Becker_Lab2

Kate Becker

1/18/2024

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.

Lab 1 required for accessing data and varaiables

```
#Load required packages and read in data
library("tidymodels")
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.1.1 --

## v broom        1.0.5     v recipes      1.0.8
## v dials        1.2.0     v rsample      1.2.0
## v dplyr        1.1.3     v tibble       3.2.1
## v ggplot2      3.4.4     v tidyr        1.3.0
## v infer        1.0.5     v tune         1.1.2
## v modeldata    1.2.0     v workflows    1.1.3
## v parsnip      1.1.1     v workflowsets 1.0.1
## v purrr        1.0.2     v yardstick    1.2.0

## -- 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()
## * Dig deeper into tidy modeling with R at https://www.tmwr.org
```

```
library("tidyverse")
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.4
## v lubridate 1.9.2     v stringr   1.5.0

## -- 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()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library("dplyr")
library("janitor")
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library("corrplot")
```

```
## corrplot 0.92 loaded
```

```
library("lubridate")
dat <- read_csv(file = "https://raw.githubusercontent.com/MaRo406/eds-232-machine-learning/main/data/pu
```

```
## 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.
## * ‘‘ -> ‘...1‘
## * ‘...25‘ -> ‘...26‘
## * ‘...26‘ -> ‘...27‘
```

```
#look at df
glimpse(dat)
```

```
## 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, ~
```

```r
#clean column names
pumpkins <- dat %>% clean_names(case = "snake")

#check for names cleaned
pumpkins %>% names()
```

```
##  [1] "x1"              "city_name"       "type"            "package"
##  [5] "variety"         "sub_variety"     "grade"           "date"
##  [9] "low_price"       "high_price"      "mostly_low"      "mostly_high"
## [13] "origin"          "origin_district" "item_size"       "color"
## [17] "environment"     "unit_of_sale"    "quality"         "condition"
## [21] "appearance"      "storage"         "crop"            "repack"
## [25] "trans_mode"      "x26"             "x27"
```

```r
#select for variety, city_name, package, city_name, package, low_price, high_price, and date
pumpkins <- pumpkins %>% select(variety, city_name, package, low_price, high_price, date)

#look at the first 5 rows of the dataset
pumpkins %>% slice_head(n = 5)
```

```
## # A tibble: 5 x 6
##   variety     city_name package      low_price high_price date
##   <chr>       <chr>     <chr>            <dbl>      <dbl> <chr>
## 1 <NA>        BALTIMORE 24 inch bins       270        280 4/29/17
## 2 <NA>        BALTIMORE 24 inch bins       270        280 5/6/17
## 3 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 9/24/16
## 4 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 9/24/16
## 5 HOWDEN TYPE BALTIMORE 24 inch bins        90        100 11/5/16
```

