

Inferential II: Regression and prediction

EDH7916

Benjamin Skinner

In this lesson, we'll move beyond t.tests to regression. As before, this lesson is too short to stand in for a full course on regression analyses. However, it should give you the basics of how to fit regression and store key parameters. We'll also cover the basics of prediction and the estimate of marginal “effects” (nothing causal here!).

Data

In this lesson, we'll use the same data from the NCES Education Longitudinal Study of 2002¹. So you don't have to go back to the prior lesson, here again is a codebook with descriptions of the variables included in our lesson today:

variable	description
stu_id	student id
sch_id	school id
strat_id	stratum
psu	primary sampling unit
bystuwt	student weight
bysex	sex-composite
byrace	student's race/ethnicity-composite
bydob_p	student's year and month of birth
bypared	parents' highest level of education
bymothed	mother's highest level of education-composite
byfathed	father's highest level of education-composite
byincome	total family income from all sources 2001-composite
byses1	socio-economic status composite, v.1
byses2	socio-economic status composite, v.2
bystexp	how far in school student thinks will get-composite
bynels2m	els-nels 1992 scale equated sophomore math score
bynels2r	els-nels 1992 scale equated sophomore reading score
f1qwt	questionnaire weight for f1
f1pnlwt	panel weight, by and f1 (2002 and 2004)
f1psepln	f1 post-secondary plans right after high school
f2ps1sec	Sector of first postsecondary institution
female	== 1 if female
moth_ba	== 1 if mother has BA/BS
fath_ba	== 1 if father has BA/BS
par_ba	== 1 if either parent has BA/BS
plan_col_grad	== 1 if student plans to earn college degree
lowinc	== 1 if income < \$25k

¹<https://nces.ed.gov/surveys/els2002/>

We'll load the same libraries and data!

```
## -----  
## libraries  
## -----  
  
library(tidyverse)  
  
## — Attaching core tidyverse packages ————— tidyverse 2.0.0 —  
## ✓ dplyr      1.1.1      ✓ readr      2.1.4  
## ✓ forcats    1.0.0      ✓ stringr    1.5.0  
## ✓ ggplot2     3.4.2      ✓ tibble     3.2.1  
## ✓ lubridate  1.9.2      ✓ tidyr      1.3.0  
## ✓ purrr      1.0.1  
## — Conflicts ————— tidyverse_conflicts() —  
## * dplyr::filter() masks stats::filter()  
## * dplyr::lag()     masks stats::lag()  
## □ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors  
  
library(haven)  
library(survey)  
  
## Error in library(survey): there is no package called 'survey'  
  
## -----  
## directory paths  
## -----  
  
## assume we're running this script from the ./scripts subdirectory  
dat_dir <- file.path(".", "data")  
  
## -----  
## input data  
## -----  
  
## assume we're running this script from the ./scripts subdirectory  
df <- read_dta(file.path(dat_dir, "els_plans.dta"))
```

Linear model

Linear models are the go-to method of making inferences for many data analysts. In R, the `lm()` command is used to compute an ordinary least squares (OLS)² regression. Unlike above, where we just let the `t.test()` output print to the console, we can and will store the output in an object.

First, let's compute the same t-test as in the prior inferential lesson, but in a regression framework. This time, we'll assume equal variances between the distributions in the t-test above (`var.equal = TRUE`), so we should get the same results as we did before.

```
## t-test of difference in math scores across parental education (BA/BA or not)  
t.test(bynels2m ~ par_ba, data = df, var.equal = TRUE)  
  
##  
## Two Sample t-test  
##  
## data: bynels2m by par_ba  
## t = -38.54, df = 15234, p-value < 2.2e-16
```

²https://en.wikipedia.org/wiki/Ordinary_least_squares

```
## alternative hypothesis: true difference in means between group 0 and group 1 is not equal to 0
## 95 percent confidence interval:
## -8.669138 -7.830008
## sample estimates:
## mean in group 0 mean in group 1
##      41.97543      50.22501
```

```
## compute same test as above, but in a linear model
```

```
fit <- lm(bynels2m ~ par_ba, data = df)
fit
```

```
##
## Call:
## lm(formula = bynels2m ~ par_ba, data = df)
##
## Coefficients:
## (Intercept)      par_ba
##      41.98      8.25
```

The output is a little thin: just the coefficients. To see the full range of information you want from regression output, use the `summary()` function wrapped around the fit object.

```
## use summary to see more information about regression
```

```
summary(fit)
```

```
##
## Call:
## lm(formula = bynels2m ~ par_ba, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -35.515  -9.685   0.595   9.885  37.015
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  41.9754      0.1375  305.35  <2e-16 ***
## par_ba       8.2496      0.2141   38.54  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.01 on 15234 degrees of freedom
## (924 observations deleted due to missingness)
## Multiple R-squared:  0.08884, Adjusted R-squared:  0.08878
## F-statistic: 1485 on 1 and 15234 DF, p-value: < 2.2e-16
```

We'll more fully discuss this output in the next section. For now, let's compare the key findings to those returned from the t test.

