Lab 1

Lewis White

2023-01-18

##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 so as to ask questions of it.

When is the best time to buy pumpkins?

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, 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 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

```r
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
```

```r
## Load lubridate
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

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

```r
# 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
## # ... with 1,747 more rows
## # i Use `print(n = ...)` to see more rows
```

```r
## 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
```

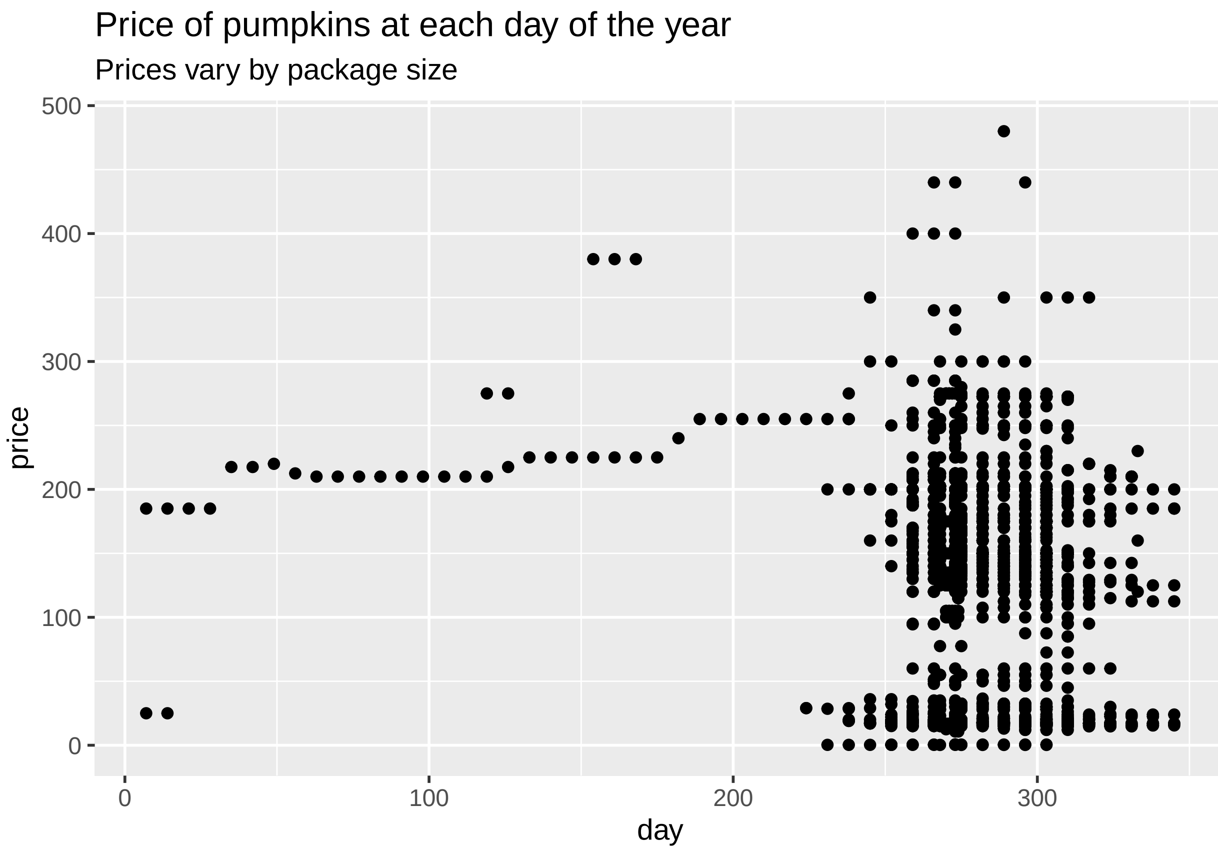
There are two column dealing with price, high and low. Let's combine them into a single average price column.

```r
# Create a new column price
pumpkins <- pumpkins %>%
  mutate(price = (low_price+ high_price)/2)
```

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.

