Data manipulation in R (using *dplyr*)

Outline

- Describe the purpose of the dplyr and tidyr packages.
- Select certain columns in a data frame with the dplyr function select.
- Select certain rows in a data frame according to filtering conditions with the dplyr function filter .
- Link the output of one dplyr function to the input of another function with the ‘pipe’ operator %>%.
- Add new columns to a data frame that are functions of existing columns with mutate.

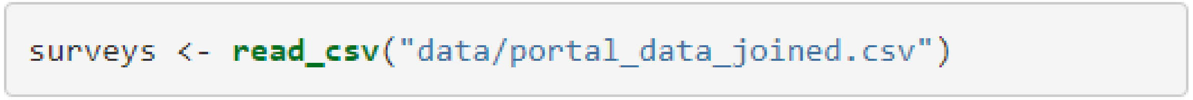
- Bracket subsetting is handy, but it can be cumbersome and difficult to read, especially for complicated operations.
- dplyr is a package for making tabular data manipulation easier.
- It pairs nicely with tidyr which enables you to swiftly convert between different data formats for plotting and analysis.
- This is not the only way to do these things.

Different dataset ‘shapes’

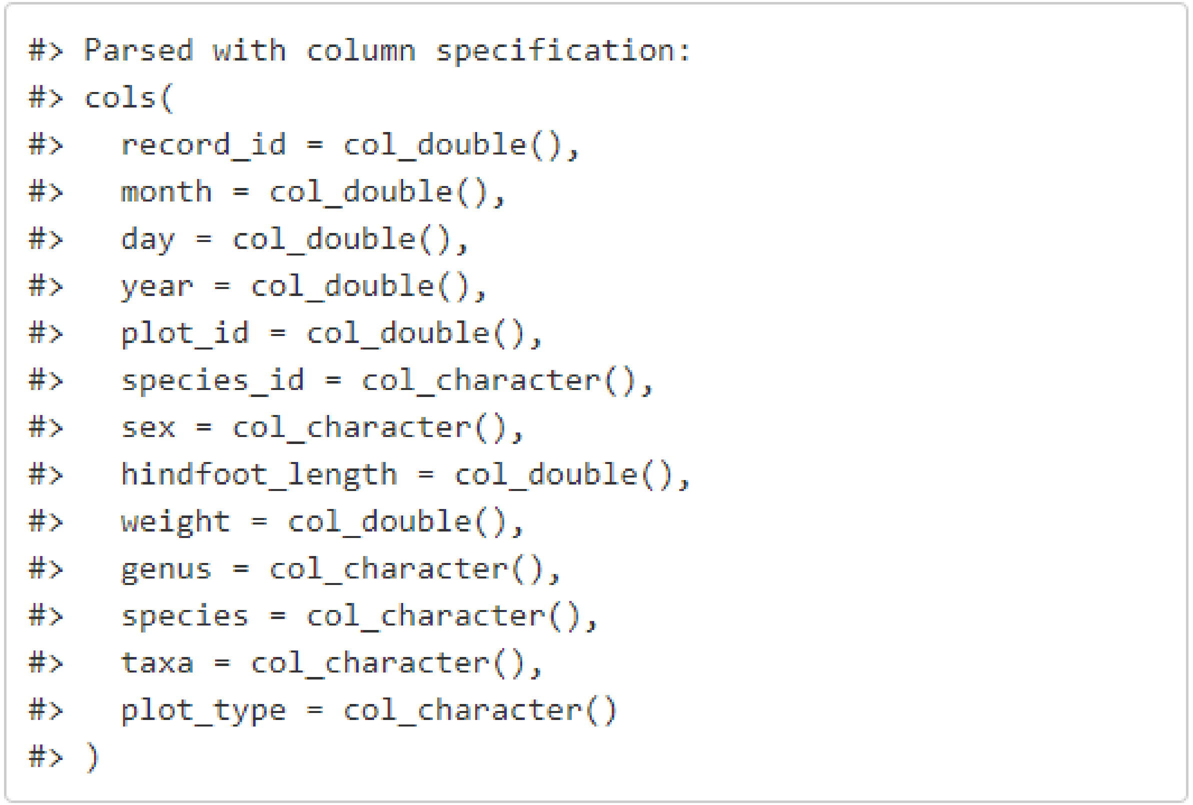
- The package tidyr addresses the common problem of wanting to reshape your data for plotting and use by different R functions
- Sometimes we want data sets where we have one row per measurement. Sometimes we want a data frame where each measurement type has its own column, and rows are instead more aggregated groups.
- Moving back and forth between these formats is nontrivial, and tidyr gives you tools for this and more sophisticated data manipulation.