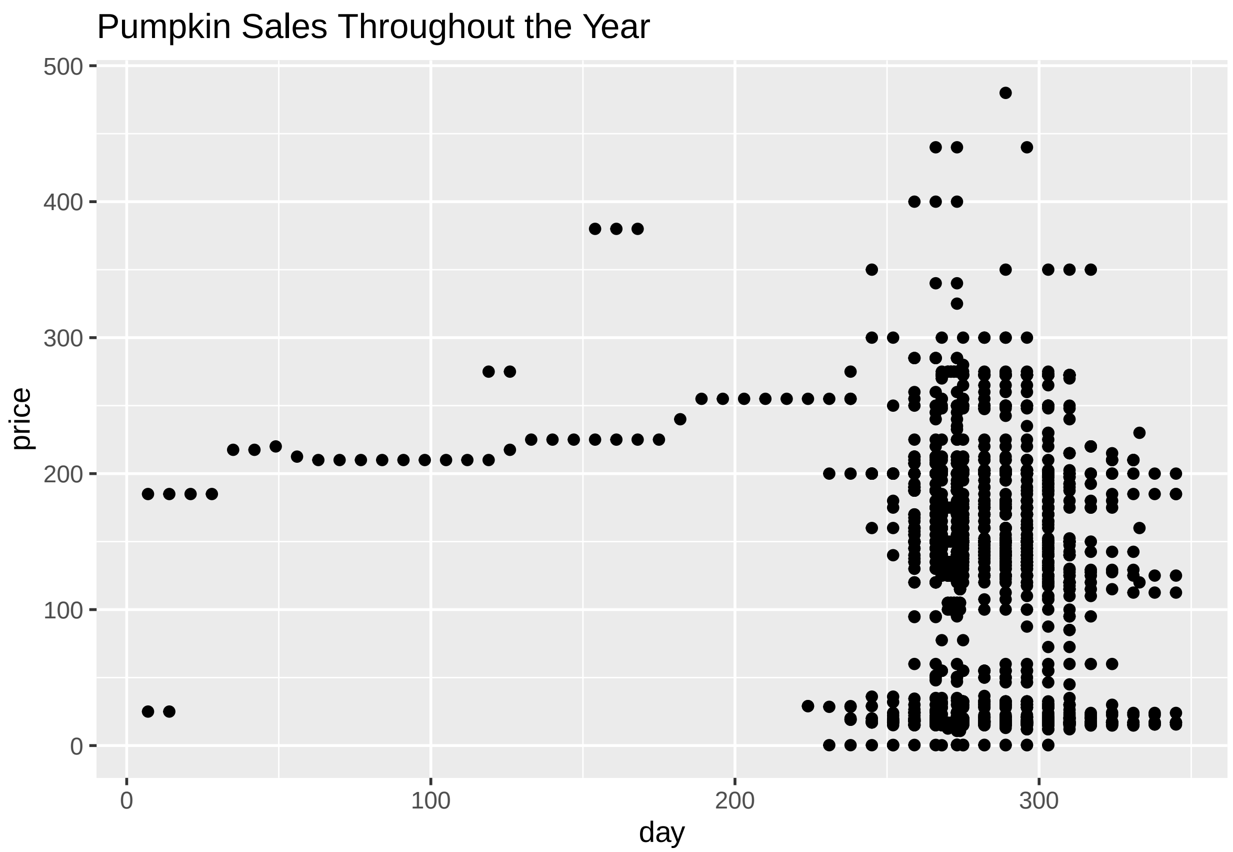
```r
#load in lubridate dataset
library(lubridate)

#  Extract the month and day from the dates and add as new columns
pumpkins <- pumpkins %>%
  mutate(date = mdy(date),
         day = yday(date),
         month = month(date))
pumpkins %>%
  select(-day)
```

```
## # A tibble: 1,757 x 7
##    variety     city_name package      low_price high_price date       month
##    <chr>       <chr>     <chr>            <dbl>      <dbl> <date>     <dbl>
##  1 <NA>        BALTIMORE 24 inch bins       270        280 2017-04-29     4
##  2 <NA>        BALTIMORE 24 inch bins       270        280 2017-05-06     5
##  3 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 2016-09-24     9
##  4 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 2016-09-24     9
##  5 HOWDEN TYPE BALTIMORE 24 inch bins        90        100 2016-11-05    11
##  6 HOWDEN TYPE BALTIMORE 24 inch bins        90        100 2016-11-12    11
##  7 HOWDEN TYPE BALTIMORE 36 inch bins       160        170 2016-09-24     9
##  8 HOWDEN TYPE BALTIMORE 36 inch bins       160        160 2016-09-24     9
##  9 HOWDEN TYPE BALTIMORE 36 inch bins       160        170 2016-10-01    10
## 10 HOWDEN TYPE BALTIMORE 36 inch bins       160        160 2016-10-01    10
## # i 1,747 more rows
```

```
#view the first 7 rows of the dataframe
pumpkins %>% slice_head(n = 7)
```

```
## # A tibble: 7 x 8
##   variety     city_name package      low_price high_price date         day month
##   <chr>       <chr>     <chr>            <dbl>      <dbl> <date>     <dbl> <dbl>
## 1 <NA>        BALTIMORE 24 inch bins       270        280 2017-04-29   119     4
## 2 <NA>        BALTIMORE 24 inch bins       270        280 2017-05-06   126     5
## 3 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 2016-09-24   268     9
## 4 HOWDEN TYPE BALTIMORE 24 inch bins       160        160 2016-09-24   268     9
## 5 HOWDEN TYPE BALTIMORE 24 inch bins        90        100 2016-11-05   310    11
## 6 HOWDEN TYPE BALTIMORE 24 inch bins        90        100 2016-11-12   317    11
## 7 HOWDEN TYPE BALTIMORE 36 inch bins       160        170 2016-09-24   268     9
```

```
#create new column "price" by adding low_price and high_price values and divide by 2

pumpkins <- pumpkins %>%
  mutate(price = (low_price+ high_price)/2)

#visualize predictor variable, date, and price variable, response variable

ggplot(data = pumpkins, aes(x = day, y = price)) +
  geom_point() +
  ggtitle("Pumpkin Sales Throughout the Year")
```



```r
# Verify the distinct observations in Package column

pumpkins %>%
  distinct(package)
```

```
## # A tibble: 15 x 1
##    package
##    <chr>
##  1 24 inch bins
##  2 36 inch bins
##  3 50 lb sacks
##  4 1 1/9 bushel cartons
##  5 1/2 bushel cartons
##  6 1 1/9 bushel crates
##  7 bushel cartons
##  8 bins
##  9 35 lb cartons
## 10 each
## 11 20 lb cartons
## 12 50 lb cartons
## 13 40 lb cartons
## 14 bushel baskets
## 15 22 lb cartons
```

```r
#look at first 5 rows of dataframe
pumpkins %>% slice_head(n = 5)
```

```
## # A tibble: 5 x 9
##   variety    city_name package low_price high_price date         day month price
##   <chr>      <chr>     <chr>       <dbl>      <dbl> <date>     <dbl> <dbl> <dbl>
## 1 <NA>       BALTIMORE 24 inc~       270        280 2017-04-29   119     4   275
## 2 <NA>       BALTIMORE 24 inc~       270        280 2017-05-06   126     5   275
## 3 HOWDEN TY~ BALTIMORE 24 inc~       160        160 2016-09-24   268     9   160
## 4 HOWDEN TY~ BALTIMORE 24 inc~       160        160 2016-09-24   268     9   160
## 5 HOWDEN TY~ BALTIMORE 24 inc~        90        100 2016-11-05   310    11    95
```

```r
# Verify the distinct observations in Package column

pumpkins %>% distinct(package)
```

```
## # A tibble: 15 x 1
##    package
##    <chr>
##  1 24 inch bins
##  2 36 inch bins
##  3 50 lb sacks
##  4 1 1/9 bushel cartons
##  5 1/2 bushel cartons
##  6 1 1/9 bushel crates
##  7 bushel cartons
##  8 bins
##  9 35 lb cartons
## 10 each
## 11 20 lb cartons
## 12 50 lb cartons
## 13 40 lb cartons
## 14 bushel baskets
## 15 22 lb cartons
```

```r
# Retain only pumpkins with "bushel" in the package column
new_pumpkins <- pumpkins %>%
    filter(str_detect(package, "bushel"))

#check dimensions
dim(new_pumpkins)
```

```
## [1] 415   9
```

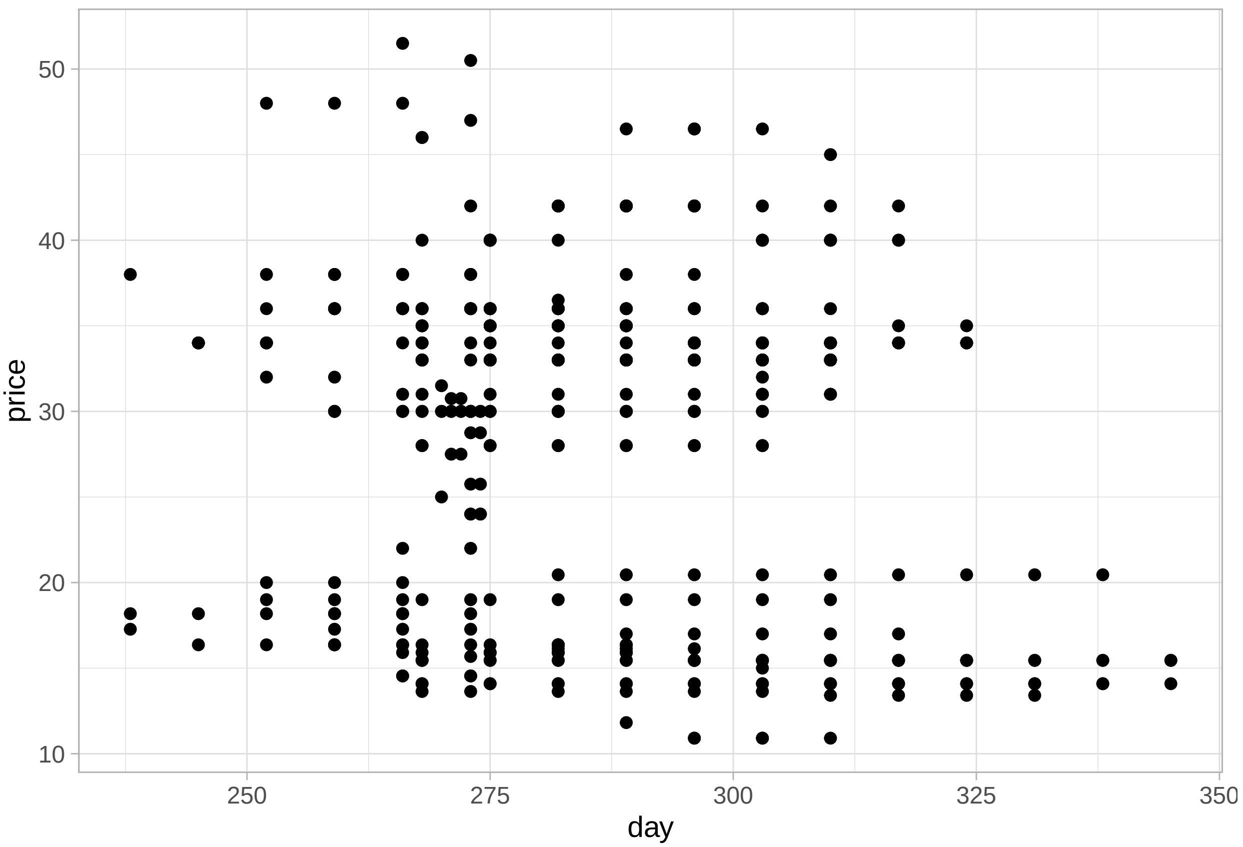
```r
#look at first 10 rows of dataset
new_pumpkins %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 9
##    variety  city_name package   low_price high_price date         day month price
##    <chr>    <chr>     <chr>         <dbl>      <dbl> <date>     <dbl> <dbl> <dbl>
##  1 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-09-24   268     9  15
##  2 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-09-24   268     9  18
##  3 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-10-01   275    10  18
##  4 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-01   275    10  17
##  5 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-10-08   282    10  15
##  6 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-10-08   282    10  18
##  7 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-08   282    10  17
##  8 PIE TYPE BALTIMORE 1 1/9 b~        17       18.5 2016-10-08   282    10  17.8
##  9 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-10-15   289    10  15
## 10 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-15   289    10  17
```

