**Introduction**

Are you trying to compare price of products across websites? Are you trying to  
monitor price changes every hour? Or planning to do some text mining or  
sentiment analysis on reviews of products or services? If yes, how would you do  
that? How do you get the details available on the website into a format in  
which you can analyse it?

* Can you copy/paste the data from their website?
* Can you see some save button?
* Can you download the data?

Hmmm.. If you have these or similar questions on your mind, you have come to  
the right place. In this post, we will learn about web scraping using R. Below  
is a video tutorial which covers the intial part of this post.

The slides used in the above video tutorial can be found  
[here](https://slides.rsquaredacademy.com/web-scraping/web-scraping.html#/section).

**The What?**



What exactly is web scraping or web mining or web harvesting? It is a  
technique for extracting data from websites. Remember, websites contain wealth  
of useful data but designed for human consumption and not data analysis. The  
goal of web scraping is to take advantage of the pattern or structure of web  
pages to extract and store data in a format suitable for data analysis.

**The Why?**



Now, let us understand why we may have to scrape data from the web.

* **Data Format**: As we said earlier, there is a wealth of data on websites  
  but designed for human consumption. As such, we cannot use it for data analysis  
  as it is not in a suitable format/shape/structure.
* **No copy/paste**: We cannot copy & paste the data into a local file. Even if  
  we do it, it will not be in the required format for data analysis.
* **No save/download**: There are no options to save/download the required data  
  from the websites. We cannot right click and save or click on a download button  
  to extract the required data.
* **Automation**: With web scraping, we can automate the process of data  
  extraction/harvesting.

**The How?**



* **robots.txt**: One of the most important and overlooked step is to check the  
  **robots.txt** file to ensure that we have the permission to access the web  
  page without violating any terms or conditions. In R, we can do this using the  
  [robotstxt](https://cran.r-project.org/package=robotstxt)  
  by [rOpenSci](https://ropensci.org/).
* **Fetch**: The next step is to fetch the web page using the  
  [xml2](https://cran.r-project.org/package=xml2)  
  package and store it so that we can extract the required data. Remember, you  
  fetch the page once and store it to avoid fetching multiple times as it may  
  lead to your IP address being blocked by the owners of the website.
* **Extract/Store/Analyze**: Now that we have fetched the web page, we will use  
  [rvest](https://rvest.tidyverse.org/) to extract the  
  data and store it for further analysis.

**Use Cases**



Below are few use cases of web scraping:

* **Contact Scraping**: Locate contact information including email addresses,  
  phone numbers etc.
* **Monitoring/Comparing Prices**: How your competitors price their products,  
  how your prices fit within your industry, and whether there are any  
  fluctuations that you can take advantage of.
* **Scraping Reviews/Ratings**: Scrape reviews of product/services and use it  
  for text mining/sentiment analysis etc.

**Things to keep in mind…**



* **Static & Well Structured**: Web scraping is best suited for static & well  
  structured web pages. In one of our case studies, we demonstrate how badly  
  structured web pages can hamper data extraction.
* **Code Changes**: The underling HTML code of a web page can change anytime  
  due to changes in design or for updating details. In such case, your script  
  will stop working. It is important to identify changes to the web page and  
  modify the web scraping script accordingly.
* **API Availability**: In many cases, an API (application programming interface)  
  is made available by the service provider or organization. It is always  
  advisable to use the API and avoid web scraping. The  
  [httr](https://httr.r-lib.org/) package has a  
  nice introduction on interacting with APIs.
* **IP Blocking**: Do not flood websites with requests as you run the risk of  
  getting blocked. Have some time gap between request so that your IP address in  
  not blocked from accessing the website.
* **robots.txt**: We cannot emphasize this enough, always review the  
  **robots.txt** file to ensure you are not violating any terms and conditions.

**Case Studies**



* **Best selling mobile phones**: In the first case study, we will scrape the  
  best selling mobile phones from website of Amazon India and extract the name,  
  color, rating, number of reviews, real and discounted price.
* **IMDB top 50 movies**: In this case study we will scrape the IMDB website  
  to extract the title, year of release, certificate, runtime, genre, rating,  
  votes and revenue of the top 50 movies.
* **Most visited websites**: In this case study, we will look at the 50 most  
  visited websites in the world including the category to which they belong,  
  average time on site, average pages browsed per vist and bounce rate.
* **List of RBI governors** : In this final case study, we will scrape the list  
  of RBI Governors from Wikipedia, and analyze the background from which they  
  came i.e whether there were more economists or bureaucrats?

[course ad](https://www.rsquaredacademy.com/)

**HTML Basics**

To be able to scrape data from websites, we need to understand how the web  
pages are structured. In this section, we will learn just enough HTML to be  
able to start scraping data from websites.

**HTML, CSS & JAVASCRIPT**



A web page typically is made up of the following:

* **HTML** (Hyper Text Markup Language) takes care of the content. You need to  
  have a basic knowledge of HTML tags as the content is located with these tags.
* **CSS** (Cascading Style Sheets) takes care of the appearance of the content.  
  While you don’t need to look into the CSS of a web page, you should be able to  
  identify the **id** or **class** that manage the appearance of content.
* **JS** (Javascript) takes care of the behavior of the web page.

**HTML Element**



HTML element consists of a start tag and end tag with content inserted in  
between. They can be nested and are case insensitive. The tags can have  
attributes as shown in the above image. The attributes usually come as  
name/value pairs. In the above image, **class** is the attribute name while  
**primary** is the attribute value. While scraping data from websites in the  
case study, we will use a combination of HTML tags and attributes to locate  
the content we want to extract. Below is a list of basic and important HTML  
tags you should know before you get started with web scraping.



**DOM**



**DOM** (Document Object Model) defines the logical structure of a document  
and the way it is accessed and manipulated. In the above image, you can see  
that HTML is structured as a tree and you trace path to any node or tag. We  
will use a similar approach in our case studies. We will try to trace the  
content we intend to extract using HTML tags and attributes. If the web page  
is well structured, we should be able to locate the content using a unique  
combination of tags and attributes.

**HTML Attributes**



* all HTML elements can have attributes
* they provide additional information about an element
* they are always specified in the start tag
* usually come in name/value pairs

The class attribute is used to define equal styles for elements with same  
class name. HTML elements with same class name will have the same format and  
style. The id attribute specifies a unique id for an HTML element. It can be  
used on any HTML element and is case sensitive. The style attribute sets the  
style of an HTML element.