```r
ggplot(data = pumpkins, mapping = aes(x = day, y = price)) +
  geom_point() +
  labs(title = "Price of pumpkins at each day of the year",
       subtitle = "Prices vary by package size")
```



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".

Question 2 In the first section of the chunk below, 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"))

# 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_p~1 high_~2 date         day month price
##    <chr>    <chr>     <chr>           <dbl>   <dbl> <date>     <dbl> <dbl> <dbl>
##  1 PIE TYPE BALTIMORE 1 1/9 bushel~      15    15   2016-09-24   268     9  15
##  2 PIE TYPE BALTIMORE 1 1/9 bushel~      18    18   2016-09-24   268     9  18
##  3 PIE TYPE BALTIMORE 1 1/9 bushel~      18    18   2016-10-01   275    10  18
##  4 PIE TYPE BALTIMORE 1 1/9 bushel~      17    17   2016-10-01   275    10  17
##  5 PIE TYPE BALTIMORE 1 1/9 bushel~      15    15   2016-10-08   282    10  15
##  6 PIE TYPE BALTIMORE 1 1/9 bushel~      18    18   2016-10-08   282    10  18
##  7 PIE TYPE BALTIMORE 1 1/9 bushel~      17    17   2016-10-08   282    10  17
##  8 PIE TYPE BALTIMORE 1 1/9 bushel~      17    18.5 2016-10-08   282    10  17.8
##  9 PIE TYPE BALTIMORE 1 1/9 bushel~      15    15   2016-10-15   289    10  15
## 10 PIE TYPE BALTIMORE 1 1/9 bushel~      17    17   2016-10-15   289    10  17
## # ... with abbreviated variable names 1: low_price, 2: high_price
```

You can see that we have narrowed down to 415 rows of data containing pumpkins by the bushel.

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 vectorise 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),
    str_detect(package, "1/2") ~ price*2,
    TRUE ~ price))

