Manipulating data with dplyr

Based on R-ecology lesson - Data carpentry

Marco Chiapello, PhD

March 29, 2017

What is dplyr?

- **dplyr** is a package for making data manipulation easier
  - Packages in R are basically sets of additional functions that let you do more stuff

What is dplyr?

- **dplyr** is a package for making data manipulation easier
  - Packages in R are basically sets of additional functions that let you do more stuff
- dplyr provides **easy tools** for the most common **data manipulation** tasks

What is dplyr?

- **dplyr** is a package for making data manipulation easier
  - Packages in R are basically sets of additional functions that let you do more stuff
- dplyr provides **easy tools** for the most common **data manipulation** tasks
- dplyr addresses this by porting much of the computation to C++

What is dplyr?

- **dplyr** is a package for making data manipulation easier
  - Packages in R are basically sets of additional functions that let you do more stuff
- dplyr provides **easy tools** for the most common **data manipulation** tasks
- dplyr addresses this by porting much of the computation to C++
- An additional feature is the ability to work directly with data stored in an **external database**

What is dplyr?

Before start to dig into dplyr functions we learn how to import data into R

- To download the data, run the following:

```
download.file("https://ndownloader.figshare.com/files/2292169",
              "portal_data_joined.csv")
```

- You are now ready to load the data:

```
surveys <- read.csv('portal_data_joined.csv')
```

What is dplyr?

Before start to dig into dplyr functions we learn how to import data into R

- To download the data, run the following:

```r
download.file("https://ndownloader.figshare.com/files/2292169",
              "portal_data_joined.csv")
```

- You are now ready to load the data:

```r
surveys <- read.csv('portal_data_joined.csv')
```

- Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen

```r
surveys <- tbl_df(surveys)
```

WHAT IS DPLYR?

We're going to learn some of the *most common dplyr functions:*

- **select**
- **filter**
- **arrange**
- **mutate**
- **group_by**
- **summarize**

Select

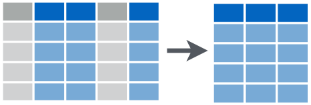

Figure 1: Select

- To select columns of a data frame, use select()
  - The **first argument** to this function is the **data frame**
  - The **subsequent arguments** are the **columns to keep**

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

Select

- select is much more powerful than just select the interest columns
  - You can remove one column

```r
select(surveys, -weight)
```

Select

- select is much more powerful than just select the interest columns
    - You can remove one column

```
select(surveys, -weight)
```

```
*  Select columns whose name contains a character string
```

```
select(surveys, contains("ec"))
```

Select

- Select columns whose name starts with a character string

```
select(surveys, starts_with("s"))
```

Select

- Select columns whose name starts with a character string

```
select(surveys, starts_with("s"))
```

- Select all columns between Sepal.Length and Petal.Width (inclusive).

```
select(surveys, plot_id:weight)
```

Select

- Select columns whose name starts with a character string

```
select(surveys, starts_with("s"))
```

- Select all columns between Sepal.Length and Petal.Width (inclusive).

```
select(surveys, plot_id:weight)
```

- Select every column

```
select(surveys, weight, everything())
```

Filter

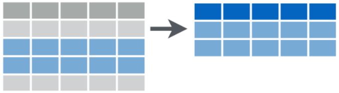

Figure 2: Filter

- To select rows of a data frame, use filter()
  - The **first argument** to this function is the **data frame**
  - The **subsequent arguments** are the **conditions for the subsetting**

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

Filter

- Extract rows that meet logical criteria

```
filter(surveys, year > 1995)
filter(surveys, year > 1995, day == 12)
filter(surveys, year > 1995, day == 12 | day == 24)
filter(surveys, year > 1995, day == 12 | (day == 24 & sex != "F"))
```

CHALLENGE

Subset the surveys dataset:

- The data from spring of 1999
- All the females heavier than 100 after 1995
- All the female heavier than 150 in days 11 and 18 and the males of NL specie

Filter

- Extract rows that meet logical criteria

```
filter(surveys, year > 1995)
filter(surveys, year > 1995, day == 12)
filter(surveys, year > 1995, day == 12 | day == 24)
filter(surveys, year > 1995, day == 12 | (day == 24 & sex != "F"))
```

CHALLENGE

Subset the surveys dataset:

- The data from spring of 1999
- All the females heavier than 100 after 1995
- All the female heavier than 150 in days 11 and 18 and the males of NL specie

```
filter(surveys, year == 1999, month >= 3 & month <= 6 )
filter(surveys, sex == "F", weight > 100, year > 1995)
filter(surveys, (sex == "F" & (day == 11 | day == 18)) |
       (sex == "M" & species_id == "NL"))
```

Pipes

But what if you wanted to **select and filter at the same time**?

