

# Lab1

Luna Herschenfeld-Catalan

2024-01-21

## Case Study: The Pumpkin Market

The data you just loaded includes 1757 lines of data about the market for pumpkins, sorted into groupings by city. This is raw data extracted from the Specialty Crops Terminal Markets Standard Reports distributed by the United States Department of Agriculture.

You are loading a pumpkin data set in order to ask questions of it:

When is the best time to buy pumpkins? - date / price

What price can I expect of a case of miniature pumpkins?

Should I buy them in half-bushel baskets or by the 1 1/9 bushel box?

### Examine the data

```
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, ~
## $ 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, ~
## $ Grade     <lgl> 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, ~
## $ '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, ~
## $ '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, ~
## $ '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, ~
## $ 'Unit of Sale' <chr> 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, ~
## $ Condition  <lgl> 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, ~
```

```
## $ Storage      <lg1> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ Crop         <lg1> 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", "N~
## $ 'Trans Mode' <lg1> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ ...26        <lg1> 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, ~
```

```
# Clean names to the snake_case convention
```

```
pumpkins <- dat %>%
  clean_names(case = "snake")
```

```
# Return column names
```

```
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"
```

## Select desired columns

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

```
## Print data set
```

```
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
```

```
## Load lubridate
```

```
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 few rows
```

```
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
```

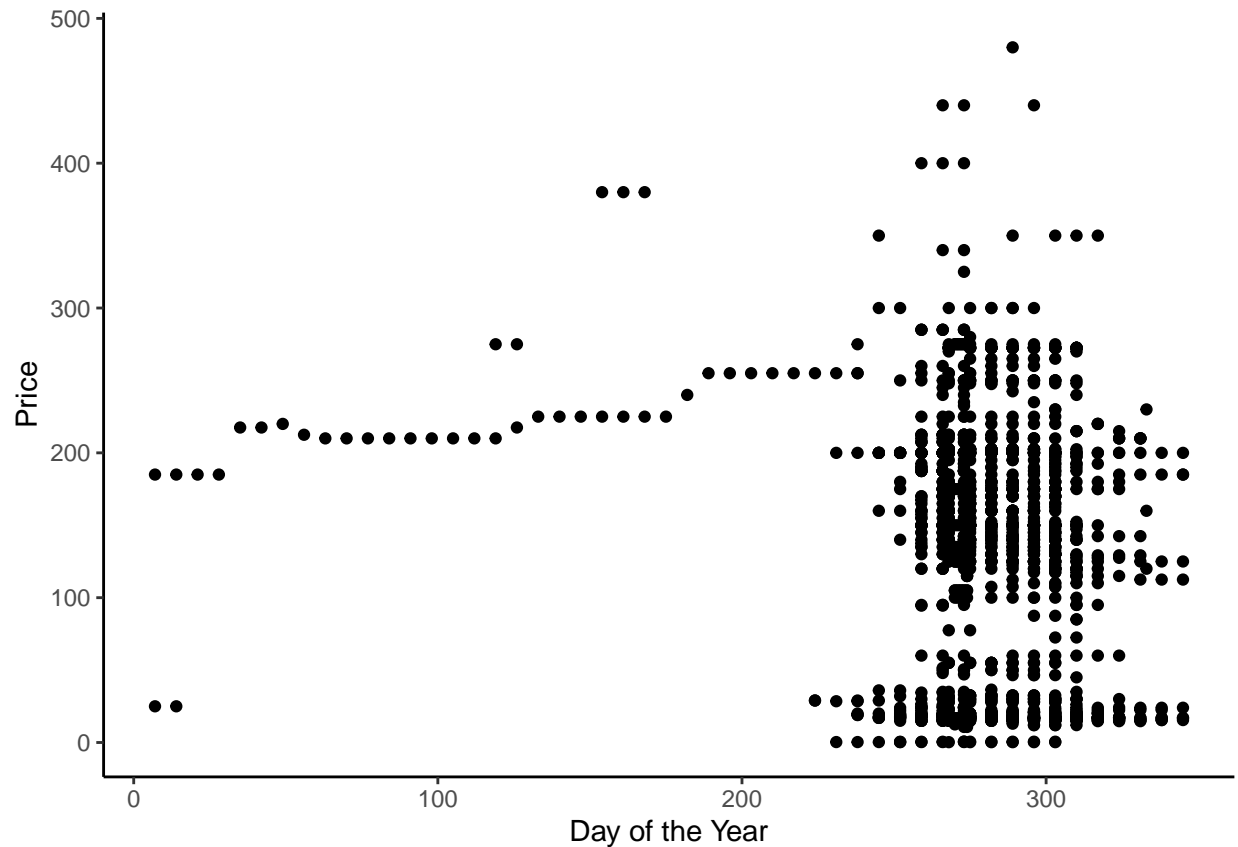
There are two column dealing with price, high and low. Let's combine them into a single average price column.

```
# Create a new column price
pumpkins <- pumpkins %>%
  mutate(price = (low_price+ high_price)/2) # find the average between these two values
```

