Web scraping and Parallel Processing

Greg Ridgeway Ruth Moyer

2026-08-02

Table of contents

1 Introduction **1**

2 Scraping one page **4**
2.1 Movies with no titles 11

3 Scraping Multiple Pages **12**

4 Parallel Computing **19**

5 Fun With Movie Data **32**
5.1 Adjusting for Inflation 33

6 Crime and the Movie Blockbuster **35**
6.1 What about really big movie days? 42

7 Solutions to the exercises **48**

1 Introduction

At the end of our discussion about regular expressions, we introduced the concept of web scraping. Not all online data is in a tidy, downloadable format such as a .csv or .RData file. Yet, patterns in the underlying HTML code and regular expressions together provide a valuable way to "scrape" data off of a webpage. Here, we are going to work through an example of web scraping. We are going to get data on ticket sales of every movie, for every day going back to 2010.

As a preliminary matter, some R packages, such as `rvest` and `chromote`, can help with web scraping. Later we will experiment with `chromote`. For now, we are going to work with basic fundamentals so that you have the most flexibility to extract data from most websites.

First, you will need to make sure that you can access the underlying HTML code for the webpage that you want to scrape. In most browsers you can simply right-click on a webpage and then click "View Page Source." If you are using Microsoft Edge, you can right-click on the webpage, click "View Source," and then look at the "Debugger" tab. In Safari select "Settings," choose the "Advanced" tab, check "Show Develop menu," and then whenever viewing a page you can right-click and select "Show Page Source."

Have a look at the webpage https://www.the-numbers.com/box-office-chart/daily/2026/07/04. This page contains information about the movies that were shown in theaters on July 4, 2026 and the amount of money (in dollars) that each of those movies grossed that day.

Have a look at the HTML code by looking at the page source for this page using the methods described above. The first 10 lines should look something like this:

```
<!DOCTYPE html>
<html xmlns:og="https://ogp.me/ns#">
<head>
<link rel='icon' href='https://www.the-numbers.com/site-images/favicon.ico'>
<script async src="https://www.googletagmanager.com/gtag/js?id=G-5K2DT3XQN5"></script>
<script>window.dataLayer = window.dataLayer || []; function gtag() { dataLayer.push(arguments
5K2DT3XQN5');</script>
<meta http-equiv="PICS-Label" content='(PICS-1.1 "https://www.icra.org/ratingsv02.html" l ge
numbers.com/" r (cb 1 lz 1 nz 1 oz 1 vz 1) "https://www.rsac.org/ratingsv01.html" l gen true
numbers.com/" r (n 0 s 0 v 0 l 0))'>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="format-detection" content="telephone=no">   <!-- for apple mobile -->
<meta name=viewport content="width=device-width, initial-scale=1">
```

This is all HTML code to set up the page. If you scroll down a few hundred lines, you will find code that looks like this:

```
<tbody>
<tr>
<td class="data">1</td>
<td class="data">(1)</td>
<td><b><a href="/movie/Minions-and-Monsters-(2026)">Minions &amp;amp; Monsters</a></b></td>
<td class="data">$9,491,820</td>
<td class="data chart_down d">-42%</td>
<td class="data Daily"> </td>
<td class="data">4,243</td>
<td class="data">$2,237</td>
<td class="data">$51,039,330</td>
<td class="data">4</td>
```

```
</tr>
<tr>
<td class="data">2</td>
<td class="data">(3)</td>
<td><b><a href="/movie/Young-Washington-(2026)">Young Washington</a></b></td>
```

I see *Minions and Monsters* and *Young Washington*. In addition to the movie name, there are ticket sales, number of theaters, and more. It is all wrapped in a lot of HTML code to make it look pretty on a web page, but for our purposes we just want to pull those numbers out.

`scan()` is a basic R function for reading in text, from the keyboard, from files, from the web, … however data might arrive. Giving `scan()` a URL causes `scan()` to pull down the HTML code for that page and return it to you. Let's try one page of movie data. `what=""` tells `scan()` to expect plain text and `sep="\n"` tells `scan()` to separate each element when it reaches a line feed character, signaling the end of a line.

Here I have wrapped `scan()` inside `try()`. More on this later. Briefly, this `try()` inside the `repeat{}` loop will allow for errors, like the website being unavailable for a moment. If `scan()` cannot reach the website, then `Sys.sleep(10)` will pause for 10 seconds before trying again.

```
library(dplyr)
library(tidyr)
repeat
{
   a <- try(scan("https://www.the-numbers.com/box-office-chart/daily/2026/07/04",
             what="",sep="\n"))
   if(!inherits(a,"try-error")) break
   Sys.sleep(10)
}
# examine the first few lines
a[1:5]
```

```
[1] "<!DOCTYPE html>"
[2] "<html xmlns:og=\"https://ogp.me/ns#\">"
[3] "<head>"
[4] "<link rel='icon' href='https://www.the-numbers.com/site-images/favicon.ico'>"
[5] "<script async src=\"https://www.googletagmanager.com/gtag/js?id=G-
5K2DT3XQN5\"></script>"
```

Some websites are more complex or use different text encoding. On those websites `scan()` produces unintelligible text. The `GET()` function from the `httr` package can sometimes resolve this.

```
library(httr)
resp <- GET("https://www.the-numbers.com/box-office-chart/daily/2026/07/04")
a1 <- content(resp, as="text")
a1 <- strsplit(a1,"\n")[[1]]
```

Also, some Mac users will encounter snags with both of these methods and receive "403 Forbidden" errors while their Mac colleague right next to them on the same network will not. I have not figured out why this happens, but have found that making R masquerade as a different browser sometimes works.

```
resp <- GET("https://www.the-numbers.com/box-office-chart/daily/2026/07/04",
            user_agent("Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/537.13+ (
a1 <- content(resp, as="text")
a1 <- strsplit(a1,"\n")[[1]]
```

2 Scraping one page

Now that we have stored in the variable `a` the HTML code for one day's movie data in R, let's apply some regular expressions to extract the data. The HTML code includes a lot of lines that do not involve data that interests us. There is code for making the page look nice and code for presenting advertisements. Let's start by finding the lines that have the movie names in them.

Going back to the HTML code, I noticed that the lines with *Minions and Monsters* and *Young Washington* both have the sequence of characters `href="/movie/`. By finding a pattern of characters that always appears on movie title lines, we can use it to grep the lines we want. Let's find every line that has `href="/movie/` in it.

```
i <- grep('href="/movie/', a)
i
```

```
 [1] 203 215 227 239 251 263 275 287 299 311 323 335 347 359 371 383 395 407 419
[20] 431 443 455 467 479 491 503 515 527 539 551 563 591 597 603 609 615 621 627
[39] 633 639 645 651 657 663 669 675 681 687 693 699 705 711 717 723 729 735 741
[58] 747 753 759 765 771
```

These are the line numbers that, if the pattern holds, contain our movie titles. Note that the numbers you get on your computer might be a little different from the line numbers shown here. Even if you run this code on a different day, you might get different line numbers because some of the code, code for advertisements in particular, can frequently change.

Let's see what these lines of HTML code look like.

```
a[i]
```

```
 [1] "<td><b><a href=\"/movie/Minions-and-Monsters-(2026)\">Minions &amp;amp; Monsters</a></b></t
 [2] "<td><b><a href=\"/movie/Young-Washington-(2026)\">Young Washington</a></b></td>"
 [3] "<td><b><a href=\"/movie/Toy-Story-5-(2026)\">Toy Story 5</a></b></td>"
 [4] "<td><b><a href=\"/movie/Supergirl-(2026)\">Supergirl</a></b></td>"
 [5] "<td><b><a href=\"/movie/Disclosure-Day-(2026)\">Disclosure Day</a></b></td>"
 [6] "<td><b><a href=\"/movie/Obsession-(2026)\">Obsession</a></b></td>"
 [7] "<td><b><a href=\"/movie/Backrooms-(2026)\">Backrooms</a></b></td>"
 [8] "<td><b><a href=\"/movie/Jackass-Best-and-Last-(2026)\">Jackass: Best and Last</a></b></
 [9] "<td><b><a href=\"/movie/Scary-Movie-(2026)\">Scary Movie</a></b></td>"
[10] "<td><b><a href=\"/movie/Masters-of-the-Universe-(2026)\">Masters of the Universe</a></
[11] "<td><b><a href=\"/movie/Invite-The-(2026)\">The Invite</a></b></td>"
[12] "<td><b><a href=\"/movie/Star-Wars-The-Mandalorian-and-Grogu-(2026)\">Star Wars: The Ma
[13] "<td><b><a href=\"/movie/Michael-(2026)\">Michael</a></b></td>"
[14] "<td><b><a href=\"/movie/Leviticus-(2026-Australia)\">Leviticus</a></b></td>"
[15] "<td><b><a href=\"/movie/Lucky-Strike-(2026)\">Lucky Strike</a></b></td>"
[16] "<td><b><a href=\"/movie/Devil-Wears-Prada-2-The-(2026)\">The Devil Wears Prada 2</a></
[17] "<td><b><a href=\"/movie/Tuner-(2026)\">Tuner</a></b></td>"
[18] "<td><b><a href=\"/movie/Sheep-Detectives-The-(2026)\">The Sheep Detectives</a></b></td
[19] "<td><b><a href=\"/movie/Furious-The-(2026-Hong-Kong)\">The Furious</a></b></td>"
[20] "<td><b><a href=\"/movie/Death-of-Robin-Hood-The-(2026)\">The Death of Robin Hood</a></
[21] "<td><b><a href=\"/movie/Rose-of-Nevada-(2026-United-Kingdom)\">Rose of Nevada</a></b></
[22] "<td><b><a href=\"/movie/Stop-That-Train-(2026)\">Stop! That! Train!</a></b></td>"
[23] "<td><b><a href=\"/movie/I-Love-Boosters-(2026)\">I Love Boosters</a></b></td>"
[24] "<td><b><a href=\"/movie/Mortal-Kombat-II-(2026)\">Mortal Kombat II</a></b></td>"
[25] "<td><b><a href=\"/movie/Couture-(2026)\">Couture</a></b></td>"
[26] "<td><b><a href=\"/movie/Terminator-2-Judgment-Day-(1991)\">Terminator 2: Judgment Day</
[27] "<td><b><a href=\"/movie/Time-And-Water-(2026-Iceland)\">Time And Water</a></b></td>"
[28] "<td><b><a href=\"/movie/Stille-Freundin-(2026-Germany)\">Silent Friend</a></b></td>"
[29] "<td><b><a href=\"/movie/Unidentified-(2026-Saudi-Arabia)\">Unidentified</a></b></td>"
[30] "<td><b><a href=\"/movie/Great-Awakening-A-(2026)\">A Great Awakening</a></b></td>"
[31] "<td><b><a href=\"/movie/Erupcja-(2026)\">Erupcja</a></b></td>"
[32] "<td><b><a href=\"/movie/Minions-and-Monsters-(2026)\">Minions &amp;amp; Monsters</a></b></t
[33] "<td><b><a href=\"/movie/Young-Washington-(2026)\">Young Washington</a></b></td>"
[34] "<td><b><a href=\"/movie/Toy-Story-5-(2026)\">Toy Story 5</a></b></td>"
[35] "<td><b><a href=\"/movie/Supergirl-(2026)\">Supergirl</a></b></td>"
[36] "<td><b><a href=\"/movie/Disclosure-Day-(2026)\">Disclosure Day</a></b></td>"
[37] "<td><b><a href=\"/movie/Obsession-(2026)\">Obsession</a></b></td>"
[38] "<td><b><a href=\"/movie/Backrooms-(2026)\">Backrooms</a></b></td>"
```

```
[39] "<td><b><a href=\"/movie/Jackass-Best-and-Last-(2026)\">Jackass: Best and Last</a></b><
[40] "<td><b><a href=\"/movie/Scary-Movie-(2026)\">Scary Movie</a></b></td>"
[41] "<td><b><a href=\"/movie/Masters-of-the-Universe-(2026)\">Masters of the Universe</a></
[42] "<td><b><a href=\"/movie/Invite-The-(2026)\">The Invite</a></b></td>"
[43] "<td><b><a href=\"/movie/Star-Wars-The-Mandalorian-and-Grogu-(2026)\">Star Wars: The Mar
[44] "<td><b><a href=\"/movie/Michael-(2026)\">Michael</a></b></td>"
[45] "<td><b><a href=\"/movie/Leviticus-(2026-Australia)\">Leviticus</a></b></td>"
[46] "<td><b><a href=\"/movie/Lucky-Strike-(2026)\">Lucky Strike</a></b></td>"
[47] "<td><b><a href=\"/movie/Devil-Wears-Prada-2-The-(2026)\">The Devil Wears Prada 2</a></
[48] "<td><b><a href=\"/movie/Tuner-(2026)\">Tuner</a></b></td>"
[49] "<td><b><a href=\"/movie/Sheep-Detectives-The-(2026)\">The Sheep Detectives</a></b></td
[50] "<td><b><a href=\"/movie/Furious-The-(2026-Hong-Kong)\">The Furious</a></b></td>"
[51] "<td><b><a href=\"/movie/Death-of-Robin-Hood-The-(2026)\">The Death of Robin Hood</a></
[52] "<td><b><a href=\"/movie/Rose-of-Nevada-(2026-United-Kingdom)\">Rose of Nevada</a></b><
[53] "<td><b><a href=\"/movie/Stop-That-Train-(2026)\">Stop! That! Train!</a></b></td>"
[54] "<td><b><a href=\"/movie/I-Love-Boosters-(2026)\">I Love Boosters</a></b></td>"
[55] "<td><b><a href=\"/movie/Mortal-Kombat-II-(2026)\">Mortal Kombat II</a></b></td>"
[56] "<td><b><a href=\"/movie/Couture-(2026)\">Couture</a></b></td>"
[57] "<td><b><a href=\"/movie/Terminator-2-Judgment-Day-(1991)\">Terminator 2: Judgment Day<
[58] "<td><b><a href=\"/movie/Time-And-Water-(2026-Iceland)\">Time And Water</a></b></td>"
[59] "<td><b><a href=\"/movie/Stille-Freundin-(2026-Germany)\">Silent Friend</a></b></td>"
[60] "<td><b><a href=\"/movie/Unidentified-(2026-Saudi-Arabia)\">Unidentified</a></b></td>"
[61] "<td><b><a href=\"/movie/Great-Awakening-A-(2026)\">A Great Awakening</a></b></td>"
[62] "<td><b><a href=\"/movie/Erupcja-(2026)\">Erupcja</a></b></td>"
```

