Assignment 6

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. The plant called kudzu was imported to the US South from Japan. It is rich in isoflavones, which are believed to be beneficial for bones. In a study, rats were randomly assigned to one of three diets: one with a low dose of isoflavones from kudzu, one with a high dose, and a control diet with no extra isoflavones. At the end of the study, each rat's bone density was measured, in milligrams per square centimetre. The data as recorded are shown in http://ritsokiguess.site/STAC32/isoflavones.txt.[1] There are 15 observations for each treatment, and hence 45 altogether.

   Here are some code ideas you might need to use later, all part of the `tidyverse`. You may need to find out how they work.

   - `col_names` (in the `read_` functions)
   - `convert` (in various `tidyverse` functions)
   - `fill`
   - `na_if`
   - `rename`
   - `separate_rows`
   - `skip` (in the `read_` functions)
   - `values_drop_na` (in the `pivot_` functions)

   If you use any of these, *cite* the webpage(s) or other source(s) where you learned about them.

   (a) Take a look at the data file. Describe briefly what you see.

   Solution:

   The data values are (at least kind of) aligned in columns, suggesting `read_table`. There are up to six bone density values in each row, with a header that spans all of them (by the looks of it). The treatment column looks all right except that some of the rows are blank. The blank treatments are the same as the ones in the row(s) above them, you can infer, because there are

15 observations for each treatment, six, six, and then three. (This is how a spreadsheet is often laid out: blank means the same as the previous line.[2])

This, you might observe, will need some tidying.

(b) Read in the data, using `read_table`, and get it into a tidy form, suitable for making a graph. This means finishing with (at least) a column of treatments with a suitable name (the treatments will be text) and a column of bone density values (numbers), one for each rat. You can have other columns as well; there is no obligation to get rid of them. Describe your process clearly enough that someone new to this data set would be able to understand what you have done and reproduce it on another similar dataset. Before you begin, think about whether or not you want to keep the column headers that are in the data file or not. (It can be done either way, but one way is easier than the other.)

Solution:

The tidying part is a fair bit easier to see if you *do not* read the column headers. A clue to this is that `bone_mineral_density` is not aligned with the values (of bone mineral density) below it. The next question is how to do that. You might remember `col_names=FALSE` from when the data file has no column headers at all, but here it *does* have headers; we just want to skip over them. Keep reading in the documentation for `read_table`, and you'll find an option `skip` that does exactly that, leading to:

```
my_url <- "http://ritsokiguess.site/STAC32/isoflavones.txt"
bmd0a <- read_table(my_url, col_names = FALSE, skip = 1)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_character(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double()
## )
```

```
bmd0a
```

```
## # A tibble: 9 x 7
##   X1             X2    X3    X4    X5    X6    X7
##   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 "control"     228   207   234   220   217   228
## 2 ""            209   221   204   220   203   219
## 3 ""            218   245   210    NA    NA    NA
## 4 "low_dose"    211   220   211   233   219   233
## 5 ""            226   228   216   225   200   208
## 6 ""            198   208   203    NA    NA    NA
## 7 "high_dose"   250   237   217   206   247   228
## 8 ""            245   232   267   261   221   219
## 9 ""            232   209   255    NA    NA    NA
```

If you miss the `skip`, the first row of "data" will be those column headers that were in the

data file, and you really don't want that. The link https://readr.tidyverse.org/reference/read_delim.html talks about both `col_names` and `skip`.

This, however, is looking very promising. A pivot_longer will get those columns of numbers into one column, which we can call something like `bmd`, and …but, not so fast. What about those blank treatments in `X1`? The first two blank ones are `control`, the next two are `low_dose` and the last two are `high_dose`. How do we fill them in? The word "fill" might inspire you to read up on `fill`. Except that this doesn't quite work, because it replaces *missings* with the non-missing value above them, and we have blanks, not missings.

