432 Class 23

Thomas E. Love, Ph.D.

2024-04-11

## Today’s Topic

The tidymodels framework

* Using tidymodels tools to develop a linear regression model
  + Pre-processing activities
  + Model building (with multiple fitting engines)
  + Measuring model effectiveness
  + Creating a model workflow

## Setup

knitr::opts\_chunk$set(comment=NA)  
options(width = 80)  
  
library(janitor)

Attaching package: 'janitor'

The following objects are masked from 'package:stats':  
  
 chisq.test, fisher.test

library(gt)  
library(tidymodels)

── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──

✔ broom 1.0.5 ✔ recipes 1.0.9  
✔ dials 1.2.0 ✔ rsample 1.2.0  
✔ dplyr 1.1.4 ✔ tibble 3.2.1  
✔ ggplot2 3.4.4 ✔ tidyr 1.3.0  
✔ infer 1.0.5 ✔ tune 1.1.2  
✔ modeldata 1.2.0 ✔ workflows 1.1.3  
✔ parsnip 1.1.1 ✔ workflowsets 1.0.1  
✔ purrr 1.0.2 ✔ yardstick 1.2.0

── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ purrr::discard() masks scales::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
✖ recipes::step() masks stats::step()  
• Search for functions across packages at https://www.tidymodels.org/find/

library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ forcats 1.0.0 ✔ readr 2.1.5  
✔ lubridate 1.9.3 ✔ stringr 1.5.1

── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ readr::col\_factor() masks scales::col\_factor()  
✖ purrr::discard() masks scales::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ stringr::fixed() masks recipes::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ readr::spec() masks yardstick::spec()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

theme\_set(theme\_bw())

## Regression Frameworks, 1/2

Generally, regression allows us to summarize how predictions (or average values) of an outcome vary across individuals defined by a set of predictors.

Some of the most important uses of regression[[1]](#footnote-22) include:

* **Prediction**, which involves both modeling existing observations and forecasting new data.
* **Exploring Associations**, where we summarize how well a set of variables predicts the outcome.

## Regression Frameworks, 2/2

* **Extrapolation**, where we are adjusting for known differences between the observed sample of data and a population of interest.
* **Causal Inference**, where we are estimating the effect of a treatment, by comparing outcomes under treatment or control, or under different levels of a treatment.

## Research Questions for Regression Models

* “How effectively can [insert quantitative outcome] be predicted using [insert predictor(s)]?” for a linear regression project, and
* “How effectively can [insert binary outcome] be predicted using [insert predictor(s)]?” for a logistic regression project.

If you’re struggling with this, or if your research question isn’t in the form of a question, consider these approaches.

## Advantages of these RQ approaches

1. regression can help provide an answer to these questions and in discussing your results you’ll need to answer the questions
2. framing models in terms of exploring associations has some value for the tools we’re discussing and
3. it’s pretty clear what you’re doing, based just on your research question.

If you’re doing something else, I still need to think that you meet standards (1) and (3) at least.

## Using R to fit Regression Models

For linear models, we have:

* lm to fit models for quantitative outcomes, compute and plot predictions and residuals, obtain confidence intervals, etc.
* ols from the rms package to save and explore additional components of the model’s fit and to (slightly) expand the capacity for lm fits to incorporate non-linear terms and multiple imputations.

## Using R to fit Regression Models

For logistic models, we have:

* glm to fit models for binary outcomes, compute and plot predictions, hypothesis tests and confidence intervals
* lrm from rms to save and explore additional components of the model’s fit and to (slightly) expand the capacity for glm fits to incorporate non-linear terms and multiple imputations.

These are by no means the only options for fitting or working with models.

## What are tidymodels?

The tidymodels collection of packages in R use tidyverse principles to facilitate modeling and machine learning work. The key idea is to develop a consistent framework for modeling.

Visit the tidymodels website at <https://www.tidymodels.org/>.

