Lab 2

Briana Barajas

2024-01-24

Today we will be continuing the pumpkin case study from last week. We will be using the data that you cleaned and split last time (`pumpkins_train`) and will be comparing our results today to those you have already obtained. Open and run your Lab 1.Rmd as a first step so those objects are available in your Environment.

Once you have done that, we'll start today's lab by specifying a recipe for a polynomial model. First we specify a recipe that identifies our variables and data, converts the package variable to a numerical form, and then adds a polynomial effect with `step_poly()`

```
# Specify a recipe
poly_pumpkins_recipe <-
  recipe(price ~ package, data = pumpkins_train) %>%
  step_integer(all_predictors(), zero_based = TRUE) %>%
  step_poly(all_predictors(), degree = 4)
```

How did that work? Later we will learn about model tuning that will let us do things like find the optimal value for degree. For now, we'd like to have a flexible model, so we'll use a relatively large value.

Polynomial regression is still linear regression, so our model specification looks similar to before.

```
# Create a model specification called poly_spec
poly_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
```

Question 1: Now take the recipe and model specification that just created and bundle them into a workflow called `poly_wf`.

```
# Bundle recipe and model spec into a workflow
poly_wf <- workflow() %>%
  add_recipe(poly_pumpkins_recipe) %>%
  add_model(poly_spec)
```

Question 2: fit a model to the `pumpkins_train` data using your workflow and assign it to `poly_wf_fit`

```
# Create a model
poly_wf_fit <- poly_wf %>%
  fit(data = pumpkins_train)

# Print learned model coefficients
poly_wf_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_integer()
## * step_poly()
##
## -- Model ----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
##    (Intercept)  package_poly_1  package_poly_2  package_poly_3  package_poly_4
##        27.9706        103.8566       -110.9068        -62.6442          0.2677
```

```
# Make price predictions on test data
poly_results <- poly_wf_fit %>% predict(new_data = pumpkins_test) %>%
  bind_cols(pumpkins_test %>% select(c(package, price))) %>%
  relocate(.pred, .after = last_col())

# Print the results
poly_results %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 3
##    package price .pred
##      <int> <dbl> <dbl>
##  1       0  13.6  15.9
##  2       0  16.4  15.9
##  3       0  16.4  15.9
##  4       0  13.6  15.9
##  5       0  15.5  15.9
##  6       0  16.4  15.9
##  7       2  34    34.4
##  8       2  30    34.4
##  9       2  30    34.4
## 10       2  34    34.4
```

Now let's evaluate how the model performed on the test_set using `yardstick::metrics()`.

```
metrics(data = poly_results, truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.27
## 2 rsq     standard       0.892
## 3 mae     standard       2.35
```

Question 3: How do the performance metrics differ between the linear model from last week and the polynomial model we fit today? Which model performs better on predicting the price of different packages of pumpkins?

ANS: The polynomial model created here has a smaller rmse, meaning this model is better is the price is off by an average of $3.27, compared to the average of $7.22 for the linear model. The r-squared value supports this, because the polynomial model has a larger r-squared value ($0.89 > 0.49$) meaning it's better at explaining the variance in price than the linear model.

Let's visualize our model results. First prep the results by binding the encoded package variable to them.

```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


# Print new results data frame
poly_results %>%
  slice_head(n = 5)
```

