Doing things over and over again: Functions and Loops

Don’t

Repeat

Yourself

Don’t
Repeat
Yourself

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = temp,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "temperature (C)")
```

Don’t Repeat Yourself

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = temp,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "temperature (C)")
```

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = sal,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "salinity (ppt)")
```

Don't Repeat Yourself

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = temp,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "temperature (C)")
```

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = sal,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "salinity (ppt)")
```

```
ggplot(wq_sub) +
  geom_boxplot(aes(x = state,
                   y = ph,
                   fill = state)) +
  theme(legend.position = "none") +
  labs(y = "temperature (C)")
```

Don’t

Repeat

Yourself

You might want to write a function if you’ve **copied and pasted** the same code at least **three times**

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)
27.14262
```

```
t.test(temp_fl, temp_ak)

##  Welch Two Sample t-test
##
## data:  temp_fl and temp_ak
## t = 13.785, df = 33.989, p-value = 1.774e-15
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  14.97478 20.15349
## sample estimates:
## mean of x mean of y
## 25.257922  7.693785
```

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)
27.14262
```

> "A function is simply a piece of code that is packaged in a way that makes it easy to use."
> -RStudio Cloud Primer "Write Functions"
> https://rstudio.cloud/learn/primers/6.1

```
t.test(temp_
```

```
##  Welch Two
##
## data:  temp
## t = 13.785,
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  14.97478 20.15349
## sample estimates:
## mean of x mean of y
## 25.257922  7.693785
```

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)
```

Name (actually a byproduct, not part of the function)

```
    {Magic Happens}

27.14262
```

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)
```

Name
Arguments

```
    {Magic Happens}


27.14262
```

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)

    {Magic Happens}

27.14262
```

Name
Arguments

Return Value – can be a single value, a data frame, a list….. anything you make it.

Good news: we've been using functions!

```
mean(wq$sal, na.rm = TRUE)
```

```
{Magic Happens}
```

```
27.14262
```

Name
Arguments

Body

Return Value

……and we can write our own.

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function
   - RStudio has a toolbar option to help with this
4. Make sure it works on new objects you want to feed it

Example: calculating standard error

1. Start with working code, that does something you want it to do

    a. In the script, I've taken a subset of the **wq** data frame and named it "test"
    b. I will start by calculating standard error of **test$temp_f**

$$\frac{standard\ deviation}{\sqrt{n}}$$

Example: calculating standard error

1. Start with working code, that does something you want it to do

    a. In the script, I've taken a subset of the **wq** data frame and named it "test"
    b. I will start by calculating standard error of **test$temp_f**

$$\frac{standard\ deviation}{\sqrt{n}}$$

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
sd(test$sal,    na.rm = TRUE) / sqrt( sum(!is.na(test$sal   )) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
sd(test$sal,    na.rm = TRUE) / sqrt( sum(!is.na(test$sal   )) )
sd(test$ph,     na.rm = TRUE) / sqrt( sum(!is.na(test$ph    )) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
    a. What am I changing when I'm copying this?
    b. I started by calculating standard error of **test$temp_f**
    c. Maybe next I want to check out **salinity**
    d. And then **pH**

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
sd(test$sal,    na.rm = TRUE) / sqrt( sum(!is.na(test$sal   )) )
sd(test$ph,     na.rm = TRUE) / sqrt( sum(!is.na(test$ph    )) )
sd(x,           na.rm = TRUE) / sqrt( sum(!is.na(x          )) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

```
sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

```
sterr <- function(x){
  sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
}
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

```
sterr <- function(x){
 sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
}
```

body

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

argument

```
sterr <- function(x){
  sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
}
```

Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

name

```
sterr <- function(x){
  sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
}
```

