Assignment 9

Instructions: Make an R Notebook and in it answer the question or questions below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook, probably an `html` or `pdf` file. An `html` file is easier for the grader to deal with. Do *not* hand in the Notebook itself. You want to show that you can (i) write code that will answer the questions, (ii) run that code and get some sensible output, (iii) write some words that show you know what is going on and that reflect your conclusions about the data. Your goal is to convince the grader that you *understand* what you are doing: not only doing the right thing, but making it clear that you know *why* it's the right thing.

Do *not* expect to get help on this assignment. The purpose of the assignments is for you to see how much *you* have understood. You will find that you also learn something from grappling with the assignments. The time to get help is after you watch the lectures and work through the problems from PASIAS, via tutorial and the discussion board, that is *before* you start work on the assignment. The only reasons to contact the instructor while working on an assignment are to report (i) something missing like a data file that cannot possibly be read, (ii) something *beyond your control* that makes it impossible to finish the assignment in time after you have started it.

There is a time limit on this assignment (you will see Quercus counting down the time remaining).

1. Forty students, some male and some female, measured their resting pulse rates. Then they marched in place for one minute and measured their pulse rate again. Our aim is to use regression to predict the pulse rate after the marching from the pulse rate before, and to see whether that is different for males and females. The data is in http://ritsokiguess.site/STAC32/pulsemarch.csv.

   (a) Read in and display (some of) the data.

Solution:

As usual:

```
my_url <- "http://ritsokiguess.site/STAC32/pulsemarch.csv"
march <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Sex = col_character(),
##   Before = col_double(),
##   After = col_double()
## )
```

```
march
```

```
## # A tibble: 40 x 3
##    Sex    Before After
##    <chr>   <dbl> <dbl>
##  1 Female     72    84
##  2 Male       60    72
##  3 Female     68    80
```

```
##  4 Male       70    72
##  5 Male       68    80
##  6 Male       61    75
##  7 Male       80    84
##  8 Male       72    76
##  9 Female     64    80
## 10 Female     62    92
## # ... with 30 more rows
```

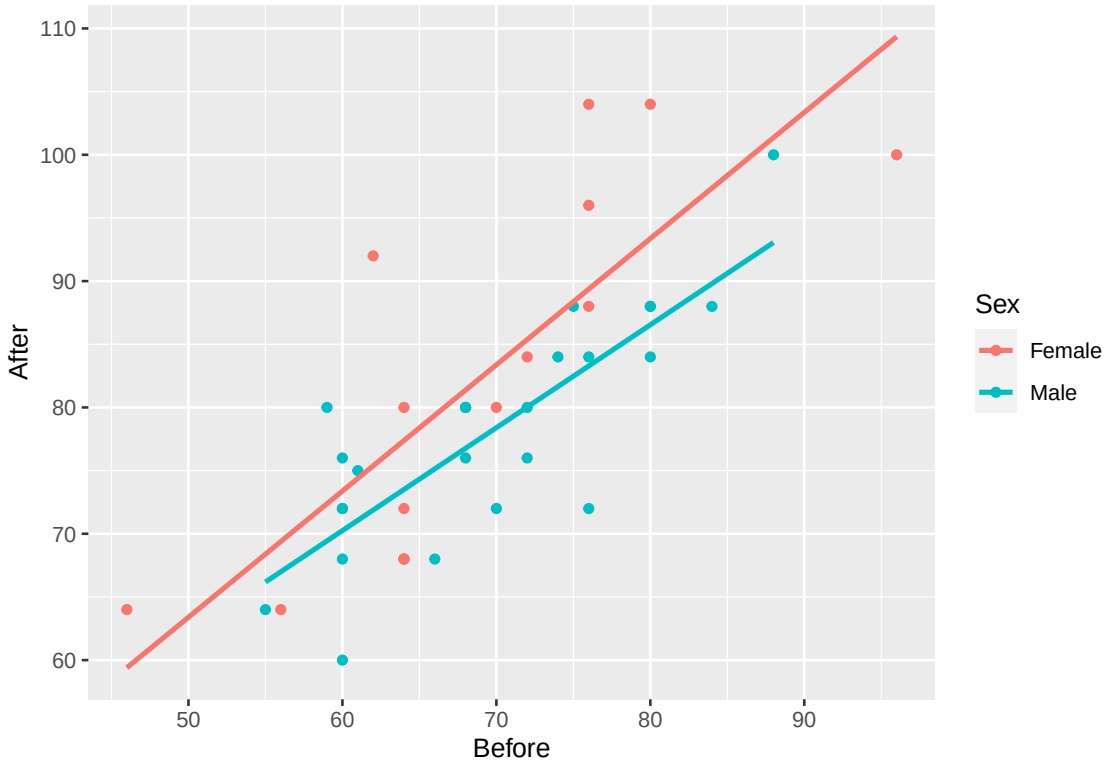
(b) Make a suitable graph using all three variables, adding appropriate regression line(s) to the plot.

