Functional Programming

Data Wrangling in R

Functional Programming

“R, at its heart, is a functional programming (FP) language. This means that it provides many tools for the creation and manipulation of functions. In particular, R has what’s known as first class functions. You can do anything with functions that you can do with vectors: you can assign them to variables, store them in lists, **pass them as arguments to other functions**, create them inside functions, and even return them as the result of a function.” - Hadley Wickham

Allows you to flexibly iterate functions to multiple elements of a data object!

Useful when you want to apply a function to:
* lots of columns in a tibble
* multiple tibbles
* multiple data files

working `across` mulptiple columns

Say we wanted to round multiple columns of the `mtcars` data. We could do so one column at a time, or we could use the `across` function from the `dplyr` package. Needs to be used **within other dplyr functions** such as `mutate`.

```
mutate(across(which_columns, which function or operation))
```

```
head(mtcars, 2)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4       21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag   21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
mtcars %>%
  mutate(across(.cols = c(disp, drat, wt, qsec), round)) %>%
  head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4       21   6  160 110    4  3   16  0  1    4    4
## Mazda RX4 Wag   21   6  160 110    4  3   17  0  1    4    4
```

Using `across` with arguments

Need to use the ~ if you wish to pass arguments to the function that you are applying to the various columns.

```
mtcars %>%
  mutate(across(.cols = c(disp, drat, wt, qsec), round)) %>%
  head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4       21   6  160 110    4  3   16  0  1    4    4
## Mazda RX4 Wag   21   6  160 110    4  3   17  0  1    4    4
```

```
mtcars %>%
  mutate(across(.cols = c(disp, drat, wt, qsec), ~ round(digits = 1))) %>%
  head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4       21   6    1 110    1  1    1  0  1    4    4
## Mazda RX4 Wag   21   6    1 110    1  1    1  0  1    4    4
```

tidy select helpers

https://tidyselect.r-lib.org/reference/select_helpers.html

?tidyr_tidy_select

- : range of consecutive variables
- ! ignore a variable
- everything(): Matches all variables.
- starts_with(): Starts with a prefix.
- ends_with(): Ends with a suffix.
- contains(): Contains a literal string.
- matches(): Matches a regular expression.

Using across with helpers to apply function to multiple columns

```
mtcars %>%
  mutate(across(.cols = disp:qsec, round)) %>%
  head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4       21   6  160 110    4  3   16  0  1    4    4
## Mazda RX4 Wag   21   6  160 110    4  3   17  0  1    4    4
```

```
mtcars %>%
  mutate(across(.cols = everything(), round))%>%
  head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4       21   6  160 110    4  3   16  0  1    4    4
## Mazda RX4 Wag   21   6  160 110    4  3   17  0  1    4    4
```

`purrr` is also a super helpful package!

The `purrr` package can be very helpful!

- https://purrr.tidyverse.org/
- https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf

purrr - apply function to all columns

```
library(purrr)
head(mtcars, 2)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
 mtcars %>%
   map_df(round) %>% # will be a tibble now - will remove rownames
   head(2)
```

```
## # A tibble: 2 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1    21     6   160   110     4     3    16     0     1     4     4
## 2    21     6   160   110     4     3    17     0     1     4     4
```

```
 mtcars %>%
   modify(round) %>% # modify keeps original data type
   head(2)
```

```
##                mpg cyl disp  hp drat wt qsec vs am gear carb
## Mazda RX4      21   6  160 110    4  3   16  0  1    4    4
## Mazda RX4 Wag  21   6  160 110    4  3   17  0  1    4    4
```

purrr apply function to some

Using `modify_if()`, we can specify what columns to modify

```
head(iris, 3)
```

```
##   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
```

```
iris %>%
  modify_if(is.numeric, as.character) %>%
  head(3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9           3          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

rowwise

```
iris %>%
  mutate(new =Sepal.Length + Petal.Width + Petal.Length + Sepal.Width) %>%
  head(2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species  new
## 1          5.1         3.5          1.4         0.2  setosa 10.2
## 2          4.9         3.0          1.4         0.2  setosa  9.5
```