Example: calculating standard error

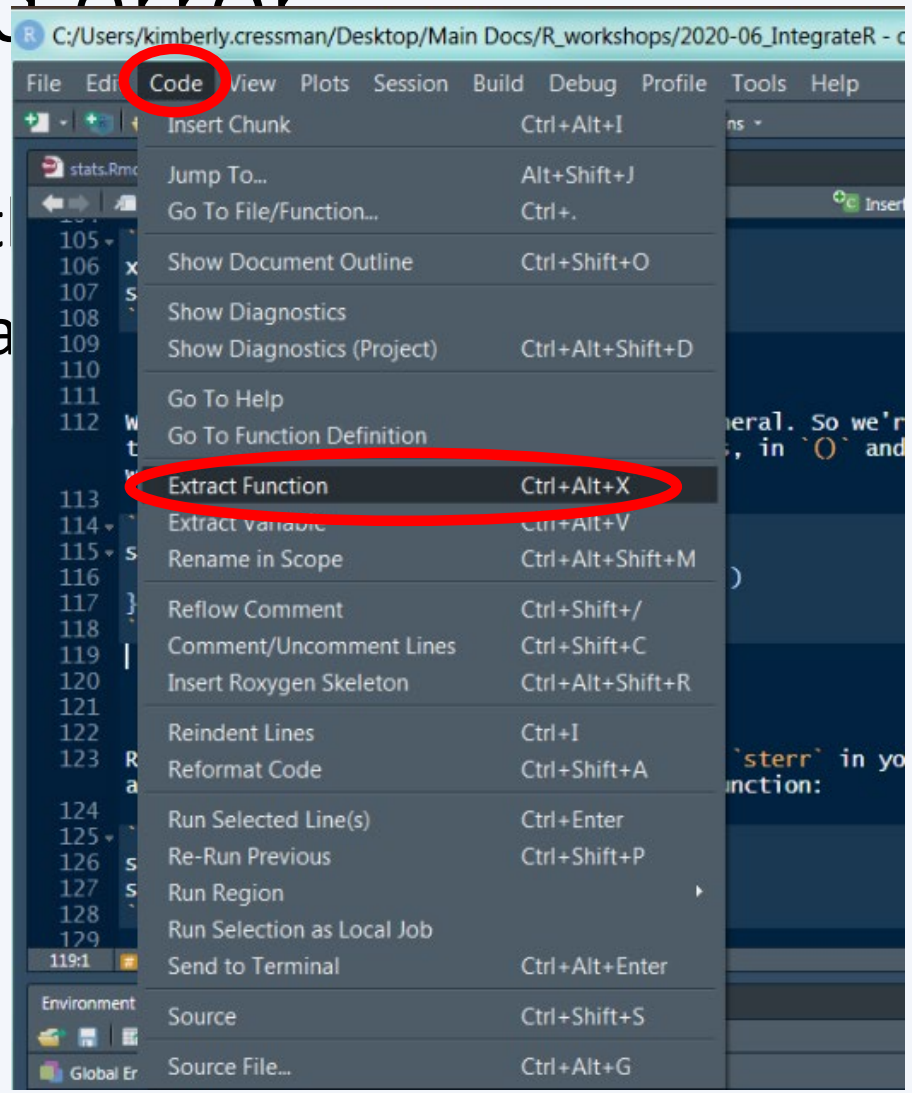
1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function

```
sterr <- function(x){
  sd(x, na.rm = TRUE) / sqrt( sum(!is.na(x)) )
}
```

Example: calculating standard error

1. Start with working code, that does somet
2. Figure out what parts of it need to genera
3. Format it as a function
   - RStudio has a toolbar option to help with this

```
sterr <- function(x){
  sd(x, na.rm = TRUE) / sqrt( sum(!is.na
}
```



Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function
4. Run it to get it into your environment as a working function
5. Make sure it works – on your original object and new ones

```
sd(test$temp_f, na.rm = TRUE) / sqrt( sum(!is.na(test$temp_f)) )
0.5139033

sterr(test$temp_f)
0.5139033
```

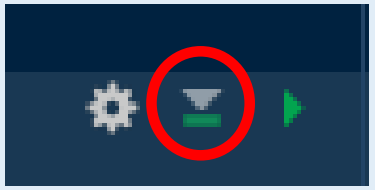
Example: calculating standard error

1. Start with working code, that does something you want it to do
2. Figure out what parts of it need to generalize
3. Format it as a function
4. Run it to get it into your environment as a working function
5. Make sure it works – on your original object and new ones

```
sterr(test$sal)
0.4054372
```

Your Turn 1: Write a function!