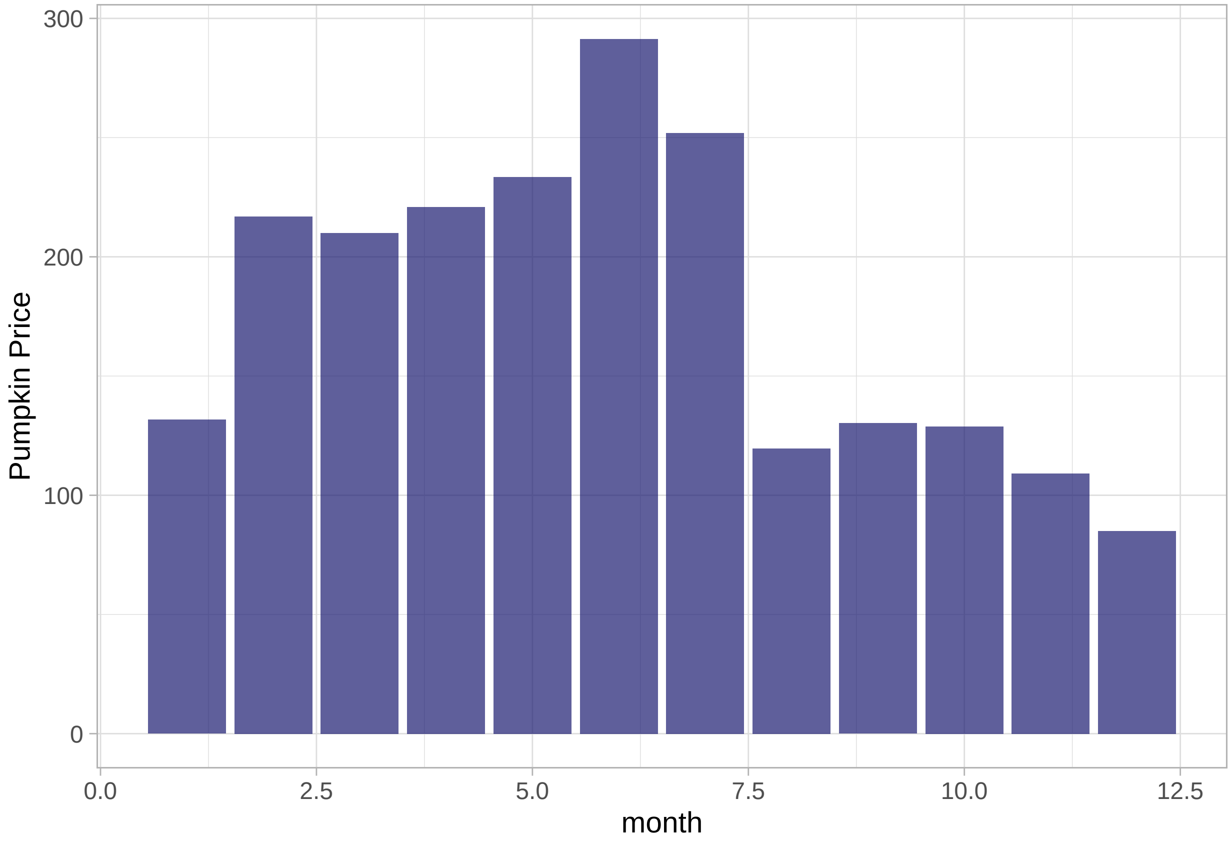
```
# Convert the price if the Package contains fractional bushel values
new_pumpkins <- new_pumpkins %>%
  mutate(price = case_when(
    str_detect(package, "1 1/9") ~ price/(1.1),
    str_detect(package, "1/2") ~ price*2,
    TRUE ~ price))

# View the first few rows of the data
new_pumpkins %>%
  slice_head(n = 30)
```

```
## # A tibble: 30 x 9
##    variety  city_name package  low_price high_price date         day month price
##    <chr>    <chr>     <chr>        <dbl>      <dbl> <date>     <dbl> <dbl> <dbl>
##  1 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-09-24   268     9  13.6
##  2 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-09-24   268     9  16.4
##  3 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-10-01   275    10  16.4
##  4 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-01   275    10  15.5
##  5 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-10-08   282    10  13.6
##  6 PIE TYPE BALTIMORE 1 1/9 b~        18       18   2016-10-08   282    10  16.4
##  7 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-08   282    10  15.5
##  8 PIE TYPE BALTIMORE 1 1/9 b~        17       18.5 2016-10-08   282    10  16.1
##  9 PIE TYPE BALTIMORE 1 1/9 b~        15       15   2016-10-15   289    10  13.6
## 10 PIE TYPE BALTIMORE 1 1/9 b~        17       17   2016-10-15   289    10  15.5
## # i 20 more rows
```

```
theme_set(theme_light())

# Make a scatter plot of day and price
new_pumpkins %>%
  ggplot(mapping = aes(x = day, y = price)) +
  geom_point(size = 1.6)
```



```
# Find the average price of pumpkins per month then plot a bar chart

pumpkins %>%
  group_by(month) %>%
  summarise(mean_price = mean(price)) %>%
  ggplot(aes(x = month, y = mean_price)) +
  geom_col(fill = "midnightblue", alpha = 0.7) +
  ylab("Pumpkin Price")
```



```
pumpkins_recipe <- recipe(price ~ ., data = new_pumpkins) %>%  #the dot means adding all x variables to
  step_integer(all_predictors(), zero_based = TRUE) #new data creates a specific recipe coverting new d


# Print out the recipe
pumpkins_recipe
```

```
##
## -- Recipe ----------------------------------------------------------------------
##
## -- Inputs
## Number of variables by role
## outcome:   1
## predictor: 8
##
## -- Operations
## * Integer encoding for: all_predictors()
```

```
#prep the recipe
pumpkins_prep <- prep(pumpkins_recipe)

# Bake the recipe to extract a preprocessed new_pumpkins data
baked_pumpkins <- bake(pumpkins_prep, new_data = NULL)

# Print out the baked data set
```

```
baked_pumpkins %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 9
##    variety city_name package low_price high_price  date   day month price
##      <int>     <int>   <int>     <int>      <int> <int> <int> <int> <dbl>
##  1       3         1       0         5          3     0     5     1  13.6
##  2       3         1       0        10          7     0     5     1  16.4
##  3       3         1       0        10          7     6    11     2  16.4
##  4       3         1       0         9          6     6    11     2  15.5
##  5       3         1       0         5          3     7    12     2  13.6
##  6       3         1       0        10          7     7    12     2  16.4
##  7       3         1       0         9          6     7    12     2  15.5
##  8       3         1       0         9          8     7    12     2  16.1
##  9       3         1       0         5          3     8    13     2  13.6
## 10       3         1       0         9          6     8    13     2  15.5
```

```
#print correlation between package and price variables

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

