Exploring Data with R

Scott Bailey

5/1/2020

Exploring Data with R

May 5, 2020

Instructors: - Scott Bailey - Natalia Lopez

Approach and Learning Goals

We're going to take a very pragmatic approach in our workshop today. Instead of thinking about R as a programming language we're trying to learn to code in, we're going to approach R as just a tool that we can use to do some fundamental data exploration and analysis. In that line of thinking, instead of learning a lot of base R, and instead of building up an understanding of syntax and semantics according to fundamental coding concepts, we're going to start using the tidyverse packages and approach right away in an applied manner. Briefly, tidyverse is a set of opinionated packages with a coherent approach to how to work with data in R. For many of us, they make R easier to learn and to use.

We'll use tidyverse to do a few specific things today, which will give us a general data exploration and analysis workflow:

1. Import data
2. Get summary statistics on data
3. Pre-process/clean data
4. Visualize data
5. Fit a model to some of our data

Our goal today isn't to get through every single line of code or every idea. Most importantly, I want you to have the experience of writing code with data and making things happen.

Orientation to RStudio

RStudio is a full-featured integrated development environment for R. Let's take a brief tour through RStudio's interface, including a look at the `Preferences`.

Creating a new R script

There are several different formats of R scripts, including RMarkdown files and R Notebooks. Just to stay focused on learning and writing R code today, we're going to stick with the straightforward `.R` scripts.

- Create new `.R` script and name it `exploration.R`.

Installing and importing packages

By default, you will have base R and and all it's capacity in your script. In order to use the tidyverse packages, and any other of the many great R packages, you'll need to install them for your version of R, and then import them into your script.

```
install.packages(c("tidyverse",
                   "forcats",
                   "lubridate",
                   "skimr",
                   "mgcv",
                   "gratia"))
```

```
library(tidyverse, warn.conflicts = FALSE)
library(forcats)
library(lubridate)
library(skimr)
library(mgcv)
library(gratia)
```

Importing Data

The data you provided for this workshop is anonymized sales related data for a major big-box store. There are a number of files, documented in the Readme file I sent out along with the data. We're going to focus on two particular files, and import both right here at the beginning.

```
features_df <- read_csv("data/features.csv")
train_df <- read_csv("data/train.csv")
```

Once you've run these lines of code, you should see new data shown in the `Environment` pane, along with the dimensions of the data. We can open up this data and view it as a table, but let's hold off on doing that and instead explore it a bit with R code.

Getting an overview of our data

There's a few different ways that we can get a high-level sense of our data. We'll start with the `glimpse` function from the tidyverse and start wth the data stored in `features_df`.

```
glimpse(features_df)
```

```
## Rows: 8,190
## Columns: 12
## $ Store        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ Date         <date> 2010-02-05, 2010-02-12, 2010-02-19, 2010-02-26, 2010-...
## $ Temperature  <dbl> 42.31, 38.51, 39.93, 46.63, 46.50, 57.79, 54.58, 51.45...
## $ Fuel_Price   <dbl> 2.572, 2.548, 2.514, 2.561, 2.625, 2.667, 2.720, 2.732...
## $ MarkDown1    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ MarkDown2    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ MarkDown3    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ MarkDown4    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ MarkDown5    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
## $ CPI          <dbl> 211.0964, 211.2422, 211.2891, 211.3196, 211.3501, 211....
## $ Unemployment <dbl> 8.106, 8.106, 8.106, 8.106, 8.106, 8.106, 8.106, 8.106...
## $ IsHoliday    <lgl> FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,...
```

We're able to see how many rows we have, how many columns or variables, their data types, and the first values in those columns. The data being shown organized by columns is a hint here; R works most effectively when you think about operating on columns of data.

If we want to take a bit deeper look while still staying at a high level, we can use the `summary` function from base R.

```
summary(features_df)
```

```
##      Store         Date             Temperature       Fuel_Price
##  Min.   : 1    Min.   :2010-02-05   Min.   : -7.29   Min.   :2.472
##  1st Qu.:12    1st Qu.:2010-12-17   1st Qu.: 45.90   1st Qu.:3.041
##  Median :23    Median :2011-10-31   Median : 60.71   Median :3.513
##  Mean   :23    Mean   :2011-10-31   Mean   : 59.36   Mean   :3.406
##  3rd Qu.:34    3rd Qu.:2012-09-14   3rd Qu.: 73.88   3rd Qu.:3.743
##  Max.   :45    Max.   :2013-07-26   Max.   :101.95   Max.   :4.468
##
##    MarkDown1        MarkDown2           MarkDown3           MarkDown4
##  Min.   : -2781   Min.   :  -265.76   Min.   :  -179.26   Min.   :    0.22
##  1st Qu.:  1578   1st Qu.:    68.88   1st Qu.:     6.60   1st Qu.:  304.69
##  Median :  4744   Median :   364.57   Median :    36.26   Median : 1176.42
##  Mean   :  7032   Mean   :  3384.18   Mean   :  1760.10   Mean   : 3292.94
##  3rd Qu.:  8923   3rd Qu.:  2153.35   3rd Qu.:   163.15   3rd Qu.: 3310.01
##  Max.   :103185   Max.   :104519.54   Max.   :149483.31   Max.   :67474.85
##  NA's   :4158     NA's   :5269        NA's   :4577        NA's   :4726
##    MarkDown5             CPI         Unemployment    IsHoliday
##  Min.   :  -185.2   Min.   :126.1   Min.   : 3.684   Mode :logical
##  1st Qu.:  1440.8   1st Qu.:132.4   1st Qu.: 6.634   FALSE:7605
##  Median :  2727.1   Median :182.8   Median : 7.806   TRUE :585
##  Mean   :  4132.2   Mean   :172.5   Mean   : 7.827
##  3rd Qu.:  4832.6   3rd Qu.:213.9   3rd Qu.: 8.567
##  Max.   :771448.1   Max.   :229.0   Max.   :14.313
##  NA's   :4140       NA's   :585     NA's   :585
```

This isn't bad - we're getting quartile breakdowns of numeric variables now. We can do a bit better with the `skim` function from the `skimr` package though, and it's my go to when I just need to dig into some data.

```
skim(features_df)
```

Table 1: Data summary

Name	features_df
Number of rows	8190
Number of columns	12
Column type frequency:	
Date	1
logical	1
numeric	10
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Date	0	1	2010-02-05	2013-07-26	2011-10-31	182

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
IsHoliday	0	1	0.07	FAL: 7605, TRU: 585

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Store	0	1.00	23.00	12.99	1.00	12.00	23.00	34.00	45.00
Temperature	0	1.00	59.36	18.68	-7.29	45.90	60.71	73.88	101.95
Fuel_Price	0	1.00	3.41	0.43	2.47	3.04	3.51	3.74	4.47
MarkDown1	4158	0.49	7032.37	9262.75	-2781.45	1577.53	4743.58	8923.31	103184.98
MarkDown2	5269	0.36	3384.18	8793.58	-265.76	68.88	364.57	2153.35	104519.54
MarkDown3	4577	0.44	1760.10	11276.46	-179.26	6.60	36.26	163.15	149483.31
MarkDown4	4726	0.42	3292.94	6792.33	0.22	304.69	1176.42	3310.01	67474.85
MarkDown5	4140	0.49	4132.22	13086.69	-185.17	1440.83	2727.14	4832.56	771448.10
CPI	585	0.93	172.46	39.74	126.06	132.36	182.76	213.93	228.98
Unemployment	585	0.93	7.83	1.88	3.68	6.63	7.81	8.57	14.31

What do get here that we didn't before?

Let's pay attention to the types of our variables. R does it's best to figure out the data types, but I notice one that seems not quite right to me: Store. Right now it's a numeric, but each of these numbers is really a unique identifier for a store. As such, we can consider them categorical data. Let's go ahead and convert that column to the factor data type.

```r
features_df$Store <- as_factor(features_df$Store)

skim(features_df$Store)
```

Table 5: Data summary

Name	features_df$Store
Number of rows	8190
Number of columns	1
Column type frequency:	
factor	1
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
data	0	1	FALSE	45	1: 182, 2: 182, 3: 182, 4: 182

We had been using these overview functions - `summary`, `glimpse`, and `skim` - on the whole dataframe or tibble, but we can also check out single columns.

Subsetting data

What if we want to narrow down our data of interest rather than looking at the whole 8000+ observations? The tidyverse provides a set of functions for common subsetting tasks. We'll start with `filter`.

