Assignment 2

1. Forty students apply to a prestigious MBA program. Some of them are admitted, and some of them are not. For each student, three other things are recorded:

   - score on GMAT (the "graduate management admission test"), a standardized test, according to Wikipedia, "intended to assess certain analytical, writing, quantitative, verbal, and reading skills in written English for use in admission to a graduate management program, such as an MBA program". Scores are between 200 and 800.
   - grade-point average in undergraduate program
   - number of years of work experience (here, between 1 and 6).

   We want to see whether any or all of these explanatory variables can be used to predict whether a student will be admitted to the MBA program. The data are in http://ritsokiguess.site/STAD29/admitted.txt.

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

   Solution:

   Take a look at the data first: separated by single spaces, so:

```
my_url <- "http://ritsokiguess.site/STAD29/admitted.txt"
mba <- read_delim(my_url, " ")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   gmat = col_double(),
##   gpa = col_double(),
##   work_experience = col_double(),
##   admitted = col_character()
## )
```

```
mba
```

```
## # A tibble: 40 x 4
##     gmat   gpa work_experience admitted
##    <dbl> <dbl>           <dbl> <chr>
##  1   780   4                   3 yes
##  2   750   3.9                 4 yes
##  3   690   3.3                 3 no
##  4   710   3.7                 5 yes
##  5   680   3.9                 4 no
##  6   730   3.7                 6 yes
##  7   690   2.3                 1 no
##  8   720   3.3                 4 yes
##  9   740   3.3                 5 yes
## 10   690   1.7                 1 no
## # ... with 30 more rows
```

   Forty rows, with the columns as promised.

(b) Explain briefly why logistic regression will be suitable here.

Solution:

The response variable `admitted` is categorical (one point) with two categories (the second point). Another way to say it is that we are predicting a categorical response, namely whether or not an applicant is admitted. This implies that there are two categories, because you named them.

"The response variable is categorical" is only 0.5, if that's all you say. You didn't say what the response variable was, and you didn't say how many categories it has; if it has more than two, another flavour of logistic regression is needed, one that we learn about later.

(c) Explain briefly why R's logistic regression function will predict the probability that a student *is* admitted.

Solution:

The column `admitted` is categorical. These are taken in alphabetical order; `no` is the baseline, and so we predict the probability of the other category, `yes`.

"Because `yes` is after `no` alphabetically" does not show enough insight: why does that matter?

(d) Fit a suitable logistic regression model and display the output.

Solution:

Predict `admitted` from the other variables, which are all quantitative. You will need to turn `admitted` into a factor, since it is not 0 and 1 (you will get a message telling you so if you forget).

```
mba.1 <- glm(factor(admitted)~gmat+gpa+work_experience, data = mba, family = binomial)
summary(mba.1)
```

```
##
## Call:
## glm(formula = factor(admitted) ~ gmat + gpa + work_experience,
##     family = binomial, data = mba)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.16716  -0.21204  -0.06016   0.33273   2.13591
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -22.02334    9.08493  -2.424   0.0153 *
## gmat              0.01572    0.01458   1.078   0.2810
## gpa               2.45454    2.00786   1.222   0.2215
## work_experience   1.00270    0.57336   1.749   0.0803 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 17.130  on 36  degrees of freedom
## AIC: 25.13
##
## Number of Fisher Scoring iterations: 6
```

You need both `glm` and the `family` piece, or else you will have fitted the wrong model.

(e) Does the sign (positive or negative) of each of the Coefficients, not counting the intercept, make practical sense for these data? Explain briefly.

Solution:

They are all positive. This means that a higher value on any of them is associated with a higher probability of being admitted to this program. That in turn makes sense because to be admitted to something like an MBA, you will need a high GPA, a high score on the standardized test, and more years of work experience. In all of those, the more you have, you would guess the higher your chances of being admitted.

Extra 1: the very negative intercept is talking about a student with a 0 GPA, 0 score on the GMAT, and no work experience at all. Such a student evidently would have no chance of being admitted to this program, and the very negative intercept says that the predicted *log-odds* of their being admitted is very negative: that is, the predicted probability is very close to zero. (This is, of course, extrapolation, since none of the actual applicants have a profile anything like that, but that's what it means.)

Extra 2: None of those P-values are at all small, which means that none of the actual slopes are significantly different from zero in this model, so it might be that we were actually lucky in getting positive slopes. We should eliminate non-significant explanatory variables, one at a time, and see whether anything becomes significant. That's next.

(f) Which, if any, explanatory variables need to be kept and which one(s), if any, need to be removed? Briefly describe the process that you used in order to decide. (Use $\alpha = 0.10$ here.)

Solution:

There are several different ideas you might pursue. The most obvious one is to remove explanatory variables one by one, until everything that remains is significant. `update` is the smoothest way to do this, since this way you don't have to type or copy the `data=` and `family=` every time:

```
mba.2 <- update(mba.1, .~.-gmat)
summary(mba.2)
```

```
##
## Call:
## glm(formula = factor(admitted) ~ gpa + work_experience, family = binomial,
##     data = mba)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.44411  -0.24046  -0.03102   0.32833   1.77084
```

```
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -15.5729     5.6086  -2.777  0.00549 **
## gpa               3.6924     1.7578   2.101  0.03567 *
## work_experience   1.0269     0.5074   2.024  0.04298 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 18.482  on 37  degrees of freedom
## AIC: 24.482
##
## Number of Fisher Scoring iterations: 6
```

The two explanatory variables that remain are both significant, so here we stop.[1]

You might remember that another way to do this whole process, no matter how many steps it has, is via `step`:

```
mba.3 <- step(mba.1, test="Chisq")
```

```
## Start:  AIC=25.13
## factor(admitted) ~ gmat + gpa + work_experience
##
##                   Df Deviance    AIC    LRT Pr(>Chi)
## - gmat             1   18.482 24.482 1.3516  0.24500
## <none>                 17.131 25.131
## - gpa              1   19.231 25.231 2.1002  0.14728
## - work_experience  1   21.456 27.456 4.3260  0.03753 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=24.48
## factor(admitted) ~ gpa + work_experience
##
##                   Df Deviance    AIC     LRT Pr(>Chi)
## <none>                 18.482 24.482
## - work_experience  1   24.575 28.575  6.0933 0.013570 *
## - gpa              1   28.753 32.753 10.2704 0.001352 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(mba.3)
```

```
##
## Call:
## glm(formula = factor(admitted) ~ gpa + work_experience, family = binomial,
##     data = mba)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
```

```
## -2.44411  -0.24046  -0.03102   0.32833   1.77084
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -15.5729     5.6086  -2.777  0.00549 **
## gpa               3.6924     1.7578   2.101  0.03567 *
## work_experience   1.0269     0.5074   2.024  0.04298 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 18.482  on 37  degrees of freedom
## AIC: 24.482
##
## Number of Fisher Scoring iterations: 6
```

If you go this way, which is fine, make sure to describe what happened: `gmat` score was the least significant and was removed, and then the two remaining explanatory variables were both significant, so the process stopped there.

Extra on this: you'll notice that the P-values in `step` and the ones at the end in `summary` are *different*. In a regression, they will be the same, but in other models they may not be. Here, the ones in `step` are likelihood ratio tests, and the ones in `summary` are Wald tests. If you have a small sample size, as we do here (40 observations is not many when each one only gives you a yes or a no in its response), they might look more different than you would like. The theory is that they are *asymptotically equal*, meaning that they would be identical if you had an infinitely large sample, and if you have a large sample they will be close, but if you have a small sample, they may not be similar. My experience is that the Wald tests tend to have larger P-values.

