Introduction to R

R studio vs command line

R studio is an interactive interface with many tools available, this is the best option when using locally, and we can also use it on demand in VACC. However, when performing memory intensive processes, we should make scripts in a text editor (e.g., BBedit) and execute these scripts on the command line.

Accessing R via the command line in VACC

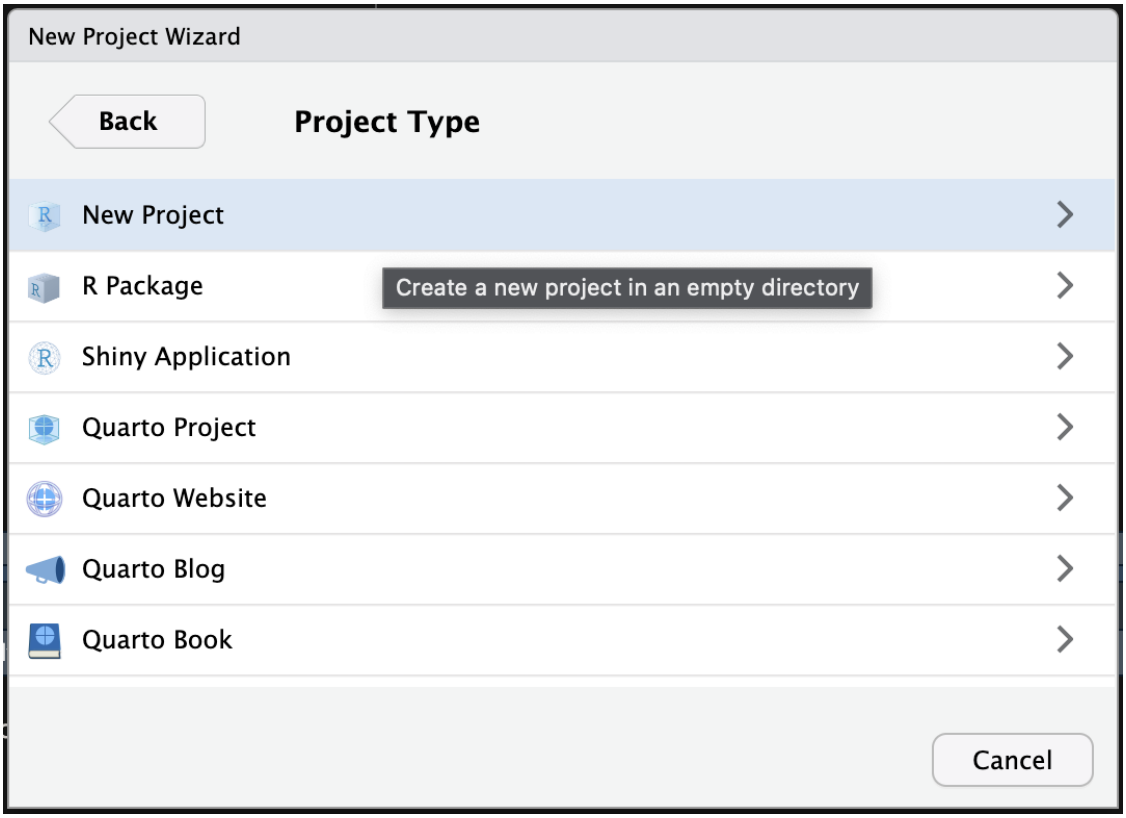
```
module load Rtidverse

R

>
```

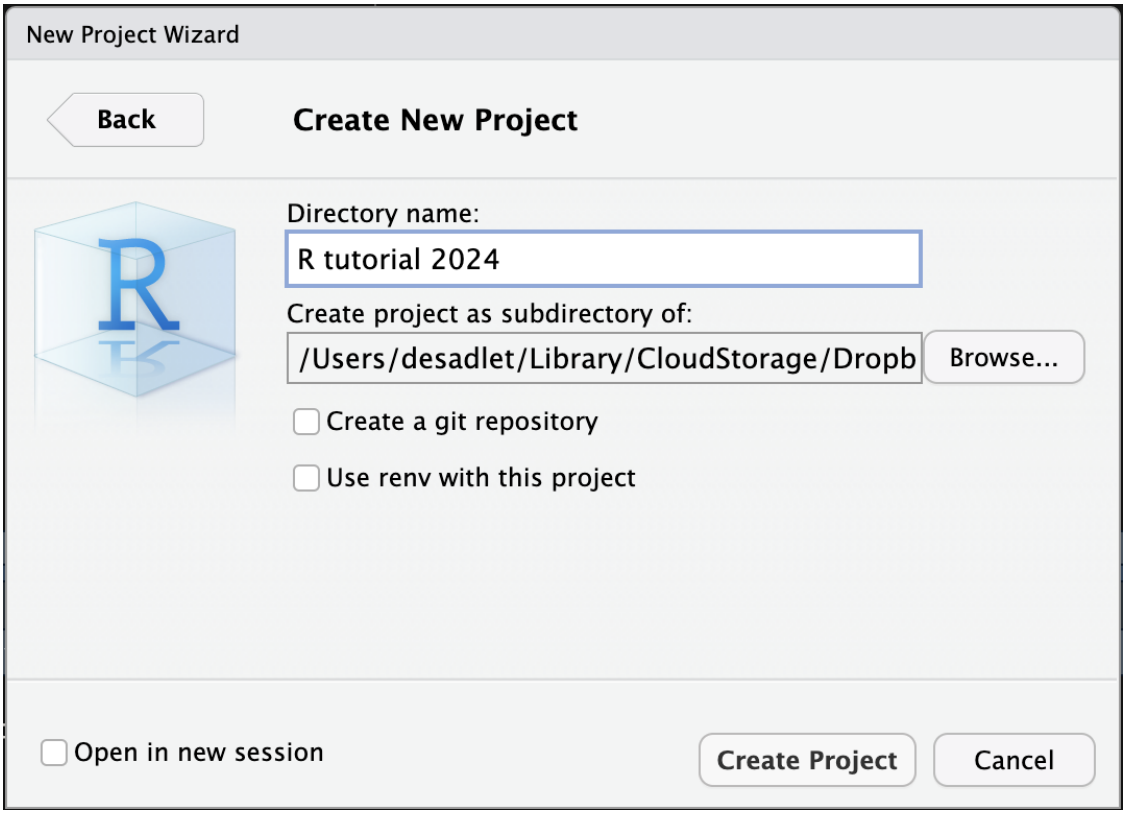
An important note early on!

TAB is your best friend! If you forget the name of a function/module e.g. what was after R? then tab will auto-complete if its on your file path!

Using R studio

Whenever starting a task in R, it is best to create a R project, to keep track of all inputs, scripts and outputs for the project. For example, lets make one for this tutorial


New Project Wizard
Back
Project Type
New Project
R Package
Create a new project in an empty directory
Shiny Application
Quarto Project
Quarto Website
Quarto Blog
Quarto Book
Cancel


New Project Wizard
Back
Create New Project
Directory name:
R tutorial 2024
Create project as subdirectory of:
/Users/desadlet/Library/CloudStorage/Dropb
Browse...
Create a git repository
Use renv with this project
Open in new session
Create Project
Cancel

Checking we are in the right place

In unix we would use `pwd` (present working directory) whilst in r we would use `getwd` to see where we are in our directory.

```
getwd()
## get working directory
setwd("/Users/desadlet/Library/CloudStorage/Dropbox/Postdoc/Rtutorial/Rtutorial2024")
##set working directory
```

Note that we need to put the working directory within ” ” in R

R tends to think objects without quotations are a data object with something in for example x vs “x”

```
X = 10

print(x)

[1] 10

print(“x”)

[1] "x"
```

Using packages

Although we can do a lot already in base R, we will often want to install and use packages to perform more specific functions, especially with population genetics.

To install packages:

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

R studio will often suggest a package as you are typing if you cant remember the exact name.