```
iris %>%
  rowwise() %>%
  mutate(new =sum(Sepal.Length:Petal.Width))
```

```
## # A tibble: 150 x 6
## # Rowwise:
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species   new
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>   <dbl>
##  1          5.1         3.5          1.4         0.2 setosa   15.5
##  2          4.9         3            1.4         0.2 setosa   14.5
##  3          4.7         3.2          1.3         0.2 setosa   13.5
##  4          4.6         3.1          1.5         0.2 setosa   13
##  5          5           3.6          1.4         0.2 setosa   15
##  6          5.4         3.9          1.7         0.4 setosa   17.4
##  7          4.6         3.4          1.4         0.3 setosa   13
##  8          5           3.4          1.5         0.2 setosa   15
##  9          4.4         2.9          1.4         0.2 setosa   12
## 10          4.9         3.1          1.5         0.1 setosa   14.5
## # … with 140 more rows
```

pmap from purrr

```
iris %>%
  pmap(paste) %>%
  head()
```

```
## [[1]]
## [1] "5.1 3.5 1.4 0.2 setosa"
##
## [[2]]
## [1] "4.9 3 1.4 0.2 setosa"
##
## [[3]]
## [1] "4.7 3.2 1.3 0.2 setosa"
##
## [[4]]
## [1] "4.6 3.1 1.5 0.2 setosa"
##
## [[5]]
## [1] "5 3.6 1.4 0.2 setosa"
##
## [[6]]
## [1] "5.4 3.9 1.7 0.4 setosa"
```

- https://jennybc.github.io/purrr-tutorial/ –>

What is a 'list'?

- Lists are the most flexible/"generic" data class in R
- Can be created using list()
- Can hold vectors, strings, matrices, models, list of other list, lists upon lists!
- Can reference data using $ (if the elements are named), or using [], or [[]]

```
> mylist <- list(letters=c("A", "b", "c"),
+          numbers=1:3, matrix(1:25, ncol=5), matrix(1:25, ncol=5))
```

List Structure

```
> head(mylist)
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3

[[3]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25

[[4]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

List referencing

```
> mylist[1] # returns a list
```

```
$letters
[1] "A" "b" "c"
```

```
> mylist["letters"] # returns a list
```

```
$letters
[1] "A" "b" "c"
```

List referencing

```
> mylist[[1]] # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

```
> mylist$letters # returns vector
```

```
[1] "A" "b" "c"
```

```
> mylist[["letters"]] # returns the vector 'letters'
```

```
[1] "A" "b" "c"
```

List referencing

You can also select multiple lists with the single brackets.

```
> mylist[1:2] # returns a list
```

```
$letters
[1] "A" "b" "c"

$numbers
[1] 1 2 3
```

List referencing

You can also select down several levels of a list at once

```
> mylist$letters[1]
```

```
[1] "A"
```

```
> mylist[[2]][1]
```

```
[1] 1
```

```
> mylist[[3]][1:2,1:2]
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    7
```

How would I encounter lists?

This comes up a lot in data cleaning and also when reading in multiple files!

```
library(here)
list.files(here::here("data", "iris"), pattern = "*.csv")
```

```
## [1] "iris_q1.csv" "iris_q4.csv" "iris_q5.csv"
```

```
file_list <- paste0(here::here(), "/data/iris/", list.files(here::here("data", "iris"), pattern = "*.csv"))

multifile_data <- file_list %>%
  map(read_csv)

class(multifile_data)
```

```
## [1] "list"
```

Reading in multiple files

```
head(multifile_data[[1]], 3)
dim(multifile_data[[1]])
head(multifile_data[[2]], 3)
dim(multifile_data[[2]])
head(multifile_data[[3]], 3)
dim(multifile_data[[3]])

multifile_data[[2]] <- separate(multifile_data[[2]],
                            col =`Sepal.Length:Sepal.Width:Petal.Length:Petal.Width:Species`,
                         into = c("Sepal.Length", "Sepal.Width",
                                  'Petal.Length', "Petal.Width", "Species"), sep = ":")
head(multifile_data[[2]], 3)

multifile_data[[2]] <-multifile_data[[2]] %>% mutate(across(!Species, as.numeric))
```

Reading in multiple files

ldply combines results of applying a function to each element in a list into a data frame