Another good approach, if you do it properly, is to look at the model we started with:

```
summary(mba.1)
```

```
##
## Call:
## glm(formula = factor(admitted) ~ gmat + gpa + work_experience,
##     family = binomial, data = mba)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.16716  -0.21204  -0.06016   0.33273   2.13591
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -22.02334    9.08493  -2.424   0.0153 *
## gmat              0.01572    0.01458   1.078   0.2810
## gpa               2.45454    2.00786   1.222   0.2215
## work_experience   1.00270    0.57336   1.749   0.0803 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 17.130  on 36  degrees of freedom
## AIC: 25.13
##
## Number of Fisher Scoring iterations: 6
```

and see what happens if you take out *both* of the non-significant x-variables, remembering that I said α was 0.10, so that `work_experience` is significant and should stay:

```
mba.4 <- update(mba.1, .~. -gmat-gpa)
summary(mba.4)
```

```
##
## Call:
## glm(formula = factor(admitted) ~ work_experience, family = binomial,
##     data = mba)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4263  -0.4215  -0.2054   0.4943   2.2203
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -5.3204     1.6830  -3.161 0.001571 **
## work_experience   1.4722     0.4406   3.341 0.000834 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 28.753  on 38  degrees of freedom
## AIC: 32.753
##
## Number of Fisher Scoring iterations: 5
```

Because you took out *more than one* explanatory variable, you now need to test whether this was too much. To do that, use `anova` to compare the fits of the two models, smallest first:

```
anova(mba.4, mba.1, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: factor(admitted) ~ work_experience
## Model 2: factor(admitted) ~ gmat + gpa + work_experience
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        38     28.753
## 2        36     17.131  2   11.622 0.002994 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is a significant difference here, which means that the bigger model is better, so that

taking out both explanatory variables was too much. From here, go back to my first method, taking out only `gmat`, and see what happens.

Extra continues: for regular regression, you can use `leaps` to find the model with, say, the best adjusted R-squared. Something similar for logistic regression is called `bestglm`: it is a package that you have to install, containing a function of the same name. There's a bit of setup involved: the response has to be a genuine 1 for success and 0 for failure, *and* it has to be the *last* column in the dataframe.[2] Oh, and the data has to be in an actual `data.frame` (not a tibble). This is all because `bestglm` was written a long time ago, before the `tidyverse` existed. But the `tidyverse` gives us tools to make what we want:

```
library(bestglm)
```

```
## Loading required package: leaps
```

```
mba %>% mutate(adm=case_when(
  admitted=="yes" ~ 1,
  admitted=="no"  ~ 0,
  TRUE            ~ -1)) %>%
  select(-admitted) %>% as.data.frame() -> mbax
mbax
```

```
##    gmat gpa work_experience adm
## 1   780 4.0               3   1
## 2   750 3.9               4   1
## 3   690 3.3               3   0
## 4   710 3.7               5   1
## 5   680 3.9               4   0
## 6   730 3.7               6   1
## 7   690 2.3               1   0
## 8   720 3.3               4   1
## 9   740 3.3               5   1
## 10  690 1.7               1   0
## 11  610 2.7               3   0
## 12  690 3.7               5   1
## 13  710 3.7               6   1
## 14  680 3.3               4   0
## 15  770 3.3               3   1
## 16  610 3.0               1   0
## 17  580 2.7               4   0
## 18  650 3.7               6   1
## 19  540 2.7               2   0
## 20  590 2.3               3   0
## 21  620 3.3               2   1
## 22  600 2.0               1   0
## 23  550 2.3               4   0
## 24  550 2.7               1   0
## 25  570 3.0               2   0
## 26  670 3.3               6   1
## 27  660 3.7               4   1
## 28  580 2.3               2   0
## 29  650 3.7               6   1
## 30  660 3.3               5   1
```

```
## 31  640 3.0              1   0
## 32  620 2.7              2   0
## 33  660 4.0              4   1
## 34  660 3.3              6   1
## 35  680 3.3              5   1
## 36  650 2.3              1   0
## 37  670 2.7              2   0
## 38  580 3.3              1   0
## 39  590 1.7              4   0
## 40  690 3.7              5   1
```

You see all 40 lines because it's a `data.frame`. At least, you do here. I used `case_when` to make the new column, called `adm`: if `admitted` is `yes`, then `adm` is 1; if it's `no` it's 0, and if it happens to be anything else it's -1 (just to check for errors, though there aren't any). The nice thing about `mutate` is it puts the new column on the end, exactly where we want it to be.

So now to business. We'll get the best model by AIC, which is also defined for generalized linear models (adjusted R-squared is only defined for regular regression):

```
bestglm(mbax, family = binomial, IC="AIC")
```

```
## Morgan-Tatar search since family is non-gaussian.
```

```
## AIC
## BICq equivalent for q in (0.231082205199096, 0.762898383783735)
## Best Model:
##                   Estimate Std. Error   z value    Pr(>|z|)
## (Intercept)     -15.572928  5.6086119 -2.776610 0.005492904
## gpa               3.692430  1.7577547  2.100651 0.035671586
## work_experience   1.026883  0.5073737  2.023919 0.042978490
```

The same thing that backward elimination gave us.

(g) Some new students are shown in http://ritsokiguess.site/STAD29/admitted2.txt, in the same format as the original data. We don't know whether or not these students have been admitted. Read in the data for these new students, and obtain and display predicted probabilities of being admitted for each student, based on your best model, side by side with the data they are predictions for.

Solution:

Several things to do here. First, a `read_delim` again:

```
my_url <- "http://ritsokiguess.site/STAD29/admitted2.txt"
new <- read_delim(my_url, " ")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   gmat = col_double(),
##   gpa = col_double(),
##   work_experience = col_double()
## )
```

```
new
```

```
## # A tibble: 5 x 3
##    gmat   gpa work_experience
##   <dbl> <dbl>           <dbl>
## 1   590   2                3
## 2   740   3.7              4
## 3   680   3.3              6
## 4   610   2.3              1
## 5   710   3                5
```

Five new students, admission status not known, but with values for the other variables, so we can predict their probabilities of being admitted. To do so, use `predict`, with input the best model you found and this new data frame. (Note that the column names in this one are exactly the same as in the original.) Don't forget the `type` to get predicted *probabilities*, otherwise you'll get log-odds:

```
mba_pred <- predict(mba.2, new, type = "response")
cbind(new, mba_pred)
```

```
##   gmat gpa work_experience    mba_pred
## 1  590 2.0               3 0.006015283
## 2  740 3.7               4 0.899943382
## 3  680 3.3               6 0.941220840
## 4  610 2.3               1 0.002344219
## 5  710 3.0               5 0.654477204
```

Extra: if you forget what the `type` option value needs to be,[3] fill in something obviously silly and the message will remind you what you need:

```
predict(mba.2, new, type = "bananas")
```

```
## Error in match.arg(type): 'arg' should be one of "link", "response", "terms"
```

This ought to be enough of a reminder to prompt you towards `response`. If it isn't, try them one at a time and see what you get:

```
predict(mba.2, new, type = "link")
```

```
##          1          2          3          4          5
## -5.1074184  2.1965956  2.7733904 -6.0534561  0.6387781
```

Log-odds, the same as if you miss out `type` entirely:

```
predict(mba.2, new)
```

```
##          1          2          3          4          5
## -5.1074184  2.1965956  2.7733904 -6.0534561  0.6387781
```

The other one:

```
predict(mba.2, new, type = "terms")
```

```
##          gpa work_experience
## 1 -4.0432106      -0.4364254
## 2  2.2339200       0.5904579
## 3  0.7569481       2.6442246
## 4 -2.9354817      -2.4901921
## 5 -0.3507808       1.6173413
```

```
## attr(,"constant")
## [1] -0.6277823
```

For each individual, add up the (log-odds) values for the constant and each of the two explanatory variables. This gives the same log-odds as above. I come back to this one below.[4]

(h) Looking at your predictions, discuss briefly whether each student is likely to be admitted or not, and whether that makes sense, given their values on the other variables.

Solution:

You can take the students in the order they are listed, or discuss the students who are (eg.) likely to be admitted all together. Remember that our best model *does not* have `gmat` in it, so don't include that in your discussion:

- Students 2 and 3 are very likely to be admitted. They both have high GPAs, and student 3 has a lot of work experience to make up for the slightly lower GPA.
- Students 1 and 4 are very unlikely to be admitted. They have very low GPAs (probably the key thing here), and not very much work experience, especially student 4.
- Student 5 has what you might call a moderate chance of getting in. Their GPA seems on the low side for getting into an MBA program, but they have lots of work experience, which presumably helps.

Extra 1: let's return to the predictions with `terms` that I got above, this time beside the data:

```
p <- predict(mba.2, new, type = "terms")
cbind(new, p)
```

```
##   gmat gpa work_experience        gpa work_experience
## 1  590 2.0               3 -4.0432106      -0.4364254
## 2  740 3.7               4  2.2339200       0.5904579
## 3  680 3.3               6  0.7569481       2.6442246
## 4  610 2.3               1 -2.9354817      -2.4901921
## 5  710 3.0               5 -0.3507808       1.6173413
```

These predictions (the last two columns here) give the "contributions" of each student's GPA and work experience to their log-odds of being admitted. For example, students 1 and 4 with the low GPAs have a very negative number in the (second) `gpa` column, which shows you directly what effect this alone is having on the chances of admission.[5] You can also see directly that student 2 has a good chance of admission mainly because of their GPA, and student 3's good chance of admission comes mostly from all the work experience they have.

As for student 5, if it were based on GPA alone, they would have a less than even chance of being admitted: a negative contribution, about -0.63, from the intercept that we saw above, plus a negative contribution from `gpa`, means that the log-odds of the admission probability would be less than 0, so the probability itself would be less than 0.5. However, the strong positive contribution to the log-odds from the high work experience for that student tips the balance in their favour.

Extra 2: I said earlier that I expected our three explanatory variables to be correlated, in that a student with a high value on one would tend to have a high value on the others. Does that seem to be true?

```
mba %>% select(-admitted) %>% cor()
```

```
##                       gmat       gpa work_experience
## gmat            1.0000000 0.5613050       0.4040216
## gpa             0.5613050 1.0000000       0.5863758
## work_experience 0.4040216 0.5863758       1.0000000
```

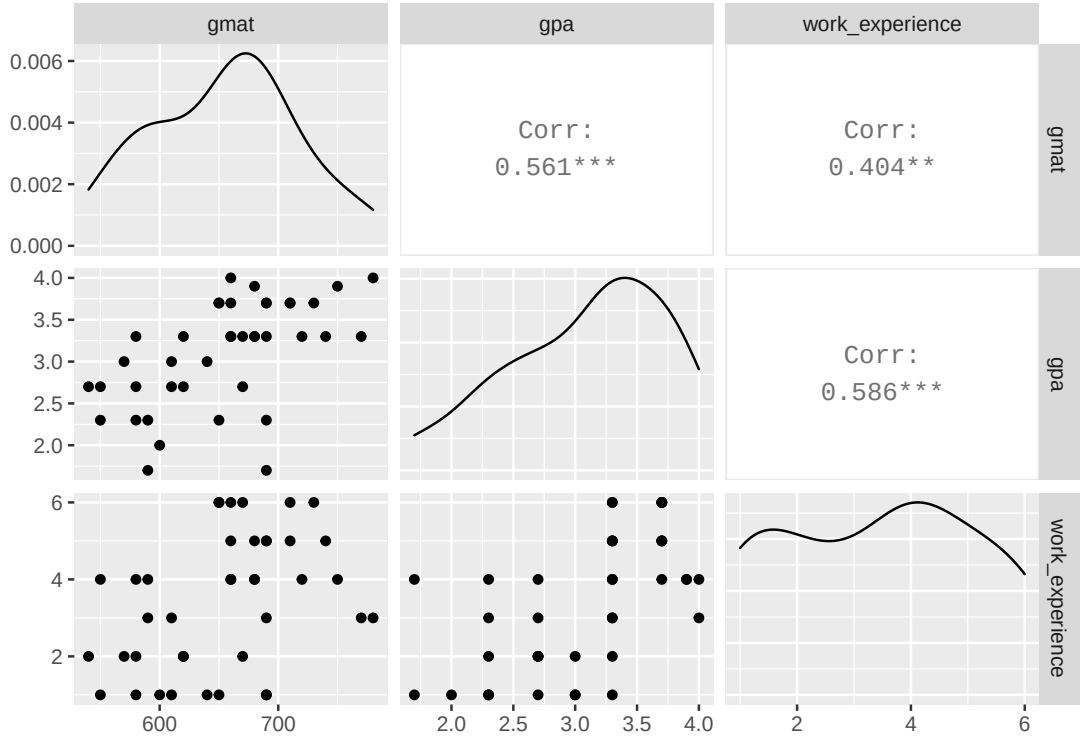
At least moderately so. I guess this is enough to make the P-values depend on which of them are in the model. Note that `cor` takes a data frame, and the one we want to use is `mba` with the categorical column removed, so we can stick it on the end of a pipeline.

The package `GGally` has a nice rendition of a "pairs plot": that is, scatterplots of each pair of variables. This one has some extras:

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
mba %>% select(-admitted) %>% ggpairs()
```



We see:

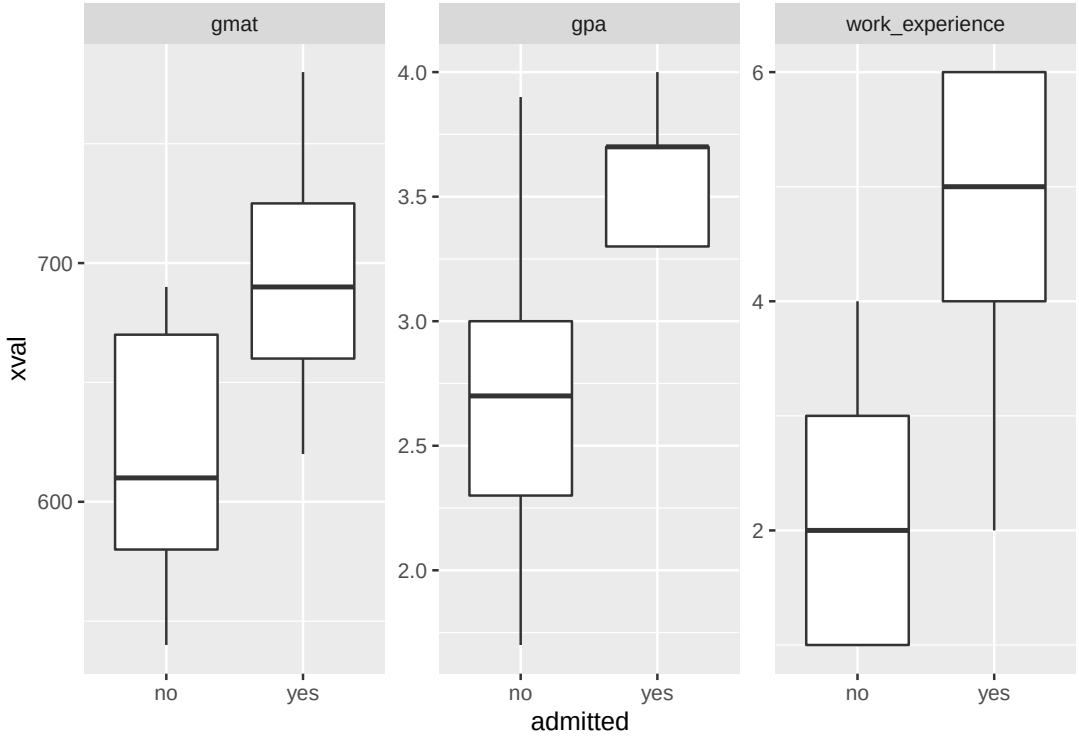
- bottom left the three scatterplots for the three pairs of explanatory variables,
- on the diagonal a density plot (smoothed-out histogram) of each variable,
- top right the correlations, with an indication of whether each one is significantly different from zero or not (these all are, with two of the P-values less than 0.001).