To then use the package:

```
library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0
──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.1     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts()
```

```
──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all co
nflicts to become errors
```

However not all packages are in the cran library…

… So in comes Bioconductor, which is a large depository for bioinformatics software, so we often require it for population genetics

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("microbiome")
```

We can use :: to find an exact function within a package, as sometimes function names overlap with other packages you may have open.

HELP! MY CODE ISNT WORKING!

As with any new language, R will be trial and error at first, but a very important function in R is `?` which will give you a manual or help file for a function

```
?library

?ggplot2::aes
```

Some common mistakes in R

- Case sensitivity, `Library()` will not do the same as `library()`
- Missing commas e.g., `1 2 3` instead of `1, 2, 3`
- Mismatched parentheses or brackets, e.g., head(df[,1] is missing the matching)
- Not quoting (“”), e.g., `read_csv(filename)` instead of `read_csv(“filename”)`
- Not finishing a command, meaning a + will come up in console
- Variable types - a factor wont do the same thing as a character! Use `str` (structure command) to check you have the correct variable type
- Being in the correct directory for your data (use `getwd` and make sure data is saved in your project folder)
- Repeating variable names, using `df<-` twice will override your variable! Be extra careful when modelling, often we save results of a model into variables like `m1` or `model1`, be more precise!

Importing data

We will mostly be using tidyverse in our coding, but I will also include solutions in base R. Tidyverse is a more efficient way of coding and has several advantages over base R in many scenarios.

```
library(tidyverse)

Met<-read_csv("ThermalStressMetabolic.csv")

Rows: 180 Columns: 16
── Column specification ────────────────────────────────────────────────────
Delimiter: ","
chr (7): Date, Time, Tank, Cage, Treat, Line, Color
dbl (9): Photo, Length, Temp, Chamber.Respiro, Last.recorded.weight, Weight,.
..

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this mess
age.

##Remember quotations!
```

Or in base R

```
Met<-read.csv("ThermalStressMetabolic.csv")
```

Exercises

1. What are some other options with read_csv?
2. What are other file types we may come across, and how would you import them?
3. What if we want to keep and read our data files from ~/data

Exploring data

Before we do any kind of analysis or data wrangling, we should see what exactly our data is. One way is just to type in the new variable we imported and assigned, in this case we called it Met.

Met data is a subset of metabolic rate related data from a large thermal stress experiment on different selection lines of zebrafish representing different harvesting regimes (https://onlinelibrary.wiley.com/doi/full/10.1002/ece3.11007)

```
Met

# A tibble: 180 × 16
   Photo Length Date     Time       Temp Tank  Cage  Treat Line  Chamber.Resp
iro
   <dbl>  <dbl> <chr>    <chr>     <dbl> <chr> <chr> <chr> <chr>           <d
bl>
```

```
 1  6789   24.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
3
 2  6790   22.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
4
 3  6791   21.1 7.8.2021 Evening      34 T1    I5    LS    LS1
1
 4  6770   19.2 6.8.2021 Morning      34 T2    H1    LS    LS1
1
 5  6772   21.4 6.8.2021 Morning      34 T2    H1    LS    LS1
2
 6  6774   19.4 9.8.2021 Morning      34 T2    H1    LS    LS1
1
 7  6775   22.0 6.8.2021 Morning      34 T2    H1    LS    LS1
3
 8  6696   18.4 9.8.2021 Morning      34 T3    G5    LS    LS1
4
 9  6698   21.2 9.8.2021 Afternoon    34 T3    G5    LS    LS1
1
10  6699   22.0 9.8.2021 Morning      34 T3    G5    LS    LS1
3
# i 170 more rows
# i 6 more variables: Last.recorded.weight <dbl>, Weight <dbl>,
#   `SMR (g/hr)` <dbl>, `MMR (g/hr)` <dbl>, r2 <dbl>, Color <chr>
```

But wait what is a tibble?!

From the tidyverse help manual: Tibbles are **data frames that are lazy and surly: they do less (i.e. they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist)**.

We can notice that data types are listed as <dbl> or <chr> in this dataset

chr= character

dbl= number

(more on variable types soon)

```
dim(Met)
```

```
[1] 180  16
```

```
head(Met)
```

```
# A tibble: 6 × 16
  Photo Length Date     Time       Temp Tank  Cage  Treat Line  Chamber.Respi
ro
  <dbl>  <dbl> <chr>    <chr>     <dbl> <chr> <chr> <chr> <chr>           <db
l>
1  6789   24.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
3
2  6790   22.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
```

```
4
3  6791   21.1 7.8.2021 Evening      34 T1    I5    LS    LS1
1
4  6770   19.2 6.8.2021 Morning      34 T2    H1    LS    LS1
1
5  6772   21.4 6.8.2021 Morning      34 T2    H1    LS    LS1
2
6  6774   19.4 9.8.2021 Morning      34 T2    H1    LS    LS1
1
# ℹ 6 more variables: Last.recorded.weight <dbl>, Weight <dbl>,
#   `SMR (g/hr)` <dbl>, `MMR (g/hr)` <dbl>, r2 <dbl>, Color <chr>
```

```
tail(Met)
```

```
# A tibble: 6 × 16
  Photo Length Date      Time       Temp Tank  Cage  Treat Line  Chamber.Resp
iro
  <dbl>  <dbl> <chr>     <chr>     <dbl> <chr> <chr> <chr> <chr>           <d
bl>
1  6858   23.5 14.8.2021 Afternoon    22 T8    B3    SS    SS2
2
2  6859   24.0 14.8.2021 Afternoon    22 T8    B3    SS    SS2
1
3  6860   21.8 14.8.2021 Afternoon    22 T8    B3    SS    SS2
3
4  6861   21.5 14.8.2021 Afternoon    22 T8    B3    SS    SS2
4
5  6871   18.7 16.8.2021 Afternoon    22 T9    A2    SS    SS2
3
6  6872   20.9 16.8.2021 Afternoon    22 T9    A2    SS    SS2
4
# ℹ 6 more variables: Last.recorded.weight <dbl>, Weight <dbl>,
#   `SMR (g/hr)` <dbl>, `MMR (g/hr)` <dbl>, r2 <dbl>, Color <chr>
```

```
str(Met)
```

```
spc_tbl_ [180 × 16] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Photo               : num [1:180] 6789 6790 6791 6770 6772 ...
 $ Length              : num [1:180] 24.8 22.8 21.1 19.2 21.4 ...
 $ Date                : chr [1:180] "7.8.2021" "7.8.2021" "7.8.2021" "6.8.20
21" ...
 $ Time                : chr [1:180] "Afternoon" "Afternoon" "Evening" "Morni
ng" ...
 $ Temp                : num [1:180] 34 34 34 34 34 34 34 34 34 34 ...
 $ Tank                : chr [1:180] "T1" "T1" "T1" "T2" ...
 $ Cage                : chr [1:180] "I5" "I5" "I5" "H1" ...
 $ Treat               : chr [1:180] "LS" "LS" "LS" "LS" ...
 $ Line                : chr [1:180] "LS1" "LS1" "LS1" "LS1" ...
 $ Chamber.Respiro     : num [1:180] 3 4 1 1 2 1 3 4 1 3 ...
 $ Last.recorded.weight: num [1:180] 0.442 0.168 0.127 0.092 0.115 0.098 0.14
5 0.086 0.135 0.051 ...
```

```
 $ Weight              : num [1:180] 0.221 0.166 0.128 0.1 0.12 0.091 0.169 0
.09 0.138 0.145 ...
 $ SMR (g/hr)          : num [1:180] 0.205 0.148 0.202 0.238 0.229 ...
 $ MMR (g/hr)          : num [1:180] 0.447 0.909 0.457 0.411 0.427 ...
 $ r2                  : num [1:180] 0.99 0.99 0.98 0.98 0.98 0.98 0.96 0.99
0.98 0.98 ...
 $ Color               : chr [1:180] "B" "O" "U" "Y" ...
 - attr(*, "spec")=
  .. cols(
  ..   Photo = col_double(),
  ..   Length = col_double(),
  ..   Date = col_character(),
  ..   Time = col_character(),
  ..   Temp = col_double(),
  ..   Tank = col_character(),
  ..   Cage = col_character(),
  ..   Treat = col_character(),
  ..   Line = col_character(),
  ..   Chamber.Respiro = col_double(),
  ..   Last.recorded.weight = col_double(),
  ..   Weight = col_double(),
  ..   `SMR (g/hr)` = col_double(),
  ..   `MMR (g/hr)` = col_double(),
  ..   r2 = col_double(),
  ..   Color = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
colnames(Met)
```

```
 [1] "Photo"               "Length"              "Date"
 [4] "Time"                "Temp"                "Tank"
 [7] "Cage"                "Treat"               "Line"
[10] "Chamber.Respiro"     "Last.recorded.weight" "Weight"
[13] "SMR (g/hr)"          "MMR (g/hr)"          "r2"
[16] "Color"
```

Variables and Data

A variable is simply a container for storing data values.

To assign variables in R we use <- or =

e.g.

