A quick introduction to dplyr

What is *dplyr*?

dplyr (rhymes with plier) is a powerful R package with tools for manipulation of tabular data.

Why use it?

Well, manipulating data frames is one of the most important things in R, and it is essential for good plotting of data and results.

You may already be familiar with base R functions such as split(), subset(), apply(), sapply(), lapply(), tapply() and aggregate(). Compared to base functions in R, the functions in dplyr are easier to work with, are more consistent in the syntax and are targeted for data analysis around data frames instead of just vectors.

How do I get it?

If you have never used it before you need to install it

```
install.packages("dplyr")
```

And every time you use it you need to load it.

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

What is in it?

dplyr is very extensive and has methods to deal with different kinds of data structures, but there are **5 functions** that you can learn quickly and use on your R data frame objects. They are:

1. select(): focus on a subset of variables
2. filter(): focus on a subset of rows
3. mutate(): add new columns
4. summarise(): get summary statistics
5. arrange(): re-order the rows

You can use column names as parameters without having to use the $ operator, which makes code more readable.

1. select()

basic pseudocode: select(your_dataframe, columns you want to select)

example:

```
head(iris) #built-in R data for example

#I only want to analyse Species and Sepal Length

myiris <- select(iris, Species, Sepal.Length)

head(myiris)

#notice it changes the column ordering to what you chose too
```

2. filter()

My favorite: only get the data that follows a certain condition

basic pseudocode: filter(your_dataframe, true/false condition)

example:

```
#I only want to analyse Species virginica and Petal length > 5

myiris2 <- filter(iris,Species == "virginica",Petal.Length > 5)

head(myiris2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          5.8         2.7          5.1         1.9 virginica
## 3          7.1         3.0          5.9         2.1 virginica
## 4          6.3         2.9          5.6         1.8 virginica
## 5          6.5         3.0          5.8         2.2 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

3. mutate()

Create new columns

example:

```
# create a new column that stores logical values for sepal.width grea
myiris3 <- mutate(iris,
           greater.half = Sepal.Width > 0.5 * Sepal.Length)

head(myiris3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species greater.
## 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
## 6          5.4         3.9          1.7         0.4  setosa
```

4. summarise()

summarise(factors to group, summary statistics to use)

```
summarise(group_by(iris,Species),mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    `mean(Sepal.Length)`
##   <fct>                     <dbl>
## 1 setosa                     5.01
## 2 versicolor                 5.94
## 3 virginica                  6.59
```

```
summarise(group_by(iris,Species),sd(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    `sd(Sepal.Length)`
```

```
##   <fct>                     <dbl>
## 1 setosa                    0.352
## 2 versicolor                0.516
## 3 virginica                 0.636
```

Notice that here we are also using group_by(). That is an useful dplyr function that allows for group operations.

5. arrange()

arrange(yourdata, columns that should be sorted by, in order)

```
head(arrange(iris, Sepal.Length))
```

```
##   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
## 6          4.6         3.1          1.5         0.2  setosa
```

```
head(arrange(iris, Sepal.Length, Petal.Length))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.3         3.0          1.1         0.1  setosa
## 2          4.4         3.0          1.3         0.2  setosa
## 3          4.4         3.2          1.3         0.2  setosa
## 4          4.4         2.9          1.4         0.2  setosa
## 5          4.5         2.3          1.3         0.3  setosa
## 6          4.6         3.6          1.0         0.2  setosa
```

```
head(arrange(iris, desc(Sepal.Length)))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
```

```
## 1          7.9         3.8          6.4         2.0 virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.7         2.6          6.9         2.3 virginica
## 4          7.7         2.8          6.7         2.0 virginica
## 5          7.7         3.0          6.1         2.3 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

Piping

One last thing I want to introduce is piping. If you search the internet for code using *dplyr*, most of it will be using the %>% operator, or a pipe. It is used to organize your code. It's easier to just show you an example:

```
setosa <- iris %>% filter(Species =="setosa")
```

Is exactly equivalent to

```
setosa <- filter(iris,Species =="setosa")
```

For one operation using %>% is not very advantageous, but frequently we want to do many operations in sequence. For that, using %>% will give you more readable, better organized code, without having to nest multiple functions or create useless variables. For example, these chunks of code all do the same thing:

```
###awful nest
plot(mutate(arrange(filter(iris,Species =="setosa"),Sepal.Length),
            Sepal.round = round(Sepal.Length)))

##many intermediates
setosa <- filter(iris,Species =="setosa")
```

```r
ordered <- arrange(setosa,Sepal.Length)
rounded <- mutate(ordered, Sepal.round = round(Sepal.Length))
plot(rounded)

##nice, orderly, and piped
iris %>% filter(Species =="setosa") %>%
         arrange(Sepal.Length) %>%
         mutate(Sepal.round = round(Sepal.Length)) %>%
         plot()
```

Practice Exercises

1) Select the first three columns of the iris dataset using their column names.
2) Select all the columns of the iris dataset except "Petal Width". HINT: Use "-".
3) Filter the rows of the iris dataset for Sepal.Length >= 4.6 and Petal.Width >= 0.5.
4) Pipe the iris data frame to the function that will select two columns (Sepal.Width and Sepal.Length).
5) Arrange rows by a particular column, such as the Sepal.Width, in descending order.
6) Select three columns from iris, arrange the rows alphabetically by species and then break ties ordering by Sepal.Width.
7) Create a new column called proportion, which is the ratio of Sepal.Length to Sepal.Width.

8) Compute the average number of Sepal.Length, apply the mean() function to the column Sepal.Length, and call the summary value "avg_slength". HINT: Use summarize().

9) Split the iris data frame by the Sepal.Length, then ask for the same summary statistics as above. HINT: Use group_by().