Solution:

Two quantitative and one categorical says scatterplot, with colour distinguishing the categories (two here). `geom_smooth` adds a regression line to the plot for each `Sex`, which is what we want. I used `se=F` to remove the grey envelopes from the plot (because I thought they confused the issue):

```
ggplot(march, aes(x=Before, y=After, colour=Sex)) + geom_point() +
  geom_smooth(method = "lm", se=F)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

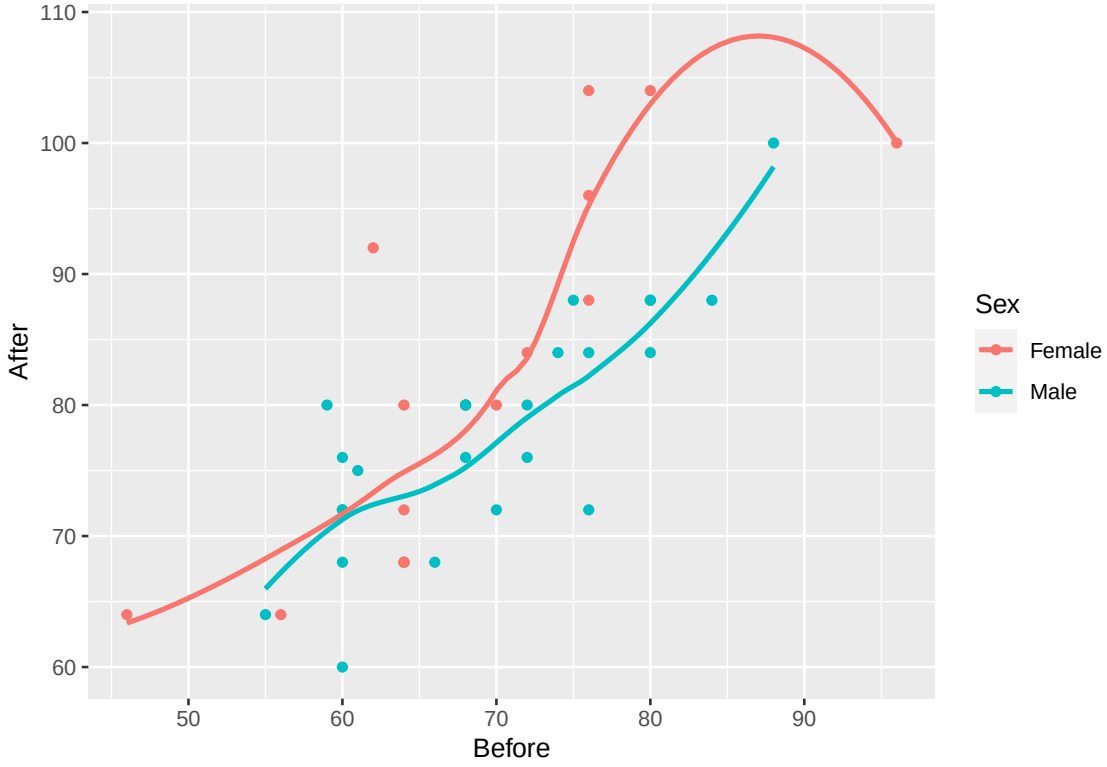


Having only one regression line is not so good because that only shows that pulse rate after goes up with pulse rate before, but not if and how the sexes differ.

Extra: I took a shortcut of the process here, to make the question shorter. In practice, what you'd do is to put smooth trends on the plot first:

```
ggplot(march, aes(x=Before, y=After, colour=Sex)) + geom_point() +
  geom_smooth(se=F)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The red trend looks curved, but if you look carefully, pretty much all of[1] the evidence for the curve comes from that point on the right with pulse rate before over 90 and pulse rate after around 100. If it weren't for that, the red trend would be pretty close to linear. As you'll recall, a decision about the kind of trend based on *one* observation is a pretty flimsy decision.

Then, having seen that the trends are not obviously curved, you would draw the plot with the straight lines. (Fitting separate *curves* is a whole different story that I didn't want to get into.)

(c) Explain briefly but carefully how any effects of pulse rate before on pulse rate after, and also of sex on pulse rate after, show up on your plot. (If either explanatory variable has no effect, explain how you know.)

Solution:

There is an upward trend, so if the pulse rate before is higher, so is the pulse rate after. This is true for both males and females. (Or, holding `Sex` fixed, that is, comparing two people of the same sex.)