Because our right-hand side (RHS) variable is an indicator variable that `== 0` for parents without a BA/BS or higher and `== 1` if either parent has a BA/BS or higher, then the intercept reflects the math test score for students when `par_ba == 0`. This matches the mean in group 0 value from the t test above.

In a regression framework, the coefficient on `par_ba` is the marginal difference when `par_ba` increases by one unit. Since `pared` is the only parameter on the RHS (besides the intercept) and only takes on values 0 and 1, we can add its coefficient to the intercept to get the math test score mean for students with parents with a BA/BS or higher.

```
## add intercept and par_ba coefficient
fit$coefficients[["(Intercept)"]] + fit$coefficients[["par_ba"]]
```

```
## [1] 50.22501
```

Looks like this value matches what we saw before (within rounding). Going the other way, the coefficient on `par_ba`, 8.2495729, is the same as the difference between the groups in the t test. Finally, notice that the absolute value of the test statistic for the t test and the `par_ba` coefficient are the same value: 38.54. Success!

Multiple regression

To fit a multiple regression, use the same formula framework that we've use before with the addition of all the terms you want on right-hand side of the equation separated by plus (+) signs.

NB From here on out, we'll spend less time interpreting the regression results so that we can focus on the tools of running regressions. That said, let me know if you have questions of interpretation.

```
## linear model with more than one covariate on the RHS
fit <- lm(bynels2m ~ byses1 + female + moth_ba + fath_ba + lowinc,
         data = df)
summary(fit)
```

```
##
## Call:
## lm(formula = bynels2m ~ byses1 + female + moth_ba + fath_ba +
##     lowinc, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -39.456  -8.775   0.432   9.110  40.921
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  45.7155     0.1811  252.420 < 2e-16 ***
## byses1       6.8058     0.2387   28.511 < 2e-16 ***
## female      -1.1483     0.1985   -5.784 7.42e-09 ***
## moth_ba       0.4961     0.2892    1.715 0.08631 .
## fath_ba       0.8242     0.2903    2.840 0.00452 **
## lowinc       -2.1425     0.2947   -7.271 3.75e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.24 on 15230 degrees of freedom
## (924 observations deleted due to missingness)
## Multiple R-squared:  0.1929, Adjusted R-squared:  0.1926
## F-statistic: 728.1 on 5 and 15230 DF, p-value: < 2.2e-16
```

The full output tells you:

- the model that you fit, under `Call`:
- a table of coefficients with
 - the point estimates (`Estimate`)
 - the point estimate errors (`Std. Error`)
 - the test statistic for each point estimate (`t value` with this model)
 - the p value for each point estimate (`Pr(>|t|)`)
- significance stars (`.` and `*`) along with legend

- the R-squared values (Multiple R-squared and Adjusted R-squared)
- the model F-statistic (F-statistic)
- number of observations dropped if any

If observations were dropped due to missing values (`lm()` does this automatically by default), you can recover the number of observations actually used with the `nobs()` function.

```
## check number of observations
nobs(fit)
```

```
## [1] 15236
```

The `fit` object also holds a lot of other information that is sometimes useful.

```
## see what fit object holds
names(fit)
```

```
## [1] "coefficients" "residuals" "effects" "rank"
## [5] "fitted.values" "assign" "qr" "df.residual"
## [9] "na.action" "xlevels" "call" "terms"
## [13] "model"
```

In addition to the `coefficients`, which you pulled out of the first model, both `fitted.values` and `residuals` are stored in the object. You can access these “hidden” attributes by treating the `fit` object like a data frame and using the `$` notation.

```
## see first few fitted values and residuals
head(fit$fitted.values)
```

```
##      1      2      3      4      5      6
## 42.86583 48.51465 38.78234 36.98010 32.82855 38.43332
```

```
head(fit$residuals)
```

```
##      1      2      3      4      5      6
##  4.974173  6.785347 27.457659 -1.650095 -2.858552 -14.153323
```

Quick exercise Add the fitted values to the residuals and store in an object (`x`). Compare these values to the math scores in the data frame.