All right, can we replace the blanks with missings, and then `fill` those? This might inspire you to go back to the list of ideas in the question, and find out what `na_if` does: namely, exactly this! Hence:

```
bmd0a %>% mutate(X1=na_if(X1, "")) %>%
  fill(X1)
```

```
## # A tibble: 9 x 7
##   X1           X2    X3    X4    X5    X6    X7
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 control     228   207   234   220   217   228
## 2 control     209   221   204   220   203   219
## 3 control     218   245   210    NA    NA    NA
## 4 low_dose    211   220   211   233   219   233
## 5 low_dose    226   228   216   225   200   208
## 6 low_dose    198   208   203    NA    NA    NA
## 7 high_dose   250   237   217   206   247   228
## 8 high_dose   245   232   267   261   221   219
## 9 high_dose   232   209   255    NA    NA    NA
```

Run this one line at a time to see how it works. `fill` takes a column with missing values to replace, namely `X1`, and `na_if` takes two things: a column containing some values to make `NA`, and the values that should be made `NA`, namely the blank ones.

So that straightens out the treatment column. It needs renaming; you can do that now, or wait until later. I'm going to wait on that.

You need to organize the treatment column first, before you do the `pivot_longer`, or else that won't work.[3]

Now, we need to get one column of bone mass densities, instead of six. This you'll recognize as a standard `pivot_longer`, with one tweak: those missing values in `X5` through `X7`, which we want to get rid of. You might remember that this is what `values_drop_na` does:

```
bmd0a %>% mutate(X1=na_if(X1, "")) %>%
  fill(X1) %>%
  pivot_longer(X2:X7, names_to="old", values_to="bmd", values_drop_na=TRUE)
```

```
## # A tibble: 45 x 3
##    X1      old     bmd
##    <chr>   <chr> <dbl>
##  1 control X2      228
##  2 control X3      207
##  3 control X4      234
##  4 control X5      220
##  5 control X6      217
```

```
##  6 control X7      228
##  7 control X2      209
##  8 control X3      221
##  9 control X4      204
## 10 control X5      220
## # ... with 35 more rows
```

If you didn't think of `values_drop_na`, do the pivot without, and then check that you have too many rows because the missings are still there (there are 45 rats but you have 54 rows), so add a `drop_na()` to the end of your pipe. The only missing values are in the column I called `bmd`.

This is almost there. We have a numeric column of bone mass densities, a column called `old` that we can ignore, and a treatment column with a stupid name that we can fix. I find `rename` backwards: the syntax is new name equals old name, so you start with the name that doesn't exist yet and finish with the one you want to get rid of:

```
bmd0a %>% mutate(X1=na_if(X1, "")) %>%
  fill(X1) %>%
  pivot_longer(X2:X7, names_to="old", values_to="bmd", values_drop_na=TRUE) %>%
  rename(treatment=X1) -> bmd1b
bmd1b
```

```
## # A tibble: 45 x 3
##    treatment old     bmd
##    <chr>     <chr> <dbl>
##  1 control   X2      228
##  2 control   X3      207
##  3 control   X4      234
##  4 control   X5      220
##  5 control   X6      217
##  6 control   X7      228
##  7 control   X2      209
##  8 control   X3      221
##  9 control   X4      204
## 10 control   X5      220
## # ... with 35 more rows
```

Done!

The best way to describe this kind of work is to run your pipeline up to a point that needs explanation, describe what comes next, and then run the *whole pipeline* again up to the next point needing explanation, rinse and repeat. (This avoids creating unnecessary temporary dataframes, since the purpose of the pipe is to avoid those.)

The guideline for description is that if *you* don't know what's going to happen next, your reader won't know either. For me, that was these steps:

- read the data file without row names and see how it looks
- fix up the treatment column (convincing myself and the reader that we were now ready to pivot-longer)
- do the `pivot_longer` and make sure it worked
- rename the treatment column

So, I said there was another way. This happens to have a simple but clever solution. It starts from wondering "what happens if I read the data file *with* column headers, the normal way?