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

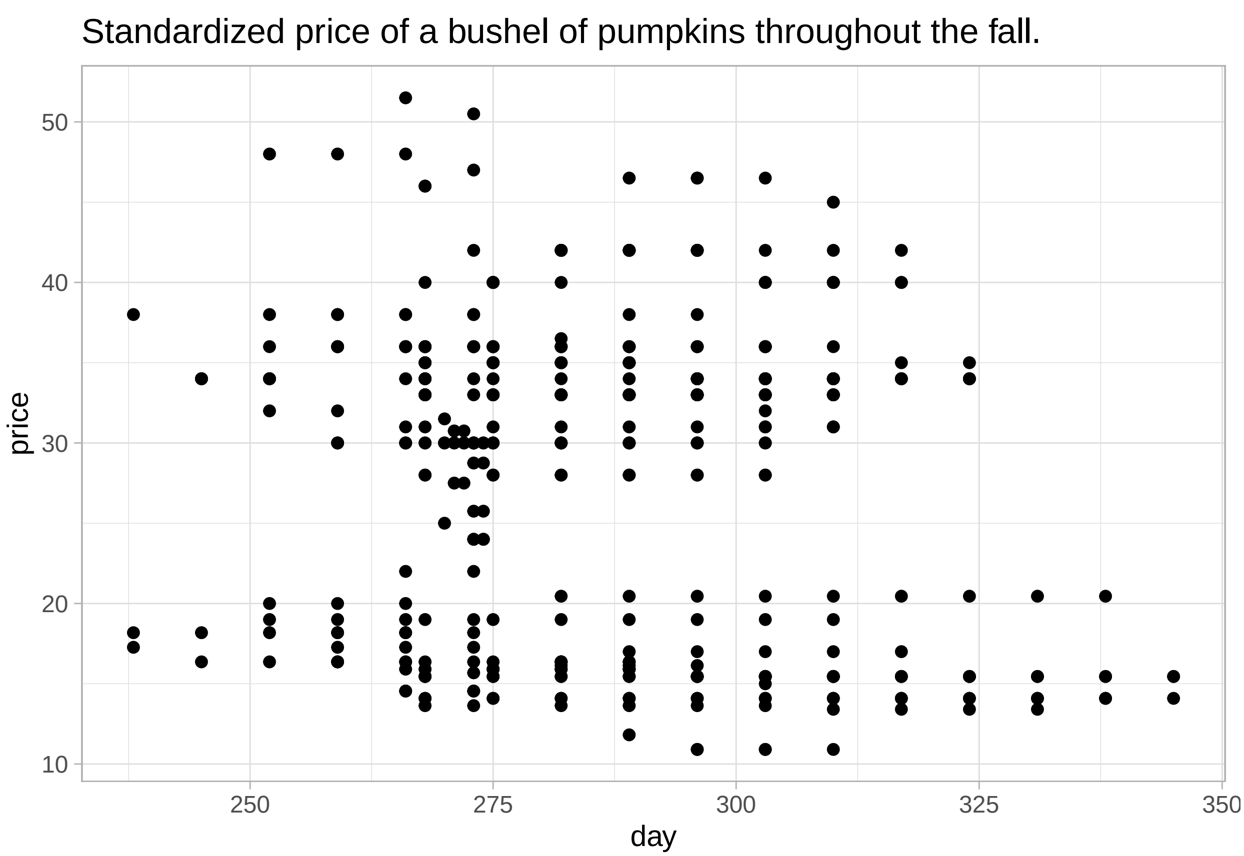
```
## # A tibble: 30 x 9
##    variety  city_name package       low_p~1 high_~2 date         day month price
```

```
##     <chr>     <chr>      <chr>              <dbl>    <dbl> <date>      <dbl> <dbl> <dbl>
##   1 PIE TYPE BALTIMORE 1 1/9 bushel~      15     15    2016-09-24    268     9  13.6
##   2 PIE TYPE BALTIMORE 1 1/9 bushel~      18     18    2016-09-24    268     9  16.4
##   3 PIE TYPE BALTIMORE 1 1/9 bushel~      18     18    2016-10-01    275    10  16.4
##   4 PIE TYPE BALTIMORE 1 1/9 bushel~      17     17    2016-10-01    275    10  15.5
##   5 PIE TYPE BALTIMORE 1 1/9 bushel~      15     15    2016-10-08    282    10  13.6
##   6 PIE TYPE BALTIMORE 1 1/9 bushel~      18     18    2016-10-08    282    10  16.4
##   7 PIE TYPE BALTIMORE 1 1/9 bushel~      17     17    2016-10-08    282    10  15.5
##   8 PIE TYPE BALTIMORE 1 1/9 bushel~      17     18.5 2016-10-08    282    10  16.1
##   9 PIE TYPE BALTIMORE 1 1/9 bushel~      15     15    2016-10-15    289    10  13.6
## 10 PIE TYPE BALTIMORE 1 1/9 bushel~      17     17    2016-10-15    289    10  15.5
## # ... with 20 more rows, and abbreviated variable names 1: low_price,
## #   2: high_price
## # i Use `print(n = ...)` to see more rows
```

Data Visualization

```r
# Set theme
theme_set(theme_light())

# Make a scatter plot of month and price
new_pumpkins %>%
  ggplot(mapping = aes(x = day, y = price)) +
  geom_point(size = 1.6) +
  labs(title = "Standardized price of a bushel of pumpkins throughout the fall.")
```

