```r
library(dplyr)

rladies_global %>%
  filter(city == 'Auckland')
```

R-Ladies

DIVE INTO DPLYR: Manipulating & exploring your data

Hello!

Welcome to R-Ladies

1.

Introduction

R language, RStudio,
tidyverse

Three things
you’ll need to install

1. **Install R** -- this is the open-source programming language we’ll use (download via CRAN -- Comprehensive R Archive Network)
2. **Install RStudio** -- this is the most popular IDE for R and will make your life a lot easier (download from rstudio.com/download)
3. **Install dplyr** -- this is the package we’ll use within R to work with data. Install with one line of code in R:

```
install.packages(“dplyr”)
```

What is
the tidyverse?

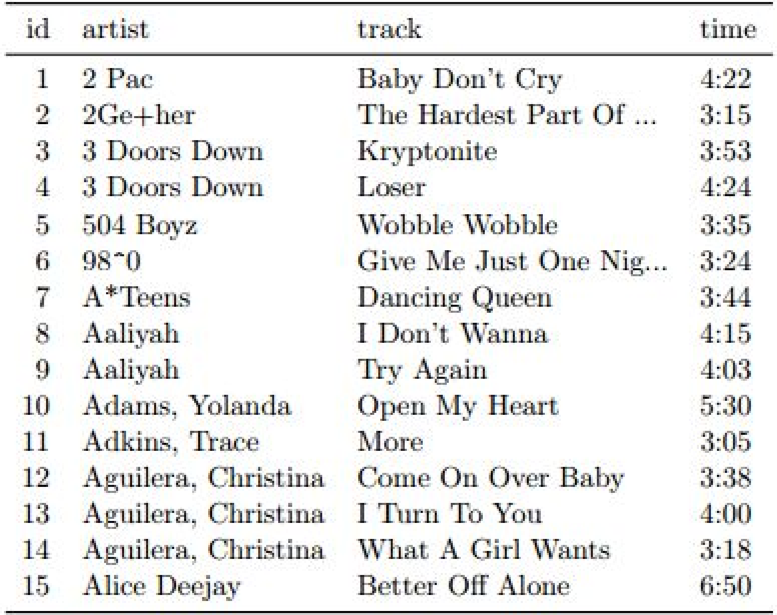
- Collection of R packages based on tidy data principles
- Designed to work together
- An easier way to code!
- AKA “Hadleyverse” (most packages written by Hadley Wickham)
- Dplyr is one of the main packages

What is
the tidyverse?

dplyr
forcats
ggplot2
haven
lubridate
%>%
Ceci n'est pas un pipe.
purrr
readr
readxl
stringr
TIBBLE
tidyr
tidyverse

What is
tidy data?

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table



id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98ˆ0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

2.
dplyr

Let’s get started!

What is
dplyr?

Grammar of data manipulation:

+ `select()` picks variables (columns) based on their names
+ `arrange()` changes the ordering of rows
+ `filter()` allows row selection based on given criteria
+ `mutate()` creates new variables (columns) from existing ones
+ `summarise()` reduces multiple values down to a single summary
+ `group_by()` performs any of the above on a group-by-group basis

dplyr
syntax

+ All calls to dplyr verbs follow the same format:
  1. The first argument is a dataframe
  2. The subsequent arguments describe what to do to that dataframe, using unquoted variable names.

+ Each call returns a new dataframe (rather than overwriting the 'old' one)

+ Example:

```
filter(.data = iris, Species == “setosa”)
```

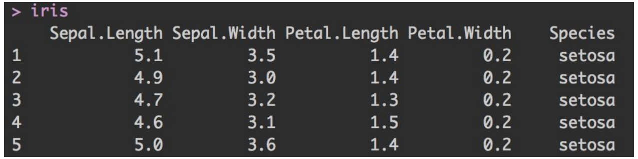
Quick aside:
iris dataset

- Included in R (iris to view)
- 150 observations of 5 variables:
  Iris type, sepal length + width, and petal length + width

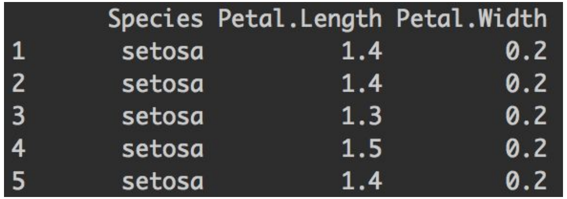
select()

