Assignment 8.6

Nothing to hand in

The usual beginning:

```
library(tidyverse)
```

1. This is a reorganization of the crickets problem that you may have seen before (minus the data tidying).

   Male tree crickets produce "mating songs" by rubbing their wings together to produce a chirping sound. It is hypothesized that female tree crickets identify males of the correct species by how fast (in chirps per second) the male's mating song is. This is called the "pulse rate". Some data for two species of crickets are in `http://www.utsc.utoronto.ca/~butler/c32/crickets2.csv` as a CSV file. The columns are species (text), temperature, and pulse rate (numbers). This is the tidied version of the data set that the previous version of this question had you create.

   The research question is whether males of the different species have different average pulse rates. It is also of interest to see whether temperature has an effect, and if so, what.

   (a) Read the data into R and display what you have.

Solution: Nothing terribly surprising here:

```
my_url="http://www.utsc.utoronto.ca/~butler/c32/crickets2.csv"
crickets=read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##  species = col_character(),
##  temperature = col_double(),
##  pulse_rate = col_double()
## )
```

```
crickets
```

```
## # A tibble: 31 x 3
##          species temperature pulse_rate
##            <chr>       <dbl>      <dbl>
##  1 exclamationis        20.8       67.9
##  2 exclamationis        20.8       65.1
##  3 exclamationis        24.0       77.3
##  4 exclamationis        24.0       78.7
##  5 exclamationis        24.0       79.4
##  6 exclamationis        24.0       80.4
##  7 exclamationis        26.2       85.8
##  8 exclamationis        26.2       86.6
##  9 exclamationis        26.2       87.5
## 10 exclamationis        26.2       89.1
## # ... with 21 more rows
```

31 crickets, which is what I remember. What species are there?

```
crickets %>% count(species)
```

```
## # A tibble: 2 x 2
##         species     n
##           <chr> <int>
## 1 exclamationis    14
## 2        niveus    17
```

That looks good. We proceed.

(b) Do a two-sample t-test to see whether the mean pulse rates differ between species. What do you conclude?

Solution: Drag your mind way back to this:

```
t.test(pulse_rate~species,data=crickets)
```

```
##
##  Welch Two Sample t-test
##
## data:  pulse_rate by species
## t = 5.2236, df = 28.719, p-value = 1.401e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  14.08583 32.22677
## sample estimates:
## mean in group exclamationis        mean in group niveus
##                    85.58571                    62.42941
```

There is strong evidence of a difference in means (a P-value around 0.00001), and the confidence interval says that the mean chirp rate is higher for *exclamationis*. That is, not just for the crickets that were observed here, but for *all* crickets of these two species.

(c) Can you do that two-sample t-test as a regression?

Solution: Hang onto the "pulse rate depends on species" idea and try that in `lm`:

```
pulse.0=lm(pulse_rate~species,data=crickets)
summary(pulse.0)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.486  -9.458  -1.729  13.342  22.271
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)     85.586      3.316  25.807  < 2e-16 ***
## speciesniveus  -23.156      4.478  -5.171 1.58e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.41 on 29 degrees of freedom
## Multiple R-squared:  0.4797,Adjusted R-squared:  0.4617
## F-statistic: 26.74 on 1 and 29 DF,  p-value: 1.579e-05
```

I had to use "model 0" for this since I already have a `pulse.1` below and I didn't want to go down and renumber everything.

Look along the `speciesniveus` line. Ignoring the fact that it is negative, the t-statistic is almost the same as before (5.17 vs. 5.22) and so is the P-value (1.4×10^{-5} vs. 1.6×10^{-5}).

Why aren't they exactly the same? Regression is assuming equal variances everywhere (that is, within the two `species`), and before, we did the Welch-Satterthwaite test that does not assume equal variances. What if we do the pooled t-test instead?

```
t.test(pulse_rate~species,data=crickets,var.equal=T)
```

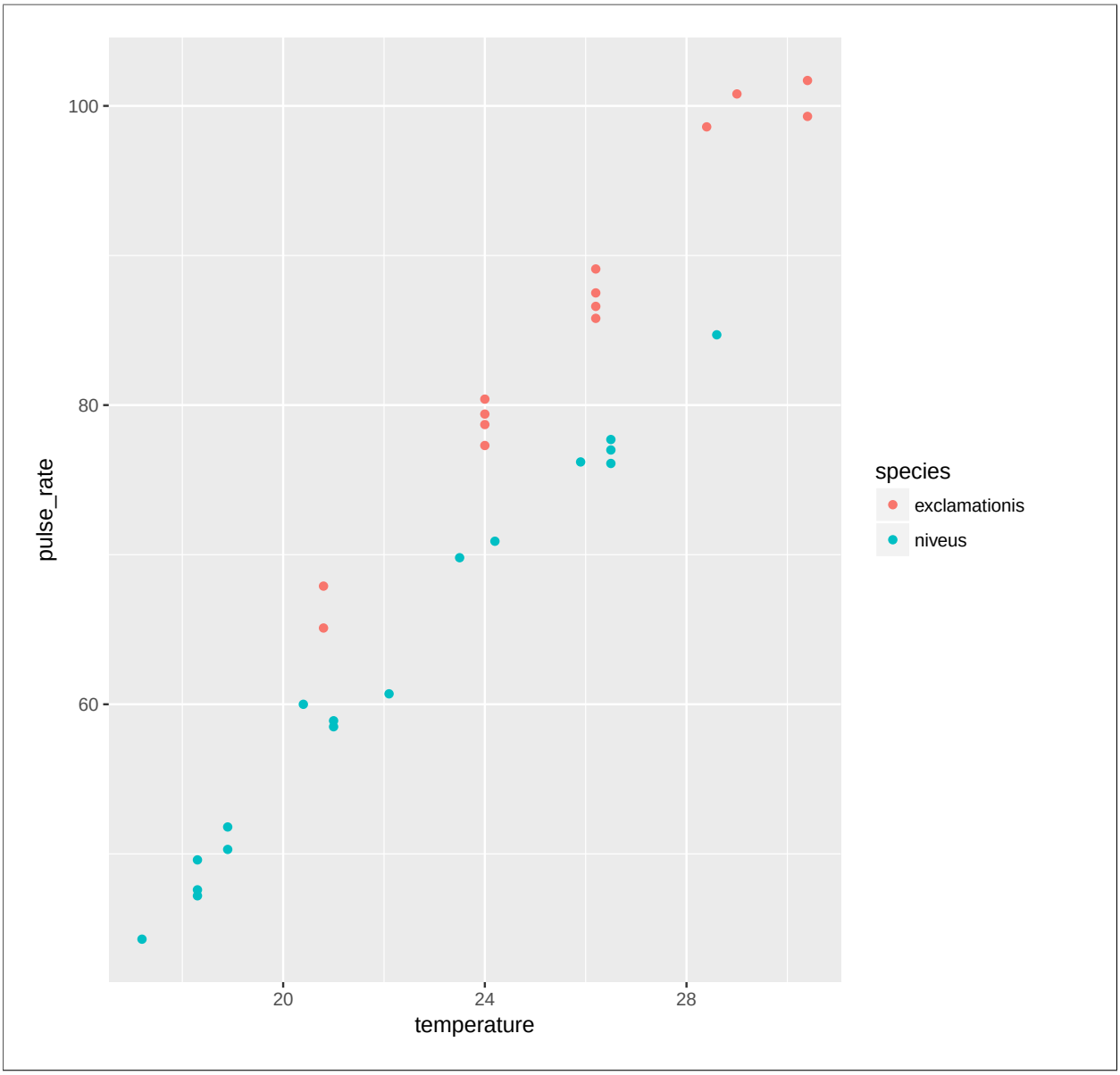
```
##
##  Two Sample t-test
##
## data:  pulse_rate by species
## t = 5.1706, df = 29, p-value = 1.579e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  13.99690 32.31571
## sample estimates:
## mean in group exclamationis        mean in group niveus
##                    85.58571                    62.42941
```

Now the regression and the t-test *do* give exactly the same answers. We'll think about that equal-spreads assumption again later.

(d) The analysis in the last part did not use temperature, however. Is it possible that temperature also has an effect? To assess this, draw a scatterplot of pulse rate against temperature, with the points distinguished, somehow, by the species they are from.[1]

Solution: One of the wonderful things about `ggplot` is that doing the obvious thing works:

```
ggplot(crickets,aes(x=temperature,y=pulse_rate,colour=species))+
  geom_point()
```



(e) What does the plot tell you that the t-test doesn't? How would you describe differences in pulse rates between species now?

Solution: The plot tells you that (for both species) as temperature goes up, pulse rate goes up as well. *Allowing for that*, the difference in pulse rates between the two species is even clearer than it was before. To see an example, pick a temperature, and note that the mean pulse rate at that temperature seems to be at least 10 higher for exclamationis, with a high degree of consistency.

The t-test mixed up all the pulse rates at all the different temperatures. Even though the conclusion was clear enough, it could be clearer if we incorporated temperature into the analysis.

There was also a potential source of unfairness in that the exclamationis crickets tended to be observed at higher temperatures than niveus crickets; since pulse rates increase with temperature, the apparent difference in pulse rates between the species might have been explainable by one species being observed

mainly in higher temperatures. This was *utterly invisible* to us when we did the t-test, but it shows the importance of accounting for all the relevant variables when you do your analysis.[2] If the species had been observed at opposite temperatures, we might have concluded[3] that niveus have the higher pulse rates on average. I come back to this later when I discuss the confidence interval for species difference that comes out of the regression model with temperature.

(f) Fit a regression predicting pulse rate from species and temperature. Compare the P-value for species in this regression to the one from the t-test. What does that tell you?

Solution: This is actually a so-called "analysis of covariance model", which properly belongs in D29, but it's really just a regression:

```
pulse.1=lm(pulse_rate~species+temperature,data=crickets)
summary(pulse.1)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species + temperature, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0128 -1.1296 -0.3912  0.9650  3.7800
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -7.21091    2.55094  -2.827  0.00858 **
## speciesniveus -10.06529    0.73526 -13.689 6.27e-14 ***
## temperature     3.60275    0.09729  37.032  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.786 on 28 degrees of freedom
## Multiple R-squared:  0.9896,Adjusted R-squared:  0.9888
## F-statistic:  1331 on 2 and 28 DF,  p-value: < 2.2e-16
```

The P-value for species is now 6.27×10^{-14} or 0.00000000000006, which is even less than the P-value of 0.00001 that came out of the t-test. That is to say, when you know temperature, you can be even more sure of your conclusion that there is a difference between the species.

The R-squared for this regression is almost 99%, which says that if you know both temperature and species, you can predict the pulse rate almost exactly.

In the regression output, the slope for species is about -10. It is labelled `speciesniveus`. Since species is categorical, `lm` uses the first category, exclamationis, as the baseline and expresses each other species relative to that. Since the slope is about -10, it says that at any given temperature, the mean pulse rate for niveus is about 10 less than for exclamationis. This is pretty much what the scatterplot told us.

We can go a little further here:

```
confint(pulse.1)
```

```
##                    2.5 %    97.5 %
## (Intercept)   -12.436265 -1.985547
## speciesniveus -11.571408 -8.559175
## temperature     3.403467  3.802038
```

The second line says that the pulse rate for *niveus* is between about 8.5 and 11.5 less than for *exclamationis*, at any given temperature (comparing the two species at the same temperature as each other, but that temperature could be anything). This is a lot shorter than the CI that came out of the t-test, that went from 14 to 32. This is because we are now accounting for temperature, which also makes a difference. (In the t-test, the temperatures were all mixed up). What we also see is that the t-interval is shifted up compared to the one from the regression. This is because the t-interval conflates[4] two things: the *exclamationis* crickets do have a higher pulse rate, but they were also observed at higher temperatures, which makes it look as if their pulse rates are more higher[5] than they really are, when you account for temperature.

This particular model constrains the slope with temperature to be the same for both species (just the intercepts differ). If you want to allow the slopes to differ between species, you add an interaction between temperature and species:

```
pulse.2=lm(pulse_rate~species*temperature,data=crickets)
summary(pulse.2)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species * temperature, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7031 -1.3417 -0.1235  0.8100  3.6330
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -11.0408     4.1515  -2.659    0.013 *
## speciesniveus               -4.3484     4.9617  -0.876    0.389
## temperature                  3.7514     0.1601  23.429   <2e-16 ***
## speciesniveus:temperature   -0.2340     0.2009  -1.165    0.254
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.775 on 27 degrees of freedom
## Multiple R-squared:  0.9901,Adjusted R-squared:  0.989
## F-statistic: 898.9 on 3 and 27 DF,  p-value: < 2.2e-16
```

To see whether adding the interaction term added anything to the prediction,[6] compare the model with and without using `anova`:

```
anova(pulse.1,pulse.2)
```

```
## Analysis of Variance Table
##
## Model 1: pulse_rate ~ species + temperature
## Model 2: pulse_rate ~ species * temperature
##   Res.Df    RSS Df Sum of Sq     F Pr(>F)
## 1     28 89.350
## 2     27 85.074  1    4.2758 1.357 0.2542
```

There's no significant improvement by adding the interaction, so there's no evidence that having different slopes for each species is necessary. This is the same interpretation as any `anova` for comparing two regressions: the two models are not significantly different in fit, so go with the simpler one, that is, the one without the interaction.