Do it and find out:

```
my_url <- "http://ritsokiguess.site/STAC32/isoflavones.txt"
bmd0b <- read_table(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   treatment = col_character(),
##   bone_mineral_density = col_character()
## )
```

```
bmd0b
```

```
## # A tibble: 9 x 2
##   treatment   bone_mineral_density
##   <chr>       <chr>
## 1 "control"   228 207 234 220 217 228
## 2 ""          209 221 204 220 203 219
## 3 ""          218 245 210
## 4 "low_dose"  211 220 211 233 219 233
## 5 ""          226 228 216 225 200 208
## 6 ""          198 208 203
## 7 "high_dose" 250 237 217 206 247 228
## 8 ""          245 232 267 261 221 219
## 9 ""          232 209 255
```

This looks … strange. There are two column headers, and so there are two columns. It so happened that this worked because the text `bone_mineral_density` is long enough to span all the columns of numbers. That second column is actually *text*: six or three numbers as text with spaces between them.

The first thing is, as before, to fill in the missing treatments, which is as above, but changing some names:

```
bmd0b %>% mutate(treatment=na_if(treatment, "")) %>%
  fill(treatment)
```

```
## # A tibble: 9 x 2
##   treatment bone_mineral_density
##   <chr>     <chr>
## 1 control   228 207 234 220 217 228
## 2 control   209 221 204 220 203 219
## 3 control   218 245 210
## 4 low_dose  211 220 211 233 219 233
## 5 low_dose  226 228 216 225 200 208
## 6 low_dose  198 208 203
## 7 high_dose 250 237 217 206 247 228
## 8 high_dose 245 232 267 261 221 219
## 9 high_dose 232 209 255
```

The way we learned in class for dealing with this kind of thing is `separate`. It is rather unwieldy here since we have to split `bone_mineral_density` into six (temporary) things:

```
bmd0b %>% mutate(treatment=na_if(treatment, "")) %>%
  fill(treatment) %>%
  separate(bone_mineral_density, into = c("z1", "z2", "z3", "z4", "z5", "z6"))
```

```
## Warning: Expected 6 pieces. Missing pieces filled with `NA` in 3 rows [3, 6, 9].
```

```
## # A tibble: 9 x 7
##   treatment z1    z2    z3    z4    z5    z6
##   <chr>     <chr> <chr> <chr> <chr> <chr> <chr>
## 1 control   228   207   234   220   217   228
## 2 control   209   221   204   220   203   219
## 3 control   218   245   210   <NA>  <NA>  <NA>
## 4 low_dose  211   220   211   233   219   233
## 5 low_dose  226   228   216   225   200   208
## 6 low_dose  198   208   203   <NA>  <NA>  <NA>
## 7 high_dose 250   237   217   206   247   228
## 8 high_dose 245   232   267   261   221   219
## 9 high_dose 232   209   255   <NA>  <NA>  <NA>
```

This works, though if you check, there's a warning that some of the rows don't have six values. However, these have been replaced by missings, which is just fine. From here, we do exactly what we did before: pivot-longer all the columns I called z-something, and get rid of the missings.

Having thought of `separate`, maybe you're now wondering what `separate_rows` does. It turns out that it bypasses the business of creating extra columns and then pivoting them longer, thus:

```
bmd0b %>% mutate(treatment=na_if(treatment, "")) %>%
  fill(treatment)  %>%
  separate_rows(bone_mineral_density, convert = TRUE) -> bmd1a
bmd1a
```

```
## # A tibble: 45 x 2
##    treatment bone_mineral_density
##    <chr>                    <int>
##  1 control                    228
##  2 control                    207
##  3 control                    234
##  4 control                    220
##  5 control                    217
##  6 control                    228
##  7 control                    209
##  8 control                    221
##  9 control                    204
## 10 control                    220
## # ... with 35 more rows
```

Boom! This takes all the things in that mess in `bone_mineral_density`, splits them up into individual data values, and puts them one per row back into the same column. The `convert` is needed because otherwise the values in the second column would be text and you wouldn't be able to plot them. (If you don't see that, use a mutate to convert the column into the numerical version of itself.)

