Exercise 14

Marc Dotson

Return again to soup\_data and the models from the previous exercises.

1. Check for right-skewed continuous variables.
2. Prepare a recipe using the necessary steps.
3. Fit the models again on the preprocessed training data.
4. Compute the RMSE again using the preprocessed testing data. Is the same model as last time still the best-fitting model?
5. Use the best-fitting model (based on RMSE) and predict Sales using three possible ways to split $6000 for the promotions budget. As with the training and testing data, be sure to preprocess the new data. What should the client do with their promotional budget? Provide an interpretation using dollars.
6. Render the Quarto document into Word and upload to Canvas.

**Five points total, one point each for:**

* **Visualizing right-skewed continuous variables (outcome and explanatory variables).**
* **Preparing a recipe using step\_log() and fitting the models again using the preprocessed training data.**
* **Computing the RMSE using the preprocessed testing data and answering the question about how the best-fitting model compares to the previous exercise.**
* **Identifying the best-fitting model and using to predict sales on preprocessed new data and answering the question about using the hypothesized promotional budget in dollars.**
* **One point for submitting a rendered Word document.**

## Check for Right-Skewed Continuous Variables

Let’s load the packages we’ll need, import the data, and evaluate possible right-skewed continuous variables.

# Load packages.  
library(tidyverse)

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.4 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.1  
✔ ggplot2 3.4.4 ✔ tibble 3.2.1  
✔ lubridate 1.9.3 ✔ tidyr 1.3.0  
✔ purrr 1.0.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

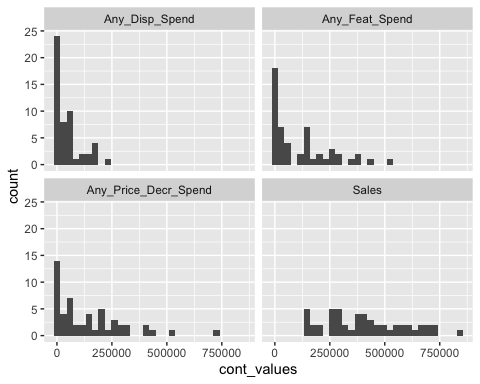
── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──  
✔ broom 1.0.5 ✔ rsample 1.2.0  
✔ dials 1.2.0 ✔ tune 1.1.2  
✔ infer 1.0.5 ✔ workflows 1.1.3  
✔ modeldata 1.2.0 ✔ workflowsets 1.0.1  
✔ parsnip 1.1.1 ✔ yardstick 1.2.0  
✔ recipes 1.0.9   
── Conflicts ───────────────────────────────────────── tidymodels\_conflicts() ──  
✖ scales::discard() masks purrr::discard()  
✖ dplyr::filter() masks stats::filter()  
✖ recipes::fixed() masks stringr::fixed()  
✖ dplyr::lag() masks stats::lag()  
✖ yardstick::spec() masks readr::spec()  
✖ recipes::step() masks stats::step()  
• Learn how to get started at https://www.tidymodels.org/start/

# Import and filter data.  
soup\_data <- read\_csv(here::here("Data", "soup\_data.csv")) |>   
 filter(Retailer\_Trade\_Areas == "WEST CENSUS TA", Brand\_High == "CAMPBELL'S")

Rows: 3192 Columns: 66  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (5): Four\_Weeks\_Periods, Retailer\_Trade\_Areas, Category, Sub\_Category, ...  
dbl (61): Sales, Base\_Sales, Incr\_Sales, Units, Base\_Units, Incr\_Units, Perc...  
  
ℹ Use `spec()` to retrieve the full column specification for this data.  
ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# Visualize the continuous variables.  
soup\_data |>   
 pivot\_longer(  
 c(Sales, Any\_Disp\_Spend, Any\_Feat\_Spend, Any\_Price\_Decr\_Spend),  
 names\_to = "var\_names",  
 values\_to = "cont\_values"  
 ) |>   
 ggplot(aes(x = cont\_values)) +  
 geom\_histogram() +   
 facet\_wrap(~ var\_names)

`stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



Clearly we need to preprocess these variables by applying the log transform.

## Prepare a Recipe

Let’s split the data, prepare a recipe using the training data, and apply it to both the training and testing data.