Let's get back just the data for store 1.

```
filter(features_df, Store == 1)
```

```
## # A tibble: 182 x 12
##    Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##    <fct> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
##  1 1     2010-02-05        42.3       2.57        NA        NA        NA
##  2 1     2010-02-12        38.5       2.55        NA        NA        NA
##  3 1     2010-02-19        39.9       2.51        NA        NA        NA
##  4 1     2010-02-26        46.6       2.56        NA        NA        NA
##  5 1     2010-03-05        46.5       2.62        NA        NA        NA
##  6 1     2010-03-12        57.8       2.67        NA        NA        NA
##  7 1     2010-03-19        54.6       2.72        NA        NA        NA
##  8 1     2010-03-26        51.4       2.73        NA        NA        NA
##  9 1     2010-04-02        62.3       2.72        NA        NA        NA
## 10 1     2010-04-09        65.9       2.77        NA        NA        NA
## # ... with 172 more rows, and 5 more variables: MarkDown4 <dbl>,
## #   MarkDown5 <dbl>, CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

We haven't stored this subset anywhere yet, just printed it out to the standard output. We'll get to saving the data into a new variable shortly.

The `filter` function takes at least two arguments: the dataframe you want to filter, and a condition to determine what to include. Here's our condition is where the value in the `Store` variable is 1.

There's a more common way to use these "verbs" in tidyverse, though, that let's us chain together multiple actions. In the spirit of Unix, we have `pipes`, unidirectional data flows, which use this character: `%>%`. Let's see it in action.

```
features_df %>%
  filter(Store == 1)
```

```
## # A tibble: 182 x 12
##    Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##    <fct> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
##  1 1     2010-02-05        42.3       2.57        NA        NA        NA
##  2 1     2010-02-12        38.5       2.55        NA        NA        NA
##  3 1     2010-02-19        39.9       2.51        NA        NA        NA
##  4 1     2010-02-26        46.6       2.56        NA        NA        NA
##  5 1     2010-03-05        46.5       2.62        NA        NA        NA
##  6 1     2010-03-12        57.8       2.67        NA        NA        NA
##  7 1     2010-03-19        54.6       2.72        NA        NA        NA
##  8 1     2010-03-26        51.4       2.73        NA        NA        NA
##  9 1     2010-04-02        62.3       2.72        NA        NA        NA
## 10 1     2010-04-09        65.9       2.77        NA        NA        NA
## # ... with 172 more rows, and 5 more variables: MarkDown4 <dbl>,
## #   MarkDown5 <dbl>, CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

The two code blocks above are the same, but you'll more often see the second version. When we use a pipe, we send the data in the variable in front of the pipe to the function after the pipe. We pipe the data into the function. This data goes into the function as the first parameter, and R knows that what we then put into the function are the rest of the parameters, such as the condition.

If that doesn't make sense yet, don't worry. We're going to keep using it, and we'll build up our knowledge with practice.

Let's run that code again, but save the filtered dataset we get back this time into a new variable, `features_store_1`.

```r
features_store_1 <- features_df %>%
  filter(Store == 1)

skim(features_store_1)
```

Table 7: Data summary

Name	features_store_1
Number of rows	182
Number of columns	12
Column type frequency:	
Date	1
factor	1
logical	1
numeric	9
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Date	0	1	2010-02-05	2013-07-26	2011-10-31	182

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Store	0	1	FALSE	1	1: 182, 2: 0, 3: 0, 4: 0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
IsHoliday	0	1	0.07	FAL: 169, TRU: 13

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	his
Temperature	0	1.00	66.91	14.13	35.40	55.32	67.51	79.86	91.65	
Fuel_Price	0	1.00	3.26	0.39	2.51	2.87	3.35	3.57	3.91	
MarkDown1	92	0.49	8536.59	9380.36	332.17	3667.81	6170.16	10262.68	72937.29	
MarkDown2	109	0.40	3346.40	7976.08	0.50	92.00	292.10	3163.89	46011.38	
MarkDown3	93	0.49	1670.80	9829.74	0.10	9.80	47.21	209.62	74910.32	
MarkDown4	92	0.49	3653.63	5708.01	8.00	454.50	1627.55	3800.23	32403.87	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	his
MarkDown5	92	0.49	4428.31	2933.24	303.32	2672.58	3736.91	5898.05	20475.32	
CPI	13	0.93	217.27	5.00	210.34	211.75	216.04	221.88	225.17	
Unemployment	13	0.93	7.44	0.53	6.31	6.91	7.74	7.84	8.11	

We can combine filter conditions by chaining together filter statements or just adding multiple conditions to a single filter function.

```
features_df %>%
  filter(Store == 1 & Date > as_date("2012-01-01"))
```

```
## # A tibble: 82 x 12
##    Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##    <fct> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
##  1 1     2012-01-06        49.0       3.16     6277.    21813.     143.
##  2 1     2012-01-13        48.5       3.26     5183.     8026.      42.2
##  3 1     2012-01-20        54.1       3.27     4140.     2807.      33.9
##  4 1     2012-01-27        54.3       3.29     1164.     1083.      44
##  5 1     2012-02-03        56.6       3.36    34577.     3579.     161.
##  6 1     2012-02-10        48.0       3.41    13925.     6927.     102.
##  7 1     2012-02-17        45.3       3.51     9873.    11062.       9.8
##  8 1     2012-02-24        57.2       3.56     9350.     7556.       3.2
##  9 1     2012-03-02        61.0       3.63    15441.     1569       10.8
## 10 1     2012-03-09        58.8       3.67    10331.      152.       6
## # ... with 72 more rows, and 5 more variables: MarkDown4 <dbl>,
## #   MarkDown5 <dbl>, CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

```
# is the same as

features_df %>%
  filter(Store == 1) %>%
  filter(Date > as_date("2012-01-01"))
```

```
## # A tibble: 82 x 12
##    Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##    <fct> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
##  1 1     2012-01-06        49.0       3.16     6277.    21813.     143.
##  2 1     2012-01-13        48.5       3.26     5183.     8026.      42.2
##  3 1     2012-01-20        54.1       3.27     4140.     2807.      33.9
##  4 1     2012-01-27        54.3       3.29     1164.     1083.      44
##  5 1     2012-02-03        56.6       3.36    34577.     3579.     161.
##  6 1     2012-02-10        48.0       3.41    13925.     6927.     102.
##  7 1     2012-02-17        45.3       3.51     9873.    11062.       9.8
##  8 1     2012-02-24        57.2       3.56     9350.     7556.       3.2
##  9 1     2012-03-02        61.0       3.63    15441.     1569       10.8
## 10 1     2012-03-09        58.8       3.67    10331.      152.       6
## # ... with 72 more rows, and 5 more variables: MarkDown4 <dbl>,
## #   MarkDown5 <dbl>, CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

```
# is the same as

features_store_1 %>%
  filter(Date > as_date("2012-01-01"))
```

```
## # A tibble: 82 x 12
##    Store Date       Temperature Fuel_Price MarkDown1 MarkDown2 MarkDown3
##    <fct> <date>           <dbl>      <dbl>     <dbl>     <dbl>     <dbl>
##  1 1     2012-01-06        49.0       3.16     6277.    21813.     143.
##  2 1     2012-01-13        48.5       3.26     5183.     8026.      42.2
##  3 1     2012-01-20        54.1       3.27     4140.     2807.      33.9
##  4 1     2012-01-27        54.3       3.29     1164.     1083.      44
##  5 1     2012-02-03        56.6       3.36    34577.     3579.     161.
##  6 1     2012-02-10        48.0       3.41    13925.     6927.     102.
##  7 1     2012-02-17        45.3       3.51     9873.    11062.       9.8
##  8 1     2012-02-24        57.2       3.56     9350.     7556.       3.2
##  9 1     2012-03-02        61.0       3.63    15441.     1569       10.8
## 10 1     2012-03-09        58.8       3.67    10331.      152.       6
## # ... with 72 more rows, and 5 more variables: MarkDown4 <dbl>,
## #   MarkDown5 <dbl>, CPI <dbl>, Unemployment <dbl>, IsHoliday <lgl>
```

We can also subset by columns with the `select` verb or function. We'll start with our full `features_df` data, filter it by store, then select just some columns.

```
features_df %>%
  filter(Store <= 10) %>%
  select(Store, Date, Temperature, Fuel_Price)
```

```
## # A tibble: 0 x 4
## # ... with 4 variables: Store <fct>, Date <date>, Temperature <dbl>,
## #   Fuel_Price <dbl>
```

Our code doesn't work. Any ideas why?

```
features_df %>%
  filter(Store %in% c(1,10)) %>%
  select(Store, Date, Temperature, Fuel_Price)
```

```
## # A tibble: 364 x 4
##    Store Date       Temperature Fuel_Price
##    <fct> <date>           <dbl>      <dbl>
##  1 1     2010-02-05        42.3       2.57
##  2 1     2010-02-12        38.5       2.55
##  3 1     2010-02-19        39.9       2.51
##  4 1     2010-02-26        46.6       2.56
##  5 1     2010-03-05        46.5       2.62
##  6 1     2010-03-12        57.8       2.67
##  7 1     2010-03-19        54.6       2.72
##  8 1     2010-03-26        51.4       2.73
##  9 1     2010-04-02        62.3       2.72
## 10 1     2010-04-09        65.9       2.77
## # ... with 354 more rows
```

What did we do differently that made our code work?

`filter` and `select` are very powerful, and we've just scratched the surface. We're going to keep moving, though, so we can try to get through a more full workflow.