```
name<-"Danny"
age<-28
##Then to call a variable we simply type what we saved it as
name
```

```
[1] "Danny"
```

```
age
```

```
[1] 28
```

R variables can be:

- Integer = whole numbers (999, -2)
- Numeric = decimals (0.1, -00.5)
- Character = text (“Hello World”)
- Logical = Booleans (TRUE or FALSE)
- Factor = Categorical (“Large”, “Medium”, “Small”)
- Missing = Logical (NA, NaN)
- Empty = NULL

R data can be:

Vector = 1 dimensional collection of variables of the same type

Matrix= 2 dimensional collection of variables of the same type

Data.frame 2 dimensional collection of variables of multiple types

Dealing with Columns

Looking at these data, the SMR and MMR are labelled strangely, for ease we can change this

```
Met %>%
  rename(SMR = `SMR (g/hr)`)
```

```
# A tibble: 180 × 16
   Photo Length Date     Time       Temp Tank  Cage  Treat Line  Chamber.Resp
iro
   <dbl>  <dbl> <chr>    <chr>     <dbl> <chr> <chr> <chr> <chr>           <d
bl>
 1  6789   24.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
3
 2  6790   22.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
4
 3  6791   21.1 7.8.2021 Evening      34 T1    I5    LS    LS1
1
 4  6770   19.2 6.8.2021 Morning      34 T2    H1    LS    LS1
```

```
1
 5  6772   21.4 6.8.2021 Morning      34 T2    H1    LS    LS1
2
 6  6774   19.4 9.8.2021 Morning      34 T2    H1    LS    LS1
1
 7  6775   22.0 6.8.2021 Morning      34 T2    H1    LS    LS1
3
 8  6696   18.4 9.8.2021 Morning      34 T3    G5    LS    LS1
4
 9  6698   21.2 9.8.2021 Afternoon    34 T3    G5    LS    LS1
1
10  6699   22.0 9.8.2021 Morning      34 T3    G5    LS    LS1
3
# i 170 more rows
# i 6 more variables: Last.recorded.weight <dbl>, Weight <dbl>, SMR <dbl>,
#   `MMR (g/hr)` <dbl>, r2 <dbl>, Color <chr>
```

A few things to note:

We have been introduced to %>% which is the pipe operator in dplyr within tidyverse (in base R it has recently been introduced as |>). The pipe takes the thing on its left and passes it along to the function on its right.

When we rename the parameter on the right is the old name and on the left is the new name.

This will not save a new file as we haven't called a new variable.

To save the changes we should save it to a new data frame

```
Met_trans<-Met %>%
  rename(SMR = `SMR (g/hr)`,
         MMR = `MMR (g/hr)`)
Met_trans
```

```
# A tibble: 180 × 16
   Photo Length Date     Time       Temp Tank  Cage  Treat Line  Chamber.Resp
iro
   <dbl>  <dbl> <chr>    <chr>     <dbl> <chr> <chr> <chr> <chr>           <d
bl>
 1  6789   24.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
3
 2  6790   22.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
4
 3  6791   21.1 7.8.2021 Evening      34 T1    I5    LS    LS1
1
 4  6770   19.2 6.8.2021 Morning      34 T2    H1    LS    LS1
1
 5  6772   21.4 6.8.2021 Morning      34 T2    H1    LS    LS1
2
 6  6774   19.4 9.8.2021 Morning      34 T2    H1    LS    LS1
```

```
1
 7  6775   22.0 6.8.2021 Morning      34 T2    H1    LS    LS1
3
 8  6696   18.4 9.8.2021 Morning      34 T3    G5    LS    LS1
4
 9  6698   21.2 9.8.2021 Afternoon    34 T3    G5    LS    LS1
1
10  6699   22.0 9.8.2021 Morning      34 T3    G5    LS    LS1
3
# ℹ 170 more rows
# ℹ 6 more variables: Last.recorded.weight <dbl>, Weight <dbl>, SMR <dbl>,
#   MMR <dbl>, r2 <dbl>, Color <chr>
```

We have a lot of columns, but what if we only want to focus on some.

We can use `select()` to display only columns we are interested in

```
Met_trans %>%
  select(Temp, Line, SMR, MMR)

# A tibble: 180 × 4
    Temp Line    SMR   MMR
   <dbl> <chr> <dbl> <dbl>
 1    34 LS1   0.205 0.447
 2    34 LS1   0.148 0.909
 3    34 LS1   0.202 0.457
 4    34 LS1   0.238 0.411
 5    34 LS1   0.229 0.428
 6    34 LS1   0.242 0.635
 7    34 LS1   0.103 0.334
 8    34 LS1   0.274 0.536
 9    34 LS1   0.209 0.506
10    34 LS1   0.290 0.360
# ℹ 170 more rows
```

Great, but what if we also want to look at thermal scope (MMR-SMR).

Here is where we use `mutate` to create a new column

```
Met_trans %>%
  mutate(ThermalScope= MMR-SMR)

# A tibble: 180 × 17
   Photo Length Date     Time       Temp Tank  Cage  Treat Line  Chamber.Resp
iro
   <dbl>  <dbl> <chr>    <chr>     <dbl> <chr> <chr> <chr> <chr>           <d
bl>
 1  6789   24.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
3
 2  6790   22.8 7.8.2021 Afternoon    34 T1    I5    LS    LS1
4
```

```
 3  6791   21.1 7.8.2021 Evening      34 T1    I5    LS    LS1
1
 4  6770   19.2 6.8.2021 Morning      34 T2    H1    LS    LS1
1
 5  6772   21.4 6.8.2021 Morning      34 T2    H1    LS    LS1
2
 6  6774   19.4 9.8.2021 Morning      34 T2    H1    LS    LS1
1
 7  6775   22.0 6.8.2021 Morning      34 T2    H1    LS    LS1
3
 8  6696   18.4 9.8.2021 Morning      34 T3    G5    LS    LS1
4
 9  6698   21.2 9.8.2021 Afternoon    34 T3    G5    LS    LS1
1
10  6699   22.0 9.8.2021 Morning      34 T3    G5    LS    LS1
3
# ℹ 170 more rows
# ℹ 7 more variables: Last.recorded.weight <dbl>, Weight <dbl>, SMR <dbl>,
#   MMR <dbl>, r2 <dbl>, Color <chr>, ThermalScope <dbl>
```

The beauty of the pipe is we can easily combine these steps together

```
Met_trans<-Met %>%
  rename(SMR = `SMR (g/hr)`,
         MMR = `MMR (g/hr)`) %>%
  mutate(ThermalScope= MMR-SMR) %>%
  select(Temp, Line, SMR, MMR, ThermalScope)
Met_trans

# A tibble: 180 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# ℹ 170 more rows
```

Dealing with rows

We may want to filter our data to show rows only within a certain range

E.g., lets look at the values of individuals kept at 34ºC

```
Met_trans %>%
  filter(Temp == 34)

# A tibble: 60 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# ℹ 50 more rows
```

Note here that we use == rather than =

As well as ==, we can filter using < (less than), > (greater than), <= (less than or equal to) >= (greater than or equal to), and != (not equal to)

We often need to combine these filters, using & or | (or)

```
##Temp of 34 and Line of LS1
Met_trans %>%
  filter(Temp == 34 & Line =="LS1")

# A tibble: 10 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701

###Show data that is a temperature of 34 or 28
Met_trans %>%
  filter(Temp == 34 | Temp ==28)

# A tibble: 120 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
```

```
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# i 110 more rows
```

There is a shortcut in dplyr for filtering, instead of Temp == 34 | Line ==28, we can use %in%

```
Met_trans %>%
  filter(Temp %in% c(28, 34))

# A tibble: 120 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# i 110 more rows
```

We may also want to order rows by lowest to highest

```
Met_trans %>%
  arrange(SMR)

# A tibble: 180 × 5
    Temp Line     SMR   MMR ThermalScope
   <dbl> <chr>  <dbl> <dbl>        <dbl>
 1    34 SS1   0.0147 0.523        0.508
 2    28 LS2   0.0288 0.146        0.118
 3    28 SS2   0.0442 0.211        0.167
 4    34 RS1   0.0476 0.403        0.355
 5    22 RS2   0.0479 0.524        0.476
 6    22 RS2   0.0503 0.351        0.301
 7    22 SS2   0.0509 0.878        0.827
 8    22 RS2   0.0529 0.526        0.473
 9    22 RS2   0.0555 0.369        0.314
10    34 LS2   0.06   0.558        0.498
# i 170 more rows
```