Let's take a look at pumpkins sales throughout the year.

**Question 1:** Create a scatter plot using price on the y-axis and day on the x-axis.

```
# Make a scatter plot of day and price
ggplot(data = pumpkins) +
  geom_point(aes(x = day,
                 y = price)) +
  labs(x = "Day of the Year",
       y = "Price") +
  theme_classic()
```



Now, before we go any further, let's take another look at the data. Notice anything odd?

That's right: pumpkins are sold in many different configurations. Some are sold in 1 1/9 bushel measures, and some in 1/2 bushel measures, some per pumpkin, some per pound, and some in big boxes with varying widths.

Let's verify this:

```
# 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
```

Pumpkins seem to be very hard to weigh consistently, so let's filter them by selecting only pumpkins with the string bushel in the package column and put this in a new data frame "new\_pumpkins".

```
## View the first few rows of the data
```

```
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
```

```
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
```

**Question 2:** Use a combination of `dplyr::filter()` and `stringr::str_detect()` to achieve what we want.

```
# Retain only pumpkins with "bushel" in the package column
```

```
new_pumpkins <- pumpkins %>%
  filter(str_detect(package, "bushel")) # select only rows with 'bushel' in them
```

```
# Get the dimensions of the new data
```

```
dim(new_pumpkins)
```

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

```
# View a few rows of the new data
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
```

You should see that we have narrowed down to 415 rows of data containing pumpkins measured in bushels.

But wait! There's one more thing to do.

Did you notice that the bushel amount varies per row? You need to normalize the pricing so that you show the pricing per bushel, not per 1 1/9 or 1/2 bushel. Time to do some math to standardize it.