As a final note, the model matrix used fit the regression can be retrieved using `model.matrix()`. Since we have a lot of observations, we’ll just look at the first few rows.

```
## see the design matrix
head(model.matrix(fit))
```

```
## (Intercept) byses1 female moth_ba fath_ba lowinc
## 1          1 -0.25      1      0      0      0
## 2          1  0.58      1      0      0      0
## 3          1 -0.85      1      0      0      0
## 4          1 -0.80      1      0      0      1
## 5          1 -1.41      1      0      0      1
## 6          1 -1.07      0      0      0      0
```

What this shows is that the `fit` object actually stores a copy of the data used to run it. That’s really convenient if you want to save the object to disk (with the `save()` function) so you can review the regression results later. But keep in mind that if you share that file, you are sharing the part of the data used to estimate it. Because a lot of education data is restricted in some way — via memorandums of understanding

(MOUs) or IRB — be careful about sharing the saved output object. Typically you’ll only share the results in a table or figure, but just be aware.

Using categorical variables or factors

It’s not necessary to pre-construct dummy variables if you want to use a categorical variable in your model. Instead you can use the categorical variable wrapped in the `factor()` function. This tells R that the underlying variable shouldn’t be treated as a continuous value, but should be discrete groups. R will make the dummy variables on the fly when fitting the model. We’ll include the categorical variable `bystexp` in this model.

```
## check values of student expectations
df %>%
  count(bystexp)

## # A tibble: 9 × 2
##   bystexp          n
##   <dbl+lbl>      <int>
## 1 -1 [{don^t know}]    1450
## 2  1 [less than high school graduation]    128
## 3  2 [high school graduation or ged only]   983
## 4  3 [attend or complete 2-year college/school] 879
## 5  4 [attend college, 4-year degree incomplete] 561
## 6  5 [graduate from college]             5416
## 7  6 [obtain master^s degree or equivalent] 3153
## 8  7 [obtain phd, md, or other advanced degree] 2666
## 9 NA                                924
```

Even though student expectations of eventual degree attainment are roughly ordered, let’s use them in our model as discrete groups. That way we can leave in “Don’t know” and don’t have to worry about that “attend college, 4-year degree incomplete” is somehow *higher* than “attend or complete 2-year college/school”.

```
## add factors
fit <- lm(bynels2m ~ byses1 + female + moth_ba + fath_ba
          + lowinc + factor(bystexp),
          data = df)
summary(fit)

##
## Call:
## lm(formula = bynels2m ~ byses1 + female + moth_ba + fath_ba +
##     lowinc + factor(bystexp), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -41.680  -8.267   0.541   8.423  38.696
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    42.9534     0.3366  127.603 < 2e-16 ***
## byses1         5.1616     0.2297   22.468 < 2e-16 ***
## female        -2.3828     0.1909  -12.481 < 2e-16 ***
## moth_ba         0.4119     0.2742    1.502  0.1331
## fath_ba         0.6250     0.2754    2.270  0.0232 *
## lowinc        -2.2017     0.2794   -7.880 3.50e-15 ***
## factor(bystexp)1 -10.0569    1.0710   -9.390 < 2e-16 ***
```

```
## factor(bystexp)2 -5.4527    0.4813 -11.329 < 2e-16 ***
## factor(bystexp)3 -1.2000    0.4966 -2.416  0.0157 *
## factor(bystexp)4 -3.5317    0.5771 -6.119 9.62e-10 ***
## factor(bystexp)5  3.6345    0.3446 10.546 < 2e-16 ***
## factor(bystexp)6  7.6366    0.3736 20.442 < 2e-16 ***
## factor(bystexp)7  7.5114    0.3860 19.460 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.6 on 15223 degrees of freedom
## (924 observations deleted due to missingness)
## Multiple R-squared:  0.2754, Adjusted R-squared:  0.2748
## F-statistic: 482.1 on 12 and 15223 DF, p-value: < 2.2e-16
```