```
## [1] 0.6061713
```

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

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

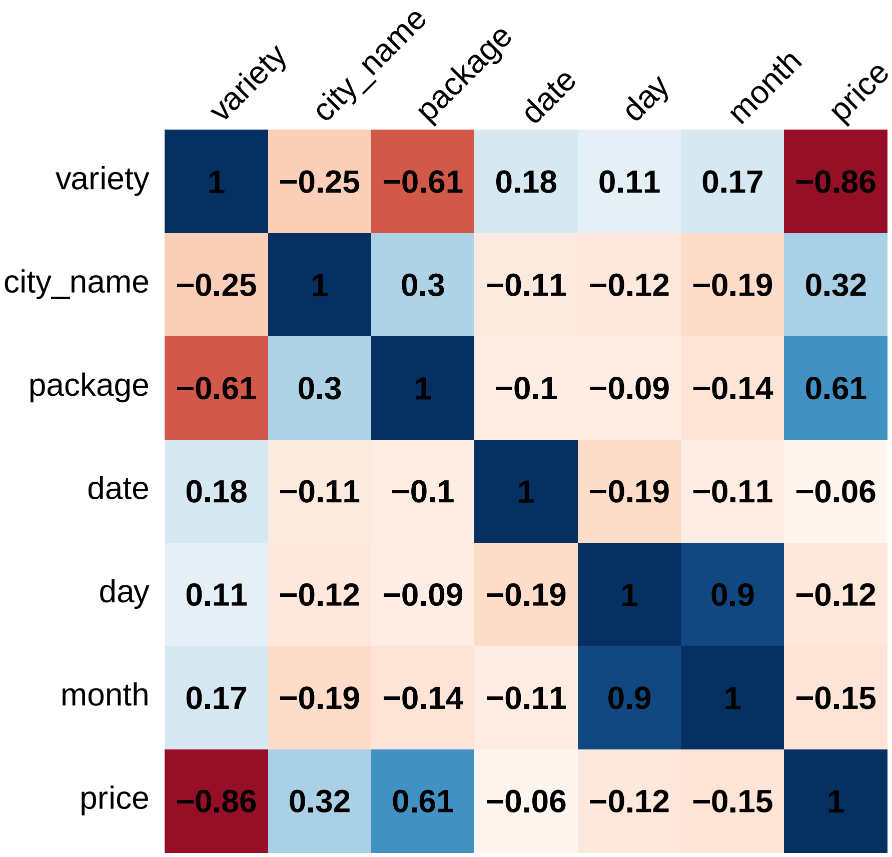
```
cor(baked_pumpkins$variety, baked_pumpkins$price)
```

```
## [1] -0.863479
```

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

```
## [1] -0.1487829
```

```
# Obtain correlation matrix
corr_mat <- cor(baked_pumpkins %>%
                  # Drop columns that are not really informative
                  select(-c(low_price, high_price)))

# Make a correlation plot between the variables
corrplot(corr_mat, method = "shade", shade.col = NA, tl.col = "black", tl.srt = 45, addCoef.col = "blac
```



```r
set.seed(123)
# Split the data into training and test sets
pumpkins_split <- baked_pumpkins %>%  #new_pumpkins should be baked_pumpkins
  initial_split(prop = 0.8)


# Extract training and test data
pumpkins_train <- training(pumpkins_split)
pumpkins_test <- testing(pumpkins_split)


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


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

# Hold modeling components in a workflow
lm_wf <- workflow() %>%
  add_recipe(lm_pumpkins_recipe) %>%
  add_model(lm_spec)
```

```r
# Print out the workflow
lm_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_integer()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

```r
# Train the model
lm_wf_fit <- lm_wf %>%
  fit(data = pumpkins_train)

# Print the model coefficients learned
lm_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)      package
##       20.14         4.76
```

```r
# Make predictions for the test set
predictions <- lm_wf_fit %>%
  predict(new_data = pumpkins_test)

# Bind predictions to the test set
lm_results <- pumpkins_test %>%
  select(c(package, price)) %>%
  bind_cols(predictions)
```

```r
# Print the first ten rows of the tibble
lm_results %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 3
##    package price .pred
##      <int> <dbl> <dbl>
##  1       0  13.6  20.1
##  2       0  16.4  20.1
##  3       0  16.4  20.1
##  4       0  13.6  20.1
##  5       0  15.5  20.1
##  6       0  16.4  20.1
##  7       2  34    29.7
##  8       2  30    29.7
##  9       2  30    29.7
## 10       2  34    29.7
```