[youtube ad](https://www.youtube.com/user/rsquaredin/)

**Libraries**

We will use the following R packages in this tutorial.

library(robotstxt)

library(rvest)

library(selectr)

library(xml2)

library(dplyr)

library(stringr)

library(forcats)

library(magrittr)

library(tidyr)

library(ggplot2)

library(lubridate)

library(tibble)

library(purrr)

**Best Selling Mobile Phones**



In this first case study, we will scrape the details of best selling smart  
phones from [Amazon](https://www.amazon.in/mobile-phones/b?ie=UTF8&node=1389401031&ref_=nav_shopall_sbc_mobcomp_all_mobiles).  
Our goal is to extract the following:

* Brand name
* Color
* Rating
* Number of reviews
* Real price
* Actual price (after discount)

**robotstxt**

As mentioned earlier, we will first check if we can scrape data from the web  
page using paths\_allowed() from the **robotstxt** package. We need to  
specify the url of the web page using the paths argument. If we can access  
the web page, paths\_allowed() will return TRUE, else FALSE.

paths\_allowed(

paths = c("https://www.amazon.in/mobile-phones/b?ie=UTF8&node=1389401031&ref\_=nav\_shopall\_sbc\_mobcomp\_all\_mobiles")

)

##

www.amazon.in No encoding supplied: defaulting to UTF-8.

## [1] TRUE

Since it has returned TRUE, let us go ahead and download the web page using  
read\_html() from the **xml2** package and store it in top\_phones. We do  
this to ensure not to make repeated requests to the website which may lead to  
our IP address being blocked.

top\_phones <- read\_html("https://www.amazon.in/mobile-phones/b?ie=UTF8&node=1389401031&ref\_=nav\_shopall\_sbc\_mobcomp\_all\_mobiles")

top\_phones

## {xml\_document}

##

## [1] \n

**Brand Name**



The first detail we want to extract is the brand name of the phone. If you look

at the HTML code, it is nested within a hyperlink, defined by . The link

is inside a section identified by the class crwTitle. We will use this

information to ask **rvest** to extract text content which will give us the

brand name.

The location is specified using html\_nodes() and the text extracted using

hmtl\_text(). Since crwTitle is a class, we use . before it but not for

a as it is a HTML tag. Both the class and the tag are specified within quotes

and separated by space.

top\_phones %>%

html\_nodes(".crwTitle a") %>%

html\_text()

## [1] "Samsung Galaxy M30 (Gradation Blue, 4+64 GB)"

## [2] "Samsung Galaxy M20 (Ocean Blue, 4+64GB)"

## [3] "Samsung Galaxy M30 (Gradation Black, 4+64 GB)"

## [4] "Redmi 6A (Black, 2GB RAM, 16GB Storage)"

## [5] "Redmi 6 Pro (Black, 3GB RAM, 32GB Storage)"

## [6] "Samsung Galaxy M10 (Ocean Blue, 3+32GB)"

## [7] "Redmi 6 Pro (Black, 4GB RAM, 64GB Storage)"

## [8] "Realme U1 (Ambitious Black, 3GB RAM, 32GB Storage)"

## [9] "Honor 8C (Blue, 4GB RAM, 32GB Storage)"

## [10] "OnePlus 6T (Mirror Black, 6GB RAM, 128GB Storage)"

If you observe the output, it includes the following:

* brand name
* color
* RAM
* storage capacity

To extract the brand name, we will do the following:

* use str\_split() from **stringr** and specify the pattern \\( i.e split
* the string @ the first opening bracket. Since ( is a special character, we
* use \\ for escaping.
* next, we use map\_chr() from the **purrr** package to extract the first
* element from the resulting list.
* finally, we remove the white space using str\_trim().

The whole point of the above exercise is to show that extracting the data

using **rvest** is just one part of web scraping. We need to spend enough

time tidying and reshaping the data to get it into a format useful for data

analysis.

top\_phones %>%

html\_nodes(".crwTitle a") %>%

html\_text() %>%

str\_split('\\(') %>%

map\_chr(1) %>%

str\_trim() -> mobile\_name

mobile\_name

## [1] "Samsung Galaxy M30" "Samsung Galaxy M20" "Samsung Galaxy M30"

## [4] "Redmi 6A" "Redmi 6 Pro" "Samsung Galaxy M10"

## [7] "Redmi 6 Pro" "Realme U1" "Honor 8C"

## [10] "OnePlus 6T"

**Color**



In the previous step, we observed that the data extracted from top\_mobiles  
included the color of the mobile as well. So the location of the color in  
the HTML is same; within the hyperlink of the crwTitle section. But now,  
we want to extract the color and not the brand name.

top\_phones %>%

html\_nodes(".crwTitle a") %>%

html\_text()

## [1] "Samsung Galaxy M30 (Gradation Blue, 4+64 GB)"

## [2] "Samsung Galaxy M20 (Ocean Blue, 4+64GB)"

## [3] "Samsung Galaxy M30 (Gradation Black, 4+64 GB)"

## [4] "Redmi 6A (Black, 2GB RAM, 16GB Storage)"

## [5] "Redmi 6 Pro (Black, 3GB RAM, 32GB Storage)"

## [6] "Samsung Galaxy M10 (Ocean Blue, 3+32GB)"

## [7] "Redmi 6 Pro (Black, 4GB RAM, 64GB Storage)"

## [8] "Realme U1 (Ambitious Black, 3GB RAM, 32GB Storage)"

## [9] "Honor 8C (Blue, 4GB RAM, 32GB Storage)"

## [10] "OnePlus 6T (Mirror Black, 6GB RAM, 128GB Storage)"

We will split the original string @ ( and extract the second part which  
includes:

* color
* RAM
* storage capacity

The color is separated from the rest by a comma. We will use the , to split  
the string and extract the color using map\_chr() i.e. extract the first  
element from the resulting list.

top\_phones %>%

html\_nodes(".crwTitle a") %>%

html\_text() %>%

str\_split('\\(') %>%

map\_chr(2) %>%

str\_split(",") %>%

map\_chr(1) -> mobile\_color

mobile\_color

## [1] "Gradation Blue" "Ocean Blue" "Gradation Black"

## [4] "Black" "Black" "Ocean Blue"

## [7] "Black" "Ambitious Black" "Blue"

## [10] "Mirror Black"

**Rating**



Let us extract the ratings for the phones now. If you look at the HTML code,  
we can locate rating within the following:

* .crwProductDetail

It is wrapped within identified by the class .a-icon-alt which is  
inside a hyperlink in the section identified by the class .crwProductDetail.

top\_phones %>%

html\_nodes(".crwProductDetail span .a-icon-alt") %>%

html\_text()

