Robinson_Lab2

Adelaide Robinson

2023-01-18

```
library(here)
library(knitr)
```

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!).

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)
#don't want a higher polynomial than the number of variables
```

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

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

```r
# 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
```

```r
# 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       3.27
## 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 new polynomial model performs better than the linear model from last time according to all the metrics we are looking at. Root Mean Squared Error is lower in the polynomial model. The r squared value is higher which indicates a better model fit. And the MAE value is lower, which also means it is more accurate**

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

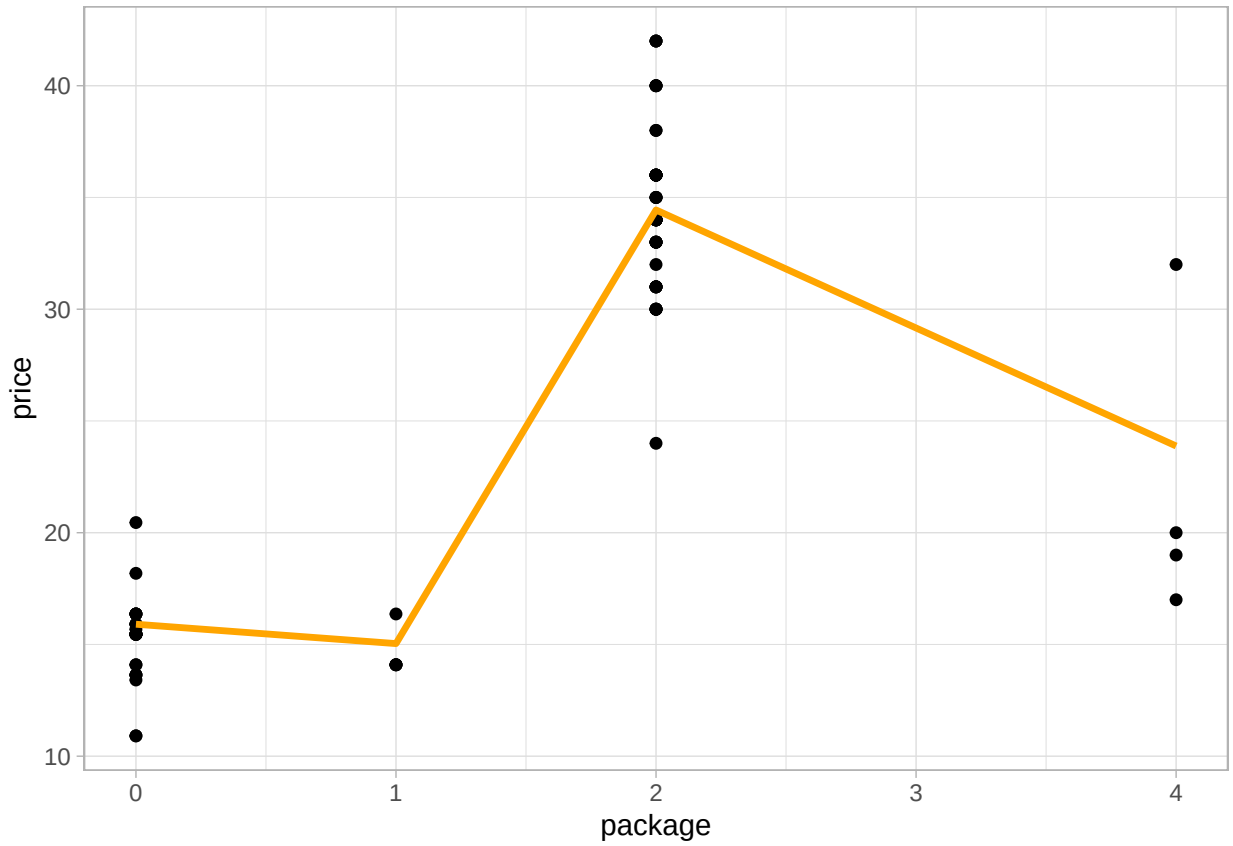
```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


# 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.

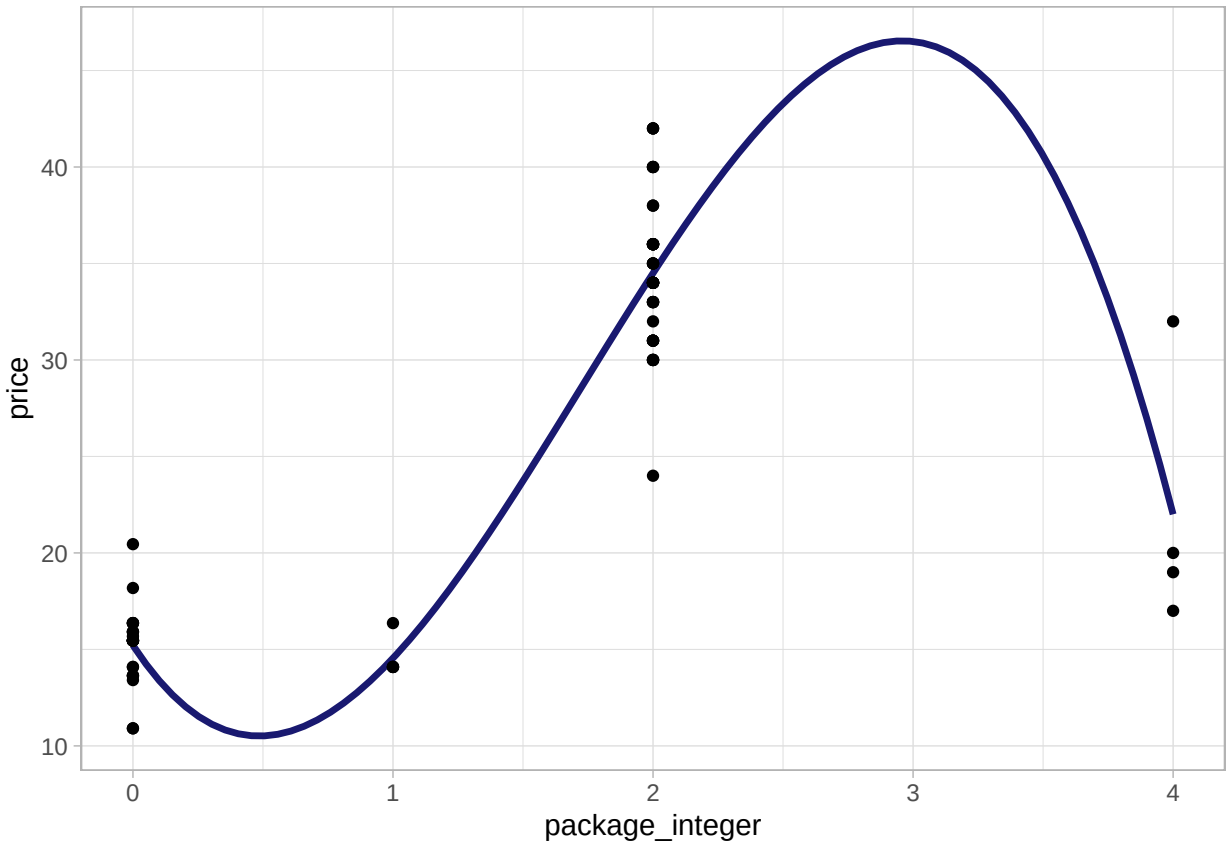
```
# Make a scatter plot
poly_results %>% ggplot(mapping = aes(x = package_integer, y = price)) + geom_point(size = 1.6) +
   # Overlay a line of best fit
   geom_line(aes(y = .pred), color = "orange", size = 1.2) +
   xlab("package")
```



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(aes(x = package_integer, y = price)) + geom_smooth(method = lm, formula = y ~ po
```



