Notes on Ch 6: Fitting Models with parsnip

The Caveman Coder

2025-08-19

Syntax for a linear regression model using `lm()`:

```
model <- lm(formula, data, ...)
```

Syntax for a linear regression model with regularization:

```
# Using the rstanarm package (Bayesian model):
model <- stan_glm(formula, data, family = "gaussian", ...)

# Using the glmnet package (non-Bayesian model)
model <- glmnet(x = matrix, y = vector, family = "gaussian", ...)
```

Prerequisites

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.3.0 --

## v broom        1.0.8     v recipes      1.3.1
## v dials        1.4.1     v rsample      1.3.1
## v dplyr        1.1.4     v tibble       3.3.0
## v ggplot2      3.5.2     v tidyr        1.3.1
## v infer        1.0.9     v tune         1.3.0
## v modeldata    1.5.0     v workflows    1.2.0
## v parsnip      1.3.2     v workflowsets 1.1.1
## v purrr        1.1.0     v yardstick    1.3.2

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
```

```
tidymodels_prefer()
```

General steps when modeling using the tidymodels approach:

1. Specify the type of model based on its mathematical structure (i.e., linear regression, random forest, KNN, etc.).

2. Specify the engine for fitting the model. This, most often reflects the software package that should be used, like Stan, glmnet, or others.

3. When required, declare the mode of the model (i.e., the type of prediction outcome - if numeric, "regression"; if qualitative, "classification").

Example: Walkthrough on predicting sale of house prices

This is a walkthrough example on making a predictiion model for house prices based on longitude and latitude using the Ames data:

```
data(ames)
glimpse(ames)
```

```
## Rows: 2,930
## Columns: 74
## $ MS_SubClass        <fct> One_Story_1946_and_Newer_All_Styles, One_Story_1946~
## $ MS_Zoning          <fct> Residential_Low_Density, Residential_High_Density, ~
## $ Lot_Frontage       <dbl> 141, 80, 81, 93, 74, 78, 41, 43, 39, 60, 75, 0, 63,~
## $ Lot_Area           <int> 31770, 11622, 14267, 11160, 13830, 9978, 4920, 5005~
## $ Street             <fct> Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pave, Pav~
## $ Alley              <fct> No_Alley_Access, No_Alley_Access, No_Alley_Access, ~
## $ Lot_Shape          <fct> Slightly_Irregular, Regular, Slightly_Irregular, Re~
## $ Land_Contour       <fct> Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, Lvl, HLS, Lvl, Lvl, L~
## $ Utilities          <fct> AllPub, AllPub, AllPub, AllPub, AllPub, AllPub, All~
## $ Lot_Config         <fct> Corner, Inside, Corner, Corner, Inside, Inside, Ins~
## $ Land_Slope         <fct> Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, Gtl, G~
## $ Neighborhood       <fct> North_Ames, North_Ames, North_Ames, North_Ames, Gil~
## $ Condition_1        <fct> Norm, Feedr, Norm, Norm, Norm, Norm, Norm, Norm, No~
## $ Condition_2        <fct> Norm, Norm, Norm, Norm, Norm, Norm, Norm, Norm, Nor~
## $ Bldg_Type          <fct> OneFam, OneFam, OneFam, OneFam, OneFam, OneFam, Twn~
## $ House_Style        <fct> One_Story, One_Story, One_Story, One_Story, Two_Sto~
## $ Overall_Cond       <fct> Average, Above_Average, Above_Average, Average, Ave~
## $ Year_Built         <int> 1960, 1961, 1958, 1968, 1997, 1998, 2001, 1992, 199~
## $ Year_Remod_Add     <int> 1960, 1961, 1958, 1968, 1998, 1998, 2001, 1992, 199~
## $ Roof_Style         <fct> Hip, Gable, Hip, Hip, Gable, Gable, Gable, Gable, G~
## $ Roof_Matl          <fct> CompShg, CompShg, CompShg, CompShg, CompShg, CompSh~
## $ Exterior_1st       <fct> BrkFace, VinylSd, Wd Sdng, BrkFace, VinylSd, VinylS~
## $ Exterior_2nd       <fct> Plywood, VinylSd, Wd Sdng, BrkFace, VinylSd, VinylS~
## $ Mas_Vnr_Type       <fct> Stone, None, BrkFace, None, None, BrkFace, None, No~
## $ Mas_Vnr_Area       <dbl> 112, 0, 108, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6~
## $ Exter_Cond         <fct> Typical, Typical, Typical, Typical, Typical, Typica~
## $ Foundation         <fct> CBlock, CBlock, CBlock, CBlock, PConc, PConc, PConc~
## $ Bsmt_Cond          <fct> Good, Typical, Typical, Typical, Typical, Typical, ~
## $ Bsmt_Exposure      <fct> Gd, No, No, No, No, No, Mn, No, No, No, No, No, No,~
## $ BsmtFin_Type_1     <fct> BLQ, Rec, ALQ, ALQ, GLQ, GLQ, GLQ, ALQ, GLQ, Unf, U~
## $ BsmtFin_SF_1       <dbl> 2, 6, 1, 1, 3, 3, 3, 1, 3, 7, 7, 1, 7, 3, 3, 1, 3, ~
## $ BsmtFin_Type_2     <fct> Unf, LwQ, Unf, Unf, Unf, Unf, Unf, Unf, Unf, Unf, U~
## $ BsmtFin_SF_2       <dbl> 0, 144, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1120, 0~
## $ Bsmt_Unf_SF        <dbl> 441, 270, 406, 1045, 137, 324, 722, 1017, 415, 994,~
## $ Total_Bsmt_SF      <dbl> 1080, 882, 1329, 2110, 928, 926, 1338, 1280, 1595, ~
## $ Heating            <fct> GasA, GasA, GasA, GasA, GasA, GasA, GasA, GasA, Gas~
## $ Heating_QC         <fct> Fair, Typical, Typical, Excellent, Good, Excellent,~
## $ Central_Air        <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, ~
## $ Electrical         <fct> SBrkr, SBrkr, SBrkr, SBrkr, SBrkr, SBrkr, SBrkr, SB~
## $ First_Flr_SF       <int> 1656, 896, 1329, 2110, 928, 926, 1338, 1280, 1616, ~
## $ Second_Flr_SF      <int> 0, 0, 0, 0, 701, 678, 0, 0, 0, 776, 892, 0, 676, 0,~
## $ Gr_Liv_Area        <int> 1656, 896, 1329, 2110, 1629, 1604, 1338, 1280, 1616~
## $ Bsmt_Full_Bath     <dbl> 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, ~
## $ Bsmt_Half_Bath     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ Full_Bath          <int> 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 3, 2, ~
```