If you're using labeled data like we have been for the past couple of modules, you can use the `as_factor()` function from the `haven` library³ in place of the base `factor()` function. You'll still see the `as_factor(<var>)` prefix on each coefficient, but now you'll have labels instead of the underlying values, which should make parsing the output a little easier.

```
## same model, but use as_factor() instead of factor() to use labels
fit <- lm(bynels2m ~ byses1 + female + moth_ba + fath_ba
          + lowinc + as_factor(bystexp),
          data = df)
summary(fit)
```

```
##
## Call:
## lm(formula = bynels2m ~ byses1 + female + moth_ba + fath_ba +
##      lowinc + as_factor(bystexp), data = df)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
	-41.680	-8.267	0.541	8.423	38.696

```
##
## Coefficients:
```

	Estimate	Std. Error
(Intercept)	42.9534	0.3366
byses1	5.1616	0.2297
female	-2.3828	0.1909
moth_ba	0.4119	0.2742
fath_ba	0.6250	0.2754
lowinc	-2.2017	0.2794
as_factor(bystexp)less than high school graduation	-10.0569	1.0710
as_factor(bystexp)high school graduation or ged only	-5.4527	0.4813
as_factor(bystexp)attend or complete 2-year college/school	-1.2000	0.4966
as_factor(bystexp)attend college, 4-year degree incomplete	-3.5317	0.5771
as_factor(bystexp)graduate from college	3.6345	0.3446
as_factor(bystexp)obtain master's degree or equivalent	7.6366	0.3736
as_factor(bystexp)obtain phd, md, or other advanced degree	7.5114	0.3860

```
##
## t value Pr(>|t|)
## (Intercept) 127.603 < 2e-16 ***
## byses1 22.468 < 2e-16 ***
## female -12.481 < 2e-16 ***
## moth_ba 1.502 0.1331
```

³https://haven.tidyverse.org/reference/as_factor.html

```
## fath_ba                2.270    0.0232 *
## lowinc                -7.880 3.50e-15 ***
## as_factor(bystexp)less than high school graduation    -9.390 < 2e-16 ***
## as_factor(bystexp)high school graduation or ged only  -11.329 < 2e-16 ***
## as_factor(bystexp)attend or complete 2-year college/school -2.416    0.0157 *
## as_factor(bystexp)attend college, 4-year degree incomplete -6.119 9.62e-10 ***
## as_factor(bystexp)graduate from college              10.546 < 2e-16 ***
## as_factor(bystexp)obtain master^s degree or equivalent 20.442 < 2e-16 ***
## as_factor(bystexp)obtain phd, md, or other advanced degree 19.460 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.6 on 15223 degrees of freedom
## (924 observations deleted due to missingness)
## Multiple R-squared:  0.2754, Adjusted R-squared:  0.2748
## F-statistic: 482.1 on 12 and 15223 DF,  p-value: < 2.2e-16
```

If you look at the model matrix, you can see how R created the dummy variables from `bystexp`: adding new columns of only 0/1s that correspond to the `bystexp` value of each student.

```
## see what R did under the hood to convert categorical to dummies
head(model.matrix(fit))
```

```
## (Intercept) byses1 female moth_ba fath_ba lowinc
## 1          1 -0.25      1      0      0      0
## 2          1  0.58      1      0      0      0
## 3          1 -0.85      1      0      0      0
## 4          1 -0.80      1      0      0      1
## 5          1 -1.41      1      0      0      1
## 6          1 -1.07      0      0      0      0
## as_factor(bystexp)less than high school graduation
## 1          0
## 2          0
## 3          0
## 4          0
## 5          0
## 6          0
## as_factor(bystexp)high school graduation or ged only
## 1          0
## 2          0
## 3          0
## 4          0
## 5          0
## 6          0
## as_factor(bystexp)attend or complete 2-year college/school
## 1          1
## 2          0
## 3          0
## 4          0
## 5          0
## 6          0
## as_factor(bystexp)attend college, 4-year degree incomplete
## 1          0
## 2          0
## 3          0
```



```
## 4 0
## 5 0
## 6 1
## as_factor(bystexp)graduate from college
## 1 0
## 2 0
## 3 0
## 4 1
## 5 1
## 6 0
## as_factor(bystexp)obtain master^s degree or equivalent
## 1 0
## 2 0
## 3 0
## 4 0
## 5 0
## 6 0
## as_factor(bystexp)obtain phd, md, or other advanced degree
## 1 0
## 2 1
## 3 0
## 4 0
## 5 0
## 6 0
```

Quick exercise Add the categorical variable *byincome* to the model above. Next use `model.matrix()` to check the *RHS* matrix.

Interactions

Add interactions to a regression using an asterisks (*) between the terms you want to interact. This will add both main terms and the interaction(s) between the two to the model. Any interaction terms will be labeled using the base name or factor name of each term joined by a colon (:).