- Picks variables (columns) based on their names
- First argument is dataframe; subsequent arguments represent columns to select

select(.data = iris , Species, Petal.Length, Petal.Width)



```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          5.1         3.5          1.4         0.2    setosa
2          4.9         3.0          1.4         0.2    setosa
3          4.7         3.2          1.3         0.2    setosa
4          4.6         3.1          1.5         0.2    setosa
5          5.0         3.6          1.4         0.2    setosa
```

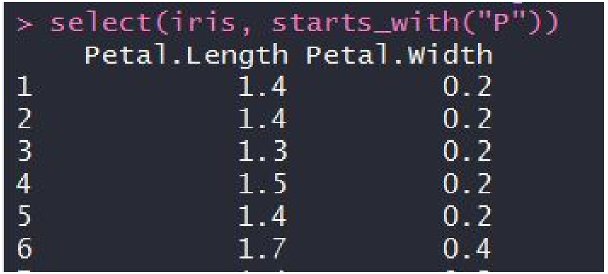


```
  Species Petal.Length Petal.Width
1  setosa          1.4         0.2
2  setosa          1.4         0.2
3  setosa          1.3         0.2
4  setosa          1.5         0.2
5  setosa          1.4         0.2
```

select() +
helper functions

Helper functions you can use within select():

+ starts_with(“a”) matches names that begin with “a”
+ ends_with(“z”) matches names that begin with “z”
+ contains(“lady”) matches names that contain “lady”
+ matches(<regex>) allows you to do regex matching on names



```
> select(iris, starts_with("P"))
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
3          1.3         0.2
4          1.5         0.2
5          1.4         0.2
6          1.7         0.4
```

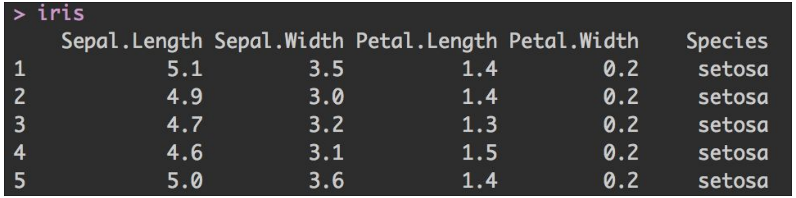
You do it!

- Select the first 3 columns of iris
- Select Petal.Width and Petal.Length
- Select columns with Sepal in them
- Select all columns BUT Species

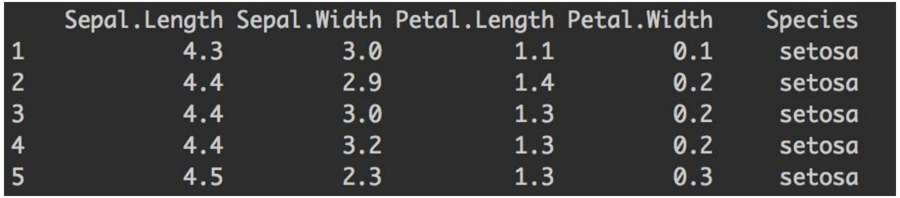
arrange()

- Changes the ordering of rows
- First argument is the dataframe, subsequent arguments are columns and/or expressions used to re-arrange the dataframe
- Note: default is ascending order, and NA’s are always at the end

```
arrange( .data = iris, Sepal.Length, Sepal.Width)
```



```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.1         3.5          1.4         0.2     setosa
2          4.9         3.0          1.4         0.2     setosa
3          4.7         3.2          1.3         0.2     setosa
4          4.6         3.1          1.5         0.2     setosa
5          5.0         3.6          1.4         0.2     setosa
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          4.3         3.0          1.1         0.1     setosa
2          4.4         2.9          1.4         0.2     setosa
3          4.4         3.0          1.3         0.2     setosa
4          4.4         3.2          1.3         0.2     setosa
5          4.5         2.3          1.3         0.3     setosa
```

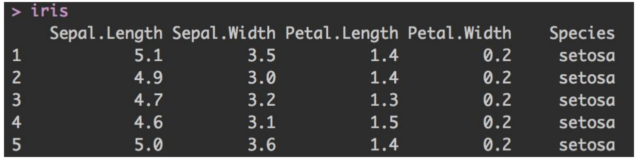
You do it!