Reading-in data

- Instead of the base R read.csv (with a dot), we’ll use the read_csv (with underscore)
- Eg



```
surveys <- read_csv("data/portal_data_joined.csv")
```



```
#> Parsed with column specification:
#> cols(
#>   record_id = col_double(),
#>   month = col_double(),
#>   day = col_double(),
#>   year = col_double(),
#>   plot_id = col_double(),
#>   species_id = col_character(),
#>   sex = col_character(),
#>   hindfoot_length = col_double(),
#>   weight = col_double(),
#>   genus = col_character(),
#>   species = col_character(),
#>   taxa = col_character(),
#>   plot_type = col_character()
#> )
```

Tibbles

- Tibbles tweak some of the behaviours of the data frame objects we introduced in the previous episode.
- The data structure is very similar to a data frame. For our purposes the only differences are that:

1. In addition to displaying the data type of each column under its name, it only prints the first few rows of data and only as many columns as fit on one screen.
2. Columns of class character are never converted into factors.

Selecting columns

- In base R we can select column by integer index or name
- In dplyr, to select columns of a data frame, use select().
- The first argument to this function is the data frame, and the subsequent arguments are the columns to keep.

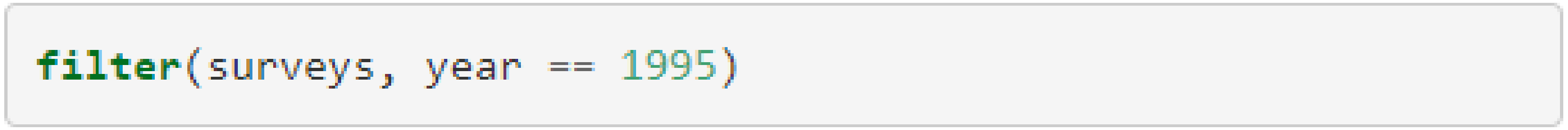
```r
select(surveys, plot_id, species_id, weight)
```

- To select all *except* certain ones, use ‘-’

```r
select(surveys, -record_id, -species_id)
```

Selecting rows

- In base R, we select rows by index
- In dplyr, we choose a row by specific criteria using filter
- Eg



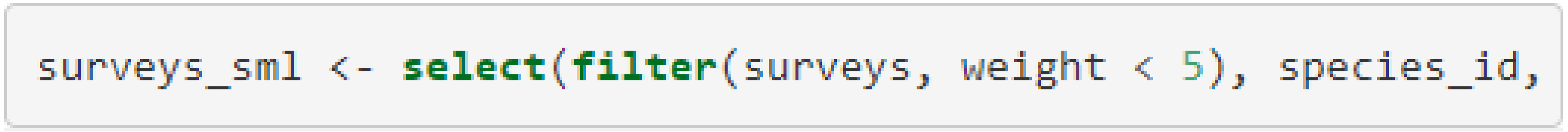
```
filter(surveys, year == 1995)
```

Pipes

- What if you want to select and filter at the same time?
- There are three ways to do this:
  - use intermediate steps
  - nested functions
  - pipes
- With intermediate steps, you create a temporary data frame and use that as input to the next function, like this:

```r
surveys2 <- filter(surveys, weight < 5)
surveys_sml <- select(surveys2, species_id, sex, weight)
```

- Nested functions



```
surveys_sml <- select(filter(surveys, weight < 5), species_id,
```

- Pipes let you take the output of one function and send it directly to the next
- Useful when you need to do many things to the same dataset
- Pipes in R look like %>% and are made available via the magrittr package, installed automatically with dplyr.
- Keyboard shortcut is