## [1] "4.0 out of 5 stars" "3.7 out of 5 stars" "4.1 out of 5 stars"

## [4] "4.2 out of 5 stars" "4.2 out of 5 stars" "4.1 out of 5 stars"

## [7] "4.1 out of 5 stars" "4.4 out of 5 stars" "4.2 out of 5 stars"

## [10] "4.5 out of 5 stars"

In the outptut, you can observe the text **out of 5 stars** for each rating.  
Let us get rid of this text by selecting the first 3 characters using  
str\_sub(). We pick the first 3 characters using the start and end  
arguments and supply them the values 1 and 3. Finally, we convert the  
rating to a number using as.numeric().

top\_phones %>%

html\_nodes(".crwProductDetail span .a-icon-alt") %>%

html\_text() %>%

str\_sub(start = 1, end = 3) %>%

as.numeric() -> mobile\_rating

mobile\_rating

## [1] 4.0 3.7 4.1 4.2 4.2 4.1 4.1 4.4 4.2 4.5

**Number of Reviews**



Now that we know the rating for each of the top 10 best selling smart phones,  
let us find out how many people have reviewed them. The number of reviews is  
located within the following:

* hyperlink identified by the class .a-link-normal
* tag identified by the class .a-size-small
* section identified by the class .crwProductDetail

top\_phones %>%

html\_nodes(".crwProductDetail span.a-size-small .a-link-normal") %>%

html\_text()

## [1] "1,100" "12,060" "553" "23,487" "6,403" "3,013" "12,366"

## [8] "10,413" "822" "14,379"

We use the above information within html\_nodes() to extract the data. Now  
let us clean it up a bit and convert it into a number instead of leaving it  
as a character. If you use as.numeric() directly, you will see NA in  
the result, the reason being the presence of comma in the number of reviews.  
First, we need to get rid of the comma, which we will do using str\_replace().  
We replace the comma with nothing as shown in the code below and then convert  
it into a number.

top\_phones %>%

html\_nodes(".crwProductDetail span.a-size-small .a-link-normal") %>%

html\_text() %>%

str\_replace(",", "") %>%

as.numeric() -> mobile\_review

mobile\_review

## [1] 1100 12060 553 23487 6403 3013 12366 10413 822 14379

**Real Price**



The price is one of the most important factor when it comes to choosing a  
smart phone. Let us look at the price of the best selling mobile phones. Again,  
looking at the HTML code, the price can be located within the following:

* tag identified by the class .a-text-strike
* section identified by the class crwPrice and .crwProductDetail

top\_phones %>%

html\_nodes(".crwProductDetail .crwPrice .a-text-strike") %>%

html\_text()

## [1] "  Rs. 16,490.00" "  Rs. 13,390.00" "  Rs. 16,490.00"

## [4] "  Rs. 6,999.00" "  Rs. 11,499.00" "  Rs. 9,290.00"

## [7] "  Rs. 13,499.00" "  Rs. 12,999.00" "  Rs. 12,999.00"

## [10] "  Rs. 37,999.00"

Using the above information, we can extract the price of the mobile phones  
which is returned as a character vector but we need to convert it to numeric  
if we are to analyze it further. Let use convert the price to a number using  
the following steps:

* use str\_trim() to remove the white spaces
* exclude the currency information using str\_sub()
* replace the comma using str\_replace()
* remove the decimal values using str\_split()
* extract the price from the resulting list using map\_chr()
* convert the price to a number using as.numeric()

top\_phones %>%

html\_nodes(".crwProductDetail .crwPrice .a-text-strike") %>%

html\_text() %>%

str\_trim() %>%

str\_sub(start = 5) %>%

str\_replace(",", "") %>%

str\_split("\\.") %>%

map\_chr(1) %>%

as.numeric() -> real\_price

real\_price

## [1] 16490 13390 16490 6999 11499 9290 13499 12999 12999 37999

**Actual Price**



Deep discounts are one of the strategies adopted by ecommerce firms to drive  
sales. Let us look at the actual price (after discount price) of the best  
selling mobile phones. The discounted price can be located within the  
following:

* tag identified by the class .crwActualPrice
* section identified by the class crwPrice and .crwProductDetail

Using the above information, we can extract the discounted price of the mobile  
phones.

top\_phones %>%

html\_nodes(".crwProductDetail .crwActualPrice") %>%

html\_text()

## [1] "  Rs. 14,990.00" "  Rs. 12,990.00" "  Rs. 14,990.00"

## [4] "  Rs. 5,999.00" "  Rs. 7,999.00" "  Rs. 8,990.00"

## [7] "  Rs. 10,000.00" "  Rs. 9,999.00" "  Rs. 10,999.00"

## [10] "  Rs. 34,999.00"

Let use convert the price to a number using the same steps as in the  
case of real price.

top\_phones %>%

html\_nodes(".crwProductDetail .crwActualPrice") %>%

html\_text() %>%

str\_trim() %>%

str\_sub(start = 5) %>%

str\_replace(",", "") %>%

str\_split("\\.") %>%

map\_chr(1) %>%

as.numeric() -> discounted\_price

discounted\_price

## [1] 14990 12990 14990 5999 7999 8990 10000 9999 10999 34999

**Putting it all together…**

best\_sellers <- tibble(title = mobile\_name, color = mobile\_color,

rating = mobile\_rating, reviews = mobile\_review, `Real Price (Rs.)` = real\_price,

`Discount Price (Rs.)` = discounted\_price)

best\_sellers

## # A tibble: 10 x 6

## title color rating reviews `Real Price (Rs~ `Discount Price (~

##

## 1 Samsung Ga~ Gradatio~ 4 1100 16490 14990

## 2 Samsung Ga~ Ocean Bl~ 3.7 12060 13390 12990

## 3 Samsung Ga~ Gradatio~ 4.1 553 16490 14990

## 4 Redmi 6A Black 4.2 23487 6999 5999

## 5 Redmi 6 Pro Black 4.2 6403 11499 7999

## 6 Samsung Ga~ Ocean Bl~ 4.1 3013 9290 8990

## 7 Redmi 6 Pro Black 4.1 12366 13499 10000

## 8 Realme U1 Ambitiou~ 4.4 10413 12999 9999

## 9 Honor 8C Blue 4.2 822 12999 10999

## 10 OnePlus 6T Mirror B~ 4.5 14379 37999 34999

[apps ad](https://apps.rsquaredacademy.com/)

**IMDB Top 50**



In this case study, we will extract the following details of the top 50 movies  
from the IMDB website:

* title
* year of release
* certificate
* runtime
* genre
* rating
* votes
* revenue

**robotstxt**

Let us check if we can scrape the data from the website using paths\_allowed()  
from **robotstxt** package.

paths\_allowed(

paths = c("https://www.imdb.com/search/title?groups=top\_250&sort=user\_rating")

)

##

www.imdb.com No encoding supplied: defaulting to UTF-8.