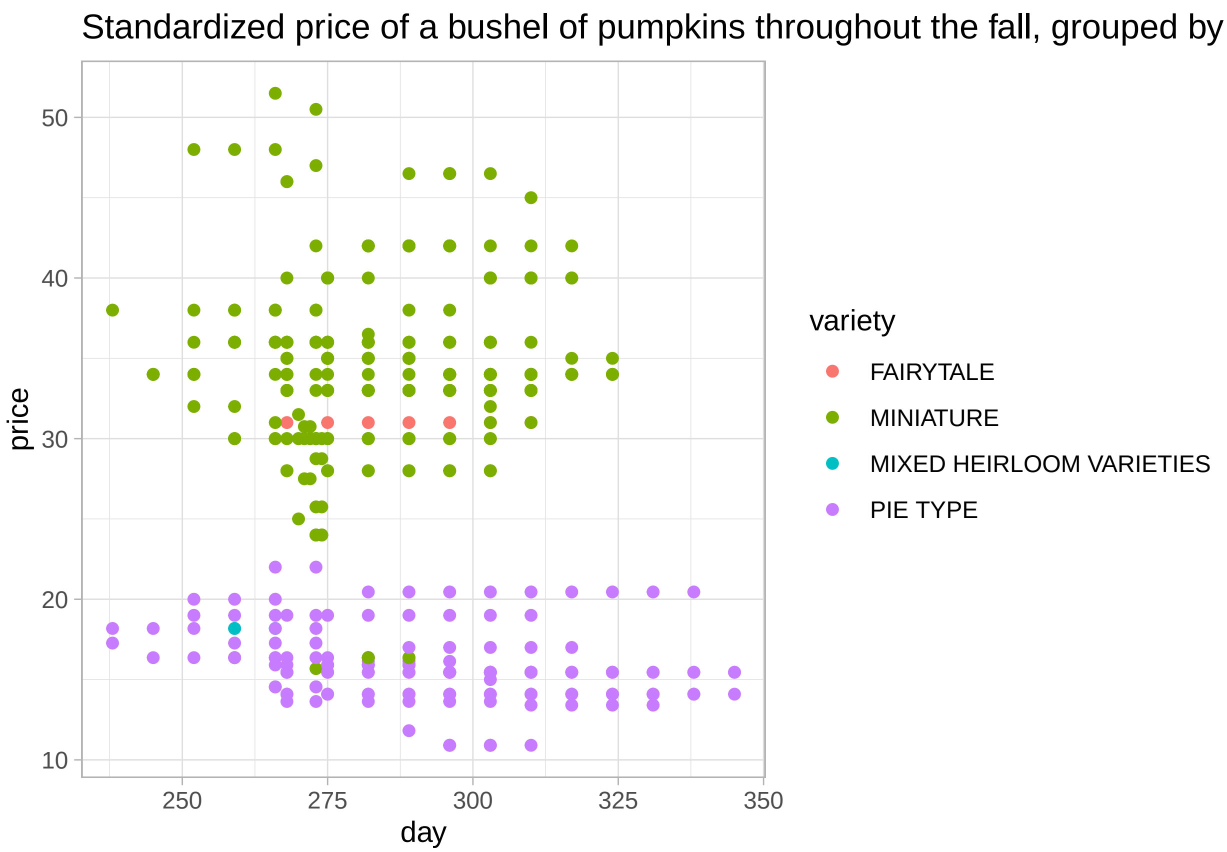


Question 3: Is this a useful plot? Does anything about it surprise you?

This plot is mildly useful in the sense that it communicates price per bushel at various days of the year; however, it would be more useful to update the visualization to show why there appears to be at least two distinct groups with different price points.

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

```
# Make a scatter plot of month and price, grouped by variety
new_pumpkins %>%
  ggplot(mapping = aes(x = day, y = price, color = variety)) +
  geom_point(size = 1.6) +
  labs(title = "Standardized price of a bushel of pumpkins throughout the fall, grouped by pumpkin vari
```



Question 4: Within new_pumpkins, group the pumpkins into groups based on the month column and then find the mean price for each month (in the next chunk).

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

```
# Find the average price of pumpkins per month
new_pumpkins %>%
  group_by(month) %>%
  summarise(mean_price = mean(price))
```

```
## # A tibble: 5 x 2
##   month mean_price
##   <dbl>      <dbl>
## 1     8       24.5
## 2     9       28.6
## 3    10       28.4
```

```
## 4     11        25.1
## 5     12        15.6
```

Question 5: Now do that again, but continue on and plot the results with a bar plot

```
# Find the average price of pumpkins per month then plot a bar chart
pumpkin_month_summary <- new_pumpkins %>%
  group_by(month) %>%
  summarise(mean_price = mean(price))

ggplot(pumpkin_month_summary, aes(x = month, y = mean_price)) +
  geom_col(fill = "#bf440b") +
  labs(title = "Average price per bushel of a pumpkin",
       y = "Mean bushel price")
```

#Preprocessing 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 mode. 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 ~ ., data = new_pumpkins) %>%
  step_integer(all_predictors(), zero_based = TRUE)

# Print out the recipe
pumpkins_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    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 its 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, 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
##      <dbl>     <dbl>   <dbl>     <dbl>      <dbl> <dbl> <dbl> <dbl> <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 6: From looking at the baked_pumpkins tibble, how many total cities are represented in the data set?

```
length(unique(baked_pumpkins$city_name))
```

```
## [1] 10
```

10 cities appear to be represented in the data set.