Pre-processing or cleaning data

There are a few different things we commonly think about in pre-proccessing and cleaning data. First, we might think about missing data. Let's check how much missing data we have.

```r
features_df %>%
  skim()
```

Table 12: Data summary

Name	Piped data
Number of rows	8190
Number of columns	12
Column type frequency:	
Date	1
factor	1
logical	1
numeric	9
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Date	0	1	2010-02-05	2013-07-26	2011-10-31	182

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Store	0	1	FALSE	45	1: 182, 2: 182, 3: 182, 4: 182

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
IsHoliday	0	1	0.07	FAL: 7605, TRU: 585

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Temperature	0	1.00	59.36	18.68	-7.29	45.90	60.71	73.88	101.95
Fuel_Price	0	1.00	3.41	0.43	2.47	3.04	3.51	3.74	4.47
MarkDown1	4158	0.49	7032.37	9262.75	-2781.45	1577.53	4743.58	8923.31	103184.98
MarkDown2	5269	0.36	3384.18	8793.58	-265.76	68.88	364.57	2153.35	104519.54
MarkDown3	4577	0.44	1760.10	11276.46	-179.26	6.60	36.26	163.15	149483.31
MarkDown4	4726	0.42	3292.94	6792.33	0.22	304.69	1176.42	3310.01	67474.85
MarkDown5	4140	0.49	4132.22	13086.69	-185.17	1440.83	2727.14	4832.56	771448.10
CPI	585	0.93	172.46	39.74	126.06	132.36	182.76	213.93	228.98
Unemployment	585	0.93	7.83	1.88	3.68	6.63	7.81	8.57	14.31

The various `Markdown` columns have quite a bit of missing data, while CPI and Unemployment have some.

There are a lot of strategies for handling missing data. Since we're just doing a quick tour through R and tidyverse, right now we're just going to drop the rows with missing values in CPI and Unemployment while getting rid of the Markdown columns completely.

```
features_df_clean <- features_df %>%
  drop_na(CPI, Unemployment) %>%
  select(-starts_with("Markdown"))
```

During pre-processing, we may also create derivative columns that we might need for our analysis. In this case, let's break the Date column up, and create columns for Month, Day, Week, and Year. You could treat the dates as strings, and split the date on the hyphen character. It's better to take advantage of R knowing that the Date column has data of a Date type though. We'll use functions from the great lubridate library to create these columns.

```
features_df_clean <- features_df_clean %>%
  mutate(
    Day = day(Date),
    Month = month(Date),
    Year = year(Date),
    Week = week(Date),
  )

head(features_df_clean)
```

```
## # A tibble: 6 x 11
##   Store Date       Temperature Fuel_Price   CPI Unemployment IsHoliday   Day
##   <fct> <date>           <dbl>      <dbl> <dbl>        <dbl> <lgl>     <int>
## 1 1     2010-02-05        42.3       2.57  211.         8.11 FALSE         5
## 2 1     2010-02-12        38.5       2.55  211.         8.11 TRUE         12
## 3 1     2010-02-19        39.9       2.51  211.         8.11 FALSE        19
## 4 1     2010-02-26        46.6       2.56  211.         8.11 FALSE        26
## 5 1     2010-03-05        46.5       2.62  211.         8.11 FALSE         5
## 6 1     2010-03-12        57.8       2.67  211.         8.11 FALSE        12
## # ... with 3 more variables: Month <dbl>, Year <dbl>, Week <dbl>
```

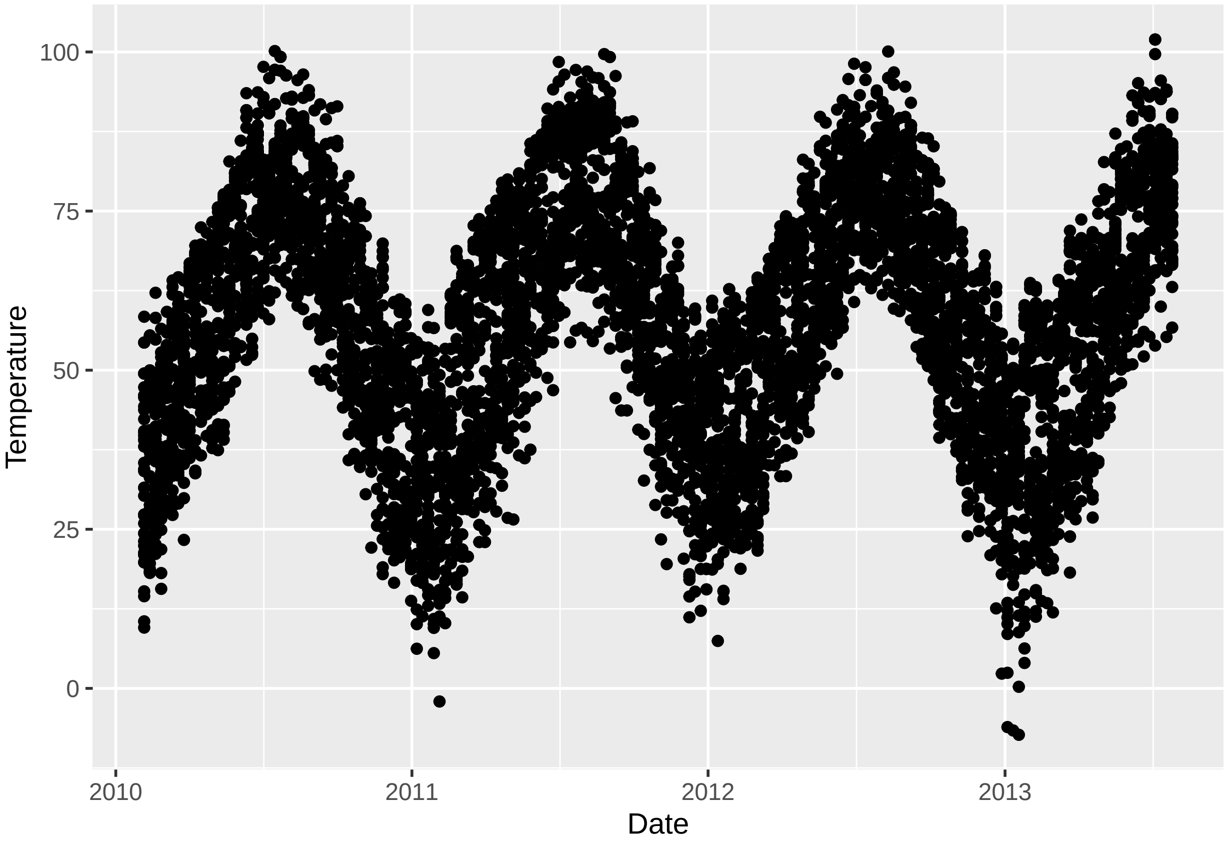
That's all we're going to do for now with pre-processing, but it's good to think about how flexible this pattern is with mutate.

Visualizing Data

We've been looking at our data in tables so far, but let's switch to using visualization to get some sense of the data. The most popular plotting tool in R these days is ggplot2, which is part of the tidyverse. It is structured by the idea of the Grammar of Graphics, and allows us to build up visualizations layer by layer.

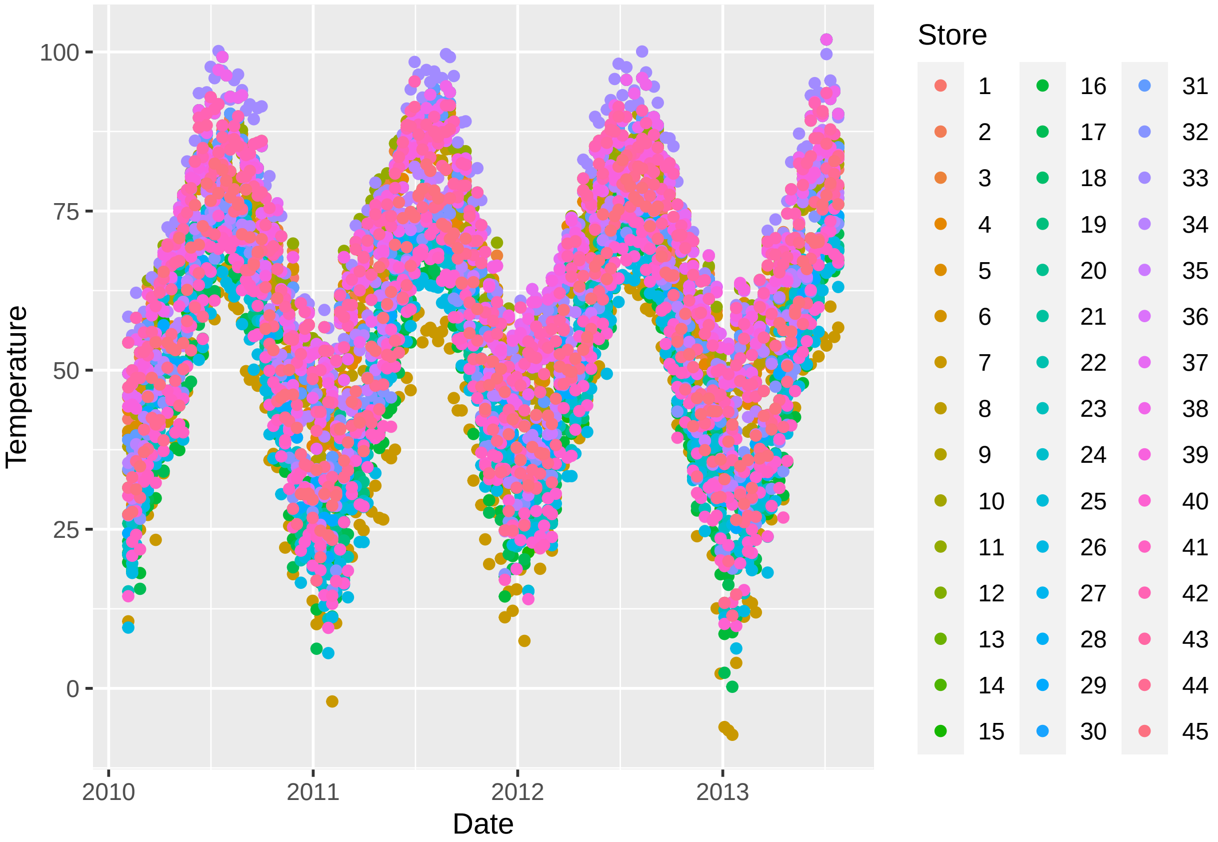
Let's start with a quick scatter plot. Notice the switch from the pipe (%>%) to the plus sign in the ggplot code.

```
features_df %>%
  ggplot(aes(x = Date, y = Temperature)) +
  geom_point()
```