```
## add interactions
fit <- lm(bynels2m ~ byses1 + factor(bypared)*lowinc, data = df)
summary(fit)

##
## Call:
## lm(formula = bynels2m ~ byses1 + factor(bypared) * lowinc, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -38.998  -8.852   0.326   9.063  39.257
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    44.0084     0.6345  69.355 < 2e-16 ***
## byses1          7.6082     0.2772  27.448 < 2e-16 ***
## factor(bypared)2    1.5546     0.6544   2.376 0.017533 *
## factor(bypared)3    0.6534     0.7136   0.916 0.359863
```

```
## factor(bypared)4          1.8902      0.7198      2.626 0.008646 **
## factor(bypared)5          1.5059      0.7200      2.091 0.036501 *
## factor(bypared)6          1.4527      0.7386      1.967 0.049235 *
## factor(bypared)7          2.0044      0.8286      2.419 0.015569 *
## factor(bypared)8          0.8190      0.9239      0.887 0.375360
## lowinc                    2.0347      0.8112      2.508 0.012140 *
## factor(bypared)2:lowinc -2.9955      0.9298     -3.222 0.001278 **
## factor(bypared)3:lowinc -4.0551      1.0682     -3.796 0.000147 ***
## factor(bypared)4:lowinc -4.8143      1.1126     -4.327 1.52e-05 ***
## factor(bypared)5:lowinc -4.6890      1.0947     -4.283 1.85e-05 ***
## factor(bypared)6:lowinc -4.5252      1.0556     -4.287 1.82e-05 ***
## factor(bypared)7:lowinc -7.2222      1.3796     -5.235 1.67e-07 ***
## factor(bypared)8:lowinc -9.8773      1.6110     -6.131 8.94e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.23 on 15219 degrees of freedom
## (924 observations deleted due to missingness)
## Multiple R-squared:  0.1948, Adjusted R-squared:  0.194
## F-statistic: 230.2 on 16 and 15219 DF, p-value: < 2.2e-16
```

Polynomials

To add quadratic and other polynomial terms to the model, use the `I()` function, which lets you raise the term to the power you want in the regression using the caret (^) operator. In the model below, we add a quadratic version of the reading score to the right-hand side.

```
## add polynomials
fit <- lm(bynels2m ~ bynels2r + I(bynels2r^2), data = df)
summary(fit)

##
## Call:
## lm(formula = bynels2m ~ bynels2r + I(bynels2r^2), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -33.462  -5.947  -0.156   5.780  46.645
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.7765815   0.6241194   20.471  <2e-16 ***
## bynels2r      1.1197116   0.0447500   25.021  <2e-16 ***
## I(bynels2r^2) -0.0006246   0.0007539   -0.828    0.407
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.921 on 15881 degrees of freedom
## (276 observations deleted due to missingness)
## Multiple R-squared:  0.5658, Adjusted R-squared:  0.5657
## F-statistic: 1.035e+04 on 2 and 15881 DF, p-value: < 2.2e-16
```

Quick exercise Fit a linear model with both interactions and a polynomial term. Then look at the model matrix to see what R did under the hood.

Generalized linear model for binary outcomes

In some cases when you have binary outcomes — 0/1 — it may be appropriate to continue using regular OLS, fitting what is typically called a *linear probability model* or LPM⁴. In those cases, just use `lm()` as you have been.

But in other cases, you'll want to fit a generalized linear model⁵, in which case you'll need to switch to the `glm()` function. It is set up just like `lm()`, but it has an extra argument, `family`. Set the argument to `binomial()` when your dependent variable is binary. By default, the link function is a logit⁶ link.