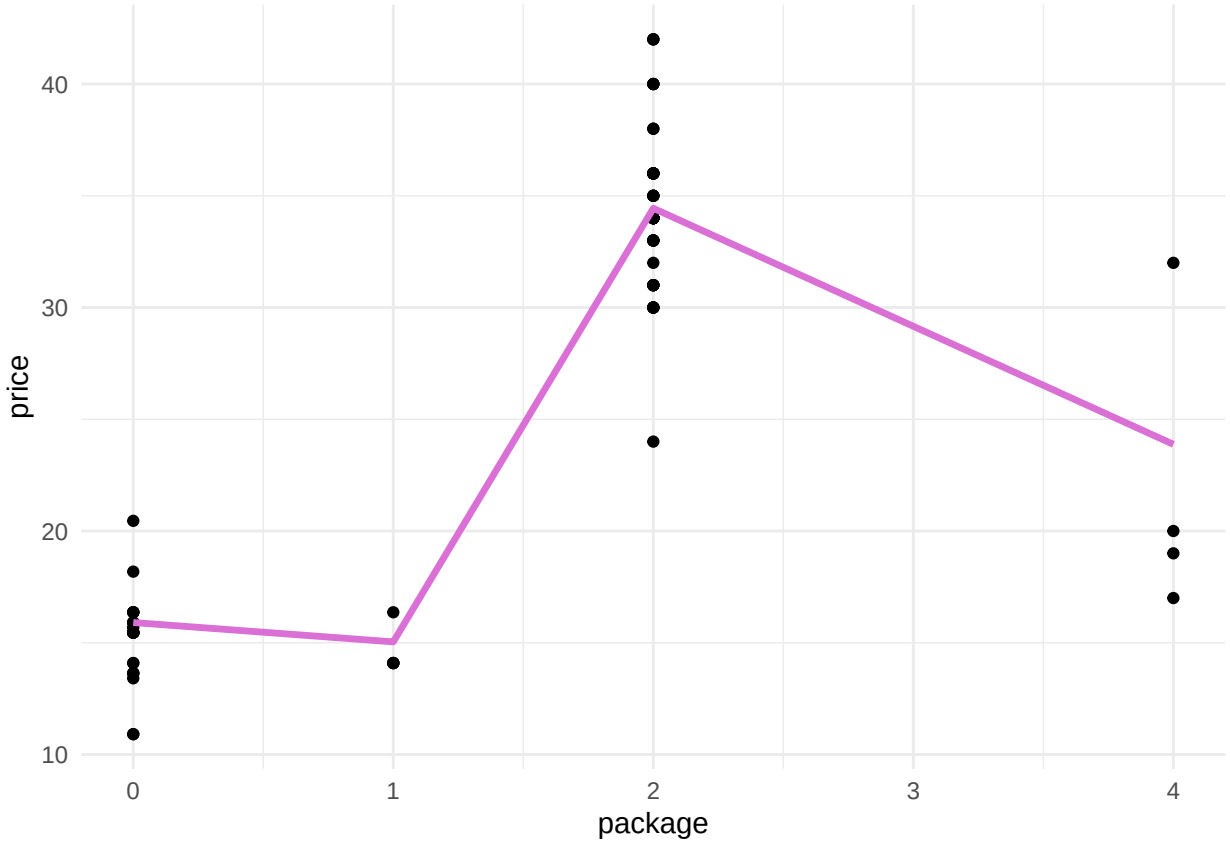
```
## # A tibble: 5 x 4
##   package package_integer price .pred
##     <int>           <int> <dbl> <dbl>
## 1       0               0  13.6  15.9
## 2       0               0  16.4  15.9
## 3       0               0  16.4  15.9
## 4       0               0  13.6  15.9
## 5       0               0  15.5  15.9
```

OK, now let's take a look!

Question 4: Create a scatter plot that takes the poly_results and plots package vs. price. Then draw a line showing our model's predicted values (.pred). Hint: you'll need separate geoms for the data points and the prediction line.

```
# Make a scatter plot
poly_results %>% ggplot(aes(x = package_integer, y = price)) +
  geom_point() +
  geom_line(aes(y = .pred), color = 'orchid', size = 1.2) +
  labs(x = 'package')
```

