Muir_Lab2

Sam Muir

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")
```

- define -> fit -> predict -> evaluate

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

```
# 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(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?

- The RSME for this model is 3.27 while the RSME for the previous model was 7.23, indicating that this model is a better fit since the RSME and MAE are both smaller. The R-square value for this new model (0.892) is also larger than the previous model's R-square (0.495), also indicating that the the fit is better.

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

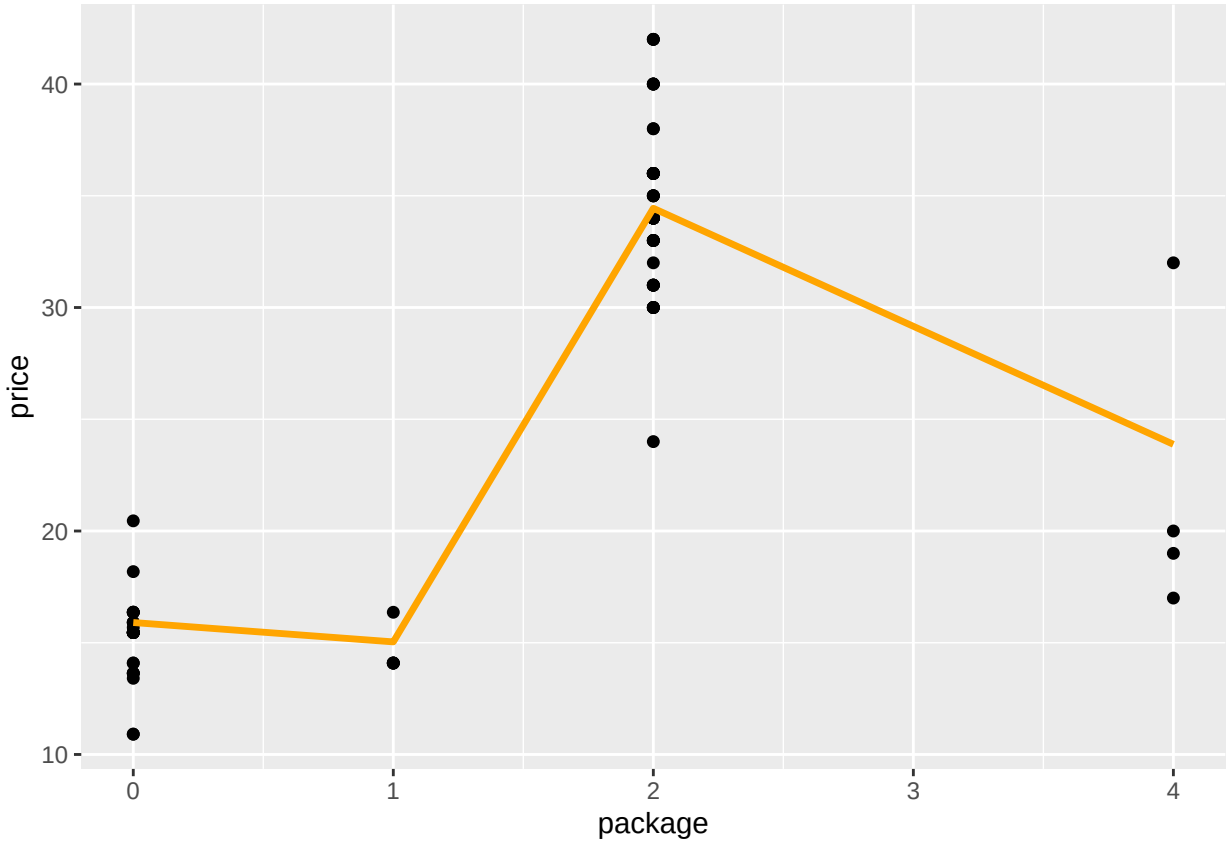
```r
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


# 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.