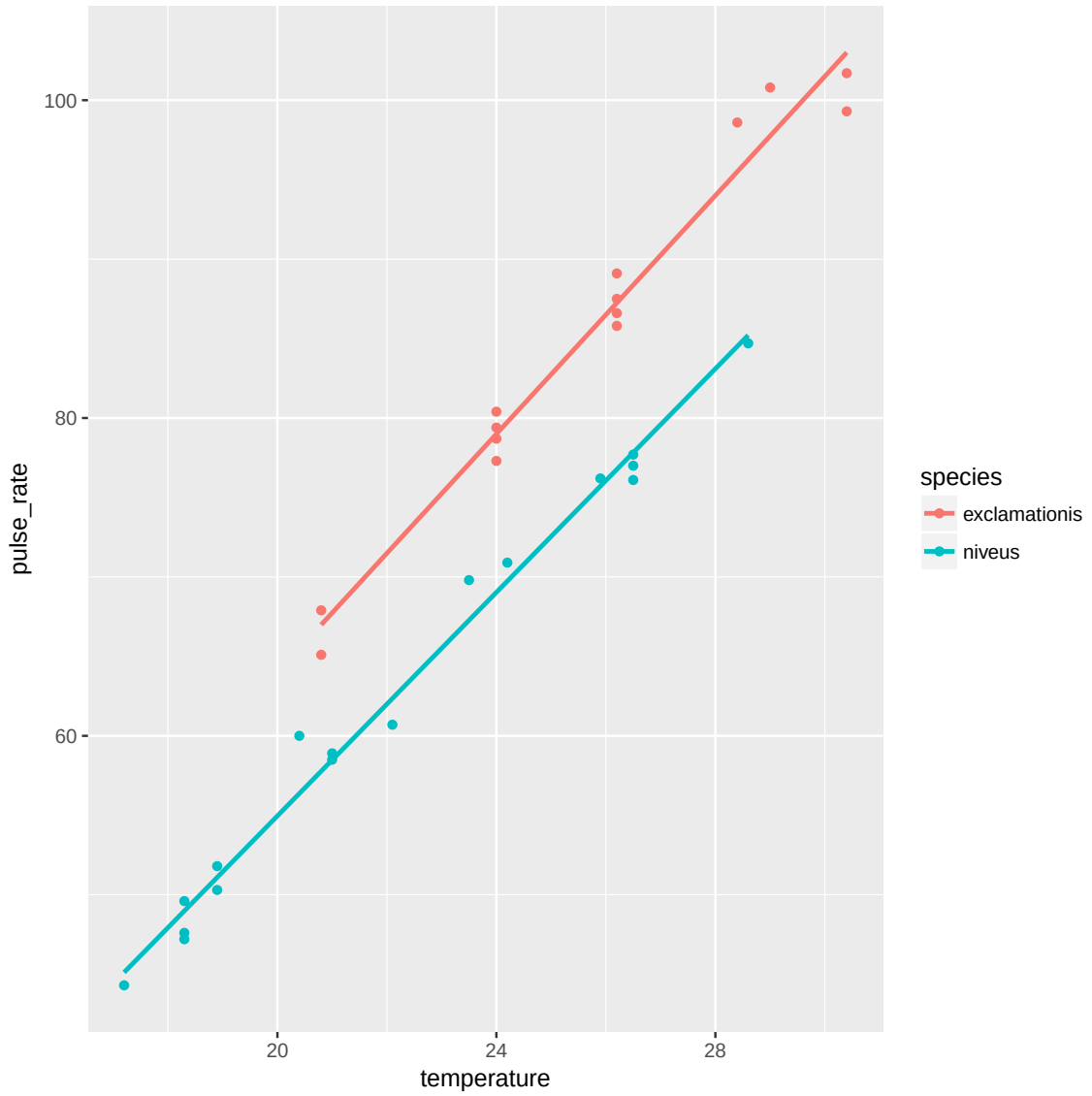
Note that `anova` gave the same P-value as did the t-test for the slope coefficient for the interaction in `summary`, 0.254 in both cases. This is because there were only two species and therefore only one slope coefficient was required to distinguish them. If there had been three species, we would have had to look at the `anova` output to hunt for a difference among species, since there would have been two slope coefficients, each with its own P-value.[7]

If you haven't seen interactions before, don't worry about this. The idea behind it is that we are testing whether we needed lines with different slopes and we concluded that we don't. Don't worry so much about the mechanism behind `pulse.2`; just worry about how it somehow provides a way of modelling two different slopes, one for each species, which we can then test to see whether it helps.

The upshot is that we do not need different slopes; the model `pulse.1` with the same slope for each species describes what is going on.

`ggplot` makes it almost laughably easy to add regression lines for each species to our plot, thus:

```
ggplot(crickets,aes(x=temperature,y=pulse_rate,colour=species))+
  geom_point()+geom_smooth(method="lm",se=F)
```

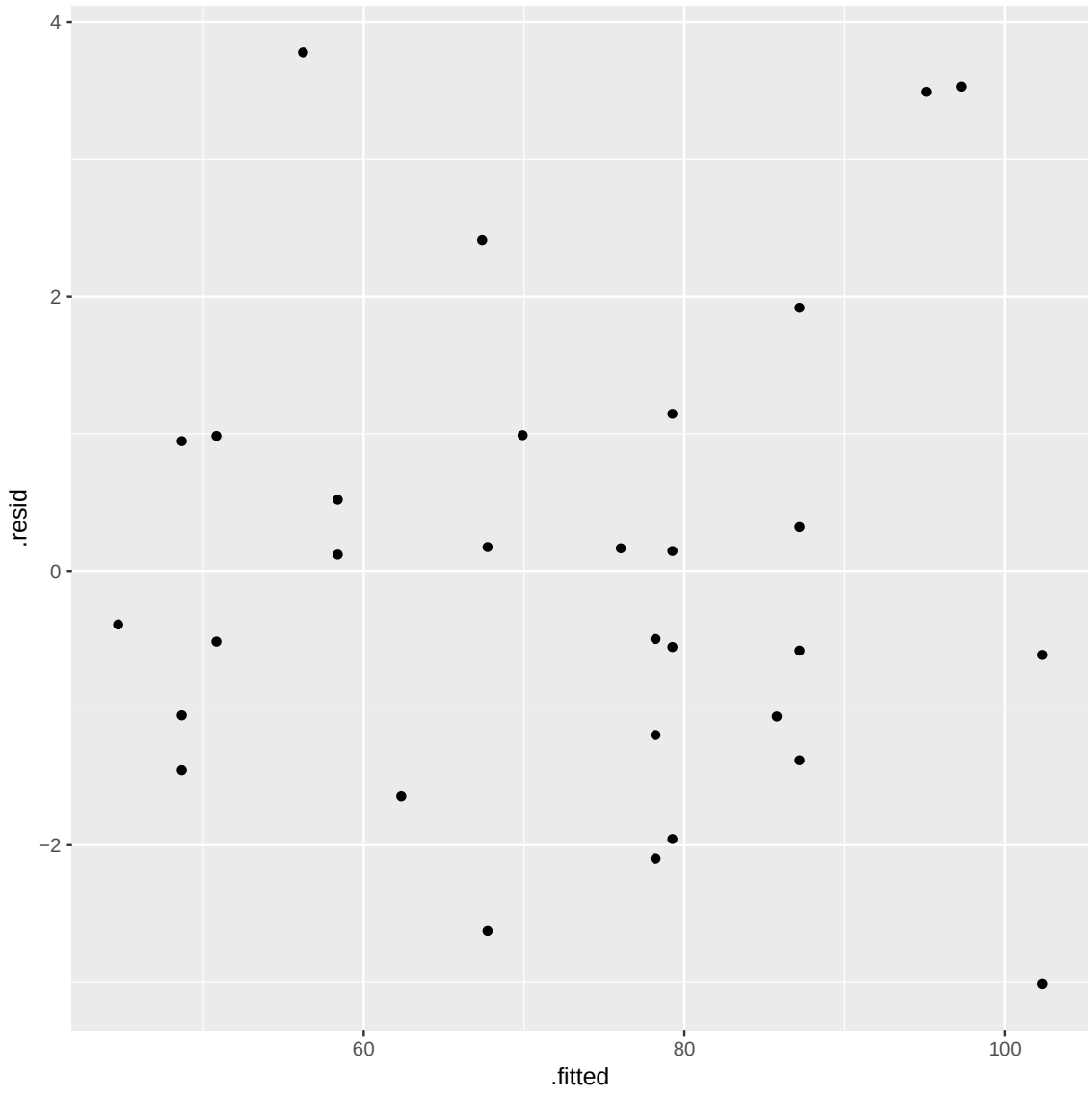


The lines are almost exactly parallel, so having the same slope for each species makes perfect sense.

(g) Make suitable residual plots for the regression `pulse.1`.

Solution: First, the plot of residuals against fitted values (after all, it *is* a regression):

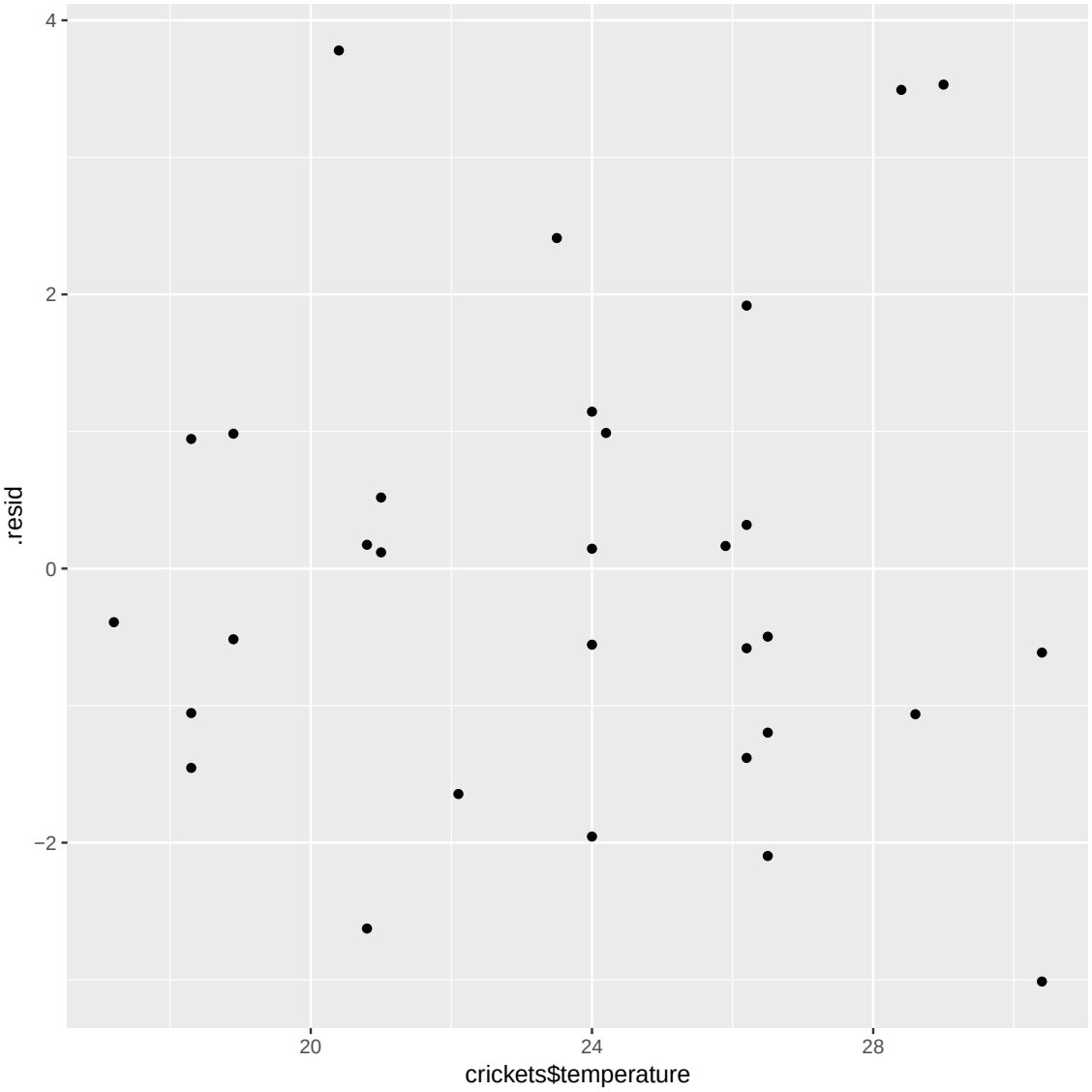
```
ggplot(pulse.1,aes(x=.fitted,y=.resid))+geom_point()
```



This looks nice and random.

Now, we plot the residuals against the explanatory variables. There are two, temperature and species, but the latter is categorical. We'll have some extra issues around species, but before we get to that, we have to remember that the data and the output from the regression are in different places when we plot them. There are different ways to get around that. Perhaps the simplest is to use `pulse.1` as our "default" data frame and then get `temperature` from the right place:

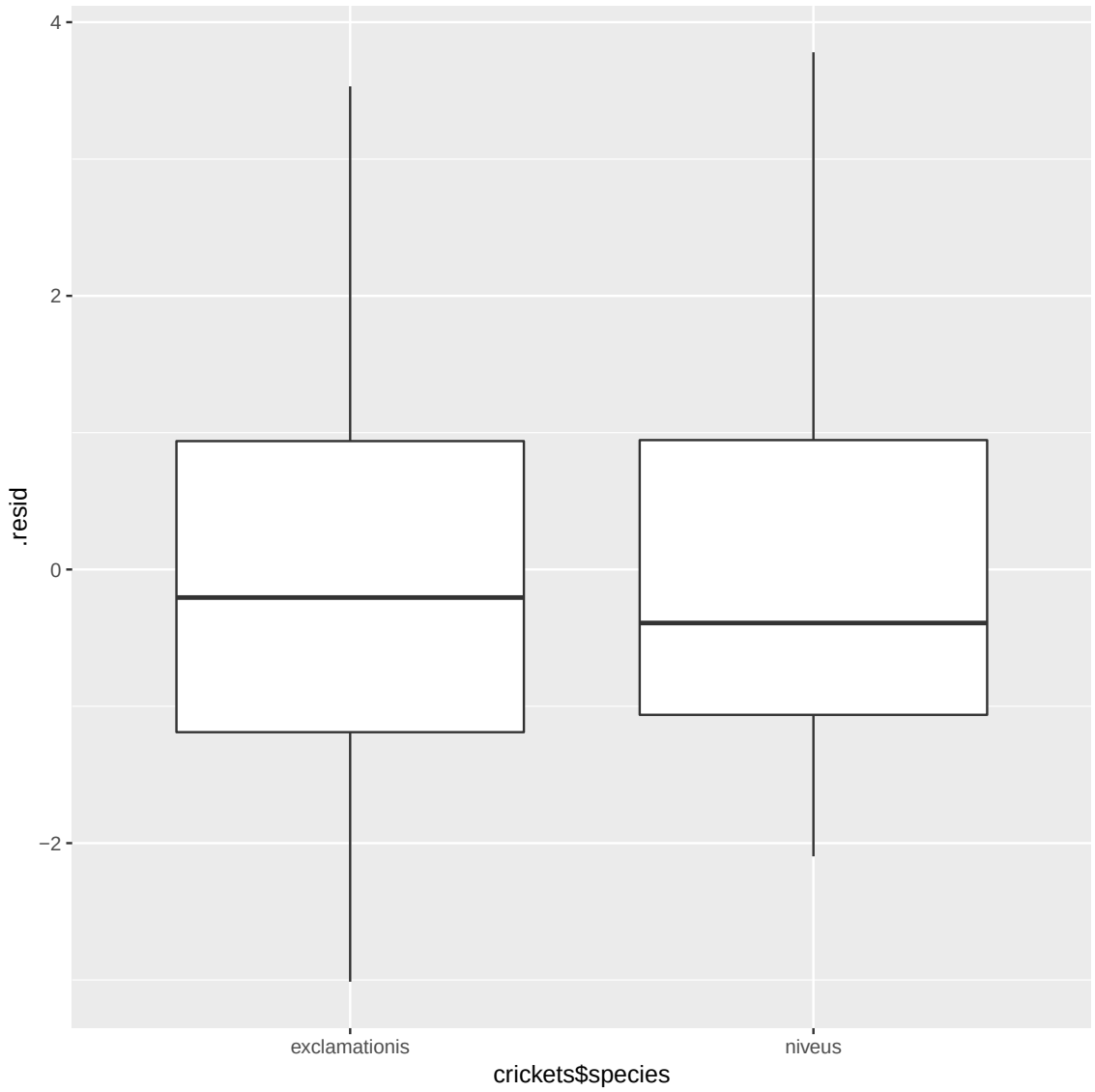
```
ggplot(pulse.1,aes(x=crickets$temperature,y=.resid))+geom_point()
```



I don't see anything untoward there.