```
## $ Half_Bath          <int> 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, ~
## $ Bedroom_AbvGr      <int> 3, 2, 3, 3, 3, 3, 2, 2, 2, 3, 3, 3, 3, 2, 1, 4, 4, ~
## $ Kitchen_AbvGr      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ TotRms_AbvGrd      <int> 7, 5, 6, 8, 6, 7, 6, 5, 5, 7, 7, 6, 7, 5, 4, 12, 8,~
## $ Functional         <fct> Typ, Typ, Typ, Typ, Typ, Typ, Typ, Typ, Typ, Typ, T~
## $ Fireplaces         <int> 2, 0, 0, 2, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, ~
## $ Garage_Type        <fct> Attchd, Attchd, Attchd, Attchd, Attchd, Attchd, Att~
## $ Garage_Finish      <fct> Fin, Unf, Unf, Fin, Fin, Fin, Fin, RFn, RFn, Fin, F~
## $ Garage_Cars        <dbl> 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, ~
## $ Garage_Area        <dbl> 528, 730, 312, 522, 482, 470, 582, 506, 608, 442, 4~
## $ Garage_Cond        <fct> Typical, Typical, Typical, Typical, Typical, Typica~
## $ Paved_Drive        <fct> Partial_Pavement, Paved, Paved, Paved, Paved, Paved~
## $ Wood_Deck_SF       <int> 210, 140, 393, 0, 212, 360, 0, 0, 237, 140, 157, 48~
## $ Open_Porch_SF      <int> 62, 0, 36, 0, 34, 36, 0, 82, 152, 60, 84, 21, 75, 0~
## $ Enclosed_Porch     <int> 0, 0, 0, 0, 0, 0, 170, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ Three_season_porch <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ Screen_Porch       <int> 0, 120, 0, 0, 0, 0, 0, 144, 0, 0, 0, 0, 0, 0, 140, ~
## $ Pool_Area          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ Pool_QC            <fct> No_Pool, No_Pool, No_Pool, No_Pool, No_Pool, No_Poo~
## $ Fence              <fct> No_Fence, Minimum_Privacy, No_Fence, No_Fence, Mini~
## $ Misc_Feature       <fct> None, None, Gar2, None, None, None, None, None, Non~
## $ Misc_Val           <int> 0, 0, 12500, 0, 0, 0, 0, 0, 0, 0, 0, 500, 0, 0, 0, ~
## $ Mo_Sold            <int> 5, 6, 6, 4, 3, 6, 4, 1, 3, 6, 4, 3, 5, 2, 6, 6, 6, ~
## $ Year_Sold          <int> 2010, 2010, 2010, 2010, 2010, 2010, 2010, 2010, 201~
## $ Sale_Type          <fct> WD , WD , WD , WD , WD , WD , WD , WD , WD , WD , W~
## $ Sale_Condition     <fct> Normal, Normal, Normal, Normal, Normal, Normal, Nor~
## $ Sale_Price         <int> 215000, 105000, 172000, 244000, 189900, 195500, 213~
## $ Longitude          <dbl> -93.61975, -93.61976, -93.61939, -93.61732, -93.638~
## $ Latitude           <dbl> 42.05403, 42.05301, 42.05266, 42.05125, 42.06090, 4~
```

```r
ames <- ames |> mutate(Sale_Price = log10(Sale_Price))
```

Splitting the data into training and testing sets:

```r
set.seed(502)

ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)

ames_train <- training(ames_split)
ames_test <- testing(ames_split)

dim(ames_train)
```

```
## [1] 2342   74
```

```r
dim(ames_test)
```

```
## [1] 588  74
```

Creating the model:

```r
lm_model <-
  linear_reg() |>
  set_engine("lm")

lm_form_fit <-
  lm_model |>
```

```r
  fit(Sale_Price ~ Longitude + Latitude, data = ames_train)

lm_xy_fit <-
  lm_model |>
  fit_xy(
    x = ames_train |> select(Longitude, Latitude),
    y = ames_train |> pull(Sale_Price)
  )

lm_form_fit
```

```
## parsnip model object
##
##
## Call:
## stats::lm(formula = Sale_Price ~ Longitude + Latitude, data = data)
##
## Coefficients:
## (Intercept)    Longitude     Latitude
##    -302.974       -2.075        2.710
```

```r
lm_xy_fit
```

```
## parsnip model object
##
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)    Longitude     Latitude
##    -302.974       -2.075        2.710
```

The interface from different packages used for the models above are consistent – this is due to the parsnip package.

Using the model results

Extracting the model fitting results using `extract_fit_engine()`:

```r
lm_form_fit |> extract_fit_engine()
```

```
##
## Call:
## stats::lm(formula = Sale_Price ~ Longitude + Latitude, data = data)
##
## Coefficients:
## (Intercept)    Longitude     Latitude
##    -302.974       -2.075        2.710
```

```r
lm_xy_fit |> extract_fit_engine()
```

```
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
```

```
## (Intercept)    Longitude     Latitude 
##    -302.974       -2.075        2.710
```

Normal methods can be applied to the extracted results:

```r
lm_form_fit |> extract_fit_engine() |> vcov()
```

```
##              (Intercept)     Longitude      Latitude
## (Intercept)  207.311311  1.5746587743 -1.4239709610
## Longitude      1.574659  0.0165462548 -0.0005999802
## Latitude      -1.423971 -0.0005999802  0.0325397353
```