- Sort iris by column Petal.Width
- Sort by Petal.Width descending
- Sort by Petal.Width and then Petal.Length

filter()

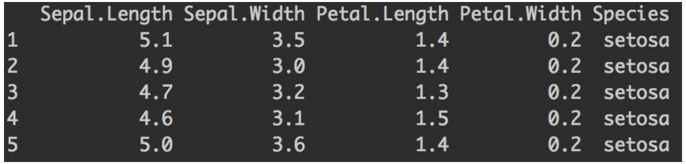
- Allows pointed row selection based on given criteria
- First argument is the dataframe, subsequent arguments are logical expressions used to filter the dataframe

```
filter( .data = iris, Species == “setosa”)
```



```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.1         3.5          1.4         0.2     setosa
2          4.9         3.0          1.4         0.2     setosa
3          4.7         3.2          1.3         0.2     setosa
4          4.6         3.1          1.5         0.2     setosa
5          5.0         3.6          1.4         0.2     setosa
```

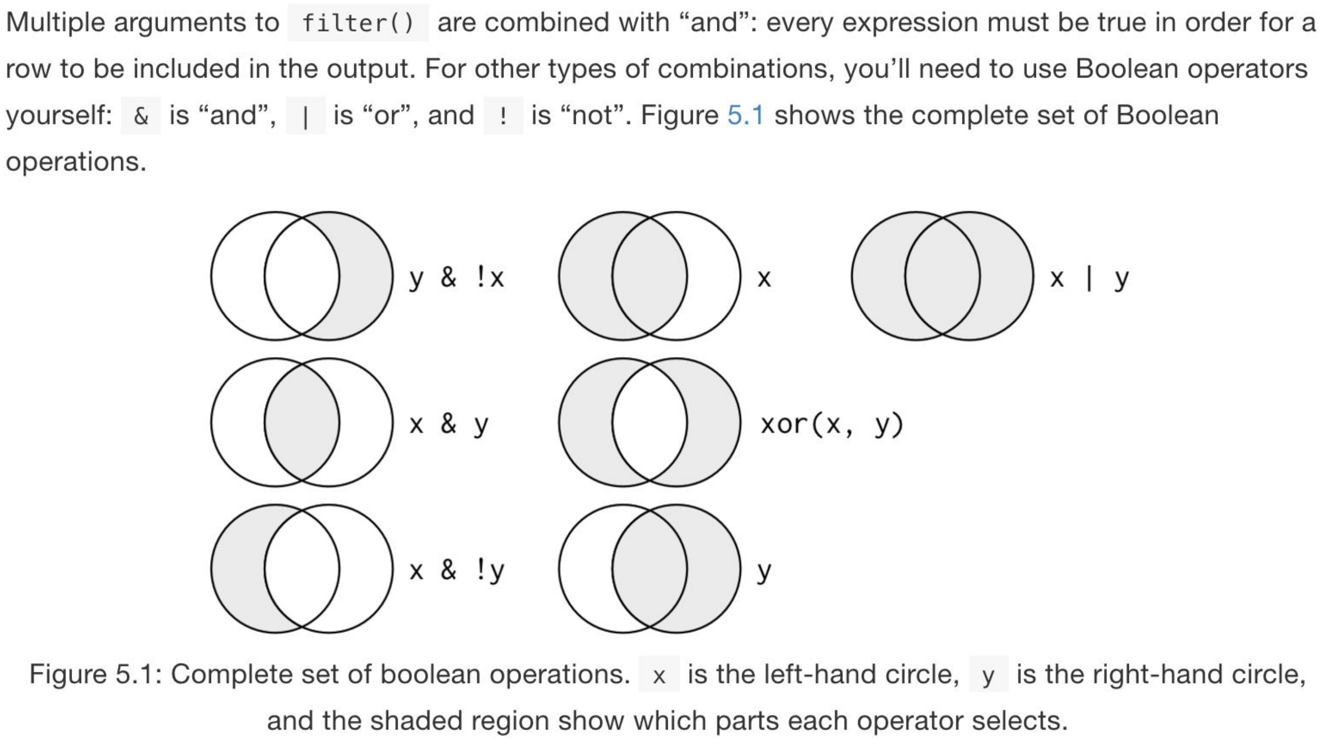
nrow = 150



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

nrow = 50

`filter()` + booleans



Multiple arguments to `filter()` are combined with “and”: every expression must be true in order for a row to be included in the output. For other types of combinations, you’ll need to use Boolean operators yourself: `&` is “and”, `|` is “or”, and `!` is “not”. Figure 5.1 shows the complete set of Boolean operations.