```
### or highest to lowest
Met_trans %>%
  arrange(desc(SMR))

# A tibble: 180 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    28 RS1   0.532 0.976        0.444
 2    28 SS1   0.487 1.13         0.647
 3    28 SS1   0.487 0.81         0.323
 4    28 RS2   0.458 0.815        0.357
 5    28 RS2   0.457 0.821        0.365
 6    34 RS1   0.445 0.680        0.235
 7    28 RS2   0.438 0.982        0.544
 8    28 SS2   0.431 0.642        0.210
 9    28 SS1   0.407 0.612        0.206
10    28 RS1   0.401 0.820        0.420
# ℹ 170 more rows
```

Another common need is to identify duplicates

```
Met_trans %>%
  distinct()

# A tibble: 180 × 5
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# ℹ 170 more rows
```

Exercises

1. Change the titles of any columns that are not clear, is there a way this could be done quicker?
2. How does Colour and Cage relate to Line and Temperature? How could you filter to identify this?
3. Order the data by R2 value, what could this mean? What about if we want to get the highest per temperature?

Some basic statistics

Using group_by to choose what variables you want values for

```
Met_trans %>%
  group_by(Temp, Line)

# A tibble: 180 × 5
# Groups:   Temp, Line [18]
    Temp Line    SMR   MMR ThermalScope
   <dbl> <chr> <dbl> <dbl>        <dbl>
 1    34 LS1   0.205 0.447       0.241
 2    34 LS1   0.148 0.909       0.761
 3    34 LS1   0.202 0.457       0.255
 4    34 LS1   0.238 0.411       0.173
 5    34 LS1   0.229 0.428       0.198
 6    34 LS1   0.242 0.635       0.394
 7    34 LS1   0.103 0.334       0.231
 8    34 LS1   0.274 0.536       0.262
 9    34 LS1   0.209 0.506       0.297
10    34 LS1   0.290 0.360       0.0701
# ℹ 170 more rows
```

You will notice at the top of the tibble it will now say what the data is grouped by

To get some meaningful statistics we can use `summarise()`. Dplyr will accept both `summarise()` and summarize if you insist on spelling it the incorrect way…

```
Met_trans %>%
  group_by(Temp) %>%
  summarise(
    avg_SMR = mean(SMR)
  )

# A tibble: 3 × 2
   Temp avg_SMR
  <dbl>   <dbl>
1    22      NA
2    28      NA
3    34      NA
```

If we get the mean of SMR per Temp in this way, we just get NA's, this is because there are NA values across the data that R will try and include in the mean calculation. So instead we include na.rm = TRUE to remove those NA values and get a mean.

```
Met_trans %>%
  group_by(Temp) %>%
  summarise(
```

```
    avg_SMR = mean(SMR, na.rm = T)
  )

# A tibble: 3 × 2
   Temp avg_SMR
  <dbl>   <dbl>
1    22   0.188
2    28   0.242
3    34   0.203
```

There are many summarise functions which you can explore with the ? function and all can be combined and printed

```
Met_trans %>%
  group_by(Temp) %>%
  drop_na() %>%
  summarise(
    mean=mean(SMR),
    sd=sd(SMR),
    min=min(SMR),
    max=max(SMR),
    n=n()
  )

# A tibble: 3 × 6
   Temp  mean     sd    min   max     n
  <dbl> <dbl>  <dbl>  <dbl> <dbl> <int>
1    22 0.188 0.0868 0.0479 0.399    59
2    28 0.242 0.114  0.0288 0.532    57
3    34 0.203 0.0893 0.0147 0.445    59
```

We can also use drop_na() as above to remove all NA's preventing the need to rm.na=T for every function

Exercises

1. What is the range for MMR across each selection line?
2. Count the number of distinct values in colour
3. What treatment combination has the highest MMR, can you explain why?

Plotting

A key component of data exploration and analysis, there are options in base R but we are often better utilising the 'grammar of graphics' using ggplot2

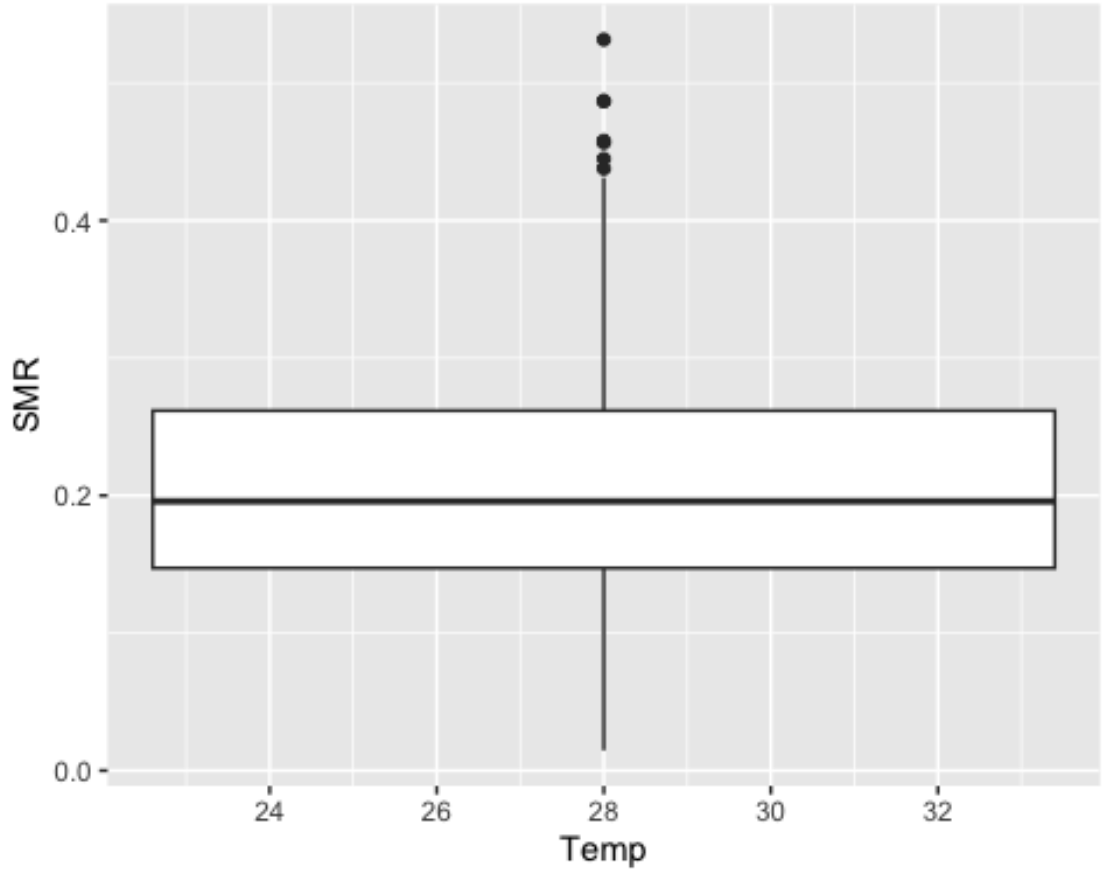
ggplot has a similar structure to dplyr with the key difference being we use + instead of pipe though we can combine both (as shown later)

```
library(ggplot2)
```

```
ggplot(Met_trans, aes(x=Temp, y=SMR))+
  geom_boxplot()
```

```
Warning: Continuous x aesthetic
ℹ did you forget `aes(group = ...)`?
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```