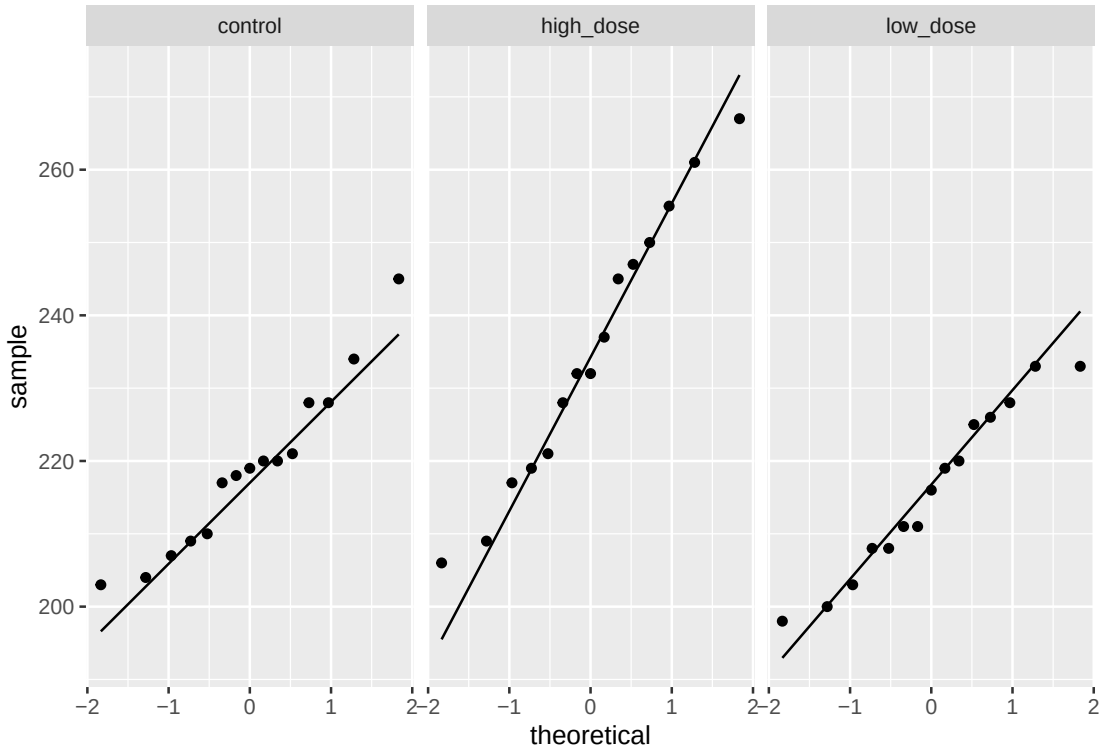
(c) The statistician on this study is thinking about running an ordinary analysis of variance to compare

the bone mineral density for the different treatments. Obtain a plot from your tidy dataframe that will help her decide whether that is a good idea.

Solution:

The key issues here are whether the values within each treatment group are close enough to normally distributed, and, if they are, whether the spreads are close enough to equal. The best plot is therefore a normal quantile plot of each of the three groups, in facets. You can do this *without* `scales="free"`:

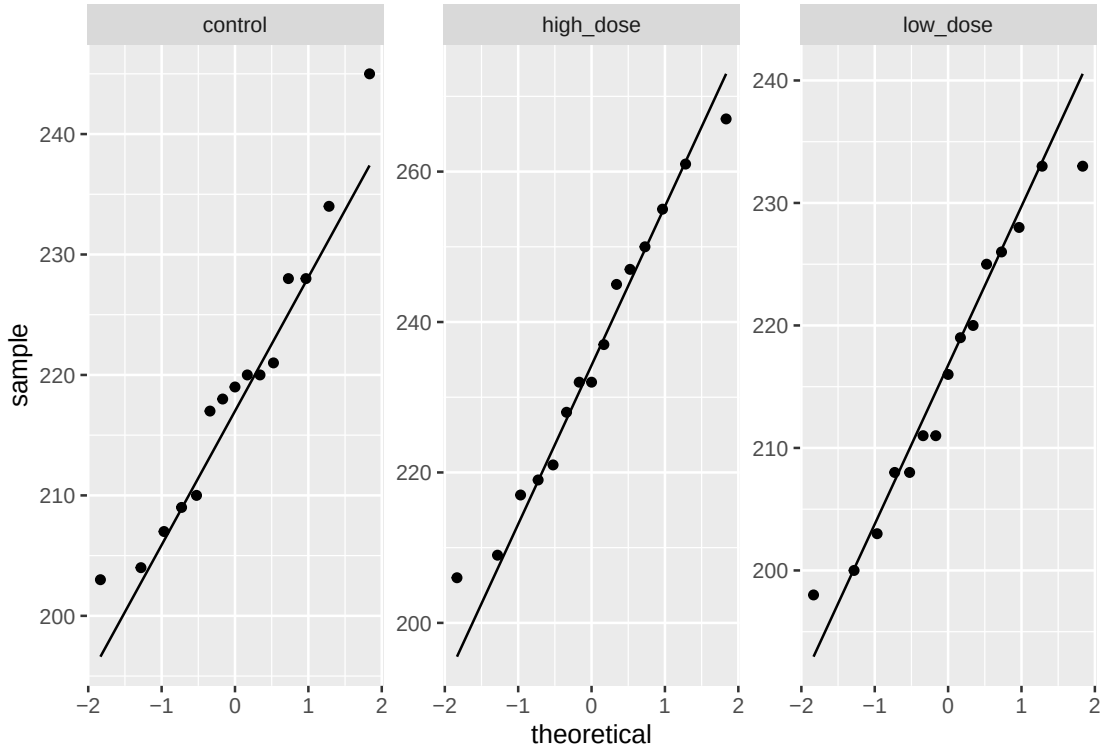
```
ggplot(bmd1b, aes(sample=bmd)) + stat_qq() + stat_qq_line() +
  facet_wrap(~treatment)
```



The value of doing it this way is that you also get a sense of variability, from the slopes of the lines, or from how much of each box is filled vertically. (Here, the high-dose values are more spread-out than the other two groups, which are similar in spread.)

You could also do it *with* `scales = "free"`:

```
ggplot(bmd1b, aes(sample=bmd)) + stat_qq() + stat_qq_line() +
  facet_wrap(~treatment, scales = "free")
```