Figure 5.1: Complete set of boolean operations. `x` is the left-hand circle, `y` is the right-hand circle, and the shaded region show which parts each operator selects.

You do it!

- Find the rows where Petal.Width > 1
- Find the rows where Petal.Width > 1 and Species is versicolor
- Find the rows where Species is not setosa

3.
magrittr

What?!?

What is
magrittr?

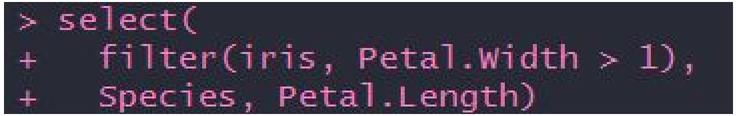
Simplifying R code with pipes (%>%)

- Easy way to pass data through functions without nesting
- First argument of each function is “piped” in to reduce redundancy
- f(x) is the same as x %>% f()
- f(x, y) is the same as x %>% f(y)

https://en.wikipedia.org/wiki/The_Treachery_of_Images

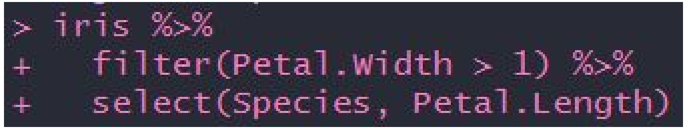
dplyr + magrittr
example

Before



```
> select(
+   filter(iris, Petal.Width > 1),
+   Species, Petal.Length)
```

After



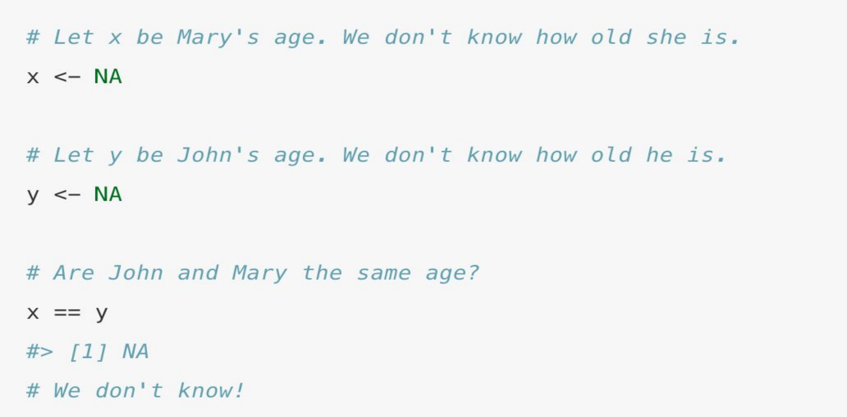
```
> iris %>%
+   filter(Petal.Width > 1) %>%
+   select(Species, Petal.Length)
```

You do it!

- Find the rows where Petal.Width > 1 and select the first 3 variables
- Select the columns that contain Petal and sort by Petal.Width and then only the first 6 rows
- Find the rows for Species setosa and order rows by descending Sepal.Width
- Select the rows where Petal.Width > 1 and view in the panel
- Find the unique values of Petal.Width for the setosa Species

Quick aside:
Missing values

- NA represents a missing (unknown) value
- Comparisons involve unknown values typically result in unknown values
- To see whether a value is missing, use **`is.na()`**
- **`filter()`** only includes rows where the condition is true (not false or NA)

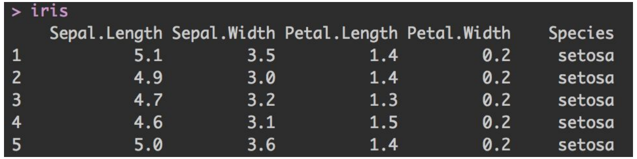


```
# Let x be Mary's age. We don't know how old she is.
x <- NA

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

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

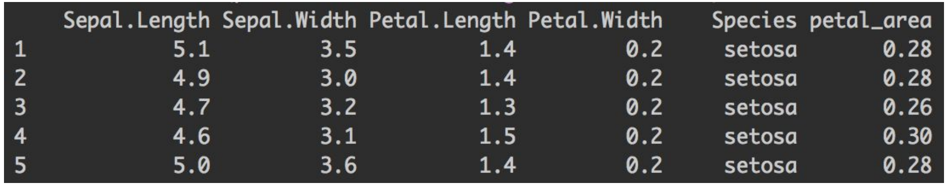
mutate()

- Creates new variables (columns) from existing ones
- Note: columns created with `mutate()` are always added to end of dataset