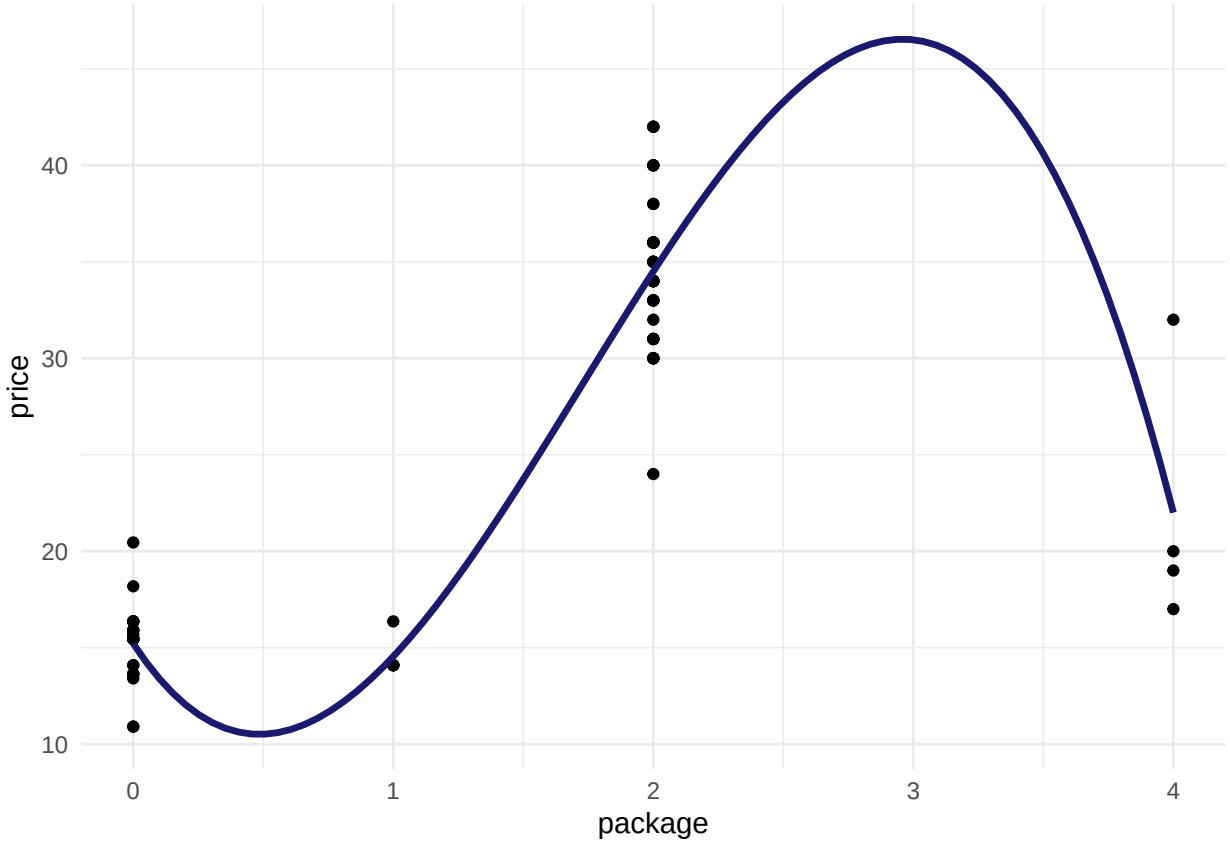
```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



You can see that a curved line fits your data much better.

Question 5: Now make a smoother line by using geom_smooth instead of geom_line and passing it a polynomial formula like this: geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "midnightblue", size = 1.2, se = FALSE)