The GPA values are left-skewed (you'd imagine that most applicants to an MBA program will have high GPAs), but the work experience values are all over the place. The scatterplots are

the kind of thing you'd expect with moderate correlations, each showing a bit of an upward trend. With 40 observations on each variable, a correlation doesn't have to be far from zero to be significantly nonzero. (In *this* context, 40 is a decent sample size because the variables we're thinking about are quantitative.)

Another thing you might look at is association between the response (being admitted or not) and the explanatory variables. This time, the response is categorical, so the right plot is a trio of boxplots. Let me see whether I can get them all at once:

```
mba %>% pivot_longer(-admitted, names_to="xname", values_to="xval") %>%
  ggplot(aes(x=admitted, y=xval)) + geom_boxplot() + facet_wrap(~xname, scales = "free")
```



The people who get admitted tended to have a clearly higher GPA and amount of work experience. Their `gmat` score also tended to be higher, but not so clearly. This might be the reason we removed `gmat` from the model. It is also possible that once you know the other two explanatory variables, you can guess something about `gmat` score, and thus knowing the actual `gmat` score doesn't really add anything to the prediction.

This was attempt 2 to get this graph. The first time, I forgot the `scales="free"` and the second and third boxplots were smooshed up against zero. The `pivot_longer` is the same trick used to make scatterplots of response against all the x-variables in ordinary regression.

2. A survey asked 5381 people "Do you think that what the government is doing for people in poverty in this country is about the right amount, too much, or too little?". The survey response is called `poverty` in our dataframe. In addition, some demographic variables were measured for each respondent:

   - `religion`: member of a religious group (`yes` or `no`)
   - `degree`: holds a university degree (`yes` or `no`)
   - `country`: where the respondent lives (Australia, Norway, Sweden, USA)
   - `age`: in years (quantitative)

- `gender`: identifies as `male` or `female`

(a) The data is in a package called `carData`. Make sure you have installed this package (you might already have it if you installed `car`), and load it with `library`. The data set is called WVS. Display the first few lines of it.

Solution:

Thus:

```
library(carData)
```

Just typing `WVS` on the next line might display the first ten lines for you. If it does, you are done. If it doesn't (it displayed a lot of lines for me), you have some options:

```
head(WVS)
```

```
##       poverty religion degree country age gender
## 1  Too Little      yes     no     USA  44   male
## 2 About Right      yes     no     USA  40 female
## 3  Too Little      yes     no     USA  36 female
## 4    Too Much      yes    yes     USA  25 female
## 5  Too Little      yes    yes     USA  39   male
## 6 About Right      yes     no     USA  80 female
```

which displays the first six lines (by default), or something like

```
WVS %>% slice(1:8)
```

```
##       poverty religion degree country age gender
## 1  Too Little      yes     no     USA  44   male
## 2 About Right      yes     no     USA  40 female
## 3  Too Little      yes     no     USA  36 female
## 4    Too Much      yes    yes     USA  25 female
## 5  Too Little      yes    yes     USA  39   male
## 6 About Right      yes     no     USA  80 female
## 7    Too Much      yes     no     USA  48 female
## 8  Too Little      yes     no     USA  32   male
```

Whatever you put into the `slice`, those lines will get displayed. Any of these, or anything else that displays a small number of lines, is fine.

You might note for later that there are actually 5381 rows in the dataframe:

```
nrow(WVS)
```

```
## [1] 5381
```

so that each row is *one* person's response and demographics, and there will be no need for a `weights`. (Also, there is no frequency column, so each row had better be one person.) That means that this one will look a bit different from the coal miners one in lecture.

(b) Explain briefly why using `polr` from `MASS` will make sense to analyze these data.

Solution:

`polr` fits ordinal logistic regression, that is to say where the response variable is categorical and the categories have an order. Our response variable is `poverty`, which is categorical with

three categories, "too little", "about right" and "too much", which come in order (either this way around or reversed, but "about right" needs to be in the middle.)

I am looking for evidence that you know:

- what `polr` is good for, that is, a response variable with ordered categories
- what the response variable is *here*
- that `poverty` has ordered categories (and how you know).

(c) Count the total number of people who responded with each `poverty` category. Do the categories appear to have come out in a sensible order? Explain briefly.

Solution:

This is, literally, `count`:

```
WVS %>% count(poverty)
```

```
##       poverty    n
## 1  Too Little 2708
## 2 About Right 1862
## 3    Too Much  811
```

Usually `count` will display the categories in alphabetical order, but I think because we have an ordered factor, they will be displayed in the correct order (that is, the order that was originally given for them). The levels are too little, about right, too much *in that order*. This is a sensible order for the levels to be in, smallest to largest on this scale.

(d) Fit a suitable model and display the output. (The output is not very illuminating, but this is a way of being sure that you have done the right thing.)

Solution:

This is (evidently) `polr`, with the model formula and `data=` that you would expect:

```
WVS.1 <- polr(poverty ~ religion + degree + country + age + gender, data=WVS)
summary(WVS.1)
```

```
##
## Re-fitting to get Hessian
```

```
## Call:
## polr(formula = poverty ~ religion + degree + country + age +
##     gender, data = WVS)
##
## Coefficients:
##                   Value Std. Error t value
## religionyes     0.17973   0.077346   2.324
## degreeyes       0.14092   0.066193   2.129
## countryNorway -0.32235   0.073766  -4.370
## countrySweden -0.60330   0.079494  -7.589
## countryUSA      0.61777   0.070665   8.742
## age             0.01114   0.001561   7.139
## gendermale      0.17637   0.052972   3.329
```

```
##
## Intercepts:
##                          Value   Std. Error t value
## Too Little|About Right  0.7298  0.1041     7.0128
## About Right|Too Much    2.5325  0.1103    22.9496
##
## Residual Deviance: 10402.59
## AIC: 10420.59
```

We will do some predictions later to understand those coefficients (which is the best way to understand them). I discuss what they mean after we have done the predictions; until then, there is not much value in looking at these.

What happens if you just display the fitted model object, rather than its `summary`?

```
WVS.1
```

```
## Call:
## polr(formula = poverty ~ religion + degree + country + age +
##     gender, data = WVS)
##
## Coefficients:
##   religionyes     degreeyes countryNorway countrySweden    countryUSA
##    0.17973194    0.14091745   -0.32235359   -0.60329785    0.61777260
##           age    gendermale
##    0.01114091    0.17636863
##
## Intercepts:
## Too Little|About Right   About Right|Too Much
##              0.7297635              2.5324787
##
## Residual Deviance: 10402.59
## AIC: 10420.59
```

This displays a lot of the same numbers. It's enough to show that you fitted the right model, so for this part is absolutely fine.

(e) Provide a good indication that you should not remove any of your explanatory variables from your model. Obtain suitable output and explain briefly. (Something to consider is that `country` is a categorical variable with four levels.)

Solution:

The summary output above is not the tool for this job, because it has no P-values. You might read the t-values as if they were z-scores and say something about how they are all bigger than 2 in absolute value, and therefore all the explanatory variables are (apparently) significant, but we can do better. One reason is that `country` is a categorical variable with 4 levels, and so comparing each country with the baseline Australia would not be sensible even if we did have P-values.