Species. We want to compare the residuals for the two species, which is categorical. Since the residuals are quantitative, this suggests a boxplot. Remembering to get species from the right place again, that goes like this:

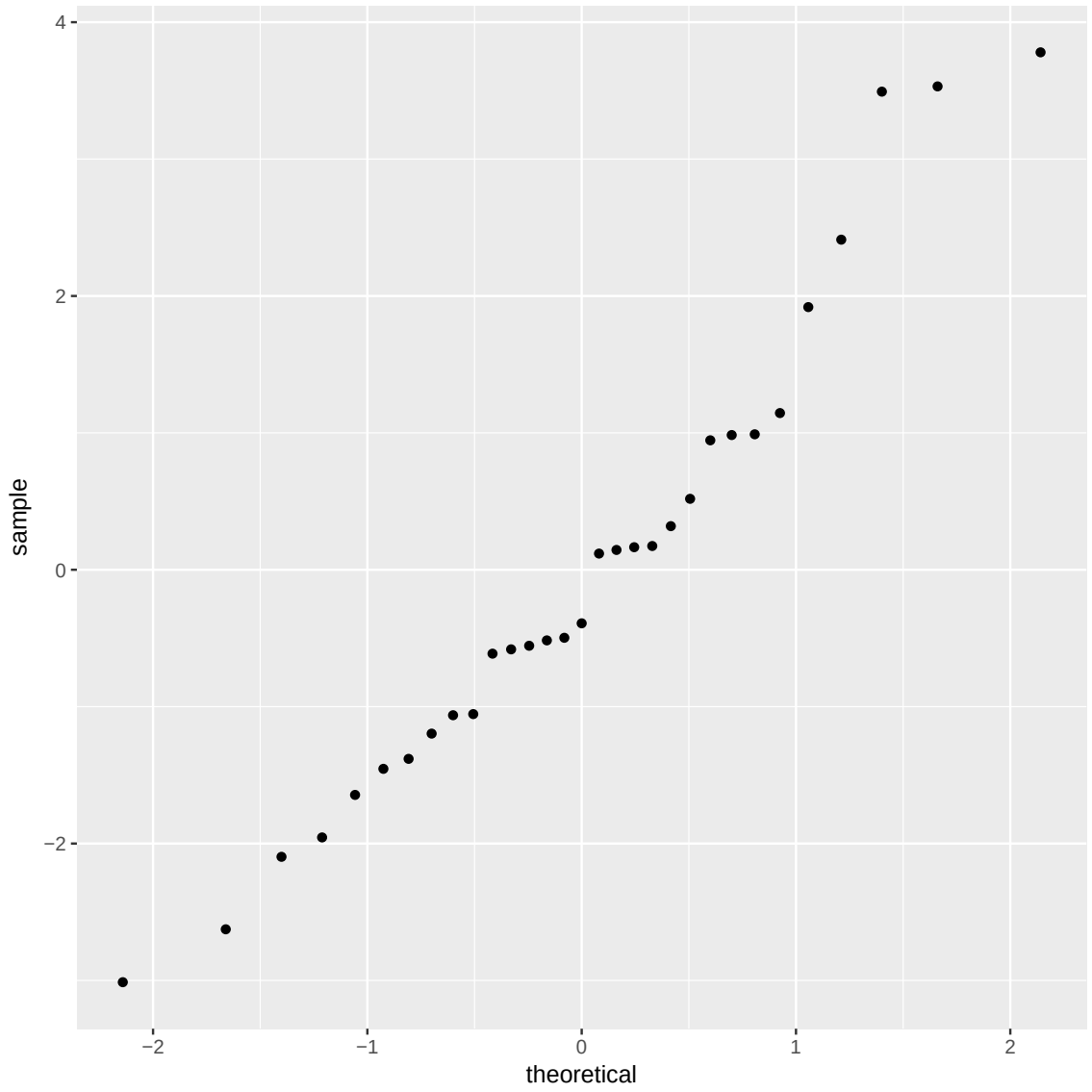
```
ggplot(pulse.1,aes(x=crickets$species,y=.resid))+geom_boxplot()
```



For the residuals, the median should be zero within each group, and the two groups should be approximately normal with mean 0 and about the same spread. Same spread looks OK, since the boxes are almost exactly the same height, but the normality is not quite there, since both distributions are a little bit skewed to the right. That would also explain why the median residual in each group is a little bit less than zero, because the mathematics requires the overall *mean* residual to be zero, and the right-skewness would make the mean higher than the median.

Is that non-normality really problematic? Well, I could look at the normal quantile plot of all the residuals together. This is the one without the line:

```
ggplot(pulse.1,aes(sample=.resid))+stat_qq()
```



There's a little weirdness at the top, and a tiny indication of a curve (that would suggest a little right-skewedness), but not really much to worry about. If that third-highest residual were a bit lower (say, 3 rather than 3.5) and maybe if the lowest residual was a bit lower, I don't think we'd have anything to complain about at all.

So, I'm not worried.

2. Again using the crickets data, let's see if we can reproduce what we did in R, following the steps below.
   (a) First, read in and display the data.

Solution: Same old same old:

```
filename myurl url "http://www.utsc.utoronto.ca/~butler/c32/crickets2.csv";
```

```
proc import
  datafile=myurl
  out=crickets
  dbms=csv
  replace;
  getnames=yes;

proc print;
```

Obs	species	temperature	pulse_rate
1	exclamationis	20.8	67.9
2	exclamationis	20.8	65.1
3	exclamationis	24	77.3
4	exclamationis	24	78.7
5	exclamationis	24	79.4
6	exclamationis	24	80.4
7	exclamationis	26.2	85.8
8	exclamationis	26.2	86.6
9	exclamationis	26.2	87.5
10	exclamationis	26.2	89.1
11	exclamationis	28.4	98.6
12	exclamationis	29	100.8
13	exclamationis	30.4	99.3
14	exclamationis	30.4	101.7
15	niveus	17.2	44.3
16	niveus	18.3	47.2
17	niveus	18.3	47.6
18	niveus	18.3	49.6
19	niveus	18.9	50.3
20	niveus	18.9	51.8
21	niveus	20.4	60
22	niveus	21	58.5
23	niveus	21	58.9
24	niveus	22.1	60.7
25	niveus	23.5	69.8
26	niveus	24.2	70.9
27	niveus	25.9	76.2
28	niveus	26.5	76.1
29	niveus	26.5	77
30	niveus	26.5	77.7
31	niveus	28.6	84.7

31 crickets, of two different species. Check.

(b) Carry out a two-sample t-test to compare mean pulse rates in the two different species.

Solution: This is actually a piece of cake (if you remember how to do it):

```
proc ttest;
  var pulse_rate;
  class species;
```

species	N	Mean	Std Dev	Std Err	Minimum	Maximum
exclamationis	14	85.5857	11.6993	3.1268	65.1000	101.7
niveus	17	62.4294	12.9568	3.1425	44.3000	84.7000
Diff (1-2)		23.1563	12.4089	4.4784		

```
species          Method               Mean       95% CL Mean        Std Dev

exclamationis                       85.5857     78.8307  92.3407     11.6993
niveus                              62.4294     55.7676  69.0912     12.9568
Diff (1-2)       Pooled             23.1563     13.9969  32.3157     12.4089
Diff (1-2)       Satterthwaite      23.1563     14.0858  32.2268

            species          Method             95% CL Std Dev

            exclamationis                       8.4815  18.8481
            niveus                              9.6499  19.7194
            Diff (1-2)       Pooled             9.8825  16.6815
            Diff (1-2)       Satterthwaite

        Method           Variances        DF    t Value    Pr > |t|

        Pooled           Equal            29       5.17      <.0001
        Satterthwaite    Unequal      28.719       5.22      <.0001
                           Equality of Variances

             Method      Num DF    Den DF    F Value    Pr > F

             Folded F        16        13       1.23    0.7188
```

Remembering to look at the pooled test, the t-statistic and its P-value are the same as we got from R. SAS gives us (at the bottom) a test that the pulse rate variances are the same for each species, and this is not rejected, so using the pooled test is sound.[8]

(c) Reproduce the two-sample t-test using a regression predicting pulse rate from species.

Solution: Think carefully here: it's a regression, but the explanatory variable is categorical, so we have to use `proc glm` rather than `proc reg`:

```
proc glm;
  class species;
  model pulse_rate=species / solution;
```

```
                          The GLM Procedure

                       Class Level Information

               Class         Levels    Values

               species            2    exclamationis niveus
                  Number of Observations Read          31
                  Number of Observations Used          31
                            The GLM Procedure

                     Dependent Variable: pulse_rate

                                      Sum of
Source                     DF        Squares    Mean Square   F Value   Pr > F

Model                       1    4116.742402    4116.742402     26.74   <.0001

Error                      29    4465.432437     153.980429

Corrected Total            30    8582.174839
```

```
           R-Square     Coeff Var      Root MSE    pulse_rate Mean

           0.479685      17.02480      12.40889           72.88710

 Source                     DF      Type I SS    Mean Square   F Value   Pr > F

 species                     1    4116.742402    4116.742402     26.74   <.0001

 Source                     DF    Type III SS    Mean Square   F Value   Pr > F

 species                     1    4116.742402    4116.742402     26.74   <.0001

                                                    Standard
 Parameter                        Estimate           Error   t Value   Pr > |t|

 Intercept                    62.42941176 B     3.00959670     20.74     <.0001
 species   exclamationis      23.15630252 B     4.47842320      5.17     <.0001
 species   niveus              0.00000000 B      .               .        .

NOTE: The X'X matrix has been found to be singular, and a generalized inverse
      was used to solve the normal equations.  Terms whose estimates are
      followed by the letter 'B' are not uniquely estimable.
```

Look along the line for the `species` that is not the baseline (*exclamationis*): the t-statistic is the same 5.17 as the t-test gave. Also, if you check the type III sums of squares table above, you'll find that the F-value of 26.74 is the *square* of the t-value, and its P-value is the same.

If we had had more than two species, this would have been like the pigs example in class, where the appropriate test would have been an analysis of variance. The F-statistic from that would have been the same as the F-statistic in the type III sums of squares table.

(d) Fit a regression predicting pulse rate from species and temperature as well. Compare your answers with R's. (Display the graphical output as well.)

Solution: Same idea again: one of the explanatory variables is categorical, so we need to use `proc glm`:

```
proc glm;
  class species;
  model pulse_rate=species temperature / solution;
```

```
                            The GLM Procedure

                         Class Level Information

                Class         Levels    Values

                species            2    exclamationis niveus
                   Number of Observations Read          31
                   Number of Observations Used          31
                            The GLM Procedure

                    Dependent Variable: pulse_rate

                                       Sum of
 Source                     DF        Squares    Mean Square   F Value   Pr > F

 Model                       2    8492.824970    4246.412485   1330.72   <.0001

 Error                      28      89.349869       3.191067

 Corrected Total            30    8582.174839
           R-Square     Coeff Var      Root MSE    pulse_rate Mean

           0.989589      2.450853      1.786356           72.88710
```

Source	DF	Type I SS	Mean Square	F Value	Pr > F
species	1	4116.742402	4116.742402	1290.08	<.0001
temperature	1	4376.082568	4376.082568	1371.35	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
species	1	598.003953	598.003953	187.40	<.0001
temperature	1	4376.082568	4376.082568	1371.35	<.0001

Parameter		Estimate	Standard Error	t Value	Pr > |t|
Intercept		-17.27619743 B	2.19552853	-7.87	<.0001
species	exclamationis	10.06529123 B	0.73526224	13.69	<.0001
species	niveus	0.00000000 B	.	.	.
temperature		3.60275287	0.09728809	37.03	<.0001

NOTE: The X'X matrix has been found to be singular, and a generalized inverse was used to solve the normal equations. Terms whose estimates are followed by the letter 'B' are not uniquely estimable.

In the bottom table, the estimates are the same as R's, at least allowing for the fact that the other species was used as the baseline (so the sign got switched). Everything else is consistent.

We don't have the output from R to compare the type III sums of squares with. This is what came from `drop1`:

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

```
## Single term deletions
##
## Model:
## pulse_rate ~ species + temperature
##             Df Sum of Sq    RSS     AIC F value    Pr(>F)
## <none>                     89.3  38.816
## species      1     598.0  687.4 100.065   187.4 6.272e-14 ***
## temperature  1    4376.1 4465.4 158.074  1371.4 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

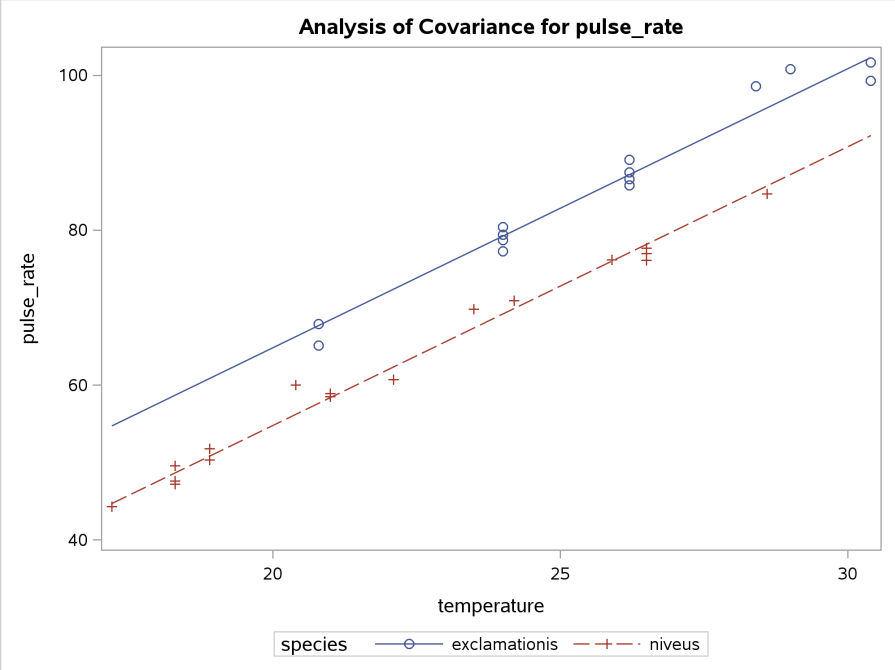
and these are the same as the type III tests.

Incidentally, `anova` corresponds to the type I sum of squares (that we ignore):

```
anova(pulse.1)
```

```
## Analysis of Variance Table
##
## Response: pulse_rate
##             Df Sum Sq Mean Sq F value    Pr(>F)
## species      1 4116.7  4116.7  1290.1 < 2.2e-16 ***
## temperature  1 4376.1  4376.1  1371.4 < 2.2e-16 ***
## Residuals   28   89.3     3.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here's the graph that comes out:

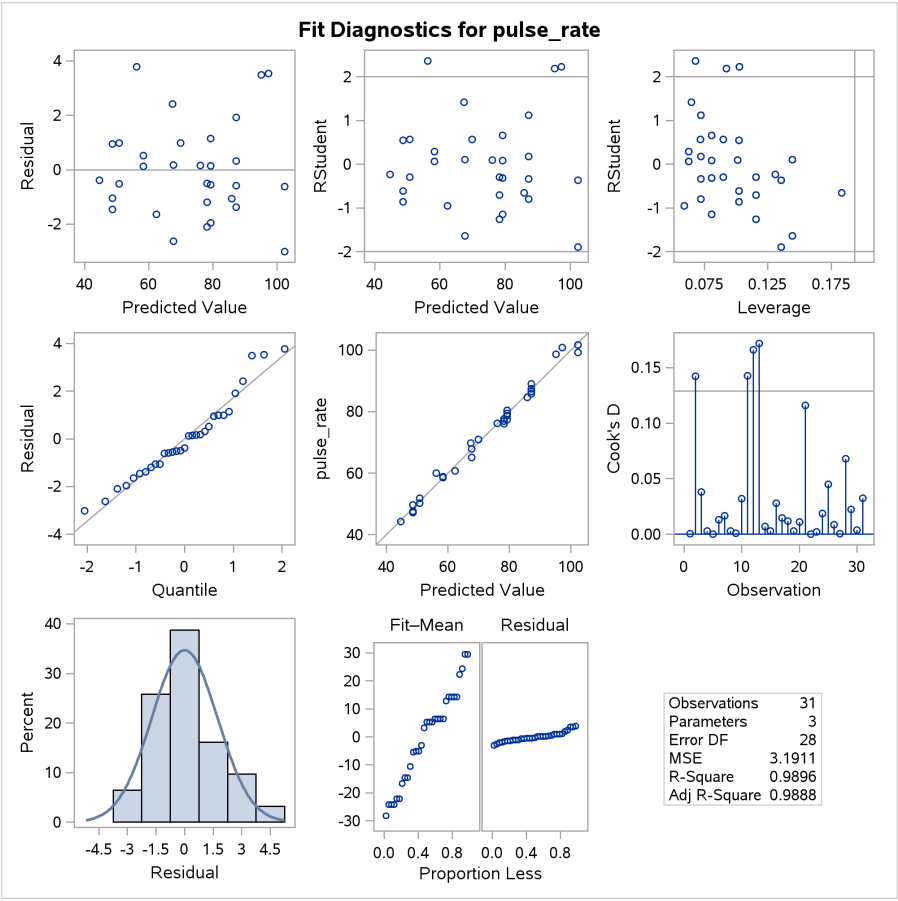

Analysis of Covariance for pulse_rate
pulse_rate
100
80
60
40
20
25
30
temperature
species
exclamationis
niveus

This is like the graph we drew in R, except that here the two lines are *constrained* to come out with the same slope, whether the data support that or not. The fact that it looks the same as R's graph suggests that identical slopes *is* supported by the data.