```
library(plyr)
combined<-ldply(multifile_data)
dim(combined)
```

```
bindrows_data <- multifile_data %>%
    map_df(bind_rows, .id = "experiment") # recall that modify keeps the same data type
# so that will not do what we want here because we want a data frame!
dim(bindrows_data)
```

```
## [1] 450   7
```

```
tail(bindrows_data, 2)
```

```
## # A tibble: 2 x 7
##   experiment Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <chr>             <dbl>       <dbl>        <dbl>       <dbl> <chr>
## 1 3                   6.2         3.4          5.4         2.3 virginica
## 2 3                   5.9         3            5.1         1.8 virginica
## # … with 1 more variable:
## #   Sepal.Length:Sepal.Width:Petal.Length:Petal.Width:Species <chr>
```

See https://www.opencasestudies.org/ocs-bp-vaping-case-study for more information!

Factors

First we will create some data about absences for different students. Each row is a different student. We have information about the number of days absent and the grade for the individual students. We will use the `tibble()` function to create the data. We will use the `sample()` function to create a random sequence of numbers from 0 to 7 with replacements for 32 hypothetical students. Since there are four grades and 8*4 is 32, we will repeat the grade values 8 times. We use the `set.seed()` function so that the random sample from 0 to 7 is the same each time the code is run.

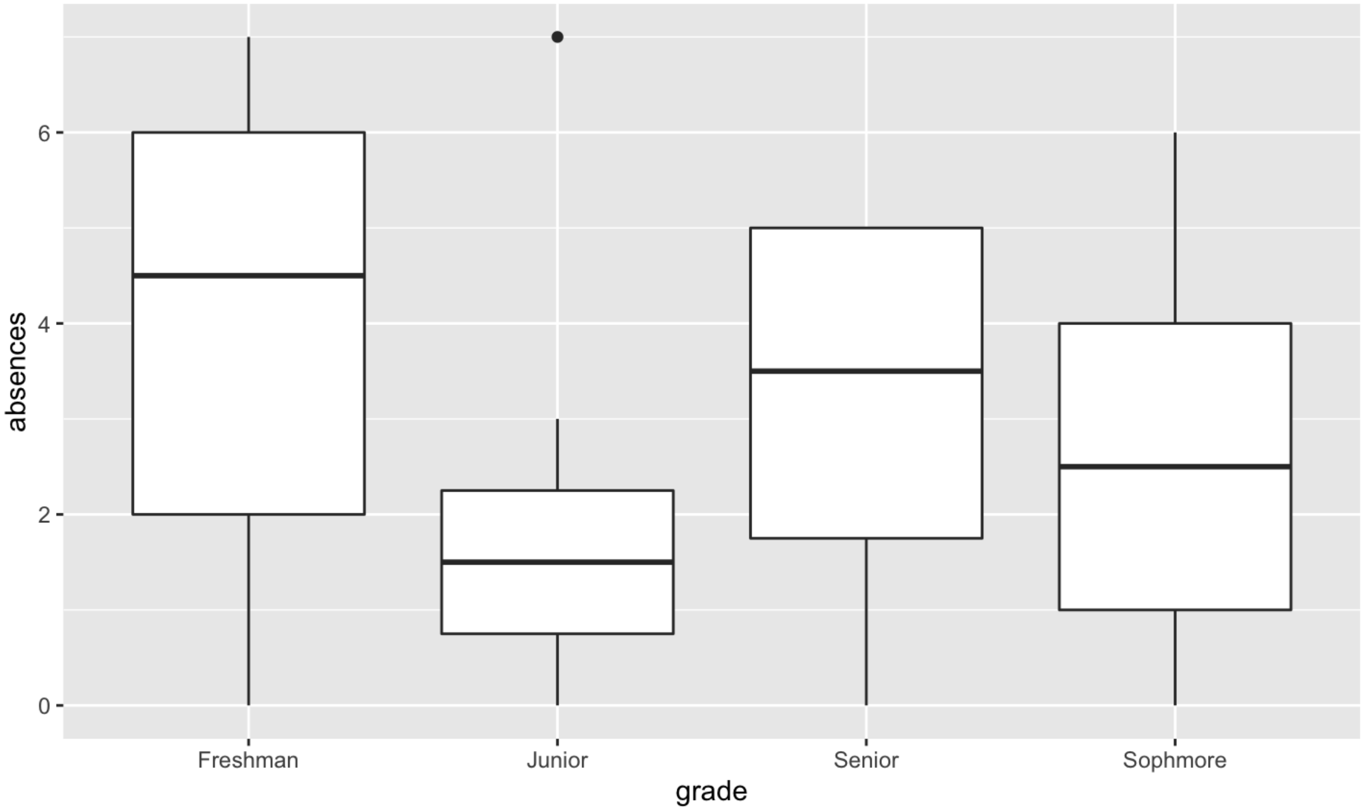
```
set.seed(123)
data_highschool <- tibble(absences = sample(0:7, size = 32, replace = TRUE),
                          grade = rep(c("Freshman","Sophmore",
                                        "Junior", "Senior"), 8))
head(data_highschool, 3)
```