## What is in the tidymodels framework?

* pre-processing data, which includes identifying variables and their roles, re-expression of outcomes, creation of features (predictors)
* building a model (potentially with multiple fitting “engines”)
* developing a re-usable workflow
* evaluating the fit of one model or various models with a variety of validation strategies

## Core Tidymodels Packages

Install many of the packages in the tidymodels ecosystem with install.packages(tidymodels).

When you use library(tidymodels), this makes the core packages available in your R session. They include:

* rsample which will help with data splitting and resampling
* parsnip which provides a tidy, unified interface for models
* recipes for data pre-processing and feature engineering
* yardstick for measuring model effectiveness
* broom for converting R objects into predictable formats
* workflows for bundling together pre-processing, modeling and post-processing work

as well as dials and tune, which help manage and optimize tuning parameters in certain types of models.

## Today’s Data (from Class 06)

Heart and Estrogen/Progestin Study (HERS)

* Clinical trial of hormone therapy for the prevention of recurrent heart attacks and deaths among 2763 post-menopausal women with existing coronary heart disease (see Hulley et al 1998 and many subsequent references, including Vittinghoff, Chapter 4.)
* We’re excluding the women in the trial with a diabetes diagnosis and those with missing LDL values.

## Ingesting the HERS data

hers\_raw <- read\_csv("c23/data/hersdata.csv", show\_col\_types = FALSE) |>   
 clean\_names()  
  
hers\_new <- hers\_raw |>  
 filter(diabetes == "no") |>  
 filter(complete.cases(ldl1, ldl)) |>  
 select(subject, ldl1, ldl, age, ht, globrat) |>  
 mutate(ht = factor(ht), globrat = factor(globrat),  
 subject = as.character(subject))

### Our Goal

Predict percentage change in ldl from baseline to followup, using baseline age, ht, ldl and globrat, restricted to women without diabetes.

## hers\_new Codebook (n = 1925)

| Variable | Description |
| --- | --- |
| subject | subject code |
| ht | factor: hormone therapy or placebo |
| ldl | baseline LDL cholesterol in mg/dl |
| age | baseline age in years |
| globrat | baseline self-reported health (5 levels) |
| ldl1 | LDL at first annual study visit |
| diabetes | yes or no (all are no in our sample) |

## hers\_new tibble

hers\_new

# A tibble: 1,925 × 6  
 subject ldl1 ldl age ht globrat   
 <chr> <dbl> <dbl> <dbl> <fct> <fct>   
 1 1 138. 122. 70 placebo good   
 2 2 151. 242. 62 placebo good   
 3 4 123. 116. 64 placebo good   
 4 5 172. 151. 65 placebo good   
 5 6 127. 138. 68 hormone therapy good   
 6 8 113. 121. 69 hormone therapy very good  
 7 9 106. 133 61 hormone therapy very good  
 8 10 173. 220 62 hormone therapy good   
 9 11 192 173. 72 placebo fair   
10 12 149. 124. 73 hormone therapy good   
# ℹ 1,915 more rows

## Steps we’ll describe today

1. Create our outcome and consider a transformation.
2. Split the data into training and testing samples.
3. Build a recipe for our model.
   * Specify roles for outcome and predictors.
   * Deal with missing data in a reasonable way.
   * Complete all necessary pre-processing so we can fit models.

Key Reference: Kuhn and Silge, [Tidy Modeling with R](https://www.tmwr.org/) or TMWR

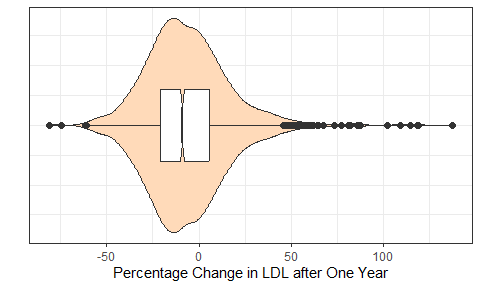
## Steps we’ll describe today

1. Specify a modeling engine for each fit we will create.
   * There are five available engines just for linear regression!
2. Create a workflow for each engine and fit model to the training data.
3. Compare coefficients graphically from two modeling approaches.
4. Assess model performance in the training data.
5. Compare models based on their test data performance.