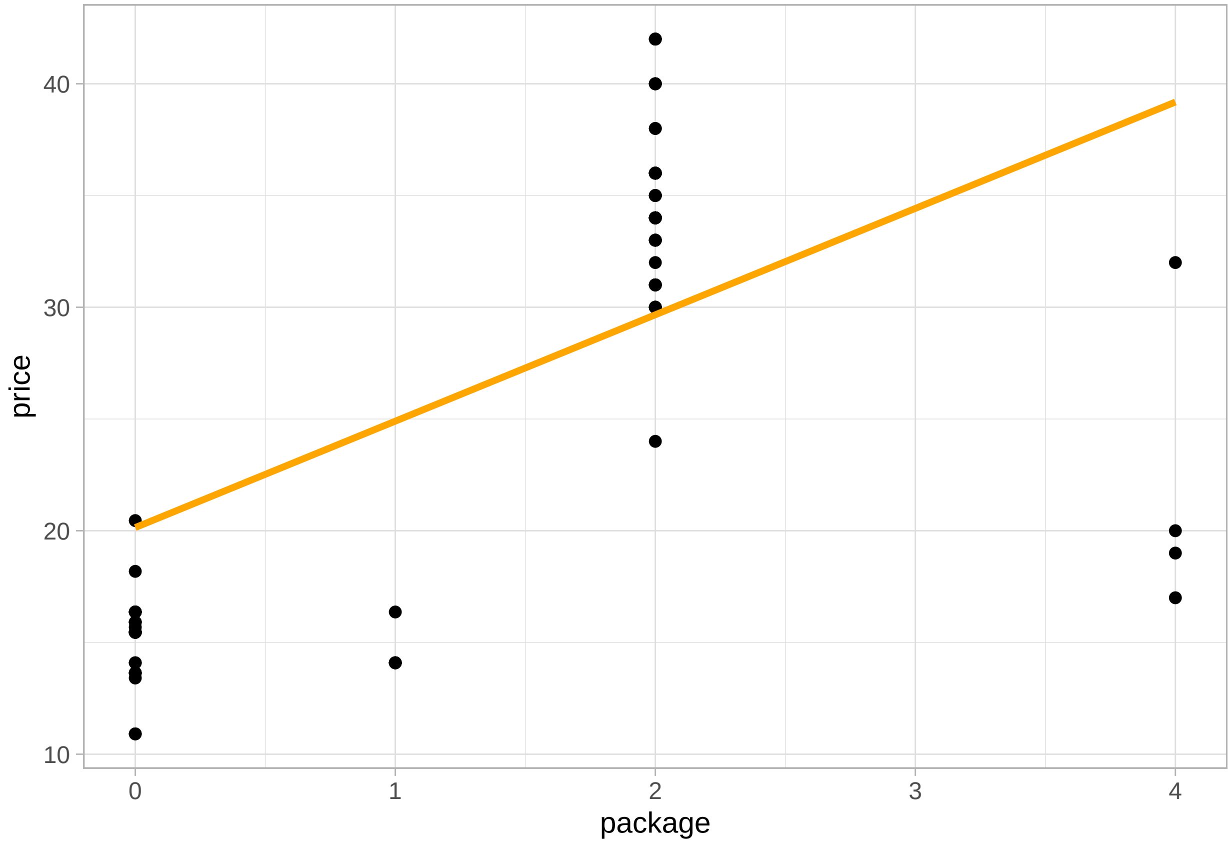
```r
# Evaluate performance of linear regression
metrics(data = lm_results,
        truth = price,
        estimate = .pred)
```

```
## # 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
```

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


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

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

```
## # A tibble: 5 x 4
##   package package_integer price .pred
##     <int>           <int> <dbl> <dbl>
## 1       0               0  13.6  20.1
## 2       0               0  16.4  20.1
## 3       0               0  16.4  20.1
## 4       0               0  13.6  20.1
## 5       0               0  15.5  20.1
```

```
# Make a scatter plot
plot_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", linewidth = 1.2) +
   xlab("package")
```



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()

Lab 2

```
# Specify a recipe
poly_pumpkins_recipe <-
  recipe(price ~ package, data = pumpkins_train) %>% #specfies a recipe that identifies variables and d
  step_integer(all_predictors(), zero_based = TRUE) %>%  #converts package variable to a numerical form
  step_poly(all_predictors(), degree = 3) #adds a 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() %>%
  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)

# Print out the workflow
poly_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_integer()
## * step_poly()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

```
#step_integer creates a specification of a recipe step that will convert new data into a set of integer
#creates specificaiton of a recipe step that will create new columns that are basic expansions of varia
```

Question 2: fit a model to the pumpkins_train data using your workflow and assign it to poly_wf_fit

```
# Train the model
poly_wf_fit <- poly_wf %>%
  fit(data = pumpkins_train)

# Print the model coefficients learned
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()) #.pred precits values based on input data

# 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().

```
#yardstick::metrics() estimates one or more common performance estimates depending on the class of trut
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
```

```
#smaller rmse, rsq, and mae all prove the model to be better
#mean absolute error
#yardstick works to estimate one or more common performance estiamtes depending on the class of truth
```

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?

Compared to the linear model from last week, the polynomial we fit today minimized rmse and mae and maximized rsq clear indicators that this model best fits our data. When applied to our data, the RMSE here implies that on average, this model mispredicts the expected price of different packages by about 328 dollars compared to 722 dollars as observed after using the linear regression. In terms of r-squared, the polynomial model has a much larger Rˆ2, 0.89 compared to 0.49, showing a stronger proportion of the variance in pumpkin price predicted from package type using the polynomial regression. Finally, in terms of MAE, the mean absolute difference between the actual and predicted values is minimized in the polynomial regression and can therefore better predict the price of different packages of pumpkins. In conclusion, the polynomial model does a better job at predicting the price of different packages of pumpkins as seen by the performance metrics.

