Bishop_Lab2

Erica Bishop

2023-01-18

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, so open and run your Lab 1 .Rmd as a first step so those objects are available in your Environment (unless you created an R Project last time, in which case, kudos to you!).

```r
#run lab 1 code into environment

sourceDir <- "/Users/ericabishop/Documents/MEDSwinter/EDS232-ml/labs/Lab1.Rmd"
library(knitr)
source(knitr::purl(sourceDir, quiet=TRUE))
```

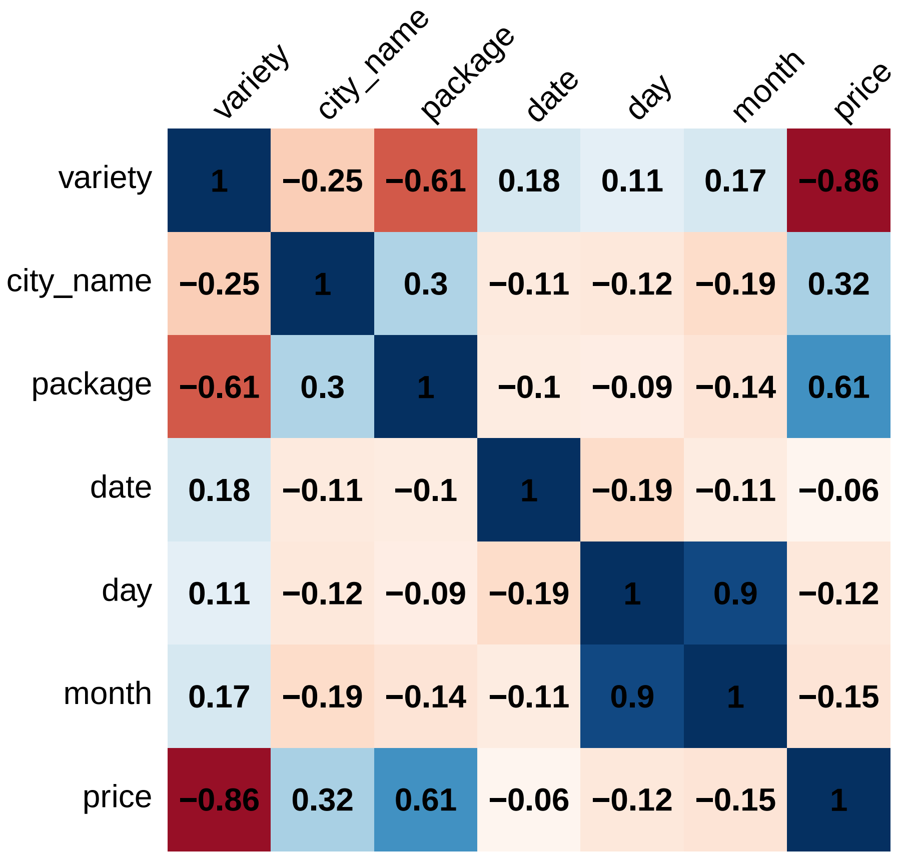
```
## -- Attaching packages -------------------------------------- tidymodels 1.0.0 --

## v broom        1.0.0     v recipes      1.0.1
## v dials        1.0.0     v rsample      1.1.0
## v dplyr        1.0.9     v tibble       3.1.8
## v ggplot2      3.3.6     v tidyr        1.2.0
## v infer        1.0.3     v tune         1.0.0
## v modeldata    1.0.1     v workflows    1.0.0
## v parsnip      1.0.1     v workflowsets 1.0.0
## v purrr        0.3.4     v yardstick    1.1.0

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v readr   2.1.2     v forcats 0.5.1
## v stringr 1.4.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x readr::spec()       masks yardstick::spec()
##
## Attaching package: 'janitor'
##
##
## The following objects are masked from 'package:stats':
##
```

```
##     chisq.test, fisher.test
##
##
## corrplot 0.92 loaded
##
## New names:
## Rows: 1757 Columns: 27
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (13): City Name, Type, Package, Variety, Sub Variety, Date, Origin, Orig...
## dbl  (5): ...1, Low Price, High Price, Mostly Low, Mostly High
## lgl  (9): Grade, Environment, Quality, Condition, Appearance, Storage, Crop,...
##
## 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.
```

```
## Rows: 1,757
## Columns: 27
## $ ...1              <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
## $ `City Name`       <chr> "BALTIMORE", "BALTIMORE", "BALTIMORE", "BALTIMORE", ~
## $ Type              <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Package           <chr> "24 inch bins", "24 inch bins", "24 inch bins", "24 ~
## $ Variety           <chr> NA, NA, "HOWDEN TYPE", "HOWDEN TYPE", "HOWDEN TYPE",~
## $ `Sub Variety`     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Grade             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Date              <chr> "4/29/17", "5/6/17", "9/24/16", "9/24/16", "11/5/16"~
## $ `Low Price`       <dbl> 270, 270, 160, 160, 90, 90, 160, 160, 160, 160, 160,~
## $ `High Price`      <dbl> 280, 280, 160, 160, 100, 100, 170, 160, 170, 160, 17~
## $ `Mostly Low`      <dbl> 270, 270, 160, 160, 90, 90, 160, 160, 160, 160, 160,~
## $ `Mostly High`     <dbl> 280, 280, 160, 160, 100, 100, 170, 160, 170, 160, 17~
## $ Origin            <chr> "MARYLAND", "MARYLAND", "DELAWARE", "VIRGINIA", "MAR~
## $ `Origin District` <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ `Item Size`       <chr> "lge", "lge", "med", "med", "lge", "lge", "med", "lg~
## $ Color             <chr> NA, NA, "ORANGE", "ORANGE", "ORANGE", "ORANGE", "ORA~
## $ Environment       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ `Unit of Sale`    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Quality           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Condition         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Appearance        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Storage           <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Crop              <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Repack            <chr> "E", "E", "N", "N", "N", "N", "N", "N", "N", "N", "N~
## $ `Trans Mode`      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ ...26             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ ...27             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
```

```
##
## Attaching package: 'lubridate'
##
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
## [1] "There is a 0.606 correlation between pumpkin package and price."
## [1] "There is a 0.324 correlation between pumpkin price and the city."
## [1] "There is a -0.863 correlation between pumpkin price and variety."
```