The red line is always above the blue line, so at any given `Before` pulse rate, the `After` pulse rate for a female is predicted to be higher than that for a male.

Note that you have to be careful: when talking about the effect of each explanatory variable, you have to *hold the other one constant* (in general, hold all the other ones constant). If you

can word that in a way that makes sense in the context of the data you are looking at, so much the better.

(d) Run a regression predicting pulse rate after from the other two variables. Display the output.

Solution:

Thus:

```
march.1 <- lm(After~Before+Sex, data=march)
summary(march.1)
```

```
##
## Call:
## lm(formula = After ~ Before + Sex, data = march)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.8653  -4.6319  -0.4271   3.3856  16.0047
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  19.8003     7.9217   2.499   0.0170 *
## Before        0.9064     0.1127   8.046  1.2e-09 ***
## SexMale      -4.8191     2.2358  -2.155   0.0377 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.918 on 37 degrees of freedom
## Multiple R-squared:  0.6468, Adjusted R-squared:  0.6277
## F-statistic: 33.87 on 2 and 37 DF,  p-value: 4.355e-09
```

Extra: if you want "all the other variables except the response" as explanatory, there is also this shortcut:

```
march.1a <- lm(After~., data=march)
summary(march.1a)
```

```
##
## Call:
## lm(formula = After ~ ., data = march)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.8653  -4.6319  -0.4271   3.3856  16.0047
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  19.8003     7.9217   2.499   0.0170 *
## SexMale      -4.8191     2.2358  -2.155   0.0377 *
## Before        0.9064     0.1127   8.046  1.2e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 6.918 on 37 degrees of freedom
## Multiple R-squared:  0.6468, Adjusted R-squared:  0.6277
## F-statistic: 33.87 on 2 and 37 DF,  p-value: 4.355e-09
```

(e) Looking at your graph, does the significance (or lack of) of each of your two explanatory variables surprise you? Explain briefly.

Solution:

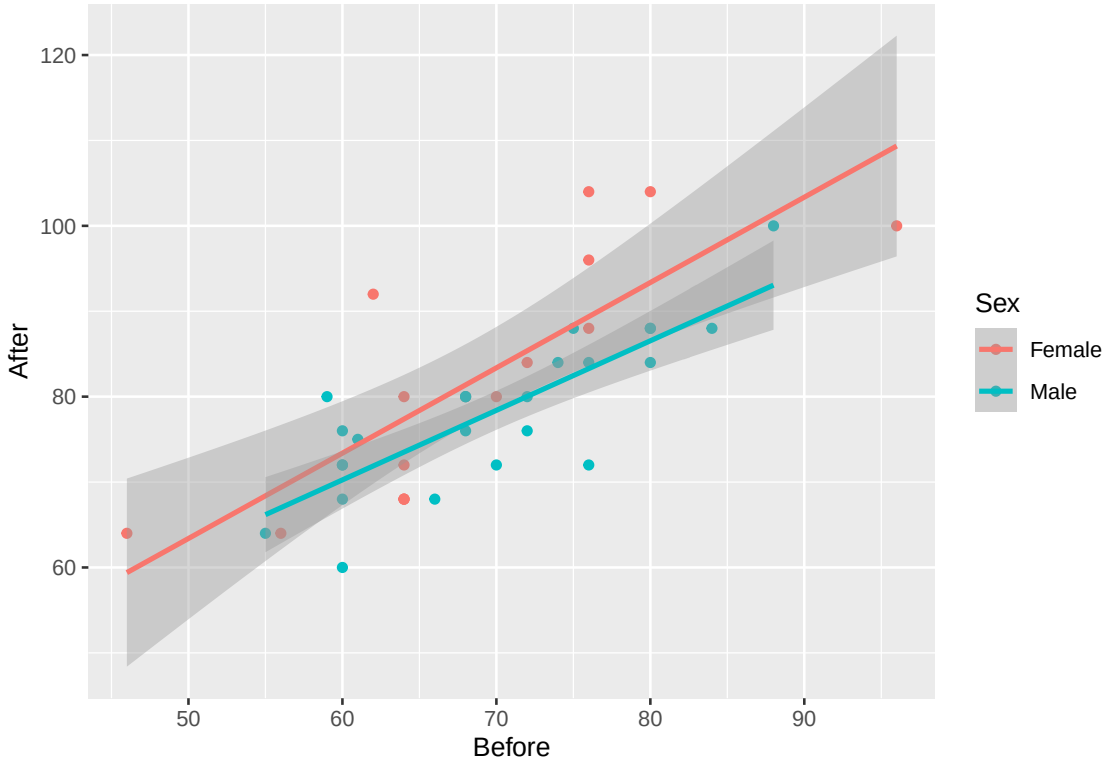
We noted a clear upward trend before, for both sexes, so there is no surprise that the `Before` pulse rate is significant.