We'll use the function `case_when()` to mutate the Price column depending on some conditions. `case_when()` allows you to vectorize multiple `if_else()` statements.

```
# 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), # when detecting this value, update the price column va
    str_detect(package, "1/2") ~ price*2,      # when detecting this value, update the price column value
    TRUE ~ price)) # keep the rest of the values as they are

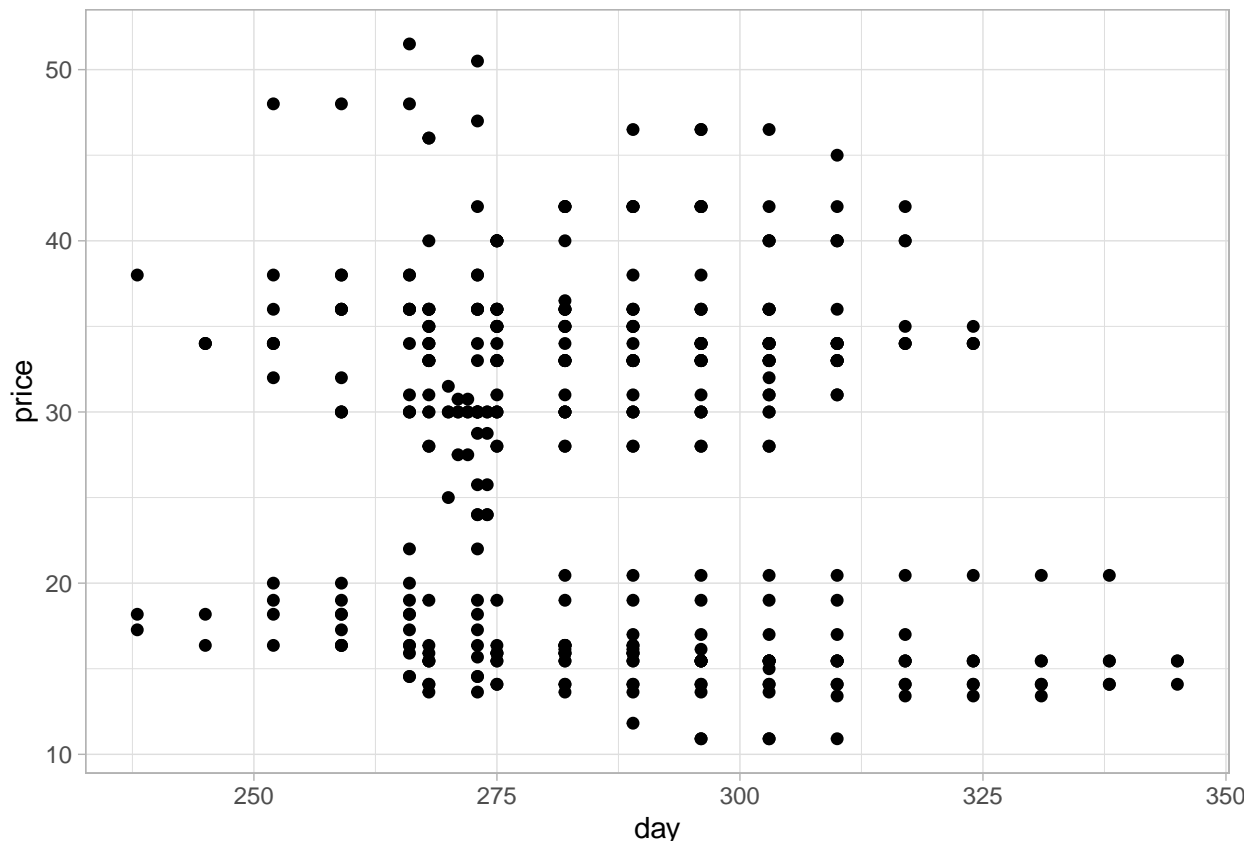
# View the first few rows of the data
new_pumpkins %>%
  slice_head(n = 30)
```

```
## # 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
```

## Data Visualization

```
# Set theme
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)
```



**Question 3:** Is this a useful plot? Does anything about it surprise you?

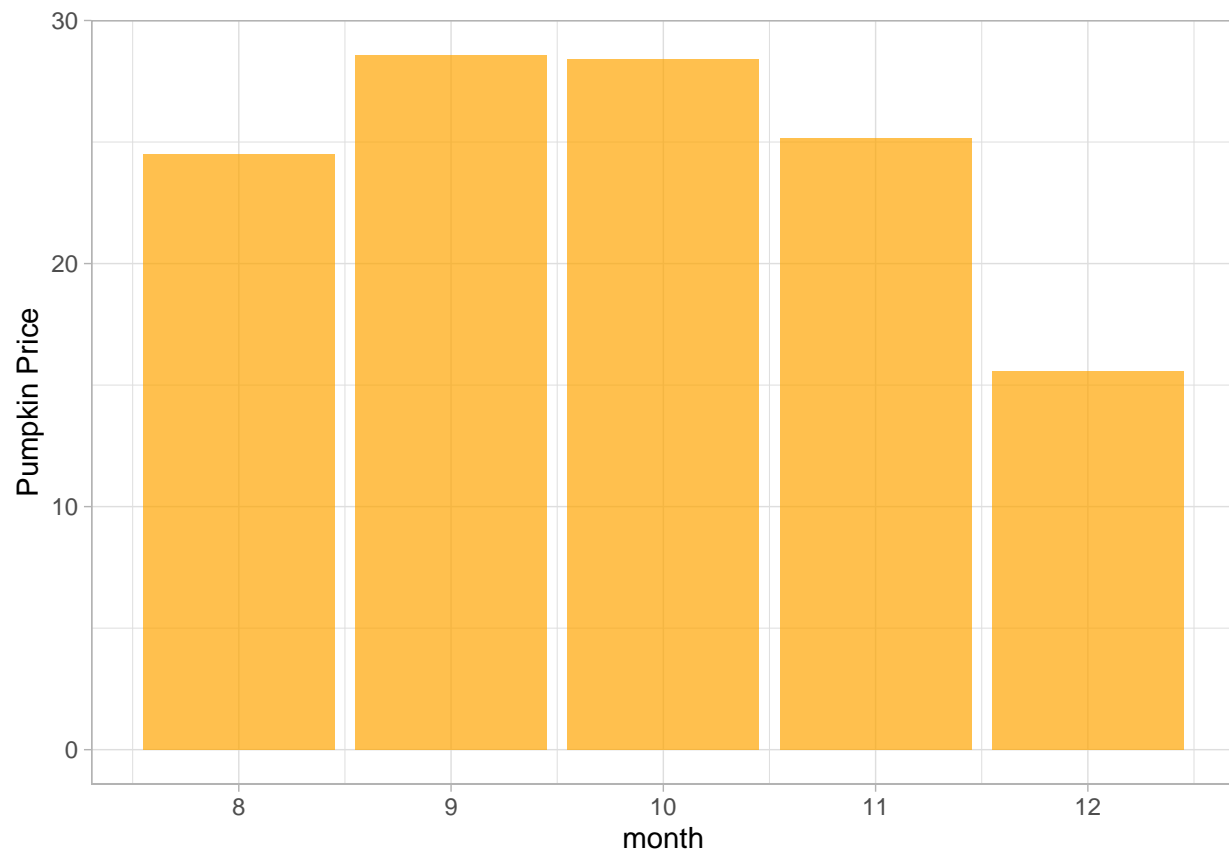
This plot is not very useful except to identify general trends. It seems as though there are two price points, though it is hard to identify if those belong to different groups. Also, whereas the first plot had data from day 0 to 365, this plot only contains data from day 225 to 350. This is about the time that the pumpkin holidays are happening, so it makes sense.