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

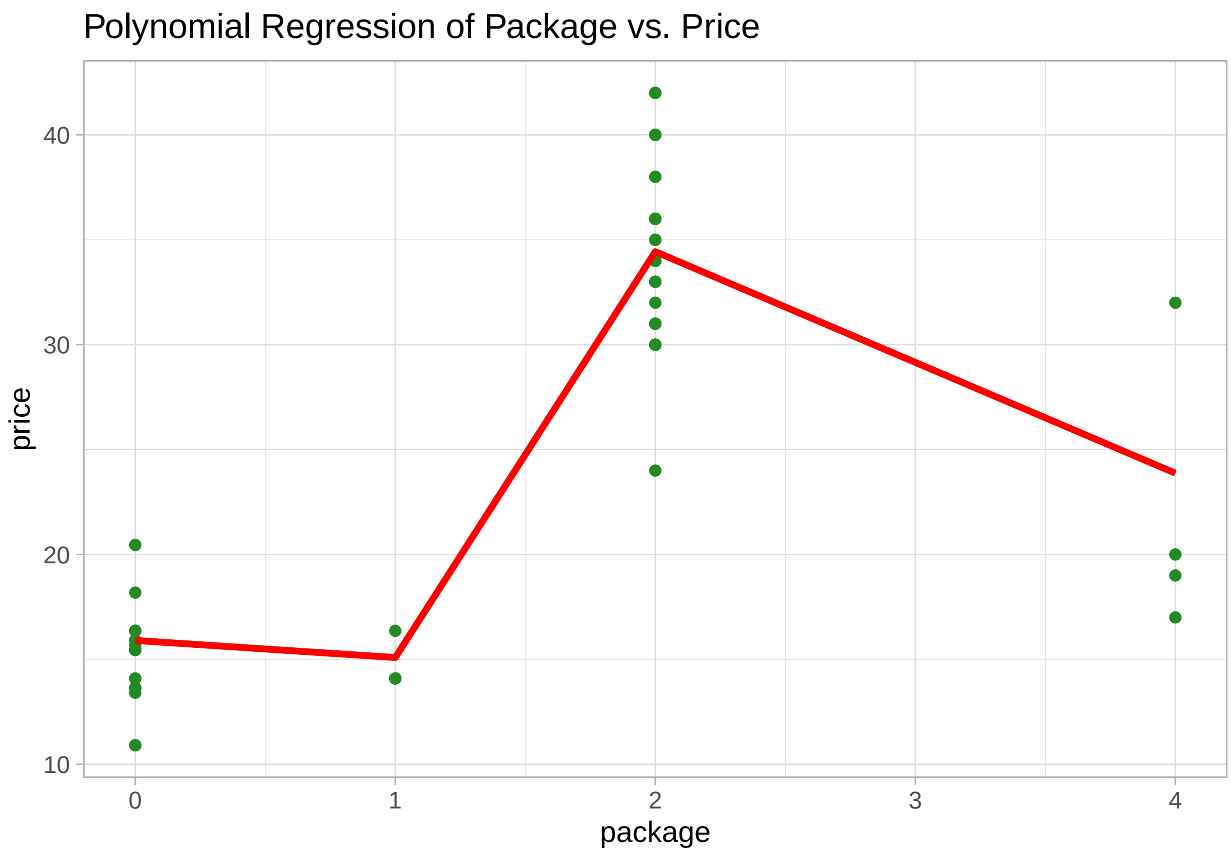
```
# Bind encoded package column to the results
poly_results <- poly_results %>%
  bind_cols(package_encode %>%
              rename(package_integer = package)) %>%
  relocate(package_integer, .after = package)


# 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(aes(x = package, y = price)) +
  geom_point(color = "forestgreen") +
  geom_line(aes(y = .pred), color = "red", linewidth = 1.2) +
  ggtitle("Polynomial Regression of Package vs. Price")
```



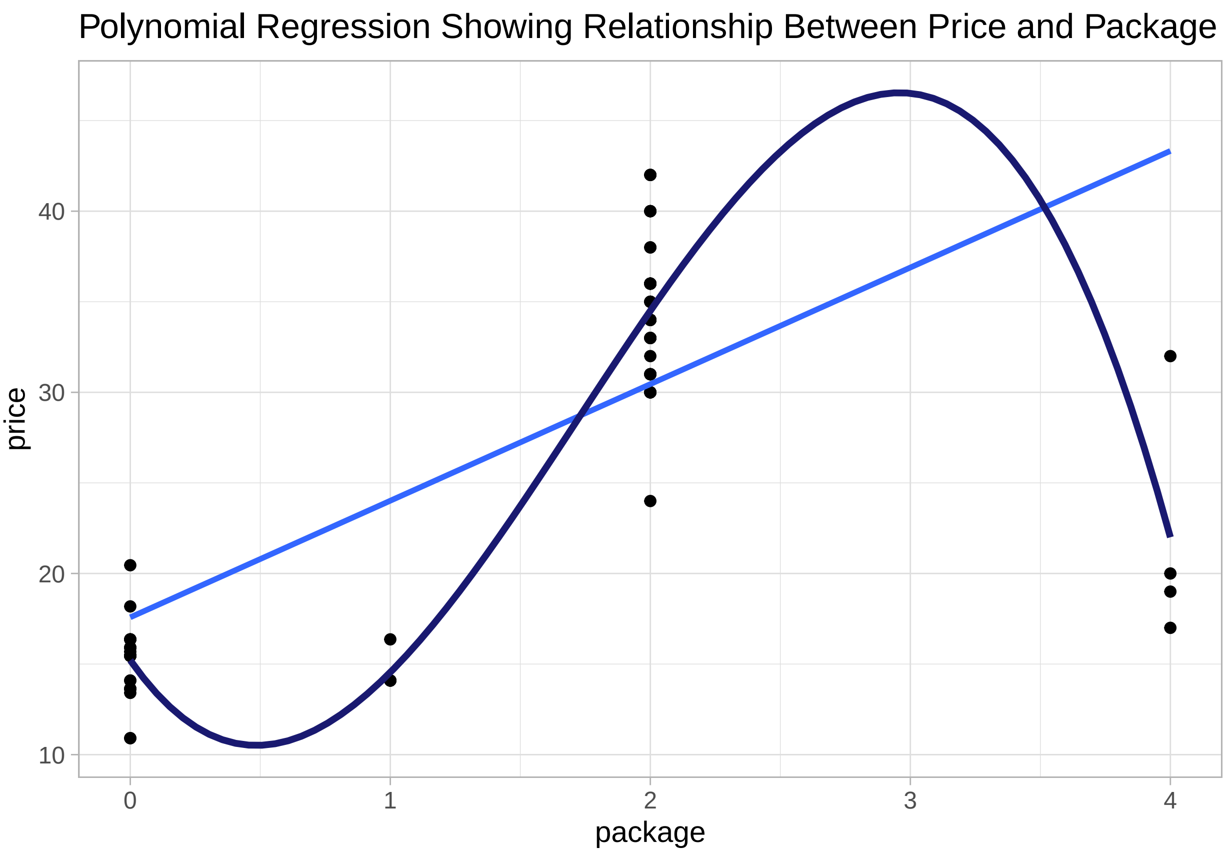
You can see that a curved line fits your data much better.