```
## # A tibble: 3 x 2
##   absences grade
##      <int> <chr>
## 1        6 Freshman
## 2        6 Sophmore
## 3        2 Junior
```

Notice that `grade` is a `chr` variable. This indicates that the values are character strings. R does not realize that there is any order related to the `grade` values. However, we know that the order is: freshman, sophomore, junior, senior.

Let’s make a plot first:

```
#boxplot(data = data_highschool, absences ~ grade)
data_highschool %>%
  ggplot(mapping = aes(x = grade, y = absences)) +
  geom_boxplot()
```



Not quite what we want

OK this is very useful, but it is a bit difficult to read, because we expect the values to be plotted by the order that we know, not by alphabetical order. Currently `grade` is class `character` but let’s change that to class `factor` which allows us to specify the levels or order of the values.

As factor now

Using `as_factor()` from the `forcats` package the levels will be in the order in which they occur in the data!

https://forcats.tidyverse.org/

```
class(data_highschool$grade)
```

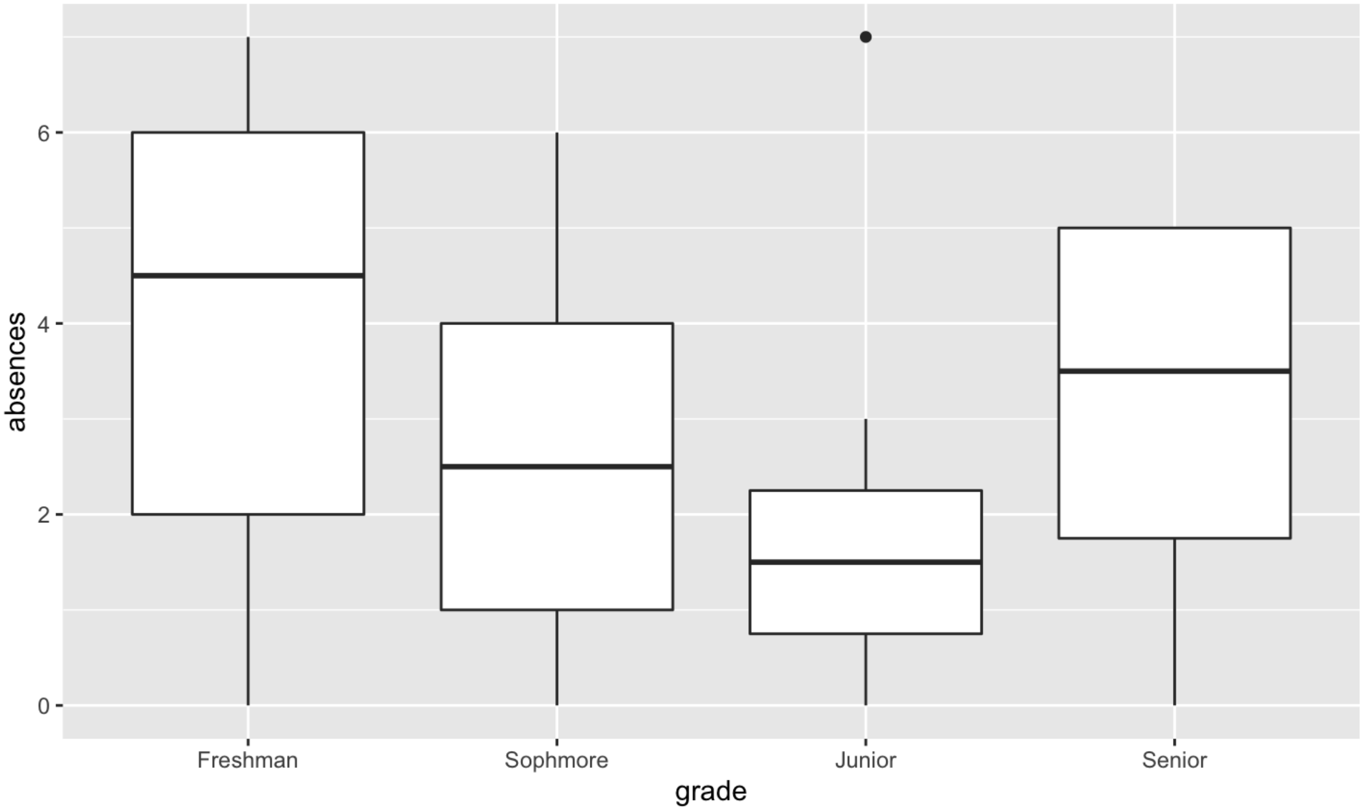
```
## [1] "character"
```

```
data_highschool_fct <- data_highschool %>%
  mutate(grade = as_factor(grade))
head(data_highschool_fct, 3)
```

```
## # A tibble: 3 x 2
##   absences grade
##      <int> <fct>
## 1        6 Freshman
## 2        6 Sophmore
## 3        2 Junior
```

Now let’s make our plot again:

```
#boxplot(data = data_highschool_fct, absences ~ grade)
data_highschool_fct %>%
  ggplot(mapping = aes(x = grade, y = absences)) +
  geom_boxplot()
```



Calculatons with factors?

Now what about results from some calculations.