## Stage 1: Create our outcome

hers\_new <- hers\_new |>  
 mutate(ldl\_pch = 100\*(ldl1 - ldl)/ldl)

ggplot(hers\_new, aes(x = ldl\_pch, y = 1)) +  
 geom\_violin(fill = "peachpuff") +   
 geom\_boxplot(width = 0.3, outlier.size = 2, notch = TRUE) +  
 guides(fill = "none") +  
 theme(axis.text.y = element\_blank(),  
 axis.ticks.y = element\_blank()) +  
 labs(y = "", x = "Percentage Change in LDL after One Year")

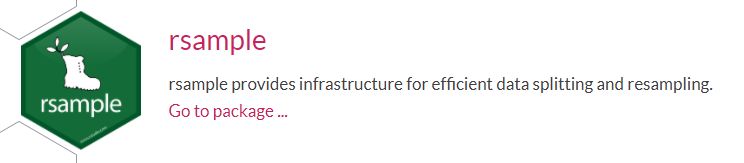


mosaic::favstats(~ ldl\_pch, data = hers\_new) |>  
 gt() |> fmt\_number(decimals = 1) |> tab\_options(table.font.size = 20)

Registered S3 method overwritten by 'mosaic':  
 method from   
 fortify.SpatialPolygonsDataFrame ggplot2

| min | Q1 | median | Q3 | max | mean | sd | n | missing |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -80.9 | -21.0 | -8.9 | 5.6 | 137.4 | -6.5 | 22.8 | 1,925.0 | 0.0 |

## Stage 2: Partition to Training/Test



Here, we’ll use the rsample package to split our data.

set.seed(20210309)  
hers\_split <- initial\_split(hers\_new, prop = 0.8)  
  
hers\_train <- training(hers\_split)  
hers\_test <- testing(hers\_split)

We start with 1925 women in hers\_new, which we split into 1540 women in the training sample, leaving 385 women in the testing sample.

## What else can we do with rsample?

* Stratified sampling (splitting) on a categorical variable to ensure similar distributions of those categories in the training and testing groups.

initial\_split(hers\_new, prop = 0.8, strata = ht)

* What if you have time series data?
  + Use initial\_time\_split() to identify the first part of the data as the training set and the rest in the test set; this assumes the data were pre-sorted in a sensible order.

## What Should Be In A Test Set?

The test set should always resemble new data we might give to the model.

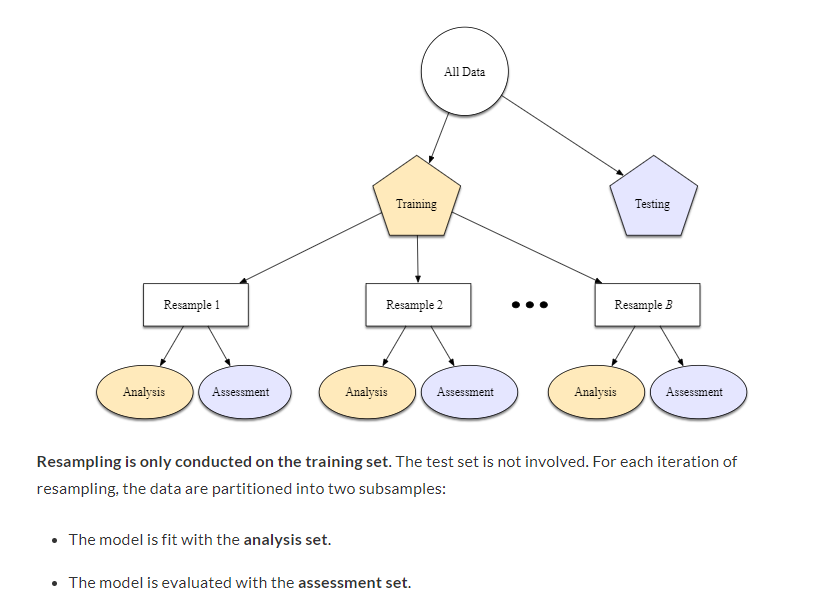
A test set should be avoided only when the data are pathologically small.

