Notes on Ch7: Workflow Basics

The Caveman Coder

2025-08-19

Why are workflows important?

- it encourages good methodology since it is a single point of entry to the components of data analysis.
- it enables the user to better organize projects.

Workflow basics

Setting up the libraries and the dataset:

```r
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()
```

```r
tidymodels_prefer()
data(ames)

ames <- mutate(ames, Sale_Price = log10(Sale_Price))

set.seed(502)
ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test <- testing(ames_split)
```

Setting up the model engine:

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

Setting up the workflow (this always requires a parsnip model object):

```r
lm_wflow <-
  workflow() |>
  add_model(lm_model)

lm_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: None
## Model: linear_reg()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Adding the formula to the workflow:

```r
lm_wflow <-
  lm_wflow |>
  add_formula(Sale_Price ~ Longitude + Latitude)
```

Creating the model by fitting the training data to the parsnip model object:

```r
lm_fit <- fit(lm_wflow, ames_train)
lm_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Formula
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Sale_Price ~ Longitude + Latitude
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)    Longitude     Latitude
##    -302.974       -2.075        2.710
```

Making predictions with the model:

```r
predict(lm_fit, ames_test |> slice(1:3))
```

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

Updating the model and preprocessor:

```r
lm_fit |> update_formula(Sale_Price ~ Longitude)
```

```
## == Workflow ====================================================================
```

```
## Preprocessor: Formula
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Sale_Price ~ Longitude
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Adding raw variables to the `workflow()`

We can do this using the `add_variables()` function. This has two primary arguments: `outcome` and `predictors`.

Example:

```
lm_wflow <-
  lm_wflow |>
  remove_formula() |>
  add_variables(outcome = Sale_Price, predictors =c(Longitude, Latitude))

lm_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Variables
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Outcomes: Sale_Price
## Predictors: c(Longitude, Latitude)
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

There are many wasy to specify the predictors. For example, it could have beens specified as:

```
predictors = c(ends_width("tude))
```

Or as:

```
predictors = everything()
```

Updating the model by fitting the training data to the updated parsnip model object:

```
fit(lm_wflow, ames_train)
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Variables
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Outcomes: Sale_Price
## Predictors: c(Longitude, Latitude)
##
```

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

Special formulas and inline functions

Fitting a regression model that has random effects for subject, to the `Orthodont` data:

```
library(nlme)
data("Orthodont")

library(lme4)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
lmer(distance ~ Sex + (age | Subject), data = Orthodont)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: distance ~ Sex + (age | Subject)
##    Data: Orthodont
## REML criterion at convergence: 471.1635
## Random effects:
##  Groups   Name        Std.Dev. Corr
##  Subject  (Intercept) 7.3912
##           age         0.6943   -0.97
##  Residual             1.3100
## Number of obs: 108, groups:  Subject, 27
## Fixed Effects:
## (Intercept)    SexFemale
##      24.517       -2.145
```

The problem with this approach is that standard R can't properly process this formula:

```
model.matrix(distance ~ Sex + (age | Subject), data = Orthodont)
```

```
## Warning in Ops.ordered(age, Subject): '|' is not meaningful for ordered factors
```

```
##      (Intercept) SexFemale age | SubjectTRUE
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$Sex
## [1] "contr.treatment"
##
## attr(,"contrasts")$`age | Subject`
## [1] "contr.treatment"
```

The solution in workflows is using an optional supplementary model that can be passed to `add_model()`. The `add_variables()` can do the trick:

```
library(multilevelmod)

multilevel_spec <- linear_reg() |>  set_engine("lmer")

multilevel_workflow <-
  workflow() |>
  # Pass the data along as-is:
  add_variables(outcome = distance, predictors = c(Sex, age, Subject)) |>
  add_model(multilevel_spec,
            # This formula is given to the model
            formula = distance ~ Sex + (age | Subject))

multilevel_fit <- fit(multilevel_workflow, data = Orthodont)
multilevel_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Variables
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Outcomes: distance
## Predictors: c(Sex, age, Subject)
##
## -- Model -----------------------------------------------------------------------
## Linear mixed model fit by REML ['lmerMod']
## Formula: distance ~ Sex + (age | Subject)
##    Data: data
## REML criterion at convergence: 471.1635
## Random effects:
##  Groups   Name        Std.Dev. Corr
##  Subject  (Intercept) 7.3912
##           age         0.6943   -0.97
##  Residual             1.3100
## Number of obs: 108, groups:  Subject, 27
## Fixed Effects:
## (Intercept)    SexFemale
##      24.517       -2.145
```

Another example using the `strata()` function from the **survival** package:

```
library(censored)
```

```
## Loading required package: survival
```

```
parametric_spec <- survival_reg()

parametric_workflow <-
  workflow() |>
  add_variables(outcome = c(fustat, futime), predictors = c(age, rx)) |>
  add_model(parametric_spec,
            formula = Surv(futime, fustat) ~ age + strata(rx))

parametric_fit <- fit(parametric_workflow, data = ovarian)
```

```
parametric_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Variables
## Model: survival_reg()
##
## -- Preprocessor ------------------------------------------------------------------
## Outcomes: c(fustat, futime)
## Predictors: c(age, rx)
##
## -- Model -------------------------------------------------------------------------
## Call:
## survival::survreg(formula = Surv(futime, fustat) ~ age + strata(rx),
##     data = data, model = TRUE)
##
## Coefficients:
## (Intercept)         age
##  12.8734120  -0.1033569
##
## Scale:
##      rx=1      rx=2
## 0.7695509 0.4703602
##
## Loglik(model)= -89.4   Loglik(intercept only)= -97.1
##  Chisq= 15.36 on 1 degrees of freedom, p= 8.88e-05
## n= 26
```

Creating multiple workflows at once

Example: modeling the different ways that house location is represented in the Ames dataset.

Creating a list of formulas that capture the predictors:

```
location <- list(
  longitude = Sale_Price ~ Longitude,
  latitude = Sale_Price ~ Latitude,
  coords = Sale_Price ~ Longitude + Latitude,
  neighborhood = Sale_Price ~ Neighborhood
)
```

Using the `workflowsets` library to be able to cross the representations above with one or more models with the `workflow_set()` function:

```
library(workflowsets)

location_models <- workflow_set(preproc = location, models = list(lm = lm_model))

location_models
```

```
## # A workflow set/tibble: 4 x 4
##   wflow_id        info             option    result
##   <chr>           <list>           <list>    <list>
## 1 longitude_lm    <tibble [1 x 4]> <opts[0]> <list [0]>
## 2 latitude_lm     <tibble [1 x 4]> <opts[0]> <list [0]>
## 3 coords_lm       <tibble [1 x 4]> <opts[0]> <list [0]>
## 4 neighborhood_lm <tibble [1 x 4]> <opts[0]> <list [0]>
```

We can “see” deeper into these workflow sets:

```
location_models$info[[1]]
```

```
## # A tibble: 1 x 4
##   workflow   preproc model      comment
##   <list>     <chr>   <chr>      <chr>
## 1 <workflow> formula linear_reg ""
```

We can extract the model details using `extract_workflow()`:

```
extract_workflow(location_models, id = "coords_lm")
```

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Sale_Price ~ Longitude + Latitude
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Creating model “fits” for each formula and saving them in a new column called `fit`:

```
location_models <-
  location_models |>
  mutate(fit = map(info, ~ fit(.x$workflow[[1]], ames_train)))

location_models
```

```
## # A workflow set/tibble: 4 x 5
##   wflow_id        info             option    result     fit
##   <chr>           <list>           <list>    <list>     <list>
## 1 longitude_lm    <tibble [1 x 4]> <opts[0]> <list [0]> <workflow>
## 2 latitude_lm     <tibble [1 x 4]> <opts[0]> <list [0]> <workflow>
## 3 coords_lm       <tibble [1 x 4]> <opts[0]> <list [0]> <workflow>
## 4 neighborhood_lm <tibble [1 x 4]> <opts[0]> <list [0]> <workflow>
```

Again, we can “see” the model details using base R functions:

```
location_models$fit[[1]]
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Formula
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## Sale_Price ~ Longitude
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)    Longitude
```

```
##    -184.396       -2.025
```

Evaluating the test set

Using the function called `last_fit()` to fit the model to the entire training set and evaluate it with the testing set:

```
final_lm_res <- last_fit(lm_wflow, ames_split)
final_lm_res
```

```
## # Resampling results
## # Manual resampling
## # A tibble: 1 x 6
##   splits             id               .metrics .notes   .predictions .workflow
##   <list>             <chr>            <list>   <list>   <list>       <list>
## 1 <split [2342/588]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

Note: the `last_fit()` function takes a data split object as an input – not a dataframe.

Pulling out the `.workflow` column from the results using `extract_workflow()`:

```
fitted_lm_wflow <- extract_workflow(final_lm_res)
```

Collecting the predictions and metrics:

```
collect_metrics(final_lm_res)
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.164 Preprocessor1_Model1
## 2 rsq     standard       0.189 Preprocessor1_Model1
```

```
collect_predictions(final_lm_res) |> slice(1:5)
```

```
## # A tibble: 5 x 5
##   .pred id                .row Sale_Price .config
##   <dbl> <chr>            <int>      <dbl> <chr>
## 1  5.22 train/test split     2       5.02 Preprocessor1_Model1
## 2  5.21 train/test split     4       5.39 Preprocessor1_Model1
## 3  5.28 train/test split     5       5.28 Preprocessor1_Model1
## 4  5.27 train/test split     8       5.28 Preprocessor1_Model1
## 5  5.28 train/test split    10       5.28 Preprocessor1_Model1
```