Double-checking and indeed the first line here is *Minions and Monsters* and the last line is *Erupcja*. This matches what is on the web page. We now are quite close to having a list of movies that played in theaters on July 4, 2026.

You might notice that the movie names are listed twice. Half way through the list you can see *Minions and Monsters* again. The technical reason is that the webpage has a "desktop" and a "mobile" version, and the HTML contains code for both versions. We can find the lines that start each version.

```
grep("chart-desktop", a)
```

```
[1] 184
```

```
grep("chart-mobile", a)
```

```
[1] 578
```

Looks like the desktop version comes first. Let's just keep the lines of HTML code that came before the mobile chart.

```
i <- grep("chart-mobile", a)
a <- a[1:i]
i <- grep('href="/movie/', a)
a[i]
```

```
 [1] "<td><b><a href=\"/movie/Minions-and-Monsters-(2026)\">Minions &amp;amp; Monsters</a></b></t
 [2] "<td><b><a href=\"/movie/Young-Washington-(2026)\">Young Washington</a></b></td>"
 [3] "<td><b><a href=\"/movie/Toy-Story-5-(2026)\">Toy Story 5</a></b></td>"
 [4] "<td><b><a href=\"/movie/Supergirl-(2026)\">Supergirl</a></b></td>"
 [5] "<td><b><a href=\"/movie/Disclosure-Day-(2026)\">Disclosure Day</a></b></td>"
 [6] "<td><b><a href=\"/movie/Obsession-(2026)\">Obsession</a></b></td>"
 [7] "<td><b><a href=\"/movie/Backrooms-(2026)\">Backrooms</a></b></td>"
 [8] "<td><b><a href=\"/movie/Jackass-Best-and-Last-(2026)\">Jackass: Best and Last</a></b></
 [9] "<td><b><a href=\"/movie/Scary-Movie-(2026)\">Scary Movie</a></b></td>"
[10] "<td><b><a href=\"/movie/Masters-of-the-Universe-(2026)\">Masters of the Universe</a></
[11] "<td><b><a href=\"/movie/Invite-The-(2026)\">The Invite</a></b></td>"
[12] "<td><b><a href=\"/movie/Star-Wars-The-Mandalorian-and-Grogu-(2026)\">Star Wars: The Ma
[13] "<td><b><a href=\"/movie/Michael-(2026)\">Michael</a></b></td>"
[14] "<td><b><a href=\"/movie/Leviticus-(2026-Australia)\">Leviticus</a></b></td>"
[15] "<td><b><a href=\"/movie/Lucky-Strike-(2026)\">Lucky Strike</a></b></td>"
[16] "<td><b><a href=\"/movie/Devil-Wears-Prada-2-The-(2026)\">The Devil Wears Prada 2</a></
[17] "<td><b><a href=\"/movie/Tuner-(2026)\">Tuner</a></b></td>"
[18] "<td><b><a href=\"/movie/Sheep-Detectives-The-(2026)\">The Sheep Detectives</a></b></td
[19] "<td><b><a href=\"/movie/Furious-The-(2026-Hong-Kong)\">The Furious</a></b></td>"
[20] "<td><b><a href=\"/movie/Death-of-Robin-Hood-The-(2026)\">The Death of Robin Hood</a></
[21] "<td><b><a href=\"/movie/Rose-of-Nevada-(2026-United-Kingdom)\">Rose of Nevada</a></b></
[22] "<td><b><a href=\"/movie/Stop-That-Train-(2026)\">Stop! That! Train!</a></b></td>"
[23] "<td><b><a href=\"/movie/I-Love-Boosters-(2026)\">I Love Boosters</a></b></td>"
[24] "<td><b><a href=\"/movie/Mortal-Kombat-II-(2026)\">Mortal Kombat II</a></b></td>"
[25] "<td><b><a href=\"/movie/Couture-(2026)\">Couture</a></b></td>"
[26] "<td><b><a href=\"/movie/Terminator-2-Judgment-Day-(1991)\">Terminator 2: Judgment Day</
[27] "<td><b><a href=\"/movie/Time-And-Water-(2026-Iceland)\">Time And Water</a></b></td>"
[28] "<td><b><a href=\"/movie/Stille-Freundin-(2026-Germany)\">Silent Friend</a></b></td>"
[29] "<td><b><a href=\"/movie/Unidentified-(2026-Saudi-Arabia)\">Unidentified</a></b></td>"
[30] "<td><b><a href=\"/movie/Great-Awakening-A-(2026)\">A Great Awakening</a></b></td>"
[31] "<td><b><a href=\"/movie/Erupcja-(2026)\">Erupcja</a></b></td>"
```

The duplicated set of movie names are now gone.

We still have a lot of excess symbols and HTML code to eliminate before we can have a neat list of movie names. HTML tags always have the form `<some code here>`. Therefore, we should remove any text between a less than and greater than symbol. Here is a regular expression that looks for a `<` followed by any number of characters that are not `>`, and then a closing `>`… and `gsub()` will delete them.

```
gsub("<[^>]*>", "", a[i])
```

```
 [1] "Minions &amp;amp; Monsters"
 [2] "Young Washington"
 [3] "Toy Story 5"
 [4] "Supergirl"
 [5] "Disclosure Day"
 [6] "Obsession"
 [7] "Backrooms"
 [8] "Jackass: Best and Last"
 [9] "Scary Movie"
[10] "Masters of the Universe"
[11] "The Invite"
[12] "Star Wars: The Mandalorian and Grogu"
[13] "Michael"
[14] "Leviticus"
[15] "Lucky Strike"
[16] "The Devil Wears Prada 2"
[17] "Tuner"
[18] "The Sheep Detectives"
[19] "The Furious"
[20] "The Death of Robin Hood"
[21] "Rose of Nevada"
[22] "Stop! That! Train!"
[23] "I Love Boosters"
[24] "Mortal Kombat II"
[25] "Couture"
[26] "Terminator 2: Judgment Day"
[27] "Time And Water"
[28] "Silent Friend"
[29] "Unidentified"
[30] "A Great Awakening"
[31] "Erupcja"
```

Perfect! Now we just have movie names. You will see some movie names have strange symbols, like `&`, `&hellip;`, or `&#039;`. These are the HTML codes for the ampersand "&", horizontal

ellipses "…" and a smart apostrophe. These make the text look prettier on a webpage. In a moment, we will use `textutils::HTMLdecode()` to clean these up.