How do we make it more useful? To get charts to display useful data, you usually need to group the data somehow.

**Question 4:** Group the pumpkins into groups based on the month column and then find the mean price for each month. Plot the results with a bar plot.

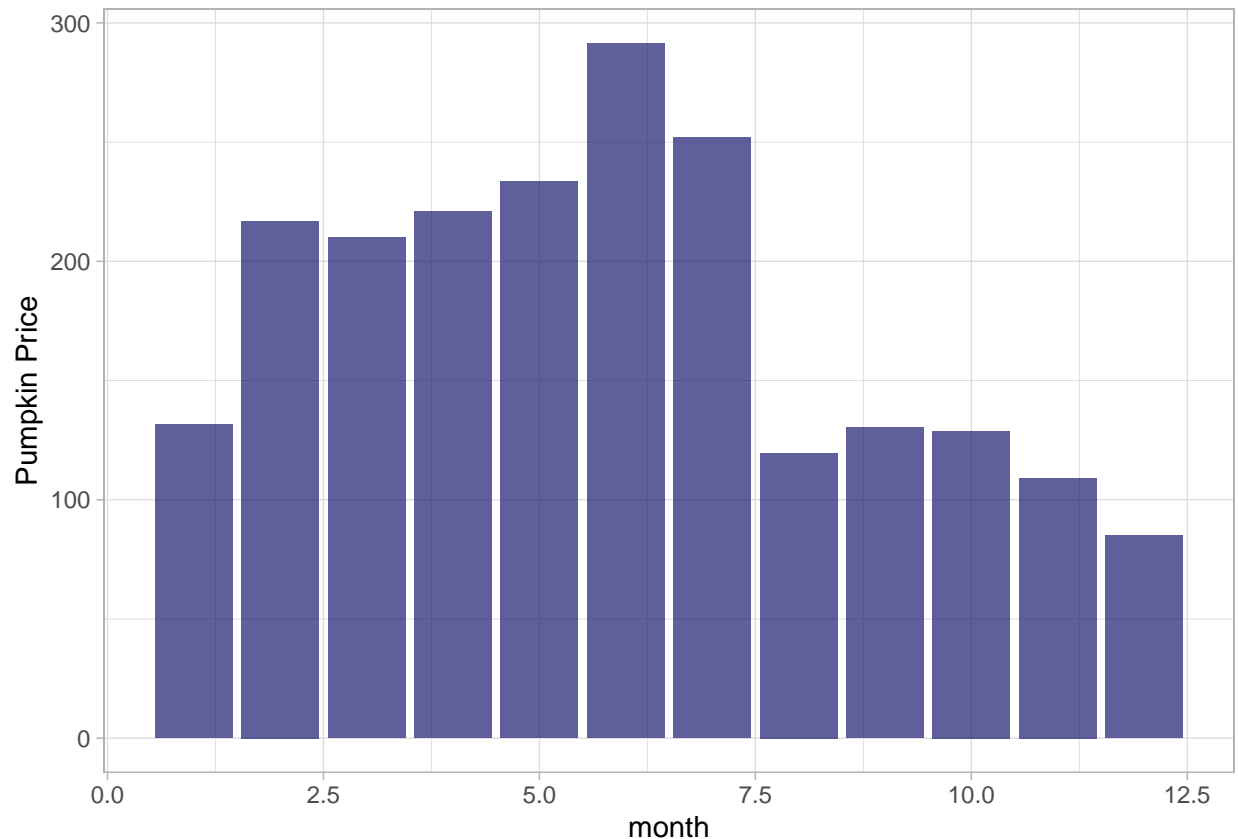
Hint: use `dplyr::group_by()` `%>% summarize()`

```
# Find the average price of pumpkins per month
new_pumpkins %>%
  group_by(month) %>%
  summarize(mean_price = mean(price)) %>%
  ggplot(aes(x = month, y = mean_price)) +
  geom_col(fill = "orange", alpha = 0.7) +
  ylab("Pumpkin Price")
```



```
# 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")
```