The red dots (females) on the graph seemed to be on average above the blue ones (males), at least for similar before pulse rates. (This is not completely convincing, so you are entitled to be surprised also; note that the P-value, while significant, is not *that* small).

Extra: comparing the *lines* is less convincing, because how do we get a feel for whether these lines are more different than chance? One deceiving way to (fail to) get a feel for this is to re-draw our plot but with the grey envelopes:

```
ggplot(march, aes(x=Before, y=After, colour=Sex)) + geom_point() +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



The grey envelopes overlap substantially, which would make you think the lines are *not* significantly different. But, this is not the right way to compare the lines. It is a similar problem to that of comparing two *means* (that we would normally do with a two-sample test of some kind)

by working out the two *one*-sample confidence intervals, and seeing whether they overlap. If they *do not*, then you can be sure that the means differ, but if they do overlap, then you cannot say anything about whether the means differ: maybe they do, maybe they don't. This one is analogous; the grey envelopes overlap, so maybe the lines differ, maybe they don't. Looking at the grey envelopes in this case gives you *no* insight about whether males and females differ.

Here is a short discussion of this issue (in the context of comparing two means).

(f) What does the numerical value of the Estimate for `Sex` in your regression output mean, in the context of this data set? Explain briefly.

Solution:

The estimate is labelled `SexMale`, and its value is -4.8.

`Sex` is a categorical variable, so it has a baseline category, which is the first one, `Female`. The Estimate `SexMale` shows how males compare to the baseline (females), at a fixed `Before` pulse rate. This value is -4.8, so, at any `Before` pulse rate, the male `After` pulse rate is predicted to be 4.8 *less* than the female one.

I think you have to mention the value -4.8, so that you can talk intelligently about what it means for these data.

Extra: the implication of our model is that the predicted difference is the *same* all the way along. You might have your doubts about that; you might think the lines are closer together on the left and further apart on the right. Another way to think about this is whether the lines are *parallel*: that is, whether they have the same slope. I'm inclined to think they do; the data points are fairly scattered, and I think the slopes would have to be a lot more different to be significantly different. But you don't have to take my word for it: we can test this by adding an **interaction term** to the model. You might have seen this in ANOVA, where you are assessing the effect of one factor at different levels of the other. This is more or less the same idea. Note the `*` rather than the `+` in the model formula:[2]

```
march.2 <- lm(After~Before*Sex, data=march)
summary(march.2)
```

```
##
## Call:
## lm(formula = After ~ Before * Sex, data = march)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.2831  -4.3638  -0.3965   3.4077  16.6188
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     13.4390    11.1416   1.206    0.236
## Before           0.9991     0.1604   6.230 3.43e-07 ***
## SexMale          7.9470    15.8095   0.503    0.618
## Before:SexMale  -0.1846     0.2263  -0.816    0.420
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 6.949 on 36 degrees of freedom
## Multiple R-squared:  0.6532, Adjusted R-squared:  0.6243
## F-statistic:  22.6 on 3 and 36 DF,  p-value: 2.11e-08
```

The `Before:SexMale` term tests the interaction, and you see it is nowhere near significant. There is no justification for having lines with different slopes for males and females.

We were lucky here in that `Sex` has only two levels, so looking at the `summary` gave us what we wanted. If we had had an `Other` category for `Sex`, for people who don't identify with either male or female, there would be two Estimates in the `summary` table, one comparing Male with Female, and one comparing Other with Female.[3] But maybe the significant difference is Male vs. Other, and we would never see it.

To look for *any* effect of a categorical variable, the right way is to use `drop1`, to see which variables, including categorical ones, can be removed as a whole, thus:[4]

```
drop1(march.2, test="F")
```

```
## Single term deletions
##
## Model:
## After ~ Before * Sex
##             Df Sum of Sq    RSS    AIC F value Pr(>F)
## <none>                   1738.5 158.88
## Before:Sex  1    32.137 1770.7 157.61  0.6655   0.42
```

This *only* lists things that can be removed, in this case the interaction. It is not significant, so out it comes (resulting in our model `march.1`):

```
drop1(march.1, test="F")
```

```
## Single term deletions
##
## Model:
## After ~ Before + Sex
##        Df Sum of Sq    RSS    AIC F value    Pr(>F)
## <none>              1770.6 157.61
## Before  1   3097.98 4868.6 196.07  64.736 1.203e-09 ***
## Sex     1    222.34 1993.0 160.34   4.646    0.0377 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Both remaining explanatory variables are significant, so we need to keep them both.