```
# Make a smoother scatter plot
poly_results %>% ggplot(aes(x = package_integer, y = price)) +
  geom_point() +
  geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "midnightblue", size = 1.2, se = F
  labs(x = 'package')
```



OK, now it's your turn to go through the process one more time.

Additional assignment components :

6. Choose a new predictor variable (anything not involving package type) in this dataset.

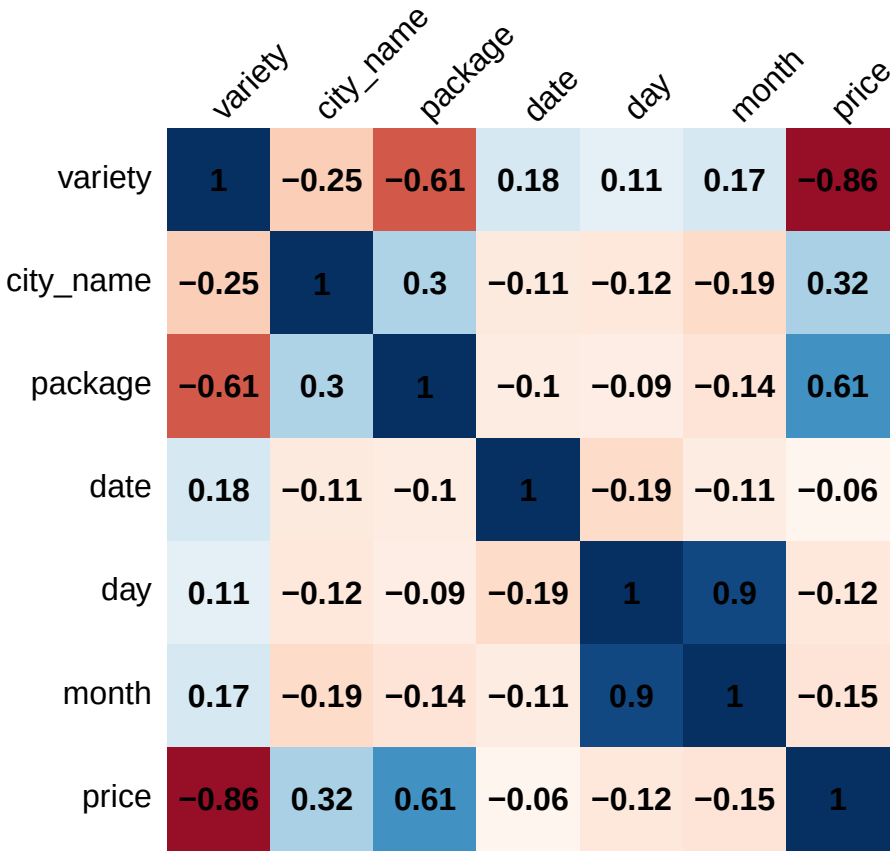
ANS: I decided to choose `variety` as the new predictor variable, and compare it to `price` in the `baked_pumpkins` data frame, which is normalized to have price per individual pumpkin and categorical variables are numeric.

7. Determine its correlation with the outcome variable (price). (Remember we calculated a correlation matrix last week)

```
# calculate correlation between price and variety
glue::glue("Correlation between individual pumpkin price and variety: {cor(baked_pumpkins$variety, bake
```

```
## Correlation between individual pumpkin price and variety: -0.863479040021441
```

```
# create correlation matrix
corr_matrix <- cor(baked_pumpkins %>%
                   select(-c(low_price, high_price)))