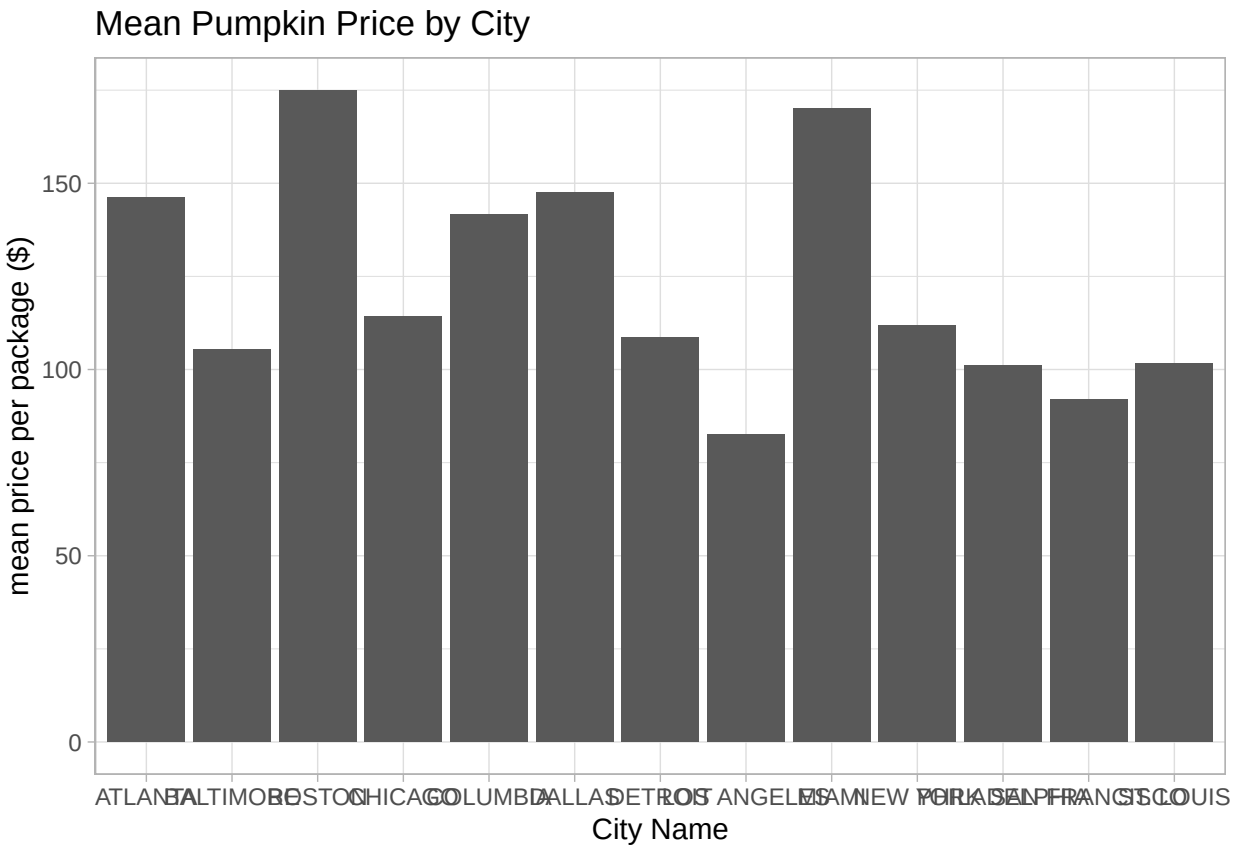
OK, now it's your turn to go through the process one more time.