There are *three ways* to do this:

Pipes

But what if you wanted to **select and filter at the same time**?

There are *three ways* to do this:

- use intermediate steps
  - you essentially create a temporary data frame and use that as input to the next function

```
tmp <- select(surveys, plot_id, species_id, weight)
filter(tmp, weight > 270)
```

Pipes

But what if you wanted to **select and filter at the same time**?

There are *three ways* to do this:

- use intermediate steps
  - you essentially create a temporary data frame and use that as input to the next function

```
tmp <- select(surveys, plot_id, species_id, weight)
filter(tmp, weight > 270)
```

- nested functions
  - one function inside of another

```
filter(select(surveys, plot_id, species_id, weight), weight > 270)
```

Pipes

- pipes
  - pipes let you take the output of one function and send it directly to the next

```
select(surveys, plot_id, species_id, weight) %>%
    filter(weight > 270)
```

Pipes

- pipes
  - pipes let you take the output of one function and send it directly to the next

```
select(surveys, plot_id, species_id, weight) %>%
    filter(weight > 270)
```

- Pipes in R look like %>% and are made available via the magrittr package, installed automatically with dplyr

Pipes

- pipes
  - pipes let you take the output of one function and send it directly to the next

```
select(surveys, plot_id, species_id, weight) %>%
    filter(weight > 270)
```

- Pipes in R look like %>% and are made available via the magrittr package, installed automatically with dplyr
- **%>% takes the object on its left and passes it as the first argument to the function on its right**, we don't need to explicitly include it as an argument to the filter() and select() functions anymore

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

Pipes

CHALLANGE

- Filter females from specie NL and select year and weight
- Filter male from 1995 and report all columns apart for month
- Filter 1995, rodent and report column: taxa, year and then all the others

Pipes

CHALLANGE

- Filter females from specie NL and select year and weight
- Filter male from 1995 and report all columns apart for month
- Filter 1995, rodent and report column: taxa, year and then all the others

```
surveys %>%
    filter(sex == "F", species_id == "NL") %>%
    select(year, weight)

surveys %>%
    filter(sex == "M", year == 1995) %>%
    select(-month)

surveys %>%
    filter(year == 1995, taxa == "Rodent") %>%
    select(taxa, year, everything())
```

Mutate

Frequently you'll want to **create new columns based on the values in existing columns**

For this we'll use **mutate()**

```
surveys %>%
      mutate(weight_kg = weight / 1000) %>%
      select(weight, weight_kg)
```

Mutate

Frequently you'll want to **create new columns based on the values in existing columns**

For this we'll use **mutate()**

```
surveys %>%
      mutate(weight_kg = weight / 1000) %>%
      select(weight, weight_kg)
```

The first few rows of the output are full of NAs, so if we wanted to remove those

```
surveys %>%
      filter(!is.na(weight)) %>%
      mutate(weight_kg = weight / 1000) %>%
      select(weight, weight_kg)
```

Mutate

CHALLENGE

Create a new data frame from the survey data that meets the following criteria:

- contains only the species_id column and a new column called hindfoot_half containing values that are half the hindfoot_length values.
- In the hindfoot_half column, there are no NAs and all values are less than 30.

Mutate

CHALLENGE

Create a new data frame from the survey data that meets the following criteria:

- contains only the species_id column and a new column called hindfoot_half containing values that are half the hindfoot_length values.
- In the hindfoot_half column, there are no NAs and all values are less than 30.

```
surveys %>%
    filter(!is.na(hindfoot_length)) %>%
    mutate(hindfoot_half = hindfoot_length / 2) %>%
    select(species_id, hindfoot_half)
```

Group_by and Summarize

Many data analysis tasks can be approached using the **split-apply-combine paradigm**:

- **split** the data into groups
- **apply** some analysis to each group
- **combine** the results

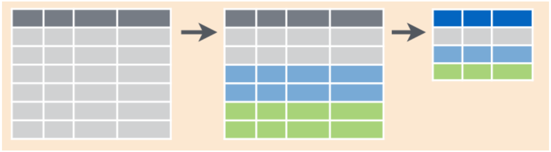

Figure 3: Grup_by and summarize

Group_by and Summarize

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

CHALLANGE

Remove the first row of the output that contains not male or female

Group_by and Summarize

- You can also group by multiple columns:

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

Group_by and Summarize

- You can also group by multiple columns:

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

* If you want to display more data:

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

Group_by and Summarize

- Once the data are grouped, you can also summarize multiple variables at the same time

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

- How to know the number of observations found for each factor or combination of factors

```
surveys %>%
      group_by(sex) %>%
        tally
```

tally() is the action applied to the groups created by group_by() and counts the total number of records for each category