## [1] TRUE

Since it has returned TRUE, we will go ahead and download the web page using  
read\_html() from **xml2** package.

imdb <- read\_html("https://www.imdb.com/search/title?groups=top\_250&sort=user\_rating")

imdb

## {xml\_document}

##

## [1] \n\n\n

**Title**



As we did in the previous case study, we will look at the HTML code of the IMDB

web page and locate the title of the movies in the following way:

* hyperlink inside

tag

* section identified with the class .lister-item-content

In other words, the title of the movie is inside a hyperlink () which

is inside a level 3 heading (

) within a section identified by the class

.lister-item-content.

imdb %>%

html\_nodes(".lister-item-content h3 a") %>%

html\_text() -> movie\_title

movie\_title

## [1] "The Shawshank Redemption"

## [2] "The Godfather"

## [3] "The Dark Knight"

## [4] "The Godfather: Part II"

## [5] "The Lord of the Rings: The Return of the King"

## [6] "Pulp Fiction"

## [7] "Schindler's List"

## [8] "Il buono, il brutto, il cattivo"

## [9] "12 Angry Men"

## [10] "Inception"

## [11] "Fight Club"

## [12] "The Lord of the Rings: The Fellowship of the Ring"

## [13] "Forrest Gump"

## [14] "The Lord of the Rings: The Two Towers"

## [15] "The Matrix"

## [16] "Goodfellas"

## [17] "Star Wars: Episode V - The Empire Strikes Back"

## [18] "One Flew Over the Cuckoo's Nest"

## [19] "Shichinin no samurai"

## [20] "Interstellar"

## [21] "Cidade de Deus"

## [22] "Sen to Chihiro no kamikakushi"

## [23] "Saving Private Ryan"

## [24] "The Green Mile"

## [25] "La vita è bella"

## [26] "The Usual Suspects"

## [27] "Se7en"

## [28] "Léon"

## [29] "The Silence of the Lambs"

## [30] "Star Wars"

## [31] "It's a Wonderful Life"

## [32] "Andhadhun"

## [33] "Dangal"

## [34] "Spider-Man: Into the Spider-Verse"

## [35] "Avengers: Infinity War"

## [36] "Whiplash"

## [37] "The Intouchables"

## [38] "The Prestige"

## [39] "The Departed"

## [40] "The Pianist"

## [41] "Memento"

## [42] "Gladiator"

## [43] "American History X"

## [44] "The Lion King"

## [45] "Terminator 2: Judgment Day"

## [46] "Nuovo Cinema Paradiso"

## [47] "Hotaru no haka"

## [48] "Back to the Future"

## [49] "Raiders of the Lost Ark"

## [50] "Apocalypse Now"

**Year of Release**



The year in which a movie was released can be located in the following way:

* tag identified by the class .lister-item-year
* nested inside a level 3 heading (

)

* part of section identified by the class .lister-item-content

imdb %>%

html\_nodes(".lister-item-content h3 .lister-item-year") %>%

html\_text()

## [1] "(1994)" "(1972)" "(2008)" "(1974)" "(2003)" "(1994)" "(1993)"

## [8] "(1966)" "(1957)" "(2010)" "(1999)" "(2001)" "(1994)" "(2002)"

## [15] "(1999)" "(1990)" "(1980)" "(1975)" "(1954)" "(2014)" "(2002)"

## [22] "(2001)" "(1998)" "(1999)" "(1997)" "(1995)" "(1995)" "(1994)"

## [29] "(1991)" "(1977)" "(1946)" "(2018)" "(2016)" "(2018)" "(2018)"

## [36] "(2014)" "(2011)" "(2006)" "(2006)" "(2002)" "(2000)" "(2000)"

## [43] "(1998)" "(1994)" "(1991)" "(1988)" "(1988)" "(1985)" "(1981)"

## [50] "(1979)"

If you look at the output, the year is enclosed in round brackets and is a  
character vector. We need to do 2 things now:

* remove the round bracket
* convert year to class Date instead of character

We will use str\_sub() to extract the year and convert it to Date using  
as.Date() with the format %Y. Finally, we use year() from **lubridate**  
package to extract the year from the previous step.

imdb %>%

html\_nodes(".lister-item-content h3 .lister-item-year") %>%

html\_text() %>%

str\_sub(start = 2, end = 5) %>%

as.Date(format = "%Y") %>%

year() -> movie\_year

movie\_year

## [1] 1994 1972 2008 1974 2003 1994 1993 1966 1957 2010 1999 2001 1994 2002

## [15] 1999 1990 1980 1975 1954 2014 2002 2001 1998 1999 1997 1995 1995 1994

## [29] 1991 1977 1946 2018 2016 2018 2018 2014 2011 2006 2006 2002 2000 2000

## [43] 1998 1994 1991 1988 1988 1985 1981 1979

**Certificate**



The certificate given to the movie can be located in the following way:

* tag identified by the class .certificate
* nested inside a paragraph (

)

* part of section identified by the class .lister-item-content

imdb %>%

html\_nodes(".lister-item-content p .certificate") %>%

html\_text() -> movie\_certificate

movie\_certificate

## [1] "A" "A" "UA" "PG-13" "A" "A" "UA" "A"

## [9] "PG-13" "PG-13" "PG-13" "A" "A" "PG" "UA" "R"

## [17] "PG" "A" "A" "PG-13" "A" "R" "A" "A"

## [25] "U" "PG" "UA" "U" "U" "UA" "A" "UA"

## [33] "PG-13" "A" "R" "R" "R" "A" "U" "U"

## [41] "R" "U" "PG" "R"

**Runtime**



The runtime of the movie can be located in the following way:

* tag identified by the class .runtime
* nested inside a paragraph (

)

* part of section identified by the class .lister-item-content

imdb %>%

html\_nodes(".lister-item-content p .runtime") %>%

html\_text()

## [1] "142 min" "175 min" "152 min" "202 min" "201 min" "154 min" "195 min"

## [8] "161 min" "96 min" "148 min" "139 min" "178 min" "142 min" "179 min"

## [15] "136 min" "146 min" "124 min" "133 min" "207 min" "169 min" "130 min"

## [22] "125 min" "169 min" "189 min" "116 min" "106 min" "127 min" "110 min"

## [29] "118 min" "121 min" "130 min" "139 min" "161 min" "117 min" "149 min"

## [36] "106 min" "112 min" "130 min" "151 min" "150 min" "113 min" "155 min"

## [43] "119 min" "88 min" "137 min" "155 min" "89 min" "116 min" "115 min"

## [50] "147 min"

If you look at the output, it includes the text **min** and is of type  
character. We need to do 2 things here:

* remove the text **min**
* convert to type numeric

We will try the following:

* use str\_split() to split the result using space as a separator
* extract the first element from the resulting list using map\_chr()
* use as.numeric() to convert to a number

imdb %>%

html\_nodes(".lister-item-content p .runtime") %>%

html\_text() %>%

str\_split(" ") %>%

map\_chr(1) %>%

as.numeric() -> movie\_runtime

movie\_runtime

## [1] 142 175 152 202 201 154 195 161 96 148 139 178 142 179 136 146 124

## [18] 133 207 169 130 125 169 189 116 106 127 110 118 121 130 139 161 117

## [35] 149 106 112 130 151 150 113 155 119 88 137 155 89 116 115 147

**Genre**



The genre of the movie can be located in the following way:

* tag identified by the class .genre
* nested inside a paragraph (

)

* part of section identified by the class .lister-item-content

imdb %>%

html\_nodes(".lister-item-content p .genre") %>%

html\_text()

## [1] "\nDrama "

## [2] "\nCrime, Drama "

## [3] "\nAction, Crime, Drama "

## [4] "\nCrime, Drama "

## [5] "\nAdventure, Drama, Fantasy "

## [6] "\nCrime, Drama "

## [7] "\nBiography, Drama, History "

## [8] "\nWestern "

## [9] "\nDrama "

## [10] "\nAction, Adventure, Sci-Fi "

## [11] "\nDrama "

## [12] "\nAdventure, Drama, Fantasy "

## [13] "\nDrama, Romance "

## [14] "\nAdventure, Drama, Fantasy "

## [15] "\nAction, Sci-Fi "

## [16] "\nBiography, Crime, Drama "

## [17] "\nAction, Adventure, Fantasy "

## [18] "\nDrama "

## [19] "\nAdventure, Drama "

## [20] "\nAdventure, Drama, Sci-Fi "

## [21] "\nCrime, Drama "

## [22] "\nAnimation, Adventure, Family "

## [23] "\nDrama, War "

## [24] "\nCrime, Drama, Fantasy "

## [25] "\nComedy, Drama, Romance "

## [26] "\nCrime, Mystery, Thriller "

## [27] "\nCrime, Drama, Mystery "

## [28] "\nAction, Crime, Drama "

## [29] "\nCrime, Drama, Thriller "

## [30] "\nAction, Adventure, Fantasy "

## [31] "\nDrama, Family, Fantasy "

## [32] "\nCrime, Thriller "

## [33] "\nAction, Biography, Drama "

## [34] "\nAnimation, Action, Adventure "

## [35] "\nAction, Adventure, Sci-Fi "

## [36] "\nDrama, Music "

## [37] "\nBiography, Comedy, Drama "

## [38] "\nDrama, Mystery, Sci-Fi "

## [39] "\nCrime, Drama, Thriller "

## [40] "\nBiography, Drama, Music "

## [41] "\nMystery, Thriller "

## [42] "\nAction, Adventure, Drama "

## [43] "\nDrama "

## [44] "\nAnimation, Adventure, Drama "

## [45] "\nAction, Sci-Fi "

## [46] "\nDrama "

## [47] "\nAnimation, Drama, War "

## [48] "\nAdventure, Comedy, Sci-Fi "

## [49] "\nAction, Adventure "

## [50] "\nDrama, War "

The output includes \n and white space, both of which will be removed using  
str\_trim().

imdb %>%

html\_nodes(".lister-item-content p .genre") %>%

html\_text() %>%

str\_trim() -> movie\_genre

movie\_genre

## [1] "Drama" "Crime, Drama"

## [3] "Action, Crime, Drama" "Crime, Drama"

## [5] "Adventure, Drama, Fantasy" "Crime, Drama"

## [7] "Biography, Drama, History" "Western"

## [9] "Drama" "Action, Adventure, Sci-Fi"

## [11] "Drama" "Adventure, Drama, Fantasy"

## [13] "Drama, Romance" "Adventure, Drama, Fantasy"

## [15] "Action, Sci-Fi" "Biography, Crime, Drama"

## [17] "Action, Adventure, Fantasy" "Drama"

## [19] "Adventure, Drama" "Adventure, Drama, Sci-Fi"

## [21] "Crime, Drama" "Animation, Adventure, Family"

## [23] "Drama, War" "Crime, Drama, Fantasy"

## [25] "Comedy, Drama, Romance" "Crime, Mystery, Thriller"

## [27] "Crime, Drama, Mystery" "Action, Crime, Drama"

## [29] "Crime, Drama, Thriller" "Action, Adventure, Fantasy"

## [31] "Drama, Family, Fantasy" "Crime, Thriller"

## [33] "Action, Biography, Drama" "Animation, Action, Adventure"

## [35] "Action, Adventure, Sci-Fi" "Drama, Music"

## [37] "Biography, Comedy, Drama" "Drama, Mystery, Sci-Fi"

## [39] "Crime, Drama, Thriller" "Biography, Drama, Music"

## [41] "Mystery, Thriller" "Action, Adventure, Drama"

## [43] "Drama" "Animation, Adventure, Drama"

## [45] "Action, Sci-Fi" "Drama"

## [47] "Animation, Drama, War" "Adventure, Comedy, Sci-Fi"

## [49] "Action, Adventure" "Drama, War"

**Rating**



The rating of the movie can be located in the following way:

* part of the section identified by the class .ratings-imdb-rating
* nested within the section identified by the class .ratings-bar
* the rating is present within the tag as well as in the  
  data-value attribute
* instead of using html\_text(), we will use html\_attr() to extract the  
  value of the attribute data-value

Try using html\_text() and see what happens! You may include the tag  
or the classes associated with tag as well.

imdb %>%

html\_nodes(".ratings-bar .ratings-imdb-rating") %>%

html\_attr("data-value")

## [1] "9.3" "9.2" "9" "9" "8.9" "8.9" "8.9" "8.9" "8.9" "8.8" "8.8"

## [12] "8.8" "8.8" "8.7" "8.7" "8.7" "8.7" "8.7" "8.7" "8.6" "8.6" "8.6"

## [23] "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.6" "8.5" "8.5"

## [34] "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5" "8.5"

## [45] "8.5" "8.5" "8.5" "8.5" "8.5" "8.5"

Since rating is returned as a character vector, we will use as.numeric() to  
convert it into a number.

imdb %>%

html\_nodes(".ratings-bar .ratings-imdb-rating") %>%

html\_attr("data-value") %>%

as.numeric() -> movie\_rating

movie\_rating

## [1] 9.3 9.2 9.0 9.0 8.9 8.9 8.9 8.9 8.9 8.8 8.8 8.8 8.8 8.7 8.7 8.7 8.7

## [18] 8.7 8.7 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.5 8.5 8.5

## [35] 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5

**XPATH**



To extract votes from the web page, we will use a different technique. In this  
case, we will use **xpath** and **attributes** to locate the total number of  
votes received by the top 50 movies.