Let's put these movie names in a data frame, `data0`. This data frame currently has only one column.

```
data0 <- data.frame(movie=gsub("<[^>]*>", "", a[i]))
```

Now we also want to get the daily gross for each movie. Let's take another look at the HTML code for *Minions and Monsters*.

```
a[i[1] + 0:8]
```

```
[1] "<td><b><a href=\"/movie/Minions-and-Monsters-(2026)\">Minions &amp; Monsters</a></b></t
[2] "<td class=\"data\">$9,491,820</td>"
[3] "<td class=\"data chart_down d\">-42%</td>"
[4] "<td class=\"data Daily\"> </td>"
[5] "<td class=\"data\">4,243</td>"
[6] "<td class=\"data\">$2,237</td>"
[7] "<td class=\"data\">$51,039,330</td>"
[8] "<td class=\"data\">4</td>"
[9] "</tr>"
```

Note that the movie gross is one line after the movie name. It turns out that this is consistent for all movies. Since `i` has the line numbers for the movie names, then `i+1` must be the line numbers containing the daily gross.

```
a[i+1]
```

```
 [1] "<td class=\"data\">$9,491,820</td>" "<td class=\"data\">$7,672,967</td>"
 [3] "<td class=\"data\">$7,537,411</td>" "<td class=\"data\">$2,506,574</td>"
 [5] "<td class=\"data\">$1,858,350</td>" "<td class=\"data\">$1,373,315</td>"
 [7] "<td class=\"data\">$791,242</td>"   "<td class=\"data\">$666,427</td>"
 [9] "<td class=\"data\">$309,488</td>"   "<td class=\"data\">$218,542</td>"
[11] "<td class=\"data\">$216,024</td>"   "<td class=\"data\">$165,284</td>"
[13] "<td class=\"data\">$154,717</td>"   "<td class=\"data\">$85,440</td>"
[15] "<td class=\"data\">$58,960</td>"    "<td class=\"data\">$48,039</td>"
[17] "<td class=\"data\">$47,984</td>"    "<td class=\"data\">$45,200</td>"
[19] "<td class=\"data\">$32,433</td>"    "<td class=\"data\">$16,289</td>"
[21] "<td class=\"data\">$13,596</td>"    "<td class=\"data\">$12,598</td>"
[23] "<td class=\"data\">$11,523</td>"    "<td class=\"data\">$7,695</td>"
```

```
[25] "<td class=\"data\">$3,513</td>"     "<td class=\"data\">$1,897</td>"
[27] "<td class=\"data\">$1,336</td>"     "<td class=\"data\">$674</td>"
[29] "<td class=\"data\">$289</td>"       "<td class=\"data\">$253</td>"
[31] "<td class=\"data\">$24</td>"
```

Again we need to strip out the HTML tags. We will also remove the dollar signs and commas so that R will recognize it as a number. We will add this to `data0` also.

```
data0$gross <- as.numeric(gsub("<[^>]*>|[$,]", "", a[i+1]))
```

Take a look at the webpage and compare it to the dataset you have now created. All the values should now match.

```
head(data0)
```

```
                  movie   gross
1 Minions &amp; Monsters 9491820
2       Young Washington 7672967
3            Toy Story 5 7537411
4              Supergirl 2506574
5         Disclosure Day 1858350
6              Obsession 1373315
```

```
tail(data0)
```

```
                        movie gross
26 Terminator 2: Judgment Day  1897
27             Time And Water  1336
28              Silent Friend   674
29               Unidentified   289
30          A Great Awakening   253
31                    Erupcja    24
```

Exercises

1. What movie that was in theaters one week ago had the longest Days in Release?

2.1 Movies with no titles

Some movies have missing titles. Have a look at the February 21, 2011 movies.

```
repeat
{
   a <- try(scan("https://www.the-numbers.com/box-office-chart/daily/2011/02/21",
             what="",sep="\n"))
   if(!inherits(a,"try-error")) break
   Sys.sleep(10)
}
i <- grep("Genesis-Code", a)[1]
a[-10:10 + i]
```

```
 [1] "<td class=\"data  d\"> </td>"
 [2] "<td class=\"data Daily\"> </td>"
 [3] "<td class=\"data\">12</td>"
 [4] "<td class=\"data\">$155</td>"
 [5] "<td class=\"data\">$951,246</td>"
 [6] "<td class=\"data\">67</td>"
 [7] "</tr>"
 [8] "<tr>"
 [9] "<td class=\"data\">62</td>"
[10] "<td class=\"data\">(-)</td>"
[11] "<td><b><a href=\"/movie/Genesis-Code-The-(2010)\"></a></b></td>"
[12] "<td class=\"data chart_estimate\">$1,800</td>"
[13] "<td class=\"data  d\"> </td>"
[14] "<td class=\"data Daily\"> </td>"
[15] "<td class=\"data\">17</td>"
[16] "<td class=\"data\">$106</td>"
[17] "<td class=\"data\">$20,300</td>"
[18] "<td class=\"data\">4</td>"
[19] "</tr>"
[20] "<tr>"
[21] "<td class=\"data\">63</td>"
```

Note that the line for *The Genesis Code* has the movie title in the `href` attribute, but no movie title is between the `<a></a>` tags. In these cases let's pull the movie title from the `href` attribute.

```
i <- grep('href=\"/movie/',a)
i <- i[i < grep("chart-mobile", a)]
data0 <- data.frame(movie = gsub("<[^>]*>","",a[i]),
                    gross = as.numeric(gsub("<[^>]*>|[,$]","",a[i+1])))
# which ones are blank?
j <- which(data0$movie=="")
a[i[j]]
```

```
[1] "<td><b><a href=\"/movie/Genesis-Code-The-(2010)\"></a></b></td>"
[2] "<td><b><a href=\"/movie/Rauber-Der\"></a></b></td>"
```

```
# test regex to pull movie title
gsub('.*/movie/([^"]*)".*', "\\1", a[i[j]])
```

```
[1] "Genesis-Code-The-(2010)" "Rauber-Der"
```

```
# replace empty movie names
data0$movie[j] <- gsub('.*/movie/([^"]*)".*', "\\1", a[i[j]]) |>
  gsub("-", " ", x=_)
```

3 Scraping Multiple Pages

We have now successfully scraped data for one day. This is usually the hardest part. But if we have R code that can correctly scrape one day's worth of data *and* the website is consistent across days, then it is simple to adapt our code to work for *all* days. So let's get all movie data from January 1, 2010 through July 31, 2026. That means we are going to be web scraping 6,056 pages of data.

First note that the URL for July 4, 2026 was

`https://www.the-numbers.com/box-office-chart/daily/2026/07/04`

We can extract data from any other date by using the same URL, but changing the ending to match the date that we want. Importantly, the 07 and the 04 in the URL must have leading zeros for the URL to return the correct page.

To start, let's make a list of all the dates that we intend to scrape.

```
library(lubridate)
# create a sequence of all days to scrape
dates2scrape <- seq(ymd("2010-01-01"), ymd("2026-07-31"), by="days")
```

Now `dates2scrape` contains a collection of all the dates with movie data that we wish to scrape.

```
dates2scrape[1:5]
```

```
[1] "2010-01-01" "2010-01-02" "2010-01-03" "2010-01-04" "2010-01-05"
```

```
# gsub() to change - to / matching appearance of thenumbers.com URL
gsub("-", "/", dates2scrape[1:5])
```

```
[1] "2010/01/01" "2010/01/02" "2010/01/03" "2010/01/04" "2010/01/05"
```

Our plan is to construct a for-loop within which we will construct a URL from `dates2scrape`, pull down the HTML code from that URL, scrape the movie data into a data frame, and then combine each day's data frame into one data frame with all of the movie data. First we create a list that will contain each day's data frame.

```
results <- vector("list", length(dates2scrape))
```

On iteration `i` of our for loop we will store that day's movie data frame in `results[[i]]`. The following for loop can take several minutes to run and its speed will depend on your network connection and how responsive the web site is. Before running the entire for loop, it may be a good idea to temporarily set the dates to a short period of time (e.g., a month or two) just to verify that your code is functioning properly. Once you have concluded that the code is doing what you want it to do, you can set the dates so that the for loop runs for the entire analysis period.

This loop takes about an hour to pull all the data.

```
timeStart <- Sys.time() # record the starting time
for(iDate in 1:length(dates2scrape))
{
   # useful to know how much is left to go... periodic messages
   if((iDate <= 10) || (iDate %% 30 == 0))
   {
      cat(as.character(dates2scrape[iDate]), "\n", file=stderr())
   }

   # construct URL
   urlText <- paste0("https://www.the-numbers.com/box-office-chart/daily/",
```

```
                    gsub("-", "/", dates2scrape[iDate]))

   # read in the HTML code
   tries <- 0
   repeat
   {
      tries <- tries + 1
      a <- try(scan(urlText, what="", sep="\n",
                    fileEncoding="UTF-8",
                    quiet = TRUE))
      if(!inherits(a, "try-error") || tries >= 5) break
      Sys.sleep(10)
   }
   # skip this date if all 5 attempts failed
   if(inherits(a, "try-error")) next

   # keep only those before the mobile version of the page
   i <- grep("chart-mobile", a)
   a <- a[1:i]

   # find movies
   i <- grep('href="/movie/', a)

   # get movie names and gross
   data0 <- data.frame(movie = gsub("<[^>]*>", "", a[i]),
                       gross = as.numeric(gsub("<[^>]*>|[$,]","",a[i+1])),
                       date  = dates2scrape[iDate])

   # replace empty movie names
   j <- which(data0$movie=="")
   data0$movie[j] <- gsub('.*/movie/([^"]*)".*', "\\1", a[i[j]]) |>
      gsub("-", " ", x=_)

   results[[iDate]] <- data0
}
```

```
Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2011/04/10': Timeout of 60
seconds was reached
```

```
Warning in file(file, "r", encoding = fileEncoding): URL
```

```
'https://www.the-numbers.com/box-office-chart/daily/2012/03/15': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2014/08/23': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2015/02/08': status was
'SSL connect error'

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2015/10/21': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2017/07/06': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2019/02/09': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2019/08/14': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2019/08/21': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2019/12/05': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2020/10/31': Timeout of 60
seconds was reached
```

```
Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2024/02/19': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2024/07/02': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2024/10/26': Timeout of 60
seconds was reached

Warning in file(file, "r", encoding = fileEncoding): URL
'https://www.the-numbers.com/box-office-chart/daily/2026/04/30': Timeout of 60
seconds was reached
```

```
# calculate how long it took
timeEnd <- Sys.time()
```

How long did that take?

```
timeEnd-timeStart
```

```
Time difference of 51.21877 mins
```

Let's look at the first 3 lines of the first and last 3 days.

```
# first 6 rows of first 3 days
results |> head(3) |> lapply(head)
```

```
[[1]]
                                    movie    gross       date
1                                  Avatar 25274008 2010-01-01
2                           Sherlock Holmes 14889882 2010-01-01
3 Alvin and the Chipmunks: The Squeakquel 12998264 2010-01-01
4                          It's Complicated  7127425 2010-01-01
5                            The Blind Side  4554779 2010-01-01
6                             Up in the Air  4112263 2010-01-01

[[2]]
```

```
                                    movie    gross       date
1                                  Avatar 25835551 2010-01-02
2                           Sherlock Holmes 14373564 2010-01-02
3 Alvin and the Chipmunks: The Squeakquel 14373273 2010-01-02
4                          It's Complicated  7691535 2010-01-02
5                            The Blind Side  4997659 2010-01-02
6                             Up in the Air  4457565 2010-01-02

[[3]]
                                    movie    gross       date
1                                  Avatar 17381129 2010-01-03
2 Alvin and the Chipmunks: The Squeakquel  7818116 2010-01-03
3                           Sherlock Holmes  7349035 2010-01-03
4                          It's Complicated  3984005 2010-01-03
5                            The Blind Side  2360311 2010-01-03
6                The Princess and the Frog  2264727 2010-01-03
```

```r
# first 6 rows of last 3 days
results |> tail(3) |> lapply(head)
```

```
[[1]]
                   movie    gross       date
1            The Odyssey 13624580 2026-07-29
2 Minions &amp;amp; Monsters  1797555 2026-07-29
3                  Moana  1716912 2026-07-29
4            Toy Story 5  1714172 2026-07-29
5 Hadestown: The Musical  1466084 2026-07-29
6             The Invite   493802 2026-07-29

[[2]]
                              movie    gross       date
1 PreviewsSpider-Man: Brand New Day 72000000 2026-07-30
2                       The Odyssey 10347260 2026-07-30
3            Minions &amp;amp; Monsters  1222005 2026-07-30
4                       Toy Story 5  1210404 2026-07-30
5                             Moana  1156815 2026-07-30
6            Hadestown: The Musical   422134 2026-07-30

[[3]]
                      movie     gross       date
1 Spider-Man: Brand New Day 169300000 2026-07-31
2               The Odyssey  14340000 2026-07-31
```

```
3               Toy Story 5   1800000 2026-07-31
4    Minions &amp;amp; Monsters   1690000 2026-07-31
5                     Moana   1600000 2026-07-31
6    Hadestown: The Musical    561807 2026-07-31
```