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

**** 1. I chose city name as my variable of interest. "**

```
#very rough exploratory graph
ggplot(data = pumpkins) +
  geom_point(aes(x = city_name, y = price)) +
  labs(title = "Pumpkin Prices by City", y = "Price per Package", x = "City")
```


```
# Find the average price of pumpkins per month then plot a bar chart
pumpkins |> group_by(city_name) |> summarize(mean_price = mean(price)) |>
  ggplot() +
  geom_col(aes(x = city_name, y = mean_price)) +
  labs(title = "Mean Pumpkin Price by City", x = "City Name", y = "mean price per package ($)")
```




```
# Specify a recipe
poly_pumpkins_recipe <-
  recipe(price ~ city_name, data = pumpkins_train) %>%
  step_integer(all_predictors(), zero_based = TRUE) %>%
  step_poly(all_predictors(), degree = 9)
#don't want a higher polynomial than the number of variables

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

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

# 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)  city_name_poly_1  city_name_poly_2  city_name_poly_3
##          27.9706           57.1517            2.8754           -2.0957
## city_name_poly_4  city_name_poly_5  city_name_poly_6  city_name_poly_7
##         -34.7686          -31.2166          -43.0757           -3.4437
## city_name_poly_8  city_name_poly_9
##           0.3299          -38.3019
```

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

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

```
## # A tibble: 10 x 3
##    city_name price .pred
##    <chr>     <dbl> <dbl>
##  1 BALTIMORE  13.6  25.3
##  2 BALTIMORE  16.4  25.3
##  3 BALTIMORE  16.4  25.3
##  4 BALTIMORE  13.6  25.3
##  5 BALTIMORE  15.5  25.3
##  6 BALTIMORE  16.4  25.3
##  7 BALTIMORE  34    25.3
##  8 BALTIMORE  30    25.3
##  9 BALTIMORE  30    25.3
## 10 BALTIMORE  34    25.3
```

