Dataframe Manipulation with dplyr

Remembering that we can use the readxl package to read external data into R, for this lesson we are going to use the wetland data:

library("readxl")  
veg <- read\_excel("data/WetPlots.xlsx", 3)   
veg

## # A tibble: 259 x 8  
## fid plot.id species.name latin pc\_cover gt\_3m eq\_1to3 lt\_1m  
## <chr> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 111563 1 Sitka sedge/ Wate~ carex aquatil~ 10 NA NA NA  
## 2 111563 1 Calamograsis cana~ calamograsis ~ 15 NA NA NA  
## 3 111563 1 Mannagrass sp. glyceria sp 5 NA NA NA  
## 4 111563 1 Water arum calla palustr~ 5 NA NA NA  
## 5 111563 1 Pink Spirea sprea salicif~ 5 NA NA 100  
## 6 111563 1 Mountain alder alnus incana 10 90 NA NA  
## 7 111563 1 Salix Sp. (Pacifi~ <NA> T NA 80 20  
## 8 111563 1 Marsh cinquefoil potentilla pa~ 15 NA NA NA  
## 9 111563 1 Open water <NA> 5 NA NA NA  
## 10 111563 1 Bur-reed sp. carex sprgani~ T NA NA NA  
## # ... with 249 more rows

Manipulation of dataframes means many things to many researchers, we often select certain observations (rows) or variables (columns), we often group the data by a certain variable(s), or we even calculate summary statistics. We can do these types of operations using the normal base R operations:

mean(veg$gt\_3m)

## [1] NA

mean(veg$gt\_3m, na = TRUE)

## [1] 45.08108

The question here is how might you take averages (or any other summary statistic) by some group you might be interested in.

## The dplyr package

Luckily, the [dplyr](https://cran.r-project.org/package=dplyr) package provides a number of very useful functions for manipulating dataframes in a way that will reduce the above repetition, reduce the probability of making errors, and probably even save you some typing. As an added bonus, you might even find the dplyr grammar easier to read.

Here we’re going to cover 5 of the most commonly used functions as well as using pipes (%>%) to combine them.

1. select()
2. filter()
3. group\_by()
4. summarize()
5. mutate()

If you have have not installed this package earlier, please do so:

install.packages('dplyr')

Now let’s load the package:

library("dplyr")

## Warning: package 'dplyr' was built under R version 3.6.2

## Using select()

If, for example, we wanted to move forward with only a few of the variables in our dataframe we could use the select() function. This will keep only the variables you select.

veg\_pc <- select(veg, plot.id, species.name, pc\_cover)

## Using filter()

To select on some subset of rows:

filter(veg, plot.id == 1)

## # A tibble: 190 x 8  
## fid plot.id species.name latin pc\_cover gt\_3m eq\_1to3 lt\_1m  
## <chr> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 111563 1 Sitka sedge/ Wate~ carex aquatil~ 10 NA NA NA  
## 2 111563 1 Calamograsis cana~ calamograsis ~ 15 NA NA NA  
## 3 111563 1 Mannagrass sp. glyceria sp 5 NA NA NA  
## 4 111563 1 Water arum calla palustr~ 5 NA NA NA  
## 5 111563 1 Pink Spirea sprea salicif~ 5 NA NA 100  
## 6 111563 1 Mountain alder alnus incana 10 90 NA NA  
## 7 111563 1 Salix Sp. (Pacifi~ <NA> T NA 80 20  
## 8 111563 1 Marsh cinquefoil potentilla pa~ 15 NA NA NA  
## 9 111563 1 Open water <NA> 5 NA NA NA  
## 10 111563 1 Bur-reed sp. carex sprgani~ T NA NA NA  
## # ... with 180 more rows

Above we used ‘normal’ grammar, but the strengths of dplyr lie in combining several functions using pipes. This [description](https://twitter.com/WeAreRLadies/status/1172576445794803713) is very useful:

I work up %>%   
 showered %>%   
 dressed %>%   
 had coffee %>%   
 came to an R workshop

Since the pipes grammar is unlike anything we’ve seen in R before, let’s repeat what we’ve done above using pipes.

veg\_pc <- veg %>% select(plot.id, species.name, pc\_cover)

To help you understand why we wrote that in that way, let’s walk through it step by step. First we summon the gapminder dataframe and pass it on, using the pipe symbol %>%, to the next step, which is the select() function. In this case we don’t specify which data object we use in the select() function since in gets that from the previous pipe.

## Using filter() with pipes

If we now wanted to move forward with the above, but only with European countries, we can combine select and filter

veg\_pc\_p1 <- veg %>%  
 filter(plot.id == 1) %>%  
 select(species.name, pc\_cover)

## Challenge 1

Write a single command (which can span multiple lines and includes pipes) that will produce a dataframe that has data for plot 2 and contains the column for latin, pc\_cover. How many rows does your dataframe have and why?