Our categorical explanatory variable has only two levels, so `drop1` and `summary` give the exact same P-values.

Extra 2:

Let's go back and look at our data set again:

```
march
```

```
## # A tibble: 40 x 3
##    Sex    Before After
##    <chr>   <dbl> <dbl>
##  1 Female     72    84
##  2 Male       60    72
```

```
##  3 Female     68    80
##  4 Male       70    72
##  5 Male       68    80
##  6 Male       61    75
##  7 Male       80    84
##  8 Male       72    76
##  9 Female     64    80
## 10 Female     62    92
## # ... with 30 more rows
```

You might have been thinking, when we started, that these are before and after measurements *on the same people*, and so what we have here is matched pairs. So we do, but it's not the kind of matched pairs we are accustomed to. Let's begin by taking differences, getting rid of the `Before` and `After` columns, and see what we have left:

```
march %>%
  mutate(difference=After-Before) %>%
  select(-After, -Before) -> march_paired
march_paired
```

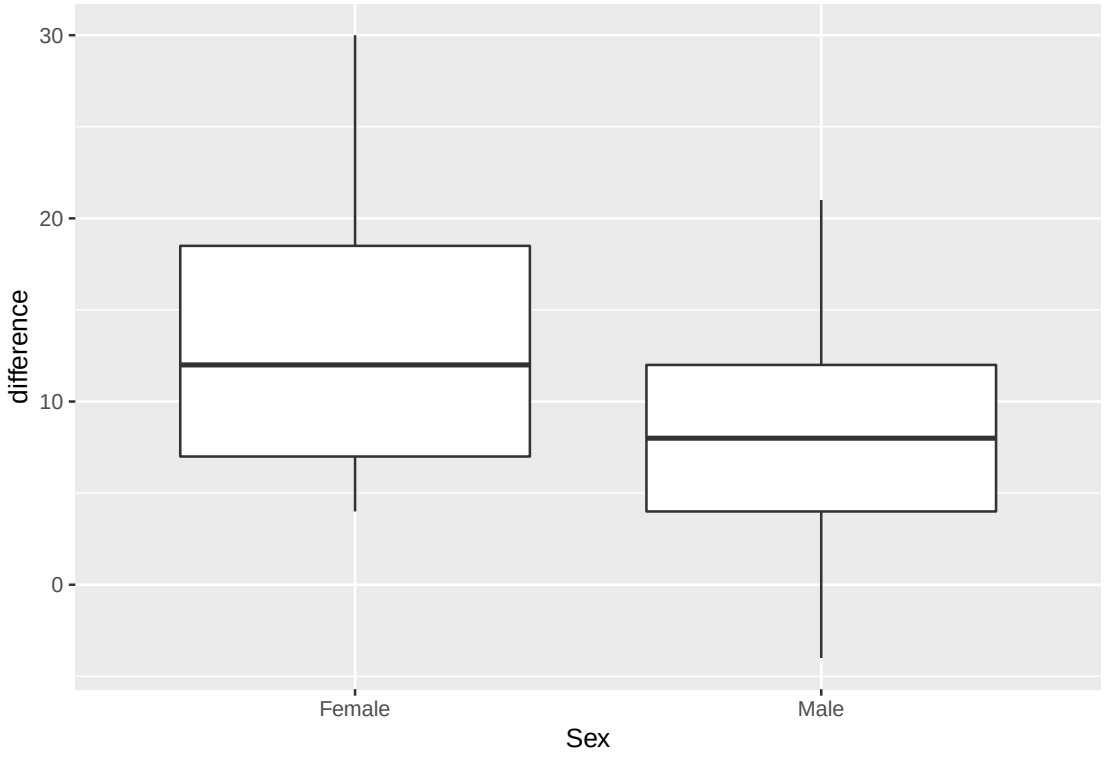
```
## # A tibble: 40 x 2
##    Sex    difference
##    <chr>       <dbl>
##  1 Female         12
##  2 Male           12
##  3 Female         12
##  4 Male            2
##  5 Male           12
##  6 Male           14
##  7 Male            4
##  8 Male            4
##  9 Female         16
## 10 Female         30
## # ... with 30 more rows
```

In matched pairs, we are used to having *one* column of differences, and we test that for a mean or median of zero, to express no difference between before and after (or whatever it was). But now, we have an extra column `Sex`. We are not interested here in whether the differences average out to zero;[5] we care more about whether the differences differ (!) between males and females. That is to say, we have a two-sample matched pairs test!

At this point, your head ought to be hurting!