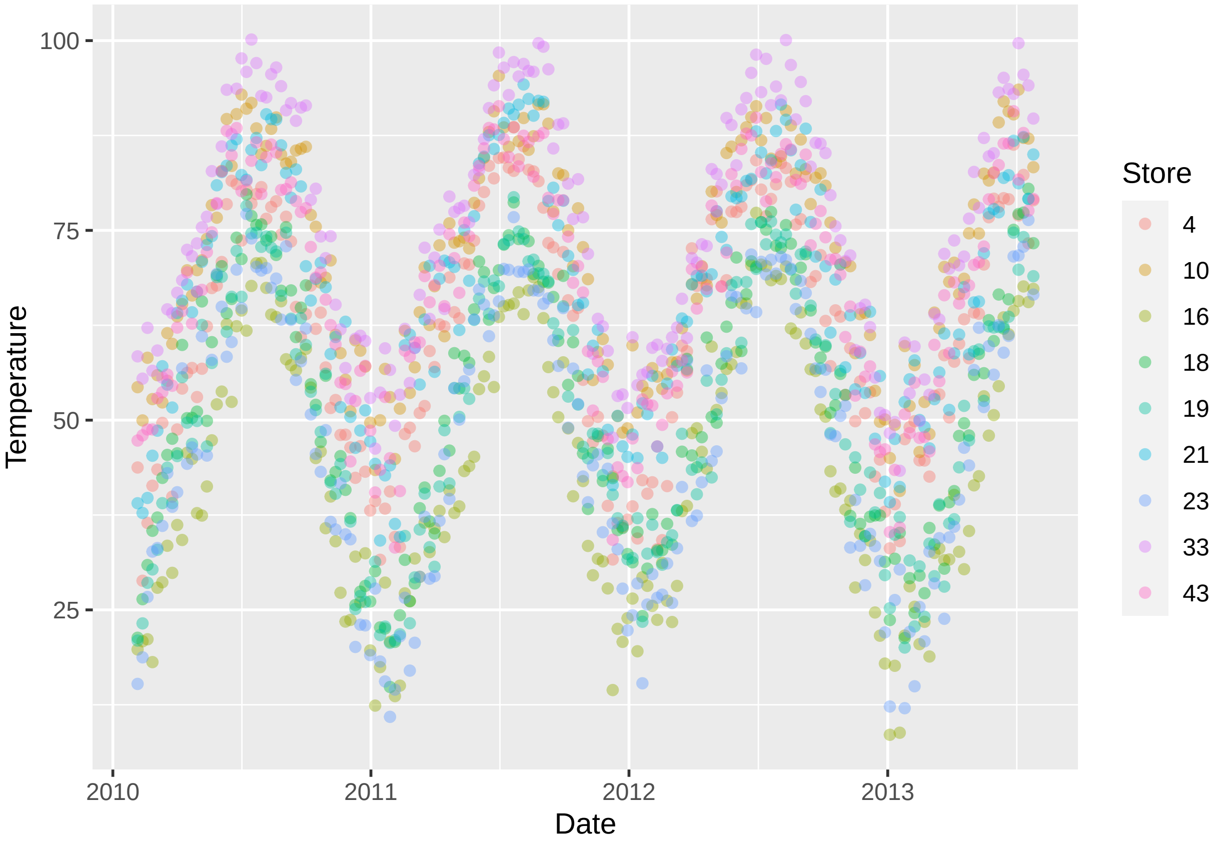
Since we're mapping columns to different aspects of the graphic, let's try to add in a bit more data.

```
features_df %>%
  ggplot(aes(x = Date, y = Temperature)) +
  geom_point(aes(color = Store))
```



This is still really busy. Let's take a sample of the data, graph that, and then place around with the opacity of the points to try to improve the image.

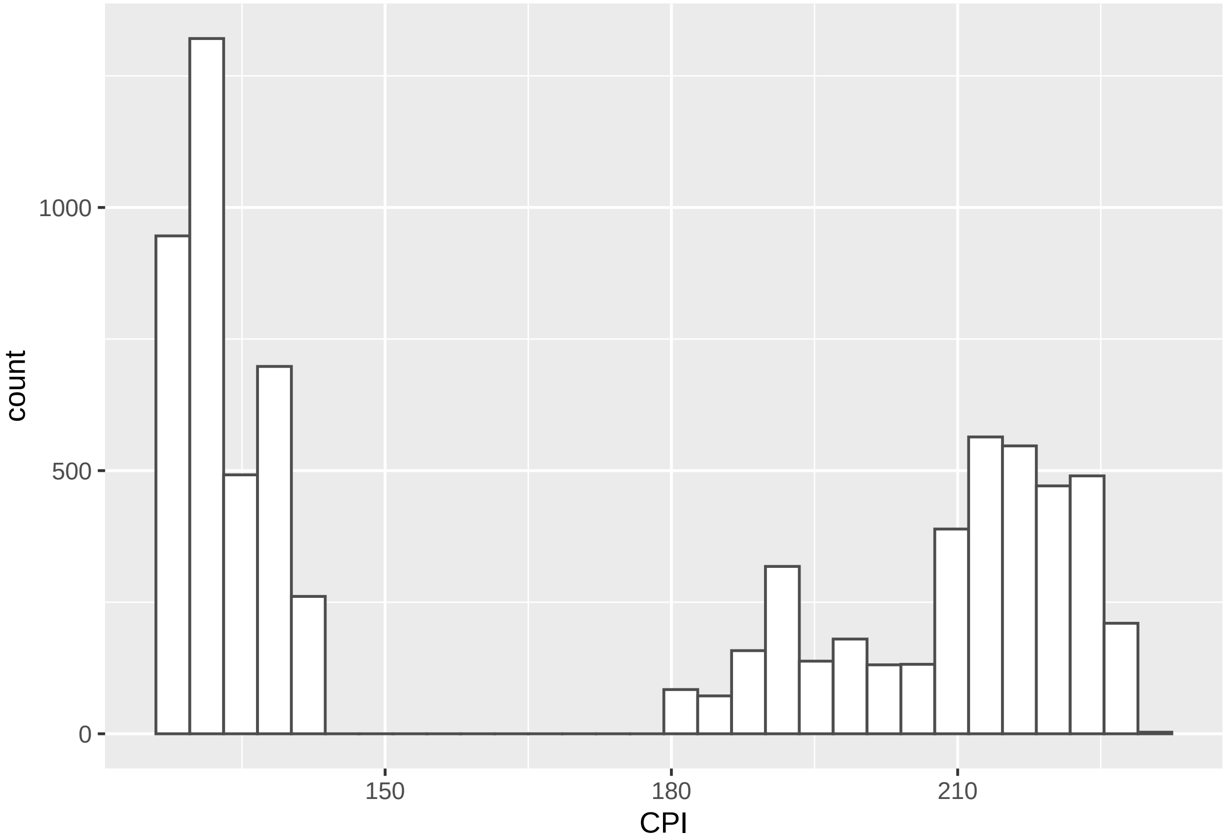
```
sample_stores = sample(1:50, 10)
features_df %>%
  filter(Store %in% sample_stores) %>%
  ggplot(aes(x = Date, y = Temperature)) +
  geom_point(aes(color=Store), alpha=0.4)
```



We can start to see trends in the temperature here, differentiated by individual stores.

We can use other shapes to get a sense of the distribution of our data. Let's build a histogram that let's us explore the distribution of the CPI over the whole dataset.