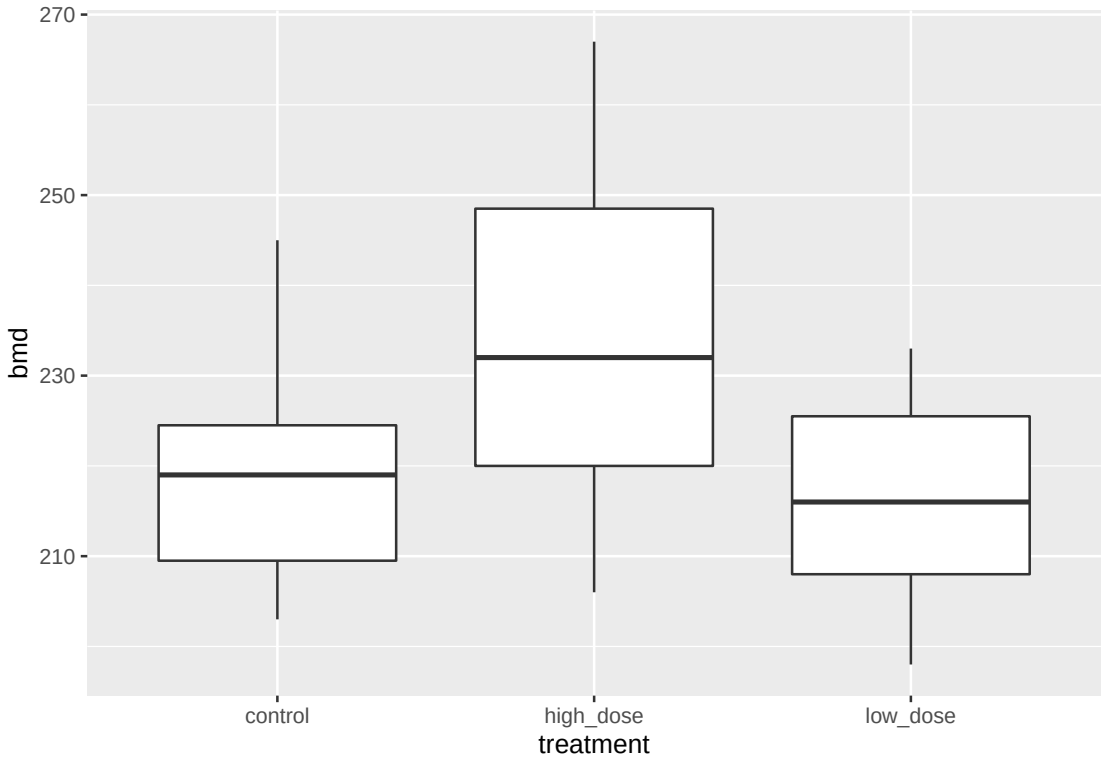


The value of doing it *this* way is that you fill the facets (what I called "not wasting real estate" on an earlier assignment), and so you get a better assessment of normality, but the downside is that you will need another plot, for example a boxplot (see below) to assess equality of spreads if you are happy with the normality.

I'm happy with either way of making the normal quantile plots, as long as you have a *reason* for your choice, coming from what you will be using the normal quantile plot for. You might not think of saying that here as you do it, but when you do the next part, you may realize that you need to assess equality of spreads, and in that case you should come back here and add a reason for using or not using `scales = "free"`.

The next-best graph here is boxplots:

```
ggplot(bmd1b, aes(x=treatment, y=bmd)) + geom_boxplot()
```



This is not so good because it doesn't address normality as directly (just giving you a general sense of shape). On the other hand, you can assess spread directly with a boxplot; see discussion above.

The grader is now probably thoroughly confused, so let me summarize possible answers in order of quality:

1. A normal quantile plot of all three groups, using `scales = "free"` or not, with a good reason. (If with `scales = "free"`, and there needs to be a comparison of spread, there needs to be a boxplot or similar below as well. That's what I meant by "any additional graphs" in the next part.)
2. A normal quantile plot of all three groups, using `scales = "free"` or not, *without* a good reason.
3. A side-by-side boxplot. Saying in addition that normality doesn't matter so much because we have moderate-sized samples of 15 and therefore that boxplots are good enough moves this answer up a place.

Note that getting the graph is (relatively) easy once you have the tidy data, but is impossible if you don't! This is the way the world of applied statistics works; without being able to get your data into the right form, you won't be able to do *anything* else. This question is consistent with that fact; I'm not going to give you a tidy version of the data so that you can make some graphs. The point of this question is to see whether you can get the data tidy enough, and if you can, you get the bonus of being able to do something straightforward with it.

(d) Based on your graph, and any additional graphs you wish to draw, what analysis would you recommend for this dataset? Explain briefly. (Don't do the analysis.)

Solution:

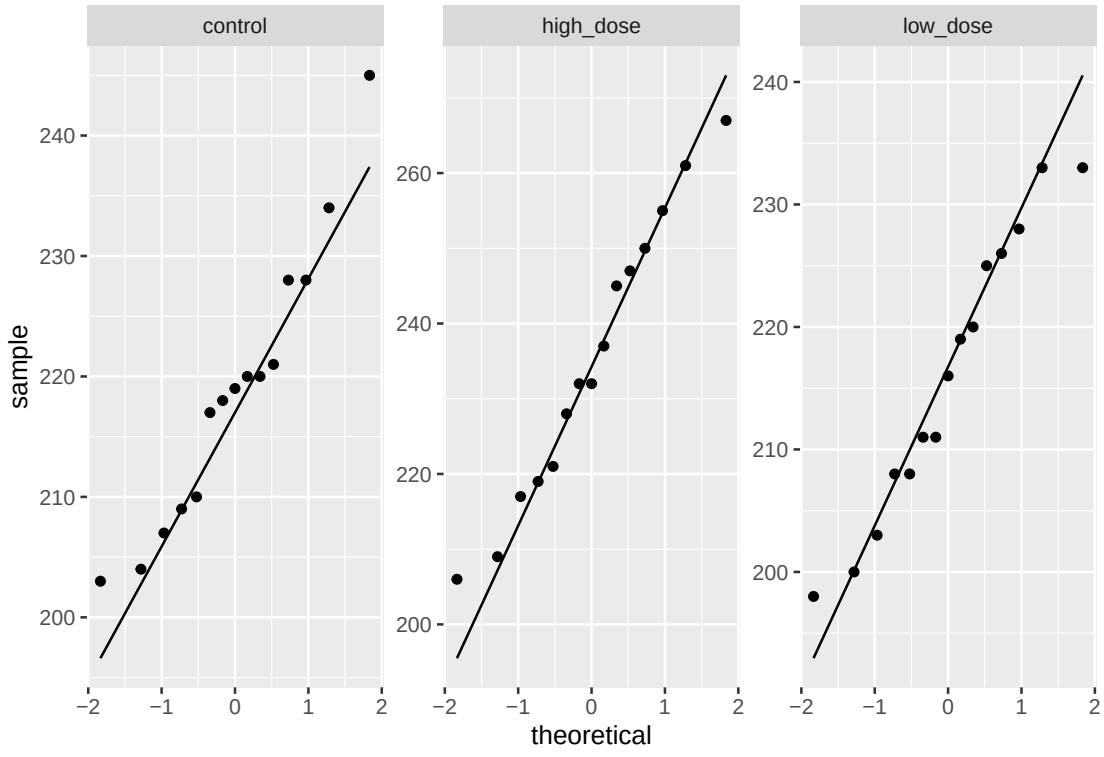
Make a decision about normality first. You need *all three* groups to be sufficiently normal. I don't think there's any doubt about the high-dose and low-dose groups; these are if anything *short*-tailed, which is not a problem for the ANOVA. You might find that the control group is OK too; make a call. Or you might find it skewed to the right, something suggested rather more by the boxplot. My take, from looking at the normal quantile plot, is that the highest value in the control group is a little too high, but with a sample size of 15, the Central Limit Theorem will take care of that. For yourself, you can find a bootstrapped sampling distribution of the sample mean for the control group and see how normal it looks.

If you are not happy with the normality, recommend Mood's median test.

If you are OK with the normality, you need to assess equal spreads. You can do this from a boxplot, where the high-dose group clearly has bigger spread. Or, if you drew normal quantile plots *without* `scales = "free"`, compare the slopes of the lines. This means that you need to recommend a Welch ANOVA.

If your normal quantile plots looked like this:

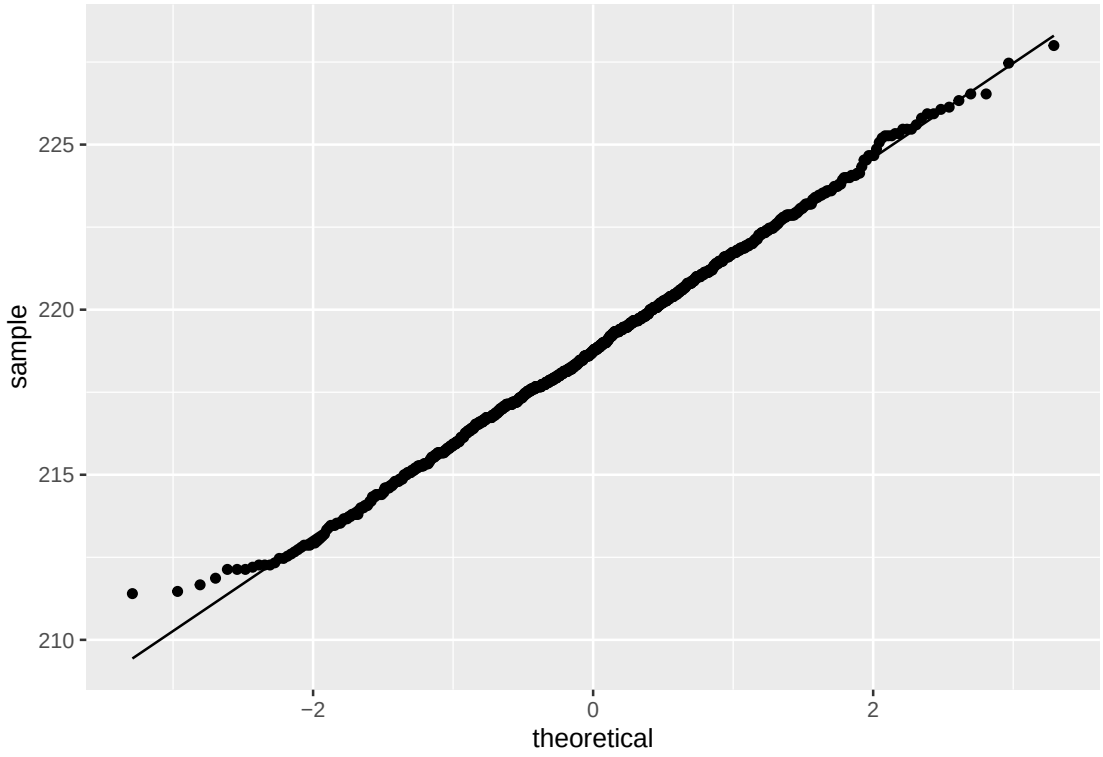
```
ggplot(bmd1b, aes(sample=bmd)) + stat_qq() + stat_qq_line() +
  facet_wrap(~treatment, scales = "free")
```