```
iris %>% mutate( petal_area = Petal.Length * Petal.Width)
```



```
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species petal_area
1          5.1         3.5          1.4         0.2     setosa       0.28
2          4.9         3.0          1.4         0.2     setosa       0.28
3          4.7         3.2          1.3         0.2     setosa       0.26
4          4.6         3.1          1.5         0.2     setosa       0.30
5          5.0         3.6          1.4         0.2     setosa       0.28
```

`mutate()`

Useful functions

- Arithmetic operators (+, -, *, /, ^)
- Log functions (like `log10()`)
- Offsets like `lead()` and `lag()`
- Logical comparisons (<, <=, >, >=, !=)
- `ifelse` statements (if this, then this, else this)
- Or when more than 1 logical split then use `case_when`
- Cumulative and rolling aggregates
- Ranking (like `ntile()`)

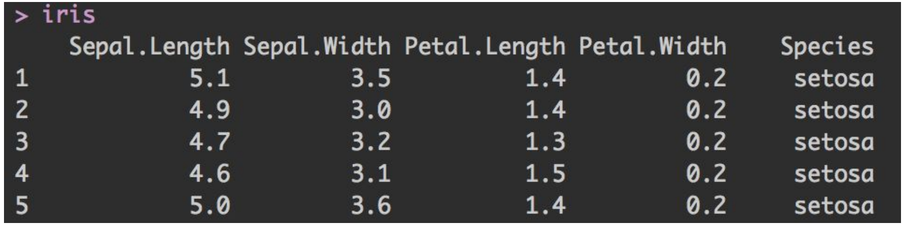
You do it!

- Create a new column with a flag showing Petal.Width > Petal.Length
- Create a new column with Petal.Width + Sepal.Length
- Create a new column with the mean of Sepal.Length and then another new column with a flag stating if each row’s Sepal.Length is greater than the mean
- Create a new column with Sepal.Length buckets (use cut)
- Create a new column stating Low when Petal.Width < 0.2, Medium when Petal.Width is between 0.2 and 0.6 and High when Petal.width > 0.6 (use case_when)

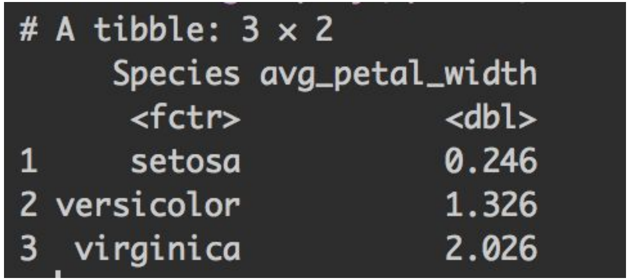
group_by() and summarise()

- group_by applies dplyr verbs by group
- summarise reduces multiple values down to a single summary

```
iris %>%
    group_by(Species) %>%
    summarise(avg_petal_width = mean(Petal.Width))
```



```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.1         3.5          1.4         0.2     setosa
2          4.9         3.0          1.4         0.2     setosa
3          4.7         3.2          1.3         0.2     setosa
4          4.6         3.1          1.5         0.2     setosa
5          5.0         3.6          1.4         0.2     setosa
```



