Fitting regression models

EDH7916 | Spring 2020

Benjamin Skinner

After your data have been wrangled from raw values to an analysis data set and you've explored it with summary statistics and graphics, you are ready to model it and begin making inferences. As one should expect from a statistical language, R has a powerful system for fitting statistical and econometric models.

In this lesson, we'll use data from the NCES Education Longitudinal Study of 2002. Much like HSLS, ELS is a nationally representative survey that initially surveyed students in their early high school career (10th grade) and followed them into college and the workforce. We'll use a smaller version of it because, unlike HSLS, the public use files for ELS include the variables we need to properly account for the survey design when we weight our data. Here's 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

Let's load the libraries and data!

…but first! You'll most likely need to install the `survey` package first, using `install.packages("survey")`.

```
## ---------------------------
## libraries
## ---------------------------

library(tidyverse)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

✔ ggplot2 3.3.0     ✔ purrr   0.3.4
✔ tibble  3.0.1     ✔ dplyr   0.8.5
✔ tidyr   1.0.2     ✔ stringr 1.4.0
✔ readr   1.3.1     ✔ forcats 0.5.0

── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

```
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"))
```

t-test

One common statistical test is a t-test for a difference in means across groups (there are, of course, others and R can compute them). This version of the test can be computed using the R formula syntax: `y ~ x`. In our example, we'll compute base-year math scores against mother's college education level. Notice that since we have the `data = df` argument after the comma, we don't need to include `df$` before the two variables.

```
## t-test of difference in math scores across mother education (BA/BA or not)
t.test(bynels2m ~ moth_ba, data = df, var.equal = TRUE)
```

```
    Two Sample t-test

data:  bynels2m by moth_ba
t = -35.751, df = 15234, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -8.961638 -8.030040
sample estimates:
mean in group 0 mean in group 1
       43.04709        51.54293
```

Quick exercise Run a t-test of reading scores against whether the father has a Bachelor's degree (`fath_ba`).

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 OLS 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 but in a regression framework. Because we assumed equal variances between the distributions in the t-test above (`var.equal = TRUE`), we should get the same results as we did before.

```
## compute same test as above, but in a linear model
fit <- lm(bynels2m ~ moth_ba, data = df)
fit
```

```
Call:
lm(formula = bynels2m ~ moth_ba, data = df)

Coefficients:
(Intercept)      moth_ba
     43.047        8.496
```

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 ~ moth_ba, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-36.073  -9.869   0.593  10.004  36.223

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  43.0471     0.1245  345.84   <2e-16 ***
moth_ba       8.4958     0.2376   35.75   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.09 on 15234 degrees of freedom
  (924 observations deleted due to missingness)
Multiple R-squared:  0.07741,   Adjusted R-squared:  0.07735
F-statistic:  1278 on 1 and 15234 DF,  p-value: < 2.2e-16
```

Looks like the coefficient on `moth_ed`, `8.4958392`, is the same as the difference between the groups in the ttest, `8.4958392`, and the test statistics are the same value: `-35.7511914`. 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.

```
## 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 main estimate (`Estimate`)
  - the estimate error (`Std. Error`)
  - the test statistic (`t value` with this model)
  - the p value (`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, you can recover the number of observations 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"
```

For example, 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.

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 `byrace` in this model.

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

```
Call:
lm(formula = bynels2m ~ byses1 + female + moth_ba + fath_ba +
    lowinc + factor(byrace), data = df)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-39.154  -8.387   0.424   8.639  39.873

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     40.6995     1.0386  39.189  < 2e-16 ***
byses1           5.7360     0.2345  24.456  < 2e-16 ***
female          -1.1955     0.1900  -6.293 3.21e-10 ***
moth_ba          0.6500     0.2780   2.338 0.019407 *
fath_ba          0.8482     0.2796   3.033 0.002423 **
lowinc          -1.2536     0.2839  -4.416 1.01e-05 ***
factor(byrace)2  9.1538     1.0740   8.523  < 2e-16 ***
factor(byrace)3 -2.2185     1.0603  -2.092 0.036419 *
factor(byrace)4  1.2778     1.0937   1.168 0.242696
factor(byrace)5  0.2387     1.0814   0.221 0.825295
factor(byrace)6  4.2457     1.1158   3.805 0.000142 ***
factor(byrace)7  6.9514     1.0379   6.698 2.19e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.71 on 15224 degrees of freedom
  (924 observations deleted due to missingness)
Multiple R-squared:  0.2613,    Adjusted R-squared:  0.2608
F-statistic: 489.6 on 11 and 15224 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 library 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(byrace),
          data = df)
summary(fit)
```

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

Residuals:
    Min      1Q  Median      3Q     Max
-39.154  -8.387   0.424   8.639  39.873

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                 40.6995     1.0386  39.189  < 2e-16 ***
byses1                       5.7360     0.2345  24.456  < 2e-16 ***
female                      -1.1955     0.1900  -6.293 3.21e-10 ***
moth_ba                      0.6500     0.2780   2.338 0.019407 *
fath_ba                      0.8482     0.2796   3.033 0.002423 **
lowinc                      -1.2536     0.2839  -4.416 1.01e-05 ***
as_factor(byrace)asian_pi    9.1538     1.0740   8.523  < 2e-16 ***
as_factor(byrace)black_aa   -2.2185     1.0603  -2.092 0.036419 *
```