```r
# Make a scatter plot
poly_results %>%
  ggplot(mapping = aes(x = package_integer, y = price)) +
   geom_point(size = 1.6) +
   geom_line(aes(y = .pred), color = "orange", linewidth = 1.2) +
   xlab("package")
```

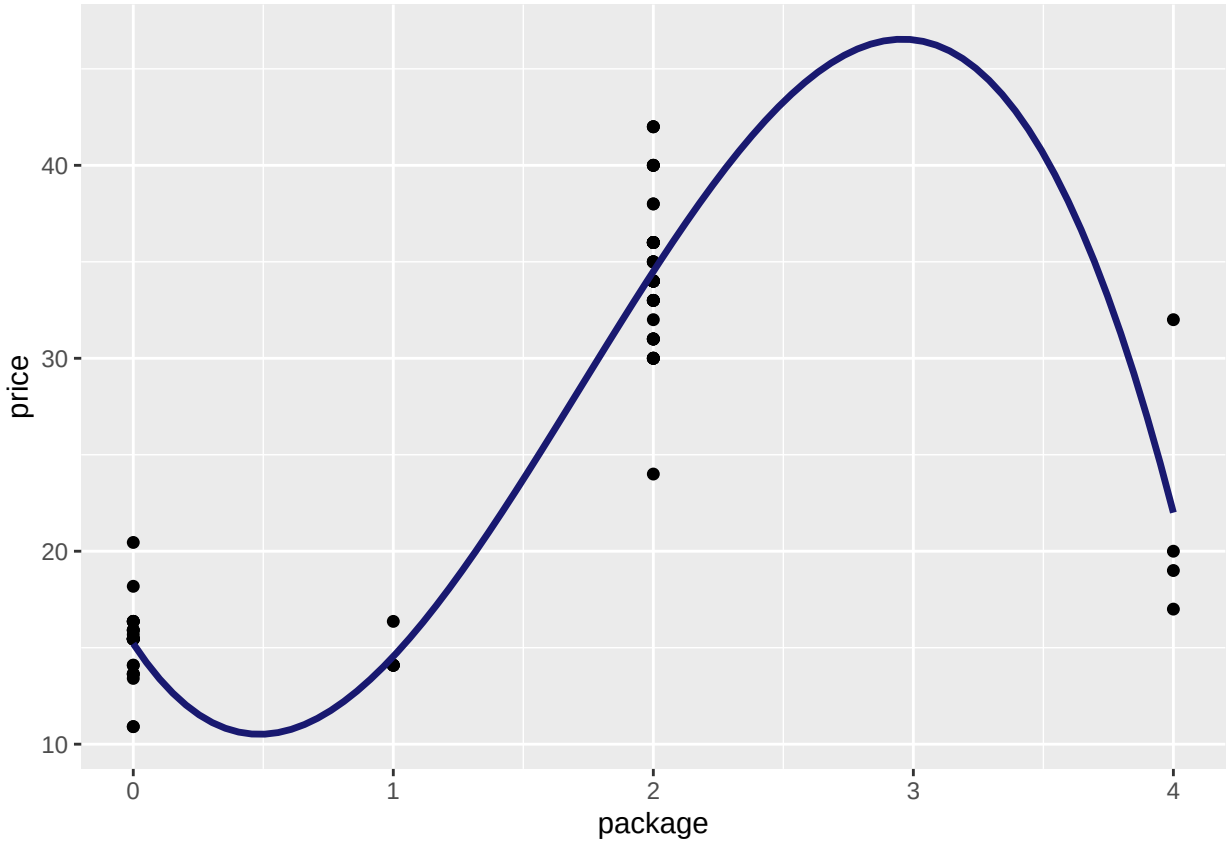


You can see that a curved line fits your data much better.

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(mapping = aes(x = package_integer, y = price)) +
   geom_point(size = 1.6) +
   geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "midnightblue", size = 1.2, se =
   xlab("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.
```



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

Answers

6. New predictor variable will be city name.

7.

```
cor(baked_pumpkins$city_name, baked_pumpkins$price)
```

```
## [1] 0.3236397
```

The correlation between city name and price is 0.3236397.

8.

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

# Create a model specification
poly_spec <- linear_reg() %>%
```

```
  set_engine("lm") %>%
  set_mode("regression")

# Bundle recipe and model spec into a workflow
poly_wf_city <- workflow() %>%
  add_recipe(poly_pumpkins_recipe_city) %>%
  add_model(poly_spec)

# Create a model
poly_wf_fit_city <- poly_wf_city %>%
  fit(pumpkins_train)

poly_wf_fit_city
```

```
## == 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)  city_name_poly_1  city_name_poly_2  city_name_poly_3
##           27.971            57.152             2.875            -2.096
## city_name_poly_4
##          -34.769
```

```
# Make price predictions on test data
poly_results_city <- poly_wf_fit_city %>%
  predict(new_data = pumpkins_test) %>%
  bind_cols(pumpkins_test %>% select(city_name, price)) %>%
  relocate(.pred, .after = last_col())