1. Navigate to “Your Turn 1” in the .Rmd file
2. Use this symbol to run all the prior code in the file:



3. Write a function, named `divide_by_10`
   a. It should take one argument, `x`, which can be a value or a vector
   b. It should divide `x` (or its members) by 10 and return the type of object you feed it (single value if you gave it a value; vector of values if you gave it a vector)
4. Turn that function into a more general one, named `divide_by`
   a. It should take two arguments, `x` and `y`
   b. `y` should be used as the value in the denominator

Your Turn 1: Answers

```
divide_by_10 <- function(x){
  x/10
}
```

```
divide_by <- function(x, y){
  x / y
}
```

Some advice from Hadley and Garrett

Hadley Wickham and Garrett Grolemund, "R for Data Science”:

- put **data** arguments first
- **detail** arguments should go at the end, and should usually have **default values**

Some advice from Hadley and Garrett

Hadley Wickham and Garrett Grolemund, "R for Data Science”:

- put **data** arguments first
- **detail** arguments should go at the end, and should usually have **default values**

```
divide_by <- function(x, y = 10){
  x / y
}
```

Some advice from Hadley and Garrett

Hadley Wickham and Garrett Grolemund, "R for Data Science”:

- put **data** arguments first
- **detail** arguments should go at the end, and should usually have **default values**

```
divide_by <- function(x, y = 10){
  x / y
}
```

```
divide_by(test$sal, 10)
1.143854 1.319896

divide_by(test$sal)
1.143854 1.319896

divide_by(test$sal, 100)
0.1143854 0.1319896
```

Some advice from Hadley and Garrett

Hadley Wickham and Garrett Grolemund, "R for Data Science”:

- put **data** arguments first
- **detail** arguments should go at the end, and should usually have **default values**

```
divide_by <- function(x, y = 10){
  x / y
}
```

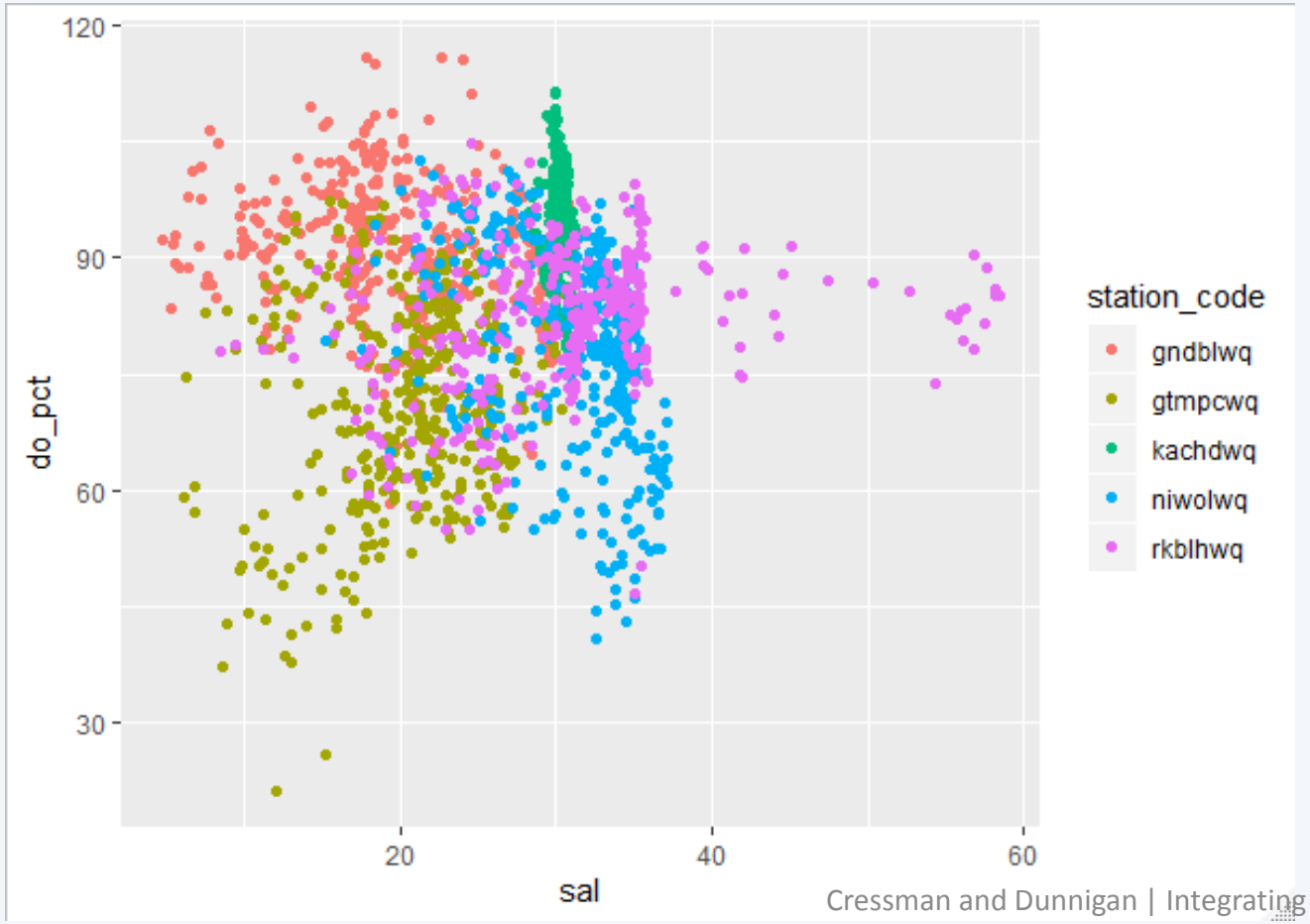
```
divide_by(test$sal, 10)
1.143854 1.319896

divide_by(test$sal)
1.143854 1.319896

divide_by(test$sal, 100)
0.1143854 0.1319896
```