```
as_factor(byrace)hisp_nr     1.2778     1.0937   1.168 0.242696
as_factor(byrace)hisp_rs     0.2387     1.0814   0.221 0.825295
as_factor(byrace)mult_race   4.2457     1.1158   3.805 0.000142 ***
as_factor(byrace)white       6.9514     1.0379   6.698 2.19e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.71 on 15224 degrees of freedom
  (924 observations deleted due to missingness)
Multiple R-squared:  0.2613,    Adjusted R-squared:  0.2608
F-statistic: 489.6 on 11 and 15224 DF,  p-value: < 2.2e-16
```

If you look at the model matrix, you can see how R created the dummy variables from `byrace`.

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

```
  (Intercept) byses1 female moth_ba fath_ba lowinc as_factor(byrace)asian_pi
1           1  -0.25      1       0       0      0                          0
2           1   0.58      1       0       0      0                          1
3           1  -0.85      1       0       0      0                          0
4           1  -0.80      1       0       0      1                          0
5           1  -1.41      1       0       0      1                          0
6           1  -1.07      0       0       0      0                          0
  as_factor(byrace)black_aa as_factor(byrace)hisp_nr as_factor(byrace)hisp_rs
1                         0                        0                        1
2                         0                        0                        0
3                         0                        0                        0
4                         1                        0                        0
5                         0                        1                        0
6                         0                        1                        0
  as_factor(byrace)mult_race as_factor(byrace)white
1                          0                       0
2                          0                       0
3                          0                       1
4                          0                       0
5                          0                       0
6                          0                       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.

```r
## 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

To fit a model with binary outcomes, 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 logit link.

```
## logit
fit <- glm(plan_col_grad ~ bynels2m + as_factor(bypared),
           data = df,
           family = binomial())
summary(fit)
```

```
Call:
glm(formula = plan_col_grad ~ bynels2m + 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 ***
bynels2m                  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
```

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

```r
## probit
fit <- glm(plan_col_grad ~ bynels2m + as_factor(bypared),
           data = df,
           family = binomial(link = "probit"))
summary(fit)
```

```
Call:
glm(formula = plan_col_grad ~ bynels2m + 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 ***
bynels2m                   0.0326902  0.0009357  34.938  < 2e-16 ***
as_factor(bypared)hsged    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
```

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

Using survey weights

So far we haven't used survey weights, but they are very important when using survey data. To use survey weights, you'll need to use the survey package (which we've already loaded above)

To use survey weights, you need to set the survey design using the `svydesign()` function. You could do this in the `svyglm()` function we'll use to actually estimate the equation, but it's easier and clearer to do it first, store it in an object, and then use that object in the `syvglm()`.

ELS has a complex sampling design that we won't get into, but the appropriate columns from our data frame, `df`, are set to the proper arguments in `svydesign()`:

- `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`

Finally, notice the ~ before each column name, which is necessary in this function.

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

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

Now that we've set the survey design, let's compare the unweighted mean with one that accounts for the survey design;

```
## compare unweighted and survey-weighted mean of math scores
mean(df$bynels2m, na.rm = TRUE)
```

```
[1] 45.35452
```

```
svymean(~bynels2m, design = svy_df, na.rm = TRUE)
```

```
Error in svymean(~bynels2m, design = svy_df, na.rm = TRUE): could not find function "svymean"
```

We can also use the survey package to properly weight our t-tests and regression models — again, using the object `svy_df` in the `design` argument in place of our unset `df` data frame.

```
## get svymeans by group
svyby(~bynels2m, by = ~moth_ba, design = svy_df, FUN = svymean, na.rm = TRUE)
```

```
Error in svyby(~bynels2m, by = ~moth_ba, design = svy_df, FUN = svymean, : could not find function "svyby"
```

```
## t-test using survey design / weights
svyttest(bynels2m ~ moth_ba, design = svy_df, var.equal = TRUE)
```

```
Error in svyttest(bynels2m ~ moth_ba, design = svy_df, var.equal = TRUE): could not find function "svyttest"
```

QUICK EXERCISE Compare this to the output from `t.test()` above.

```
## fit the svyglm regression and show output
svyfit <- svyglm(bynels2m ~ byses1 + female + moth_ba + fath_ba + lowinc,
                 design = svy_df)
```

```
Error in svyglm(bynels2m ~ byses1 + female + moth_ba + fath_ba + lowinc, : could not find function "svyglm"
```

```
summary(svyfit)
```

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

Note that the survey library has a ton of features and is worth diving into if you regularly work with survey data.

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(bynels2m ~ 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
```

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 matrix 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 set. 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.

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 commands.

For example, after fitting our multiple regression, we might ask ourselves, what is the marginal "effect" of having a low family income on math scores, holding all other terms in our model constant?

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
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)
```

```
  byses1 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.data.frame(mm)

## (4) new data frame of means where only lowinc changes
new_df <- data.frame(byses1 = mean(mm$byses1),
                     female = mean(mm$female),
                     moth_ba = mean(mm$moth_ba),
                     fath_ba = mean(mm$fath_ba),
                     lowinc = c(0,1))

## see new data
new_df
```

```
      byses1    female   moth_ba   fath_ba lowinc
1 0.04210423 0.5027566 0.2743502 0.3195064      0
2 0.04210423 0.5027566 0.2743502 0.3195064      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.

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.

In this example, we held the other covariates at their means. We could have chosen other values (*e.g.* `fath_ba == 1` or `female == 1`), however, meaning that we could use the same procedure to produce predictions for low-income status (or other model covariates) across a range of margins.