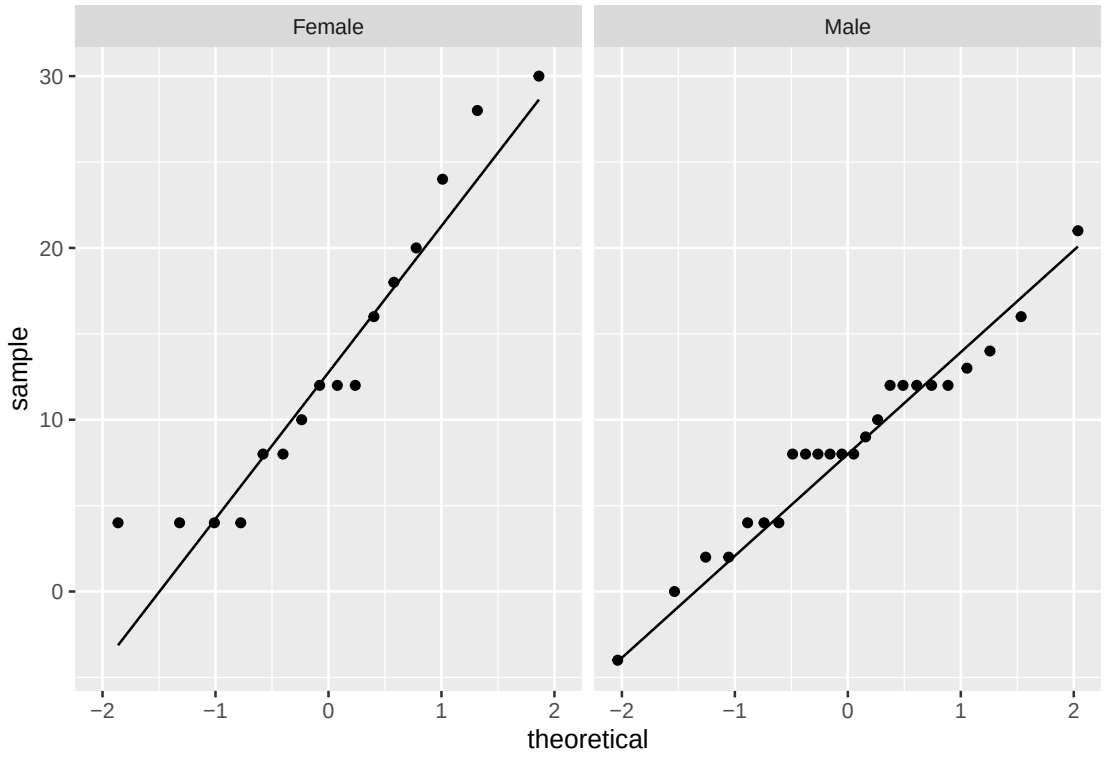
However, at this point what we are saying is that *if* you believe that the difference is a good way to summarize the effect of the exercise, then we have one measurement for each person, independent because different people's measurements will be independent. It doesn't matter where they came from. We have measurements on two groups, so some kind of two-sample test will be good. Which kind? Let's look at a graph, a good one now being a boxplot:

```
ggplot(march_paired, aes(x=Sex, y=difference)) + geom_boxplot()
```



Or, if you like, a facetted normal quantile plot:

```
ggplot(march_paired, aes(sample=difference)) +
  stat_qq() + stat_qq_line() +
  facet_wrap(~Sex)
```



It seems to me that these are normal enough for a t-test, given the sample sizes (feel free to disagree):

```
march_paired %>% count(Sex)
```

```
## # A tibble: 2 x 2
##   Sex        n
##   <chr>  <int>
## 1 Female    16
## 2 Male      24
```

The spreads look a bit different, so I think I would prefer the Welch test here:

```
t.test(difference~Sex, data=march_paired)
```

```
##
##  Welch Two Sample t-test
##
## data:  difference by Sex
## t = 2.0307, df = 23.296, p-value = 0.05386
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.08848652  9.92181986
## sample estimates:
## mean in group Female   mean in group Male
##            13.375000             8.458333
```

This time, there is not quite a significant difference between males and females. (The P-value is just the other side of 0.05.) Though the conclusion is different, the P-values are fairly similar.

Which test is better? I think treating it as matched pairs is assuming that the differences after minus before are the things to be looking at. This assumes that the after measurements are the before measurement plus a something that depends on treatment, but *not* on the before measurement. This would fail, for example, if all the after measurements are two times the before ones (so that the difference would be bigger if the before score was bigger). The regression approach is more flexible, because *any* linear relationship is taken care of. A matched-pairs model of this kind is a special case of the regression model but with the slope set to be 1. In our regression, the slope is less than 1, but not by much.