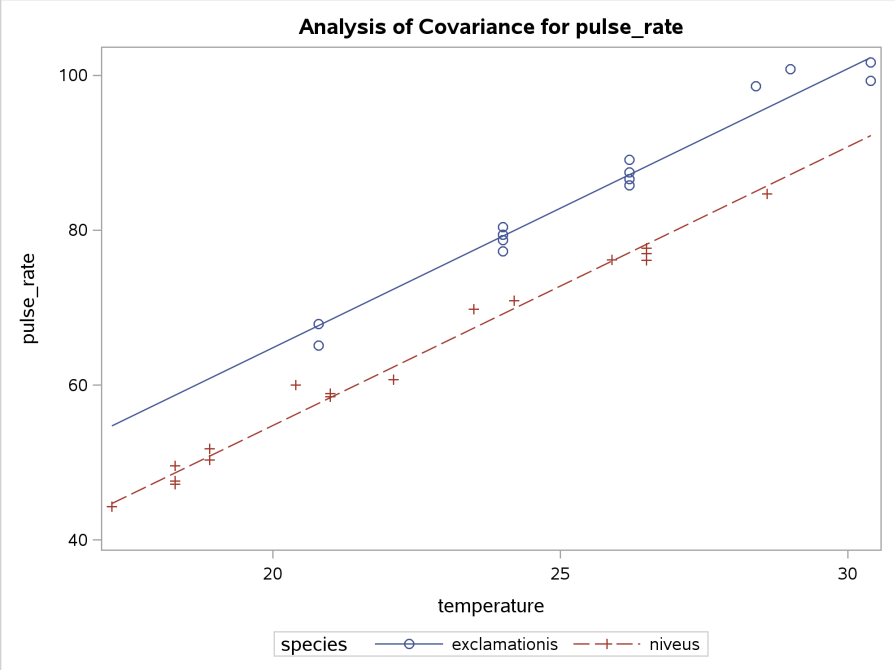
As far as I currently understand (and I am typing this while heading home on the bus, so I can't check just yet), `proc glm` doesn't naturally produce residual plots, because it takes an ANOVA-like approach to the analysis rather than a regression-like one.

I'm home now. This is how it's done:

```
proc glm plots=(diagnostics residuals);
  class species;
  model pulse_rate=species temperature / solution;
```


Fit Diagnostics for pulse_rate
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Leverage
Residual
Quantile
pulse_rate
Predicted Value
Cook's D
Observation
Percent
Residual
Fit–Mean
Residual
Proportion Less
Observations 31
Parameters 3
Error DF 28
MSE 3.1911
R-Square 0.9896
Adj R-Square 0.9888


Residual Plot for pulse_rate
Residual
temperature

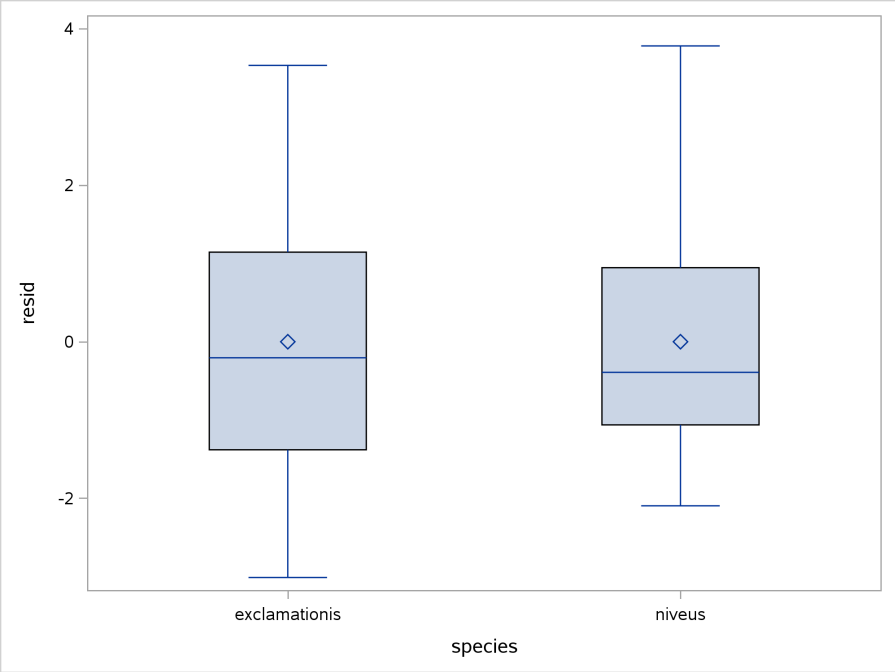

Analysis of Covariance for pulse_rate
100
80
60
40
pulse_rate
20
25
30
temperature
species
exclamationis
niveus

This looks like a regression output. In the first set of plots, we see that the residuals against fitted (top left) look random and the residuals are close to normal (2nd row, 1st plot). Below that, the residuals against temperature also look random. But we don't get a plot of residuals against species (that we made with a boxplot before). To get *that*, I need to do this:

```
proc glm;
  class species;
  model pulse_rate=species temperature / solution;
  output out=res p=fitted r=resid;
```

That makes us a dataset (which becomes the current one) containing all the data plus the fitted values and residuals from this model. Then we use `proc sgplot` to plot whatever we want to plot, namely this:

```
proc sgplot;
  vbox resid / category=species;
```



All of these plots give us the same results as before.

A remark: now that I've gotten the output data set with residuals and stuff in it, I could have used that to make all of my residual plots, and dispensed with the `plots` on my `proc glm` earlier. It's normally more convenient to take the plots SAS gives you, but there is no problem in obtaining an output data set and using that to make your plots. If it works, it's good.[9]

3. In Denali National Park, Alaska, the size of the wolf population depends on the size of the caribou population (since wolves hunt and kill caribou). This is a large national park, so caribou are found in very large herds, so big, in fact, that the well-being of the entire herd is not threatened by wolf attacks.[10] Can the size of the caribou population be used to predict the size of the wolf population?

   The data can be found at `http://www.utsc.utoronto.ca/~butler/c32/caribou.txt`. The columns are: the date of the survey,[11] the name of the park employee in charge of the survey, the caribou

population (in hundreds) and the wolf population (actual count).[12]

We are going to use SAS for this question, but this format of data file is one we don't know how to read into SAS, so we are going to use R to help us first.

(a) Take a look at the data file. How would you describe its format? Read it into R, and check that you got something sensible.

Solution: This looks at first sight as if it's separated by spaces, but most of the data values are separated by *more than one* space. If you look further, you'll see that the values are *lined up in columns*, with the variable names aligned at the top. This is exactly the kind of thing that `read_table` will read:

```
my_url="http://www.utsc.utoronto.ca/~butler/c32/caribou.txt"
denali=read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   date = col_character(),
##   name = col_character(),
##   caribou = col_integer(),
##   wolf = col_integer()
## )
```

```
denali
```

```
## # A tibble: 7 x 4
##         date          name caribou  wolf
##        <chr>         <chr>   <int> <int>
## 1 09/01/1995      David S.      30    66
## 2 09/24/1996   Youngjin K.      34    79
## 3 10/03/1997 Srinivasan M.      27    70
## 4 09/15/1998   Lee Anne J.      25    60
## 5 09/08/1999  Stephanie T.      17    48
## 6 09/03/2000   Angus Mc D.      23    55
## 7 10/06/2001      David S.      20    60
```

That worked: four columns with the right names, and the counts of caribou and wolf are numbers. The only (small) weirdness is that the dates are text rather than having been converted into dates. This is because they are not year-month-day, which is the only format that gets automatically converted into dates when read in. (You could use `mdy` from `lubridate` to make them dates.)

(b) Save your R data frame as a `.csv` file. This goes using `write_csv`, which is the exact opposite of `read_csv`. It takes two things: a data frame to save as a `.csv`, and the name of a file to save it in.

Solution: You probably haven't seen this before, so I hope I gave you some clues:

```
write_csv(denali,"/home/ken/denali.csv")
```

This saves the data frame as a `.csv` into my home folder (on Linux; Mac probably looks similar, Windows different). The file looks like this:

```
date,name,caribou,wolf
09/01/1995,David S.,30,66
09/24/1996,Youngjin K.,34,79
10/03/1997,Srinivasan M.,27,70
```

```
09/15/1998,Lee Anne J.,25,60
09/08/1999,Stephanie T.,17,48
09/03/2000,Angus Mc D.,23,55
10/06/2001,David S.,20,60
```

The columns don't line up any more, but there are no extra spaces, so that reading this into SAS as a `.csv` should go smoothly.

(c) Read your `.csv` file into SAS, and list the values.

Solution: This is the usual `proc import`. First, though, upload the `.csv` from wherever it is now to your account on SAS Studio, and then:

```
proc import
  datafile='/home/ken/denali.csv'
  out=denali
  dbms=csv
  replace;
  getnames=yes;
```

and then

```
proc print;
```

Obs	date	name	caribou	wolf
1	09/01/1995	David S.	30	66
2	09/24/1996	Youngjin K.	34	79
3	10/03/1997	Srinivasan M.	27	70
4	09/15/1998	Lee Anne J.	25	60
5	09/08/1999	Stephanie T.	17	48
6	09/03/2000	Angus Mc D.	23	55
7	10/06/2001	David S.	20	60

Go searching in the log tab for `proc import`, and below it you'll see some lines with `format` on them. This tells you how the variables were read. Mine is:

```
1596            informat date mmddyy10. ;
1597            informat name $13. ;
1598            informat caribou best32. ;
1599            informat wolf best32. ;
1600            format date mmddyy10. ;
1601            format name $13. ;
1602            format caribou best12. ;
1603            format wolf best12. ;
```

The dates were correctly deduced to be month-day-year, the names as text, and the caribou and wolf counts as numbers (that's what the `best` followed by a number is).

These appear to be duplicated because the `informat` lines are how the values are read in, and the `format` lines are how they are listed out (by default). These are not necessarily the same.

(d) Display the data set with the dates in Canadian/British format, day before month.

Solution: Add a `format` to the `proc print`. The right one is `ddmmyy10.`, to get the day before the month; a width of 10 characters gives room for the slashes and 4-digit years.

```
proc print;
format date ddmmyy10.;
```

Obs	date	name	caribou	wolf
1	01/09/1995	David S.	30	66
2	24/09/1996	Youngjin K.	34	79
3	03/10/1997	Srinivasan M.	27	70
4	15/09/1998	Lee Anne J.	25	60
5	08/09/1999	Stephanie T.	17	48
6	03/09/2000	Angus Mc D.	23	55
7	06/10/2001	David S.	20	60

Compare this one:

```
proc print;
  format date ddmmyy8.;
```

Obs	date	name	caribou	wolf
1	01/09/95	David S.	30	66
2	24/09/96	Youngjin K.	34	79
3	03/10/97	Srinivasan M.	27	70
4	15/09/98	Lee Anne J.	25	60
5	08/09/99	Stephanie T.	17	48
6	03/09/00	Angus Mc D.	23	55
7	06/10/01	David S.	20	60

This one has only two-digit years, leaving us prone to the "Y2K problem",[13] where it is not clear which century each year belongs to.

(e) Display the data set in such a way that you see the days of the week and the month names for the dates.

Solution: I left this open to you to make the precise choice of format, but one possibility is this one:

```
proc print;
  format date weekdate20.;
```

Obs	date	name	caribou	wolf
1	Fri, Sep 1, 1995	David S.	30	66
2	Tue, Sep 24, 1996	Youngjin K.	34	79
3	Fri, Oct 3, 1997	Srinivasan M.	27	70
4	Tue, Sep 15, 1998	Lee Anne J.	25	60
5	Wed, Sep 8, 1999	Stephanie T.	17	48
6	Sun, Sep 3, 2000	Angus Mc D.	23	55
7	Sat, Oct 6, 2001	David S.	20	60

The number on the end of `weekdate` is how many characters SAS uses to display the date. It makes the best of the space you give it:

```
proc print;
  format date weekdate5.;
```

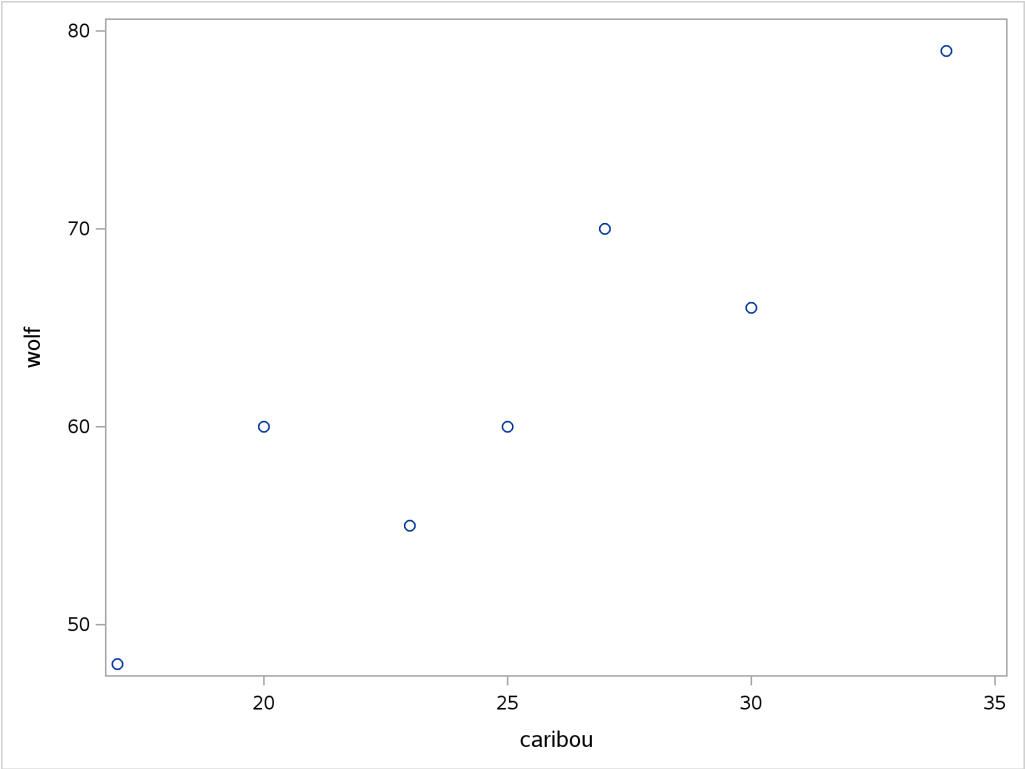
Obs	date	name	caribou	wolf
1	Fri	David S.	30	66
2	Tue	Youngjin K.	34	79
3	Fri	Srinivasan M.	27	70
4	Tue	Lee Anne J.	25	60
5	Wed	Stephanie T.	17	48
6	Sun	Angus Mc D.	23	55
7	Sat	David S.	20	60

The best it can do is to show you the day of the week.