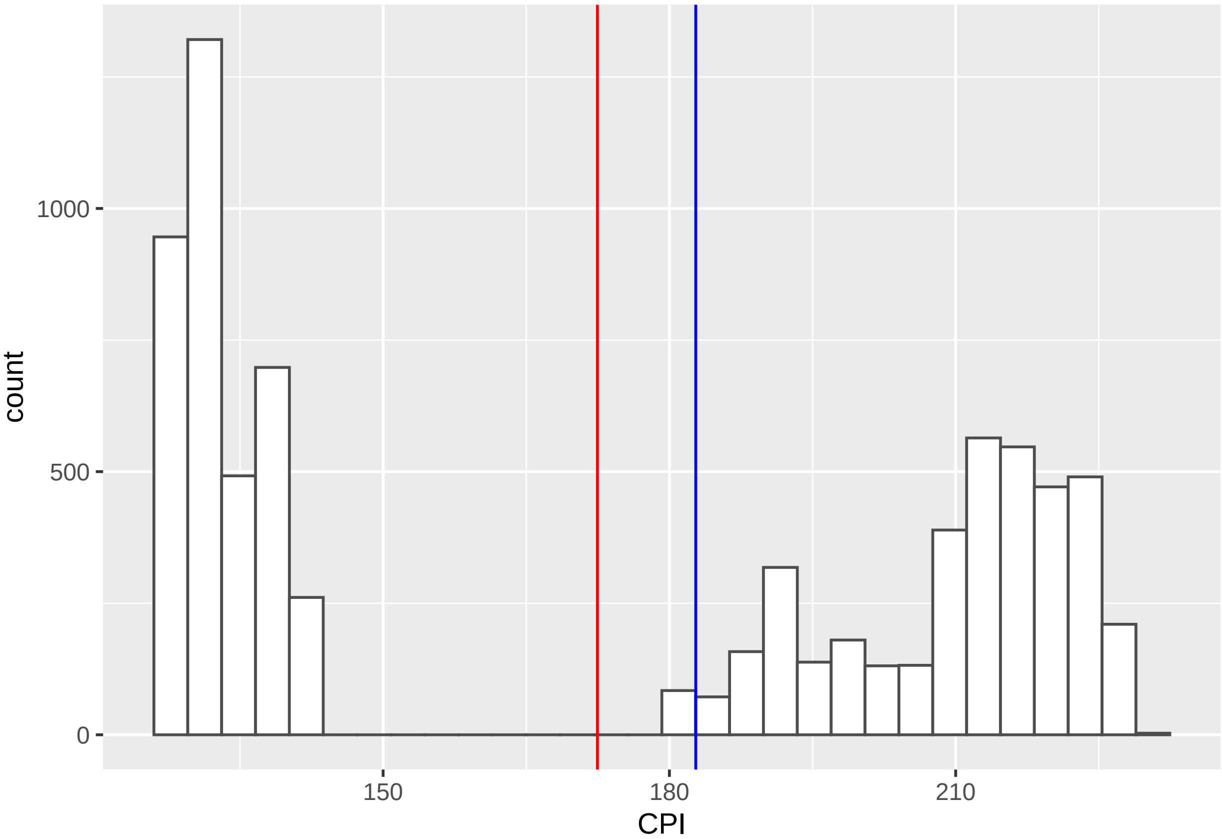
```
features_df %>%
  ggplot(aes(CPI)) +
  geom_histogram(fill = "white", color = "grey30")
```



This shape is interesting. I'm definitely not a business person or an economist, so there might be a clear explanation you know for why we have such a high number of observations with a low CPI, then a huge gap in the middle.

Let's add some reference lines to this graph: a mean line that will help us see how much of this distribution falls above or below the mean CPI for the data, and the same for median.

```r
features_df %>%
  ggplot(aes(CPI)) +
  geom_histogram(fill = "white", color = "grey30") +
  geom_vline(aes(xintercept = mean(features_df$CPI, na.rm = TRUE)), color = "red") +
  geom_vline(aes(xintercept = median(features_df$CPI, na.rm = TRUE)), color = "blue")
```



We're going to keep playing around with `ggplot2`, but let's also learn a bit about aggregate measures while we're doing that.

Aggregate measures

The tidyverse way to do aggregations is using the "split-apply-combine" approach, facilitated by functions like `group_by` and `summarise`. Let's see it in action.

Maybe we want to know what the average temperature is for each store. We can group the observations by `Store`, and take the mean of the values in the `Temperature` column.

```
features_df %>%
  group_by(Store) %>%
  summarise(avg_temp = mean(Temperature))
```

```
## # A tibble: 45 x 2
##    Store avg_temp
##    <fct>    <dbl>
##  1 1         66.9
##  2 2         66.7
##  3 3         70.4
##  4 4         61.4
##  5 5         68.2
##  6 6         68.5
##  7 7         37.9
##  8 8         61.2
##  9 9         66.3
## 10 10        71.3
## # ... with 35 more rows
```

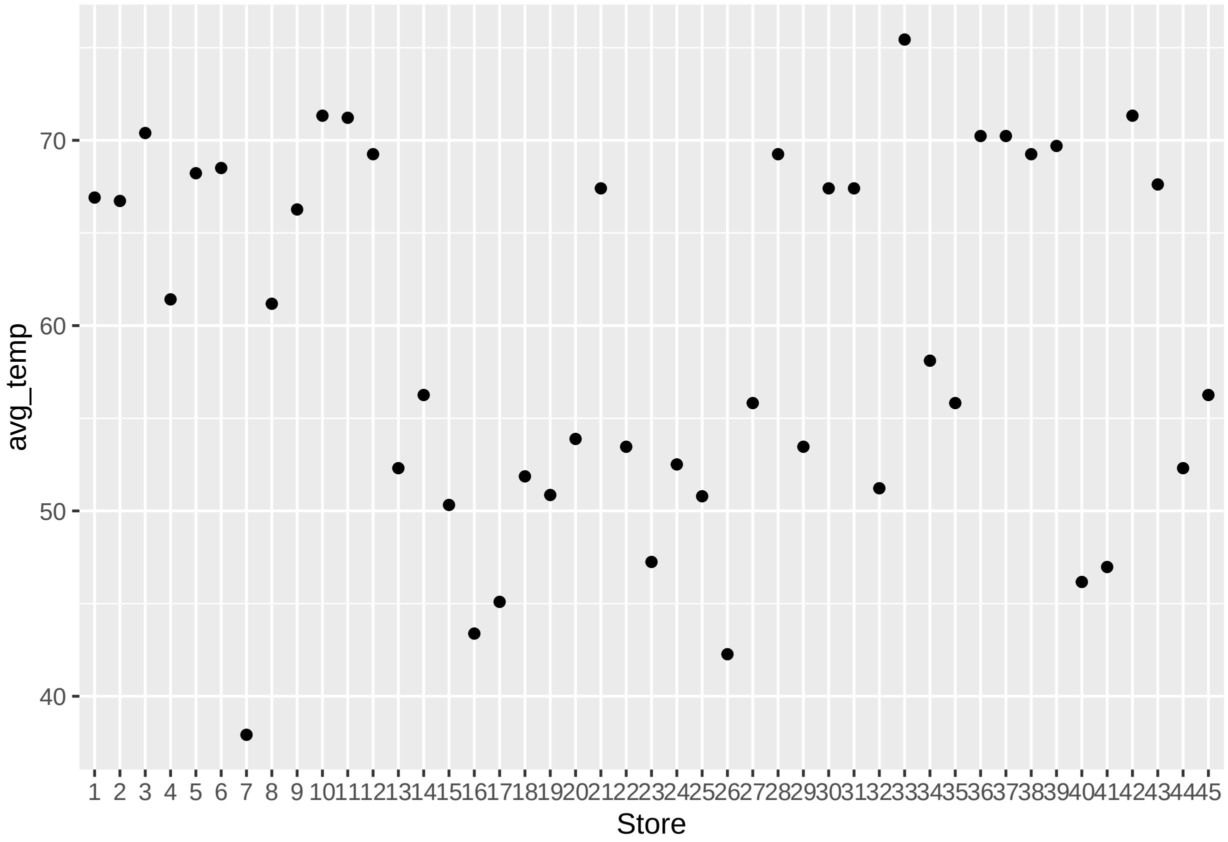
We can sort this data by the value in avg_temp.

```
features_df %>%
  group_by(Store) %>%
  summarise(avg_temp = mean(Temperature)) %>%
  arrange(desc(avg_temp))
```

```
## # A tibble: 45 x 2
##    Store avg_temp
##    <fct>    <dbl>
##  1 33        75.4
##  2 10        71.3
##  3 42        71.3
##  4 11        71.2
##  5 3         70.4
##  6 36        70.2
##  7 37        70.2
##  8 39        69.7
##  9 12        69.3
## 10 28        69.3
## # ... with 35 more rows
```

The result itself can be saved as a dataframe or tibble, then modified or used as we want. We can also just add ggplot code to the end of our statement to see results on the fly when we're just doing some data exploration.

```
features_df %>%
  group_by(Store) %>%
  summarise(avg_temp = mean(Temperature)) %>%
  ggplot(aes(x = Store, y = avg_temp)) +
  geom_point()
```



We've just used `mean` throughout this example, but you could use any number of different built in or custom functions, as long as they take a vector or series of values, and condense them to a single value.

A brief look at modeling

Let's switch over to our other dataset, stored in `train_df`, and apply some of what we've learned. At the end, we'll also take a quick look at modeling.

