Herbst_Lab2

Amanda Herbst

01/18/2024

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 <-
  # identify variables and data
  recipe(price ~ package, data = pumpkins_train) %>%
  # convert package variable to numeric form
  step_integer(all_predictors(), zero_based = TRUE) %>%
  # add polynomial effect
  step_poly(all_predictors(), degree = 3)
```

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_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 <- fit(data = pumpkins_train, object = poly_wf)
```

```
# 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
##          27.97          103.86         -110.91          -62.64
```

```
# 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
##    <chr>                <dbl> <dbl>
##  1 1 1/9 bushel cartons  13.6  15.9
##  2 1 1/9 bushel cartons  16.4  15.9
##  3 1 1/9 bushel cartons  16.4  15.9
##  4 1 1/9 bushel cartons  13.6  15.9
##  5 1 1/9 bushel cartons  15.5  15.9
##  6 1 1/9 bushel cartons  16.4  15.9
##  7 1/2 bushel cartons    34    34.4
##  8 1/2 bushel cartons    30    34.4
##  9 1/2 bushel cartons    30    34.4
## 10 1/2 bushel cartons    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       3.28
## 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 RMSE and MAE from last week are about twice as large as the those from the polynomial model. Additionally, the R squared for the polynomial model is almost twice as large as the linear model from last week. Therefore, since the RMSE and MAE are smaller and the R squared is larger, the polynomial model peforms better on predicting the price of different

packages of pumpkins.

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

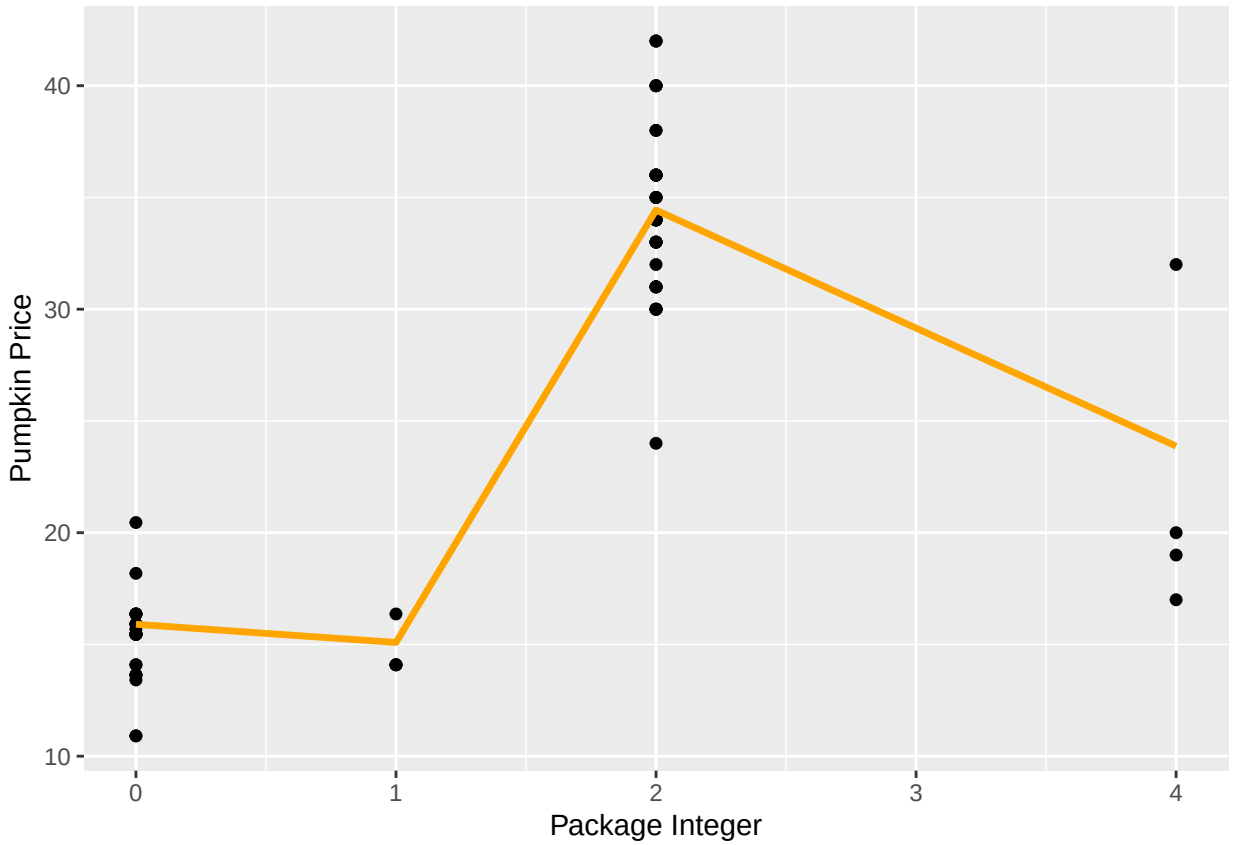
```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