Looks like we got them all. Now let's combine them into one big data frame. `bind_rows()` takes a list of data frames, like `results[[1]]`, `results[[2]]`, ..., and stacks them all on top of each other.

```
movieData <- bind_rows(results)

# check that the number of rows and dates seem reasonable
nrow(movieData)
```

```
[1] 237158
```

```
range(movieData$date)
```

```
[1] "2010-01-01" "2026-07-31"
```

```
head(movieData)
```

```
                                    movie    gross       date
1                                  Avatar 25274008 2010-01-01
2                           Sherlock Holmes 14889882 2010-01-01
3 Alvin and the Chipmunks: The Squeakquel 12998264 2010-01-01
4                          It's Complicated  7127425 2010-01-01
5                            The Blind Side  4554779 2010-01-01
6                             Up in the Air  4112263 2010-01-01
```

```
tail(movieData)
```

```
                                      movie  gross       date
237153                            Backrooms 122958 2026-07-31
237154                        Disclosure Day  60000 2026-07-31
237155                       Sheep in the Box  25514 2026-07-31
237156                       Her Private Hell  15000 2026-07-31
237157 Star Wars: The Mandalorian and Grogu   5000 2026-07-31
237158               The Death of Robin Hood   2874 2026-07-31
```

If you ran that for-loop to gather over 15 years worth of data, most likely you walked away from your computer to do something more interesting than watch its progress. In these situations, I like to send myself a text message when it is complete. The `emayili` package is a convenient way to send yourself an email or text. If you fill it in with your email, username, and gmail app password, the following code will send you an email or text message when the script reaches this point.

```
library(emayili)

# https://myaccount.google.com/apppasswords
#    get a 16 character "app password"
smtp <- server(host = "smtp.gmail.com",
               port = 587,
               username = "you@gmail.com",
               password = "REPLACE WITH 16 CHARACTER APP PASSWORD")

# Verizon:  5551234567@vtext.com
# AT&T:     5551234567@txt.att.net
# T-Mobile: 5551234567@tmomail.net
email <- envelope() |>
   from("you@gmail.com") |>
   to("5551234567@tmomail.net") |>
   text("Come back! Your movie data is ready!")

smtp(email, verbose = TRUE)
```

Note that the password here is in plain text so do not try this on a public computer. R also saves your history so even if it is not on the screen it might be saved somewhere else on the computer.

4 Parallel Computing

Since 1965 Moore's Law has predicted the power of computation over time. Moore's Law predicted the doubling of transistors about every two years. Moore's prediction has held true for decades. However, to get that speed the transistors were made smaller and smaller. Moore's Law cannot continue indefinitely. The diameter of a silicon atom is 0.2nm. Transistors today contain less than 70 atoms and some transistor dimensions are between 10nm and 40nm. Since 2012, computing power has not changed greatly signaling that we might be getting close to the end of Moore's Law, at least with silicon-based computing. What has changed is the widespread use of multicore processors. Rather than having a single processor, a typical laptop

might have an 8 or 16 core processor (meaning they have 8 or 16 processors that share some resources like high speed memory). Penn's PARCC computing cluster has 640.

R can guess how many cores your computer has on hand.

```
library(future)
library(doFuture)
parallelly::availableCores()
```

```
system
    16
```

Having access to multiple cores allows you to write scripts that send different tasks to different processors to work on simultaneously. While one processor is busy scraping the data for January 1st, the second can get to work on January 2nd, and another can work on January 3rd. All the processors will be fighting over the one connection you have to the internet, but they can `grep()` and `gsub()` at the same time other processors are working on other dates.

To write a script to work in parallel, you will need the `foreach` and `future` packages. Let's first test whether parallelization actually speeds things up. There are two `foreach` loops below. In both of them, each iteration of the loop does not really do anything except pause for 2 seconds. The first loop, which does not use parallelization, includes 10 iterations and so should take 20 seconds to run. The second `foreach` loop looks the same, except right before the `foreach` loop we have told R to make use of two of the computer's processors rather than the default of one processor. With two processors, each processor will generally handle five iterations and sleep for 2 seconds five times. The exact allocation can depend on how the parallel tasks are scheduled. In total this should take about 10 seconds.

```
library(foreach)

# should take 10*2=20 seconds
system.time( # time how long this takes
  foreach(i=1:10) %do% # not in parallel
  {
     Sys.sleep(2)  # wait for 2 seconds
     return(i)
  }
)
```

```
   user  system elapsed
   0.01    0.00   20.45
```

```
# set up R to use 2 cores
plan(multisession, workers = 2)
# tells %dopar% to use the plan's 2 cores
registerDoFuture()

# with two processors should take about 10 seconds
system.time(
  foreach(i=1:10) %dopar% # run in parallel
  {
    Sys.sleep(2)
    return(i)
  }
)
```

```
   user  system elapsed
   0.19    0.05   10.32
```

Sure enough, the parallel implementation was able to complete 20 seconds worth of sleeping in about 10 seconds. To set up code to run in parallel, the key steps are to set up the cores using `plan()` and to tell parallel `foreach()` to use that cluster of processors with `registerDoFuture()`. Note that the key difference between the two `foreach()` statements is that the first `foreach()` is followed by a `%do%` while the second is followed by a `%dopar%`. When `foreach()` sees the `%dopar%` it will check what was set up in the `registerDoFuture()` call and spread the computation among those cores.

Note that the `foreach()` differs a little bit in its syntax compared with our previous use of for-loops. While for-loops have the syntax `for(i in 1:10)` the syntax for `foreach()` looks like `foreach(i=1:10)` and is followed by a `%do%` or a `%dopar%`. Lastly, note that the final step inside the `{ }` following a `foreach()` is a `return()` statement. `foreach()` will take the returned values of each of the iterations and assemble them into a single list by default. In the following `foreach()` we have added `.combine=bind_rows` to the `foreach()` so that the final results will be stacked into one data frame, avoiding the need for a separate `bind_rows()` like we used previously.

Parallelization introduces some complications. If anything goes wrong in a parallelized script, then the whole `foreach()` fails. For example, let's say that after scraping movie data from 2000-2016 you briefly lose your internet connection. If this happens, then `scan()` fails and the whole `foreach()` will end with an error, tossing all of your completed work. To avoid this you need to either be sure you have a solid internet connection, or wrap the call to `scan()` in a `try()` and a `repeat` loop that is smart enough to wait a few seconds and try the scan again rather than fail completely.

This causes an error since this website does not exist (or not yet!).

```
res <- scan("https://www.jaywalkingIsNotACrime.org", what="", sep="\n")
```

```
Warning in file(file, "r"): URL 'https://www.jaywalkingIsNotACrime.org/':
status was 'Could not resolve hostname'

Error in `file()`:
! cannot open the connection to 'https://www.jaywalkingIsNotACrime.org'
```

```
# res does not exist
res
```

```
Error:
! object 'res' not found
```

If we wrap `scan()` with `try()`, then we can catch the error and write R code to gracefully handle the problem.

```
res <- try(scan("https://www.jaywalkingIsNotACrime.org", what="", sep="\n"),
           silent = TRUE) |>
   suppressWarnings()
is(res)
```

```
[1] "try-error"
```

```
if(inherits(res, "try-error"))
{
   message("Could not find that website")
} else
{
   message("Found that website")
}
```

```
Could not find that website
```

With all this in mind, let's web scrape the movie data using multiple cores with `try()`/`repeat{}`. Typically, any attempts to print from inside a parallel `foreach()` do not appear in the console, since that print is running in a separate, parallel R session. The `progressr` package offers a way to print a progress bar to the console that also offers an estimated time to completion.

```
# setup a Command-Line Interface progress bar
library(progressr)
handlers("cli")

plan(multisession, workers = 8)
registerDoFuture()

timeStart <- Sys.time() # record the starting time
# wrap the foreach inside the progress monitor
movieData <- with_progress(
{
   # create a progress bar how many total steps in the foreach loop
   p <- progressor(steps = length(dates2scrape))

   result <- foreach(iDate=1:length(dates2scrape),
                     .combine = bind_rows) %dopar%
   {
      # update progress bar
      p(paste("Working on", dates2scrape[iDate]))
      urlText <- paste0("https://www.the-numbers.com/box-office-chart/daily/",
                        gsub("-", "/", dates2scrape[iDate]))

      # retry up to 10 times with 10 sec sleep if fail
      tries <- 0
      repeat
      {
         tries <- tries + 1
         a <- try(scan(urlText, what = "", sep = "\n",
                       fileEncoding = "UTF-8", quiet = TRUE),
                  silent = TRUE)
         if(!inherits(a, "try-error") || tries >= 10) break
         Sys.sleep(10)
      }
      # skip this date on persistent failure
      if(inherits(a, "try-error")) return(NULL)

      i <- grep("chart-mobile",a)[1]
      a <- a[1:i]

      i <- grep('href="/movie/', a)
      data0 <- data.frame(movie = gsub("<[^>]*>", "", a[i]),
                          gross = as.numeric(gsub("<[^>]*>|[$,]","",a[i+1])),
```

```
                          date  = dates2scrape[iDate])

      # replace empty movie names
      j <- which(data0$movie=="")
      if(length(j) > 0)
      {
         data0$movie[j] <- gsub('.*/movie/([^"]*)".*', "\\1", a[i[j]]) |>
            gsub("-", " ", x=_)
      }

      return(data0)
   }

   # the last object in with_progress() will be returned
   result
})

# calculate how long it took
timeEnd <- Sys.time()
timeEnd-timeStart
```

This code made use of 8 processors. Unlike our 2 second sleep example, this script may not be exactly 8 times faster. Each processor still needs to wait its turn in order to pull down its webpage from the internet. However, you should observe the parallel version finishing much sooner than the first version. In just a few lines of code and about 10 minutes of waiting, you now have 15 years worth of movie data.