We'll start with a high-level look at the data.

```
skim(train_df)
```

Table 17: Data summary

Name	train_df
Number of rows	421570
Number of columns	5
Column type frequency:	
Date	1
logical	1
numeric	3
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Date	0	1	2010-02-05	2012-10-26	2011-06-17	143

Variable type: logical

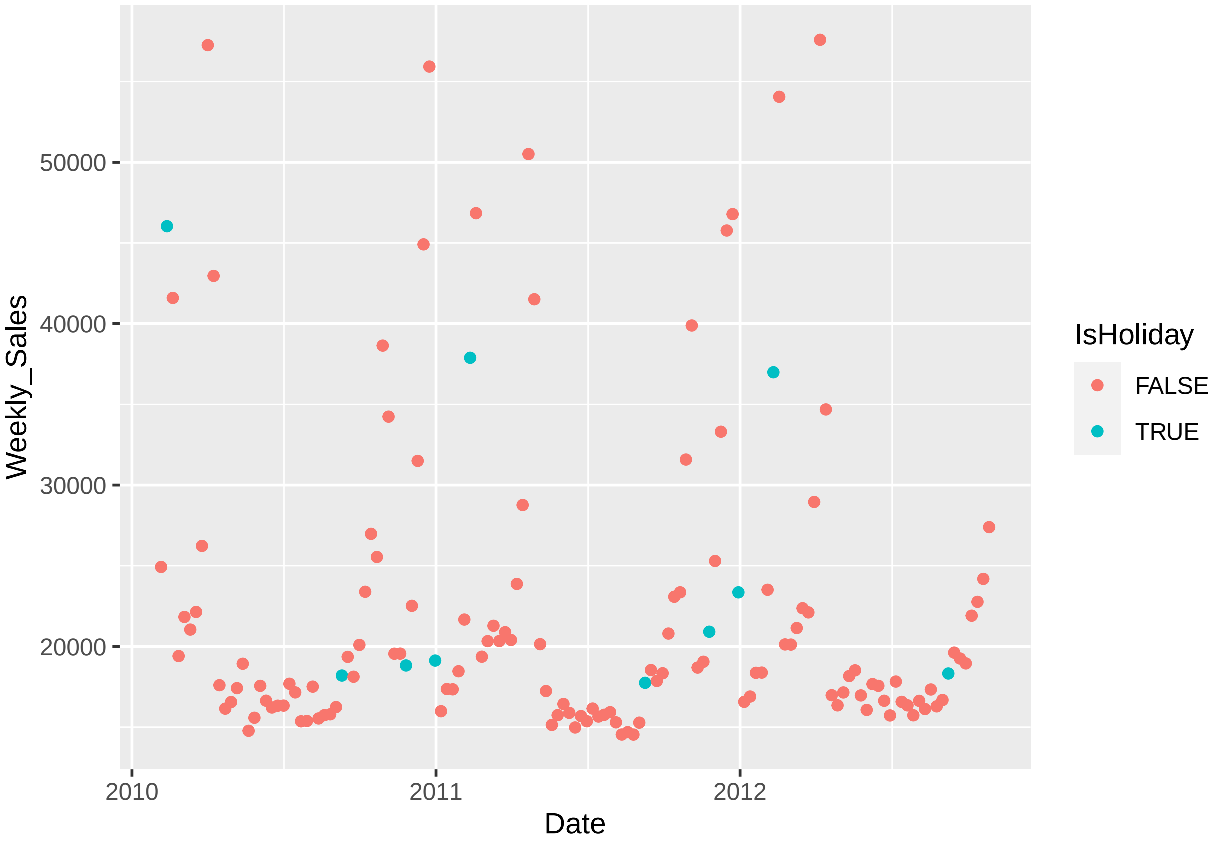
skim_variable	n_missing	complete_rate	mean	count
IsHoliday	0	1	0.07	FAL: 391909, TRU: 29661

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Store	0	1	22.20	12.79	1.00	11.00	22.00	33.00	45.0
Dept	0	1	44.26	30.49	1.00	18.00	37.00	74.00	99.0
Weekly_Sales	0	1	15981.26	22711.18	-4988.94	2079.65	7612.03	20205.85	693099.4

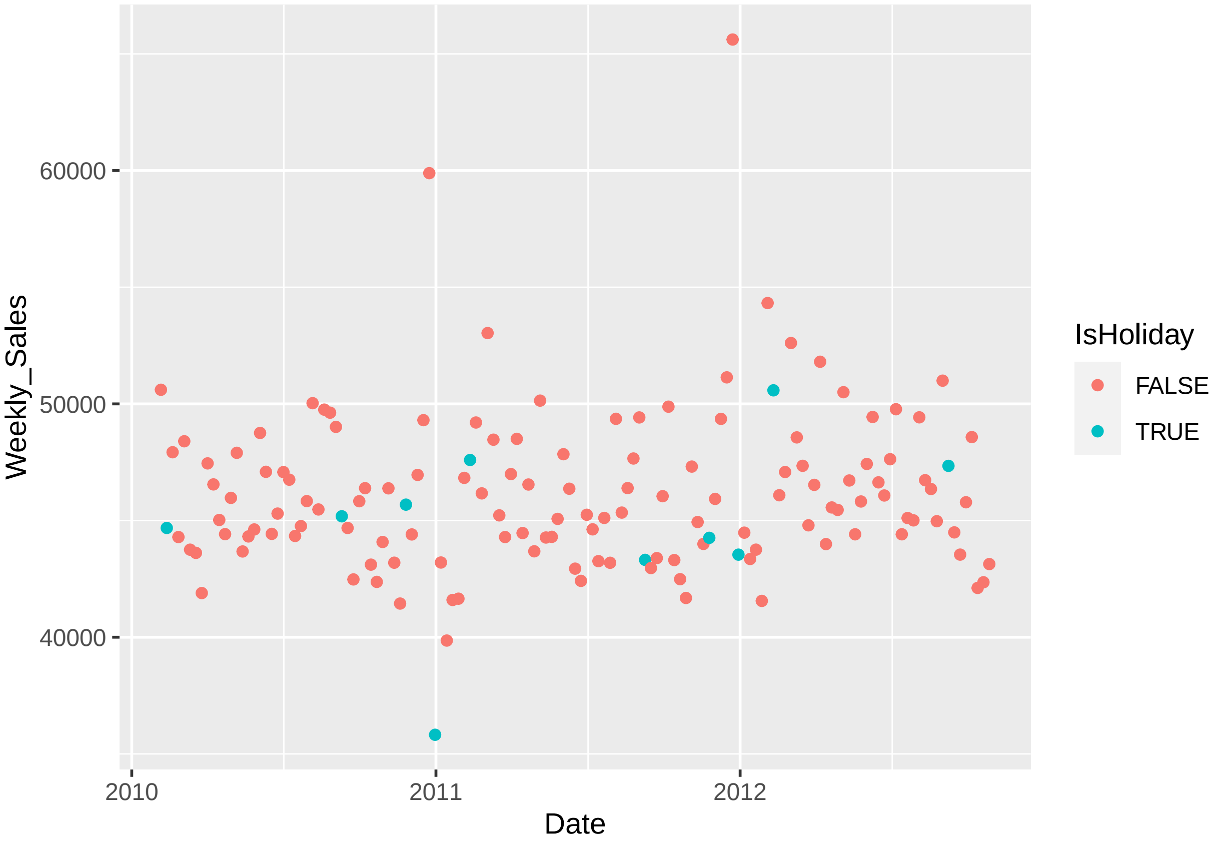
Here each row or observation is the sales data for a week for a single department in a single store. There are 421,570 observations, so we may not have much success trying to put every single point on a graph. Let's make the `Store` and `Dept` columns factors, then start with some filtering.

```
train_df$Store = as_factor(train_df$Store)
train_df$Dept = as_factor(train_df$Dept)


train_df %>%
  filter(Store == 1) %>%
  filter(Dept == 1) %>%
  ggplot(aes(Date, Weekly_Sales, color = IsHoliday)) +
  geom_point()
```



We can see that there is some pattern to the sales of a single department in a single store, though it's a bit loose at points. Let's try another department.