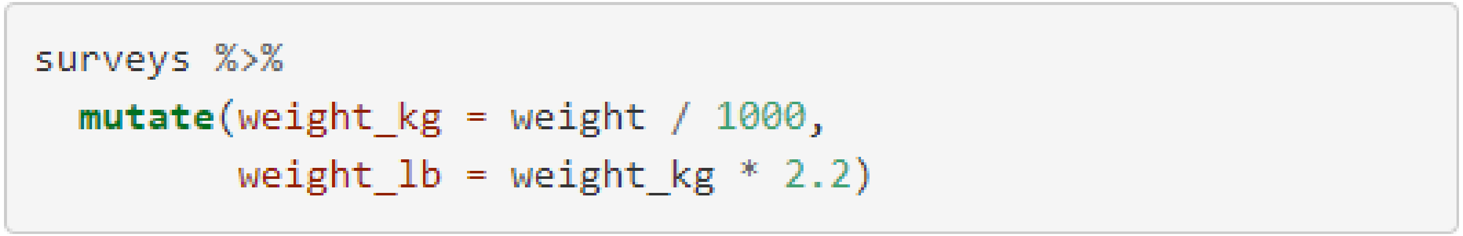
`Ctrl` + `Shift` + `M`

- Eg

```
surveys %>%
  filter(weight < 5) %>%
  select(species_id, sex, weight)
```

Mutate

- Frequently you’ll want to create new columns based on the values in existing columns, for example to do unit conversions, or to find the ratio of values in two columns
- In base R we could use cbind() or data.frame()
- In dplyr we’ll use mutate()
- E.g.

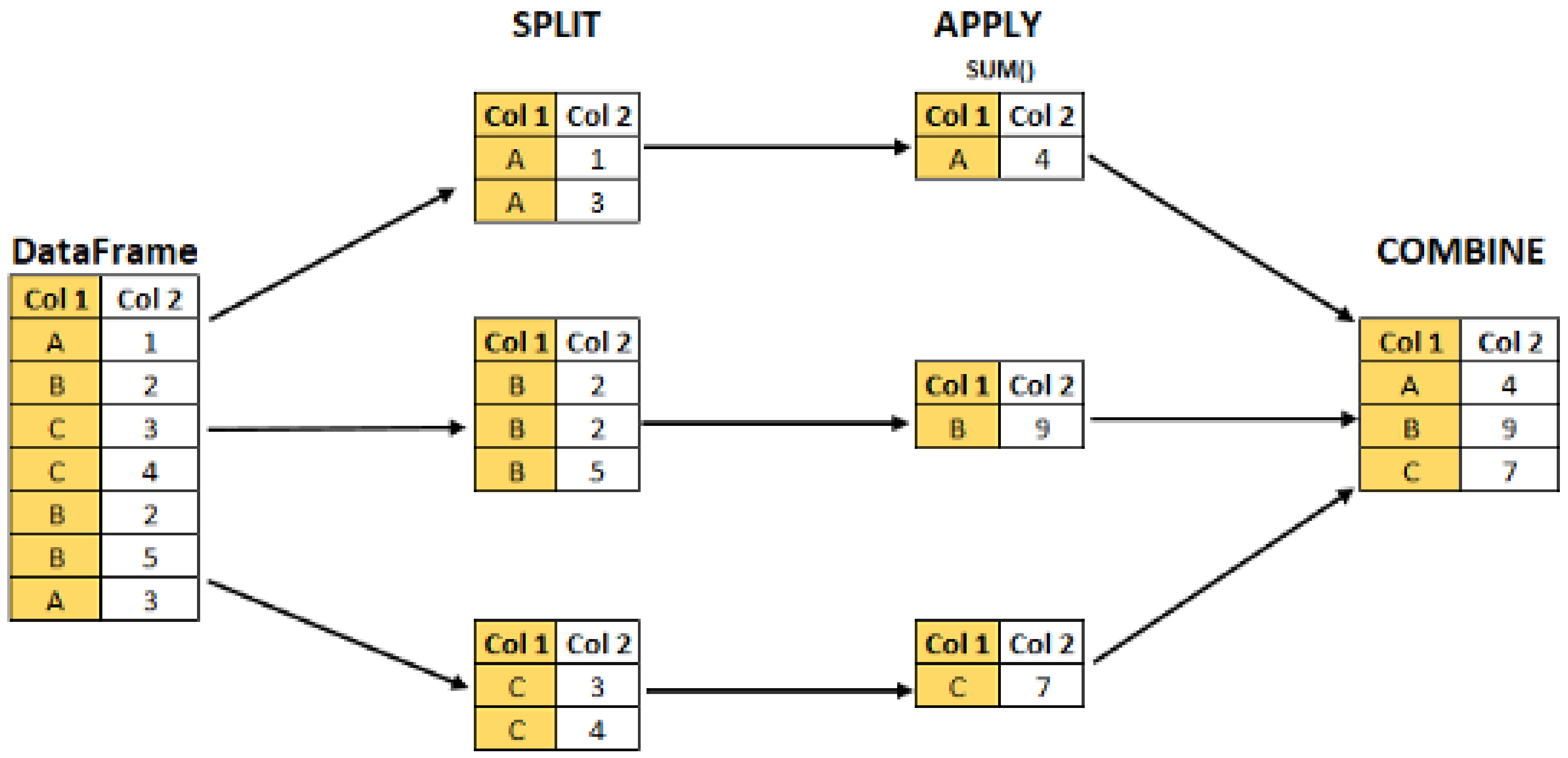


```
surveys %>%
  mutate(weight_kg = weight / 1000,
         weight_lb = weight_kg * 2.2)
```

