Lab2

Luna Herschenfeld-Catalán

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, # y ~ x variables
         data = pumpkins_train) %>% # specify data
  step_integer(all_predictors(), zero_based = TRUE) %>% # data to numerical form
  step_poly(all_predictors(), degree = 4) # add polynomial effect
```

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() %>% # specify type of model
  set_engine("lm") %>% # set to linear regression
  set_mode("regression") # define as 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 <- poly_wf %>%
  fit(data = pumpkins_train) # use training data
```

```
# 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) %>% # make predictions using test data
  bind_cols(pumpkins_test %>%
              select(c(package, price))) %>% # add predictions column to the df
  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 performance metrics for the polynomial model are more accurate than the metrics for the linear model. The .estimate values for the polynomial model are smaller than the values for the linear model. The polynomial model therefore performs better because it is predicting the values to a more accurate degree.

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

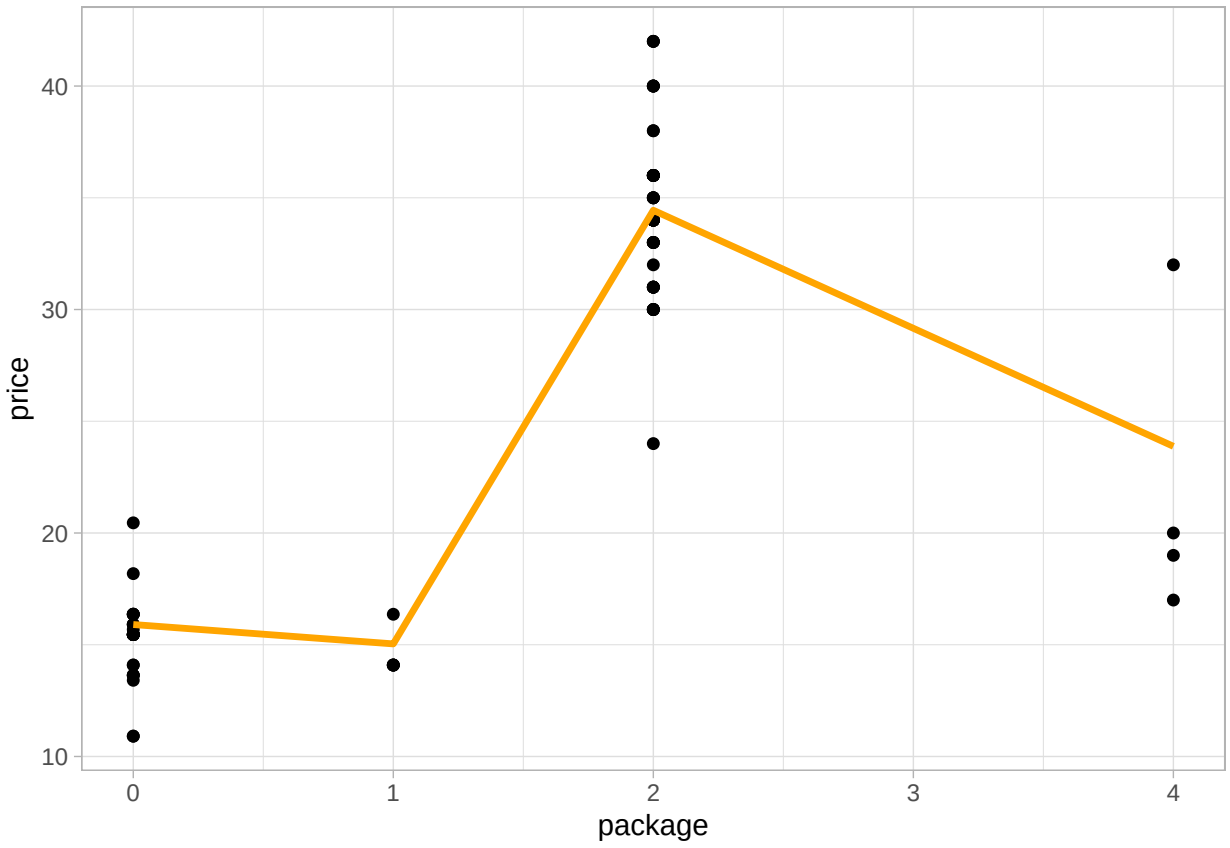
```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