As with last time, first we pass the gapminder dataframe to the filter() function, then we pass the filtered version of the gapminder dataframe to the select() function. **Note:** The order of operations is very important in this case. If we used ‘select’ first, filter would not be able to find the variable plot.id since we would have removed it in the previous step.

## Using group\_by() and summarize()

Now, we were supposed to be reducing the error prone repetitiveness of what can be done with base R, but up to now we haven’t done that since we would have to repeat the above for each continent. Instead of filter(), which will only pass observations that meet your criteria (in the above: plot.id== 1), we can use group\_by(), which will essentially use every unique criteria that you could have used in filter.

veg

## # A tibble: 259 x 8  
## fid plot.id species.name latin pc\_cover gt\_3m eq\_1to3 lt\_1m  
## <chr> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 111563 1 Sitka sedge/ Wate~ carex aquatil~ 10 NA NA NA  
## 2 111563 1 Calamograsis cana~ calamograsis ~ 15 NA NA NA  
## 3 111563 1 Mannagrass sp. glyceria sp 5 NA NA NA  
## 4 111563 1 Water arum calla palustr~ 5 NA NA NA  
## 5 111563 1 Pink Spirea sprea salicif~ 5 NA NA 100  
## 6 111563 1 Mountain alder alnus incana 10 90 NA NA  
## 7 111563 1 Salix Sp. (Pacifi~ <NA> T NA 80 20  
## 8 111563 1 Marsh cinquefoil potentilla pa~ 15 NA NA NA  
## 9 111563 1 Open water <NA> 5 NA NA NA  
## 10 111563 1 Bur-reed sp. carex sprgani~ T NA NA NA  
## # ... with 249 more rows

veg %>% group\_by(plot.id)

## # A tibble: 259 x 8  
## # Groups: plot.id [4]  
## fid plot.id species.name latin pc\_cover gt\_3m eq\_1to3 lt\_1m  
## <chr> <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl>  
## 1 111563 1 Sitka sedge/ Wate~ carex aquatil~ 10 NA NA NA  
## 2 111563 1 Calamograsis cana~ calamograsis ~ 15 NA NA NA  
## 3 111563 1 Mannagrass sp. glyceria sp 5 NA NA NA  
## 4 111563 1 Water arum calla palustr~ 5 NA NA NA  
## 5 111563 1 Pink Spirea sprea salicif~ 5 NA NA 100  
## 6 111563 1 Mountain alder alnus incana 10 90 NA NA  
## 7 111563 1 Salix Sp. (Pacifi~ <NA> T NA 80 20  
## 8 111563 1 Marsh cinquefoil potentilla pa~ 15 NA NA NA  
## 9 111563 1 Open water <NA> 5 NA NA NA  
## 10 111563 1 Bur-reed sp. carex sprgani~ T NA NA NA  
## # ... with 249 more rows

You will notice that the structure of the dataframe where we used group\_by() (grouped\_df) is not the same as the original veg (data.frame). A grouped\_df can be thought of as a list where each item in the list is a data.frame which contains only the rows that correspond to the a particular value plot.id (at least in the example above).

## Using summarize()

The above was a bit on the uneventful side but group\_by() is much more exciting in conjunction with summarize(). This will allow us to create new variable(s) by using functions that repeat for each of the plot-specific data frames. That is to say, using the group\_by() function, we split our original dataframe into multiple pieces, then we can run functions (e.g. n(), mean() or sd()) within summarize().

veg\_sp\_plot <- veg %>%  
 group\_by(fid) %>%  
 summarize(count = n())

That allowed us to calculate the number of rows (species) per site, but it gets even better.

## Challenge 2

Calculate the average (mean) percent cover per plot. Which plot has the highest pc\_cover? Hint calculations can only apply to numberic data types so you will need to convert the pc\_cover

Another way to do this is to use the dplyr function arrange(), which arranges the rows in a data frame according to the order of one or more variables from the data frame. It has similar syntax to other functions from the dplyr package. You can use desc() inside arrange() to sort in descending order.

veg$pc\_cover <- as.numeric(veg$pc\_cover)

## Warning: NAs introduced by coercion

plot\_mean <- veg %>%  
 group\_by(plot.id) %>%  
 summarize(mean\_cover = mean(pc\_cover, na.rm = TRUE))  
  
plot\_mean %>%  
 arrange(mean\_cover) %>%  
 head(1)

## # A tibble: 1 x 2  
## plot.id mean\_cover  
## <dbl> <dbl>  
## 1 3 21.5

plot\_mean %>%  
 arrange(desc(mean\_cover)) %>%  
 head(1)

## # A tibble: 1 x 2  
## plot.id mean\_cover  
## <dbl> <dbl>  
## 1 2 35.0

The function group\_by() allows us to group by multiple variables. Let’s group by plot.id and species.name.