(f) Enough playing around with dates. Make a scatterplot of caribou population (explanatory) against wolf population (response). Do you see any relationship?

Solution: The usual with `proc sgplot`:

```
proc sgplot;
  scatter x=caribou y=wolf;
```



That looks like an upward trend: when the caribou population is large, the wolf population is large too.[14]

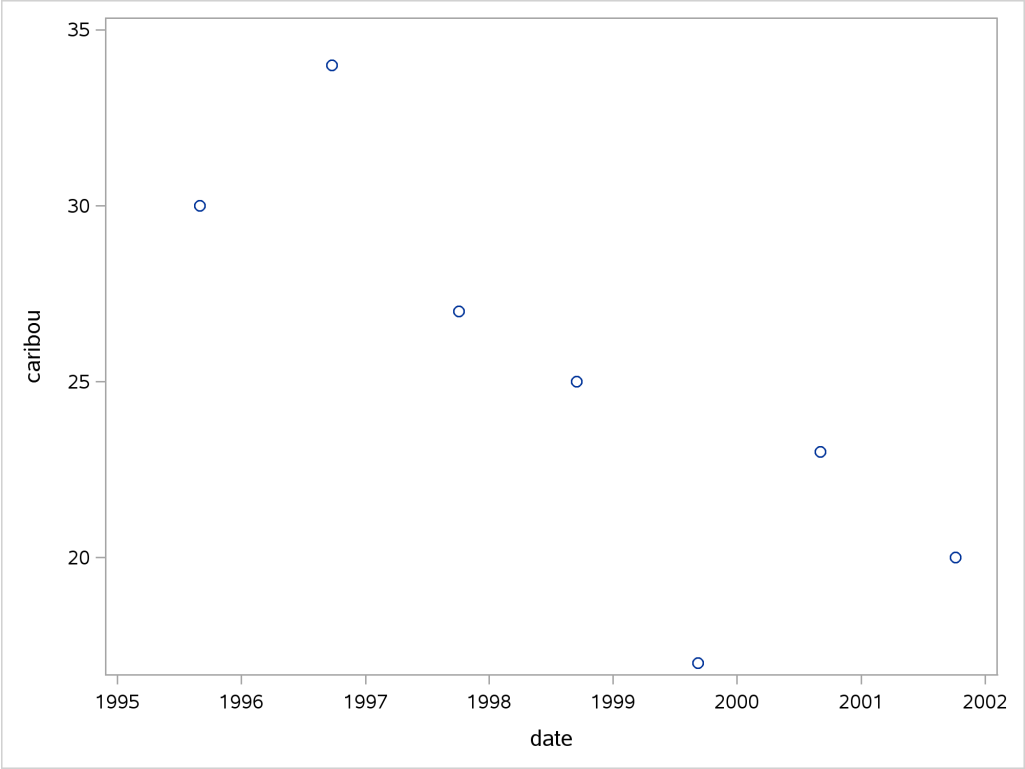
I should point out that the wolf and caribou surveys were taken at different times of the year. The dates in the data file were actually of the caribou surveys (in the fall, as I said). The wolf surveys were taken in the "late winter" (of the following year). This makes sense if you think of the wolf population as varying as a response to the caribou population; you need to allow some time for this "response" to happen. It might even be that the response happens over a longer time frame than this, if you think of the time required for wolves to have and raise pups,[15] which might be a period of years. In the grand scheme of things, there might be a multi-year cyclic variation in caribou and wolf populations; they go up and down together, but there might also be a time lag.

According to `http://www.pbs.org/wnet/nature/river-of-no-return-gray-wolf-fact-sheet/7659/`, wolves in the wild typically live 6–8 years, but many die earlier, often of starvation (so the size of the population of the wolves' prey animals matters a lot).

(g) Make a plot of caribou population against time (this is done the obvious way). What seems to be happening to the caribou population over time?

Solution: Make a scatterplot, with the survey date as explanatory variable, and caribou population as response (since time always goes on the x-axis):

```
proc sgplot;
  scatter x=date y=caribou;
```



A coding note here: I didn't need a `format` on my `proc sgplot`, because the dates are already formatted (from `proc import`). If they had been dates that we constructed ourselves, eg. from year, month and day as numbers, they would not have come with a format, and we would have had to supply one when we printed or plotted them.

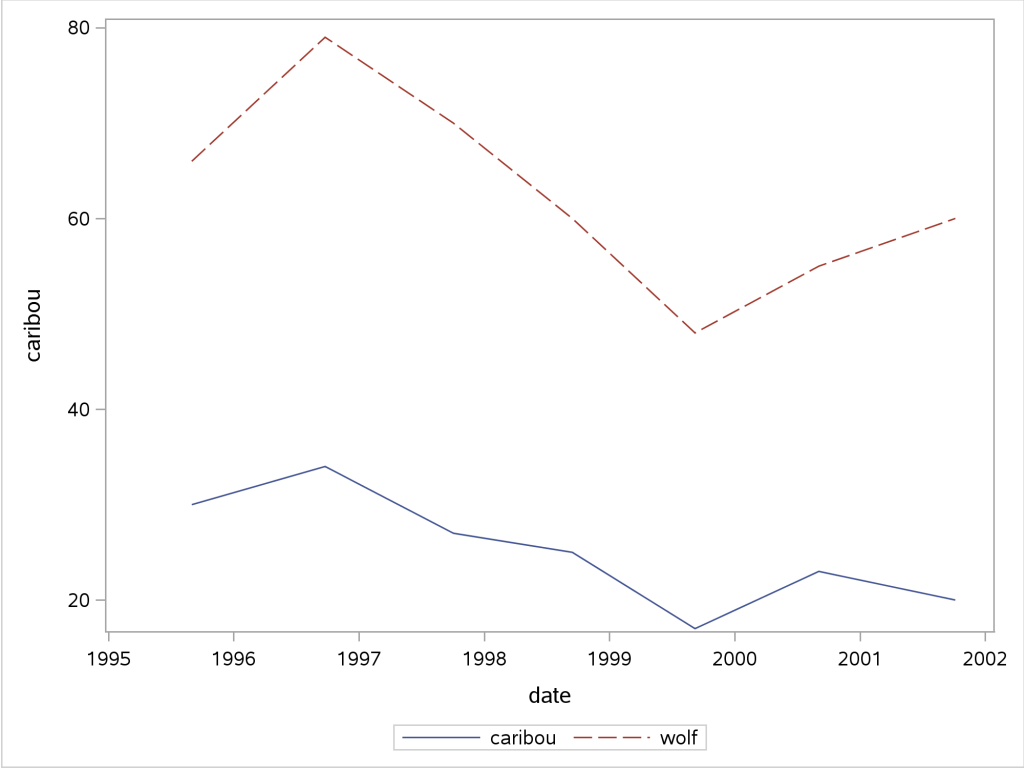
The caribou population is declining over time. We only have seven years' data, though, so it's not clear whether this is to do with climate change or some multi-year cycle in which wolf and caribou populations go up and down together, and we just happen to have hit the "down" part of the cycle.

Where the tick marks are on the x-axis mark the *start* of the year in question, so that the surveys come correctly about $\frac{3}{4}$ of the way through the year. SAS displayed just the years on the x-axis, since that was the scale of the data. If our dates happened to be all one year or all one month, you would have seen more of the format.

(h) The caribou and wolf populations over time are really "time series", so they can be plotted against time by `series` instead of `scatter`. Make a plot of *both* the caribou and wolf populations against time. (That is, use one `sgplot` with two `series` lines, where the `series` lines look just as `scatter` lines would.) How is `series` different from `scatter`?

Solution: I tried to give you enough hints:

```
proc sgplot;
  series x=date y=caribou;
  series x=date y=wolf;
```



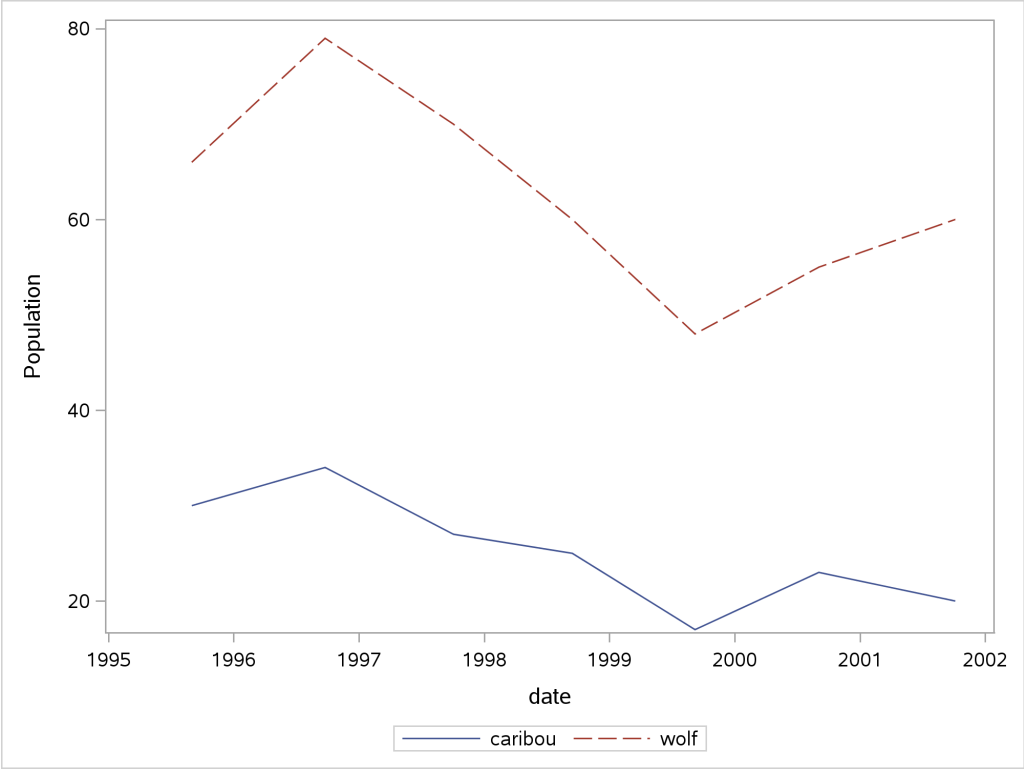
The big difference is that the points are joined by lines, each one to the next one in time order, so that the time nature of the data is more apparent. (Without the lines, it could be much less obvious which data value belongs to which series.)

Because we plotted two series, we also get a legend, and the two series are distinguished by colour and line type.

You should probably recall the scales: the caribou population was measured in hundreds, so the caribou numbers are a lot bigger than the wolf numbers. Evidently they measured caribou population in hundreds to get comparable numbers with the wolf population. In fact, I expect that park officials produced a graph very like this one. Probably in Excel, though.[16]

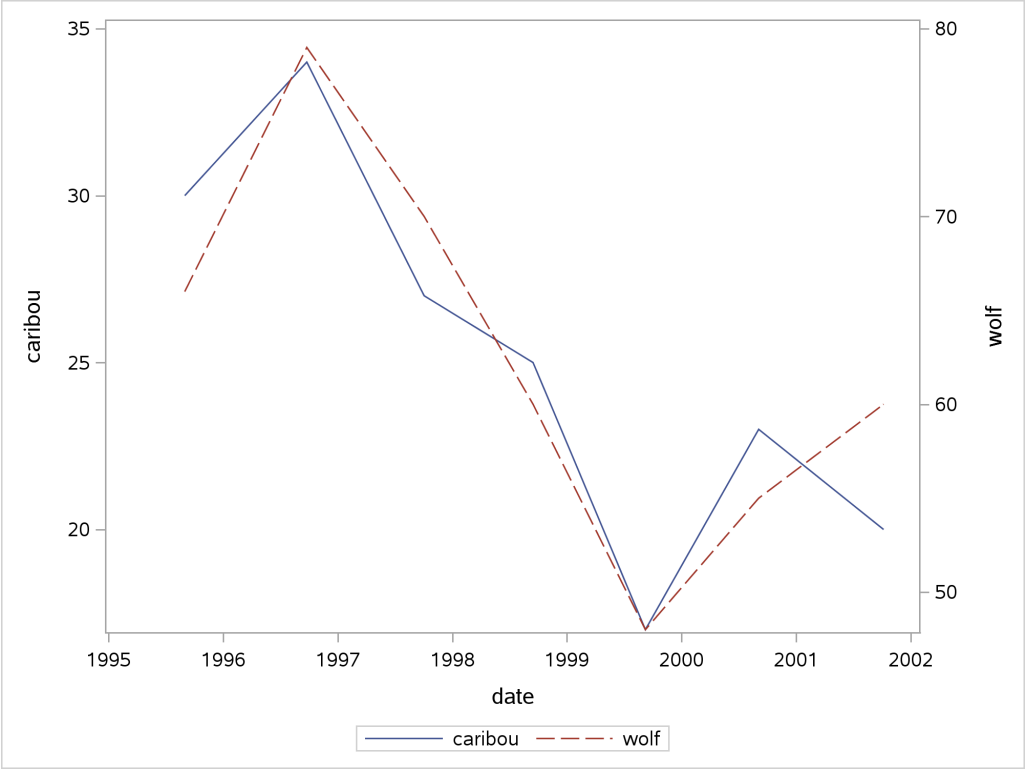
I should have labelled the y-axis "population". That can be done:

```
proc sgplot;
  series x=date y=caribou;
  series x=date y=wolf;
  yaxis label='Population';
```



This is, in fact, one of those rare cases where we can justify having a second y-axis: left one for caribou, right one for wolf. Be aware, though, that you can scale the second y-axis how you like, which means that you can obtain a variety of apparent relationships between the two variables. This is the right way to do it (letting SAS choose the scale):

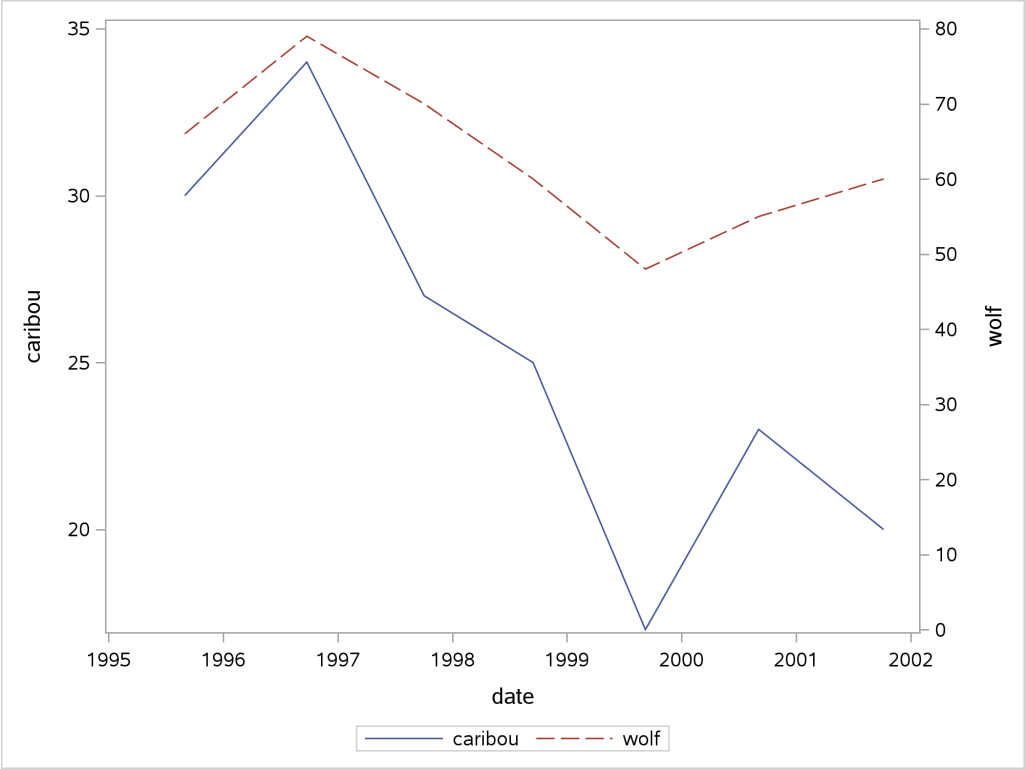
```
proc sgplot;
  series x=date y=caribou;
  series x=date y=wolf / y2axis;
```



This makes it even clearer that caribou and wolf populations rise and fall together.

