9. Statistics & Models

Self-Paced Workshop #9

Prof. Jack Reilly

Fall 2026

From Describing to Testing

Most of this class focuses on *describing* data: counts, shares, means, a graph of a trend. That said, you likely know something about statistics from other classes, and probably know that statistics is something you can do in R. While we're not going to cover all the ground from introductory statistics and the linear model - that's for other classes - this week's lecture points you in the right direction. Our lecture examples used the NES; this workshop runs the same commands on the GSS, through `gssr` from Workshop 8.

Statistics at its most basic: the test you reach for depends on the kind of variables you have:

Variable types	**Explanatory variable categorical**	**Explanatory variable continuous**
Outcome categorical	Tabular analysis, chi-square: `chisq.test()` (Step 2)	Logit / probit: `glm()` (Step 13)
Outcome continuous	Difference of means: `t.test()` (Steps 3 and 4)	Correlation, regression: `cor.test()`, `lm()` (Steps 5 to 12)

All of these can be done in R with just a few lines of code. (Model code is, on average, much easier than wrangling code!) Remember: you're not out to memorize every function here; you're just here to dip your toe in the water and see how to calculate all the statistics you learn about in your stats classes.[1]

> **Before you start**
>
> This page uses `gssr` from Workshop 8 and `mutate()` from Workshop 4. As there, `gssr` is a 30 MB download, so the page takes several seconds longer than usual to get ready. The boxes share one R session; run them top to bottom.

[1] Because this is not a statistics course, the page explains what each number in the output *is*, not the theory behind it. For that, see the statistics references on the Week 8 page and your actual stats classes.

Step 1. Load the packages and build the data

```
library(dplyr)
library(ggplot2)
library(gssr)
data(gss_sub)
```

We will use one survey year, 2022. The labelled columns from Workshop 8 go straight into R's tests and models: each one is a number underneath its label, and the GSS's coded non-answers ("don't know", "iap" for not asked, and so on) already count as `NA`. So the only preparation is making a few 0/1 *dummy* variables, the same `if_else()` pattern the lecture used to make `white`:

```
gss22 <- gss_sub |>
  filter(year == 2022) |>
  mutate(female      = if_else(sex == 2, 1, 0),
         white       = if_else(race == 1, 1, 0),
         college     = if_else(degree >= 3, 1, 0),
         fefam_agree = if_else(fefam <= 2, 1, 0))
gss22
```

The numbers are the GSS's codes: `sex` is 1 for male and 2 for female; `race` is 1 white, 2 Black, 3 other; `degree` 3 and 4 are a bachelor's and a graduate degree; `fefam` runs from 1 (strongly agree) to 4 (strongly disagree). The printout shows each code's label, and `haven::print_labels(gss22$race)` lists every code and label for a column.[2] `polviews` is the outcome for most of the page: a 7-point scale from 1 (extremely liberal) to 7 (extremely conservative).

Part 1. "Simple" Hypothesis Tests

Each of these produces one estimate — a statistic — and the p-value that goes with it. Each also prints its results in the same layout, so learn to read one and you can read them all.

Step 2. Tabular analysis with chi-square

Two categorical variables: is agreeing that "it is better for the man to work and the woman to tend the home" (`fefam_agree`) related to sex? Start with the table, then test it:

```
table(gss22$sex, gss22$fefam_agree)
chisq.test(gss22$sex, gss22$fefam_agree)
```

The `X-squared` is the chi-square statistic and `df` its degrees of freedom; the p-value, well

[2] Because the tests and models see the codes, their output will too: groups print as `1` and `2`, not `male` and `female`. You could convert the columns to factors with `as_factor()` to get names instead, but run `haven::zap_missing()` first if you do — otherwise `as_factor()` turns the coded non-answers into categories of their own, and a regression will cheerfully estimate a coefficient for the 53 people whose race is "iap".

below 0.05, says a difference this large would be very unlikely if men and women answered alike. The test does not say *which way* the difference runs. The table does, more easily as row shares:

```
prop.table(table(gss22$sex, gss22$fefam_agree), margin = 1)
```

Row `1` is men and row `2` women: about 29% of men agree, against 20% of women. (`margin = 1` makes each row add to 1.)