cover\_site\_plot <- veg %>%  
 group\_by(fid, plot.id) %>%  
 summarize(mean\_pc\_cover = mean(pc\_cover, na.rm = TRUE))

That is already quite powerful, but it gets even better! You’re not limited to defining 1 new variable in summarize().

cover\_detail\_pc <- veg %>%  
 group\_by(fid, plot.id) %>%  
 summarize(mean\_pc\_cover = mean(pc\_cover, na.rm = TRUE),  
 sd\_pc\_cover = sd(pc\_cover, na.rm = TRUE),  
 mean\_gt3m = mean(gt\_3m, na.rm = TRUE),  
 sd\_gt3m = sd(gt\_3m, na.rm = TRUE))

## count() and n()

A very common operation is to count the number of observations for each group. The dplyr package comes with two related functions that help with this.

For instance, if we wanted to check the number of species included in the dataset for one site, we can use the count() function. It takes the name of one or more columns that contain the groups we are interested in, and we can optionally sort the results in descending order by adding sort=TRUE:

veg %>%  
 filter(plot.id == 1) %>%  
 count(species.name, sort = TRUE)

## # A tibble: 113 x 2  
## species.name n  
## <chr> <int>  
## 1 Sphagnum sp. 8  
## 2 Water Sedge 8  
## 3 Black Spruce 6  
## 4 Labrador Tea 6  
## 5 Marsh Cinquefoil 6  
## 6 Pink Spirea 6  
## 7 Scrub Birch 6  
## 8 Bog Cranberry 5  
## 9 Sedge sp. 4  
## 10 Spruce sp. 4  
## # ... with 103 more rows

If we need to use the number of observations in calculations, the n() function is useful. It will return the total number of observations in the current group rather than counting the number of observations in each group within a specific column.

You can also chain together several summary operations; in this case calculating the minimum, maximum, mean and se of each continent’s per-country life-expectancy:

veg %>%  
 group\_by(fid) %>%  
 summarize(  
 mean\_cover = mean(pc\_cover, na.rm = TRUE),  
 min\_cover = min(pc\_cover,na.rm = TRUE),  
 max\_cover = max(pc\_cover, na.rm = TRUE))

## # A tibble: 21 x 4  
## fid mean\_cover min\_cover max\_cover  
## <chr> <dbl> <dbl> <dbl>  
## 1 109171 25.5 1 95  
## 2 109411 24.3 1 90  
## 3 109454 48.8 10 90  
## 4 109507 39.1 5 90  
## 5 109507b 25.6 1 95  
## 6 111036 NaN Inf -Inf  
## 7 111563 16.2 5 50  
## 8 12519 58.3 25 100  
## 9 139285 24.4 1 100  
## 10 14539 27.9 5 80  
## # ... with 11 more rows

## Using mutate()

We can also create new variables prior to (or even after) summarizing information using mutate().

library(stringr)  
  
veg\_groups <- veg %>%  
 mutate(latin\_family = word(latin, 1)) %>%  
 group\_by(fid, latin\_family) %>%  
 summarize(  
 mean\_cover = mean(pc\_cover, na.rm = TRUE),  
 min\_cover = min(pc\_cover,na.rm = TRUE),  
 max\_cover = max(pc\_cover, na.rm = TRUE))

## Connect mutate with logical filtering: ifelse

When creating new variables, we can hook this with a logical condition. A simple combination of mutate() and ifelse() facilitates filtering right where it is needed: in the moment of creating something new. This easy-to-read statement is a fast and powerful way of discarding certain data (even though the overall dimension of the data frame will not change) or for updating values depending on this given condition.

veg\_groups <- veg %>%  
 mutate(latin\_family = word(latin, 1)) %>%  
 filter(latin\_family == "carex") %>%  
 group\_by(fid ) %>%  
 summarize(  
 mean\_cover = mean(pc\_cover, na.rm = TRUE))

## Combining dplyr and ggplot2

First install and load ggplot2:

install.packages('ggplot2')

library("ggplot2")

In the plotting lesson we looked at how to make a multi-panel figure by adding a layer of facet panels using ggplot2. Here is the code we used (with some extra comments):