**xpath** is specified using the following:

* tab
* attribute name
* attribute value

**Votes**

  
In case of votes, they are the following:

* meta
* itemprop
* ratingCount

Next, we are not looking to extract text value as we did in the previous examples  
using html\_text(). Here, we need to extract the value assigned to the  
content attribute within the tag using html\_attr().

imdb %>%

html\_nodes(xpath = '//meta[@itemprop="ratingCount"]') %>%

html\_attr('content')

## [1] "2072234" "1421858" "2038192" "986722" "1475228" "1620525" "1073941"

## [8] "615037" "585330" "1816814" "1658255" "1491788" "1588549" "1334175"

## [15] "1488615" "894737" "1039788" "822026" "279949" "1276303" "637576"

## [22] "549215" "1095877" "1000529" "545073" "897347" "1271115" "913037"

## [29] "1118408" "1109433" "352753" "38978" "118321" "172855" "616710"

## [36] "605162" "666065" "1052597" "1063762" "633432" "1021256" "1197974"

## [43] "941670" "822889" "897372" "198316" "192593" "922864" "802805"

## [50] "542139"

Finally, we convert the votes to a number using as.numeric().

imdb %>%

html\_nodes(xpath = '//meta[@itemprop="ratingCount"]') %>%

html\_attr('content') %>%

as.numeric() -> movie\_votes

movie\_votes

## [1] 2072234 1421858 2038192 986722 1475228 1620525 1073941 615037

## [9] 585330 1816814 1658255 1491788 1588549 1334175 1488615 894737

## [17] 1039788 822026 279949 1276303 637576 549215 1095877 1000529

## [25] 545073 897347 1271115 913037 1118408 1109433 352753 38978

## [33] 118321 172855 616710 605162 666065 1052597 1063762 633432

## [41] 1021256 1197974 941670 822889 897372 198316 192593 922864

## [49] 802805 542139

**Revenue**



We wanted to extract both revenue and votes without using **xpath** but the way  
in which they are structured in the HTML code forced us to use **xpath** to  
extract votes. If you look at the HTML code, both votes and revenue are located  
inside the same tag with the same attribute name and value i.e. there is no  
distinct way to identify either of them.

In case of revenue, the **xpath** details are as follows:

* name
* nv

Next, we will use html\_text() to extract the revenue.

imdb %>%

html\_nodes(xpath = '//span[@name="nv"]') %>%

html\_text()

## [1] "2,072,234" "$28.34M" "1,421,858" "$134.97M" "2,038,192"

## [6] "$534.86M" "986,722" "$57.30M" "1,475,228" "$377.85M"

## [11] "1,620,525" "$107.93M" "1,073,941" "$96.07M" "615,037"

## [16] "$6.10M" "585,330" "$4.36M" "1,816,814" "$292.58M"

## [21] "1,658,255" "$37.03M" "1,491,788" "$315.54M" "1,588,549"

## [26] "$330.25M" "1,334,175" "$342.55M" "1,488,615" "$171.48M"

## [31] "894,737" "$46.84M" "1,039,788" "$290.48M" "822,026"

## [36] "$112.00M" "279,949" "$0.27M" "1,276,303" "$188.02M"

## [41] "637,576" "$7.56M" "549,215" "$10.06M" "1,095,877"

## [46] "$216.54M" "1,000,529" "$136.80M" "545,073" "$57.60M"

## [51] "897,347" "$23.34M" "1,271,115" "$100.13M" "913,037"

## [56] "$19.50M" "1,118,408" "$130.74M" "1,109,433" "$322.74M"

## [61] "352,753" "38,978" "$1.19M" "118,321" "$12.39M"

## [66] "172,855" "$190.24M" "616,710" "$678.82M" "605,162"

## [71] "$13.09M" "666,065" "$13.18M" "1,052,597" "$53.09M"

## [76] "1,063,762" "$132.38M" "633,432" "$32.57M" "1,021,256"

## [81] "$25.54M" "1,197,974" "$187.71M" "941,670" "$6.72M"

## [86] "822,889" "$312.90M" "897,372" "$204.84M" "198,316"

## [91] "$11.99M" "192,593" "922,864" "$210.61M" "802,805"

## [96] "$248.16M" "542,139" "$83.47M"

To extract the revenue as a number, we need to do some string hacking as  
follows:

* extract values that begin with $
* omit missing values
* convert values to character using as.character()
* append NA where revenue is missing (rank 31 and 47)
* remove $ and M
* convert to number using as.numeric()

imdb %>%

html\_nodes(xpath = '//span[@name="nv"]') %>%

html\_text() %>%

str\_extract(pattern = "^\\$.\*") %>%

na.omit() %>%

as.character() %>%

append(values = NA, after = 30) %>%

append(values = NA, after = 46) %>%

str\_sub(start = 2, end = nchar(.) - 1) %>%

as.numeric() -> movie\_revenue

movie\_revenue

## [1] 28.34 134.97 534.86 57.30 377.85 107.93 96.07 6.10 4.36 292.58

## [11] 37.03 315.54 330.25 342.55 171.48 46.84 290.48 112.00 0.27 188.02

## [21] 7.56 10.06 216.54 136.80 57.60 23.34 100.13 19.50 130.74 322.74

## [31] NA 1.19 12.39 190.24 678.82 13.09 13.18 53.09 132.38 32.57

## [41] 25.54 187.71 6.72 312.90 204.84 11.99 NA 210.61 248.16 83.47

**Putting it all together…**

top\_50 <- tibble(title = movie\_title, release = movie\_year,

`runtime (mins)` = movie\_runtime, genre = movie\_genre, rating = movie\_rating,

votes = movie\_votes, `revenue ($ millions)` = movie\_revenue)

top\_50

## # A tibble: 50 x 7

## title release `runtime (mins)` genre rating votes `revenue ($ mil~

##

## 1 The Sha~ 1994 142 Drama 9.3 2.07e6 28.3

## 2 The God~ 1972 175 Crime,~ 9.2 1.42e6 135.

## 3 The Dar~ 2008 152 Action~ 9 2.04e6 535.

## 4 The God~ 1974 202 Crime,~ 9 9.87e5 57.3

## 5 The Lor~ 2003 201 Advent~ 8.9 1.48e6 378.

## 6 Pulp Fi~ 1994 154 Crime,~ 8.9 1.62e6 108.

## 7 Schindl~ 1993 195 Biogra~ 8.9 1.07e6 96.1

## 8 Il buon~ 1966 161 Western 8.9 6.15e5 6.1

## 9 12 Angr~ 1957 96 Drama 8.9 5.85e5 4.36

## 10 Incepti~ 2010 148 Action~ 8.8 1.82e6 293.

## # ... with 40 more rows

[packages ad](https://pkgs.rsquaredacademy.com/)