* TMWR, Section 5.2

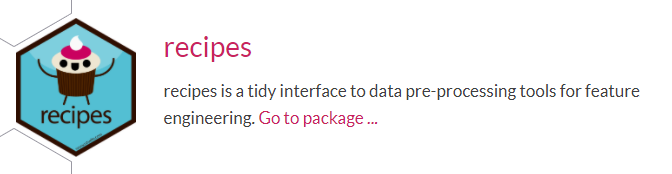
## What about a validation set?

* Would like to avoid overfitting (where the models do much better on the training set samples than you do on the test set)
* Idea is to hold back a validation set of data to measure performance while training prior to moving on with a model to the test set.
* This is really just a special case of a resampling method used on the training set, as described in TMWR section 10 (see next slide).

## From TMWR, Section 10.2



## Stage 3: Pre-Processing the Data



A recipe for pre-modeling work might

* establish the roles (outcome, predictors, identifiers) for variables
* establish pre-processing steps for predictors

<https://www.tidymodels.org/find/recipes/> lists all recipes

## Examples of Pre-Processing steps for predictors (feature engineering)

* transforming predictors, including all of our usual power transformations, but also centering, scaling or normalizing and more complex mutations
* creating dummy (indicator) variables for categorical data
* dealing with factors and factor levels
* including interactions, polynomials or splines
* filtering out variables with zero variance
* dealing with missing data via imputation or removal

## Building a Recipe for our modeling

hers\_rec <-   
 recipe(ldl\_pch ~ age + ht + ldl + globrat,   
 data = hers\_new) |> # 1  
 step\_impute\_bag(all\_predictors()) |> # 2  
 step\_poly(ldl, degree = 2) |> # 3  
 step\_dummy(all\_nominal()) |> # 4  
 step\_normalize(all\_predictors()) # 5

1. Specify the roles for the outcome and the predictors.
2. Impute missing predictors with bagged tree models.
3. Use an orthogonal polynomial of degree 2 in baseline LDL.
4. Form dummy variables to represent all categorical variables.
5. Normalize (subtract mean, divide by SD) all quantitative predictors.

## Column Roles

hers\_rec <- recipe(ldl\_pch ~ age + ht + ldl + globrat, data = hers\_new)

* Everything to the left of the ~ is an outcome.
* Everything to the right of the ~ is a predictor.

Sometimes we want to assign other roles, like “id” for an important identifier, or “split” for a splitting variable.

* Any character string can be a role, and columns can have multiple roles
* add\_role(), remove\_role() and update\_role() functions are helpful

## Common steps in a recipe, 1

* Power Transformations of Predictors
  + step\_log(x1, base = 10) (default base is exp(1)), step\_sqrt, step\_inverse
  + step\_BoxCox() will transform predictors using a simple Box-Cox transformation to make them more symmetric (remember this does require a strictly positive variable, and will be something we’d use more for an outcome using the residuals for a statistical model).
  + step\_YeoJohnson() uses the Yeo\_Johnson transformation (again, typically on the outcome model) which is like Box-Cox but doesn’t require the input variables to be strictly positive.
* step\_logit and step\_invlogit