The hint, or this last consideration, ought to push you towards `drop1`. The right `test` to use with this one is `Chisq`, since we are no longer in linear-model territory, like regression or ANOVA, where the response variable has a normal distribution given the values of the explanatory variables:

```
drop1(WVS.1, test="Chisq")
```

```
## Single term deletions
##
## Model:
## poverty ~ religion + degree + country + age + gender
##          Df   AIC     LRT  Pr(>Chi)
## <none>      10421
## religion  1 10424   5.434  0.019753 *
## degree    1 10423   4.518  0.033542 *
## country   3 10666 250.881 < 2.2e-16 ***
## age       1 10470  51.120  8.69e-13 ***
## gender    1 10430  11.096  0.000865 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

All of the explanatory variables are significant, since their P-values are all less than 0.05. So we should not remove any of them from the model. Note that, properly, `country` has 3 degrees of freedom, because there are 4 countries, and "a country effect" means that not all the countries are the same in terms of attitudes towards poverty, allowing for the effects of the other variables.

If you forget the `test=`, you won't get any P-values, and those are what you really wanted, so including the right `test=` is best. Without that, you will need to say something about "removing nothing is best", and to say something about "best" having to do with AIC, since that is all you have without the P-values.

(f) We will be comparing the effects of country on attitudes towards poverty. To do this, (i) make a dataframe with all the values of country, along with age 43 (the median age), gender male, religion "yes" and no university degree. Then, (ii), use it to obtain predicted probabilities of attitudes towards poverty for people from each country.

Solution:

First, make a data frame with those things in it. I did it with `crossing`, but any of the methods in Extra 4 below will work:

```
countries <- levels(WVS$country)
countries
```

```
## [1] "Australia" "Norway"    "Sweden"    "USA"
```

```
new <- crossing(country = countries,
                age = 43,
                gender = "male",
                religion = "yes",
                degree = "no")
new
```

```
## # A tibble: 4 x 5
##   country     age gender religion degree
##   <chr>     <dbl> <chr>  <chr>    <chr>
## 1 Australia    43 male   yes      no
## 2 Norway       43 male   yes      no
```

```
## 3 Sweden       43 male   yes      no
## 4 USA          43 male   yes      no
```

Getting to here, somehow, is the key to (i). I don't mind much what way you find to do it, as long as it comes from this course or C32.

Some strategy tips:

- when I have several distinct values (here for the four different countries), I like to list them out separately first, defining a vector with a plural name (my habit). This is because the `crossing` can get long and complicated if I put the four country names in it, and doing it this way makes it easier to read. (You are treating "future you" or your collaborators at work with respect here.)
- in this case, `country` is a `factor` in the original dataframe (go back and look for a `fctr` at the top of the column), and so, instead of typing out the values, I can get hold of them using `levels`. There is nothing wrong with typing out the values; it will just take you longer and you will need to get them right.

Next up, (ii), the predictions:

```
p <- predict(WVS.1, new, type = "probs")
cbind(new, p)
```

```
##     country age gender religion degree Too Little About Right  Too Much
## 1 Australia  43   male      yes     no  0.4736753   0.3715090 0.1548158
## 2    Norway  43   male      yes     no  0.5540276   0.3288196 0.1171529
## 3    Sweden  43   male      yes     no  0.6219661   0.2869620 0.0910719
## 4       USA  43   male      yes     no  0.3266955   0.4197126 0.2535919
```

That is as far as we need to go in this part.

(g) Compare the countries in terms of attitudes towards government support for people in poverty. Discuss briefly.

Solution:

The general tendency in most of the countries, but particularly the Scandinavian ones (Norway and Sweden), is that people think their government gives too little support to the poor, rather than too much (for the most part). The USA is different, in that there is a more substantial proportion of people thinking that the poor get too much support from the government. Australia is in between the Scandinavian countries and the USA.

I am looking for an overall picture like this. You need to look beyond the individual numbers to see what story is being told here. That is to say, look at all the columns of predictions and deduce an overall story.

That is the end of the assignment for you, but I thought there were several points that were worth looking at in Extras. There are actually four of them, and they get longer as they go (sorry).

Extra 1: one of the reasons that we might be seeing this picture is the attitude towards taxation in general in different countries. Scandinavians believe in large-scale government support (and are more or less happy to pay the taxes to support it), while Americans believe in small government, low taxes and that people should be able to support themselves. This, I think, is an explanation of what we see above.

Extra 2: the model we have fitted is a "main effects" model that says, for example, that the effect of religion on attitudes is the same for each country. We can investigate a bit further by looking at interactions. This model, for example, fits all two-way interactions (as in analysis of variance):[6]

```
WVS.2 <- polr(poverty ~ (religion + degree + country + age + gender)^2, data=WVS)
```

To see if anything can come out of this model, `drop1` again:

```
drop1(WVS.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## poverty ~ (religion + degree + country + age + gender)^2
##                   Df   AIC     LRT  Pr(>Chi)
## <none>               10390
## religion:degree    1 10394  5.2297 0.0222048 *
## religion:country   3 10402 17.7263 0.0005009 ***
## religion:age       1 10390  2.0694 0.1502785
## religion:gender    1 10391  2.4743 0.1157236
## degree:country     3 10394  9.7835 0.0204993 *
## degree:age         1 10388  0.0716 0.7890412
## degree:gender      1 10390  1.7280 0.1886683
## country:age        3 10402 17.8635 0.0004693 ***
## country:gender     3 10387  2.3653 0.5001320
## age:gender         1 10389  0.2412 0.6233488
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Some of these are significant, and some are clearly not. For example, the effect of age is the same whether or not a respondent has a degree. But some of them seem clearly significant: for example, the effect of age is different depending on which country a respondent lives in.

We could go through these and remove the least significant one in turn, or we could let `step` do it for us, which I will do here:

```
WVS.3 <- step(WVS.2, direction = "backward", test = "Chisq")
```

```
## Start:  AIC=10390.44
## poverty ~ (religion + degree + country + age + gender)^2
##
##                     Df   AIC     LRT  Pr(>Chi)
## - country:gender     3 10387  2.3653 0.5001320
## - degree:age         1 10388  0.0716 0.7890412
## - age:gender         1 10389  0.2412 0.6233488
## - degree:gender      1 10390  1.7280 0.1886683
## <none>                 10390
## - religion:age       1 10390  2.0694 0.1502785
## - religion:gender    1 10391  2.4743 0.1157236
## - religion:degree    1 10394  5.2297 0.0222048 *
## - degree:country     3 10394  9.7835 0.0204993 *
## - religion:country   3 10402 17.7263 0.0005009 ***
## - country:age        3 10402 17.8635 0.0004693 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=10386.81
## poverty ~ religion + degree + country + age + gender + religion:degree +
##     religion:country + religion:age + religion:gender + degree:country +
##     degree:age + degree:gender + country:age + age:gender
##
##                     Df   AIC     LRT  Pr(>Chi)
## - degree:age         1 10385  0.0640 0.8003455
## - age:gender         1 10385  0.3020 0.5826518
## <none>                 10387
## - religion:age       1 10387  2.0246 0.1547700
## - religion:gender    1 10387  2.4665 0.1162934
## - degree:gender      1 10388  2.7458 0.0975088 .
## - religion:degree    1 10390  5.2141 0.0224050 *
## - degree:country     3 10391 10.1720 0.0171589 *
## - religion:country   3 10398 17.4959 0.0005587 ***
## - country:age        3 10399 17.9191 0.0004571 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=10384.87
## poverty ~ religion + degree + country + age + gender + religion:degree +
##     religion:country + religion:age + religion:gender + degree:country +
##     degree:gender + country:age + age:gender
##
##                     Df   AIC     LRT  Pr(>Chi)
## - age:gender         1 10383  0.3014 0.5829846
## <none>                 10385
## - religion:age       1 10385  2.0261 0.1546169
## - religion:gender    1 10385  2.4854 0.1149103
## - degree:gender      1 10386  2.7042 0.1000863
## - religion:degree    1 10388  5.1881 0.0227422 *
## - degree:country     3 10389 10.1176 0.0175925 *
## - religion:country   3 10396 17.4896 0.0005604 ***
## - country:age        3 10398 18.9545 0.0002794 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=10383.17
## poverty ~ religion + degree + country + age + gender + religion:degree +
##     religion:country + religion:age + religion:gender + degree:country +
##     degree:gender + country:age
##
##                     Df   AIC     LRT  Pr(>Chi)
## - religion:age       1 10383  1.8852 0.1697441
## <none>                 10383
## - religion:gender    1 10384  2.3374 0.1262998
## - degree:gender      1 10384  2.5520 0.1101547
## - religion:degree    1 10386  5.2245 0.0222702 *
```

```
## - degree:country    3 10387 10.1332 0.0174671 *
## - religion:country  3 10395 17.4915 0.0005599 ***
## - country:age       3 10396 18.9616 0.0002784 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=10383.06
## poverty ~ religion + degree + country + age + gender + religion:degree +
##     religion:country + religion:gender + degree:country + degree:gender +
##     country:age
##
##                     Df   AIC     LRT  Pr(>Chi)
## <none>                  10383
## - religion:gender    1 10383  2.1272 0.1447085
## - degree:gender      1 10384  2.6216 0.1054182
## - religion:degree    1 10386  5.0777 0.0242359 *
## - degree:country     3 10387 10.2851 0.0162920 *
## - country:age        3 10395 17.7127 0.0005041 ***
## - religion:country   3 10396 19.1130 0.0002591 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
drop1(WVS.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## poverty ~ religion + degree + country + age + gender + religion:degree +
##     religion:country + religion:gender + degree:country + degree:gender +
##     country:age
##                    Df   AIC     LRT  Pr(>Chi)
## <none>                 10383
## religion:degree    1 10386  5.0777 0.0242359 *
## religion:country   3 10396 19.1130 0.0002591 ***
## religion:gender    1 10383  2.1272 0.1447085
## degree:country     3 10387 10.2851 0.0162920 *
## degree:gender      1 10384  2.6216 0.1054182
## country:age        3 10395 17.7127 0.0005041 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

You might have noticed, if you are following it through yourself, that these take an actually appreciable amount of time. Some of the remaining terms don't look very close to significant, but they stay because their AIC is (slightly) higher than the AIC for "none".

The fact that some interactions are definitely significant and should stay indicates that this is a better model than the one I called `WVS.1`. So, in fact, the inferences we get from this model are better than the ones that you got earlier.[7]

In this model, the effect of religion is evidently different for the different countries. Let's investigate how, by doing some predictions:

```
countries
```

```
## [1] "Australia" "Norway"    "Sweden"    "USA"
```

```
religions <- c("yes", "no")
new <- crossing(country = countries, religion = religions,
                age = 43, gender = "female", degree = "no")
p <- predict(WVS.3, new, type = "probs")
cbind(new, p)
```

```
##     country religion age gender degree Too Little About Right   Too Much
## 1 Australia       no  43 female     no  0.5407429   0.3383755 0.12088157
## 2 Australia      yes  43 female     no  0.5144638   0.3529917 0.13254451
## 3    Norway       no  43 female     no  0.5299664   0.3444718 0.12556174
## 4    Norway      yes  43 female     no  0.5817147   0.3140093 0.10427601
## 5    Sweden       no  43 female     no  0.4757512   0.3728540 0.15139479
## 6    Sweden      yes  43 female     no  0.6795665   0.2495077 0.07092576
## 7       USA       no  43 female     no  0.4976649   0.3618702 0.14046489
## 8       USA      yes  43 female     no  0.3506333   0.4186942 0.23067248
```

In Australia, there is not much effect of religion (the two sets of three probabilities are almost the same). In Norway and Sweden (especially Sweden), religious respondents are more or a lot more likely to say that their government should do more, but in the US, this is exactly reversed! Religious people there are more likely to say that their government is doing *too much* for poor people. I suspect that a lot of religious people in the US are politically conservative and object to paying higher taxes, but they may think that *their church* has the duty to help the poor. Americans are, as you see, different from the rest of the world in this regard. (We don't have political viewpoint in this dataset, but I think it would have been interesting to see it.)

The other strong interaction is between country and age, meaning that the effect of age is different in the different countries. Let's use the quartiles of age this time:[8]

```
countries
```

```
## [1] "Australia" "Norway"    "Sweden"    "USA"
```

```
ages <- c(31, 58)
new <- crossing(country = countries, religion = "no", age = ages,
                gender = "female", degree = "no")
p <- predict(WVS.3, new, type = "probs")
cbind(new, p)
```

```
##     country religion age gender degree Too Little About Right  Too Much
## 1 Australia       no  31 female     no  0.5866211   0.3109744 0.1024045
## 2 Australia       no  58 female     no  0.4825029   0.3695454 0.1479517
## 3    Norway       no  31 female     no  0.5295703   0.3446933 0.1257365
## 4    Norway       no  58 female     no  0.5304616   0.3441947 0.1253436
## 5    Sweden       no  31 female     no  0.4943066   0.3635987 0.1420947
## 6    Sweden       no  58 female     no  0.4526567   0.3836269 0.1637164
## 7       USA       no  31 female     no  0.5460913   0.3352992 0.1186094
## 8       USA       no  58 female     no  0.4372996   0.3902909 0.1724095
```

In Norway and Sweden, there is not much effect of age, but this time older people in both Australia and the US are less favourably disposed to how much help their government gives to the poor. The difference in age effects between countries is what drives the significant interaction.

Extra 3 (yes, I know, but I promised to come back to this): let's go back to the `summary` output from the first model we fitted, the one without interactions:

```
summary(WVS.1)
```

```
##
## Re-fitting to get Hessian
```

```
## Call:
## polr(formula = poverty ~ religion + degree + country + age +
##     gender, data = WVS)
##
## Coefficients:
##                   Value Std. Error t value
## religionyes     0.17973   0.077346   2.324
## degreeyes       0.14092   0.066193   2.129
## countryNorway -0.32235   0.073766  -4.370
## countrySweden -0.60330   0.079494  -7.589
## countryUSA      0.61777   0.070665   8.742
## age             0.01114   0.001561   7.139
## gendermale      0.17637   0.052972   3.329
##
## Intercepts:
##                           Value   Std. Error t value
## Too Little|About Right  0.7298  0.1041     7.0128
## About Right|Too Much    2.5325  0.1103    22.9496
##
## Residual Deviance: 10402.59
## AIC: 10420.59
```

Most of the interesting stuff is in the Coefficients table. The general idea is that a higher (more positive) coefficient means "towards the high end of the scale", in this case an opinion that the government is doing too much, and a lower (more negative) coefficient means "towards the low end", here that the government is doing too little.

The other thing to worry about is that most of our explanatory variables are categorical, so we have in those cases a baseline and comparisons with that.

Age is quantitative, and its coefficient is positive, so, overall, the effect of age is that an older person is more likely to say that the government is doing too much. As we saw at the end of Extra 2, this actually varies by country, and only in Australia and the US do you see an appreciable age effect, but both of those, and thus the overall impression, is pointing that way.

`religionyes` has a positive coefficient, meaning that overall, a religious people tend to think that the government is doing *too much*, compared to non-religious people. Given what we saw in Extra 2, it's really only in the US that you see that kind of effect (on the face of it, a surprising one). So I don't know.

`degreeyes` has a positive coefficient, meaning that a person with a degree is more likely than one without to think that the government is doing too much. (I don't know whether you find this surprising or not.)

The differences between the countries are consistent with what we found before.

Last, males are more likely that females to think that the government is doing too much.

On this goes to Extra 4 (which originally I asked you to do as well, but that made things too long):

Let's investigate the effect of age now. Let's do so for ages 31, 43, and 58 (the median and quartiles), for people who are religious, do not have a degree, are female and are from Australia. Steps: (i) We create a data frame containing these values, (ii) we obtain predicted probabilities of attitudes towards poverty of people of those ages, (iii) we describe the effect of age.

(i) Make a data frame (that I would call `new`, but you can call it what you like) that has those three ages and the other values repeated three times. Make sure you use *exactly* the same names for the columns in your new data frame as they had in the original. The easiest way is `tribble`:

```
new <- tribble(
  ~age, ~religion, ~degree, ~gender, ~country,
  31, "yes", "no", "female", "Australia",
  43, "yes", "no", "female", "Australia",
  58, "yes", "no", "female", "Australia",
)
new
```