```
# A tibble: 3 × 2
     Species avg_petal_width
      <fctr>           <dbl>
1     setosa           0.246
2 versicolor           1.326
3  virginica           2.026
```

`summarise()`

Useful functions

- Counts (`n()`, `n_distinct()`)
- Measures of location (`mean()`, `median()`)
- Measures of spread (`sd()`, `IQR()`)
- Measures of position (`first()`, `last()`)

You do it!

- Find the mean of the Petal.Length and the sd of Petal.Width
- Create a new column with the mean of Sepal.Length per species
- Create a table with the mean Sepal.Length per species
- Find how many observations there are for each species and Petal.Length combination

Tips & Tricks

- If you don't have the result of a dplyr chain to a dataframe, it will print
- Include arguments for inner functions like na.rm = TRUE
- `rename()` is a cool function to clean up messy column names
- `slice()` will give you rows by row number
- After grouping with `group_by()`, you can `ungroup()` to remove groupings
- There is a cheat sheet for data wrangling!
- If you want to print and save, wrap assignment in parenthesis
  - Example: `(iris_names <- iris %>% filter(Species == “setosa”))`

4.

Scoped verbs

More dplyr goodness . . .

Scoped Verbs

- Terminology: we have been using “single table verbs”
- Now we can affect multiple variables simultaneously with the scoped verbs
- Three extensions
  - `_if` pick variables based on a predicate function like is.numeric()
  - `_at` pick variables using the same syntax as select().
  - `_all` operates on all variables

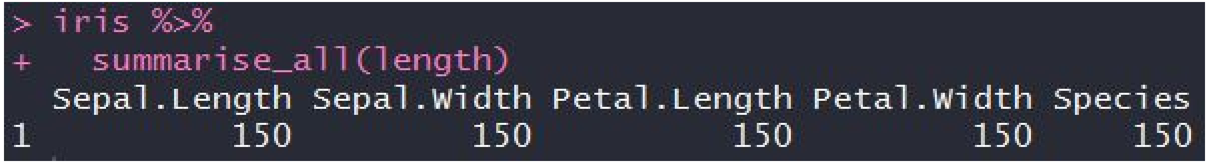
`summarise_all()`

Scoped verbs

- apply the function to all variables

```
summarise_all( .tbl, function)
```

- Multiple functions `summarise_all( .tbl, funs(f1, f2))`
- Combine functions: `summarise_all( .tbl, ~f1(f2(.)))`



```
> iris %>%
+   summarise_all(length)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          150         150          150         150     150
```

You do it!

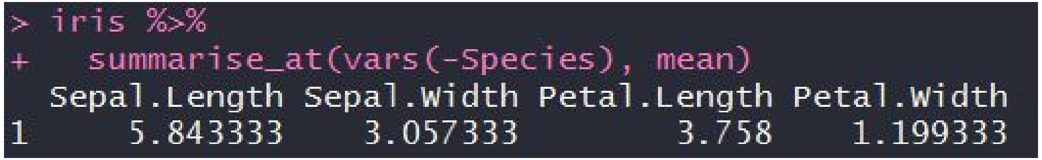
- Find the mean of the first 4 variables
- Find the mean and min of the first 4 variables
- Find how many unique observations there are for each of the variables

summarise_at()

Scoped verbs

- apply the function to variables using their names like select

```
summarise_at( .tbl, vars(...), function)
```



```
> iris %>%
+   summarise_at(vars(-Species), mean)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.843333    3.057333        3.758    1.199333
```

You do it!

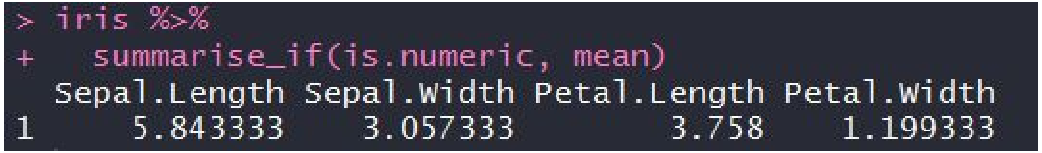
- Find the mean of the first 4 variables
- Find the mean and min of the variables with Petal in their names

summarise_if()

Scoped verbs

- apply the function to variables using logical conditions based on some property of the column

```
summarise_if( .tbl, condition, function)
```



```
> iris %>%
+   summarise_if(is.numeric, mean)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.843333    3.057333        3.758    1.199333
```

You do it!

- Find the sd of the numerical variables
- Find the number of unique observations of all the categorical variables

mutate & filter

My frequent examples

- `mutate_if( .tbl, is.character, as.factor)`

- `filter_all()` used with `all_vars()` or `any_vars()`
- `filter_all(data, any_vars(is.na(.)))`

You do it!

- Turn numerical variables into characters
- Round the Petal variable to 0dp
- Find the rows of the airquality dataset where there is at least 1 NA value

5.
Wrap-up

Announcements, upcoming events, etc.

R-Ladies Auckland
Upcoming Events

Lessons and highlights from the rOpenSci project [Feb 20]

useR! [Brizzie Jul 10-13]

Looking for presenters!

BIG THANKS TO

R-Ladies Austin

https://github.com/rladiesaustin/R4DS_workshop_series

Hadley Wickham vignettes and Stanford material

https://dcl-2017-04.github.io/curriculum/manip-scoped.html