	variety	city_name	package	date	day	month	price
variety	1	−0.25	−0.61	0.18	0.11	0.17	−0.86
city_name	−0.25	1	0.3	−0.11	−0.12	−0.19	0.32
package	−0.61	0.3	1	−0.1	−0.09	−0.14	0.61
date	0.18	−0.11	−0.1	1	−0.19	−0.11	−0.06
day	0.11	−0.12	−0.09	−0.19	1	0.9	−0.12
month	0.17	−0.19	−0.14	−0.11	0.9	1	−0.15
price	−0.86	0.32	0.61	−0.06	−0.12	−0.15	1

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 package to a numerical form, and then add 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? Choose another value for degree if you need to. 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 find the highest value for degree that is consistent with our data.

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 <- 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
##    <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().

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

print(poly_metrics)
```

```
## # 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
```

```
#compare to the results from the linear model
lm_metrics <- metrics(data = lm_results, truth = price, estimate = .pred)
print(lm_metrics)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       7.23
## 2 rsq     standard       0.495
## 3 mae     standard       5.94
```

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 polynomial model fits the data better, with a much higher R squared value (0.89 compared to 0.49). The root mean square error is also much smaller for the polynomial model, meaning there is less variance in the errors. The mean absolute errors (MAE) is also smaller for the polynomial model than the linear model.

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

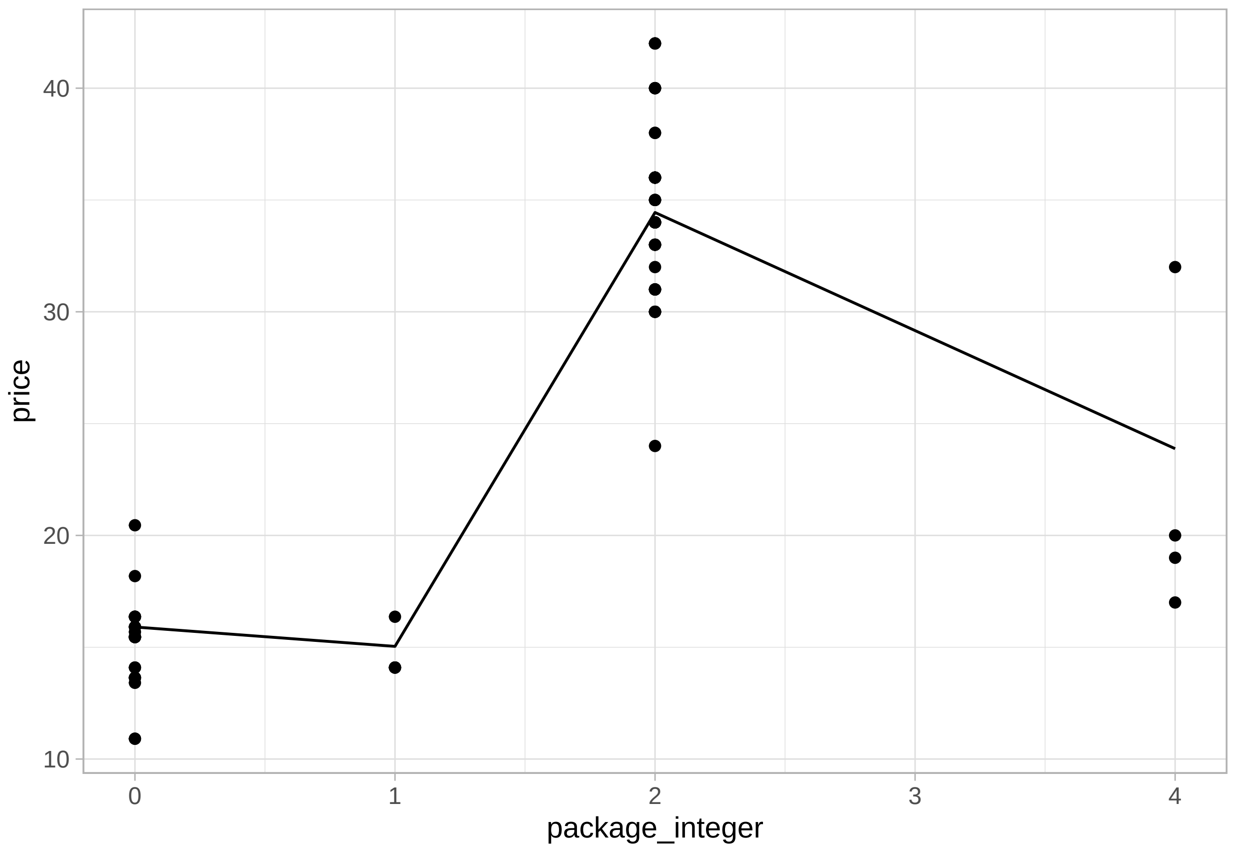
```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


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

```
## # A tibble: 5 x 4
##   package              package_integer price .pred
##   <chr>                          <dbl> <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.

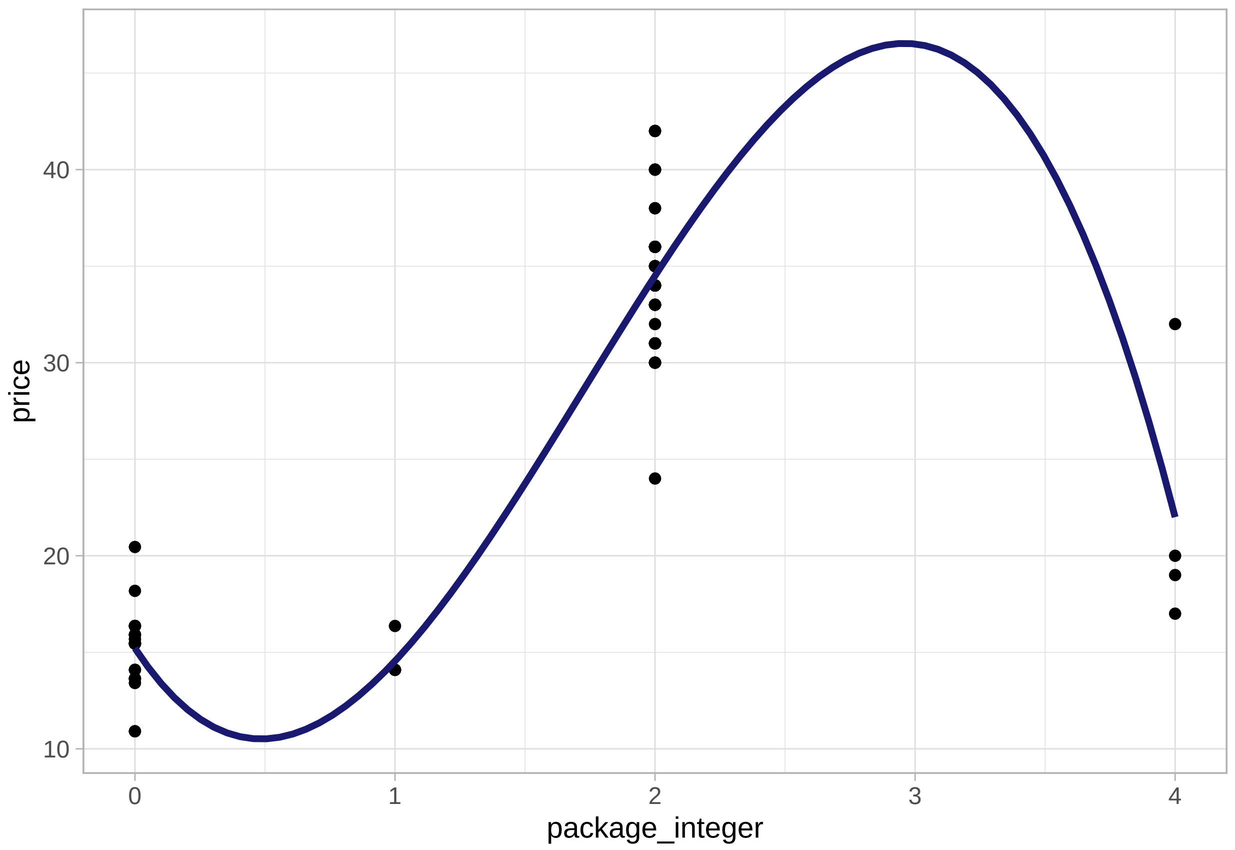
```
# Make a scatter plot
poly_results |> ggplot(
       aes(x = package_integer,
           y = price)) +
  geom_point() +
  geom_line(aes(y = .pred))
```



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 = FALSE)
```