## Pre-processing data for modelling using recipes

What if we wanted to predict the price of a pumpkin based on the city or package columns which are of type character? How could we find the correlation between, say, package and price?

Machine learning models work best with numeric features rather than text values, so you generally need to convert categorical features into numeric representations.

This means that we have to find a way to reformat our predictors to make them easier for a model to use effectively, a process known as **feature engineering**.

Different models have different preprocessing requirements. For instance, least squares requires encoding categorical variables such as month, variety and city\_name. This simply involves translating a column with categorical values into one or more numeric columns that take the place of the original.

Now let's introduce another useful tidymodels package: recipes - which will help you preprocess data before training your model. A recipe is an object that defines what steps should be applied to a data set in order to get it ready for modelling.

Now, let's create a recipe that prepares our data for modelling by substituting a unique integer for all the observations in the predictor columns:

```
# Specify a recipe
pumpkins_recipe <- recipe(price ~ ., # specify outcome
                           data = new_pumpkins) %>%
  step_integer(all_predictors(),
              zero_based = TRUE)
```

```
# Print out the recipe
pumpkins_recipe
```

```
##
```

```
## -- Recipe -----
```

```
##
```

```
## -- Inputs
```

```
## Number of variables by role
```

```
## outcome: 1
```

```
## predictor: 8
```

```
##
```

```
## -- Operations
```

```
## * Integer encoding for: all_predictors()
```

OK, we created our first recipe that specifies an outcome (price) and its corresponding predictors and that all the predictor columns should be encoded into a set of integers. Let's quickly break it down:

The call to `recipe()` with a formula tells the recipe the roles of the variables using `new_pumpkins` data as the reference. For instance the price column has been assigned an outcome role while the rest of the columns have been assigned a predictor role.

`step_integer(all_predictors(), zero_based = TRUE)` specifies that all the predictors should be converted into a set of integers with the numbering starting at 0.

How can we confirm that the recipe is doing what we intend? Once your recipe is defined, you can estimate the parameters required to preprocess the data, and then extract the processed data. You don't typically need to do this when you use `tidymodels` (we'll see the normal convention in just a minute with workflows) but it's a good sanity check for confirming that recipes are doing what you expect.

For that, you'll need two more verbs: `prep()` and `bake()`

`prep()`: estimates the required parameters from a training set that can be later applied to other data sets.

`bake()`: takes a prepped recipe and applies the operations to any data set.

Now let's prep and bake our recipes to confirm that under the hood, the predictor columns will be first encoded before a model is fit.

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

# Bake the recipe to extract a preprocessed new_pumpkins data
baked_pumpkins <- bake(pumpkins_prep, # use the prepped version
                      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
```

The processed data `baked_pumpkins` has all its predictors encoded confirming that indeed the preprocessing steps defined as our recipe will work as expected. This makes it harder for you to read but more intelligible for tidymodels. Take a look at how the observations have been mapped to numbers.

**Question 5: From looking at the `baked_pumpkins` tibble, how many total cities are represented in the data set?**

There are 10 total cities that are represented in the data set. Each of the cities has been assigned a value from 0 to 9.

`baked_pumpkins` is a data frame that we can perform computations on. For instance, let's try to find a good correlation between two variables to potentially build a good predictive model. We'll use the function `cor()` to do this.