Before moving on, let's do a final check that everything looks okay.

```
nrow(movieData)
```

```
[1] 237158
```

```
range(movieData$date)
```

```
[1] "2010-01-01" "2026-07-31"
```

```
head(movieData)
```

```
                                     movie    gross       date
1                                   Avatar 25274008 2010-01-01
2                            Sherlock Holmes 14889882 2010-01-01
3 Alvin and the Chipmunks: The Squeakquel 12998264 2010-01-01
4                           It's Complicated  7127425 2010-01-01
5                             The Blind Side  4554779 2010-01-01
6                              Up in the Air  4112263 2010-01-01
```

```
tail(movieData)
```

```
                                      movie  gross       date
237153                            Backrooms 122958 2026-07-31
237154                      Disclosure Day  60000 2026-07-31
237155                     Sheep in the Box  25514 2026-07-31
237156                     Her Private Hell  15000 2026-07-31
237157 Star Wars: The Mandalorian and Grogu   5000 2026-07-31
237158               The Death of Robin Hood   2874 2026-07-31
```

Check for movie names with HTML codes.

```
i <- grep("&[#0-9A-Za-z]+;", movieData$movie)
movieData$movie[i] |> unique()
```

```
  [1] "The Slammin&#039; Salmon"
  [2] "Le combat dans l&#039;île"
  [3] "Valentine&#039;s Day"
  [4] "Percy Jackson &amp; the Olympians: The Lightning Thief"
  [5] "She&#039;s Out of My League"
  [6] "George A. Romero&#039;s Survival of the Dead"
  [7] "Winter&#039;s Bone"
  [8] "Gangster&#039;s Paradise: Jerusalema"
  [9] "The Sorcerer&#039;s Apprentice"
 [10] "Cats &amp; Dogs: The Revenge of Kitty Galore"
 [11] "The People I&#039;ve Slept With"
 [12] "Mao&#039;s Last Dancer"
 [13] "L&#039;armée du crime"
 [14] "It&#039;s Kind of a Funny Story"
 [15] "Today&#039;s Special"
 [16] "The Warrior&#039;s Way"
 [17] "Saint Misbehavin&#039;: The Wavy Gravy Movie"
 [18] "Barney&#039;s Version"
```

```
[19] "Now &amp; Later"
[20] "The Girl Who Kicked the Hornet&#039;s Nest"
[21] "Journal d&#039;un curé de campagne"
[22] "Madea&#039;s Big Happy Family"
[23] "I&#039;m Not Jesus Mommy"
[24] "Meek&#039;s Cutoff"
[25] "L&#039;Amour Fou"
[26] "Henry&#039;s Crime"
[27] "HEY BOO: Harper Lee and &quot;To Kill a Mockingbird&quot;"
[28] "Mr. Popper&#039;s Penguins"
[29] "Conan O&#039;Brien Can&#039;t Stop"
[30] "Sarah&#039;s Key"
[31] "The Devil&#039;s Double"
[32] "Don&#039;t Be Afraid of the Dark "
[33] "Beats, Rhymes &amp; Life: The Travels of a Tribe Called Quest"
[34] "I&#039;m Glad My Mother is Alive"
[35] "I Don&#039;t Know How She Does It"
[36] "What&#039;s Your Number?"
[37] "A Very Harold &amp; Kumar 3D Christmas"
[38] "Tyler Perry&#039;s Good Deeds"
[39] "Dr. Seuss&#039; The Lorax"
[40] "What to Expect When You&#039;re Expecting"
[41] "Madagascar 3: Europe&#039;s Most Wanted"
[42] "BRO&#039;"
[43] "Tyler Perry&#039;s Madea&#039;s Witness Protection"
[44] "Hit &amp; Run"
[45] "Won&#039;t Back Down"
[46] "&#039;Luv Shuv Tey Chicken Khurana"
[47] "Hansel &amp; Gretel: Witch Hunters"
[48] "Free Angela &amp; All Political Prisoners"
[49] "Pain &amp; Gain"
[50] "Dead Man&#039;s Burden"
[51] "Lee Daniels&#039; The Butler"
[52] "The World&#039;s End"
[53] "You&#039;re Next"
[54] "Jayne Mansfield&#039;s Car"
[55] "Romeo &amp; Juliet"
[56] "I&#039;m in Love with a Church Girl"
[57] "Return to Nuke &#039;Em High Volume 1"
[58] "Devil&#039;s Due"
[59] "Ernest &amp; Celestine"
[60] "Mr. Peabody &amp; Sherman"
[61] "Tyler Perry&#039;s The Single Moms Club"
```

```
 [62] "Child&#039;s Pose"
 [63] "Water &amp; Power"
 [64] "Kirk Cameron&#039;s Saving Christmas"
 [65] "The Devil&#039;s Violinist"
 [66] "While We&#039;re Young"
 [67] "I&#039;ll See You in My Dreams"
 [68] "Love &amp; Mercy"
 [69] "Jimmy&#039;s Hall"
 [70] "Dragon Ball Z: Resurrection &quot;F&quot;"
 [71] "Kahlil Gibran&#039;s The Prophet"
 [72] "She&#039;s Funny That Way"
 [73] "Hell &amp; Back"
 [74] "This Isn&#039;t Funny"
 [75] "Kapoor &amp; Sons"
 [76] "Elvis &amp; Nixon"
 [77] "Love &amp; Friendship"
 [78] "My Love, Don&#039;t Cross That River"
 [79] "Professor Marston &amp; The Wonder Women"
 [80] "Gosnell: The Trial of America&#039;s Biggest Serial Killer"
 [81] "The Big Bad Fox &amp; Other Tales"
 [82] "Beauty &amp; the Beholder"
 [83] "Holmes &amp; Watson"
 [84] "Faith, Hope &amp; Love"
 [85] "Marianne &amp; Leonard: Words of Love"
 [86] "PreviewsFast &amp; Furious Presents: Hobbs &amp; Shaw"
 [87] "Fast &amp; Furious Presents: Hobbs &amp; Shaw"
 [88] "Vita &amp; Virginia"
 [89] "Queen &amp; Slim"
 [90] "PreviewsGretel &amp; Hansel"
 [91] "Gretel &amp; Hansel"
 [92] "Bill &amp; Ted Face the Music"
 [93] "The Emperor&#039;s New Groove"
 [94] "Dating &amp; New York"
 [95] "Green Ghost &amp; The Masters Of The Stone"
 [96] "My Donkey, My Lover &amp; I"
 [97] "Mack &amp; Rita"
 [98] "Gigi &amp; Nate"
 [99] "The Chosen Season 3: Episodes 1 &amp; 2"
[100] "PreviewsDungeons &amp; Dragons: Honor Among Thieves"
[101] "Dungeons &amp; Dragons: Honor Among Thieves"
[102] "The Melt Goes on Forever: The Art &amp; Times of David Hammons"
[103] "Ernest &amp; Celestine: A Trip to Gibberitia"
[104] "Mother Teresa &amp; Me"
```

```
[105] "PreviewsThe Hunger Games: The Ballad of Songbirds &amp;amp; Snakes"
[106] "The Hunger Games: The Ballad of Songbirds &amp;amp; Snakes"
[107] "Deer Camp &amp;#039;86"
[108] "Jatt &amp;amp; Juliet 3"
[109] "PreviewsDeadpool &amp;amp; Wolverine"
[110] "Deadpool &amp;amp; Wolverine"
[111] "Sight &amp;amp; Sound Presents: Daniel LIVE"
[112] "The Cowboy &amp;amp; the Queen"
[113] "Howl&amp;#039;s Moving Castle"
[114] "Love &amp;amp; Pop"
[115] "Pride &amp;amp; Prejudice"
[116] "Sod &amp;amp; Stubble"
[117] "Juliet &amp;amp; Romeo"
[118] "PreviewsLilo &amp;amp; Stitch"
[119] "Lilo &amp;amp; Stitch"
[120] "Tim Travers &amp;amp; the Time Traveler's Paradox"
[121] "given The Double Album: Hiiragi mix &amp;amp; To the Sea Double Feature"
[122] "Hearts of Darkness: A Filmmaker&amp;#039;s Apocalypse"
[123] "Sight &amp;amp; Sound Presents: NOAH Live!"
[124] "Truth &amp;amp; Treason"
[125] "PreviewsYou, Me &amp;amp; Tuscany"
[126] "You, Me &amp;amp; Tuscany"
[127] "Minions &amp;amp; Monsters"
[128] "Evangelion: Death (True)2 &amp;amp; Rebirth"
```

Those HTML characters in movie titles are annoying to look at. Let's fix it now.

```
# change HTML codes to something prettier
movieData <- movieData |>
   mutate(movie = textutils::HTMLdecode(movie))

# check that it worked
movieData$movie[i] |> unique()
```

```
  [1] "The Slammin' Salmon"
  [2] "Le combat dans l'île"
  [3] "Valentine's Day"
  [4] "Percy Jackson & the Olympians: The Lightning Thief"
  [5] "She's Out of My League"
  [6] "George A. Romero's Survival of the Dead"
  [7] "Winter's Bone"
  [8] "Gangster's Paradise: Jerusalema"
```

```
 [9] "The Sorcerer's Apprentice"
[10] "Cats & Dogs: The Revenge of Kitty Galore"
[11] "The People I've Slept With"
[12] "Mao's Last Dancer"
[13] "L'armée du crime"
[14] "It's Kind of a Funny Story"
[15] "Today's Special"
[16] "The Warrior's Way"
[17] "Saint Misbehavin': The Wavy Gravy Movie"
[18] "Barney's Version"
[19] "Now & Later"
[20] "The Girl Who Kicked the Hornet's Nest"
[21] "Journal d'un curé de campagne"
[22] "Madea's Big Happy Family"
[23] "I'm Not Jesus Mommy"
[24] "Meek's Cutoff"
[25] "L'Amour Fou"
[26] "Henry's Crime"
[27] "HEY BOO: Harper Lee and \"To Kill a Mockingbird\""
[28] "Mr. Popper's Penguins"
[29] "Conan O'Brien Can't Stop"
[30] "Sarah's Key"
[31] "The Devil's Double"
[32] "Don't Be Afraid of the Dark "
[33] "Beats, Rhymes & Life: The Travels of a Tribe Called Quest"
[34] "I'm Glad My Mother is Alive"
[35] "I Don't Know How She Does It"
[36] "What's Your Number?"
[37] "A Very Harold & Kumar 3D Christmas"
[38] "Tyler Perry's Good Deeds"
[39] "Dr. Seuss' The Lorax"
[40] "What to Expect When You're Expecting"
[41] "Madagascar 3: Europe's Most Wanted"
[42] "BRO'"
[43] "Tyler Perry's Madea's Witness Protection"
[44] "Hit & Run"
[45] "Won't Back Down"
[46] "'Luv Shuv Tey Chicken Khurana"
[47] "Hansel & Gretel: Witch Hunters"
[48] "Free Angela & All Political Prisoners"
[49] "Pain & Gain"
[50] "Dead Man's Burden"
[51] "Lee Daniels' The Butler"
```

```
[52] "The World's End"
[53] "You're Next"
[54] "Jayne Mansfield's Car"
[55] "Romeo & Juliet"
[56] "I'm in Love with a Church Girl"
[57] "Return to Nuke 'Em High Volume 1"
[58] "Devil's Due"
[59] "Ernest & Celestine"
[60] "Mr. Peabody & Sherman"
[61] "Tyler Perry's The Single Moms Club"
[62] "Child's Pose"
[63] "Water & Power"
[64] "Kirk Cameron's Saving Christmas"
[65] "The Devil's Violinist"
[66] "While We're Young"
[67] "I'll See You in My Dreams"
[68] "Love & Mercy"
[69] "Jimmy's Hall"
[70] "Dragon Ball Z: Resurrection \"F\""
[71] "Kahlil Gibran's The Prophet"
[72] "She's Funny That Way"
[73] "Hell & Back"
[74] "This Isn't Funny"
[75] "Kapoor & Sons"
[76] "Elvis & Nixon"
[77] "Love & Friendship"
[78] "My Love, Don't Cross That River"
[79] "Professor Marston & The Wonder Women"
[80] "Gosnell: The Trial of America's Biggest Serial Killer"
[81] "The Big Bad Fox & Other Tales"
[82] "Beauty & the Beholder"
[83] "Holmes & Watson"
[84] "Faith, Hope & Love"
[85] "Marianne & Leonard: Words of Love"
[86] "PreviewsFast & Furious Presents: Hobbs & Shaw"
[87] "Fast & Furious Presents: Hobbs & Shaw"
[88] "Vita & Virginia"
[89] "Queen & Slim"
[90] "PreviewsGretel & Hansel"
[91] "Gretel & Hansel"
[92] "Bill & Ted Face the Music"
[93] "The Emperor's New Groove"
[94] "Dating & New York"
```

```
 [95] "Green Ghost & The Masters Of The Stone"
 [96] "My Donkey, My Lover & I"
 [97] "Mack & Rita"
 [98] "Gigi & Nate"
 [99] "The Chosen Season 3: Episodes 1 & 2"
[100] "PreviewsDungeons & Dragons: Honor Among Thieves"
[101] "Dungeons & Dragons: Honor Among Thieves"
[102] "The Melt Goes on Forever: The Art & Times of David Hammons"
[103] "Ernest & Celestine: A Trip to Gibberitia"
[104] "Mother Teresa & Me"
[105] "PreviewsThe Hunger Games: The Ballad of Songbirds & Snakes"
[106] "The Hunger Games: The Ballad of Songbirds & Snakes"
[107] "Deer Camp '86"
[108] "Jatt & Juliet 3"
[109] "PreviewsDeadpool & Wolverine"
[110] "Deadpool & Wolverine"
[111] "Sight & Sound Presents: Daniel LIVE"
[112] "The Cowboy & the Queen"
[113] "Howl's Moving Castle"
[114] "Love & Pop"
[115] "Pride & Prejudice"
[116] "Sod & Stubble"
[117] "Juliet & Romeo"
[118] "PreviewsLilo & Stitch"
[119] "Lilo & Stitch"
[120] "Tim Travers & the Time Traveler’s Paradox"
[121] "given The Double Album: Hiiragi mix & To the Sea Double Feature"
[122] "Hearts of Darkness: A Filmmaker's Apocalypse"
[123] "Sight & Sound Presents: NOAH Live!"
[124] "Truth & Treason"
[125] "PreviewsYou, Me & Tuscany"
[126] "You, Me & Tuscany"
[127] "Minions & Monsters"
[128] "Evangelion: Death (True)2 & Rebirth"
```