<https://www.tidymodels.org/find/recipes/> lists available recipes

## Common steps in a recipe, 2

* Non-Linear Terms for Quantitative Predictors
  + step\_poly() produces orthogonal polynomial basis functions
  + step\_ns(x5, deg\_free = 10) from the splines package can create things called natural splines - the number of spline terms is a tuning parameter, step\_bs() adds B-spline basis functions
* Dealing with Categorical Predictors
  + step\_dummy(all\_nominal()) which converts all factor or categorical variables into indicator (also called dummy) variables: numeric variables which take 1 and 0 as values to encode the categorical information
    - Other helpful selectors: all\_numeric(), all\_predictors() and all\_outcomes()
    - If you want to select specific variables, you could use step\_dummy(x2, x3)

## Common steps in a recipe, 3

* Dealing with Categorical Predictors (continued)
  + step\_relevel() reorders the provided factor columns so that a level you specify is first (the baseline)
  + If you have ordered factors in R, try step\_unorder() to convert to regular factors or step\_ordinalscore() to map specific numeric values to each factor level
  + step\_unknown() to change missing values in a categorical variable to a dedicated factor level
  + step\_novel() creates a new factor level that may be encountered in future data
  + step\_other() converts infrequent values to a catch-all labeled “other” using a threshold
    - step\_other(x5, threshold = 0.05) places bottom 5% of data in x5 into “other”.

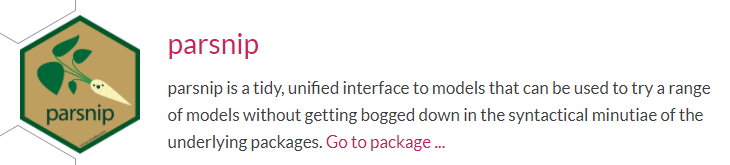
## Common steps in a recipe, 4

* Create Interaction Terms
  + step\_interact(~ interaction terms) can be used to set up interactions
* Filter rows?
  + step\_filter() can be used to filter rows using dplyr tools
* step\_mutate() can be used to conduct a variety of basic operations
* step\_ratio() can be used to create ratios of current variables
* Zero Variance Filters
  + step\_zv() is the zero variance filter which removes variables that contain only a single value.
  + step\_nzv() removes variables with very few unique values or for whom the ratio of the frequency of the most common value to the second most common value is large

## Common steps in a recipe, 5

* Centering and Scaling Predictors
  + step\_normalize() to center and scale quantitative predictors
  + step\_center() just centers predictors
  + step\_scale() just scales numeric data and
  + step\_range() to scale numeric data to a specific range
* Step options for imputation include things like
  + step\_meanimpute() and step\_medianimpute() to impute with mean or median,
  + step\_modelimpute() to impute nominal data using the most common value,
  + step\_bagimpute() for imputation via bagged trees,
  + step\_knnimpute() to impute via k-nearest neighbors
* step\_naomit() can be used to remove observations with missing values

## Stage 4: Specify modeling engine



hers\_lm\_model <- linear\_reg() |> set\_engine("lm")

Other available engines for linear regression include:

* stan to fit Bayesian models, spark, and keras

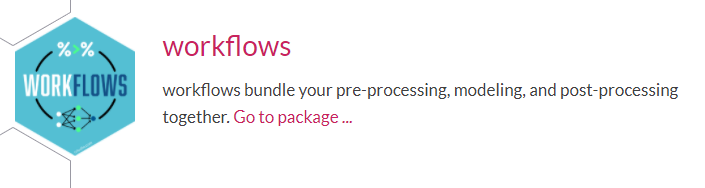
All parsnip engines: <https://www.tidymodels.org/find/parsnip/>

## Stage 4: Specify stan modeling engine for fit2

As an alternative, we’ll often consider a Bayesian linear regression model as fit with the “stan” engine. This requires the pre-specification of a prior distribution for the coefficients, for instance:

prior\_dist\_int <- rstanarm::student\_t(df = 1)  
prior\_dist\_preds <- rstanarm::student\_t(df = 1)  
  
hers\_stan\_model <- linear\_reg() |>   
 set\_engine("stan",  
 prior\_intercept = prior\_dist\_int,  
 prior = prior\_dist\_preds)