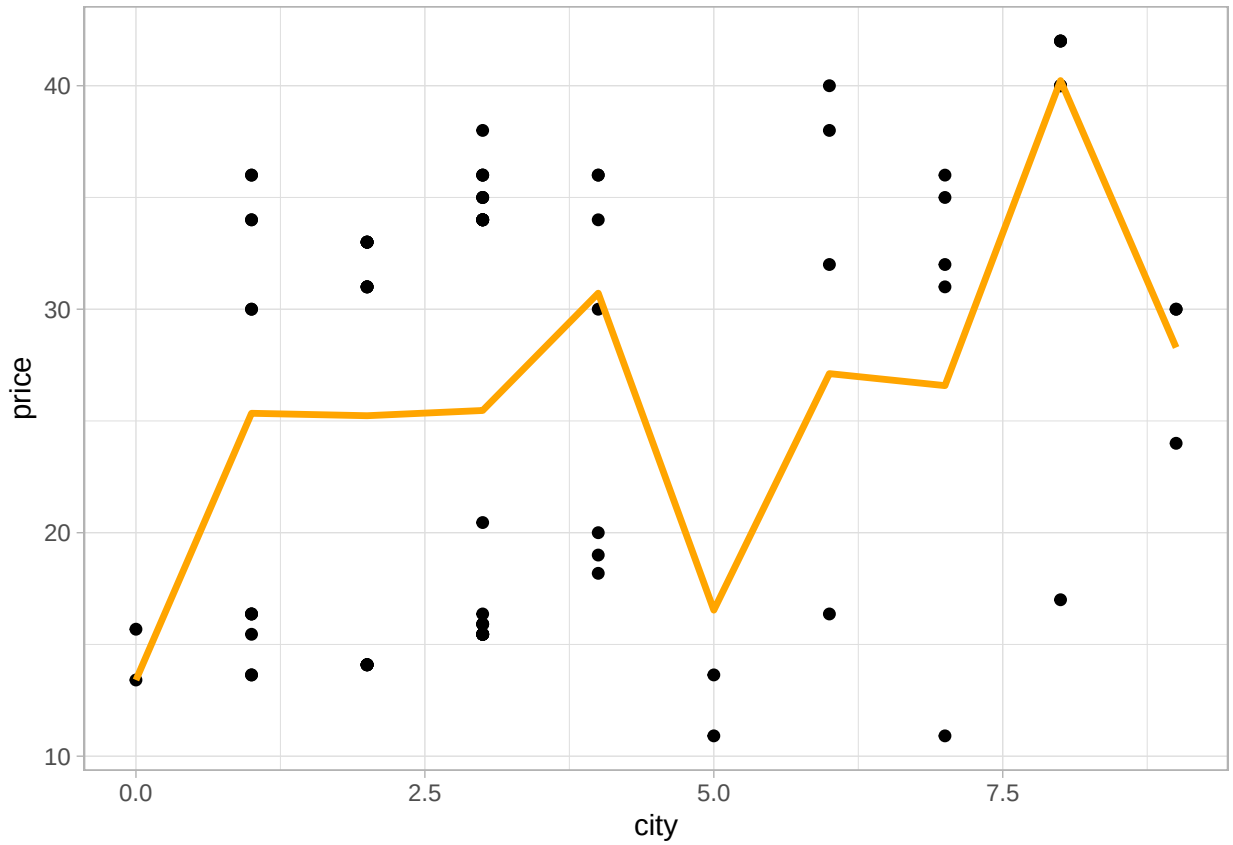
```
metrics(data = poly_results, truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       8.88
## 2 rsq     standard       0.191
## 3 mae     standard       7.98
```