Try it yourself: test whether `fefam_agree` is related to `college` instead, and use `prop.table()` to say which way.

```
table(gss22$sex, gss22$fefam_agree)
chisq.test(gss22$sex, gss22$fefam_agree)
```

Step 3. Difference in means: two groups, one variable

A continuous outcome and a two-category group: are men and women equally conservative on average? `t.test()` takes a *formula*, read "`polviews` by `sex`":

```
t.test(polviews ~ sex, data = gss22, conf.level = .95)
```

Read it from the bottom up. The two group means are 4.13 for men (group 1) and 3.88 for women (group 2) — both near the scale's midpoint of 4, "moderate", with men a quarter of a point more conservative. The confidence interval is for the *difference*, and it does not include zero; the p-value agrees.[3]

Step 4. Difference in means: two variables, same people

The other kind of t-test compares two variables measured on the same units. The GSS asks every respondent about both parents' education, so: is the average father more educated than the average mother? Each respondent supplies a *pair*, so the test is paired:

```
t.test(gss22$padeg, gss22$madeg, paired = TRUE, conf.level = .95)
```

This one uses the "two vectors" form rather than a formula. The mean difference is small — about 0.06 on the 0-4 degree scale — but with nearly 2,500 pairs it is statistically distinguishable from zero. Statistically significant and *large* are not the same thing; this is a good example of the first without the second.

Step 5. Correlation

Two continuous variables: are older people more conservative? `cor.test()` takes either form, and gives the same answer:

```
cor.test(~ polviews + age, data = gss22, method = "pearson", conf.level = .95)
cor.test(gss22$polviews, gss22$age, method = "pearson", conf.level = .95)
```

[3] The `~` is the same formula notation as `facet_wrap(~ continent)` in Workshop 7, and you will see it in every model on this page: outcome on the left, explanatory variables on the right.

A correlation of 0.13: positive, clearly not zero, and weak. Age is related to ideology, but it is far from the whole story.

Try it yourself: correlate `degree` with `padeg`. Then try `method = "spearman"`, which uses ranks and is the better choice for an ordinal scale like degree.

```
cor.test(gss22$polviews, gss22$age, method = "pearson", conf.level = .95)
```

Part 2. Models

A model has an outcome and any number of explanatory variables, so it estimates many things at once. The approach is the same however complicated the model: run it, store it as an object, then ask the object questions.

Step 6. Bivariate regression

```
lm(polviews ~ age, data = gss22)
```

How underwhelming: two numbers. The fit has much more in it, but printing the call only shows the coefficients. Save it, and ask for the `summary()`:

```
model <- lm(polviews ~ age, data = gss22)
summary(model)
```

That is the standard regression table, top to bottom:

In the output	What it is	Here
`Residuals`	the spread of the prediction errors	
`(Intercept) Estimate`	predicted `polviews` when `age` is 0	3.45
`age Estimate`	the slope: change in `polviews` per year of age	0.011
`Std. Error`	the uncertainty in each estimate	
`t value`	estimate divided by its standard error	7.4
`Pr(...)`, the last column	the p-value for that t	tiny
stars	a shorthand for the p-value, keyed on the `Signif. codes` line	***
`Multiple R-squared`	the share of the variation in `polviews` the model explains	0.017

In the output	What it is	Here
`F-statistic`	a test of the whole model at once	

A slope of 0.011 per year is about a third of a point across thirty years, and an R-squared under 2% says age explains very little of who is conservative. The line `(309 observations deleted due to missingness)` matters too: anyone missing either variable is dropped, silently in most software and politely here.

Step 7. Diagnostics

Base R draws four standard diagnostic plots from any fitted model:

```
par(mfrow = c(2, 2))
plot(model)
```

(`par(mfrow = c(2, 2))` puts them in a 2 by 2 grid; without it you get four separate plots.) The first, residuals against fitted values, is the one to look at first. Here it is a set of diagonal stripes, which looks alarming and is not: `polviews` takes only seven values, so each stripe is one answer. Diagnostics are for noticing things like that about your data, not a pass/fail test.