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

```
## # A tibble: 5 x 4
##   package              package_integer price .pred
##   <chr>                          <int> <dbl> <dbl>
## 1 1 1/9 bushel cartons               0  13.6  15.9
## 2 1 1/9 bushel cartons               0  16.4  15.9
## 3 1 1/9 bushel cartons               0  16.4  15.9
## 4 1 1/9 bushel cartons               0  13.6  15.9
## 5 1 1/9 bushel cartons               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.

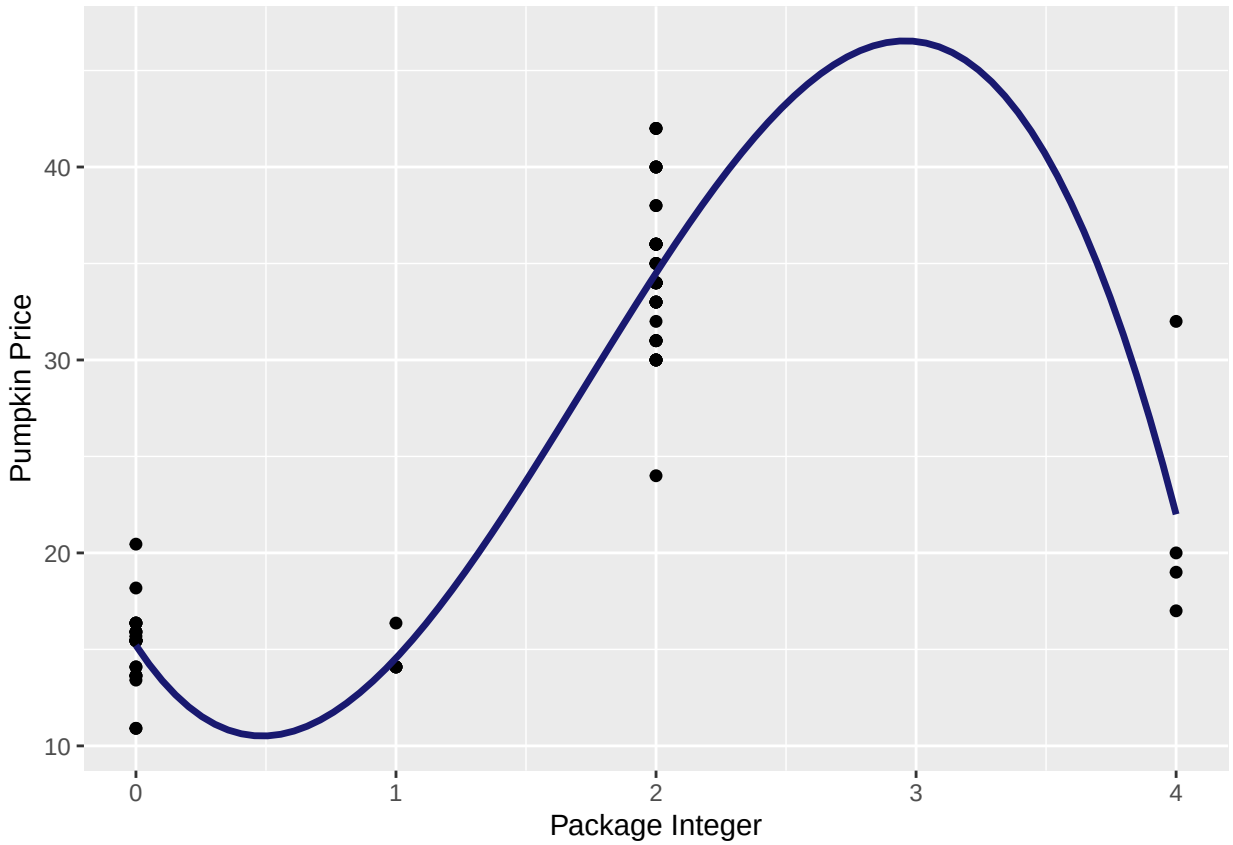
```
# Make a scatter plot
poly_results %>%
  ggplot(mapping = aes(x = package_integer, y = price)) +
  geom_point(size = 1.6) +
  geom_line(aes(y = .pred), color = "orange", linewidth = 1.2) +
  labs(x = "Package Integer",
       y = "Pumpkin Price")
```



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",
              linewidth = 1.2,
              se = FALSE) +
  labs(x = "Package Integer",
       y = "Pumpkin Price")
```



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.