```
# Find the correlation between the package and the price
cor(baked_pumpkins$package, baked_pumpkins$price)
```

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

**Question 6: Calculate the correlation between pumpkin price and two other variables in the data set**

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

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

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

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

**Question 7: Which of these three variables is most highly correlated with price? Why might this be?**

The variety is most highly correlated with price. This might be because variety has to do with the size of the pumpkin, and it makes sense that the sizes of the pumpkin will affect the price.

Now let's visualize a correlation matrix of all the columns using the `corrplot` package.

```

# Load the corrplot package
library(corrplot)

# 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 = "black",
  cl.pos = "n",
  order = "original")

```

	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

## Build a linear regression model

Now that we have build a recipe, and actually confirmed that the data will be pre-processed appropriately, let's now build a regression model to answer the question: What price can I expect of a given pumpkin package?

## Train the model using the training set

As you may have already figured out, the column price is the outcome variable while the package column is the predictor variable.

To do this, we'll first split the data. Data splitting is a key part of the machine learning process. For now we'll do a 80/20 split, where 80% of the data goes into training and 20% into the test set. Then we'll define a recipe that will encode the predictor column into a set of integers, then build a model specification. We won't prep and bake our recipe since we already know it will preprocess the data as expected.

```
set.seed(123)

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

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

# Create a recipe for pre-processing the data
lm_pumpkins_recipe <- recipe(price ~ package, # equation is outcome ~ predictor variable
                             data = pumpkins_train) %>%
  step_integer(all_predictors(), zero_based = TRUE)

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

Now that we have a recipe and a model specification, we need to find a way of bundling them together into an object that will first preprocess the data (prep+bake behind the scenes), fit the model on the preprocessed data and also allow for potential post-processing activities.

So let's bundle everything up into a workflow. A workflow is a container object that aggregates information required to fit and predict from a model.

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

# 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
```

A workflow can be fit/trained in much the same way a model can.

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

# 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
```

From the model output, we can see the coefficients learned during training. They represent the coefficients of the line of best fit that gives us the lowest overall error between the actual and predicted variable.

Evaluate model performance using the test set. It's time to see how the model performed! How do we do this?

Now that we've trained the model, we can use it to make predictions for the test\_set using `parsnip::predict()`. Then we can compare these predictions to the actual label values to evaluate how well (or not!) the model is working.

Let's start with making predictions for the test set then bind the columns to the test set.

```
# Make predictions for the test set
predictions <- lm_wf_fit %>% # use trained model specified above
  predict(new_data = pumpkins_test) # use testing data

# Bind predictions to the test set
lm_results <- pumpkins_test %>% # use test set
```

```

select(c(package, price)) %>% # select the predictor and outcome columns
bind_cols(predictions) # make new prediction column

# 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

```

OK, you have just trained a model and used it to make predictions! Let's evaluate the model's performance.

In tidymodels, we do this using `yardstick::metrics()`. For linear regression, let's focus on the following metrics:

**Root Mean Square Error (RMSE):** The square root of the MSE. This yields an absolute metric in the same unit as the label (in this case, the price of a pumpkin). The smaller the value, the better the model (in a simplistic sense, it represents the average price by which the predictions are wrong)

**Coefficient of Determination (usually known as R-squared or R2):** A relative metric in which the higher the value, the better the fit of the model. In essence, this metric represents how much of the variance between predicted and actual label values the model is able to explain.

```

# Evaluate performance of linear regression
metrics(data = lm_results,
  truth = price, # variable that is being predicted
  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

```

OK, so that is the model performance. Let's see if we can get a better indication by visualizing a scatter plot of the package and price then use the predictions made to overlay a line of best fit.