```
## logit
fit <- glm(plan_col_grad ~ byncls2m + as_factor(bypared),
           data = df,
           family = binomial())
summary(fit)

##
## Call:
## glm(formula = plan_col_grad ~ byncls2m + as_factor(bypared),
##      family = binomial(), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.6467  -0.9581   0.5211   0.7695   1.5815
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.824413    0.090000  -20.271 < 2e-16 ***
## byncls2m         0.056427    0.001636   34.491 < 2e-16 ***
## as_factor(bypared)hsGED    0.042315    0.079973    0.529  0.5967
## as_factor(bypared)att2yr   0.204831    0.088837    2.306  0.0211 *
## as_factor(bypared)grad2ry  0.480828    0.092110    5.220 1.79e-07 ***
## as_factor(bypared)att4yr   0.499019    0.090558    5.511 3.58e-08 ***
## as_factor(bypared)grad4yr  0.754817    0.084271    8.957 < 2e-16 ***
## as_factor(bypared)ma       0.943558    0.101585    9.288 < 2e-16 ***
## as_factor(bypared)phprof   1.052006    0.121849    8.634 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 17545  on 15235  degrees of freedom
## Residual deviance: 15371  on 15227  degrees of freedom
## (924 observations deleted due to missingness)
## AIC: 15389
##
## Number of Fisher Scoring iterations: 4
```

⁴https://en.wikipedia.org/wiki/Linear_probability_model

⁵https://en.wikipedia.org/wiki/Generalized_linear_model

⁶<https://en.wikipedia.org/wiki/Logit>

If you want a probit⁷ model, just change the link to probit.

```
## probit
fit <- glm(plan_col_grad ~ byncls2m + as_factor(bypared),
           data = df,
           family = binomial(link = "probit"))
summary(fit)

##
## Call:
## glm(formula = plan_col_grad ~ byncls2m + as_factor(bypared),
##      family = binomial(link = "probit"), data = df)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.7665  -0.9796   0.5238   0.7812   1.5517
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.0522131   0.0539072  -19.519  < 2e-16 ***
## byncls2m         0.0326902   0.0009357   34.938  < 2e-16 ***
## as_factor(bypared)hsge    0.0325415   0.0488225    0.667   0.5051
## as_factor(bypared)att2yr   0.1316456   0.0539301    2.441   0.0146 *
## as_factor(bypared)grad2ry  0.2958810   0.0554114    5.340 9.31e-08 ***
## as_factor(bypared)att4yr   0.3065176   0.0544813    5.626 1.84e-08 ***
## as_factor(bypared)grad4yr  0.4553127   0.0505009    9.016  < 2e-16 ***
## as_factor(bypared)ma       0.5525198   0.0588352    9.391  < 2e-16 ***
## as_factor(bypared)phprof   0.6115358   0.0688820    8.878  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 17545  on 15235  degrees of freedom
## Residual deviance: 15379  on 15227  degrees of freedom
## (924 observations deleted due to missingness)
## AIC: 15397
##
## Number of Fisher Scoring iterations: 4
```

Note that the interpretation of parameters in logit and probit models differs from that in linear model with a normal / gaussian link function. For example, a one-unit change for a parameter in a logistic regression is associated with a change in *log odds* of the outcome,

$$\log\left(\frac{p}{1-p}\right)$$

, holding all other values at some fixed value (their means, for example). If this doesn't sound particularly interpretable, it's not! This is one reason people continue to use linear probability models (LPMs) rather than logits/probits (there are other reasons⁸ too). As always, make the choice that best fits your data, research questions, and intended audience.

⁷https://en.wikipedia.org/wiki/Probit_model

⁸<https://opa.hhs.gov/sites/default/files/2020-07/lpm-tabrief.pdf>

Quick exercise *Fit a logit or probit model to another binary outcome.*

Using survey weights in a regression

We spent time in the Inferential I lesson on setting up a data frame that accounted for survey weights. Review that if need a reminder of the intuition.

Important for this lesson is that you can (and should!) use survey weights in a regression framework. As a reminder, here's the information you need to set up the `svydesign()` that accounts for ELS's complex sampling design⁹:

- `ids` are the primary sampling units or `psus`
- `strata` are indicated by the `strat_ids`
- `weight` is the base-year student weight or `bystuwt`
- `data` is our data frame object, `df`
- `nest = TRUE` because the `psus` are nested in `strat_ids`

Before running our survey-weighted regression, we'll once again set up our data in a new object.