(there is a second question on the next page)

2. The coefficient of variation of a vector `x` is defined as the standard deviation of `x` divided by the mean of `x`.

   (a) Write a function called `cv` that calculates the coefficient of variation of its input and returns the result. You should use base R's functions that reliably compute the pieces that you need.

Solution:

I like to make a function skeleton first to be sure I remember all the pieces. You can do this by making a code chunk, typing `fun`, waiting a moment, and selecting the "snippet" that is offered to you. That gives you this, with your cursor on `name`:

```
name <- function(variables) {

}
```

Give your function a name (I asked you to use `cv`), hit tab, enter a name like x for the input, hit tab again, and then fill in the body of the function, to end up with something like this:

```
cv <- function(x) {
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
```

I think this is best, for reasons discussed below.

Some observations:

- in the last line of the function, you calculate something, and that something is the thing that gets returned. You *do not* need to save it in a variable, because then you have to return that variable by having its name alone on the last line.
- R has a `return()` function, but it is *bad style* to use it to return a value at the end of a function. The time to use it is when you test something early in the function and want to return early with a value like zero or missing without doing any actual calculation. Looking ahead a bit, you might want to return "missing" if the mean is zero *before* dividing by it, which you could do like this, this being *good* style:

```
cv2 <- function(x) {
  mean_x <- mean(x)
  if (mean_x == 0) return(NA)
  sd_x <- sd(x)
  sd_x/mean_x
}
```

- use the built-in `mean` and `sd` rather than trying to roll your own, because they have been tested by many users over a long time and they work. That's what I meant by "pieces" in the question.
- you might be tempted to do something like this:

```
cv3 <- function(x) {
  sd(x)/mean(x)
}
```

This will work, but it is not a good habit to get into, and thus not best as an answer to this question. This one line of code says three things: "work out the SD of `x`", "work out the mean

of `x`", and "divide them by each other". It is much clearer, for anyone else reading the code (including yourself in six months when you have forgotten what you were thinking) to have the three things one per line, so that anyone reading the code sees that each line does one thing and what that one thing is. There is a reason why production code and code golf are two very different things. Code that is easy to read is also easy to maintain, either by you or others.

- using something other than `mean` and `sd` as the names of your intermediate results is a good idea because these names are already used by R (as the names of the functions that compute the mean and SD). Redefining names that R uses can make other code behave unpredictably and cause hard-to-find bugs.[6]

(b) Use your function to find the coefficient of variation of the set of integers 1 through 5.

Solution:

This can be as simple as

```
cv(1:5)
```

```
## [1] 0.5270463
```

or, equally good, define this into a vector first:

```
y <- 1:5
cv(y)
```

```
## [1] 0.5270463
```

or, displaying a little less understanding, type the five numbers into the vector first (or directly as input to the function):

```
y <- c(1, 2, 3, 4, 5)
cv(y)
```

```
## [1] 0.5270463
```

```
cv(c(1, 2, 3, 4, 5))
```

```
## [1] 0.5270463
```

(c) Define a vector as follows:

```
v <- c(-2.8, -1.8, -0.8, 1.2, 4.2)
```

What is its coefficient of variation, according to your function? Does this make sense? Why did this happen? Explain briefly.

Solution:

Try it and see:

```
cv(v)
```

```
## [1] 6.248491e+16
```

A very large number, much bigger than any of the data values; with these human-sized numbers, we'd expect a human-sized coefficient of variation as well.

What actually happened was that the mean of `v` is this:

```
mean(v)
```

```
## [1] 4.440892e-17
```

Zero, or close enough, so that in calculating the coefficient of variation, we divided by something that was (almost) zero and got a result that was (almost) infinite.[7]

The possibility of getting a zero mean is why most people only calculate a coefficient of variation if all of the numbers are positive, which brings us to the next part:

(d) Most people only calculate a coefficient of variation if there are no negative numbers. Rewrite your function so that it gives an error if there are any negative numbers in the input, and test it with the vector `v` above. Hint: you might need to add `error=TRUE` to your chunk header to allow your document to preview/knit (inside the curly brackets at the top of the chunk, after a comma).

Solution:

This is a case for `stopifnot`, or of `if` coupled with `stop`. Check this up front, as the first thing you do before you calculate anything else. As to what to check, there are several possibilities:

- stop if any of the numbers are negative
- continue if all of the numbers are positive
- stop if the *smallest* number is negative (the smallest number is negative if and only if not all the numbers are positive)

R has functions `any` and `all` that do what you'd expect:

```
w <- 1:5
w
```

```
## [1] 1 2 3 4 5
```

```
any(w>3.5)
```

```
## [1] TRUE
```

```
all(w<4.5)
```

```
## [1] FALSE
```

Are there any numbers greater than 3.5 (yes, 4 and 5); are all the numbers less than 4.5 (no, 5 isn't).