Split-apply-combine data analysis and the summarize() function

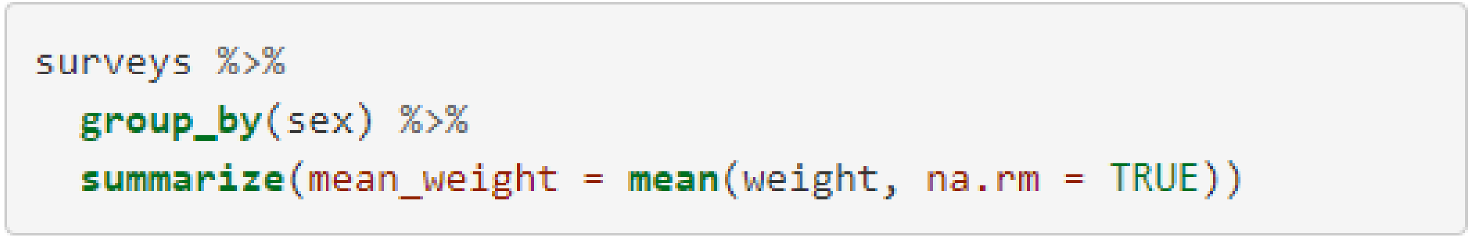
- Strategy in which

1. Split: Split the data into groups based on some criteria thereby creating a GroupBy object. (We can use the column or a combination of columns to split the data into groups)
2. Apply: Apply a function to each group independently. (Aggregate, Transform, or Filter the data in this step)
3. Combine: Combine the results into a data structure


DataFrame
Col 1 | Col 2
A 1
B 2
C 3
C 4
B 2
B 5
A 3
SPLIT
Col 1 | Col 2
A 1
A 3
Col 1 | Col 2
B 2
B 2
B 5
Col 1 | Col 2
C 3
C 4
APPLY
SUM()
Col 1 | Col 2
A 4
Col 1 | Col 2
B 9
Col 1 | Col 2
C 7
COMBINE
Col 1 | Col 2
A 4
B 9
C 7