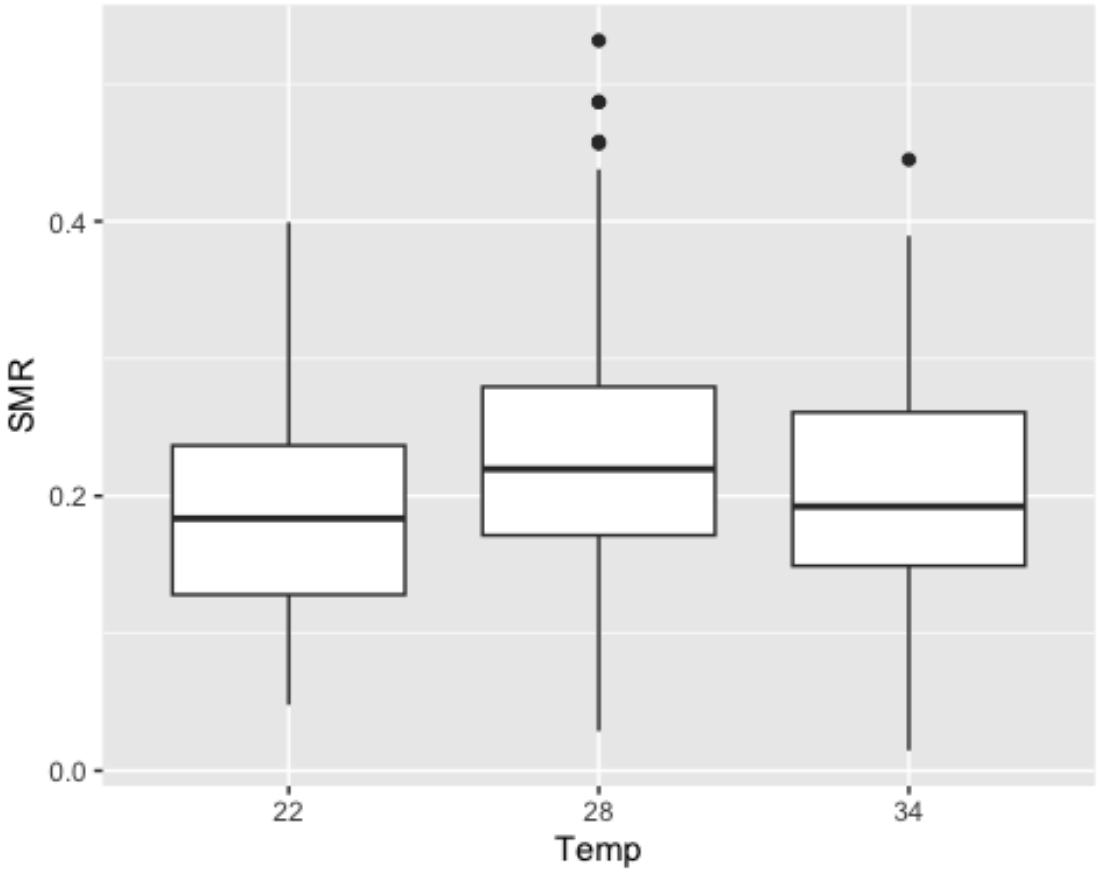


Oops! That´s just one big boxplot!

If we look back into our data structure using `str`, and we can see that Temp is coded as a number, when in reality it is a discrete variable that needs to be changed to a factor

```
Met_trans$Temp<-as.factor(Met_trans$Temp)

ggplot(Met_trans, aes(x=Temp, y=SMR))+
  geom_boxplot()
```

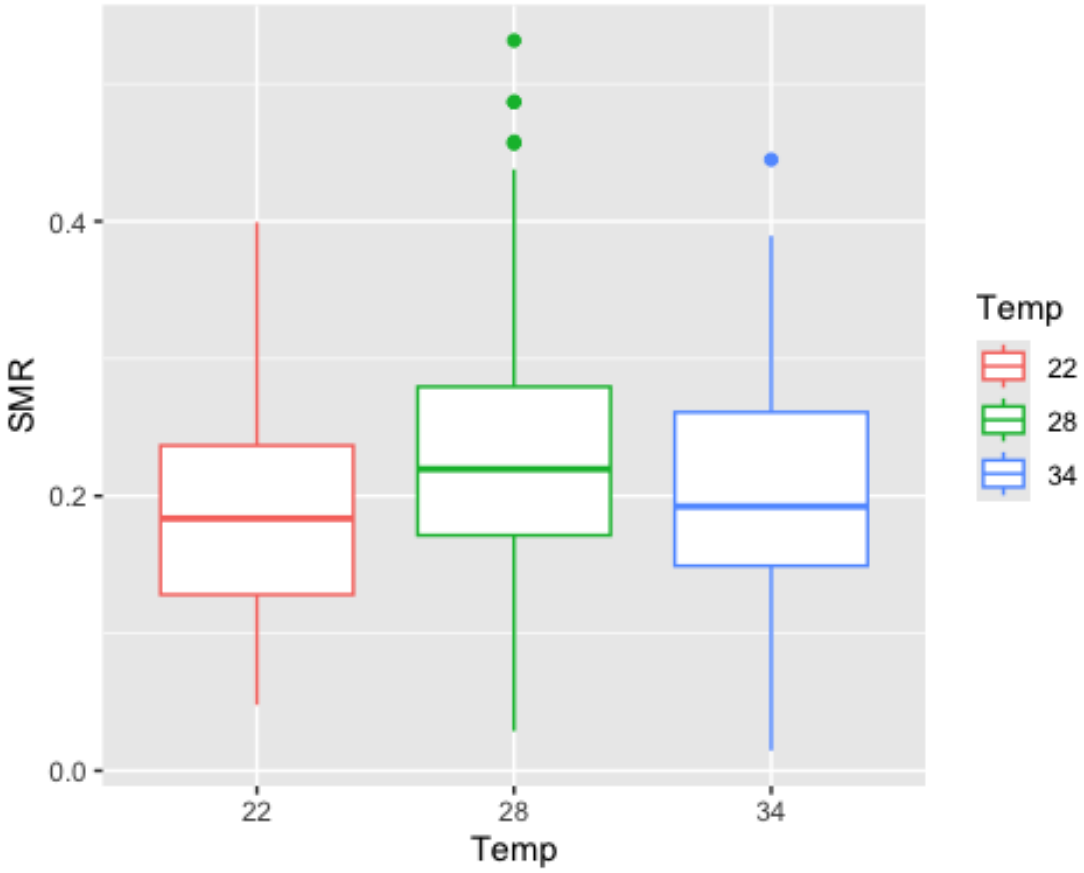
```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



Much better, but lets add a splash of colour, by adding fill= or col= in the aesthetic function (aes)

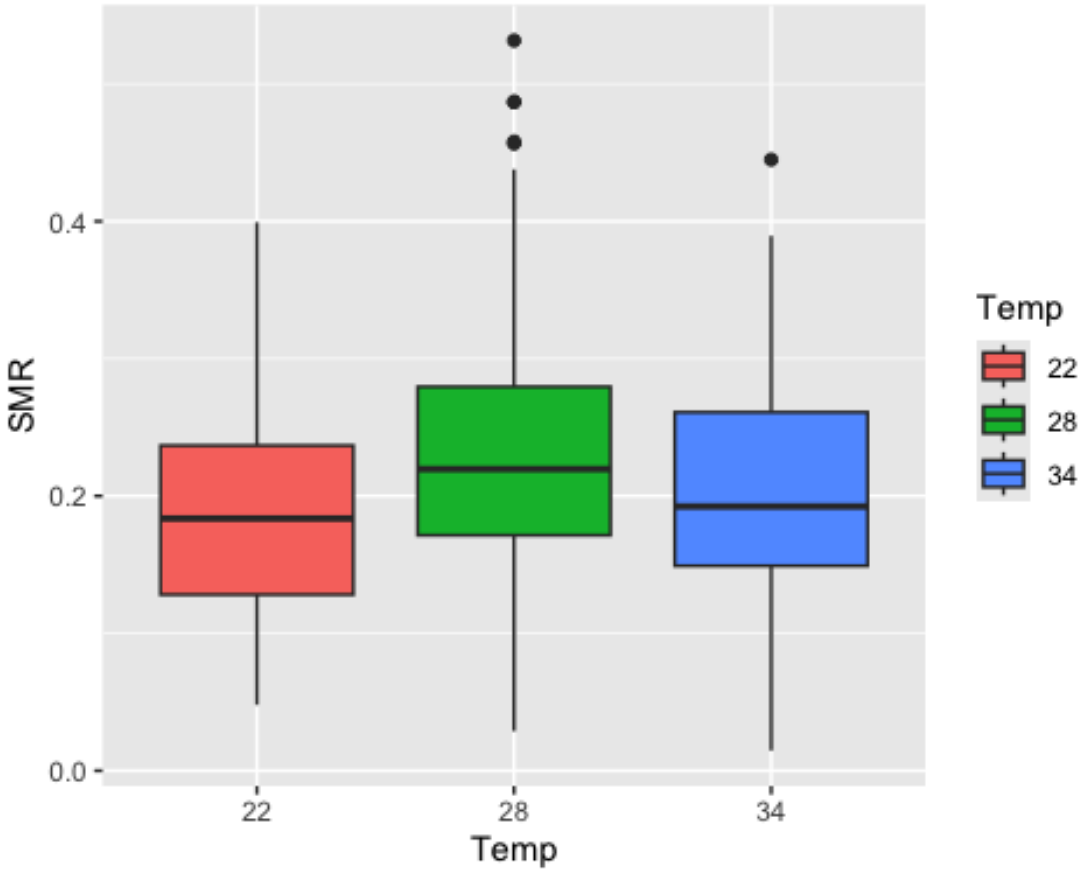
```
ggplot(Met_trans, aes(x=Temp, y=SMR, colour=Temp))+
  geom_boxplot()
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```




```
ggplot(Met_trans, aes(x=Temp, y=SMR, fill=Temp))+
  geom_boxplot()
```