## Stage 5: Create a workflow for the lm model



hers\_lm\_wf <- workflow() |>  
 add\_model(hers\_lm\_model) |>  
 add\_recipe(hers\_rec)

### Fit the lm model to the training sample

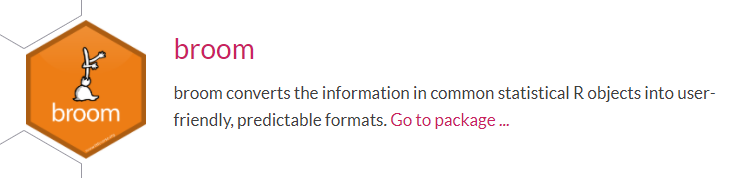
fit1 <- fit(hers\_lm\_wf, hers\_train)

## fit1 results

fit1

══ Workflow [trained] ══════════════════════════════════════════════════════════  
Preprocessor: Recipe  
Model: linear\_reg()  
  
── Preprocessor ────────────────────────────────────────────────────────────────  
4 Recipe Steps  
  
• step\_impute\_bag()  
• step\_poly()  
• step\_dummy()  
• step\_normalize()  
  
── Model ───────────────────────────────────────────────────────────────────────  
  
Call:  
stats::lm(formula = ..y ~ ., data = data)  
  
Coefficients:  
 (Intercept) age ldl\_poly\_1 ldl\_poly\_2   
 -6.3678 -0.7718 -7.8566 2.2361   
 ht\_placebo globrat\_fair globrat\_good globrat\_poor   
 4.8928 -0.7234 -1.5689 -1.1226   
globrat\_very.good   
 -1.3897

## Tidy the coefficients for fit1?



We know about tidy(), glance() and augment(), of course.

## Tidied fit1 coefficients

tidy(fit1, conf.int = T) |>   
 select(term, estimate, std.error, conf.low, conf.high) |>   
 gt() |> fmt\_number(decimals = 3) |> tab\_options(table.font.size = 20)

| term | estimate | std.error | conf.low | conf.high |
| --- | --- | --- | --- | --- |
| (Intercept) | -6.368 | 0.523 | -7.394 | -5.342 |
| age | -0.772 | 0.525 | -1.802 | 0.258 |
| ldl\_poly\_1 | -7.857 | 0.524 | -8.884 | -6.829 |
| ldl\_poly\_2 | 2.236 | 0.524 | 1.208 | 3.265 |
| ht\_placebo | 4.893 | 0.523 | 3.866 | 5.920 |
| globrat\_fair | -0.723 | 1.002 | -2.689 | 1.242 |
| globrat\_good | -1.569 | 1.196 | -3.915 | 0.777 |
| globrat\_poor | -1.123 | 0.572 | -2.244 | -0.001 |
| globrat\_very.good | -1.390 | 1.114 | -3.574 | 0.795 |

## glance at the fit1 summaries?

fit1 |> extract\_fit\_parsnip() |>   
 glance() |> select(1:6) |>   
 gt() |> fmt\_number(columns = 1:5, decimals = 3) |>  
 tab\_options(table.font.size = 20)

| r.squared | adj.r.squared | sigma | statistic | p.value | df |
| --- | --- | --- | --- | --- | --- |
| 0.180 | 0.175 | 20.522 | 41.923 | 0.000 | 8 |

fit1 |> extract\_fit\_parsnip() |>   
 glance() |> select(7:12) |>   
 gt() |> fmt\_number(columns = 1:4, decimals = 1) |>   
 tab\_options(table.font.size = 20)

| logLik | AIC | BIC | deviance | df.residual | nobs |
| --- | --- | --- | --- | --- | --- |
| -6,833.8 | 13,687.6 | 13,741.0 | 644,801.3 | 1531 | 1540 |

## Stage 5: Create a workflow for the stan model

hers\_stan\_wf <- workflow() |>  
 add\_model(hers\_stan\_model) |>  
 add\_recipe(hers\_rec)