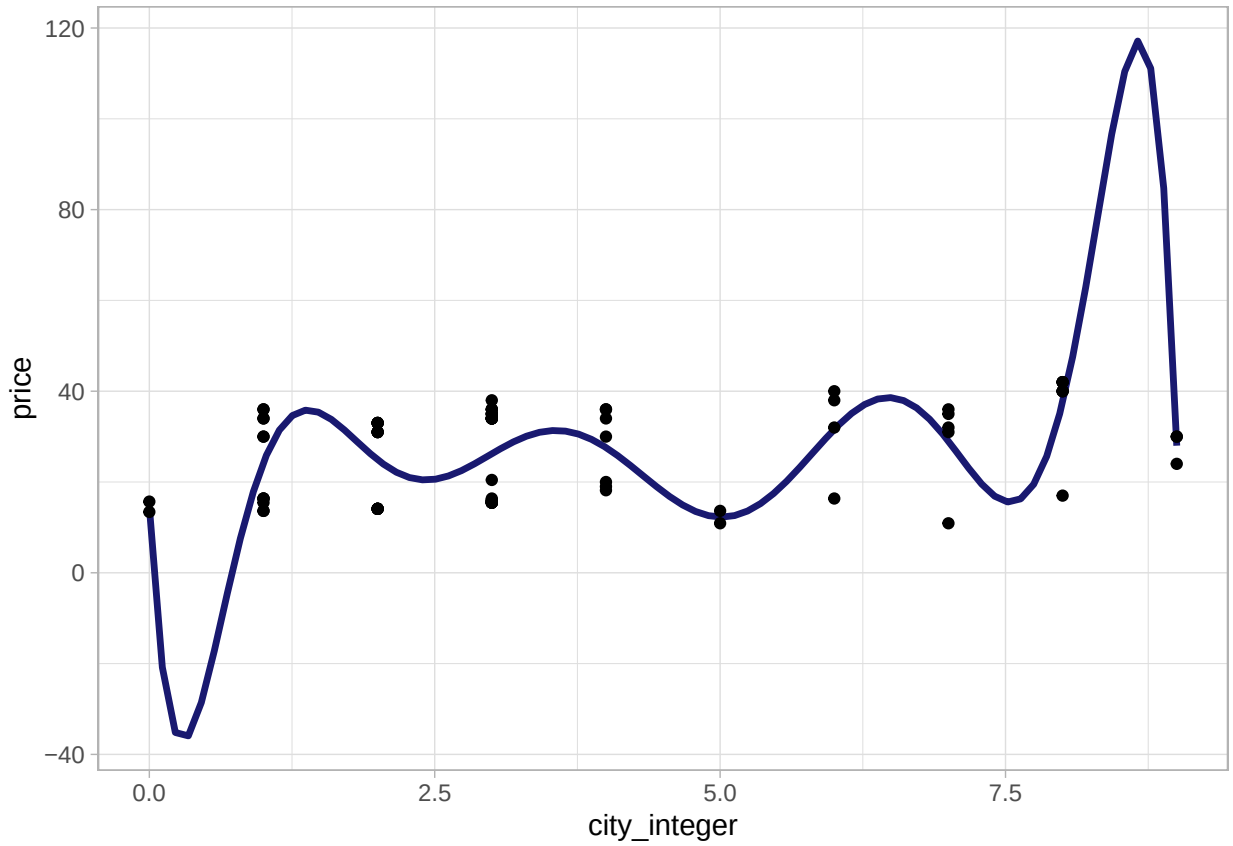
```r
# Encode city_name column
package_encode <- pumpkins_recipe %>%
  prep() %>%
  bake(new_data = pumpkins_test) %>%
  select(city_name)

# Bind encoded package column to the results
poly_results2 <- poly_results %>%
  bind_cols(package_encode %>%
              rename(city_integer = city_name)) %>%
  relocate(city_integer, .after = city_name)


# Print new results data frame
poly_results2 %>%
  slice_head(n = 5)
```

```
## # A tibble: 5 x 4
##   city_name city_integer price .pred
##   <chr>            <int> <dbl> <dbl>
## 1 BALTIMORE            1  13.6  25.3
## 2 BALTIMORE            1  16.4  25.3
## 3 BALTIMORE            1  16.4  25.3
## 4 BALTIMORE            1  13.6  25.3
## 5 BALTIMORE            1  15.5  25.3
```

```r
# Make a scatter plot
poly_results2 %>% 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", size = 1.2) + xlab("city")
```




```
# Make a smoother scatter plot
poly_results2 %>% ggplot(aes(x = city_integer, y = price)) + geom_smooth(method = lm, formula = y ~ poly
```


120
80
40
0
−40
price
0.0
2.5
5.0
7.5
city_integer