```
## subset data
svy_df <- df %>%
  select(psu,                # primary sampling unit
         strat_id,           # stratum ID
         bystuwt,            # weight we want to use
         byncls2m,           # variables we want...
         moth_ba,
         fath_ba,
         par_ba,
         byses1,
         lowinc,
         female) %>%
  ## go ahead and drop observations with missing values
  drop_na()

## set svy design data
svy_df <- svydesign(ids = ~psu,
                   strata = ~strat_id,
                   weight = ~bystuwt,
                   data = svy_df,
                   nest = TRUE)
```

```
## Error in svydesign(ids = ~psu, strata = ~strat_id, weight = ~bystuwt, : could not find function "svydesign"
```

Now that we've done that, here's how we run a weighted regression. Notice that it's `svyglm()`, even though we used `lm()` before (the default "link" function in `glm()` is `gaussian` or `normal`; if we had binary outcomes and wanted to use a logit link then we could include `family = binomial()` as we did before).

```
## fit the svyglm regression and show output
svyfit <- svyglm(byncls2m ~ byses1 + female + moth_ba + fath_ba + lowinc,
                 design = svy_df)
```

```
## Error in svyglm(byncls2m ~ byses1 + female + moth_ba + fath_ba + lowinc, : could not find function "svyglm"
```

⁹https://nces.ed.gov/training/datauser/ELS_04.html

```
summary(svyfit)
```

```
## Error in summary(svyfit): object 'svyfit' not found
```

As when compared t.test results when using unweighted and weighted data, our regression results are little different when using the weights. As a reminder, there's no single quick answer to using weights in a regression framework. It's up to you and your investigation of the code book to decide:

- the appropriate weights to choose (there are usually many options!)
- what weights mean in the context of missing data
- what weights mean in a complex research design

Predictions

Being able to generate predictions from new data can be a powerful tool. Above, we were able to return the predicted values from the fit object. We can also use the `predict()` function to return the standard error of the prediction in addition to the predicted values for new observations.

First, we'll get predicted values using the original data along with their standard errors.

```
## predict from first model
fit <- lm(byncls2m ~ byses1 + female + moth_ba + fath_ba + lowinc,
         data = df)

## old data
fit_pred <- predict(fit, se.fit = TRUE)

## show options
names(fit_pred)

## [1] "fit"          "se.fit"       "df"           "residual.scale"
head(fit_pred$fit)

##          1          2          3          4          5          6
## 42.86583 48.51465 38.78234 36.98010 32.82855 38.43332
head(fit_pred$se.fit)

## [1] 0.1755431 0.2587681 0.2314676 0.2396327 0.2737971 0.2721818
```

With the standard errors, we can get a better feel for how much faith we want to put in our predictions. If the standard errors are low, then maybe we feel more secure than if the errors are large. Alternatively, perhaps the errors are uniformly lower for some parts of our data than others. If so, that might suggest more investigation or a note on the limitations of our predictions for parts of the sample.

Two other types of predictions

We won't practice these since in application they work similarly to what we did above. However, I do want to note two other types of predictions that you might want to make. Both involve making predictions for data that you didn't use to fit your model. Predictions using new data or held-out data in a train/test framework are another way to evaluate your model. If you predict well to new/held-out data, that can be a good sign for the utility of your model.

Predictions with new data Ideally, we would have a new observations with which to make predictions. Then we could test our modeling choices by seeing how well they predicted the outcomes of the new observations.

With discrete outcomes (like binary 0/1 data), for example, we could use our model and right-hand side variables from new observations to predict whether the new observation should have a 0 or 1 outcome. Then we could compare those predictions to the actual observed outcomes by making a 2 by 2 confusion matrix¹⁰ that counted the numbers of true positives and negatives (correct predictions) and false positives and negatives (incorrect predictions).

With continuous outcomes, we could follow the same procedure as above, but rather than using a confusion matrix, instead assess our model performance by measuring the error between our predictions and the observed outcomes. Depending on our problem and model, we might care about minimizing the root mean square error, the mean absolute error, or some other metric of the error.

Predictions using training and testing data In the absence of new data, we instead could have separated our data into two data sets, a training set and test set¹¹. After fitting our model to the training data, we could have tested it by following either above procedure with the testing data (depending on the outcome type). Setting a rule for ourselves, we could evaluate how well we did, that is, how well our training data model classified test data outcomes, and perhaps decide to adjust our modeling assumptions. This is a fundamental way that many machine learning algorithm assess fit.

Margins

Using the `predict()` function alongside some other skills we have practiced, we can also make predictions on the margin a la Stata's `-margins-` suite of commands¹².

For example, after fitting our multiple regression, we might ask ourselves, what is the marginal association of coming from a family with low income on math scores, holding all other terms in our model constant? In other words, if student A and student B are similar along dimensions we can observe (let's say the average student in our sample) except for the fact that student A's family is considered lower income and student B's is not, what if any test score difference might we expect?

To answer this question, we first need to make a "new" data frame with a column each for the variables used in the model and rows that equal the number of predictive margins that we want to create. In our example, that means making a data frame with two rows and five columns.

With `lowinc`, the variable that we want to make marginal predictions for, we have two potential values: 0 and 1. This is the reason our "new" data frame has two rows. If `lowinc` took on four values, for example, then our "new" data frame would have four rows, one for each potential value. But since we have two, `lowinc` in our "new" data frame will equal 0 in one row and 1 in the other row.

All other columns in the "new" data frame should have consistent values down their rows. Often, each column's repeated value is the variable's average in the data. Though we could use the original data frame (`df`) to generate these averages, the resulting values may summarize different data from what was used to fit the model if there were observations that `lm()` dropped due to missing values. That happened with our model. We could try to use the original data frame and account for dropped observations, but I think it's easier to use the design matrix that's retrieved from `model.matrix()`.

The code below goes step-by-step to make the "new" data frame.