City Name

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

```
#Correlation between price and city.
cor(baked_pumpkins$city_name, baked_pumpkins$price)
```

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

8. Create and test a model for your new predictor:

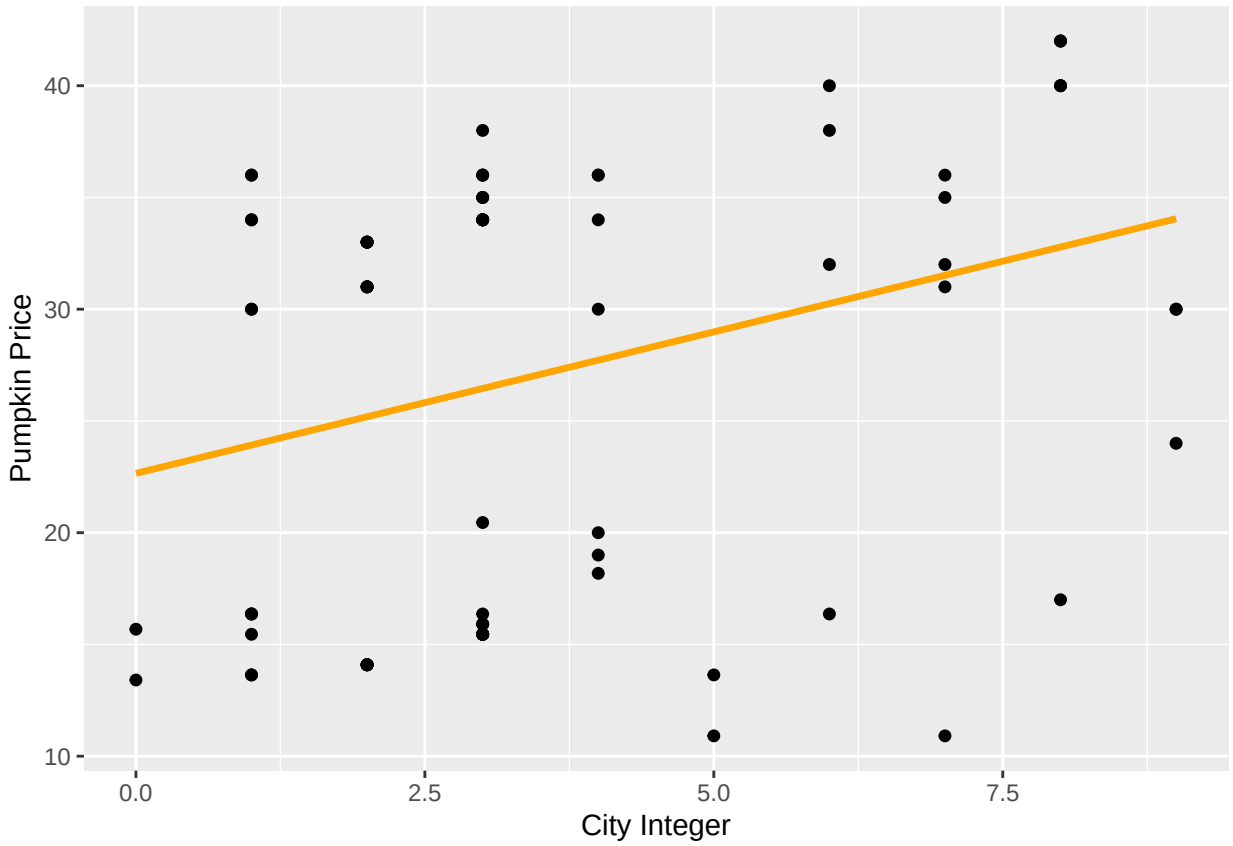
- Create a recipe
- Build a model specification (linear or polynomial)
- Bundle the recipe and model specification into a workflow
- Create a model by fitting the workflow
- Evaluate model performance on the test data
- Create a visualization of model performance

```
# Create a recipe for preprocessing the data
city_pumpkins_recipe <- recipe(price ~ city_name, data = pumpkins_train) %>%
  step_integer(all_predictors(), zero_based = TRUE)

# build polynomial model specification
city_spec <- linear_reg() %>%
  set_engine("lm") %>%
```

```
  set_mode("regression")

# bundle recipe and model specification into workflow
city_wf <- workflow() %>%
  add_recipe(city_pumpkins_recipe) %>%
  add_model(city_spec)

# create model by fitting the workflow
city_wf_fit <- fit(data = pumpkins_train, object = city_wf)

# evaluate model perfomance on test data
city_results <- city_wf_fit %>% predict(new_data = pumpkins_test) %>%
  bind_cols(pumpkins_test %>% select(c(city_name, price))) %>%
  relocate(.pred, .after = last_col())

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

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       9.35
## 2 rsq     standard       0.102
## 3 mae     standard       8.76
```

```
# create visualization of model performance
# Encode city names column
city_encode <- city_pumpkins_recipe %>%
  prep() %>%
  bake(new_data = pumpkins_test) %>%
  select(city_name)

# Bind encoded city name column to the results
city_results <- city_results %>%
  bind_cols(city_encode %>%
              rename(city_integer = city_name)) %>%
  relocate(city_integer, .after = city_name)

# Make a scatter plot
city_results %>%
  ggplot(mapping = aes(x = city_integer, y = price)) +
   geom_point(size = 1.6) +
   # Overlay a line of best fit
   geom_line(aes(y = .pred), color = "orange", linewidth = 1.2) +
   labs(x = "City Integer",
        y = "Pumpkin Price")
```



Lab 2 due 1/24 at 11:59 PM