Functions using the tidyverse

```
ggplot(wq_trimmed) +
  geom_point(aes(x = sal,
                 y = do_pct,
                 col = station_code))
```



Functions using the tidyverse

```
ggplot(wq_trimmed) +
  geom_point(aes(x = sal,
                 y = do_pct,
                 col = station_code))
```

```
my_plot <- function(wq_trimmed, sal,
                    do_pct, station_code) {
  ggplot(wq_trimmed) +
    geom_point(aes(x = sal,
                   y = do_pct,
                   col = station_code))
}
```

Functions using the tidyverse

```
my_plot <- function(wq_trimmed, sal,
                    do_pct, station_code) {
  ggplot(wq_trimmed) +
    geom_point(aes(x = sal,
                   y = do_pct,
                   col = station_code))
}

my_plot <- function(data, param1,
                    param2, param3) {
  ggplot(data) +
    geom_point(aes(x = param1,
                   y = param2,
                   col = param3))
}
```

Functions using the tidyverse

```
my_plot <- function(wq_trimmed, sal,
                    do_pct, station_code) {
  ggplot(wq_trimmed) +
    geom_point(aes(x = sal,
                   y = do_pct,
                   col = station_code))
}
```

```
my_plot <- function(data, param1,
                    param2, param3) {
  ggplot(data) +
    geom_point(aes(x = param1,
                   y = param2,
                   col = param3))
}
```

Functions using the tidyverse

```
my_plot <- function(data, param1,
                    param2, param3) {
  ggplot(data) +
    geom_point(aes(x = param1,
                   y = param2,
                   col = param3))
}
```

```
my_plot(wq_trimmed, sal, do_pct, station_code)
```

Functions using the tidyverse….. uh-oh

```
my_plot <- function(data, param1,
                    param2, param3) {
  ggplot(data) +
    geom_point(aes(x = param1,
```

```
> my_plot(wq_trimmed, sal, do_pct, station_code)
Error in FUN(X[[i]], ...) : object 'sal' not found
```

```
my_plot(wq_trimmed, sal, do_pct, station_code)
```