# plot average cover of all species per plot.   
# first we have to summarise the data   
  
cover\_detail\_pc <- veg %>%  
 group\_by(fid, plot.id) %>%  
 summarize(mean\_pc\_cover = mean(pc\_cover, na.rm = TRUE),  
 sd\_pc\_cover = sd(pc\_cover, na.rm = TRUE),  
 mean\_gt3m = mean(gt\_3m, na.rm = TRUE),  
 sd\_gt3m = sd(gt\_3m, na.rm = TRUE))  
  
cover\_detail\_pc

## # A tibble: 34 x 6  
## # Groups: fid [21]  
## fid plot.id mean\_pc\_cover sd\_pc\_cover mean\_gt3m sd\_gt3m  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 109171 1 25.5 31.6 NaN NaN  
## 2 109411 1 24.3 33.1 1 NaN  
## 3 109454 1 48.8 39.7 NaN NaN  
## 4 109507 1 42.5 44.1 NaN NaN  
## 5 109507 2 70 0 NaN NaN  
## 6 109507 3 5 NaN NaN NaN  
## 7 109507 4 28.8 34.5 NaN NaN  
## 8 109507b 1 25.6 32.8 20 NaN  
## 9 111036 1 NaN NaN NaN NaN  
## 10 111563 1 8.5 4.12 90 NaN  
## # ... with 24 more rows

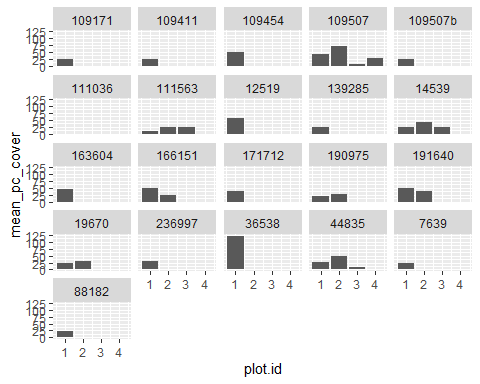
ggplot(data = veg, aes(x = plot.id, y = pc\_cover)) +  
 geom\_bar(stat = "identity")

## Warning: Removed 59 rows containing missing values (position\_stack).



# Make the plot per site:   
  
ggplot(data = cover\_detail\_pc, aes(x = plot.id, y = mean\_pc\_cover)) +  
 geom\_bar(stat = "identity") + facet\_wrap( ~ fid)

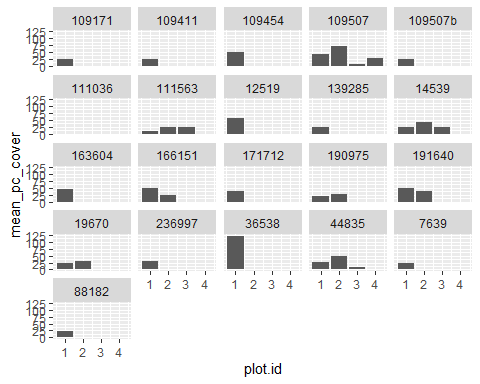
## Warning: Removed 1 rows containing missing values (position\_stack).



This code makes the right plot but it also provides a way to chain operations. Just as we used %>% to pipe data along a chain of dplyr functions we can use it to pass data to ggplot(). Because %>% replaces the first argument in a function we don’t need to specify the data = argument in the ggplot() function. By combining dplyr and ggplot2 functions we can alter this figure for only those continents in Europe.

veg %>%  
 group\_by(fid, plot.id) %>%  
 summarize(mean\_pc\_cover = mean(pc\_cover, na.rm = TRUE),  
 sd\_pc\_cover = sd(pc\_cover, na.rm = TRUE),  
 mean\_gt3m = mean(gt\_3m, na.rm = TRUE),  
 sd\_gt3m = sd(gt\_3m, na.rm = TRUE)) %>%  
 ggplot(aes(x = plot.id, y = mean\_pc\_cover)) +  
 geom\_bar(stat = "identity") +   
 facet\_wrap( ~ fid)

## Warning: Removed 1 rows containing missing values (position\_stack).



Using dplyr functions also helps us do calculations on the fly.

## Advanced Challenge

Calculate the average pc\_cover of 2 randomly selected species for each fid. Then arrange the continent names in reverse order. **Hint:** Use the dplyr functions arrange() and sample\_n(), they have similar syntax to other dplyr functions.

## Solution to Challenge 1

veg\_pc\_p2 <- veg %>%  
 filter(plot.id == 2) %>%  
 select(latin, pc\_cover)

## Solution to Challenge 2

veg$pc\_cover <- as.numeric(veg$pc\_cover)  
  
plot\_mean <- veg %>%  
 group\_by(plot.id) %>%  
 summarize(mean\_cover = mean(pc\_cover, na.rm = TRUE))

## Solution to Advanced Challenge

veg %>%  
 group\_by(fid) %>%  
 sample\_n(2) %>%  
 summarize(mean\_cover = mean(pc\_cover)) %>%  
 arrange(desc(mean\_cover))

## # A tibble: 21 x 2  
## fid mean\_cover  
## <chr> <dbl>  
## 1 7639 57.5  
## 2 171712 47.5  
## 3 163604 45   
## 4 12519 37.5  
## 5 36538 35   
## 6 44835 20   
## 7 166151 12.5  
## 8 109411 10.5  
## 9 190975 10   
## 10 109171 8   
## # ... with 11 more rows