# Split the data.  
soup\_split <- initial\_time\_split(soup\_data, prop = 0.90)  
  
# Prepare a recipe.  
soup\_recipe <- training(soup\_split) |>   
 recipe(Sales ~ Any\_Disp\_Spend + Any\_Feat\_Spend + Any\_Price\_Decr\_Spend) |>   
 step\_log(all\_numeric(), offset = 1) |>   
 prep()  
  
# Apply the recipe to the training data.  
soup\_training <- soup\_recipe |>  
 bake(training(soup\_split))  
  
# Apply the recipe to the testing data.  
soup\_testing <- soup\_recipe |>  
 bake(testing(soup\_split))

## Fit the Models

Now let’s refit the models we ran previously, now using the preprocessed training data.

# Full model.  
fit\_01 <- linear\_reg() |>   
 set\_engine(engine = "lm") |>   
 fit(  
 Sales ~ Any\_Disp\_Spend + Any\_Feat\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )  
  
# Model without display spend.  
fit\_02 <- linear\_reg() |>   
 set\_engine(engine = "lm") |>   
 fit(  
 Sales ~ Any\_Feat\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )  
  
# Model without feature spend.  
fit\_03 <- linear\_reg() |>   
 set\_engine(engine = "lm") |>   
 fit(  
 Sales ~ Any\_Disp\_Spend + Any\_Price\_Decr\_Spend,   
 data = soup\_training  
 )  
  
# Model without price decrease spend.  
fit\_04 <- linear\_reg() |>   
 set\_engine(engine = "lm") |>   
 fit(  
 Sales ~ Any\_Disp\_Spend + Any\_Feat\_Spend,   
 data = soup\_training  
 )

## Overall Model Fit

Now let’s compute and compare RMSE using the preprocessed testing data.

# Compute RMSE.  
rmse\_01 <- fit\_01 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 rmse(truth = Sales, estimate = .pred)  
  
rmse\_02 <- fit\_02 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 rmse(truth = Sales, estimate = .pred)  
  
rmse\_03 <- fit\_03 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 rmse(truth = Sales, estimate = .pred)  
  
rmse\_04 <- fit\_04 |>   
 predict(new\_data = soup\_testing) |>  
 bind\_cols(soup\_testing) |>  
 rmse(truth = Sales, estimate = .pred)  
  
# Compare RMSEs.  
tibble(  
 model = c(  
 "Full model",   
 "Model without display spend",   
 "Model without feature spend",   
 "Model without price decrease spend"  
 )  
) |>   
 bind\_cols(  
 bind\_rows(  
 rmse\_01,  
 rmse\_02,  
 rmse\_03,  
 rmse\_04  
 )  
 ) |>   
 arrange(.estimate)

# A tibble: 4 × 4  
 model .metric .estimator .estimate  
 <chr> <chr> <chr> <dbl>  
1 Model without price decrease spend rmse standard 0.140  
2 Full model rmse standard 0.350  
3 Model without feature spend rmse standard 0.353  
4 Model without display spend rmse standard 0.469

Based on RMSE, the best-fitting model is now the model without price decrease spend. Without the log transform, the best-fitting model was the full model.

## Predict Sales

We’ll use fit\_04, the best fitting model without price decrease spend, to predict Sales using three possible ways to split 6 thousand dollars for the promotions budget two ways. As with the training and testing data, we will apply the recipe to preprocess the new data.

# Column names and preprocessing needs to match the fitted model.  
scenarios <- tibble(  
 Any\_Disp\_Spend = seq(from = 0, to = 6000, by = 3000),  
 Any\_Feat\_Spend = seq(from = 6000, to = 0, by = -3000),  
 Any\_Price\_Decr\_Spend = 1,  
 Sales = 1  
)  
  
# Apply the recipe to the training data.  
scenarios <- soup\_recipe |>  
 bake(scenarios) |>   
 select(-c(Sales, Any\_Price\_Decr\_Spend))  
  
# Predict and bind on prediction intervals.  
bind\_cols(  
 predict(fit\_04, new\_data = scenarios),  
 predict(fit\_04, new\_data = scenarios, type = "pred\_int"),  
 scenarios  
) |>   
 arrange(desc(.pred))

# A tibble: 3 × 5  
 .pred .pred\_lower .pred\_upper Any\_Disp\_Spend Any\_Feat\_Spend  
 <dbl> <dbl> <dbl> <dbl> <dbl>  
1 12.8 12.2 13.3 8.01 8.01  
2 12.5 11.9 13.0 8.70 0   
3 12.3 11.6 12.9 0 8.70

The client should split the promotional budget between display spend and feature spend. By splitting the $6000, they can expect to make, on average, $344,913 (exp(12.75105) = 344913.90) in the category.