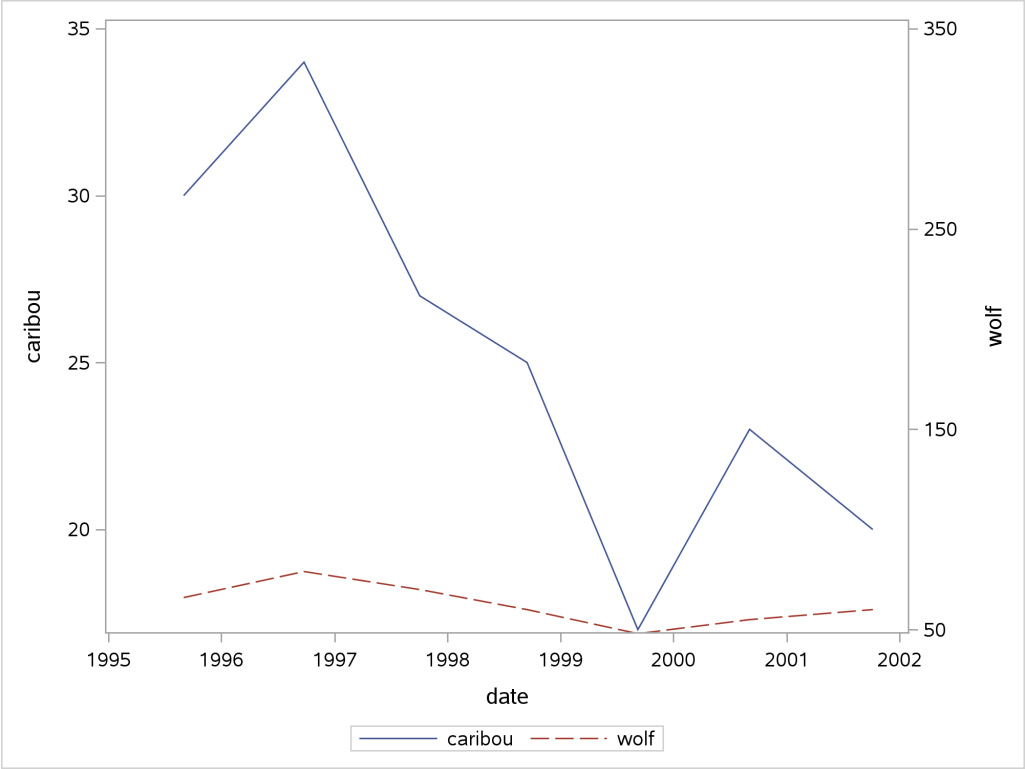
This is about the only double y-axis setup that I like, because you can choose the scale of the second y-axis however you like, to make it look as if the wolf population is very big:

```
proc sgplot;
  y2axis values=(0 to 80 by 10);
  series x=date y=caribou;
  series x=date y=wolf / y2axis;
```



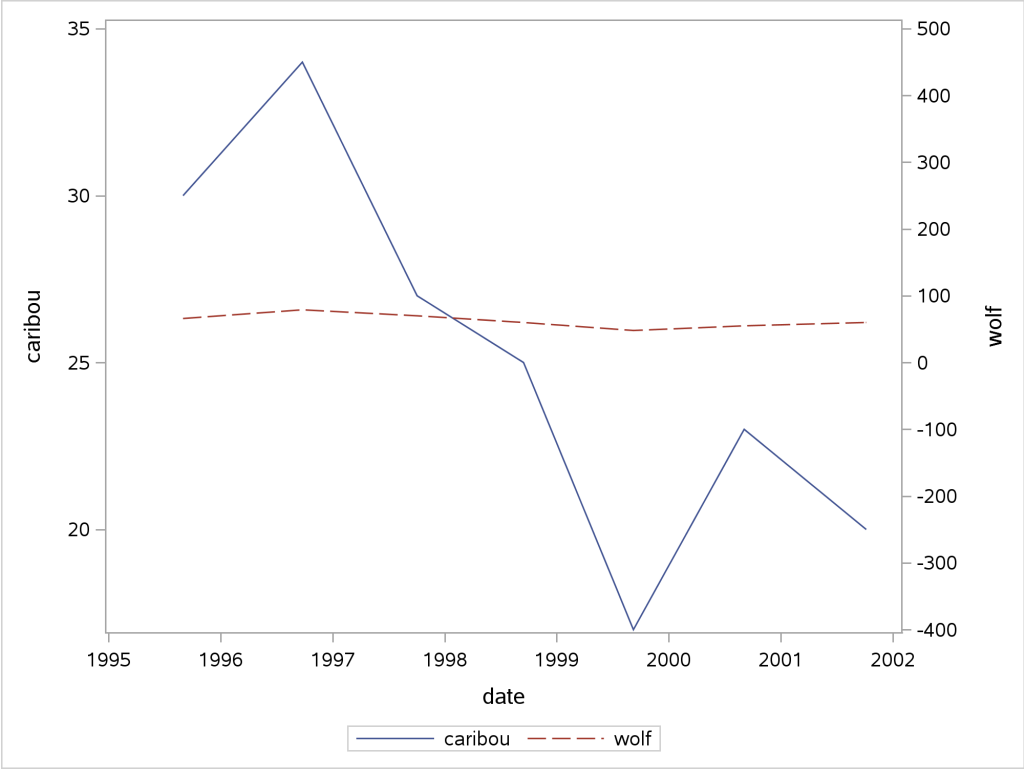
or very small:

```
proc sgplot;
  y2axis values=(50 to 400 by 100);
  series x=date y=caribou;
  series x=date y=wolf / y2axis;
```



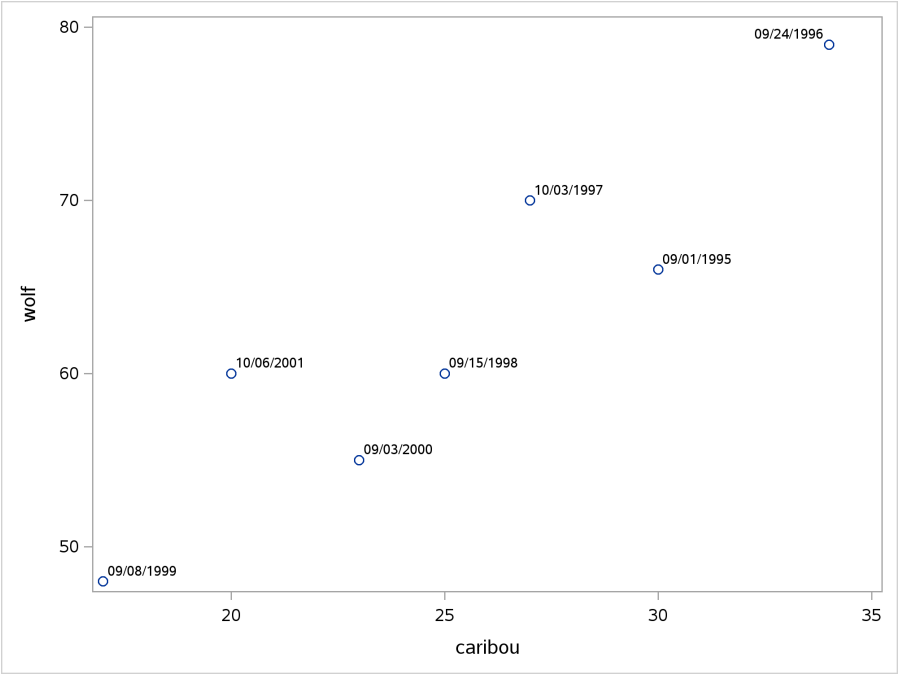
or hardly varies at all:

```
proc sgplot;
  y2axis values=(-400 to 500 by 100);
  series x=date y=caribou;
  series x=date y=wolf / y2axis;
```



In my opinion, too many people just plot series against time, possibly with a second y-axis. Variables that vary together, like the wolf and caribou populations here, ought to be plotted *against each other* on a scatterplot, possibly with the time points labelled:

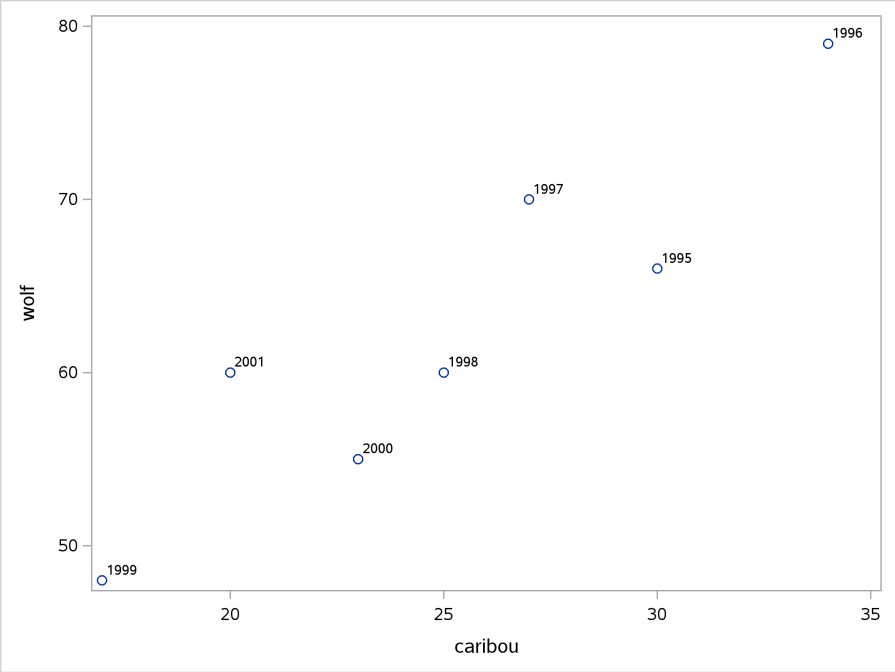
```
proc sgplot;
  scatter x=caribou y=wolf / datalabel=date;
```



or, better, with just the years, which we obtain first:

```
data denali2;
  set denali;
  year=year(date);

proc sgplot;
  scatter x=caribou y=wolf / datalabel=year;
```



1996 was the high year for the populations, followed by a precipitous decline, and by 2000 or 2001 we would guess that the populations were beginning to increase again.

The ambitious among you may like to join the points by arrows, in time order. The further ambitious may like to compare the graphs here with other predator-prey relationships.

(i) Back in part (f), you drew a scatterplot of wolf population against caribou population. How does any trend there show up in the time plot you just drew?

Solution: Back in (f), we found an upward trend with the two populations: they tended to be large or small together. That should show up here as this: in a year where caribou population is large, wolf population is also large. In the fall 1996 survey, both populations were at their biggest, and in the fall 1999 survey, both populations were smallest. Also, the shapes of the time trends are very similar. So the plot of (f) and this one are telling the same story, with this plot giving the additional information that both populations (over this time span) are decreasing over time.

4. The Worcester Heart Attack Study is an ongoing study of heart attacks in the Worcester, MA area. The main purpose of the study is to investigate changes over time in incidence and death rates, and also the use of different treatment approaches. We will be mainly using this data set to investigate data handling and dealing with dates. The data can be found at `http://www.utsc.utoronto.ca/~butler/`

c32/whas500.txt.

(a) Read the data into SAS. Display the first five rows of your dataset as read in, with the dates shown as year-month-day.

Solution: There would normally be a `delimiter` line here. There is no harm in including it, as usual, but leaving out the `delimiter` assumes the delimiting character to be a space, which is usually what you want (as it is here).

```
filename myurl url 'http://www.utsc.utoronto.ca/~butler/c32/whas500.txt';

proc import
  datafile=myurl
  out=whas
  dbms=dlm
  replace;
  getnames=yes;

proc print data=whas(obs=5);
  format admitdate--fdate yymmdd10.;
```

This is actually my second attempt. My first attempt skipped the `format` while I looked at what variables I had. I saw that the columns `admitdate` through `fdate` were dates, and they were consecutive columns, so that the `format` as shown would work.[17]

I said only 5 rows because there are a lot of variables:

Obs	id	age	gender	hr	sysbp
1	1	83	0	89	152
2	2	49	0	84	120
3	3	70	1	83	147
4	4	70	0	65	123
5	5	70	0	63	135

Obs	diasbp	bmi	cvd	afb	sho
1	78	25.54051	1	1	0
2	60	24.02398	1	0	0
3	88	22.1429	0	0	0
4	76	26.63187	1	0	0
5	85	24.41255	1	0	0

Obs	chf	av3	miord	mitype	year	admitdate
1	0	0	1	0	1	1997-01-13
2	0	0	0	1	1	1997-01-19
3	0	0	0	1	1	1997-01-01
4	1	0	0	1	1	1997-02-17
5	0	0	0	1	1	1997-03-01

Obs	disdate	fdate	los	dstat	lenfol	fstat
1	1997-01-18	2002-12-31	5	0	2178	0
2	1997-01-24	2002-12-31	5	0	2172	0
3	1997-01-06	2002-12-31	5	0	2190	0
4	1997-02-27	1997-12-11	10	0	297	1
5	1997-03-07	2002-12-31	6	0	2131	0

This looks like success. The clue that it worked is that `lenfol` in the original data file was usually a big number and `fstat`, on the end of the line, was 0 or 1, and so they appear here. `admitdate`, `disdate` and `fdate` are all properly formatted with the year first.