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(aes(x = package, y = price)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE) +
  geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "midnightblue", size = 1.2, se = F
  ggtitle("Polynomial Regression Showing Relationship Between Price and 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.
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



```
# view poly_results df and class of package and price variables
View(poly_results)
class(poly_results$package)
```

```
## [1] "integer"
```

```
class(poly_results$price)
```

```
## [1] "numeric"
```

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)

```
#find correlation between city_name predictor variable and price response variable
cor(baked_pumpkins$city_name, baked_pumpkins$price)
```

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

The correlation between the predictor variable (city_name) and the outcome variable (price) is 0.324 approximately.

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

Testing the Linear Model

```r
#random number generator
set.seed(123)

# Split the data into training and test sets
pumpkins_city_split <- baked_pumpkins %>%
  initial_split(prop = 0.8)

# Extract training and test data
pumpkins_train2 <- training(pumpkins_city_split)
pumpkins_test2 <- testing(pumpkins_city_split)

# Create a recipe for preprocessing the data
lm_city_recipe <- recipe(price ~ city_name, data = pumpkins_train2) %>%
  step_integer(all_predictors(), zero_based = TRUE)


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

# Hold modeling components in a workflow
lm_workflow <- workflow() %>%
  add_recipe(lm_city_recipe) %>%
  add_model(lm_specify)

# Print out the workflow
lm_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_integer()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
```

```
## Computational engine: lm
```

```
# Train the model
lm_workflow_train <- lm_workflow %>%
  fit(data = pumpkins_train2)

# Print the model coefficients learned
lm_workflow_train
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_integer()
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
## (Intercept)    city_name
##      22.656        1.266
```

```
# Make predictions for the test set
test_predictions <- lm_workflow_train %>%
  predict(new_data = pumpkins_test2)

# Bind predictions to the test set
results1 <- pumpkins_test2 %>%
  select(c(city_name, price)) %>%
  bind_cols(test_predictions)


# Print the first ten rows of the tibble
results1 %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 3
##    city_name price .pred
##        <int> <dbl> <dbl>
##  1         1  13.6  23.9
##  2         1  16.4  23.9
##  3         1  16.4  23.9
##  4         1  13.6  23.9
##  5         1  15.5  23.9
##  6         1  16.4  23.9
##  7         1  34    23.9
##  8         1  30    23.9
##  9         1  30    23.9
## 10         1  34    23.9
```

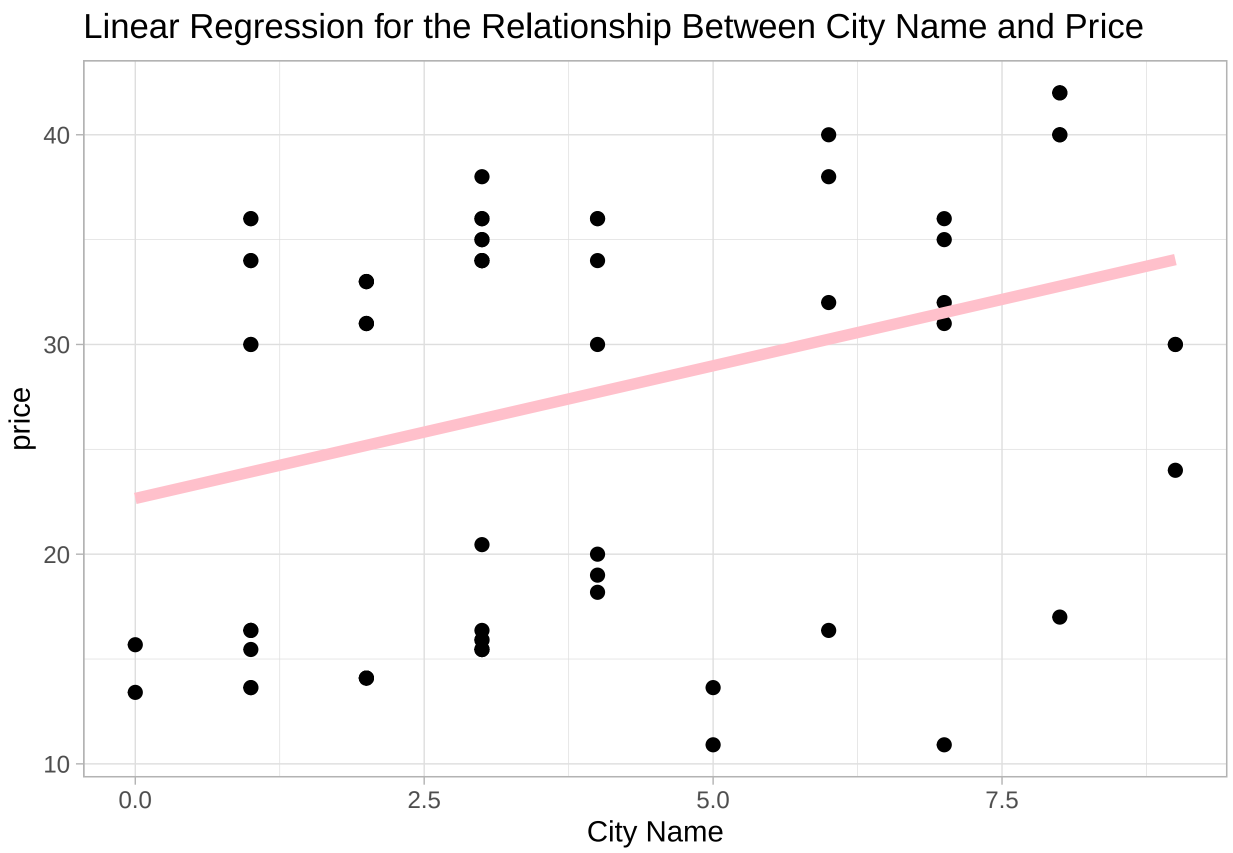
```r
#yardstick::metrics() estimates one or more common performance estimates depending on the class of trut
metrics(data = results1,
        truth = price,
        estimate = .pred)
```

```
## # 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
```

```r
# Encode package column
city_encode <- lm_city_recipe %>%
  prep() %>%
  bake(new_data = pumpkins_test2) %>%
  select(city_name)


# Bind encoded package column to the results
 plot <- results1 %>%
 bind_cols(city_encode %>%
               rename(city_integer = city_name)) %>%
  relocate(city_integer, .after = city_name)

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

```
## # A tibble: 5 x 4
##   city_name city_integer price .pred
##       <int>        <int> <dbl> <dbl>
## 1         1            1  13.6  23.9
## 2         1            1  16.4  23.9
## 3         1            1  16.4  23.9
## 4         1            1  13.6  23.9
## 5         1            1  15.5  23.9
```

```r
# Make a scatter plot
plot %>%
  ggplot(mapping = aes(x = city_integer, y = price)) +
   geom_point(size = 2) +
   # Overlay a line of best fit
   geom_line(aes(y = .pred), color = "pink", linewidth = 2) +
   xlab("City Name") +
  ggtitle("Linear Regression for the Relationship Between City Name and Price")
```



## Testing the Polynomial Model