```
## # A tibble: 3 x 5
##     age religion degree gender country
##   <dbl> <chr>    <chr>  <chr>  <chr>
## 1    31 yes      no     female Australia
## 2    43 yes      no     female Australia
## 3    58 yes      no     female Australia
```

Don't forget the quotes on the text things. I typed the first line of the dataframe (the one beginning with 31), copied and pasted it, and changed the ages, checking that this did indeed work.

There are at least two other ways to do this. One is to use `tibble` instead of `tribble`, which means thinking of the columns as rows:

```
new <- tibble(age = c(31, 43, 58), religion = "yes", degree = "no",
              gender = "female", country = "Australia")
new
```

```
## # A tibble: 3 x 5
##     age religion degree gender country
##   <dbl> <chr>    <chr>  <chr>  <chr>
## 1    31 yes      no     female Australia
## 2    43 yes      no     female Australia
## 3    58 yes      no     female Australia
```

You don't actually need to repeat the single ones, since `tibble` will repeat them for you. You can also split the line where you like as long as it is logically incomplete where you split it. After a comma is good, since a line cannot end with a comma.

The other way to do this is to think of it as "combinations" of ages and the other things, and use `crossing`:

```
ages <- c(31, 43, 58)
new <- crossing(age = ages, religion = "yes", degree = "no",
                gender = "female", country = "Australia")
new
```

```
## # A tibble: 3 x 5
##     age religion degree gender country
##   <dbl> <chr>    <chr>  <chr>  <chr>
## 1    31 yes      no     female Australia
## 2    43 yes      no     female Australia
## 3    58 yes      no     female Australia
```

Or put the three ages directly in the `crossing`.

In any of these, it does not matter what *order* the columns are in. All that matters is that you have all of them somewhere. If you don't, you'll get an error when you come to do the predictions. If that happens to you, I imagine the error message will clue you in to what happened.

Having gotten one of these, the next thing is (ii) to predict for them, and then display next to what they're predictions for:

```
p <- predict(WVS.1, new, type = "probs")
cbind(new, p)
```

```
##   age religion degree gender   country Too Little About Right  Too Much
## 1  31      yes     no female Australia  0.5509880   0.3305816 0.1184304
## 2  43      yes     no female Australia  0.5177356   0.3491487 0.1331156
## 3  58      yes     no female Australia  0.4759831   0.3704082 0.1536086
```

If your predictions run off the end of the line, use `select` to remove some columns.

The last thing, (iii), is to describe the effect of age. It seems not to be a very big effect (even though it is significant), but older people are less likely to say that the government gives too little to poor people, and more likely to say that the amount given is about right or too much. For the best answer, summarize this to say that there is a trend for older people to say that the government should be less generous, combining what you see from the first and third columns of the predictions.

Don't hand in the question below, but I think it's an interesting one to work through on your own time. (Including it in the assignment would have made it even longer than it already is.)

3. A "quantal assay" is a study of how the effectiveness of a drug depends on the dose of the drug that is given. (This is part of Phase 3 of a clinical trial; once a drug has been found to be safe (phase 1) and effective (phase 2), attention can be devoted to how large a dose should be given, in order to have a reasonably high chance that the drug has the desired effect.)

   I wrote this question last summer, but reading it now, I realize that the same considerations apply to vaccines: is it safe, does it work, how much do you need to give for it to work well.

   The data are in http://ritsokiguess.site/STAD29/quantal.csv, in three columns:

   - the log-dose of the drug administered (taking the logarithm of the dose usually makes for a more linear relationship with the log-odds)
   - the number of subjects who responded positively to the drug (the more, the better)
   - the total number of subjects who were given that dose.

(a) Read in and display the data. (You will probably see it all this time.)

Solution:

```
my_url <- "http://ritsokiguess.site/STAD29/quantal.csv"
assay <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   log_dose = col_double(),
##   responded = col_double(),
##   exposed = col_double()
## )
```

```
assay
```

```
## # A tibble: 9 x 3
##   log_dose responded exposed
##      <dbl>     <dbl>   <dbl>
## 1     2.68        10      31
## 2     2.76        17      30
## 3     2.82        12      31
## 4     2.9          7      27
## 5     3.02        23      26
## 6     3.04        22      30
## 7     3.13        29      31
## 8     3.2         29      30
## 9     3.21        23      30
```

There are only nine rows, so you should see all the data.

(b) Describe the data briefly. In particular, discuss how many different doses were tested, approximately how many subjects there are altogether, and thus what would be the easiest way to construct a response variable for a logistic regression.

Solution:

Nine different doses were tested (the nine rows of the dataframe). There are about 30 subjects total in each row (the number in the `exposed` column), so there were about $9 \times 30 = 270$ subjects altogether.[9] This is accurate enough, since the point is to realize that there is a lot more than one subject per row, but if you want to be more precise than I was:

```
assay %>% summarize(total=sum(exposed))
```

```
## # A tibble: 1 x 1
##   total
##   <dbl>
## 1   266
```

266 subjects.

With more than one subject per row and doses repeated lots of times, this is like the second rats example in the notes: it will need a two-column response, with successes (number of subjects who responded at each dose) in one column and failures (number who did not respond) in the

other. This is as far as you need to go here, since you'll be actually doing this in the next part.

(c) Create and display a response variable as you described it in the previous part.

Solution:

You will need to calculate a column of "failures" first, since all you have so far is successes and totals. Then grab the columns you want and save them in an R `matrix`.

The way I do this in the notes is to use the idea that cbind will glue columns or other things together and make a matrix, the exact thing that we want. We are going to be taking columns from a dataframe, so we will need to wrap the whole thing in `with`:

```
assay %>% mutate(no_response = exposed - responded) -> assayx
y <- with(assayx, cbind(responded, no_response))
y
```

```
##       responded no_response
##  [1,]        10          21
##  [2,]        17          13
##  [3,]        12          19
##  [4,]         7          20
##  [5,]        23           3
##  [6,]        22           8
##  [7,]        29           2
##  [8,]        29           1
##  [9,]        23           7
```

This needs two steps: make the new column and save it, then create the response using that new column.

A more `tidyverse`-looking way to do it is this, in one pipeline:

```
assay %>% mutate(no_response = exposed - responded) %>%
  select(responded, no_response) %>% as.matrix() -> y
y
```

```
##       responded no_response
##  [1,]        10          21
##  [2,]        17          13
##  [3,]        12          19
##  [4,]         7          20
##  [5,]        23           3
##  [6,]        22           8
##  [7,]        29           2
##  [8,]        29           1
##  [9,]        23           7
```

The last `as.matrix` is needed since `glm` requires a two-column matrix, not a dataframe. (The idea is "it looks like the right thing (a matrix) but it isn't one, so make it into one", which is what the `as.` functions do.)

The numbers in square brackets down the left side are the clue that this is a `matrix`: this is how matrices display.

(d) Fit a logistic regression to predict the probability of a patient responding to the drug from the log-dose, and display the results.

Solution:

If you have `responded` as the first column of your response matrix, you are good to go. If you chose to put your no-response column first in the last part, here is a chance to reconsider that decision, since the event whose probability is modelled is the thing in the *first* column, not the first alphabetically or anything like that:

```
assay.1 <- glm(y~log_dose, data = assay, family = binomial)
summary(assay.1)
```