It is probably wise at this point to save `movieData` so that you will not have to rerun this if you mess up your dataset. With `movieData` saved you should feel free to test out your ideas. You can always `load("movieData.RData")` if you make a mistake.

```
save(movieData, file="movieData.RData", compress=TRUE)
```

Exercises

2. What were the top grossing movies on July 4th for 2010 to the present? Run in parallel.

5 Fun With Movie Data

You can use the dataset to answer questions such as "which movie yielded the largest single-day gross?"

```
movieData |> slice_max(gross)
```

```
                     movie     gross       date
1 Spider-Man: Brand New Day 169300000 2026-07-31
```

Which ten movies had the largest total gross during the period this dataset covers?

```
movieData |>
   summarize(gross=sum(gross), .by=movie) |>
   slice_max(gross, n=10)
```

```
                                  movie     gross
1  Star Wars Ep. VII: The Force Awakens 935642689
2                      Avengers: Endgame 858373000
3                Spider-Man: No Way Home 804793477
4                      Top Gun: Maverick 718732821
5                          Black Panther 700059566
6               Avatar: The Way of Water 688459501
7                 Avengers: Infinity War 678815482
8                            Inside Out 2 652980194
9                         Jurassic World 652198010
10                         The Lion King 637188286
```

Which days of the week yielded the largest total gross?

```
movieData |>
   mutate(weekday=wday(date, label=TRUE)) |>
   summarize(gross=sum(gross), .by=weekday) |>
   arrange(desc(gross))
```

```
  weekday       gross
1     Sat 41425679200
2     Fri 35660713032
3     Sun 29077380480
4     Thu 13560888929
5     Tue 13073323005
6     Mon 12312783195
7     Wed 10891034005
```

5.1 Adjusting for Inflation

As you may have noticed, the price of a movie ticket keeps increasing (along with everything else!). The Bureau of Labor Statistics (BLS) tracks the Consumer Price Index (CPI) for "Admission to movies, theaters, and concerts" (series `CUUR0000SS62031`). The CPI is an index, not a dollar price. The base period (1998) equals 100, so a value of 240 means tickets cost 2.4× what they did in the late 1990s. What matters for us is the *ratio* between years, which gives us the inflation adjustment factor. That factor will vary by year. It will tell you how much you need to multiply, say, a ticket purchased in 2010 so that it equates to 2026 prices.

The BLS public API limits unauthenticated requests to 10 years of data per call, so we make two requests and combine them. The API returns monthly values (periods `M01`–`M12`). We will average those to get one index value per year.

```
library(httr2)

# max 10 years per call (if run without API key)
a <- request("https://api.bls.gov/publicAPI/v2/timeseries/data/") |>
   req_body_json(list(seriesid  = list("CUUR0000SS62031"),
                      startyear = 2010,
                      endyear   = 2019)) |>
   req_perform() |>
   resp_body_json() |>
   _$Results$series[[1]]$data

b <- request("https://api.bls.gov/publicAPI/v2/timeseries/data/") |>
   req_body_json(list(seriesid  = list("CUUR0000SS62031"),
                      startyear = 2020,
                      endyear   = 2026)) |>
   req_perform() |>
   resp_body_json() |>
   _$Results$series[[1]]$data
```

```r
inflation <- c(a,b)  |>
   lapply(function(x) data.frame(
      year   = as.integer(x$year),
      period = x$period,
      value  = ifelse(x$value=="-", # missing val
                      NA, as.numeric(x$value)))) |>
   bind_rows() |>
   # one month in 2025 missing due to gov't shutdown
   summarize(cpiMovie = mean(value, na.rm=TRUE), .by = year) |>
   arrange(desc(year)) |>
   mutate(adjustment = cpiMovie[1] / cpiMovie)
```

Now we link each movie to our inflation factor table to compute ticket sales adjusted to 2026 prices.

```r
movieData <- movieData |>
   mutate(year=year(date)) |>
   left_join(inflation, join_by(year==year)) |>
   mutate(grossAdj=gross*adjustment) |>
   select(-cpiMovie, -adjustment)

movieData |>
  summarize(grossAdj=sum(grossAdj), .by=movie) |>
  slice_max(grossAdj, n=10)
```

```
                                  movie   grossAdj
1  Star Wars Ep. VII: The Force Awakens 1341532683
2                      Avengers: Endgame 1126347753
3               Spider-Man: No Way Home  993167650
4                        Jurassic World  945146334
5                          The Avengers  943901751
6                         Black Panther  935307196
7                 Avengers: Infinity War  906924262
8                         The Lion King  857790414
9                      Top Gun: Maverick  850689676
10    Star Wars Ep. VIII: The Last Jedi  842779122
```

```r
# save again with inflation adjustment
save(movieData, file="movieData.RData", compress=TRUE)
```

The Avengers and *Star Wars: The Last Jedi* join the top 10 list and *Avatar* and *Inside Out 2* are out.

Now that you have movie data and, in a previous section, you assembled Chicago crime data, combine the two datasets so that you can answer the question "What happens to crime when big movies come out?"

6 Crime and the Movie Blockbuster

Start by loading our Chicago crime data. Since we just scraped movie data back to 2010, we can filter out earlier crime incidents.

```
load("dataChicagoCrime.RData")
```

Now let's merge our movie and crime data. The following code does not do what you think it does! Can you identify the problem?

```
crime |>
   filter(date(date) == "2026/07/04") |>
   mutate(date = date(date)) |>
   select(id, date) |>
   inner_join(movieData, by="date") |>
   head()
```

```
Warning in inner_join(select(mutate(filter(crime, date(date) == "2026/07/04"), : Detected an
to-many relationship between `x` and `y`.
i Row 1 of `x` matches multiple rows in `y`.
i Row 236325 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

```
        id       date              movie   gross year grossAdj
1 14253382 2026-07-04 Minions & Monsters 9491820 2026  9491820
2 14253382 2026-07-04   Young Washington 7672967 2026  7672967
3 14253382 2026-07-04        Toy Story 5 7537411 2026  7537411
4 14253382 2026-07-04          Supergirl 2506574 2026  2506574
5 14253382 2026-07-04     Disclosure Day 1858350 2026  1858350
6 14253382 2026-07-04          Obsession 1373315 2026  1373315
```

Every crime incident on July 4, 2026 is matched to every movie that was in theaters on that day. If we total the gross and crime counts for this one day we get

```
crime |>
   filter(date(date) == "2026/07/04") |>
   mutate(date = date(date)) |>
   select(id, date) |>
   inner_join(movieData, by="date") |>
   summarize(crimecount = n(),
             moviegross = sum(grossAdj))
```

```
Warning in inner_join(select(mutate(filter(crime, date(date) == "2026/07/04"), : Detected an
to-many relationship between `x` and `y`.
i Row 1 of `x` matches multiple rows in `y`.
i Row 236325 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

```
  crimecount  moviegross
1      20398 21944236832
```

I can assure you that there were no 20,000 crimes in Chicago on one day and there were not $200B in ticket sales either. The US economy is roughly $32T… If movie ticket sales were $200B/day that would be more than twice the entire US economy. Always ask yourself if the results make sense.

What we really want is to aggregate crime counts by date, aggregate movie gross by date, and then merge the two datasets. Let's focus just on Friday and Saturday nights.

```
dCrimeMovie <- crime |>
   filter(wday(date, label=TRUE) %in% c("Fri","Sat") &
             hour(date) >= 18) |>
   mutate(date = date(date)) |>
   count(date, name = "crimeCount") |>
   inner_join(movieData |>
                 summarize(movieGross = sum(grossAdj) / 10^6,
                           .by = "date"),
              by="date")

head(dCrimeMovie)
```

```
        date crimeCount movieGross
1 2010-01-01        231  130.38787
2 2010-01-02        265  136.01848
3 2010-01-08        294   71.69270
4 2010-01-09        267  106.53668
5 2010-01-15        381   75.03696
6 2010-01-16        298  105.22533
```

Here is a first test to see the relationship between movie gross and crime counts.

```
plot(crimeCount~movieGross, data=dCrimeMovie)

# which points are in 2020 & 2021?
points(crimeCount~movieGross,
       data = dCrimeMovie |>
         filter(date >= "2020-03-15" &
                date <= "2021-05-01"),
       col="red")
```

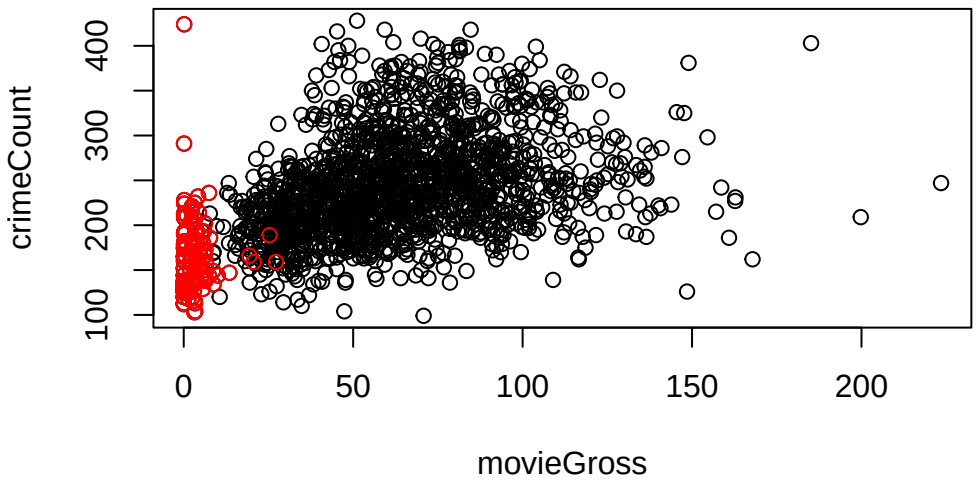

Figure 1: Daily movie ticket revenue and reported crime on Friday and Saturday evenings. Red points indicate dates during the COVID-19 pandemic period.