```
data_highschool %>% group_by(grade) %>% summarise(mean = mean(absences))
```

```
## # A tibble: 4 x 2
##   grade     mean
##   <chr>    <dbl>
## 1 Freshman  4
## 2 Junior    2
## 3 Senior    3.12
## 4 Sophmore  2.62
```

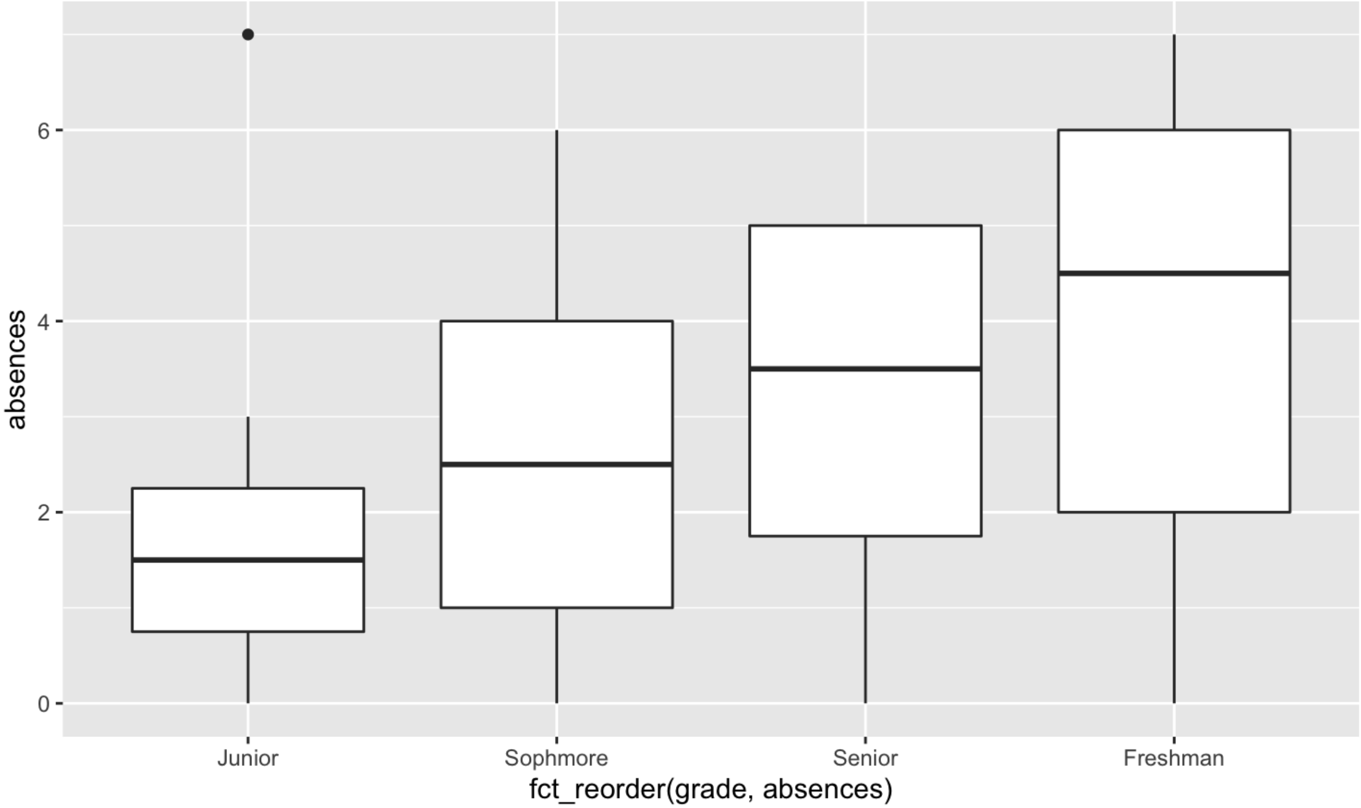
```
data_highschool_fct %>% group_by(grade) %>% summarise(mean = mean(absences))
```