### Fit the stan model to the training sample

set.seed(43202)  
fit2 <- fit(hers\_stan\_wf, hers\_train)

## fit2 results

fit2

══ Workflow [trained] ══════════════════════════════════════════════════════════  
Preprocessor: Recipe  
Model: linear\_reg()  
  
── Preprocessor ────────────────────────────────────────────────────────────────  
4 Recipe Steps  
  
• step\_impute\_bag()  
• step\_poly()  
• step\_dummy()  
• step\_normalize()  
  
── Model ───────────────────────────────────────────────────────────────────────  
stan\_glm  
 family: gaussian [identity]  
 formula: ..y ~ .  
 observations: 1540  
 predictors: 9  
------  
 Median MAD\_SD  
(Intercept) -6.3 0.5   
age -0.7 0.5   
ldl\_poly\_1 -7.8 0.5   
ldl\_poly\_2 2.2 0.5   
ht\_placebo 4.8 0.5   
globrat\_fair -0.2 0.8   
globrat\_good -0.9 0.9   
globrat\_poor -0.9 0.5   
globrat\_very.good -0.8 0.9   
  
Auxiliary parameter(s):  
 Median MAD\_SD  
sigma 20.5 0.4   
  
------  
\* For help interpreting the printed output see ?print.stanreg  
\* For info on the priors used see ?prior\_summary.stanreg

## Tidy the fit2 coefficients?

stan model requires the broom.mixed package to tidy fit.

broom.mixed::tidy(fit2, conf.int = T) |> gt() |>   
 fmt\_number(decimals = 3) |> tab\_options(table.font.size = 20)

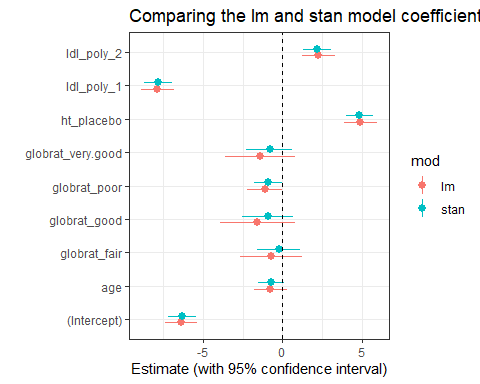
| term | estimate | std.error | conf.low | conf.high |
| --- | --- | --- | --- | --- |
| (Intercept) | -6.288 | 0.541 | -7.179 | -5.410 |
| age | -0.741 | 0.507 | -1.536 | 0.084 |
| ldl\_poly\_1 | -7.796 | 0.545 | -8.697 | -6.923 |
| ldl\_poly\_2 | 2.157 | 0.541 | 1.284 | 3.019 |
| ht\_placebo | 4.806 | 0.515 | 3.960 | 5.658 |
| globrat\_fair | -0.227 | 0.800 | -1.571 | 1.097 |
| globrat\_good | -0.913 | 0.914 | -2.548 | 0.640 |
| globrat\_poor | -0.913 | 0.519 | -1.810 | -0.054 |
| globrat\_very.good | -0.788 | 0.867 | -2.286 | 0.609 |

## Stage 6: Compare model coefficients

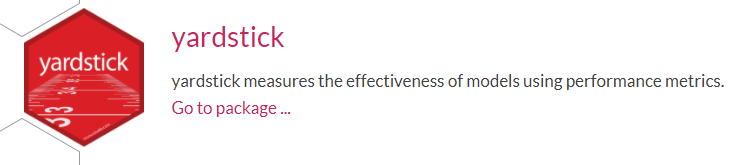
coefs\_lm <- tidy(fit1, conf.int = TRUE) |>  
 select(term, estimate, conf.low, conf.high) |>  
 mutate(mod = "lm")  
  
coefs\_stan <- tidy(fit2, conf.int = TRUE) |>  
 select(term, estimate, conf.low, conf.high) |>  
 mutate(mod = "stan")  
  
coefs\_comp <- bind\_rows(coefs\_lm, coefs\_stan)