# Make a correlation plot between the variables
corrplot(corr_matrix, method = "shade", shade.col = NA,
         tl.col = "black", tl.srt = 45, addCoef.col = "black",
         cl.pos = "n", order = "original")
```



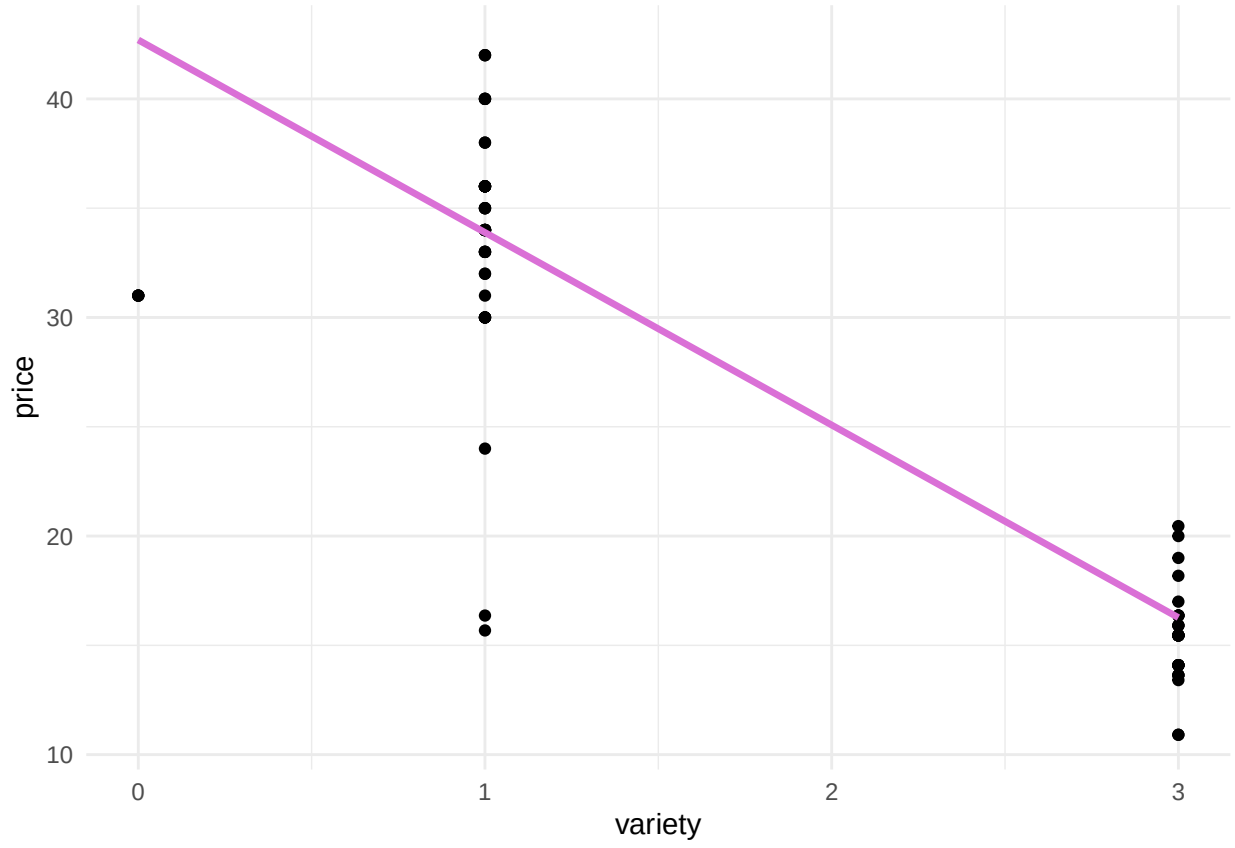
8. Create and test a model for your new predictor: - Create a recipe - Build a model specification (linear or polynomial) - Bundle the recipe and model specification into a workflow - Create a model by fitting the workflow - Evaluate model performance on the test data - Create a visualization of model performance

```
## ============== Create and Run Model ==============
# Create a recipe for preprocessing the data
lm_variety_recipe <- recipe(price ~ variety, data = pumpkins_train) %>%
  step_integer(all_predictors(), zero_based = TRUE)

# Create a linear model specification
lm_variety_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# add recipe and model specifications into a workflow
lm_variety_wf <- workflow() %>%
  add_recipe(lm_variety_recipe) %>%
  add_model(lm_variety_spec)

# create model
variety_lm_wf_fit <- lm_variety_wf %>%
  fit(data = pumpkins_train)

# predict price using test data
variety_lm_results <- variety_lm_wf_fit %>%
  predict(new_data = pumpkins_test) %>%
```

```
  bind_cols(pumpkins_test %>%
              select(c(variety, price))) %>%
  relocate(.pred, .after = last_col())

## ============== Model Assessment ==============
# view and analyze model metrics
metrics(data = variety_lm_results, truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       4.86
## 2 rsq     standard       0.766
## 3 mae     standard       3.11
```

```
# plot model
variety_lm_results %>%
  ggplot(aes(x = variety, y = price)) +
  geom_point() +
  geom_line(aes(y = .pred), color = 'orchid', size = 1.2) +
  labs(x = 'variety')
```



ANS: Based on the metrics, the model explains 76.6% variation on price, which is relatively high for just one variable. However, the plot demonstrates that the data points do not follow the suggested trend line. This result is not surprising as it replicates the linear model for package versus price, except the line now

has a negative slope. The negative slope can be expected since the correlation between variety and price was a negative value (-0.86)

Lab 2 due 1/24 at 11:59 PM