While this is NOT an inference course, here's how to fit linear models in R

```r
lm1 <- lm(crimeCount~movieGross, data=dCrimeMovie)
summary(lm1)
```

```
Call:
lm(formula = crimeCount ~ movieGross, data = dCrimeMovie)

Residuals:
    Min      1Q  Median      3Q     Max
-183.83  -35.69   -5.82   27.74  231.66

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 192.23766    2.77658   69.23   <2e-16 ***
movieGross    0.79173    0.04102   19.30   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 53.35 on 1726 degrees of freedom
Multiple R-squared:  0.1775,    Adjusted R-squared:  0.177
F-statistic: 372.5 on 1 and 1726 DF,  p-value: < 2.2e-16
```

The `Estimate` associated with `movieGross` is an estimate of the slope drawn through the middle of the cloud of points. It suggests that every additional million dollars in ticket sales is associated with an increase in 0.79 crimes.

The linear model fit using `lm()` assumes that an increase from \$10M to \$11M in ticket sales has the same effect on crime as an increase from \$100M to \$101M. This is probably not a reasonable assumption. Perhaps a more plausible assumption is multiplicative instead of additive… a \$1M increase in ticket sales is associated with the same percentage change in crime, regardless of whether ticket sales increase from \$10M to \$11M or from \$100M to \$101M. To model this, we can use a generalized linear model (GLM) with a log link function.

```r
glm1 <- glm(crimeCount~movieGross,
            family = quasipoisson,
            data = dCrimeMovie)
summary(glm1)
```

```
Call:
glm(formula = crimeCount ~ movieGross, family = quasipoisson,
    data = dCrimeMovie)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 5.2802807  0.0119126  443.25   <2e-16 ***
movieGross  0.0032348  0.0001681   19.24   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 11.95284)

    Null deviance: 24585  on 1727  degrees of freedom
Residual deviance: 20222  on 1726  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4
```

The coefficients from this model are on the log scale. To convert to a percentage change, we exponentiate the coefficient and subtract 1. This suggests that a \$1M increase in ticket sales is associated with a $100(\exp(\beta_1) - 1) = 0.3\%$ increase in crime.

Perhaps the COVID pandemic period is really throwing off this analysis.

```
lm1 <- dCrimeMovie |>
   filter(!between(date, ymd("2020-03-15"), ymd("2021-05-01"))) |>
   lm(crimeCount~movieGross,
          data = _)
summary(lm1)
```

```
Call:
lm(formula = crimeCount ~ movieGross, data = filter(dCrimeMovie,
    !between(date, ymd("2020-03-15"), ymd("2021-05-01"))))

Residuals:
     Min       1Q   Median       3Q      Max
-173.332  -35.982   -6.081   28.459  191.377

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 203.68958    3.30725   61.59   <2e-16 ***
movieGross    0.64396    0.04718   13.65   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 53.44 on 1608 degrees of freedom
Multiple R-squared:  0.1038,    Adjusted R-squared:  0.1033
F-statistic: 186.3 on 1 and 1608 DF,  p-value: < 2.2e-16
```

```
glm1 <- dCrimeMovie |>
   filter(!between(date, ymd("2020-03-15"), ymd("2021-05-01"))) |>
   glm(crimeCount~movieGross,
       family = quasipoisson,
       data = _)
summary(glm1)
```

```
Call:
glm(formula = crimeCount ~ movieGross, family = quasipoisson,
    data = filter(dCrimeMovie, !between(date, ymd("2020-03-15"),
        ymd("2021-05-01"))))

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 5.3344597  0.0135525  393.61   <2e-16 ***
movieGross  0.0025587  0.0001874   13.66   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 11.64889)

    Null deviance: 20514  on 1609  degrees of freedom
Residual deviance: 18380  on 1608  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4
```

The effect is still positive (more movie tickets is associated with more reported crime), but the effect is slightly smaller after dropping pandemic dates.

Here's the model fit.

```
plot(crimeCount~movieGross, data=dCrimeMovie)

# which points are in 2020 & 2021?
points(crimeCount~movieGross,
       data = dCrimeMovie |>
         filter(date >= "2020-03-15" &
```

```
                date <= "2021-05-01"),
       col="red")

x <- seq(min(dCrimeMovie$movieGross),
         max(dCrimeMovie$movieGross),
         length=100)
y <- predict(glm1,
             newdata=data.frame(movieGross=x),
             type="response")
lines(y~x)
```

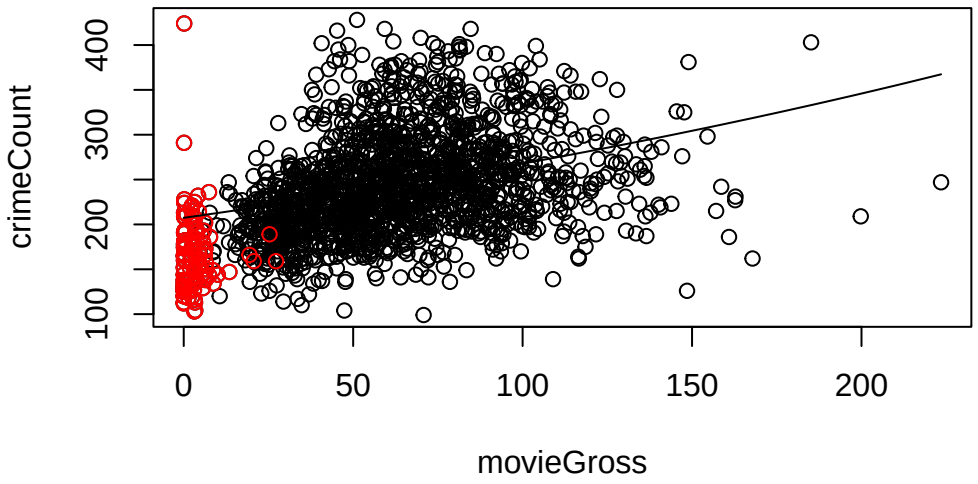

Figure 2: Daily movie ticket revenue and reported crime with the fitted quasi-Poisson model after excluding the COVID-19 pandemic period. Red points indicate dates during the pandemic period.

Exercise

3. Compare the relationship between crime and revenue for weekdays

6.1 What about really big movie days?

The previous analysis risks a lot of confounding. For example, a hot summer day might increase both movie ticket sales and crime. To reduce confounding, we can look at the change in movie ticket sales from one week to the next and see if that is associated with a change in crime. In particularly, I am interested in those days where the big blockbuster comes out while one week earlier ticket sales were low. Let's conduct a matched analysis where each day is matched to itself one week earlier.

```
# add crime counts and movie gross from one week ago
dCrimeMovie <- dCrimeMovie |>
   mutate(lastweekdate = date - weeks(1)) |>
   left_join(dCrimeMovie |>
               select(date, movieGross, crimeCount),
             join_by(lastweekdate == date)) |>
   rename(crimeCount1 = crimeCount.x,
          crimeCount0 = crimeCount.y,
          movieGross1  = movieGross.x,
          movieGross0  = movieGross.y) |>
   filter(!is.na(movieGross0) & !is.na(crimeCount0))

head(dCrimeMovie)
```

```
        date crimeCount1 movieGross1 lastweekdate movieGross0 crimeCount0
1 2010-01-08         294    71.69270   2010-01-01   130.38787         231
2 2010-01-09         267   106.53668   2010-01-02   136.01848         265
3 2010-01-15         381    75.03696   2010-01-08    71.69270         294
4 2010-01-16         298   105.22533   2010-01-09   106.53668         267
5 2010-01-22         354    63.86189   2010-01-15    75.03696         381
6 2010-01-23         332   102.58412   2010-01-16   105.22533         298
```

We are looking for days where movie sales are much bigger than 1 week earlier.

```
boxplot(dCrimeMovie$movieGross1,
        ylab="Daily gross (millions of $)")
```

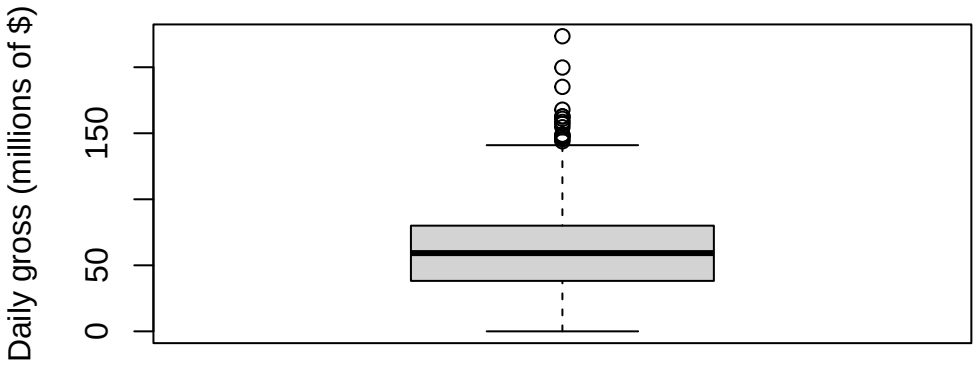

Figure 3: Distribution of daily movie ticket revenue on Friday and Saturday evenings.

```
dCrimeMovie <- dCrimeMovie |>
   mutate(pctgain = 100*(movieGross1-movieGross0)/movieGross0)

boxplot(dCrimeMovie$pctgain, ylab="% gross increase over 7 days")
```

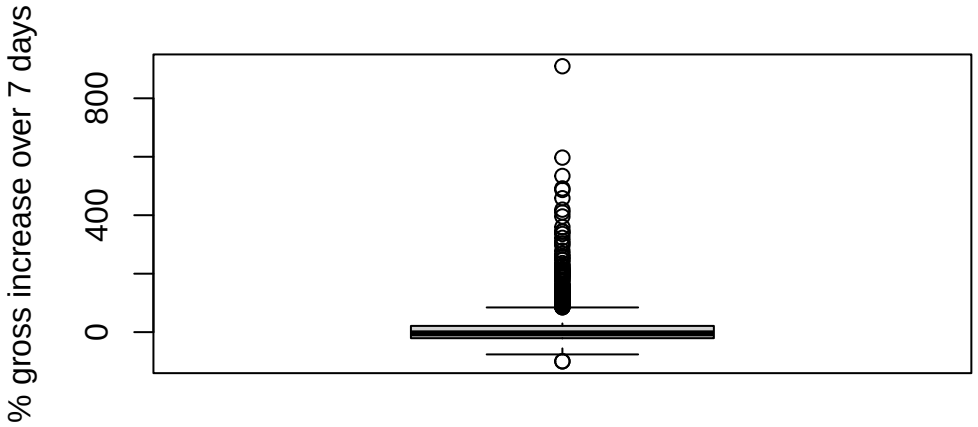

Figure 4: Percentage change in daily movie ticket revenue relative to one week earlier.