```
##
## Call:
## glm(formula = y ~ log_dose, family = binomial, data = assay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4286  -0.9736   0.3699   1.6777   1.9302
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -15.8339     2.4371  -6.497 8.20e-11 ***
## log_dose      5.5778     0.8319   6.705 2.02e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.631  on 8  degrees of freedom
## Residual deviance: 29.346  on 7  degrees of freedom
## AIC: 62.886
##
## Number of Fisher Scoring iterations: 4
```

Once you have the response variable in place, there is nothing new here. (The thing I called `y` is a thing out on its own in your workspace, but the log-dose is in the dataframe. The logic behind the `data=` is that if it exists in your workspace, as `y` does, that's what it will use, but if it doesn't, as `log_dose` doesn't, it will go looking in the dataframe that you give to `data`.)

(e) Is there a significant effect of log-dose on whether or not a subject responds to the drug? What kind of effect is it? Explain briefly.

Solution:

The P-value for log-dose of 2×10^{-11} is extremely small, so there certainly is an effect of dose on whether or not a subject responds. The Estimate for log-dose is 5.58, positive, so that as log-dose (or dose) increases, the probability of a response increases: that is to say, a bigger dose is better.

(f) The researchers in this study want to estimate the log-dose at which 75% of subjects would respond (that is to say, of all subjects, of which the ones in this study are a sample). By doing some

predictions, or otherwise, what would you estimate that log-dose to be?

Solution:

This is an "inverse prediction": we know the probability of success, but we don't know the log-dose that goes with it. So the easiest way to think about this is to predict the probability of response for some log-doses, and see whether any of the results come out close to 0.75. I don't have any good guesses about what log-doses to use, so let's try the ones in the data, which means that we don't supply a `new` to `predict`:

```
p <- predict(assay.1, type = "response")
cbind(assay, p)
```

```
##   log_dose responded exposed         p
## 1     2.68        10      31 0.2920773
## 2     2.76        17      30 0.3919576
## 3     2.82        12      31 0.4739183
## 4     2.90         7      27 0.5846286
## 5     3.02        23      26 0.7332436
## 6     3.04        22      30 0.7544889
## 7     3.13        29      31 0.8354426
## 8     3.20        29      30 0.8823785
## 9     3.21        23      30 0.8880452
```

A log-dose of 3.04 comes out pretty close. I think 3.03 would be too low. I would call 3.04 close enough.

If you want to make a better guess, either to start with or to go on from the one above, you can look at the above, if you have it, or the original data, in which you eyeball or calculate the observed proportions:

```
assay %>% mutate(proportion = responded/exposed)
```

```
## # A tibble: 9 x 4
##   log_dose responded exposed proportion
##      <dbl>     <dbl>   <dbl>      <dbl>
## 1     2.68        10      31      0.323
## 2     2.76        17      30      0.567
## 3     2.82        12      31      0.387
## 4     2.9          7      27      0.259
## 5     3.02        23      26      0.885
## 6     3.04        22      30      0.733
## 7     3.13        29      31      0.935
## 8     3.2         29      30      0.967
## 9     3.21        23      30      0.767
```

There is some unnevenness here, but 2.9 is too low and 3.1 is too high, so let's go between those values:

```
new <- tibble(log_dose=seq(2.9, 3.1, 0.01))
p <- predict(assay.1, new, type = "response")
cbind(new, p)
```

```
##    log_dose         p
## 1      2.90 0.5846286
## 2      2.91 0.5981066
```

```
## 3      2.92 0.6114379
## 4      2.93 0.6246045
## 5      2.94 0.6375894
## 6      2.95 0.6503766
## 7      2.96 0.6629510
## 8      2.97 0.6752991
## 9      2.98 0.6874080
## 10     2.99 0.6992665
## 11     3.00 0.7108644
## 12     3.01 0.7221926
## 13     3.02 0.7332436
## 14     3.03 0.7440108
## 15     3.04 0.7544889
## 16     3.05 0.7646738
## 17     3.06 0.7745624
## 18     3.07 0.7841528
## 19     3.08 0.7934440
## 20     3.09 0.8024360
## 21     3.10 0.8111297
```

Scroll down until you find 0.75 in the `p` column; for me, 3.03 is too low and 3.04 is too high, by about the same amount, so 3.035 would be my guess.

If you want to do an actual piece of algebra, you can do it that way too without any predictions, but there is more thinking involved.

The first step is to recognize that we are working with log-odds in the logistic regression, so the first thing we need is the log-odds of 75%:

```
log(0.75/(1-0.75))
```

```
## [1] 1.098612
```

Go back to the `summary`:

```
summary(assay.1)
```

```
##
## Call:
## glm(formula = y ~ log_dose, family = binomial, data = assay)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4286  -0.9736   0.3699   1.6777   1.9302
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -15.8339     2.4371  -6.497 8.20e-11 ***
## log_dose      5.5778     0.8319   6.705 2.02e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.631  on 8  degrees of freedom
```

```
## Residual deviance: 29.346  on 7  degrees of freedom
## AIC: 62.886
##
## Number of Fisher Scoring iterations: 4
```

And then we have to find where $-15.8339 + 5.5778d$ passes through 1.098612, d being my shorthand for the log-dose. This is where the algebra comes in, since we need to solve for d:

$$\begin{aligned} -15.8339 + 5.5778d &= 1.098612 \\ 5.5778d &= 1.098612 + 15.8339 \\ d &= \frac{1.098612 + 15.8339}{5.5778} \end{aligned}$$

and then use R to work this out:

```
d <- (1.098612 + 15.8339) / 5.5778
d
```

```
## [1] 3.035697
```

There's your estimate of the log-dose, 3.036 rounded off.

Extra: it is also possible to get a confidence interval for this dose. I'm not going to get into the details of how to do this, but will defer to the package MASS:

```
library(MASS)
the_dose <- dose.p(assay.1, p=0.75)
the_dose
```

```
##                Dose          SE
## p = 0.75: 3.035682 0.03054312
```

This differs in the fifth decimal from the one I got, because I was using the rounded-off intercept and slope (they really have more decimals than shown). This here is the one to trust.

To get the interval, go up and down twice the SE, which is a little fiddly. First, get hold of the standard error, as a number:[10]

```
se <- attr(the_dose, "SE")
se
```

```
##                   [,1]
## p = 0.75: 0.03054312
```

It's a 1×1 matrix, so make it into a number:

```
se <- as.vector(se)
se
```

```
## [1] 0.03054312
```

and then go up and down twice this from the estimated (log-)dose:

```
the_dose + c(-2, 2) * se
```

```
## [1] 2.974596 3.096768
```

A nice small interval, since we have a good amount of data.

Notes

1. My suspicion is that all three explanatory variables have a strongish positive correlation, so that if you take one out, the P-values change substantially.

2. Another possibility is the "two-column response", a count of successes and failures at each combination of explanatory variable values.

3. Its name varies for different models and it's easy to get confused.

4. Up until just now, I had no idea what this one did. So I tried it, to find out. It's actually rather useful, because it shows you whether each explanatory variable value is making a positive or negative contribution to the probability of "success" for each individual, without having to do another calculation to see whether the values are unusually high or low. In this one, I told you what kind of work experience values we had, and you probably have a good intuition about whether a GPA is high or low. In other cases, though, you might not know whether the x values you are predicting for are high or low, and using *terms* tells you without you having to find out.

5. A negative one, literally. Unless one of these students has a very positive number coming out of work experience (they don't), their log-odds of being admitted will be very negative, and that goes with a probability of being admitted that is a lot less than 0.5, in fact down near zero.

6. This, by the way, is the reason that you have to be careful with raising things to powers in regression, because the up arrow doesn't mean what you think it means.

7. In fact, we could fit the three-way interactions and see if any of *those* should stay, and if so, we have an even better model. But three-way interactions are hard to interpret. All in all, since we haven't seen interactions in this course yet, I didn't want to make you grapple with them at this point, hence the main-effects-only model that I asked you to fit.

8. We have to make the age vary to see what effect it has.

9. I did this in my head, because I couldn't be bothered to work it out more precisely.

10. MASS goes back a few years. This is a rather old-fashioned way to do it: the output from dose.p contains the estimated dose, but also some other things hidden in it called *attributes*, one of which is the SE. The modern way to do this stores the result in eg. a tibble, and looking at that shows you all the numbers in it. If the result of the function is a bunch of different things, as in a regression where you have intercept, slope(s) and stuff like residuals, you store them in a list.