Cite your sources for these if you use either of them, since this is the first place in the course that I'm mentioning either of them.

Remember that if you use `stopifnot`, the condition that goes in there is what has to be true if the function is to run; if you use `if` and `stop`, the condition is what will *stop* the function running. With that in mind, I would code my three possibilities above this way. First off, here's the original:

```
cv <- function(x) {
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
```

then, stop if any of the numbers are negative:

```
cv <- function(x) {
  if (any(x<0)) stop("A value is negative")
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): A value is negative
```

continue if all the numbers are positive

```
cv <- function(x) {
  stopifnot(all(x>0))
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): all(x > 0) is not TRUE
```

stop if the smallest value is negative

```
cv <- function(x) {
  if (min(x)<0) stop("Smallest value is negative")
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): Smallest value is negative
```

There are (at least) three other possibilities: you can negate the logical condition and interchange `if/stop` and `stopifnot`, thus (at the expense of some clarity of reading):

continue if it is not true that any of the numbers are negative

```
cv <- function(x) {
  stopifnot(!any(x<0))
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): !any(x < 0) is not TRUE
```

(you might be thinking of De Morgan's laws here)

stop if it is not true that all the numbers are positive

```
cv <- function(x) {
  if (!all(x>0)) stop("Not all values are positive")
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): Not all values are positive
```

continue if the smallest value is not negative

```
cv <- function(x) {
  stopifnot(min(x)>=0)
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): min(x) >= 0 is not TRUE
```

or another way to do the last one, a more direct negation of the condition, which at my guess needs some extra brackets:

```
cv <- function(x) {
  stopifnot(!(min(x)<0))
  mean_x <- mean(x)
  sd_x <- sd(x)
  sd_x/mean_x
}
cv(v)
```

```
## Error in cv(v): !(min(x) < 0) is not TRUE
```

This one is hard to parse: what does that last message mean? I would take a negative off each side and read it as "min of x is negative is TRUE", but that takes extra effort.

I said that last one needed some extra brackets. This is, I thought, to get the order of operations right (operator precedence); it turns out not to matter because "not" has lower precedence than most other things, so that these do actually work (the "not" is evaluated *after* the less-than and the other things, so last of all here, even though it appears to be "glued" to the `min`):

```
!min(v)<0
```

```
## [1] FALSE
```

```
!min(1:5)<0
```

```
## [1] TRUE
```

See this for details. See especially the second set of examples, the ones beginning with "Special operators", and see especially-especially the comment at the bottom of these examples! That is to say, *you* should put in the extra brackets unless you also make the case that they are not needed, because anyone reading your code is guaranteed to be confused by it when they read it (including you in six months, because you *will not* remember the operator priority of "not").

My take is that one of the first three of the seven possibilities for coding `stopifnot` or `if` with `stop` is the best, since these more obviously encode the condition for continuing or stopping as appropriate. There are two things here: one is that you have to get the code right, but the second is that you have to get the code *clear*, so that it is obvious to anyone reading it that it does the right thing (once again, this includes you in six months). On that score, the first three alternatives are a direct expression of what you want to achieve, and the last four make it look as if you found a way of coding it that worked and stopped there, without thinking about whether there were any other, clearer or more expressive, possibilities.

Notes

1. My mind just jumped to a former German soccer player by the name of Klaus Allofs.

2. To be precise, the * means "the main effects and the interaction together"; if you want to talk about just the interaction term, you denote it by ":"; note the "Before:SexMale" term in the summary table.

3. Female is the baseline, so everything gets compared with that, whether you like it or not.

4. The *test* piece says to do an F-test, which is different from without the quotes, which would mean not to do any tests, F without quotes meaning *FALSE*.

5. I think it's a given that pulse rates will be higher after exercise than before.

6. This is something I've done in the past and been bitten by, so I am trying to get myself not to do it any more.

7. In computing with decimal numbers, things are almost never exactly zero or exactly infinite; they are very small or very big. The mean here, which you would calculate to be zero, is less than "machine epsilon", which is about 10^{-16} in R (R works in "double precision"). A mean that small is, to the computer, indistinguishable from zero. It came out that way because the last value is added to a total that is by that point negative, and so you have a loss of accuracy because of subtracting nearly equal quantities. I learned all about this stuff in my first real computer science course, which I think was a "numerical math" course, some absurd number of years ago. It looks as if this gets taught in CSCC37 these days. See https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html for a properly detailed treatment.