There are a number of days where the gross more than doubled from one week earlier. Let's study crime changes on those days where the ticket sales at least double.

```
jumpMovies <- dCrimeMovie |>
  filter(pctgain>=100 &
           (date < "2020-03-15" | date > "2021-05-01"))

head(jumpMovies)
```

```
        date crimeCount1 movieGross1 lastweekdate movieGross0 crimeCount0
1 2010-11-19         350   127.87569   2010-11-12    61.19357         377
2 2011-07-15         403   185.11629   2011-07-08    78.89244         368
3 2012-03-23         299   127.86388   2012-03-16    53.86985         318
4 2012-05-04         326   145.47398   2012-04-27    47.45511         331
5 2012-12-14         302    76.61783   2012-12-07    33.63190         262
6 2013-05-03         273   122.22456   2013-04-26    41.07012         318
   pctgain
1 108.9691
2 134.6439
3 137.3570
```

```
4 206.5507
5 127.8130
6 197.5997
```

Is there a hint that crime counts are different on big movie days?

```
hist(jumpMovies$crimeCount1/jumpMovies$crimeCount0,
     breaks=15,
     main = "", xlab="Ratio of crime counts (this week / last week)")
```

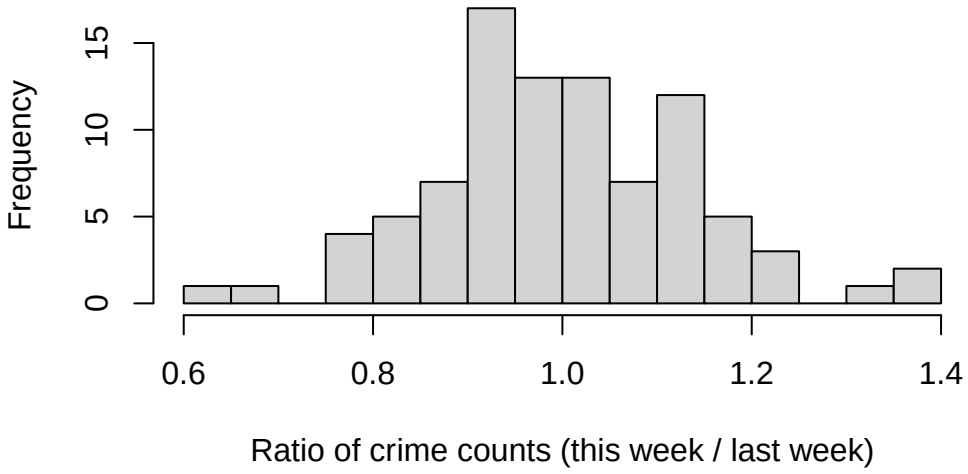

Figure 5: Ratio of reported crime counts on big movie days to counts one week earlier.

The histogram seems centered around 1, but there is some variation. To get the data ready for analysis, we need to pivot the data longer so that every date has two rows, one for the current week and one for the previous week. We will also create a variable `bigmovie` that is 1 if the row is for the current week and 0 if it is for the previous week. Finally, we will convert `date` to a factor so that it can be used as a blocking variable in our analysis.

```
a <- jumpMovies |>
  select(date, crimeCount0, crimeCount1) |>
  pivot_longer(cols=c(crimeCount0, crimeCount1),
               values_to = "crimeCount",
```

```
               names_to = "bigmovie") |>
  mutate(bigmovie = as.numeric(bigmovie=="crimeCount1"),
         date     = factor(as.character(date)))

head(a)
```

```
# A tibble: 6 x 3
  date       bigmovie crimeCount
  <fct>         <dbl>      <int>
1 2010-11-19        0        377
2 2010-11-19        1        350
3 2011-07-15        0        368
4 2011-07-15        1        403
5 2012-03-23        0        318
6 2012-03-23        1        299
```

Since we have matched data, we can use a conditional Poisson model to estimate the effect of big movie days on crime counts. In this case, we will match on the `date` variable to account for the matched pairs.

```
# conditional poisson model for matched data
library(gnm)
```

```
Warning: package 'gnm' was built under R version 4.6.1
```

```
cpois1 <- gnm(crimeCount~bigmovie, data=a,
              family    = quasipoisson,
              eliminate = date)
summary(cpois1)
```

```
Call:

gnm(formula = crimeCount ~ bigmovie, eliminate = date, family = quasipoisson,
    data = a)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.896e+00  -6.448e-01  -1.028e-05   6.340e-01   2.678e+00

Coefficients of interest:
```

```
          Estimate Std. Error t value Pr(>|t|)
bigmovie -0.008282   0.013650  -0.607    0.546

(Dispersion parameter for quasipoisson family taken to be 1.912126)

Residual deviance: 172.55 on 90 degrees of freedom
AIC: NA

Number of iterations: 2
```

Looks like there is no evidence that big movies affect crime. A 95% confidence interval for the percent change is (-3.4, 1.9)%. The confidence interval includes 0.

Here's everything we did in a compact block of code.

```
# Same analysis with two chains of |>
#   compact, but harder to check and debug
dCrimeMovie <- crime |>
   filter(wday(date, label=TRUE) %in% c("Fri","Sat") &
             hour(date) >= 18 &
             !between(date, ymd("2020-03-15"), ymd("2021-05-01"))) |>
   mutate(date = date(date)) |>
   count(date, name = "crimeCount") |>
   inner_join(movieData |>
                summarize(movieGross = sum(grossAdj) / 10^6,
                          .by = "date"),
              by="date")

dCrimeMovie |>
   mutate(lastweekdate = date - weeks(1)) |>
   left_join(dCrimeMovie |>
               select(date, movieGross, crimeCount),
             join_by(lastweekdate == date)) |>
   rename(crimeCount1 = crimeCount.x,
          crimeCount0 = crimeCount.y,
          movieGross1  = movieGross.x,
          movieGross0  = movieGross.y) |>
   filter(!is.na(movieGross0) & !is.na(crimeCount0)) |>
   mutate(pctgain = 100*(movieGross1-movieGross0)/movieGross0) |>
   filter(pctgain>=100) |>
   select(date, crimeCount0, crimeCount1) |>
   pivot_longer(cols=c(crimeCount0, crimeCount1),
                values_to = "crimeCount",
```

```
                names_to = "bigmovie") |>
  mutate(bigmovie = as.numeric(bigmovie=="crimeCount1"),
         date     = factor(as.character(date))) |>
  gnm(crimeCount~bigmovie, data=_,
              family    = quasipoisson,
              eliminate = date) |>
  summary()
```

```
Call:

gnm(formula = crimeCount ~ bigmovie, eliminate = date, family = quasipoisson,
    data = mutate(pivot_longer(select(filter(mutate(filter(rename(left_join(mutate(dCrimeMovi
        lastweekdate = date - weeks(1)), select(dCrimeMovie,
        date, movieGross, crimeCount), join_by(lastweekdate ==
        date)), crimeCount1 = crimeCount.x, crimeCount0 = crimeCount.y,
        movieGross1 = movieGross.x, movieGross0 = movieGross.y),
        !is.na(movieGross0) & !is.na(crimeCount0)), pctgain = 100 *
        (movieGross1 - movieGross0)/movieGross0), pctgain >=
        100), date, crimeCount0, crimeCount1), cols = c(crimeCount0,
        crimeCount1), values_to = "crimeCount", names_to = "bigmovie"),
        bigmovie = as.numeric(bigmovie == "crimeCount1"), date = factor(as.character(date))))


Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.896e+00  -6.448e-01  -1.028e-05   6.340e-01   2.678e+00

Coefficients of interest:
          Estimate Std. Error t value Pr(>|t|)
bigmovie -0.008282   0.013650  -0.607    0.546

(Dispersion parameter for quasipoisson family taken to be 1.912126)

Residual deviance: 172.55 on 90 degrees of freedom
AIC: NA

Number of iterations: 2
```

7 Solutions to the exercises

1. What movie that was in theaters one week ago had the longest Days in Release?

```
a <- scan(paste0("https://www.the-numbers.com/box-office-chart/daily/",
                 gsub("-","/",today() - weeks(1))),
          what="",sep="\n")
a <- a[1:grep("chart-mobile",a)]
i <- grep('href="/movie/', a)
data.frame(movie=gsub("<[^>]*>", "", a[i]),
           days=gsub("<[^>]*>|,", "", a[i+7]) |>
              as.numeric()) |>
   slice_max(days)
```

```
                    movie  days
1 Manhunter: The Final Cut 14591
```

2. What were the top grossing movies on July 4th for 2010 to the present? Run in parallel.

```
plan(multisession, workers = 8)
registerDoFuture()

foreach(iDate = 1:17,
        .combine = bind_rows) %dopar%
{
   urlText <- paste0("https://www.the-numbers.com/box-office-chart/daily/",
                     gsub("-", "/",
                          ymd("2009/07/04") + years(iDate)))
   a <- scan(urlText, what = "", sep = "\n",
             fileEncoding = "UTF-8", quiet = TRUE)

   i <- grep('href="/movie/', a)

   data0 <- data.frame(movie = gsub("<[^>]*>", "", a[i]),
                       gross = as.numeric(gsub("<[^>]*>|[$,]", "", a[i+1])),
                       year  = 2009 + iDate)

   # replace empty movie names
   j <- which(data0$movie=="")
   if(length(j) > 0)
   {
      data0$movie[j] <- gsub('.*/movie/([^"]*)".*', "\\1", a[i[j]]) |>
         gsub("-", " ", x=_)
   }

   data0 <- data0 |>
```

```
    slice_max(gross) |>
    distinct()

  return(data0)
}
```

```
                            movie    gross year
1       The Twilight Saga: Eclipse 13258246 2010
2   Transformers: Dark of the Moon 18033185 2011
3           Amazing Spider Man The 23335925 2012
4                  Despicable Me 2 24546980 2013
5  Transformers: Age of Extinction 10619669 2014
6                    Jurassic World  8539045 2015
7                      Finding Dory  9619207 2016
8                  Despicable Me 3 12728355 2017
9   Jurassic World: Fallen Kingdom 11501395 2018
10       Spider-Man: Far From Home 25720128 2019
11                           Relic    48259 2020
12               F9: The Fast Saga  5278700 2021
13        Minions: The Rise of Gru 16077100 2022
14                Sound of Freedom 14242063 2023
15                 Despicable Me 4 20398275 2024
16          Jurassic World Rebirth 26235450 2025
17          Minions &amp;amp; Monsters  9491820 2026
```