```
## # A tibble: 4 x 2
##   grade     mean
##   <fct>    <dbl>
## 1 Freshman  4
## 2 Sophmore  2.62
## 3 Junior    2
## 4 Senior    3.12
```

Here we see that the mean is calculated in the order we would like only for the version of the data that has absences coded as a factor!

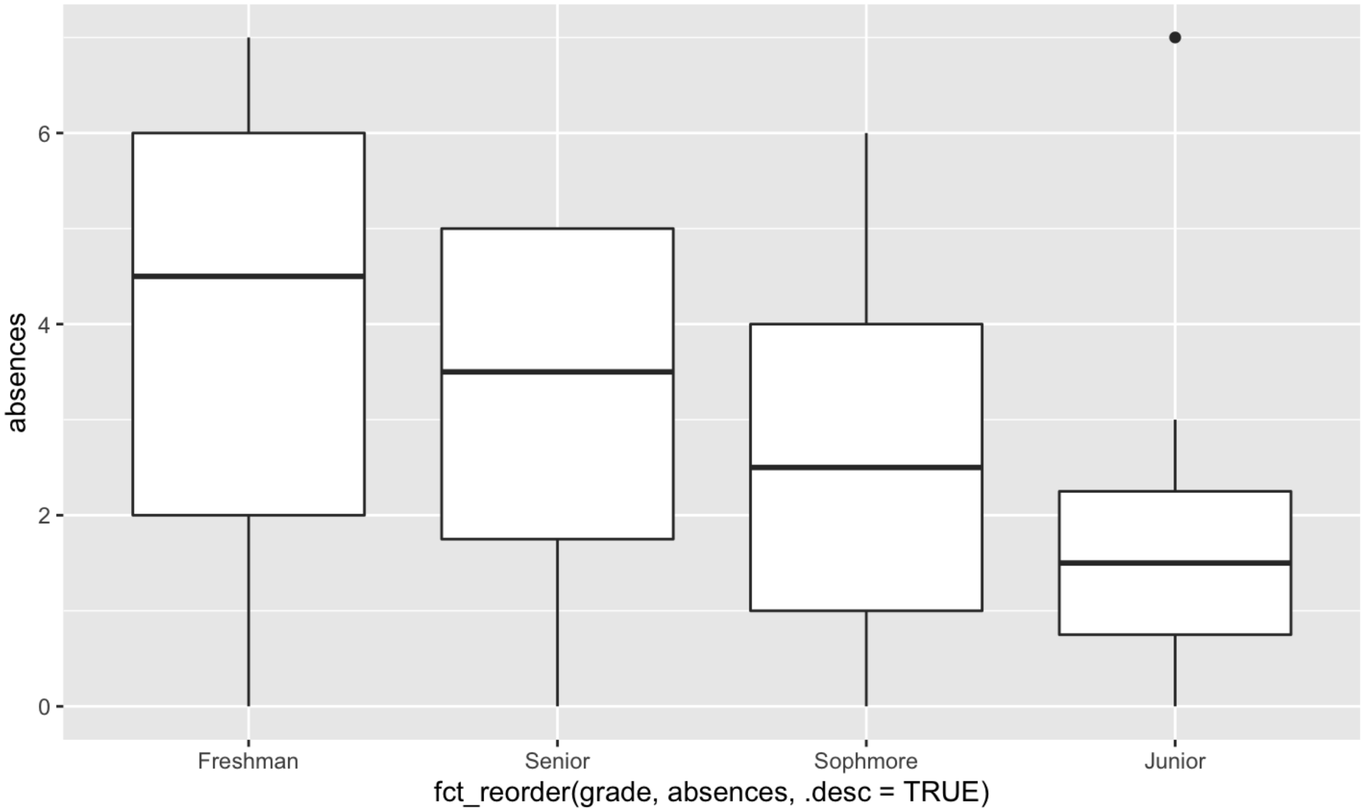
What if we want to change the factor level order?

```
#boxplot(data = data_highschool_fct, absences ~ fct_reorder(grade, absences))
data_highschool_fct %>%
  ggplot(mapping = aes(x = fct_reorder(grade, absences),
                       y = absences)) +
  geom_boxplot()
```



Descending factor order

```
#boxplot(data = data_highschool_fct, absences ~ fct_reorder(grade, absences, .desc = TRUE))
data_highschool_fct %>%
  ggplot(mapping = aes(x = fct_reorder(grade, absences, .desc = TRUE),
                          y = absences)) +
  geom_boxplot()
```



Claculations with reoder

```
data_highschool_fct %>% group_by(grade) %>% summarise(mean = mean(absences))
```

```
## # A tibble: 4 x 2
##   grade      mean
##   <fct>     <dbl>
## 1 Freshman   4
## 2 Sophmore   2.62
## 3 Junior     2
## 4 Senior     3.12
```

```
data_highschool_fct$grade <- fct_reorder(data_highschool_fct$grade,
                                         data_highschool_fct$absences,
                                         .desc = TRUE)
data_highschool_fct %>% group_by(grade) %>% summarise(mean = mean(absences))
```

```
## # A tibble: 4 x 2
##   grade      mean
##   <fct>     <dbl>
## 1 Freshman   4
## 2 Senior     3.12
## 3 Sophmore   2.62
## 4 Junior     2
```