the only way to assess spread is to make another plot, and for this job, the boxplot is best.

Extra 1: the bootstrapped sampling distribution of the sample mean for the control group goes this way:

```
bmd1b %>%
  filter(treatment == "control") -> d
rerun(1000, sample(d$bmd, replace = TRUE)) %>%
  map_dbl(~mean(.)) %>%
  enframe() %>%
  ggplot(aes(sample = value)) + stat_qq() + stat_qq_line()
```



No problems there. The Welch ANOVA is fine.

Extra 2: You might be curious how the analysis comes out. Here is Welch:

```
oneway.test(bmd~treatment, data=bmd1b)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  bmd and treatment
## F = 5.6941, num df = 2.000, denom df = 27.075, p-value = 0.008627
```

Not all the same means, so use Games-Howell to explore:

```
gamesHowellTest(bmd~factor(treatment), data = bmd1b)
```

```
##
##  Pairwise comparisons using Games-Howell test

## data: bmd by factor(treatment)

##           control high_dose
## high_dose 0.0238  -
```

```
## low_dose  0.7680  0.0072

##
## P value adjustment method: none

## alternative hypothesis: two.sided
```

High dose is significantly different from both the other two, which are not significantly different from each other.

Mood's median test, for comparison:

```
median_test(bmd1b, bmd, treatment)
```

```
## $table
##            above
## group       above below
##   control       5     8
##   high_dose    11     4
##   low_dose      5     9
##
## $test
##        what     value
## 1 statistic 5.1018315
## 2        df 2.0000000
## 3   P-value 0.0780102
```

Not *any* significant differences, although it is a close thing.

The table of aboves and belows suggests the same thing as the Welch test: the high-dose values are mainly high, and the others are mostly low. But with these sample sizes it is not strong enough evidence. My guess is that the median test is lacking power compared to the Welch test; having seen that the Welch test is actually fine, it is better to use that here.[4]

Notes

1. Evidently the units were chosen for ease of recording; had the values been in grams instead, the person recording the data would have had to put a 0 and a decimal point on the front of each value. This is the old meaning of the word "coding"; making the data values be whole numbers and/or small deviations from something makes them easier to record, and in pre-computer days easier to calculate with. You will also see the same word used for classifying survey responses into categories, which is not quite the same thing.

2. It shouldn't be, but it often is.

3. Data tidying has a lot of this kind of thing: try something, see that it doesn't work, figure out what went wrong, fix that, repeat. The work you hand in won't necessarily look very much like your actual process.

4. This is the opposite way to the usual: when two tests disagree, it is usually the one with fewer assumptions that is preferred, but in this case, the Welch ANOVA is fine, and the median test fails to give significance because it is not using the data as efficiently.