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) #0.4944854
```

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

Question 7: Calculate the correlation between pumpkin price and two other variables in the data set

```
#variety and price
cor(baked_pumpkins$variety, baked_pumpkins$price) #-0.8719442
```

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

```
cor(baked_pumpkins$day, baked_pumpkins$price) #-0.1178958
```

```
## [1] -0.1245279
```

Question 8: Which of these three variables is most highly correlated with price? Why might this be?

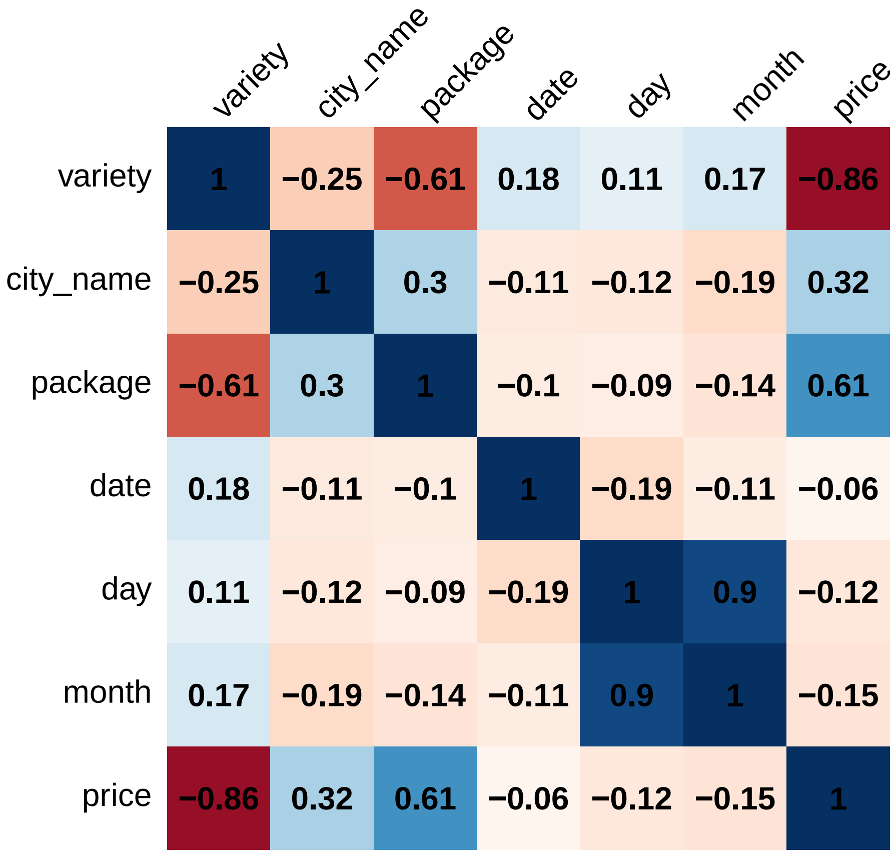
Of the three variables I explored, variety was most strongly correlated with price. This makes sense, as different types of pumpkins may be more desirable to consumers.

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 = "blac
```



Build a linear regression model

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

#Train a linear regression 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 <- new_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
```

```r
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")
```

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.

```r
# Hold modelling 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.

```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
```

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 %>%
  predict(new_data = pumpkins_test)

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

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

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

```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
```

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.