# 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.

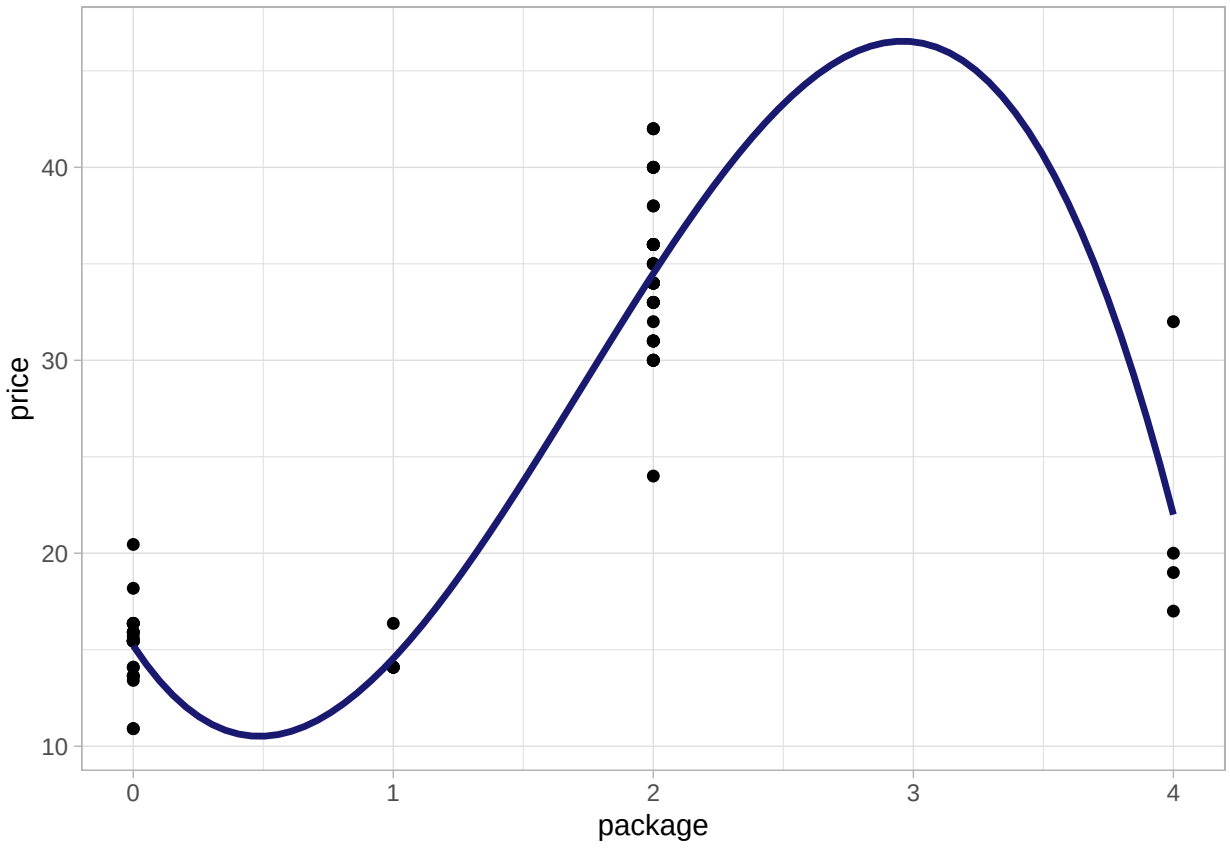
```
# Make a scatter plot
poly_results %>%
  ggplot(mapping = aes(x = package_integer,
                       y = price)) +
  geom_point(size = 1.6) + # plot the points
  geom_line(aes(y = .pred), # Overlay a line of best fit using predicted values
            color = "orange",
            linewidth = 1.2) +
  xlab("package") +
  theme_light()
```



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)) +

  # plot points
  geom_point(size = 1.6) +

  # Overlay a line of best fit
  geom_smooth(method = lm, # line model
              formula = y ~ poly(x, degree = 3), # specify polynomial line
              color = "midnightblue",
              linewidth = 1.2,
              se = FALSE) + # dont include the standard error
  xlab("package") +
  theme_light()
```



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.

I am selecting the `city_name` variable to determine how well it is the predictor variable.

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

```
#Correlation between price and other vars
corr <- cor(baked_pumpkins$city_name, # predictor
          baked_pumpkins$price) # outcome
```

The correlation between `city_name` and `price` is 0.324.

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

```
# Specify a recipe with polynomial effect
city_pumpkins_recipe <-
  recipe(price ~ city_name, # y ~ x variables
         data = pumpkins_train) %>% # specify data
  step_integer(all_predictors(), zero_based = TRUE) %>% # data to numerical form
  step_poly(all_predictors(), degree = 4) # add polynomial effect

# Build a linear model specification
city_spec <- linear_reg() %>% # specify type of model
  set_engine("lm") %>% # set to linear regression
  set_mode("regression") # define as regression


# Bundle recipe and model spec into a workflow
city_wf <- workflow() %>%
  add_recipe(city_pumpkins_recipe) %>%
  add_model(city_spec)

# Create a model by fitting the workflow
city_wf_fit <- city_wf %>%
  fit(data = pumpkins_train) # use training data
```

```
# Evaluate model performance on the 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, # test data
        truth = price, # predicted variable
        estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       9.15
## 2 rsq     standard       0.142
## 3 mae     standard       8.26
```

The model performance on the test data is pretty poor. The .estimate values are high numbers compared to the price values in the dataset we are using. This makes sense once we look at the graph because there is a clear pattern of very low values and very high values for `price` at each `city_name` value. However, despite the slight possible upward trend seen in the graphs below, fitting a polynomial regression to the data does not fit the data points very well. Considering the low correlation value between `city_name` and `price`, there may be two problems with this model: (1) the variable chosen is not a good predictor of our variable of interest and (2) the model chosen does not fit the data well.

```
# Create a visualization of model performance

# Make a scatter plot
city_plot <- city_results %>%
  ggplot(mapping = aes(x = city_name, y = price)) +
```

```
  geom_point(size = 1.6) +
  # Overlay a line of best fit
  geom_line(aes(y = .pred),
            color = "orange",
            linewidth = 1.2) +
  xlab("City Name (numerical)") +
  ylab("Price") +
  theme_light()

# Make a smoother scatter plot
city_poly_smooth <- city_results %>%
  ggplot(mapping = aes(x = city_name, y = price)) +
  geom_point(size = 1.6) +
  # Overlay a line of best fit
  geom_smooth(method = lm, # line model
              formula = y ~ poly(x, degree = 4), # specify polynomial line
              color = "midnightblue",
              linewidth = 1.2,
              se = FALSE) +
  xlab("City Name (numerical)") +
  ylab("Price") +
  theme_light()

city_plot + city_poly_smooth
```