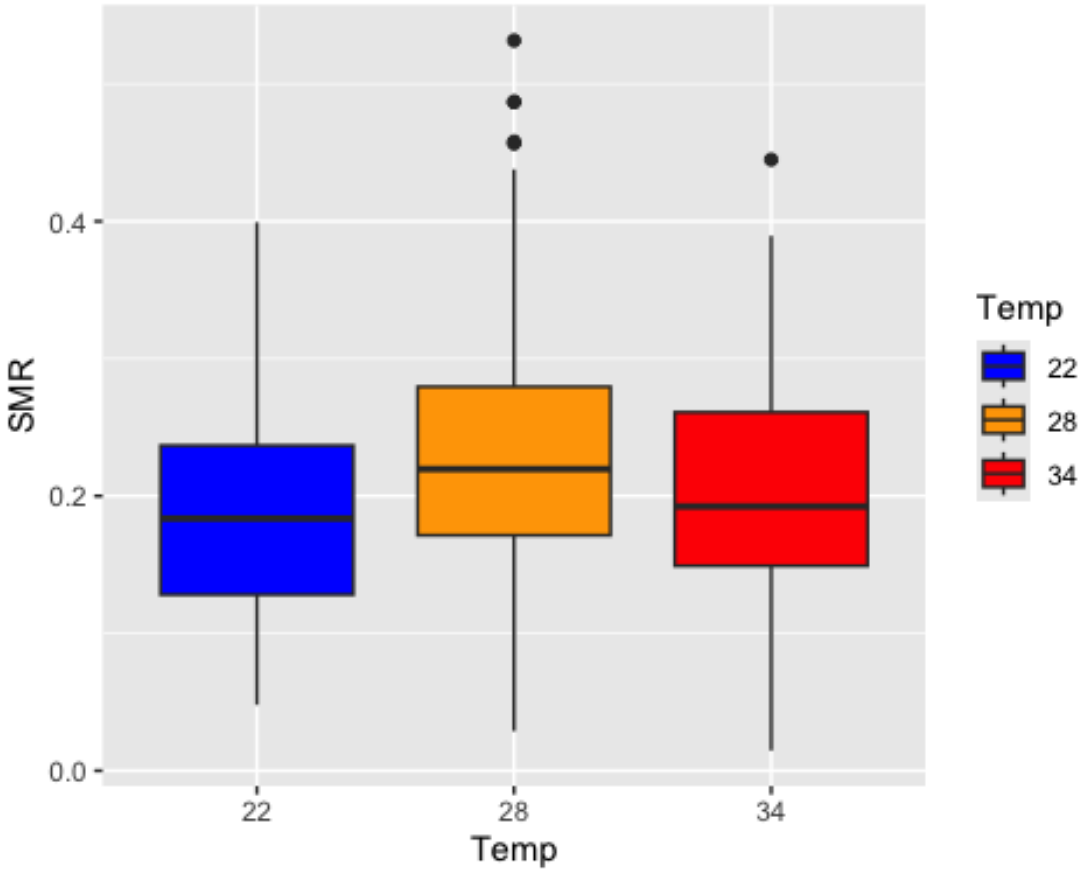
```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



By default ggplot will use their default colour scheme, but we can add our own colours

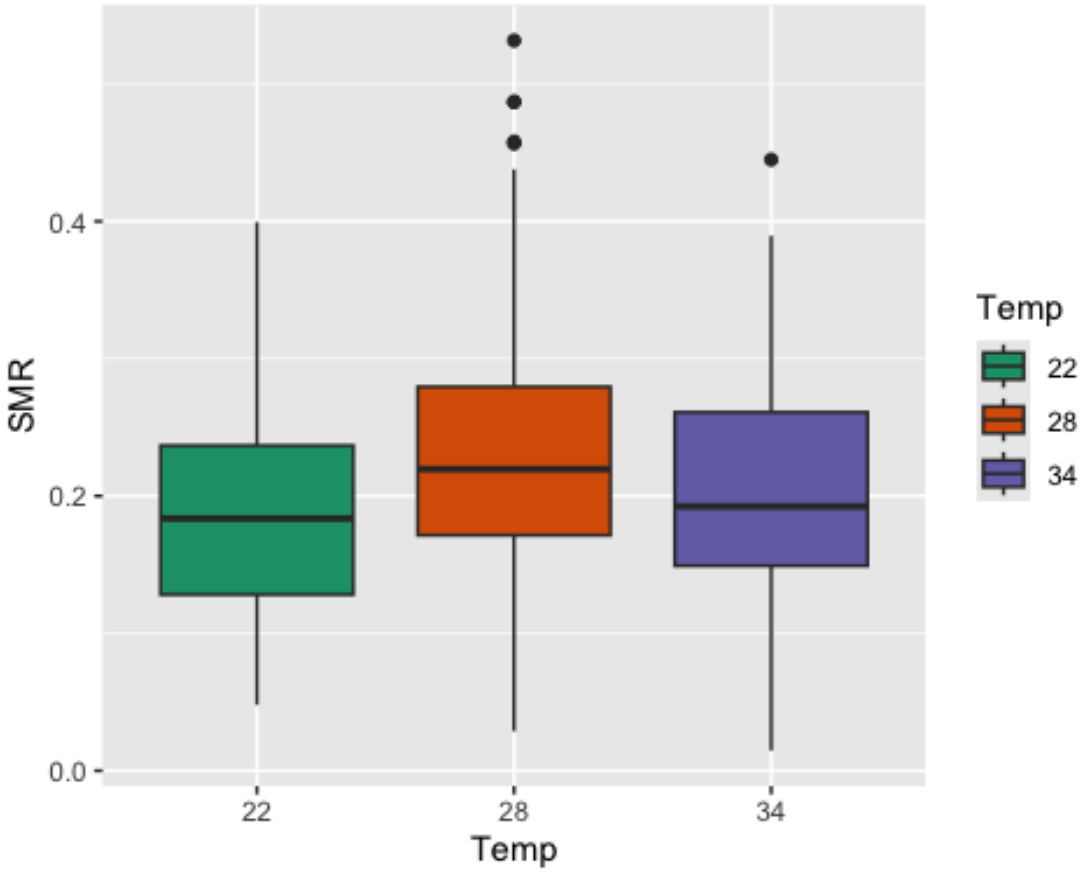
```
ggplot(Met_trans, aes(x=Temp, y=SMR, fill=Temp))+
  geom_boxplot()+
  scale_fill_manual(values=c("blue", "orange", "red"))
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