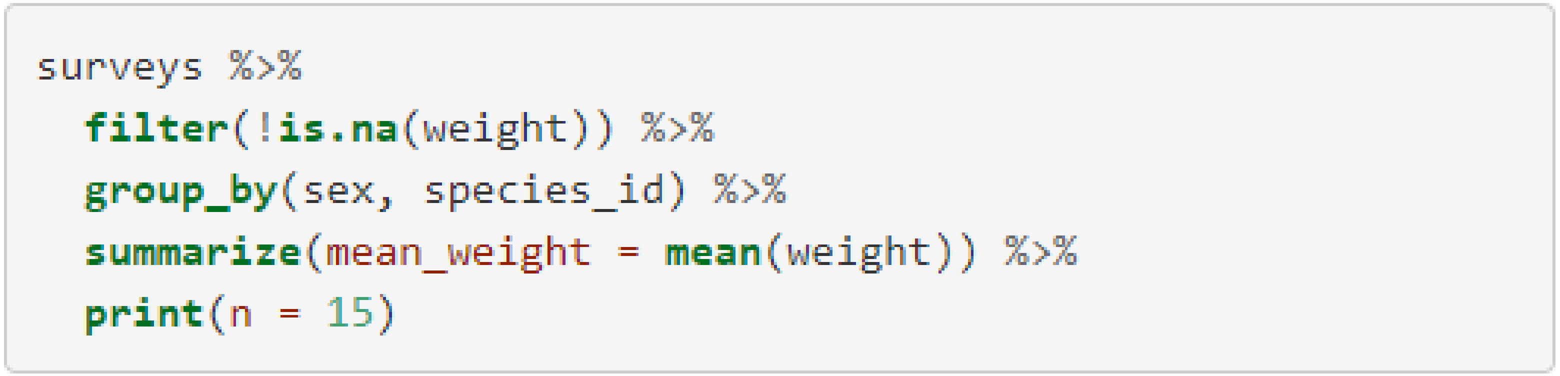
Summarise function

- group_by() is often used together with summarize(), which collapses each group into a single-row summary of that group.
- group_by() takes as arguments the column names that contain the categorical variables for which you want to calculate the summary statistics.
- So to compute the mean weight by sex:



```
surveys %>%
  group_by(sex) %>%
  summarize(mean_weight = mean(weight, na.rm = TRUE))
```

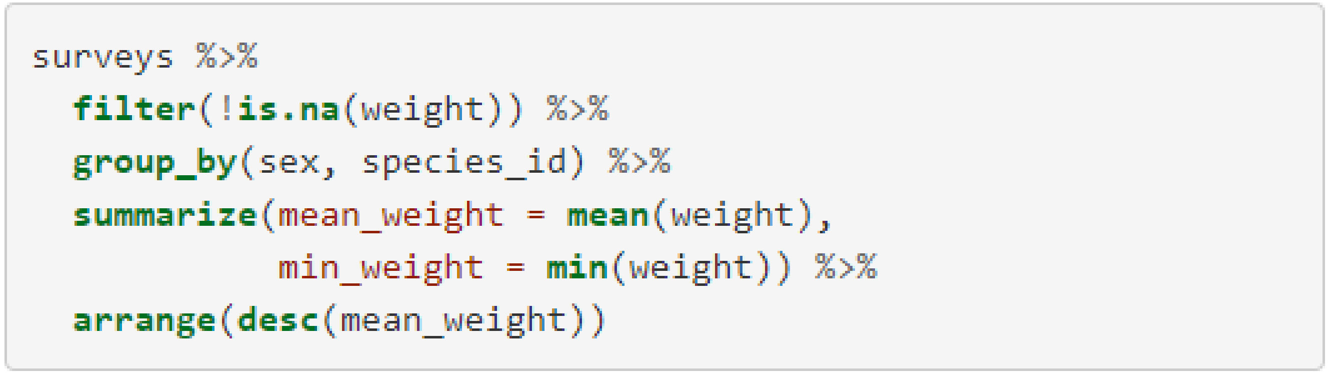
- You can also group by multiple columns, subset and control how the output is printed:



```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(sex, species_id) %>%
  summarize(mean_weight = mean(weight)) %>%
  print(n = 15)
```

Rearranging columns

- It is sometimes useful to rearrange the result of a query to inspect the values.
- To sort in descending order, we need to add the desc() function. If we want to sort the results by decreasing order of mean weight.



```
surveys %>%
  filter(!is.na(weight)) %>%
  group_by(sex, species_id) %>%
  summarize(mean_weight = mean(weight),
            min_weight = min(weight)) %>%
  arrange(desc(mean_weight))
```

Counting

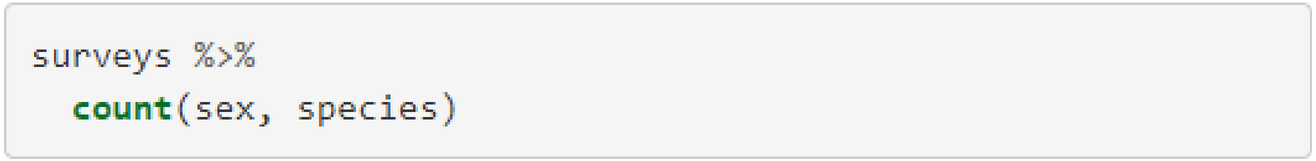
- When working with data, we often want to know the number of observations found for each factor or combination of factors.
- For this task, dplyr provides count().
- For example, if we wanted to count the number of rows of data for each sex, we would do:

```r
surveys %>%
    count(sex)
```

- Same as

```r
surveys %>%
    group_by(sex) %>%
    summarise(count = n())
```

- Also works for combinations of factors
- Eg



```
surveys %>%
  count(sex, species)
```

Exporting data (the dplyr way!)

- Similar to write.csv() use write_csv()

- We can create new directories to save our data to from inside of R
- First check whether it already exist or not

  dir.exists(paths)

- Then create a new folder

  dir.create(path)