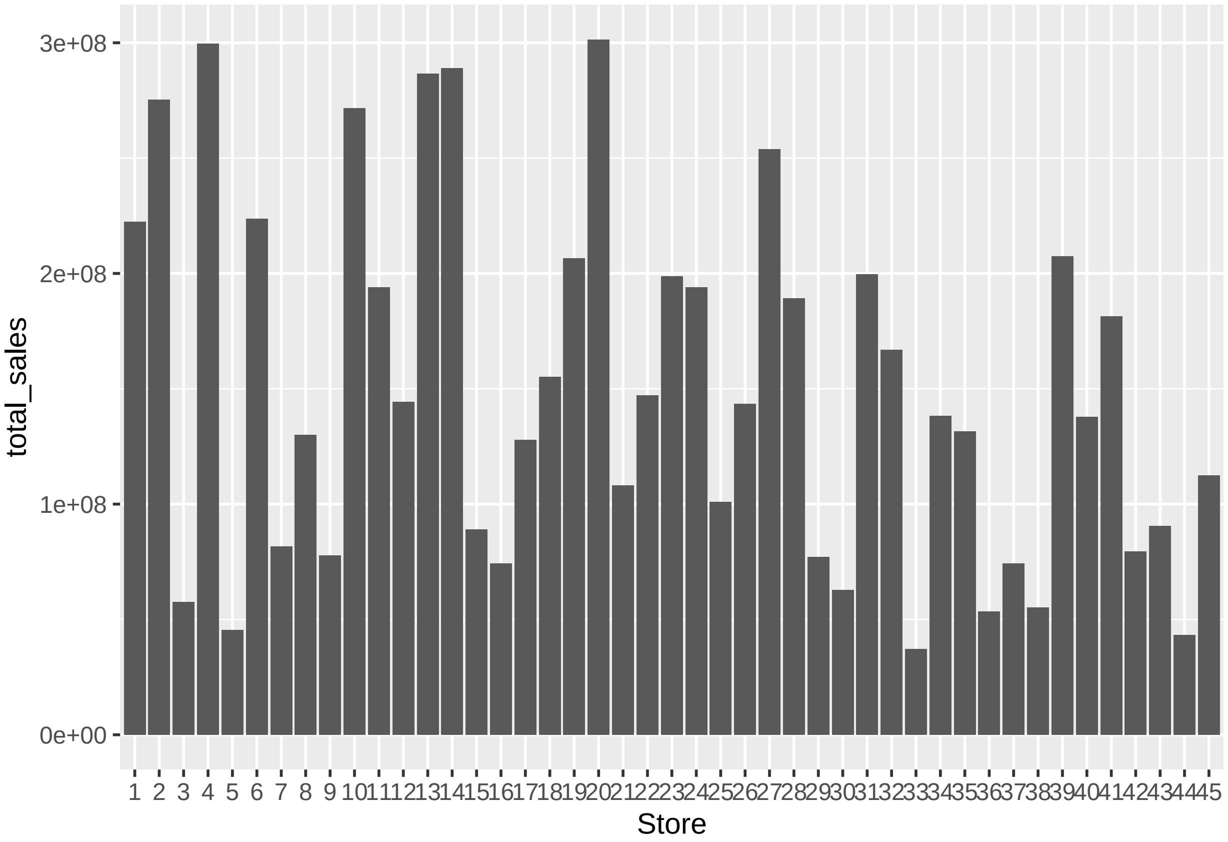
```r
train_df %>%
  filter(Store == 1) %>%
  filter(Dept == 2) %>%
  ggplot(aes(Date, Weekly_Sales, color = IsHoliday)) +
  geom_point()
```



This looks like a pretty different pattern. We could definitely keep going like this, and could programmatically generate and save figures like this for every department and every store. Let's do a bit of aggregation, though, to look at the whole data set.

Maybe we're interested in the total sales for each store over the multi-year period.

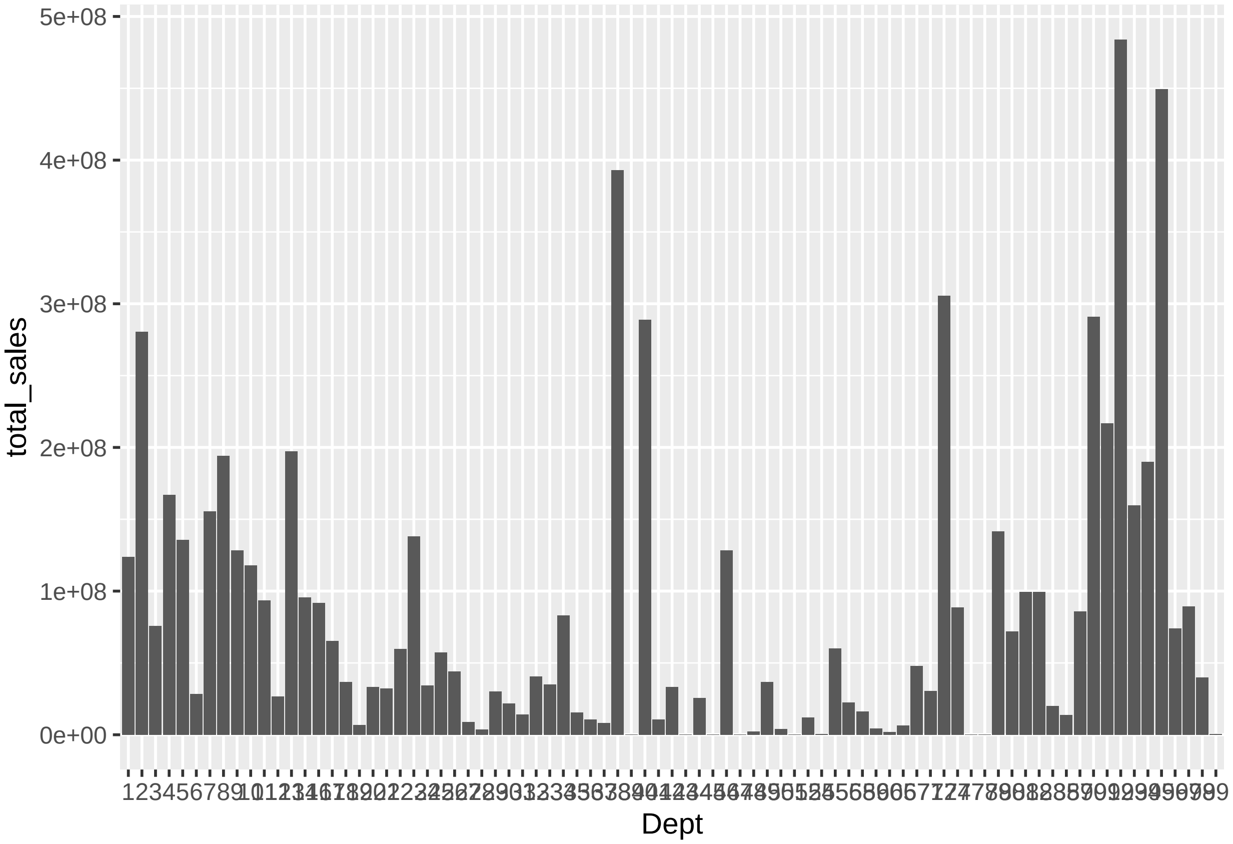
```
train_df %>%
  group_by(Store) %>%
  summarise(total_sales = sum(Weekly_Sales)) %>%
  ggplot(aes(x = Store, y = total_sales)) +
  geom_col()
```



Some of these stores have had substantially higher sales than others.

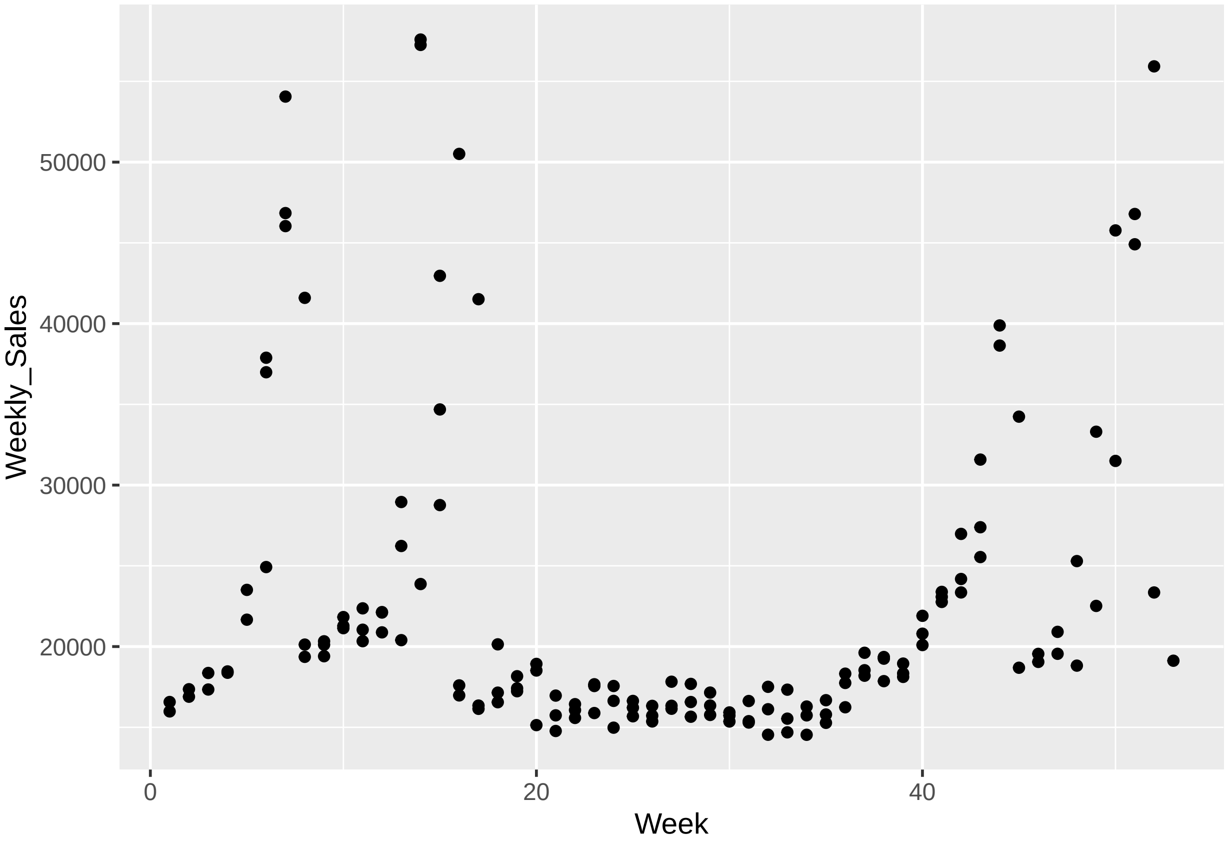
We could also group the data by department instead, to see which departments typically sell more.

```
train_df %>%
  group_by(Dept) %>%
  summarise(total_sales = sum(Weekly_Sales)) %>%
  ggplot(aes(x = Dept, y = total_sales)) +
  geom_col()
```