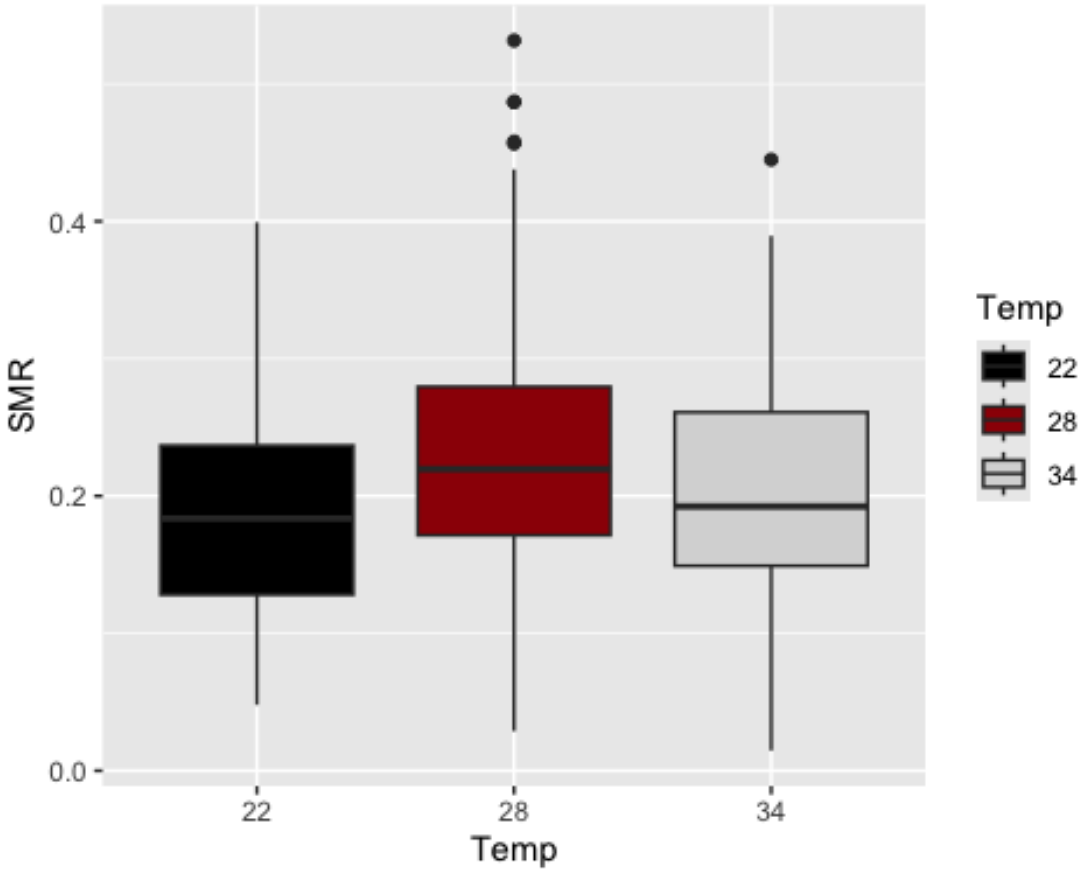

```
##or with brewer palettes

ggplot(Met_trans, aes(x=Temp, y=SMR, fill=Temp))+
  geom_boxplot()+
  scale_fill_brewer(palette = "Dark2")
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```




```
##or some fun library (I like the got colour schemes and use them in most my publications)
library(gameofthrones)

ggplot(Met_trans, aes(x=Temp, y=SMR, fill=Temp))+
  geom_boxplot()+
  scale_fill_got_d(option = "Targaryen")
```

```
Warning: Removed 5 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```

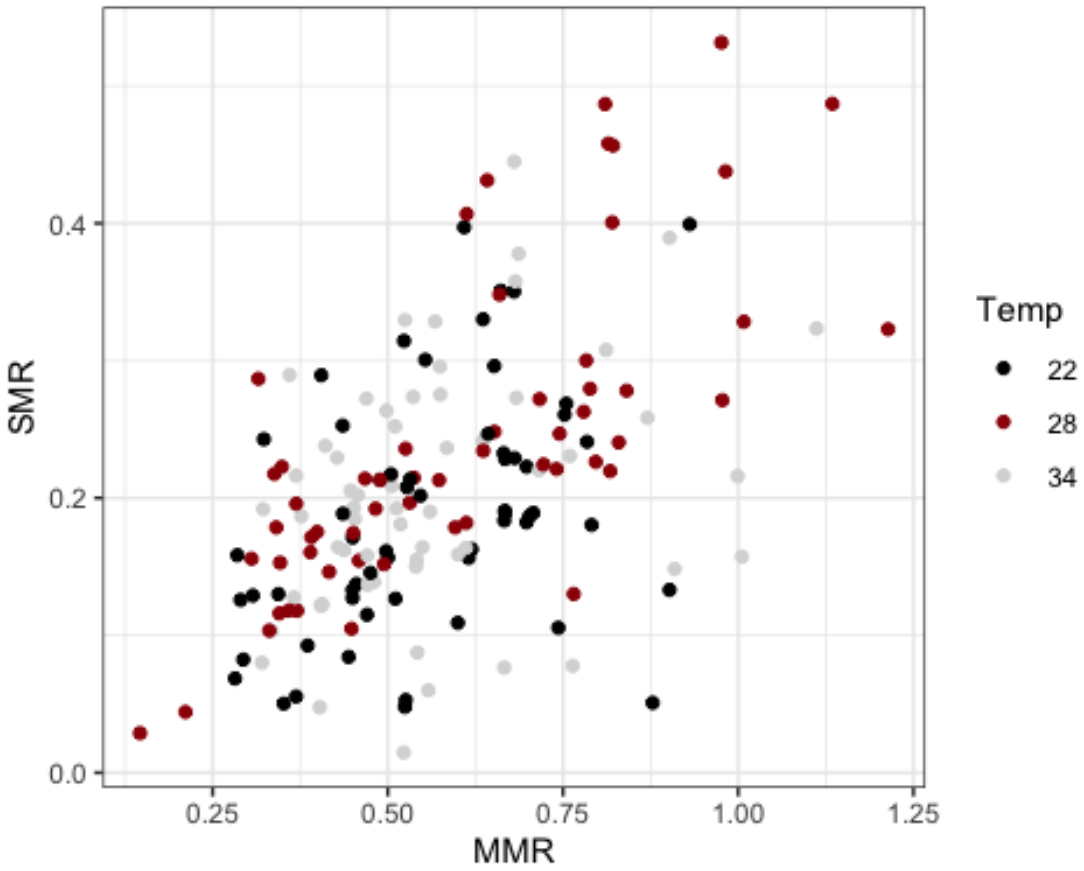


There are many many options for colour palettes out there, whether you want your own colours manually or use precreated colour palettes

There are many different options to visualise your data within the argument 'geom', we have been working with boxplots but we can use `geom_point` to visualise two continuous variables.

```
ggplot(Met_trans, aes(x=MMR, y=SMR, colour=Temp))+
  geom_point()+
  scale_colour_got_d(option = "Targaryen")+
  theme_bw()
```

```
Warning: Removed 5 rows containing missing values or values outside the scale
range
(`geom_point()`).
```

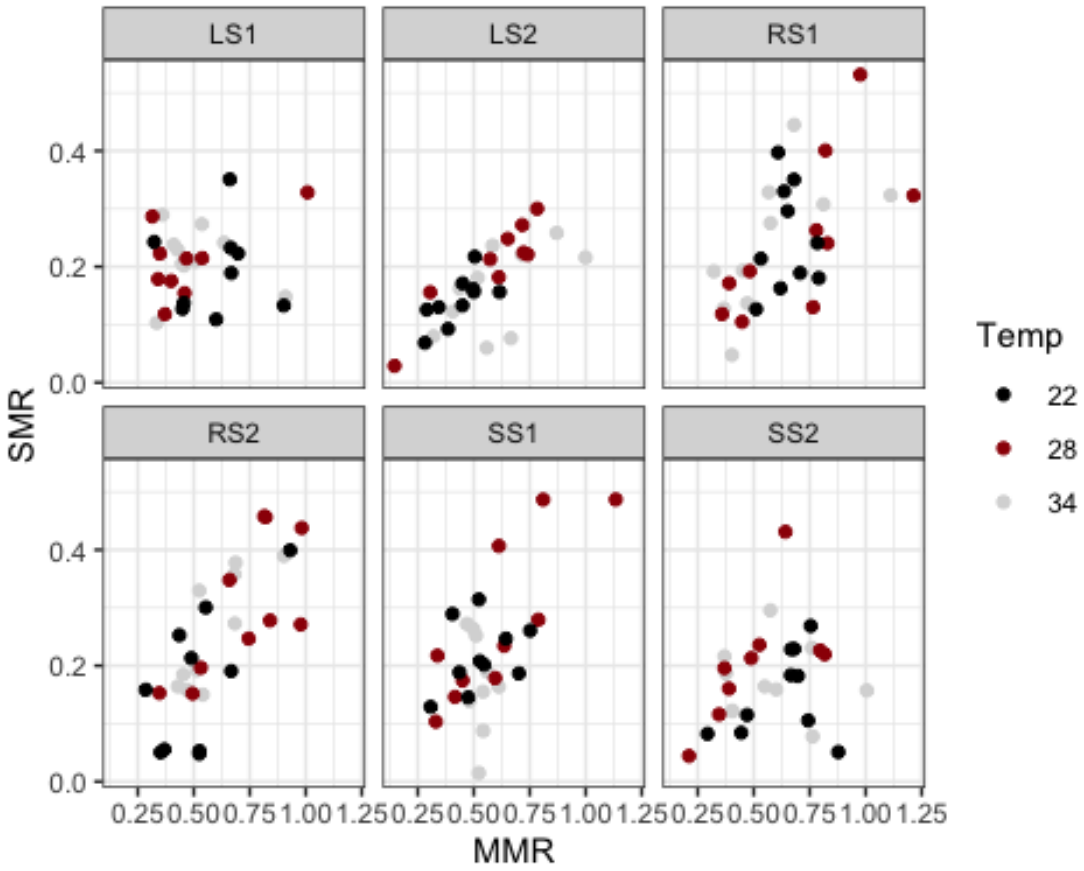


Note the use of theme, this changes the look of the background, there are many options including `theme_bw(), theme_classic(), theme_void(),` have a play around with some! Some packages exist for custom themes too.