This means we'll have to prep and bake the test set in order to encode the package column then bind this to the predictions made by our model.

```

# Encode package column
package_encode <- lm_pumpkins_recipe %>% # recipe for pre-processing the data
  prep() %>%
  bake(new_data = pumpkins_test) %>% # use test data
  select(package) # select column

# Bind encoded package column to the results
plot_results <- lm_results %>% # use bound predictions / test set
  bind_cols(package_encode %>%
    rename(package_integer = package)) %>% # add columns from package_encode df and rename col
  relocate(package_integer, .after = package) # reorder columns

# 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

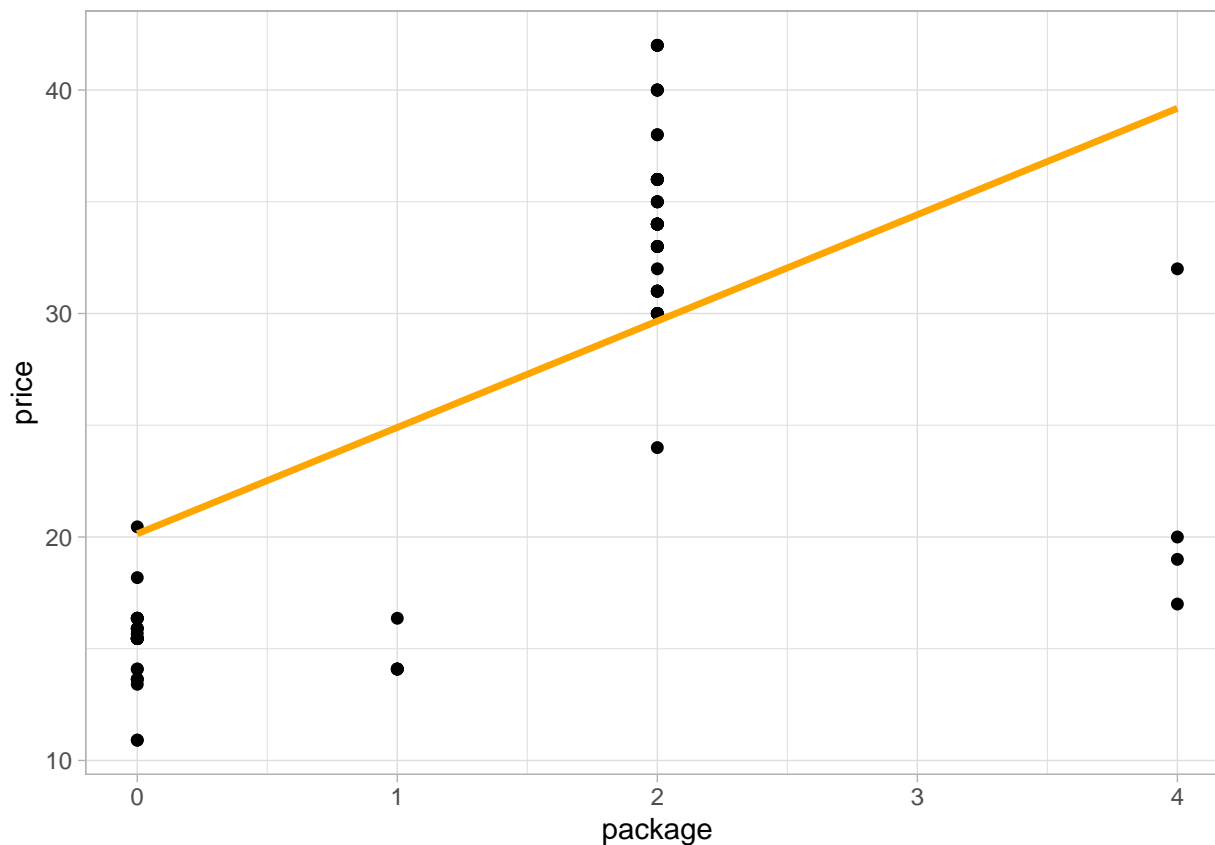
```

```

# 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")

```





Hmm. The model does not do good job of generalizing the relationship between a package and its corresponding price.

**Question 8:** What issues do you see with fitting a linear regression to this data?

Fitting a line to this data is not great because the data is categorical, and this model is underfitting the data. There are a lot of points that are not represented by the model, which means that using this model would not lead to very accurate predictions for prices based on the package the pumpkins come in.

Congratulations, you just created a model that can help predict the price of a few varieties of pumpkins. But you can probably create a better model! To be continued next week...