```
## create new data that has two rows, with averages and one marginal change

## (1) save model matrix
mm <- model.matrix(fit)
head(mm)
```

```
## (Intercept) byses1 female moth_ba fath_ba lowinc
## 1          1 -0.25      1         0         0         0
```

¹⁰https://en.wikipedia.org/wiki/Confusion_matrix

¹¹https://en.wikipedia.org/wiki/Training,_test,_and_validation_sets

¹²<https://www.stata.com/help.cgi?margins>

```
## 2      1  0.58      1      0      0      0
## 3      1 -0.85      1      0      0      0
## 4      1 -0.80      1      0      0      1
## 5      1 -1.41      1      0      0      1
## 6      1 -1.07      0      0      0      0
```

(2) drop intercept column of ones (predict() doesn't need them)

```
mm <- mm[,-1]
head(mm)
```

```
##  bytes1 female moth_ba fath_ba lowinc
## 1 -0.25      1      0      0      0
## 2  0.58      1      0      0      0
## 3 -0.85      1      0      0      0
## 4 -0.80      1      0      0      1
## 5 -1.41      1      0      0      1
## 6 -1.07      0      0      0      0
```

(3) convert to data frame so we can use \$ notation in next step

```
mm <- as_tibble(mm)
```

(4) new data frame of means where only lowinc changes

```
new_df <- tibble(bytes1 = mean(mm$bytes1),
                 female = mean(mm$female),
                 moth_ba = mean(mm$moth_ba),
                 fath_ba = mean(mm$fath_ba),
                 lowinc = c(0,1))
```

see new data

```
new_df
```

```
## # A tibble: 2 × 5
##  bytes1 female moth_ba fath_ba lowinc
##    <dbl> <dbl>   <dbl>   <dbl>   <dbl>
## 1 0.0421  0.503   0.274   0.320     0
## 2 0.0421  0.503   0.274   0.320     1
```

Notice how the new data frame has the same terms that were used in the original model, but has only two rows. In the `lowinc` column, the values switch from 0 to 1. All the other rows are averages of the data used to fit the model. This of course makes for a somewhat nonsensical average (what does it mean that a single father to have .32 of a BA/BS?), but that's okay. Again, what we want right now are two students who represent the "average" student except one is low income and the other is not.

To generate the prediction, we use the same function call as before, but use our `new_df` object with the `newdata` argument.

predict margins

```
predict(fit, newdata = new_df, se.fit = TRUE)
```

```
## $fit
##      1      2
## 45.82426 43.68173
##
## $se.fit
##      1      2
## 0.1166453 0.2535000
##
```



```
## $df
## [1] 15230
##
## $residual.scale
## [1] 12.24278
```

Our results show that compared to otherwise similar students, those with a family income less than \$25,000 a year are predicted to score about two points lower on their math test. To be clear, this is not a casual estimate, but rather associational. This means we would be wrong to say that having a lower family income *causes* lower math test scores (which doesn't jibe with our domain knowledge either; low income is almost certainly a proxy for other omitted variables — access to educational resources for just one thing — that are much more directly linked to test scores).

I also want to note that we held the other covariates at their means. We could have instead chosen other values (*e.g.* `fath_ba == 1` or `female == 1`), which would have given us different marginal associations. Dropping or including other covariates likely would change our results, as well. The takeaway is that margins you compute are a function of your model as well as (obviously) the margin you investigate.