## Graph model coefficients

ggplot(coefs\_comp, aes(x = term, y = estimate, col = mod,  
 ymin = conf.low, ymax = conf.high)) +  
 geom\_point(position = position\_dodge2(width = 0.4)) +  
 geom\_pointrange(position = position\_dodge2(width = 0.4)) +  
 geom\_hline(yintercept = 0, lty = "dashed") +  
 coord\_flip() +  
 labs(x = "", y = "Estimate (with 95% confidence interval)",  
 title = "Comparing the lm and stan model coefficients")



## Stage 7. Assess model in training sample



Available regression performance metrics include:

* rsq (r-squared, via correlation - always between 0 and 1)
* rmse (root mean squared error)
* mae (mean absolute error)
* rsq\_trad (r-squared, calculated via sum of squares)

## Make predictions using fit1 in training sample

lm\_pred\_train <-   
 predict(fit1, hers\_train) |>  
 bind\_cols(hers\_train |> dplyr::select(ldl\_pch))  
  
# we'll use R-squared and RMSE here, today  
mets <- metric\_set(rsq, rmse)  
  
lm\_res\_train <-   
 mets(lm\_pred\_train, truth = ldl\_pch, estimate = .pred)

## Make predictions using fit2 in training sample

stan\_pred\_train <-   
 predict(fit2, hers\_train) |>  
 bind\_cols(hers\_train |> select(ldl\_pch))  
  
# remember  
mets <- metric\_set(rsq, rmse)  
  
stan\_res\_train <-   
 mets(stan\_pred\_train, truth = ldl\_pch, estimate = .pred)

We’ll see the results from each fit on the next slide.

## fit1 and fit2 (training sample)

lm\_res\_train |> gt() |>   
 fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| rsq | standard | 0.1797 |
| rmse | standard | 20.4622 |

stan\_res\_train |> gt() |>   
 fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| rsq | standard | 0.1795 |
| rmse | standard | 20.4651 |

## What about adjusted ?

The yardstick package doesn’t use adjusted .

* tidymodels wants you to compute performance on a separate data set for comparing models rather than doing what adjusted tries to do, which is evaluate the model on the same data as were used to fit the model.

I wrote more on the adjusted statistic in the Class 23 README.

## Stage 8. Compare model performance on test data

lm\_pred\_test <-   
 predict(fit1, hers\_test) |>  
 bind\_cols(hers\_test |> dplyr::select(ldl\_pch))  
  
lm\_res\_test <-   
 mets(lm\_pred\_test, truth = ldl\_pch, estimate = .pred)  
  
stan\_pred\_test <-   
 predict(fit2, hers\_test) |>  
 bind\_cols(hers\_test |> select(ldl\_pch))  
  
stan\_res\_test <-   
 mets(stan\_pred\_test, truth = ldl\_pch, estimate = .pred)

## fit1 and fit2 (training sample)

lm\_res\_test |> gt() |>   
 fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| rsq | standard | 0.1998 |
| rmse | standard | 21.0827 |

stan\_res\_test |> gt() |>   
 fmt\_number(decimals = 4) |> tab\_options(table.font.size = 20)

| .metric | .estimator | .estimate |
| --- | --- | --- |
| rsq | standard | 0.2000 |
| rmse | standard | 21.0847 |

## Where to Learn More

* [Tidy Modeling with R](https://www.tmwr.org/) by Max Kuhn and Julia Silge.
  + The Basics section (Chapters 4-9) as well as chapters 10-11 were my main tools for learning about these ideas.
* Julia Silge has many nice videos on YouTube demonstrating various things that tidymodels can accomplish.
  + I’ve recommended several in today’s README.

### Next Time

We’ll apply ideas from the tidymodels framework to fit a logistic regression model.

1. Gelman, Hill and Vehtari, *Regression and Other Stories* [↑](#footnote-ref-22)