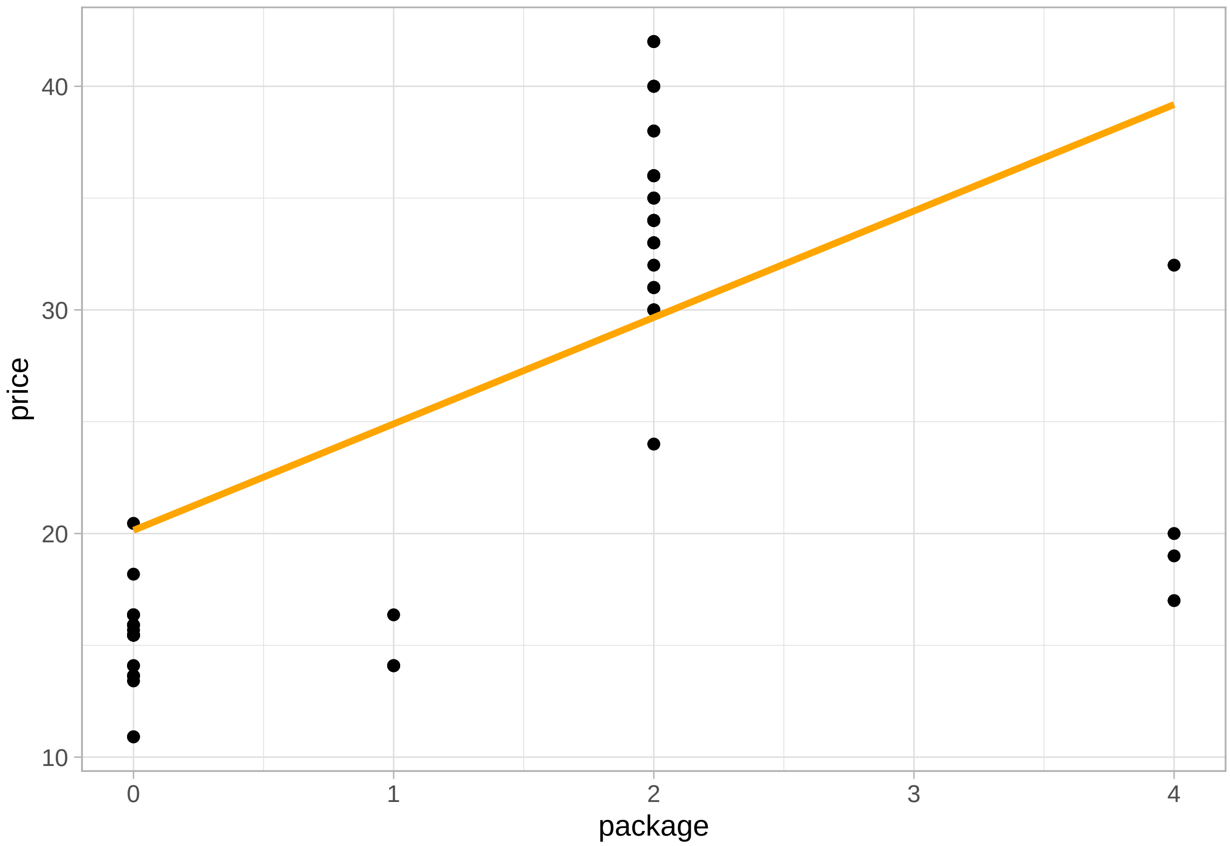
```r
# Encode package column
package_encode <- lm_pumpkins_recipe %>%
  prep() %>%
  bake(new_data = pumpkins_test) %>%
  select(package)

# 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
##   <chr>                          <dbl> <dbl> <dbl>
## 1 1 1/9 bushel cartons               0  13.6  20.1
## 2 1 1/9 bushel cartons               0  16.4  20.1
## 3 1 1/9 bushel cartons               0  16.4  20.1
## 4 1 1/9 bushel cartons               0  13.6  20.1
## 5 1 1/9 bushel cartons               0  15.5  20.1
```

```r
# 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", size = 1.2) +
   xlab("package")
```



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

Question 9 What issues do you see with fitting a linear regression to this data?

The model is trying to model categorical data as if it is numeric. Since the pumpkin packages don't have a natural ordering, assigning values between 1 and 5 and having these numbers indicate an order in the model is making it less accurate than if they were treated as factors.

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!