```r
lm_xy_fit |> extract_fit_engine() |> vcov()
```

```
##              (Intercept)     Longitude      Latitude
## (Intercept)  207.311311  1.5746587743 -1.4239709610
## Longitude      1.574659  0.0165462548 -0.0005999802
## Latitude      -1.423971 -0.0005999802  0.0325397353
```

Extracting the results using the `summary()` method:

```r
model_res <-
  lm_form_fit |>
  extract_fit_engine() |>
  summary()

model_res
```

```
## 
## Call:
## stats::lm(formula = Sale_Price ~ Longitude + Latitude, data = data)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.02861 -0.09798 -0.01345  0.09648  0.57925 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -302.9736    14.3983  -21.04   <2e-16 ***
## Longitude     -2.0749     0.1286  -16.13   <2e-16 ***
## Latitude       2.7097     0.1804   15.02   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1604 on 2339 degrees of freedom
## Multiple R-squared:  0.1684, Adjusted R-squared:  0.1677 
## F-statistic: 236.8 on 2 and 2339 DF,  p-value: < 2.2e-16
```

Acessing the model coefficient table using the `coef()` method:

```r
param_est <- coef(model_res)
class(param_est)
```

```
## [1] "matrix" "array"
```

```r
param_est
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -302.973554 14.3983093 -21.04230 3.640103e-90
```

```
## Longitude     -2.074862  0.1286322 -16.13019 1.395257e-55
## Latitude       2.709654  0.1803877  15.02128 9.289500e-49
```

Notice that the column names of the result are not valid names for a dataframe in R. Also, the p-value colummn might end up with a different name had we used another model. This is not ideal when we are collecting results from different models.

The solution is to use the **broom** package which can convert many types of model objects into a tidy structure. Using this package on our example:

```
tidy(lm_form_fit)
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  -303.      14.4       -21.0 3.64e-90
## 2 Longitude      -2.07     0.129     -16.1 1.40e-55
## 3 Latitude        2.71     0.180      15.0 9.29e-49
```

Making predictions

Model prediction is generally performed using the `predict()` method. In parsnip, predictions are always performed under these rules:

1. The results are always a tibble.
2. The column names of the tibble are always predictable (or consistent).
3. There are always as many rows in the tibble as there are in the input data set.

Example:

```
ames_test_small <- ames_test |> slice(1:5)
predict(lm_form_fit, new_data = ames_test_small)
```

```
## # A tibble: 5 x 1
##   .pred
##   <dbl>
## 1  5.22
## 2  5.21
## 3  5.28
## 4  5.27
## 5  5.28
```

The three rules make it easy to merge the predictions with the original data:

```
ames_test_small |>
  select(Sale_Price) |>
  bind_cols(predict(lm_form_fit, ames_test_small)) |>
  # add 95% prediction intervals to the results:
  bind_cols(predict(lm_form_fit, ames_test_small, type = "pred_int"))
```

```
## # A tibble: 5 x 4
##   Sale_Price .pred .pred_lower .pred_upper
##        <dbl> <dbl>       <dbl>       <dbl>
## 1       5.02  5.22        4.91        5.54
## 2       5.39  5.21        4.90        5.53
## 3       5.28  5.28        4.97        5.60
## 4       5.28  5.27        4.96        5.59
```

```
## 5         5.28  5.28         4.97         5.60
```

Example using a decision tree:

```
tree_model <-
  decision_tree(min_n = 2) |>
  set_engine("rpart") |>
  set_mode("regression")

tree_fit <-
  tree_model |>
  fit(Sale_Price ~ Longitude + Latitude, data = ames_train)

ames_test_small |>
  select(Sale_Price) |>
  bind_cols(predict(tree_fit, ames_test_small))
```

```
## # A tibble: 5 x 2
##   Sale_Price .pred
##        <dbl> <dbl>
## 1       5.02  5.15
## 2       5.39  5.15
## 3       5.28  5.32
## 4       5.28  5.32
## 5       5.28  5.32
```

Parsnip extension packages

The **descrim** package, for example, has model definitions for the family of classification techniques called discriminant analysis methods.

Creating model specifications

From the *Addins* toolbar menu of RStudio, we can see a list of possible models for each model mode. These can be written to the source code panel.