{{ Embrace }} the arguments

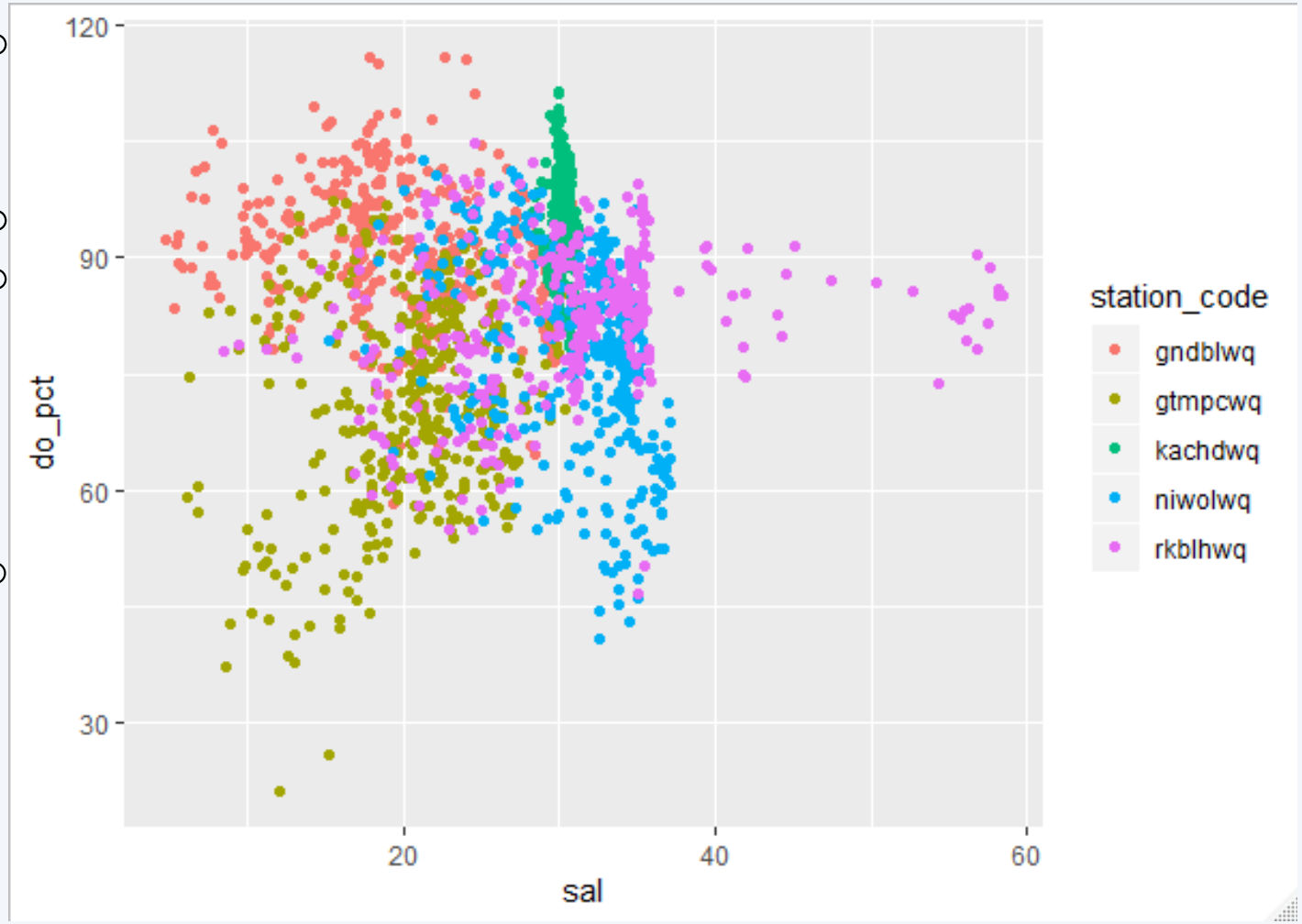
```
my_plot <- function(data, param1,
                    param2, param3) {
  ggplot(data) +
    geom_point(aes(x = {{ param1 }},
                   y = {{ param2 }},
                   col = {{ param3 }}))
}


my_plot(wq_trimmed, sal, do_pct, station_code)
```

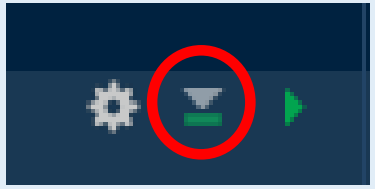
{{ Embrace }} the arguments

```
my_plot <- function(data, p
                    param2,
  ggplot(data) +
    geom_point(aes(x = {{ p
                   y = {{ p
                   col = {{
}


my_plot(wq_trimmed, sal, do
```



Your Turn 2: Improve the plotting function

1. Navigate to “Your Turn 2” in the .Rmd file
2. Again, use this symbol to run all the prior code: 
3. Add theme elements (like last week) into that `my_plot` function, to make it a plot you actually *want* to reproduce
   - it is okay if you want to reproduce a very ugly plot
4. Use your new function on at least two combinations of parameters
   - we like *do_pct* vs. *temp* and *do_pct* vs. *sal*
5. Paste your favorite one into the google doc.