**Top Websites**

**Unfortunately, we had to drop this case study as the HTML code changed while we  
were working on this blog post. Remember, the third point we mentioned in the  
things to keep in mind, where we had warned that the design or underlying HTML  
code of the website may change. It just happened as we were finalizing this  
post.**

**RBI Governors**



In this case study, we are going to extract the list of  
RBI (Reserve Bank of India) Governors. The author of this blog post comes from  
an Economics background and as such was intereseted in knowing the professional  
background of the Governors prior to their taking charge at India’s central  
bank. We will extact the following details:

* name
* start of term
* end of term
* term (in days)
* background

**robotstxt**

Let us check if we can scrape the data from Wikipedia website using  
paths\_allowed() from **robotstxt** package.

paths\_allowed(

paths = c("https://en.wikipedia.org/wiki/List\_of\_Governors\_of\_Reserve\_Bank\_of\_India")

)

##

en.wikipedia.org

## [1] TRUE

Since it has returned TRUE, we will go ahead and download the web page using  
read\_html() from **xml2** package.

rbi\_guv <- read\_html("https://en.wikipedia.org/wiki/List\_of\_Governors\_of\_Reserve\_Bank\_of\_India")

rbi\_guv

## {xml\_document}

##

## [1] \n

**List of Governors**

The data in the Wikipedia page is luckily structured as a table and we can  
extract it using html\_table().

rbi\_guv %>%

html\_nodes("table") %>%

html\_table()

## [[1]]

## Governor of the Reserve Bank of India

## 1 IncumbentShaktikanta Das, IASsince 12 December 2018; 3 months ago (2018-12-12)

## 2 Appointer

## 3 Term length

## 4 Constituting instrument

## 5 Inaugural holder

## 6 Formation

## 7 Deputy

## 8 Website

## Governor of the Reserve Bank of India

## 1 IncumbentShaktikanta Das, IASsince 12 December 2018; 3 months ago (2018-12-12)

## 2 Appointments Committee of the Cabinet

## 3 Three years

## 4 Reserve Bank of India Act, 1934

## 5 Osborne Smith (1935–1937)

## 6 1 April 1935; 84 years ago (1935-04-01)

## 7 Deputy Governors of the Reserve Bank of India

## 8 rbi.org.in

##

## [[2]]

## No. Officeholder Portrait Term start Term end

## 1 1 Osborne Smith NA 1 April 1935 30 June 1937

## 2 2 James Braid Taylor NA 1 July 1937 17 February 1943

## 3 3 C. D. Deshmukh NA 11 August 1943ii 30 May 1949

## 4 4 Benegal Rama Rau NA 1 July 1949 14 January 1957

## 5 5 K. G. Ambegaonkar NA 14 January 1957 28 February 1957

## 6 6 H. V. R. Iyengar NA 1 March 1957 28 February 1962

## 7 7 P. C. Bhattacharya NA 1 March 1962 30 June 1967

## 8 8 Lakshmi Kant Jha NA 1 July 1967 3 May 1970

## 9 9 B. N. Adarkar NA 4 May 1970 15 June 1970

## 10 10 Sarukkai Jagannathan NA 16 June 1970 19 May 1975

## 11 11 N. C. Sen Gupta NA 19 May 1975 19 August 1975

## 12 12 K. R. Puri NA 20 August 1975 2 May 1977

## 13 13 M. Narasimham NA 3 May 1977 30 November 1977

## 14 14 I. G. Patel NA 1 December 1977 15 September 1982

## 15 15 Manmohan Singh NA 16 September 1982 14 January 1985

## 16 16 Amitav Ghosh NA 15 January 1985 4 February 1985

## 17 17 R. N. Malhotra NA 4 February 1985 22 December 1990

## 18 18 S. Venkitaramanan NA 22 December 1990 21 December 1992

## 19 19 C. Rangarajan NA 22 December 1992 21 November 1997

## 20 20 Bimal Jalan NA 22 November 1997 6 September 2003

## 21 21 Y. Venugopal Reddy NA 6 September 2003 5 September 2008

## 22 22 D. Subbarao NA 5 September 2008 4 September 2013

## 23 23 Raghuram Rajan NA 4 September 2013 4 September 2016

## 24 24 Urjit Patel NA 4 September 2016 11 December 2018

## 25 25 Shaktikanta Das NA 12 December 2018 Incumbent

## Term in office Background

## 1 821 days Banker

## 2 2057 days Indian Civil Service (ICS) officer

## 3 2150 days ICS officer

## 4 2754 days ICS officer

## 5 45 days ICS officer

## 6 1825 days ICS officer

## 7 1947 days Indian Audit and Accounts Service officer

## 8 1037 days ICS officer

## 9 42 days Economist

## 10 1798 days ICS officer

## 11 92 days ICS officer

## 12 621 days

## 13 211 days Career Reserve Bank of India officer

## 14 1749 days Economist

## 15 851 days Economist

## 16 20 days Banker

## 17 2147 days Indian Administrative Service (IAS) officer

## 18 730 days IAS officer

## 19 1795 days Economist

## 20 2114 days Economist

## 21 1826 days IAS officer

## 22 1825 days IAS officer

## 23 1096 days Economist

## 24 947 days Economist

## 25 118 days IAS officer

## Prior office(s)

## 1 Managing Governor of the Imperial Bank of India

## 2 Deputy Governor of the Reserve Bank of India\n\nController of Currency

## 3 Deputy Governor of the Reserve Bank of India\nCustodian of Enemy Property

## 4 Ambassador of India to the United States\n\nAmbassador of India to Japan\n\nChairman of Bombay Port Trust

## 5 Finance Secretary

## 6 Chairman of the State Bank of India

## 7 Chairman of the State Bank of India\nSecretary in the Ministry of Finance

## 8 Secretary to the Prime Minister of India

## 9 Executive Director at the International Monetary Fund

## 10 Executive Director at the World Bank

## 11 Banking Secretary

## 12 Chairman and Managing Director of the Life Insurance Corporation

## 13 Deputy Governor of the Reserve Bank of India

## 14 Director of the London School of Economics\n\nDeputy Administrator of the United Nations Development Programme\nChief Economic Adviser to the Government of India