Step 8. Predicted values and residuals

`predict()` gives the model's prediction for every row, and `resid()` the error. A home-made version of the first diagnostic plot:

```
plot(predict(model), resid(model),
     xlab = "Predicted values", ylab = "Residuals",
     main = "Residuals vs. Predicted")
abline(h = 0, col = "red")
```

The lecture also stored them as columns, `nes2000$pred <- predict(model)`. With the GSS that fails, because `predict()` returns a value only for the 3,235 rows the model used, and the data has 3,544. Fit the model with `na.action = na.exclude` and R puts `NA` back in the dropped rows, so the lengths line up:

```
model <- lm(polviews ~ age, data = gss22, na.action = na.exclude)
gss22$pred  <- predict(model)
gss22$resid <- resid(model)
```

More often you want predictions for *particular* values. `newdata` takes a data frame with a column for each explanatory variable:

```
predict(model, newdata = data.frame(age = c(20, 50, 80)))
```

3.67 at 20, 4.00 at 50, 4.33 at 80: the slope, made concrete.

Step 9. Multiple regression

Add a second explanatory variable with `+`:

```
model2 <- lm(polviews ~ age + white, data = gss22)
summary(model2)
```

Everything from Step 6 is still there, now with a row per variable. The `white` coefficient, 0.23, is the difference between white and non-white respondents *of the same age*; the `age` slope is now the effect of age *holding race constant*, and it barely moved. `predict()` works the same way, and with two variables it becomes more useful — every combination of three ages and two groups:

```
predict(model2, newdata = data.frame(age   = c(20, 50, 80, 20, 50, 80),
                                     white = c(0, 0, 0, 1, 1, 1)))
```

Try it yourself: add `female` and `college` to `model2`. Which way does each one push `polviews`, holding the others constant?

```
model2 <- lm(polviews ~ age + white, data = gss22)
summary(model2)
```

Step 10. Interactions, and `margins`

Model 2 assumes age has the same effect on everyone. An *interaction* lets it differ by group; write it with `*`:

```
model3 <- lm(polviews ~ age * white, data = gss22)
summary(model3)
```

`age * white` expands to `age + white + age:white`, and the `age:white` row is the interaction. It is positive and significant: the age slope is steeper for white respondents. But the other rows are now hard to read — `age` alone is the slope only for non-white respondents, and `white` alone is the white/non-white difference only at age 0. The `margins` package does that arithmetic for you:

```
library(margins)
marg <- margins(model3)
summary(marg)
```

`AME` is the *average marginal effect*: the effect of each variable averaged over everyone in the data, with its standard error and confidence interval. Usually more useful is the effect of one variable at chosen values of the other:

```
marg_white <- margins(model3, variables = "age", at = list(white = c(0, 1)))
summary(marg_white)
```

Among non-white respondents, age has no detectable relationship with ideology (0.004 per year, interval includes zero). Among white respondents it is 0.012 per year and clearly

positive. That is a finding the single-slope model in Step 9 averaged away.[4]

Step 11. Categorical explanatory variables

A dummy handles two groups. For a categorical variable with more, put the *factor* in the model and R makes the dummies for you:

```
model4 <- lm(polviews ~ age + factor(race), data = gss22)
summary(model4)
```

`factor()` tells R to treat the codes 1, 2, 3 as categories rather than as a number (the lecture's `as.factor()` does the same). There are rows for `factor(race)2` and `factor(race)3` but not `factor(race)1`: R drops one category as the *baseline*, and each coefficient is a difference from it. The baseline is the first category, here 1, white — so Black respondents are 0.28 points more liberal than white respondents of the same age. To compare to a different group, change the baseline with `relevel()`:

```
model4b <- lm(polviews ~ age + relevel(factor(race), ref = "2"), data = gss22)
summary(model4b)
```

Same model, same fit, same predictions; only the comparisons in the table changed. Predictions take the codes, one row per group:

```
predict(model4, newdata = data.frame(age = 50, race = c(1, 2, 3)))
```

Step 12. Tidy model output with `broom`

`summary()` is made for reading, not for using: to graph the coefficients, you need them in a data frame. `broom::tidy()` does that, one row per coefficient:

```
library(broom)
tidy(model2, conf.int = TRUE)
```

and `glance()` puts the whole-model statistics in a single row:

```
glance(model2)
```