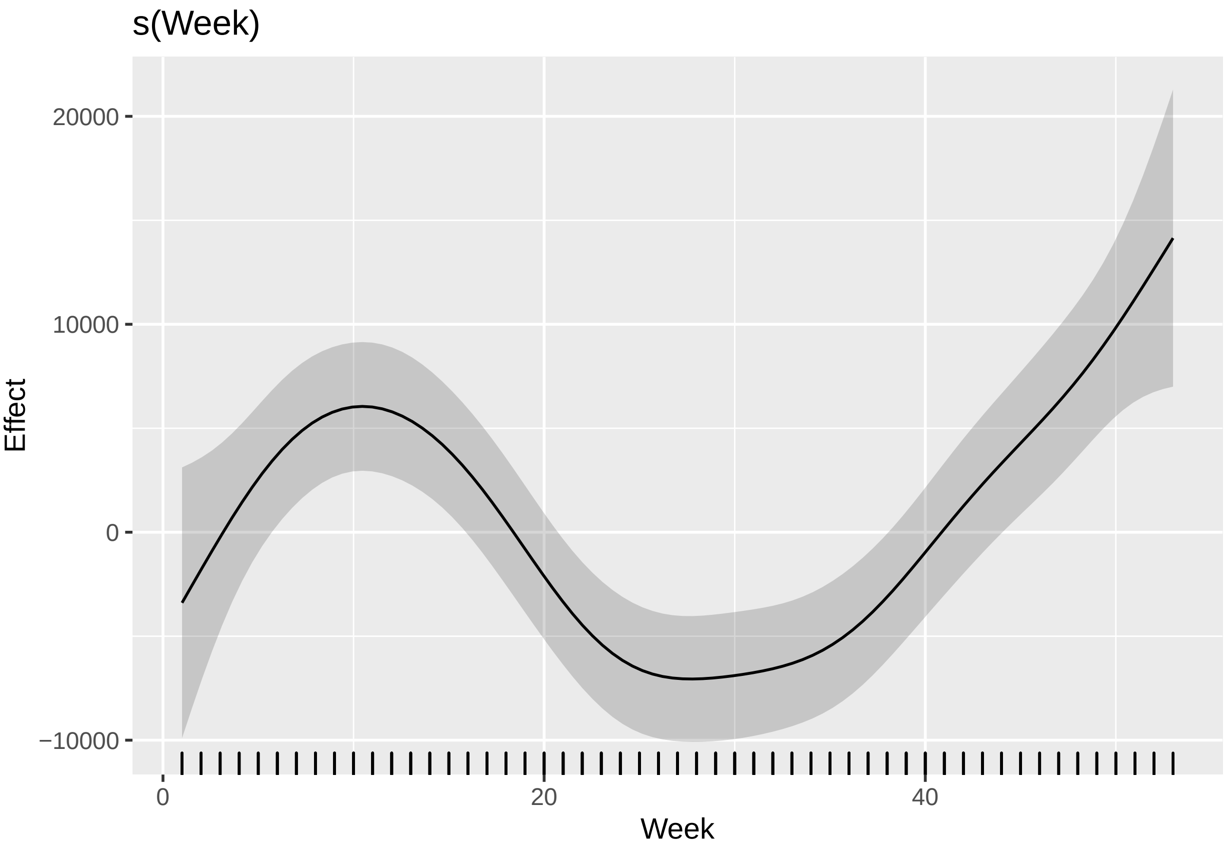
As a final exercise, let's think about how we might model some of this data, and see what R could do for us there. We will need to create a column for the `Week` as we did in the features data.

```
train_mod <- train_df %>%
  filter(Store == 1) %>%
  filter(Dept == 1) %>%
  mutate(
    Day = day(Date),
    Month = month(Date),
    Year = year(Date),
    Week = week(Date)
  )

train_mod %>%
  ggplot(aes(x = Week, y = Weekly_Sales)) +
  geom_point()
```

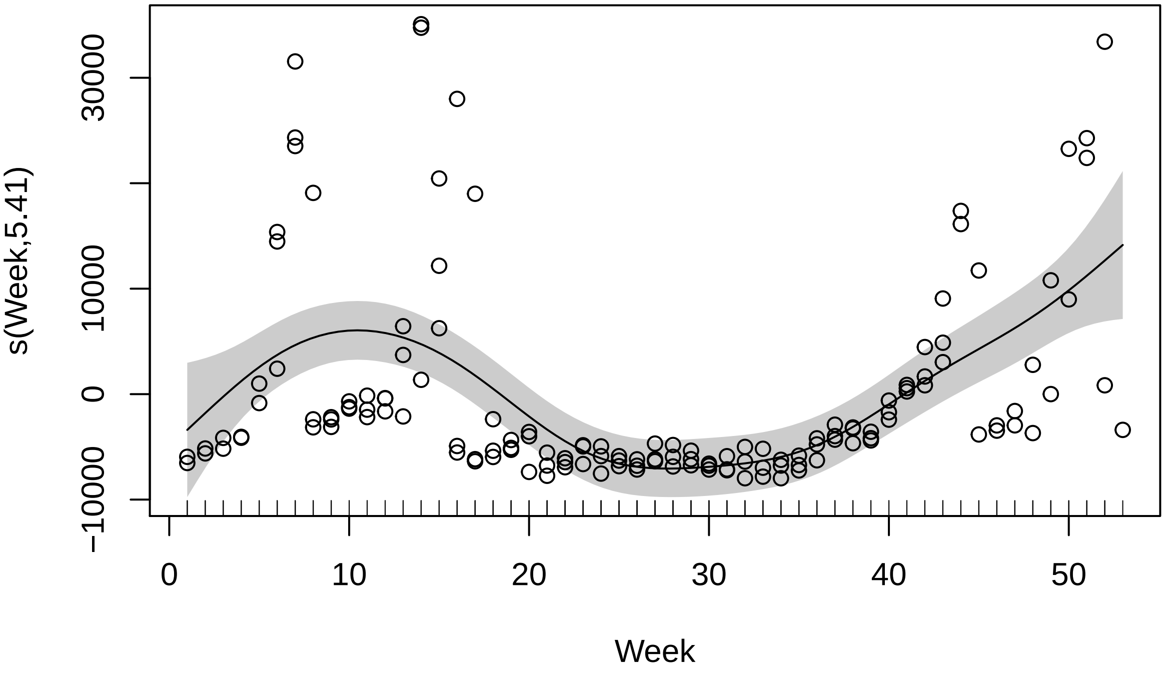


We have a non-linear shape with multiple curves, so we'll try out a generalized additive model.

```
gam_mod <- gam(Weekly_Sales ~ s(Week), data = train_mod, method="REML")

draw(gam_mod)
```



```
plot(gam_mod, residuals = TRUE, pch = 1, cex = 1, shade = TRUE)
```



This is just a first pass at a model. We could pull the coefficients and examine them. We could start modifying parameters in the gam, and try to pick up a bit more on the two high curves in the early distribution of the data. For now, though, it's enough to see that we can do this advanced sort of analysis fairly quickly because of the great packages that have been created for R. For anything related to data exploration, analysis, visualization, R is incredibly power, and incredibly flexible.

Further Resources

There is a ton of great content to help with learning R online. Here's just a few resources that I think are solid.

- Hadley Wickham's R for Data Science - the essential guide from the creator of the tidyverse: https://r4ds.had.co.nz/
- DataQuest - online learning - free and paid: https://www.dataquest.io/path/data-analyst-r/
- Chris Engelhardt's list of data science resources: https://github.com/Chris-Engelhardt/data_sci_guide
- Roger Peng's R Programming for Data Science: https://bookdown.org/rdpeng/rprogdatascience/
- Jenny Bryan's Stats545 course: https://stat545.com/
- RStudio Conf videos - more advanced sometimes, but great: https://rstudio.com/resources/rstudioconf-2020/
- For intermediate folks with a bit of experience already, What They Forgot to Teach You about R from Jenny Bryan and Jim Hester is wonderful, and should be definitive for best practices for projects and workflows: https://rstats.wtf/