## 15 Secretary in the Ministry of Finance\n\nChief Economic Adviser to the Government of India

## 16 Deputy Governor of the Reserve Bank of India\n\nChairman of the Allahabad Bank

## 17 Finance Secretary\n\nExecutive Director at the International Monetary Fund

## 18 Finance Secretary

## 19 Deputy Governor of the Reserve Bank of India

## 20 Finance Secretary\n\nBanking Secretary\n\nChief Economic Adviser to the Government of India

## 21 Executive Director at the International Monetary Fund\n\nDeputy Governor of the Reserve Bank of India

## 22 Finance Secretary\n\nMember-Secretary of the Prime Minister's Economic Advisory Council

## 23 Chief Economic Adviser to the Government of India

## 24 Deputy Governor of the Reserve Bank

## 25 Member of the Fifteenth Finance Commission\nSherpa of India to the G20\nEconomic Affairs Secretary\nRevenue Secretary

## Reference(s)

## 1 [1]

## 2 [2]

## 3

## 4

## 5

## 6

## 7

## 8

## 9

## 10

## 11

## 12

## 13

## 14

## 15

## 16

## 17

## 18

## 19

## 20

## 21

## 22

## 23

## 24

## 25 [3][4][5]

##

## [[3]]

## vte Governors of the Reserve Bank of India

## 1 NA

## vte Governors of the Reserve Bank of India

## 1 Osborne Smith (1935–37)\nJames Braid Taylor (1937–43)\nC. D. Deshmukh (1943–49)\nBenegal Rama Rau (1949–57)\nK. G. Ambegaonkar (1957)\nH. V. R. Iyengar (1957–62)\nP. C. Bhattacharya (1962–67)\nLakshmi Kant Jha (1967–70)\nB. N. Adarkar (1970)\nS. Jagannathan (1970–75)\nN. C. Sen Gupta (1975)\nK. R. Puri (1975–77)\nM. Narasimham (1977)\nI. G. Patel (1977–82)\nManmohan Singh (1982–85)\nAmitav Ghosh (1985)\nR. N. Malhotra (1985–90)\nS. Venkitaramanan (1990–92)\nC. Rangarajan (1992–97)\nBimal Jalan (1997–2003)\nY. Venugopal Reddy (2003–08)\nDuvvuri Subbarao (2008–13)\nRaghuram Rajan (2013–16)\nUrjit Patel (2016–2018)\nShaktikanta Das (2018–Incumbent)

## vte Governors of the Reserve Bank of India

## 1 Osborne Smith (1935–37)\nJames Braid Taylor (1937–43)\nC. D. Deshmukh (1943–49)\nBenegal Rama Rau (1949–57)\nK. G. Ambegaonkar (1957)\nH. V. R. Iyengar (1957–62)\nP. C. Bhattacharya (1962–67)\nLakshmi Kant Jha (1967–70)\nB. N. Adarkar (1970)\nS. Jagannathan (1970–75)\nN. C. Sen Gupta (1975)\nK. R. Puri (1975–77)\nM. Narasimham (1977)\nI. G. Patel (1977–82)\nManmohan Singh (1982–85)\nAmitav Ghosh (1985)\nR. N. Malhotra (1985–90)\nS. Venkitaramanan (1990–92)\nC. Rangarajan (1992–97)\nBimal Jalan (1997–2003)\nY. Venugopal Reddy (2003–08)\nDuvvuri Subbarao (2008–13)\nRaghuram Rajan (2013–16)\nUrjit Patel (2016–2018)\nShaktikanta Das (2018–Incumbent)

## vte Governors of the Reserve Bank of India

## 1 NA

There are 2 tables in the web page and we are interested in the second table.  
Using extract2() from the **magrittr** package, we will extract the table  
containing the details of the Governors.

rbi\_guv %>%

html\_nodes("table") %>%

html\_table() %>%

extract2(2) -> profile

**Sort**

Let us arrange the data by number of days served. The Term in office column  
contains this information but it also includes the text days. Let us split this  
column into two columns, term and days, using separate() from tidyr and  
then select the columns Officeholder and term and arrange it in descending  
order using desc().

profile %>%

separate(`Term in office`, into = c("term", "days")) %>%

select(Officeholder, term) %>%

arrange(desc(as.numeric(term)))

## Officeholder term

## 1 Benegal Rama Rau 2754

## 2 C. D. Deshmukh 2150

## 3 R. N. Malhotra 2147

## 4 Bimal Jalan 2114

## 5 James Braid Taylor 2057

## 6 P. C. Bhattacharya 1947

## 7 Y. Venugopal Reddy 1826

## 8 H. V. R. Iyengar 1825

## 9 D. Subbarao 1825

## 10 Sarukkai Jagannathan 1798

## 11 C. Rangarajan 1795

## 12 I. G. Patel 1749

## 13 Raghuram Rajan 1096

## 14 Lakshmi Kant Jha 1037

## 15 Urjit Patel 947

## 16 Manmohan Singh 851

## 17 Osborne Smith 821

## 18 S. Venkitaramanan 730

## 19 K. R. Puri 621

## 20 M. Narasimham 211

## 21 Shaktikanta Das 118

## 22 N. C. Sen Gupta 92

## 23 K. G. Ambegaonkar 45

## 24 B. N. Adarkar 42

## 25 Amitav Ghosh 20

**Backgrounds**

What we are interested is in the background of the Governors? Use count()  
from **dplyr** to look at the backgound of the Governors and the respective  
counts.

profile %>%

count(Background)

## # A tibble: 9 x 2

## Background n

##

## 1 "" 1

## 2 Banker 2

## 3 Career Reserve Bank of India officer 1

## 4 Economist 7

## 5 IAS officer 4

## 6 ICS officer 7

## 7 Indian Administrative Service (IAS) officer 1

## 8 Indian Audit and Accounts Service officer 1

## 9 Indian Civil Service (ICS) officer 1

Let us club some of the categories into **Bureaucrats** as they belong to the  
Indian Administrative/Civil Services. The missing data will be renamed as No Info.  
The category Career Reserve Bank of India officer is renamed as RBI Officer  
to make it more concise.

profile %>%

pull(Background) %>%

fct\_collapse(

Bureaucrats = c("IAS officer", "ICS officer",

"Indian Administrative Service (IAS) officer",

"Indian Audit and Accounts Service officer",

"Indian Civil Service (ICS) officer"),

`No Info` = c(""),

`RBI Officer` = c("Career Reserve Bank of India officer")

) %>%

fct\_count() %>%

rename(background = f, count = n) -> backgrounds

backgrounds

## # A tibble: 5 x 2

## background count

##

## 1 No Info 1

## 2 Banker 2

## 3 RBI Officer 1

## 4 Economist 7

## 5 Bureaucrats 14

Hmmm.. So there were more bureaucrats than economists.

backgrounds %>%

ggplot() +

geom\_col(aes(background, count), fill = "blue") +

xlab("Background") + ylab("Count") +

ggtitle("Background of RBI Governors")