```
#random number generator
set.seed(123)
# Split the data into training and test sets
pumpkins_city_poly <- baked_pumpkins %>%  #new_pumpkins should be baked_pumpkins
  initial_split(prop = 0.8)


# Extract training and test data
pumpkins_train2 <- training(pumpkins_city_poly)
pumpkins_test2 <- testing(pumpkins_city_poly)

# Create a recipe for preprocessing the data
polynomial_recipe <-
  recipe(price ~ city_name, data = pumpkins_train2) %>% #specfies a recipe that identifies variables an
  step_integer(all_predictors(), zero_based = TRUE) %>%  #converts package variable to a numerical form
  step_poly(all_predictors(), degree = 4) #adds a polynomial effect


# Create a polynomial specification
poly_spec1 <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Bundle recipe and model spec into a workflow
poly_workflow <- workflow() %>%
```

```r
  add_recipe(polynomial_recipe) %>%
  add_model(poly_spec1)

# Print out the workflow
poly_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_integer()
## * step_poly()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

```r
# Train the model
poly_workflow_fitted <- poly_workflow %>%
  fit(data = pumpkins_train2)

# Print the model coefficients learned
poly_workflow_fitted
```

```
## == 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
```

```r
# Make price predictions on test data
results2 <- poly_workflow_fitted %>% predict(new_data = pumpkins_test2) %>%
  bind_cols(pumpkins_test2 %>% select(c(city_name, price))) %>%
  relocate(.pred, .after = last_col()) #.pred precits values based on input data


# Print the results
results1 %>%
  slice_head(n = 10)
```

```
## # A tibble: 10 x 3
##    city_name price .pred
##        <int> <dbl> <dbl>
##  1         1  13.6  23.9
##  2         1  16.4  23.9
##  3         1  16.4  23.9
##  4         1  13.6  23.9
##  5         1  15.5  23.9
##  6         1  16.4  23.9
##  7         1  34    23.9
##  8         1  30    23.9
##  9         1  30    23.9
## 10         1  34    23.9
```

```
#yardstick::metrics() estimates one or more common performance estimates depending on the class of trut
metrics(data = results2, truth = price, estimate = .pred)
```

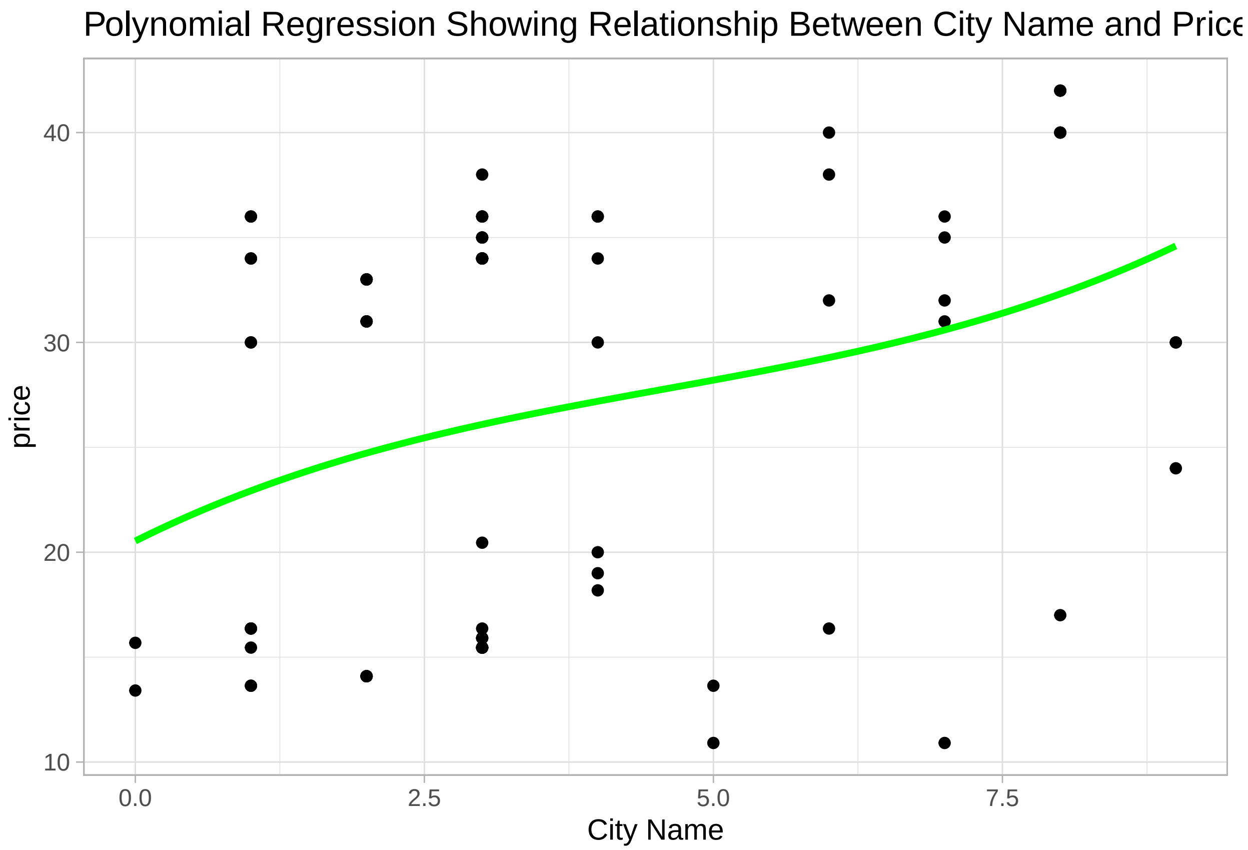
```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       9.15
## 2 rsq     standard       0.142
## 3 mae     standard       8.26
```

```
#smaller rmse, rsq, and mae all prove the model to be better
#mean absolute error
#yardstick works to estimate one or more common performance estiamtes depending on the class of truth
```

```
# Bind encoded package column to the results
output <- results2 %>%
  bind_cols(city_encode %>%
              rename(city_integer = city_name)) %>%
  relocate(city_integer, .after = city_name)


# Printed new results
output %>%
  slice_head(n = 5)
```

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

```
#Linear Regression Plot
output %>%
  ggplot(aes(x = city_name, y = price)) +
  geom_point() +
  geom_smooth(method = lm, formula = y ~ poly(x, degree = 3), color = "green", size = 1.2, se = FALSE) +
  xlab("City Name") +
  ggtitle("Polynomial Regression Showing Relationship Between City Name and Price")
```



Lab 2 due 1/24 at 11:59 PM