But wait what if we want to plot both selection line and temperature together?

```
ggplot(Met_trans, aes(x=MMR, y=SMR, colour=Temp))+
  geom_point()+
  scale_colour_got_d(option = "Targaryen")+
  facet_wrap(~Line)+
  theme_bw()
```

```
Warning: Removed 5 rows containing missing values or values outside the scale
range
(`geom_point()`).
```



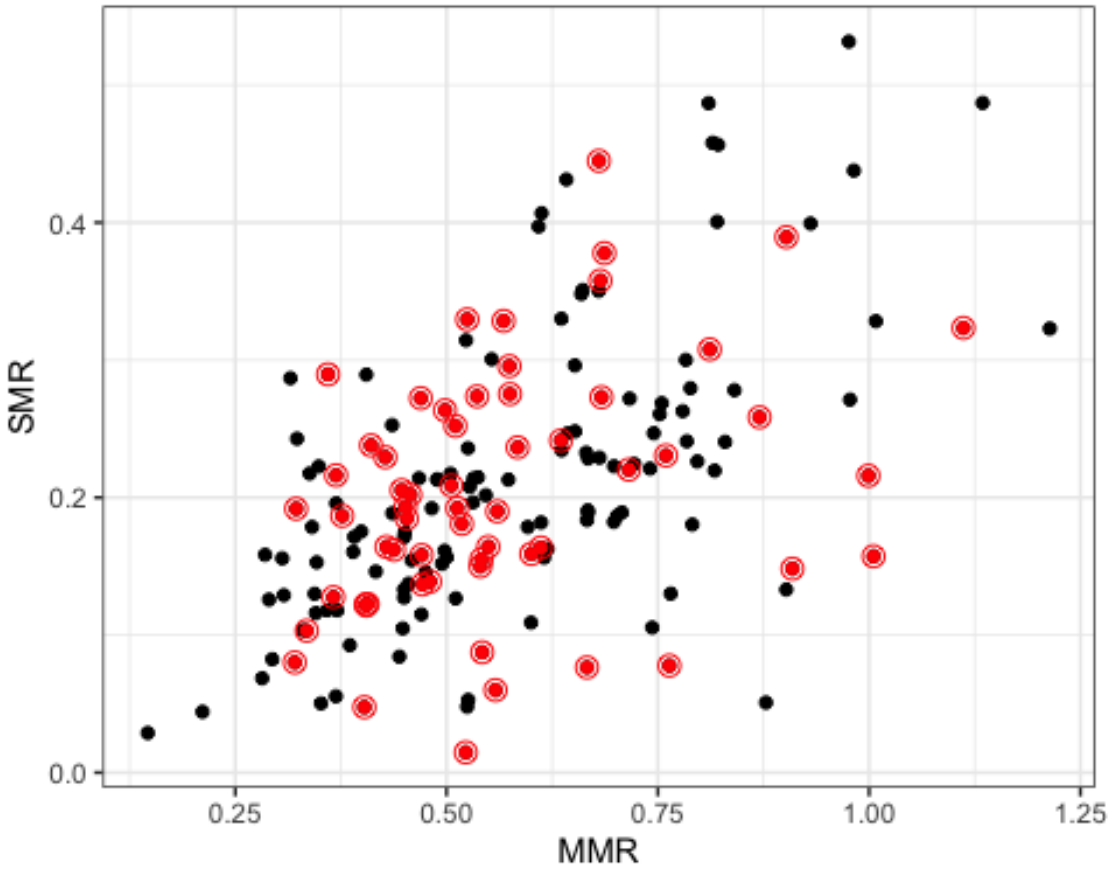
We can use `facet_wrap()` to show multiple panels allowing two factors to easily be displayed

We can also combine our pipe with ggplot, for example we can filter to display specific values

```
ggplot(Met_trans, aes(x = MMR, y = SMR)) +
  geom_point() +
  geom_point(
    data = Met_trans %>%
      filter(Temp == "34"),
    colour = "red"
  ) +
  geom_point(
    data = Met_trans %>%
      filter(Temp == "34"),
    shape = "circle open", size = 3, colour = "red"
  )+
  theme_bw()
```

```
Warning: Removed 5 rows containing missing values or values outside the scale
range
(`geom_point()`).
```

```
Warning: Removed 1 row containing missing values or values outside the scale
range
(`geom_point()`).
Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).
```



Exercises

1. Check through the documentation, how might you change the font and size of the axis text? Can you add a title?
2. Create a violin plot of lines faceted by temperature, using a colour blind friendly palette
3. What plot would you use to show the relationship of weight and MMR based on temperature treatment? Can you add a statistical value?
4. What does geom_smooth do? why might it be useful?
5. How might you visualise the normal distibution of a trait?

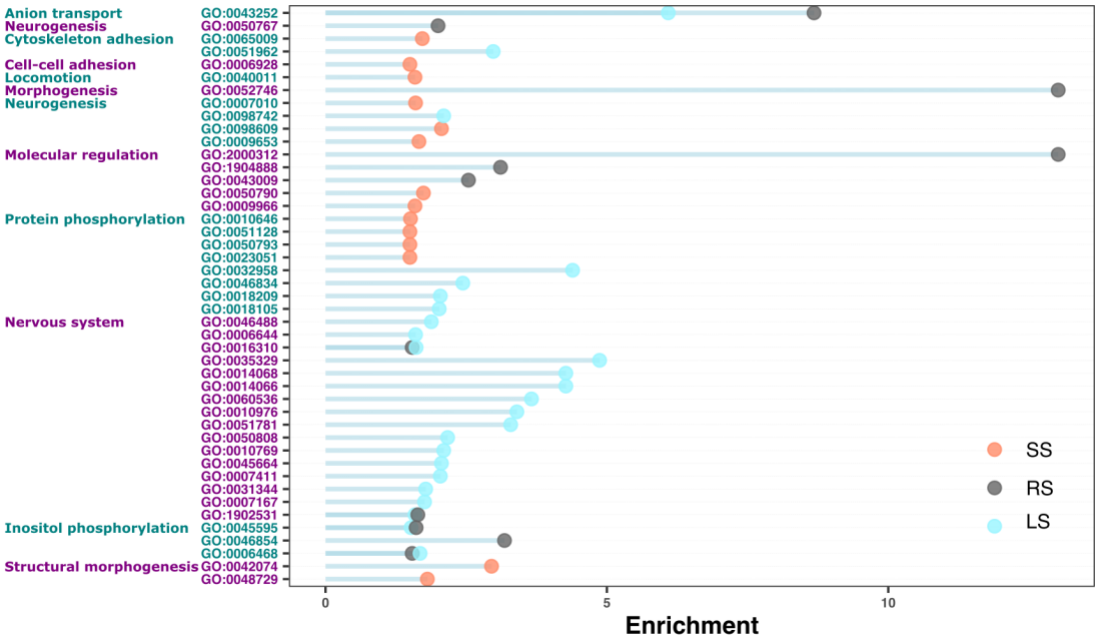
‘

That is the very basics of R! Keep practicing as you delve into more advanced techniques including modelling and bioinformatics.

Now you know how to code several steps before you get an error!

Advanced Exercise - using genomic data

1. I have provided data from a literature review (**https://doi.org/10.3390/fishes8100510**)-
   - The file is in an excel format, how would you import this format? How does it differ from a csv or txt file?
   - Create a subset dataframe of species, and factors you may be interested in, check for NAs and any mistakes
   - How does longevity influence He?
   - Plot He across max length highlighting Order by colour, labelling any outliers and use a colour palette package of your choice!
2. I have provided some GO Term data from **https://doi.org/10.1111/jfb.15901**. How would you recreate the following plot? Provide your code. How would you improve it?!



3. COMPETITION TIME, mastery of plot manipulation is an essential skill for communcating your research, create the best looking plot you can using your choice of data, and also create the worst possible graph you can!