(b) The variables `los` and `lenfol` are numbers of days. Create a new data set that contains the differences between each of your dates, and see if you can work out what `los` and `lenfol` actually are. (When you display the results, display only the columns you care about and only enough rows to convince yourself that it worked.)

Solution: The strategy is to make a copy of the data set you read in from the file, create your new variables and then either (i) keep only the variables you want and print out the whole resulting data set or (ii) `print` only the variables you care about. I think 20 rows is enough:

```
data whas2;
  set whas;
  diff1=disdate-admitdate;
  diff2=fdate-admitdate;
  diff3=fdate-disdate;
  keep diff1 diff2 diff3 los lenfol;


proc print data=whas2(obs=20);
```

Obs	los	lenfol	diff1	diff2	diff3
1	5	2178	5	2178	2173
2	5	2172	5	2172	2167
3	5	2190	5	2190	2185
4	10	297	10	297	287
5	6	2131	6	2131	2125
6	1	1	1	1	0
7	5	2122	5	2122	2117
8	4	1496	4	1496	1492
9	4	920	4	920	916
10	5	2175	5	2175	2170
11	5	2173	5	2173	2168
12	10	1671	10	1671	1661
13	7	2192	7	2192	2185
14	21	865	21	865	844
15	4	2166	4	2166	2162
16	1	2168	1	2168	2167
17	13	905	13	905	892
18	14	2353	14	2353	2339
19	6	2146	6	2146	2140
20	17	61	17	61	44

Alternatively, less work on the `data` step and more on the `proc print`:

```
data whas3;
  set whas;
  diff1=disdate-admitdate;
  diff2=fdate-admitdate;
  diff3=fdate-disdate;


proc print data=whas3(obs=20);
  var diff1 diff2 diff3 los lenfol;
```

Obs	diff1	diff2	diff3	los	lenfol
1	5	2178	2173	5	2178
2	5	2172	2167	5	2172
3	5	2190	2185	5	2190
4	10	297	287	10	297
5	6	2131	2125	6	2131
6	1	1	0	1	1
7	5	2122	2117	5	2122
8	4	1496	1492	4	1496
9	4	920	916	4	920
10	5	2175	2170	5	2175
11	5	2173	2168	5	2173
12	10	1671	1661	10	1671
13	7	2192	2185	7	2192
14	21	865	844	21	865
15	4	2166	2162	4	2166
16	1	2168	2167	1	2168
17	13	905	892	13	905
18	14	2353	2339	14	2353
19	6	2146	2140	6	2146
20	17	61	44	17	61

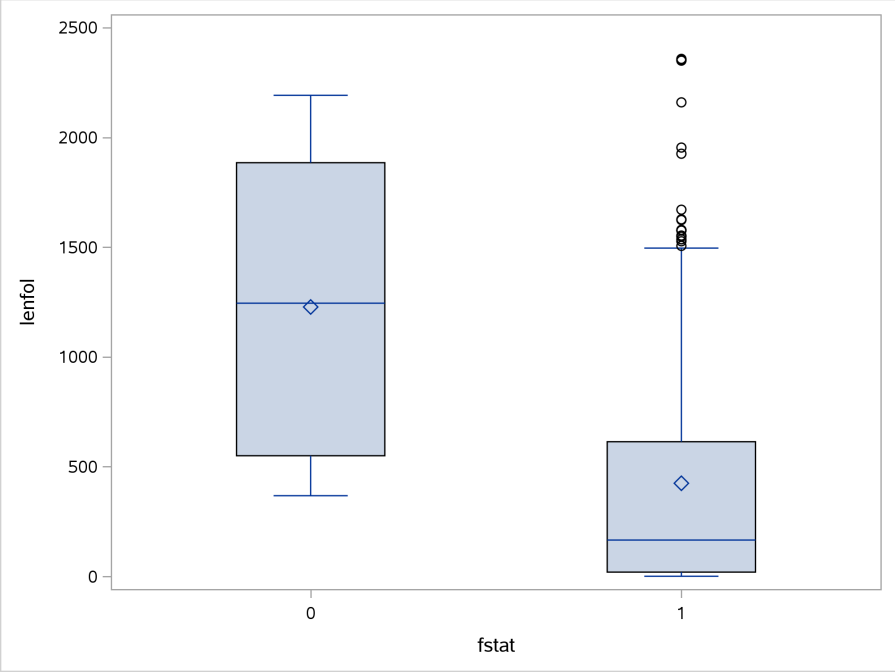
My `diff1` is the same as `los`. I calculated `diff1` as `disdate` minus `admitdate`. It doesn't take much imagination to realize that these are the dates each patient was admitted to and discharged from the hospital (but the other way around). This is the number of days each patient stayed in the hospital: that is, `los` is the "length of stay" in hospital.

`lenfol` is the same as my `diff2`, which is the time from being admitted to hospital to `fdate`, which is the latest of any of the dates. What usually happens in a study like this is that the doctor[18] checks in with each patient every so often to see how they are doing, and thus `fdate` is the "latest followup date". This will be either the end of the study, or the date at which the patient was noted to have died. That is, `lenfol` is the "length of followup", from the first time the patient was seen to the last.

(c) What do you think `fstat` represents? To help you guess, make side-by-side boxplots of `lenfol` for each value of `fstat`. (That will mean going back to the dataset you read in from the file.)

Solution: The dataset I read in was called `whas`, so I need to specify that on the `proc sgplot`:

```
proc sgplot data=whas;
  vbox lenfol / category=fstat;
```


2500
2000
1500
1000
500
0
lenfol
0
1
fstat

What's showing here? When `fstat` is 0, `lenfol` has a nice symmetric distribution, with a maximum around 2200 (days) and no outliers. But when `fstat` is 1, `lenfol` has a very right-skewed distribution with a lot of outliers at the upper end. For these patients, the length of followup is usually short, with a few patients having longer followup. The likely meaning of this is that patients with `fstat` equal to 1 are the ones that died (often fairly quickly), while the patients with `fstat` of zero were the ones that were still alive at the end of the study.

Patients continued to be enrolled into the study at various different times, so the time between enrolment and last followup could be quite variable. If I was right about these patients being followed until the study ended, though, the time of last followup should be consistent for all of them:

```
proc means min q1 median q3 max;
  where fstat=0;
  var fdate;
```

```
                          The MEANS Procedure

                        Analysis Variable : fdate

                    Lower                          Upper
     Minimum        Quartile          Median       Quartile          Maximum
-------------------------------------------------------------------------------
    15705.00        15705.00        15705.00        15705.00        15705.00
-------------------------------------------------------------------------------
```

For the patients still alive at the end of the study, the last followup of *all* of them was on the same date. I could even work out what date that was, something like this:

```
proc sort out=sorted;
  where fstat=0;
  by descending fdate;

proc print data=sorted(obs=10);
  var fdate;
```

```
                              Obs         fdate

                                1    31/12/2002
                                2    31/12/2002
                                3    31/12/2002
                                4    31/12/2002
                                5    31/12/2002
                                6    31/12/2002
                                7    31/12/2002
                                8    31/12/2002
                                9    31/12/2002
                               10    31/12/2002
```

The last day of 2002.

It occurs to me that if you made your data set with the time differences by keeping all of the original variables, (and then sending only some of them to `proc print`) that data set will be your most recent one, and you'll be able to run the `proc`s above without specifying a data set, as I did on `proc means` just above.

The place we would go next in terms of analysis would be to think about whether survival time depends on treatment, after allowing for the effects of any of the other variables. This sounds like a regression. What confuses things here is that for the patients who "haven't died yet", we don't know how long they are going to live: all we know is they have lived 2200 days (or whatever) since being admitted to hospital and are still alive the last we heard. These patients are known in the jargon as "censored" (or, I suppose, their survival time is what's "censored" since it hasn't been observed yet). Also, time until death has a lower limit of 0 and (in principle) no upper limit, so it will have a right-skewed distribution, rather than the normal that we would like for a regression. With all this in mind, we would tend to reach for a "survival analysis", often Cox's Proportional Hazards model,[19] which you'll see if you take STAD29.

(d) Now read the data into R. The reading-in part is straightforward, but check what type of thing each column is. Is that what it should be?

Solution: This is `read_delim`:

```
my_url="http://www.utsc.utoronto.ca/~butler/c32/whas500.txt"
whas=read_delim(my_url," ")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   bmi = col_double(),
##   admitdate = col_character(),
##   disdate = col_character(),
##   fdate = col_character()
## )
```

```
## See spec(...)  for full column specifications.
```

```
whas
```

```
## # A tibble: 500 x 22
##       id   age gender    hr sysbp diasbp      bmi   cvd   afb   sho   chf
##    <int> <int>  <int> <int> <int>  <int>    <dbl> <int> <int> <int> <int>
##  1     1    83      0    89   152     78 25.54051     1     1     0     0
##  2     2    49      0    84   120     60 24.02398     1     0     0     0
##  3     3    70      1    83   147     88 22.14290     0     0     0     0
##  4     4    70      0    65   123     76 26.63187     1     0     0     1
##  5     5    70      0    63   135     85 24.41255     1     0     0     0
##  6     6    70      0    76    83     54 23.24236     1     0     0     0
##  7     7    57      0    73   191    116 39.49046     1     0     0     0
##  8     8    55      0    91   147     95 27.11609     1     0     0     0
##  9     9    88      1    63   209    100 27.43554     1     0     0     1
## 10    10    54      0   104   166    106 25.54448     1     0     0     0
## # ... with 490 more rows, and 11 more variables: av3 <int>, miord <int>,
## #   mitype <int>, year <int>, admitdate <chr>, disdate <chr>, fdate <chr>,
## #   los <int>, dstat <int>, lenfol <int>, fstat <int>
```

To see what type everything is, note that when you display a `tibble`, the type of all the columns is displayed, whether or not the column itself is displayed.

All the numbers are properly integer (`int`) or decimal (`dbl`) numbers, but the date columns are `chr` or text. This means that they haven't been read as `Dates` (because they were not in year-month-day order). This is unlike SAS, which determined that they were dates, and even used the first 20 rows of the file to determine what format of dates they were.

(e) The date columns should be R dates. They are not year-month-day, so converting them via `as.Date` (which is what `read_delim` tries to do) will not work. Load the `lubridate` package, and create new columns in your data frame that are properly dates. Save your data frame, and list it to demonstrate that it worked.

Solution:

Load `lubridate` first:

```
library(lubridate)
```

```
##
## Attaching package:  'lubridate'
```

```
## The following object is masked from 'package:base':
##
##    date
```

These dates are day-month-year, so we need `dmy` from `lubridate`:

```
whas2=whas %>% mutate(admit=dmy(admitdate),
                 dis=dmy(disdate),
                 f=dmy(fdate))
glimpse(whas2)
```

```
## Observations: 500
## Variables: 25
## $ id        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
## $ age       <int> 83, 49, 70, 70, 70, 70, 57, 55, 88, 54, 48, 75, 48, ...
## $ gender    <int> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1...
## $ hr        <int> 89, 84, 83, 65, 63, 76, 73, 91, 63, 104, 95, 154, 85...
## $ sysbp     <int> 152, 120, 147, 123, 135, 83, 191, 147, 209, 166, 160...
## $ diasbp    <int> 78, 60, 88, 76, 85, 54, 116, 95, 100, 106, 110, 123,...
## $ bmi       <dbl> 25.54051, 24.02398, 22.14290, 26.63187, 24.41255, 23...
## $ cvd       <int> 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0...
## $ afb       <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0...
## $ sho       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ chf       <int> 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1...
## $ av3       <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ miord     <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0...
## $ mitype    <int> 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1...
## $ year      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ admitdate <chr> "13-01-1997", "19-01-1997", "01-01-1997", "17-02-199...
## $ disdate   <chr> "18-01-1997", "24-01-1997", "06-01-1997", "27-02-199...
## $ fdate     <chr> "31-12-2002", "31-12-2002", "31-12-2002", "11-12-199...
## $ los       <int> 5, 5, 5, 10, 6, 1, 5, 4, 4, 5, 5, 10, 7, 21, 4, 1, 1...
## $ dstat     <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ lenfol    <int> 2178, 2172, 2190, 297, 2131, 1, 2122, 1496, 920, 217...
## $ fstat     <int> 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1...
## $ admit     <date> 1997-01-13, 1997-01-19, 1997-01-01, 1997-02-17, 199...
## $ dis       <date> 1997-01-18, 1997-01-24, 1997-01-06, 1997-02-27, 199...
## $ f         <date> 2002-12-31, 2002-12-31, 2002-12-31, 1997-12-11, 200...
```

There are a lot of columns, so I used `glimpse`. The three new variables we created are at the end of the list. They are correctly `Date`s, and they have the right values, the ones we can see at least.

The indentation is up to you. I think it's nice to make the creations of the three new variables line up. You can also make the opening and closing brackets on the long `mutate` aligned, or you can do as I have done here and put two closing brackets on the end. The rationale for this is that each of the variable definition lines in the `mutate` ends either with a comma or an extra closing bracket, the latter being on the last line. Your choice here is a matter of taste or (in your working life) the coding norms of the team you're working with.

You may have been offended by the repetition above. It so happens that these columns' names all end in `date` and they are the only ones that do, so we can use a "select helper" to select only them, and then submit all of them to a `mutate` via `mutate_at`, which goes like this:

```
whas %>% mutate_at(vars(ends_with("date")),funs(d=dmy)) %>% glimpse()
```

```
## Observations: 500
## Variables: 25
## $ id          <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
## $ age         <int> 83, 49, 70, 70, 70, 70, 57, 55, 88, 54, 48, 75, 48...
## $ gender      <int> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,...
## $ hr          <int> 89, 84, 83, 65, 63, 76, 73, 91, 63, 104, 95, 154, ...
## $ sysbp       <int> 152, 120, 147, 123, 135, 83, 191, 147, 209, 166, 1...
## $ diasbp      <int> 78, 60, 88, 76, 85, 54, 116, 95, 100, 106, 110, 12...
## $ bmi         <dbl> 25.54051, 24.02398, 22.14290, 26.63187, 24.41255, ...
## $ cvd         <int> 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,...
## $ afb         <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,...
## $ sho         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ chf         <int> 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,...
## $ av3         <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ miord       <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,...
## $ mitype      <int> 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,...
## $ year        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ admitdate   <chr> "13-01-1997", "19-01-1997", "01-01-1997", "17-02-1...
## $ disdate     <chr> "18-01-1997", "24-01-1997", "06-01-1997", "27-02-1...
## $ fdate       <chr> "31-12-2002", "31-12-2002", "31-12-2002", "11-12-1...
## $ los         <int> 5, 5, 5, 10, 6, 1, 5, 4, 4, 5, 5, 10, 7, 21, 4, 1,...
## $ dstat       <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ lenfol      <int> 2178, 2172, 2190, 297, 2131, 1, 2122, 1496, 920, 2...
## $ fstat       <int> 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1,...
## $ admitdate_d <date> 1997-01-13, 1997-01-19, 1997-01-01, 1997-02-17, 1...
## $ disdate_d   <date> 1997-01-18, 1997-01-24, 1997-01-06, 1997-02-27, 1...
## $ fdate_d     <date> 2002-12-31, 2002-12-31, 2002-12-31, 1997-12-11, 2...
```

One line, as you see, not three. The syntax of this is that we first say which columns we want to mutate. We can use any of the select-helpers for this, including listing the column numbers or names; in this case our date variables all ended with `date`. Then we have to give a function, or more than one function, to mutate them with; in this case we wanted to run all the dates-as-text through `dmy`. Because I said `d=dmy`, it takes the original date variable names, glues an underscore on the end and then the `d` that I said, so we create new variables by those names (at the end of the `glimpse` output). If I had just said `funs(dmy)`, we would have *overwritten* the original values, and `admitdate`, `disdate` and `fdate` would now be `date`s. Losing the original variables would have been OK here, but I wanted to show you how to create new variables.