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.

Predictor variable: variety

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

```r
price_var <- cor(baked_pumpkins$price, baked_pumpkins$variety)

print(paste("There is a", round(price_var, 3), "correlation between pumpkin price and variety."))
```

```
## [1] "There is a -0.863 correlation between pumpkin price and variety."
```

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

- Create a recipe

```r
set.seed(321) #sets starting point for random number generation (to randomly split data)
# Split the data into training and test sets
pumpkins_split2 <- new_pumpkins %>%
  initial_split(prop = 0.8)

# Extract training and test data
ptrain <- training(pumpkins_split2)
ptest <- testing(pumpkins_split2)

# Create a recipe for preprocessing the
```

```r
#use 3rd degree (4th throws error)
# pumvar_recipe <- recipe(price ~ variety, data = ptrain) |>
#   step_integer(all_predictors(), zero_based = TRUE) |>
#   step_poly(all_predictors(), degree = 3) #NOT USING poly recipe because need linear to encode variety

#Using a linear recipe to make the encode column work
lin_pumvarrecipe <- recipe(price ~ variety,
                           data = ptrain) |>
  step_integer(all_predictors(), zero_based = TRUE)
```

- Build a model specification (linear or polynomial)

```r
# Create a linear model specification
mod_spec <- linear_reg() |>
  set_engine("lm") |>
  set_mode("regression")
```

- Bundle the recipe and model specification into a workflow

```r
mod_wf <- workflow() %>%
  add_recipe(lin_pumvarrecipe) %>%
  add_model(mod_spec)
```

- Create a model by fitting the workflow

```r
# Train the model
mod_wf_fit <- mod_wf |>
  fit(data = ptrain)

# Print the model coefficients learned
print(mod_wf_fit)
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_integer()
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)      variety
##      42.100       -8.619
```

- Make predictions with the test data

```r
# Make price predictions on test data
mod_results <- mod_wf_fit %>% predict(new_data = ptest) %>%
  bind_cols(ptest %>% select(c(variety, price))) %>%
  relocate(.pred, .after = last_col())
```

```
head(mod_results)
```

```
## # A tibble: 6 x 3
##   variety  price .pred
##   <chr>    <dbl> <dbl>
## 1 PIE TYPE  16.4  16.2
## 2 PIE TYPE  13.6  16.2
## 3 PIE TYPE  16.4  16.2
## 4 PIE TYPE  16.1  16.2
## 5 PIE TYPE  16.1  16.2
## 6 PIE TYPE  16.4  16.2
```

- Evaluate model performance on the test data

```
# Evaluate performance of linear regression
metrics(data = mod_results,
        truth = price,
        estimate = .pred)
```