Once a model is a tibble, everything from Workshops 5 to 7 applies. A *coefficient plot* — each estimate with its confidence interval — is the standard graph of a model:

```
tidy(model2, conf.int = TRUE) |>
  filter(term != "(Intercept)") |>
  ggplot(aes(x = estimate, y = term)) +
  geom_pointrange(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, linetype = "dashed") +
  labs(x = "Effect on polviews (1 = extremely liberal, 7 = extremely conservative)",
       y = NULL)
```

[4] Interactions are where statistics courses spend weeks, for good reason; see the `margins` vignette for the programming side. A newer package, `marginaleffects`, does the same job with more options and is where the field is heading.

An interval that crosses the dashed line at zero is one you cannot distinguish from no effect. Neither does here — but notice that `age` sits almost on the line anyway. Its coefficient is per *year*, while `white` compares two whole groups, so the plot puts effects of very different sizes of change side by side. Coefficient plots compare best when the variables are on comparable scales.

Try it yourself: make the coefficient plot for the bigger model you built in Step 9's exercise.

```
tidy(model2, conf.int = TRUE) |>
  filter(term != "(Intercept)") |>
  ggplot(aes(x = estimate, y = term)) +
  geom_pointrange(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, linetype = "dashed")
```

Step 13. Logit and probit

When the outcome is 0/1, use `glm()`, the *generalized* linear model. It takes a formula and data like `lm()`, plus a `family` saying what kind of outcome it is — for 0/1, `binomial`, with a *link* of `"logit"` or `"probit"`:

```
logit_model <- glm(fefam_agree ~ age + female + college,
                   data   = gss22,
                   family = binomial(link = "logit"))
summary(logit_model)
```

The table looks familiar, with `z value` where `lm()` had `t value`. The signs read as before: older respondents are more likely to agree, women and college graduates less likely. The *sizes* do not read as before, because a logit coefficient is a change in the log-odds, which nobody thinks in.[5] The probit is the same idea with a different curve:

```
probit_model <- glm(fefam_agree ~ age + female + college,
                    data   = gss22,
                    family = binomial(link = "probit"))
summary(probit_model)
```

Different coefficients, same signs, same story. The way to make either one readable is predicted *probabilities*: `predict()` with `type = "response"`. For a 50-year-old man and woman, without and with a degree:

```
predict(logit_model,
        newdata = data.frame(age     = 50,
                             female  = c(0, 0, 1, 1),
                             college = c(0, 1, 0, 1)),
        type = "response")
```

[5] As the lecture noted, the outcome must be coded 0/1 for a binomial `glm()`, which is what `fefam_agree` is for. Other families handle other outcomes: `poisson` for counts, `gaussian` for an ordinary regression (which makes `glm()` give the same answer as `lm()`).

34% for a man without a degree, 14% for a woman with one. Run the same line on `probit_model` (the box below) and the probabilities agree to within a fraction of a percentage point, which is the usual result: the choice between logit and probit rarely matters.

Try it yourself: `margins(logit_model)` reports average marginal effects on the *probability* scale, the logit's answer to Step 10. Compare its `college` estimate to the gap between the predicted probabilities above.

```
predict(probit_model,
        newdata = data.frame(age     = 50,
                             female  = c(0, 0, 1, 1),
                             college = c(0, 1, 0, 1)),
        type = "response")
```

A Word on Weights

Every estimate on this page treats the GSS as a simple random sample, which it is not. For anything you would publish, the estimates should use the survey weight (`wtssps`) and the standard errors should account for the sample design — the `srvyr` and `survey` packages from Workshop 5 handle both, with their own versions of `lm()` and `glm()` (`svyglm()`). For learning what the commands do and what their output means, the unweighted versions are fine.

Work on Your Own

Run the model, store it, then ask it questions about its contents: `summary()`, `plot()`, `predict()`, `margins()`, `tidy()`. That pattern carries over to every model you will meet, however specialized — packages like `MASS` and `VGAM` add more families, and they work the same way. For more, DV ch. 6 works through models and `broom` with the same tidyverse tools, and ends with exactly the kind of coefficient and prediction graphs this page ended on.