(f) Create three new variables `diff1, diff2, diff3` that are the numbers of days between each of your dates, and save the data frame in which they have been created. Verify that at least some of them are the same as `los` and `lenfol`.

Solution: I don't know what R's internal storage is for dates (it might be seconds or milliseconds or anything, not necessarily days), so subtracting them requires care; you have to divide by the length of a day (in whatever units), thus:

```
whas3=whas2 %>% mutate(
                  diff1=(dis-admit)/ddays(1),
                  diff2=(f-admit)/ddays(1),
                  diff3=(f-dis)/ddays(1))
glimpse(whas3)
```

```
## Observations: 500
## Variables: 28
## $ id         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
## $ age        <int> 83, 49, 70, 70, 70, 70, 57, 55, 88, 54, 48, 75, 48, ...
## $ gender     <int> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1...
## $ hr         <int> 89, 84, 83, 65, 63, 76, 73, 91, 63, 104, 95, 154, 85...
## $ sysbp      <int> 152, 120, 147, 123, 135, 83, 191, 147, 209, 166, 160...
## $ diasbp     <int> 78, 60, 88, 76, 85, 54, 116, 95, 100, 106, 110, 123,...
## $ bmi        <dbl> 25.54051, 24.02398, 22.14290, 26.63187, 24.41255, 23...
## $ cvd        <int> 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0...
## $ afb        <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0...
## $ sho        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ chf        <int> 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1...
## $ av3        <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ miord      <int> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0...
## $ mitype     <int> 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1...
## $ year       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ admitdate <chr> "13-01-1997", "19-01-1997", "01-01-1997", "17-02-199...
## $ disdate   <chr> "18-01-1997", "24-01-1997", "06-01-1997", "27-02-199...
## $ fdate     <chr> "31-12-2002", "31-12-2002", "31-12-2002", "11-12-199...
## $ los        <int> 5, 5, 5, 10, 6, 1, 5, 4, 4, 5, 5, 10, 7, 21, 4, 1, 1...
## $ dstat      <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ lenfol     <int> 2178, 2172, 2190, 297, 2131, 1, 2122, 1496, 920, 217...
## $ fstat      <int> 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1...
## $ admit      <date> 1997-01-13, 1997-01-19, 1997-01-01, 1997-02-17, 199...
## $ dis        <date> 1997-01-18, 1997-01-24, 1997-01-06, 1997-02-27, 199...
## $ f          <date> 2002-12-31, 2002-12-31, 2002-12-31, 1997-12-11, 200...
## $ diff1      <dbl> 5, 5, 5, 10, 6, 1, 5, 4, 4, 5, 5, 10, 7, 21, 4, 1, 1...
## $ diff2      <dbl> 2178, 2172, 2190, 297, 2131, 1, 2122, 1496, 920, 217...
## $ diff3      <dbl> 2173, 2167, 2185, 287, 2125, 0, 2117, 1492, 916, 217...
```

The extra `d` on the front of `ddays` indicates that these are what is known to `lubridate` as "durations": a period of time 1 day long that could be any day (as opposed to "June 1, 1970" which is 1 day long, but tied to a particular day).

`los` should be the number of days in hospital, what I calculated as `diff1`, and `lenfol` should be the time from being admitted to last followup, which is my `diff2`. My output from `glimpse` confirms that.

Of course, checking that the first few values match is a nice confirmation, but is not actually a *proof*. For that, we should compare all 500 values, and it would be best to do it in such a way that R is comparing all 500 values for us, since it would be a lot more reliable than the human eye. R has a function `all.equal` which does exactly that. By way of warmup:

```
x=1:4
y=1:4
z=c(1,2,3,5)
all.equal(x,y)
```

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

```
all.equal(x,z)
```

```
## [1] "Mean relative difference: 0.25"
```

I thought the second one was just going to say `FALSE`, but it gave us a message instead, saying how close `x` and `z` were on average, so that we could decide whether they were close enough to call equal, or, as in this case, not.

Anyway:

```
with(whas3,all.equal(lenfol,diff2))
```

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

```
with(whas3,all.equal(los,diff1))
```

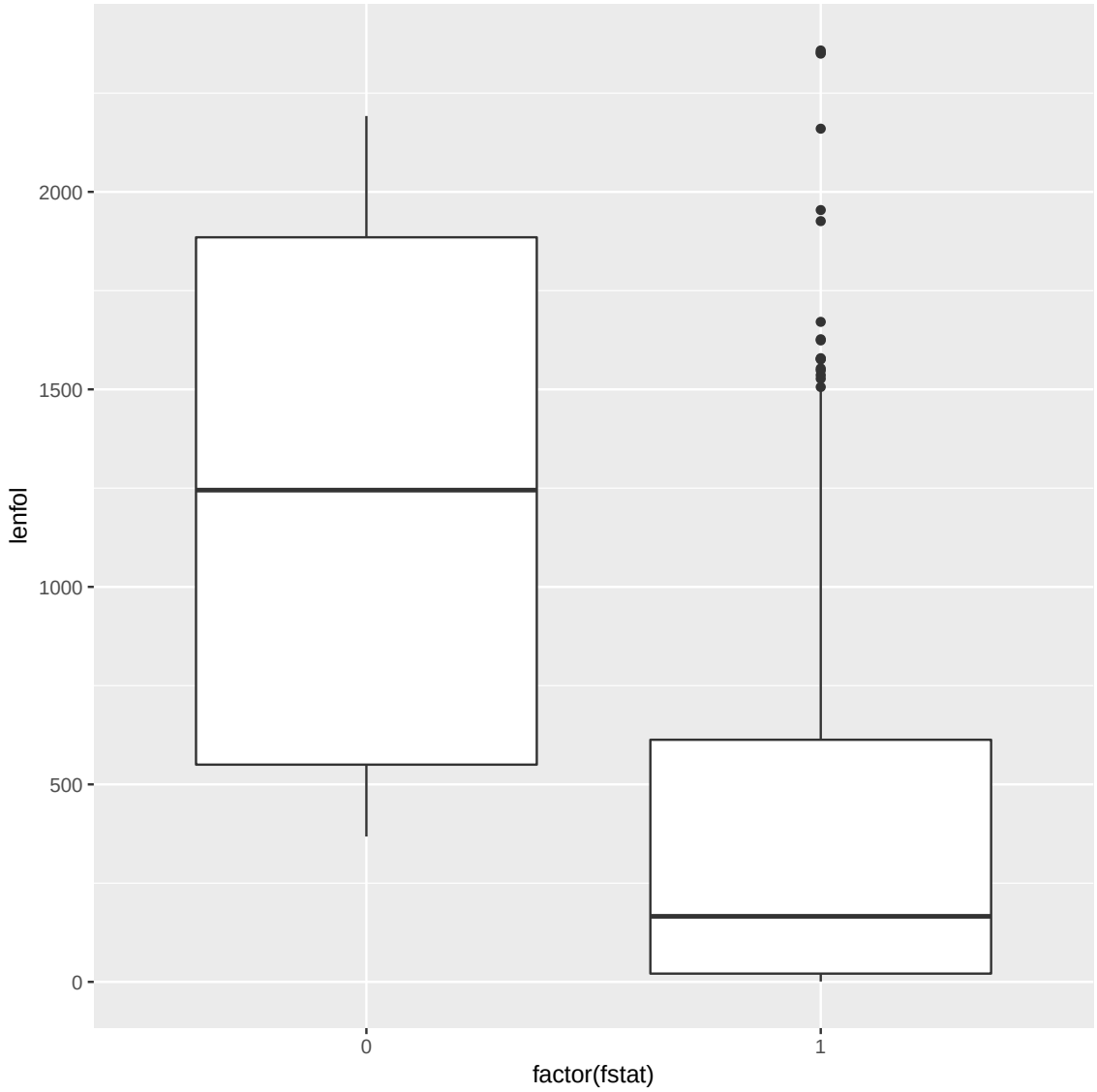
```
## [1] TRUE
```

so they really are all equal, all 500 of them.[20]

(g) Construct side-by-side boxplots of the length of followup by each followup status. (This should come out the same as the corresponding boxplot you did with SAS.) You'll need to make sure that the followup status, as it gets fed into `ggplot`, is a `factor`.

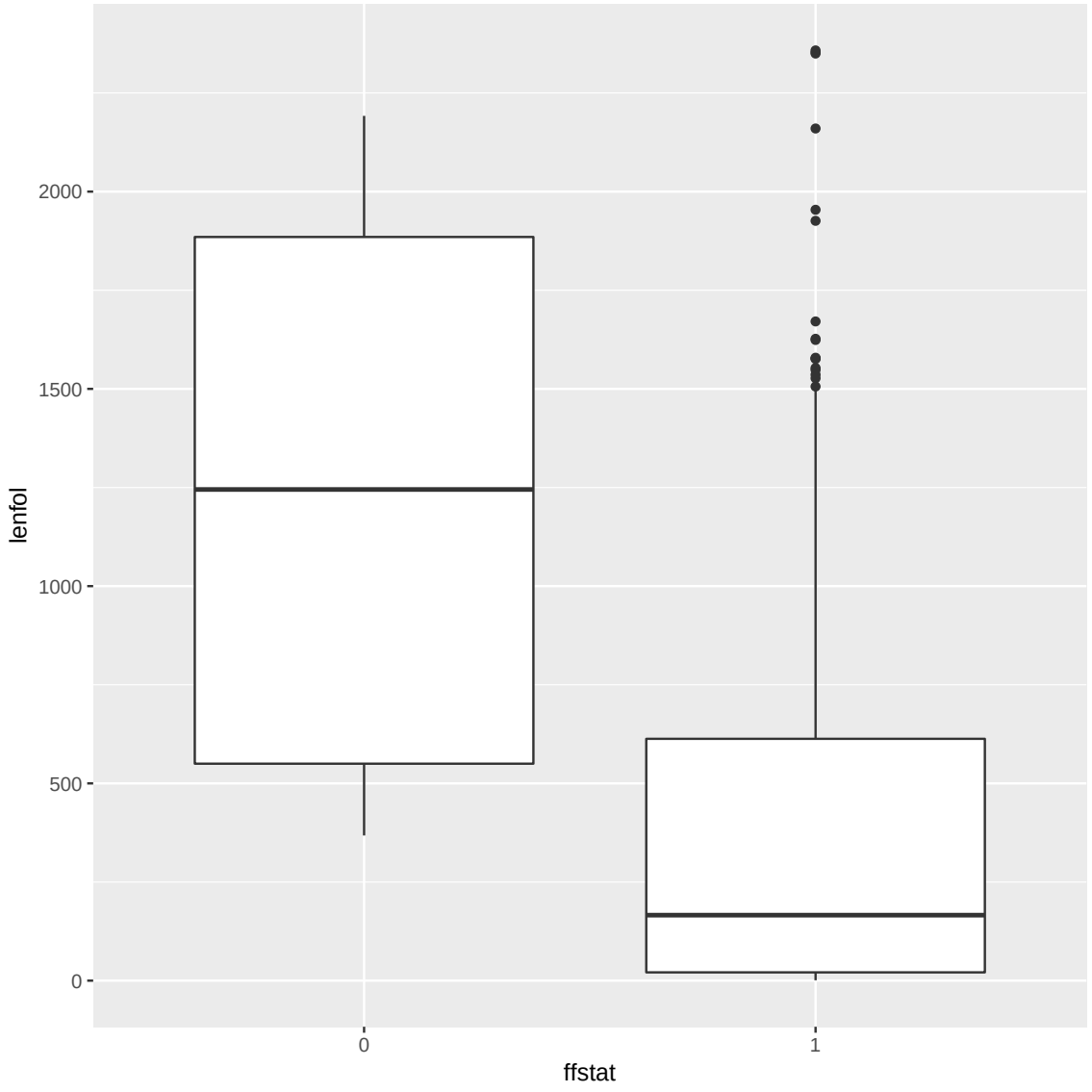
Solution: The easiest way to make a factor is to wrap `fstat`, which is a numeric 0 or 1, in `factor()`:

```
ggplot(whas3,aes(x=factor(fstat),y=lenfol))+geom_boxplot()
```



Or create a factor `fstat` first:

```
whas3 %>% mutate(ffstat=factor(fstat)) %>%
  ggplot(aes(x=ffstat,y=lenfol))+geom_boxplot()
```



Indeed, as in SAS. I think the second way looks better, because you get a cleaner x-axis on your plot. But if you're doing this for exploration, rather than as something that's going to appear in a report for your boss, the first way is fine.

Notes

[1]This was the actual reason I thought of this question originally: I wanted you to do this.

[2]And it shows the value of looking at relevant plots.

[3]Mistakenly.

[4]Mixes up.

[5]This is actually grammatically correct.

[6]Though it's hard to imagine being able to improve on an R-squared of 99%.

[7]This wouldn't have told us about the overall effect of `species`.

[8]This is actually an accident, since the regression *assumes* each observation has the same variance, and so the regression will always correspond to the pooled test in this situation, regardless of whether the pooled test is actually the right thing to do.

[9]I made a whole set of residual plots this way while on the bus, in case this was the way it had to be done..

[10]In fact, it is believed that wolves help keep caribou herds strong by preventing over-population: that is, the weakest caribou are the ones taken by wolves.

[11]The survey is always taken in the fall, but the date varies.

[12]Counting animals in a region, especially rare, hard-to-find animals, is a whole science in itself. These numbers are probably estimates (with some uncertainty).

[13]Remember that?

[14]Because the wolves have more to eat.

[15]Baby wolves.

[16]Ken makes a rude noise.

[17]The two dashes between the variable names are like `:` in R: "the first one through the second one, inclusive."

[18]Or, more likely, one of the doctor's sleep-deprived surgical residents.

[19]The same Sir David Cox of Box-Cox.

[20]The computer scientists among you will note that I shouldn't have done this, because `diff1` through `diff3` are double-precision decimal numbers, so I should have tested their equality with `lenfol` and `los` by working out the absolute differences and testing whether they were all *small*. On consulting the help for `all.equal`, though, I find that it *does* work properly, because it actually tests whether the things being compared differ by less than a quantity `tolerance` which defaults to 1.5×10^{-8}, and if they do it calls them equal. This is all tied in with the difference between integers and decimal numbers as they are represented on a computer: exactly and approximately, respectively. A double-precision number has about 16 significant digits of accuracy; equal things won't have all 16 digits equal, most likely, but they would be expected to have at least 8 of those digits the same. CSCA08 stuff, I imagine. This is where you can casually toss around terms like "machine epsilon". Oh! I just realized something. You know how very very small P-values are shown in R as `<2.2e-16`? *That's* the machine epsilon, 2.2×10^{-16}. Anything smaller than that is indistinguishable from zero, and you can't have a P-value be *exactly* zero. The default `tolerance` I mentioned above is the square root of this, which is normally used for such things.