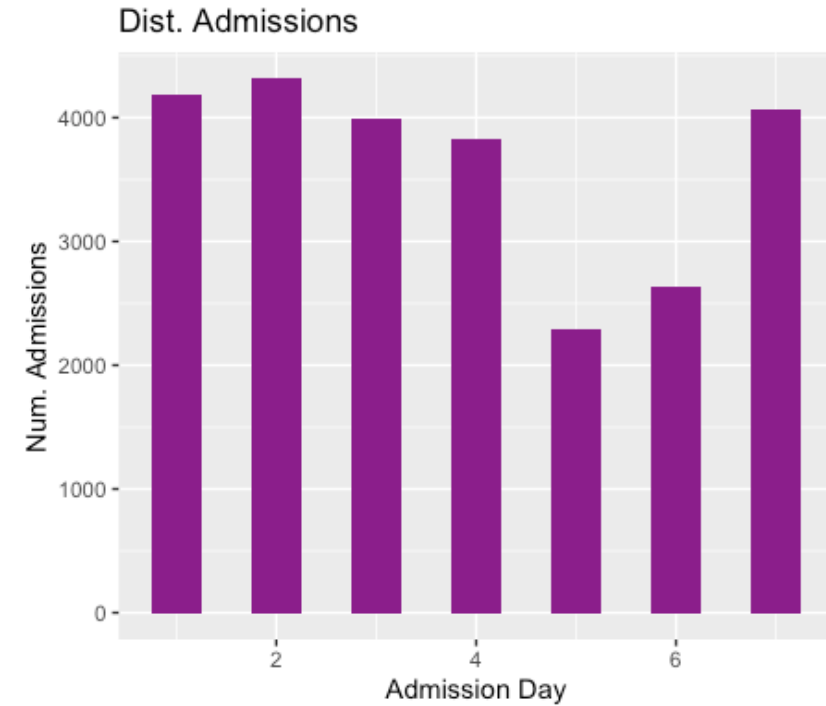
# Scatter Plot

```
# Define data  
data <- read_csv("  
  data/BORDERS (inc Age).csv")  
  
# Plot the data  
scatter_plot <-  
  ggplot(data, aes(x =  
    ageonadmission,  
    y = lengthofstay1)) +  
  geom_point()
```



# Customisation

```
# bar_plot colour and width
bar_plot <- ggplot(data) +
  geom_bar(aes(x =
    admissionday),
    fill = "magenta4",
    width = 0.5) +
  # add titles
  xlab("Admission Day") +
  ylab("Num. Admissions") +
  ggtitle("Dist. Admissions")
```



## Exercise 5

1. Read in "Borders.csv" (giving the data frame an appropriate name)
2. Filter data to show only records from HospitalCode "B109H"
3. Create a histogram for patient's LengthOfStay, check the output
4. Add axis labels and a title to your histogram
5. Save the plot (as a PNG) to the plots folder in your working directory



**Output**

# rmarkdown

- allows for the production of fully reproducible documents, with R (and other language) 'chunks' included
- supports many static and dynamic outputs, including PDF, MS Word, HTML, books, presentations, etc.
- is used in RAP publications, infographics, and more within PHS



# shiny

- is for building interactive web apps through R;
- it provides an interactive approach to sharing data and analyses and can be hosted online or embedded in RMarkdown documents
- is used for some PHS publications and in development for more



**Review**



# Getting Help

- Vignettes (Help) / ``?<function>``
- Google / Stack Overflow  
tag queries "[r]"
- [R User Group Teams](#) – [Technical Queries](#)
- [Transforming Publishing](#)



# Next Steps

- 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)