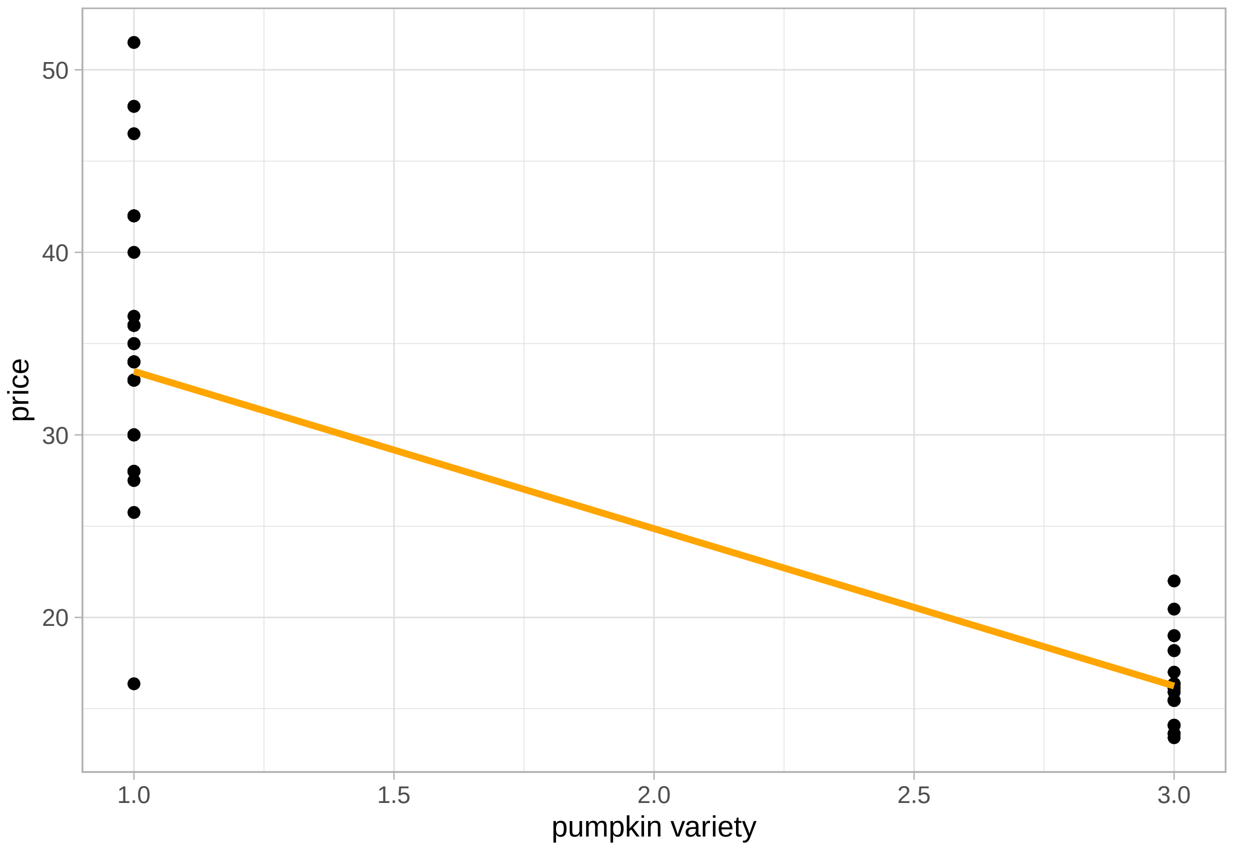
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       4.96
## 2 rsq     standard       0.763
## 3 mae     standard       3.15
```

- Create a visualization of model performance

```
#need to encode variety column in order to visualize

#Encode variety column with pumvar_recipe
variety_encode <- lin_pumvarrecipe %>%  #WHY won't this step work with a polynomial recipe???
  prep() %>%
  bake(new_data = ptest) %>%
  select(variety)

# Bind encoded package column to the results
 mod_plot <- mod_results %>%
 bind_cols(variety_encode %>%
              rename(variety_integer = variety)) %>%
  relocate(variety_integer, .after = variety)


# Print new results data frame
print(head(mod_plot))
```

```
## # A tibble: 6 x 4
##   variety  variety_integer price .pred
##   <chr>              <dbl> <dbl> <dbl>
## 1 PIE TYPE               3  16.4  16.2
## 2 PIE TYPE               3  13.6  16.2
## 3 PIE TYPE               3  16.4  16.2
## 4 PIE TYPE               3  16.1  16.2
## 5 PIE TYPE               3  16.1  16.2
## 6 PIE TYPE               3  16.4  16.2
```

```r
# Make a scatter plot
mod_plot %>%
  ggplot(aes(x = variety_integer,
             y = price)) +
   geom_point(size = 1.6) +
   geom_line(aes(y = .pred),
              color = "orange",
              size = 1.2) +
   xlab("pumpkin variety") +
  ylab("price")
```



Lab 2 due 1/24 at 11:59 PM