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

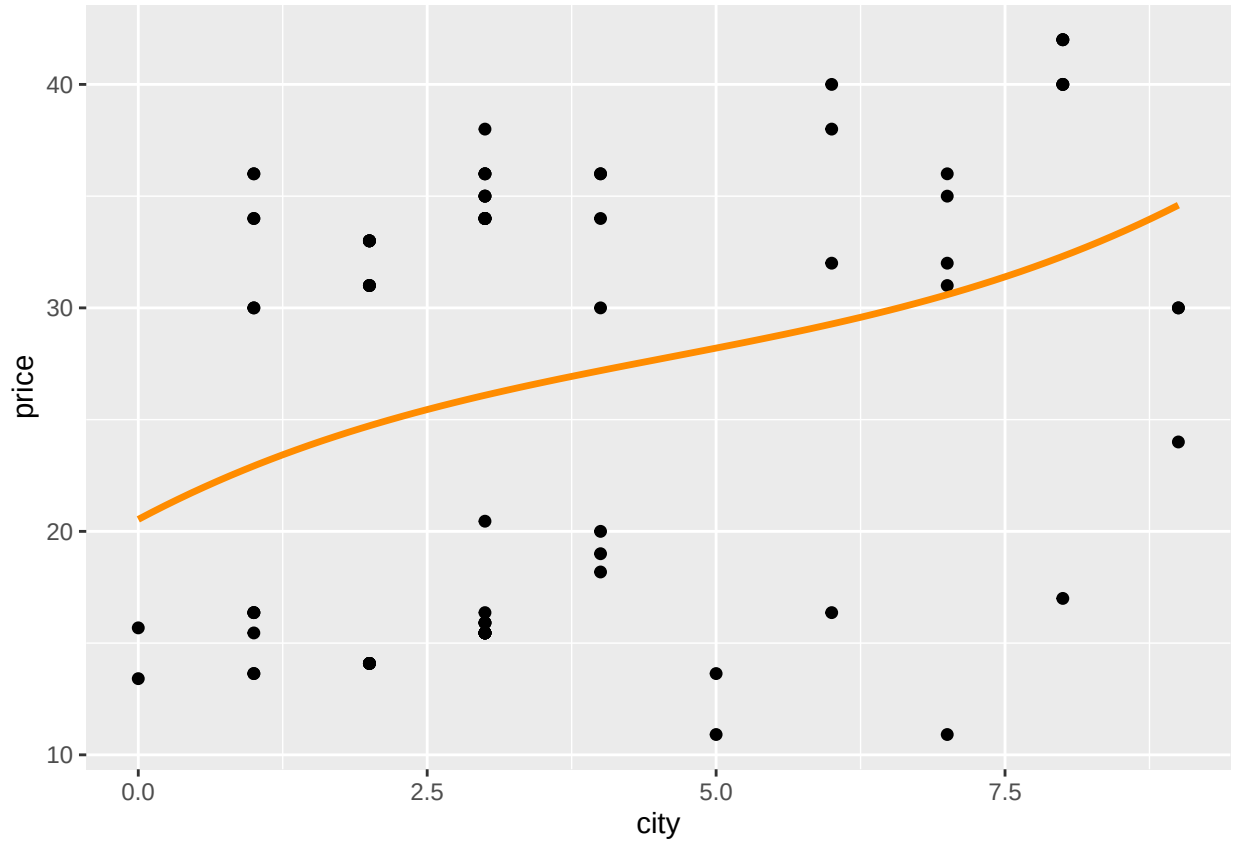
```
## # A tibble: 10 x 3
##    city_name price .pred
##        <int> <dbl> <dbl>
##  1         1  13.6  24.3
##  2         1  16.4  24.3
##  3         1  16.4  24.3
##  4         1  13.6  24.3
##  5         1  15.5  24.3
##  6         1  16.4  24.3
```

```
##  7         1  34    24.3
##  8         1  30    24.3
##  9         1  30    24.3
## 10         1  34    24.3
```

```
metrics(data = poly_results_city, truth = city_name, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      23.9
## 2 rsq     standard       0.710
## 3 mae     standard      23.8
```

```
# Make a smoother scatter plot
poly_results_city %>%
  ggplot(mapping = aes(x = city_name, y = price)) +
   geom_point(size = 1.6) +
   geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "darkorange", size = 1.2, se = F
   xlab("city")
```



Lab 2 due 1/24 at 11:59 PM