**R Packages Covered**:

* rvest & jsonlite – Web Scraping HTML and working with JSON data
* purrr – Iteration through lists using map() and safely()
* stringr – Text manipulation
* ggplot2 – Data visualization and understanding data

**Scraping Website Data and Analyzing Specialized Bicycles**

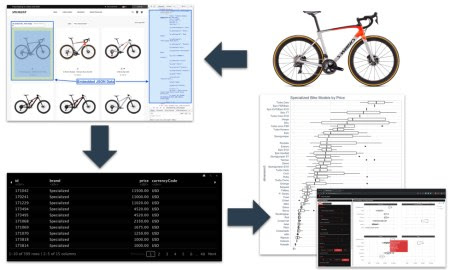
Happy Monday everyone!

I welcome any questions and would appreciate any feedback. Thank you for your time, BSU community!

**My Workflow**

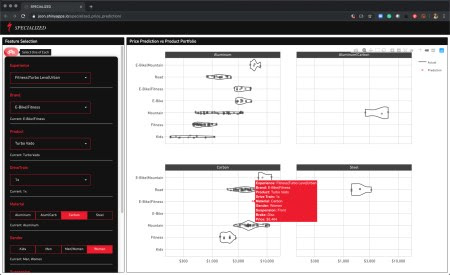
Here’s a diagram of the workflow I used to web scrape the Specialized Data and create an application:

1. Start with URL of Specialized Bicycles
2. Use rvest and jsonlite to extract product data
3. Clean up data into “tidy” format using purrr and stringr
4. Visualize product prices with ggplot2
5. Make a Shiny Web App using the Business Science 102 Course.

[](https://www.business-science.io/code-tools/2019/10/07/rvest-web-scraping.html#workflow)

**My Shiny App**

I built a shiny web application to recommend product prices of new bicylces, which you can try out:

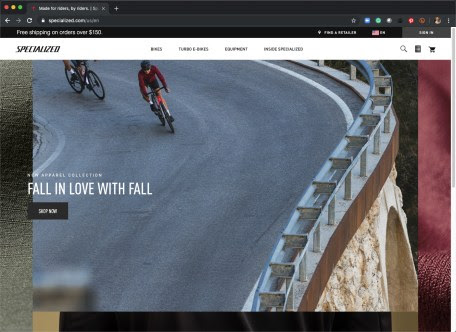
[](https://joon.shinyapps.io/specialized_price_prediction/)

**Tutorial – Web Scraping with rvest**

This tutorial showcases how to web scrape websites using rvest and purrr. I’ll show how to collect data on the 2020 Specialized Bicycles Product Collection, a useful task in ***building a strategic database of product and competitive information for an organization***.

**1. Set Up**

**1.1 Introduction**



**1.2 Check Robots**

Always look at the website’s robots.txt to check crawling permissions.

**1.3 Load Libraries**

Let’s start with loading libraries that we know we will need.

# Load libraries

library(rvest) # HTML Hacking & Web Scraping

library(jsonlite) # JSON manipulation

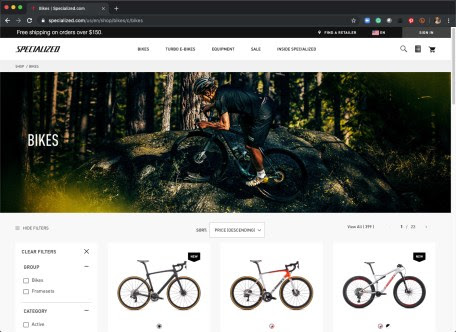
library(tidyverse) # Data Manipulation

library(tidyquant) # ggplot2 theme

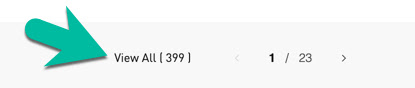
library(xopen) # Opens URL in Browser

library(knitr) # Pretty HTML Tables

**1.4 Check Out the Products**



We can click “View All” to view all 399 bikes on a single page. This makes things a bit easier when it comes time to scrape so we don’t have to iterate over multiple pages.



Save the URL.

# URL to View All Bikes

url <- "<https://www.specialized.com/us/en/shop/bikes/c/bikes?q=%3Aprice-desc%3Aarchived%3Afalse&show=All>"

You can then use xopen() to open the URL in your default web browser.

# View URL in Browser

xopen(url)

**1.5 Read HTML**

Load the HTML code into an object using read\_html(). We’ve just grabbed all of the HTML from that page.

# Read HTML from URL

html <- read\_html(url)

html

## {html\_document}

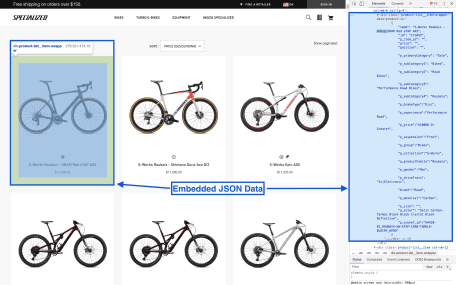
##

## [1] \n\n2. Get the Raw Data

Use [Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools) to locate the product information. In our case, there is a JSON-like dictionary containing what we need.

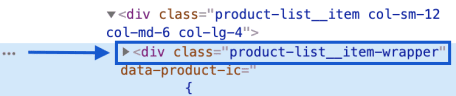
**2.1 Locate Data with Chrome DevTools**

Find the data by using the hover tool.



**2.2 Find Product Data Nodes**

Find the nodes where the product data lives.



**2.3 Filter HTML to Isolate Nodes**

Copy and paste the class into the html\_nodes() function from the rvest library.

html %>%

html\_nodes(".product-list\_\_item-wrapper")

**2.4 Find the Attribute That Contains the Data**



**2.4 Extract the Attribute Data**

Extract the attributes with the html\_attr() function and store it as a JSON object. Note that we’ll need to convert the JSON into a better format for analysis (more on this in a minute).

# Store JSON as object

json <- html %>%

html\_nodes(".product-list\_\_item-wrapper") %>%

html\_attr("data-product-ic")

# Show the 1st JSON element (1st bike of 399 bikes)

json[1]

## [1] "{\"name\":\"S-Works Roubaix - SRAM Red eTap AXS\",\"id\":\"171042\",\"brand\":\"Specialized\",\"price\":11500,\"currencyCode\":\"USD\",\"position\":\"\",\"variant\":\"61\",\"dimension1\":\"Bikes\",\"dimension2\":\"Road\",\"dimension3\":\"Roubaix\",\"dimension4\":\"\",\"dimension5\":\"Performance Road\",\"dimension6\":\"S-Works\",\"dimension7\":\"\",\"dimension8\":\"Men/Women\"}"

**3. Format as Tidy Data with purrr**

Tidy data is a tibble (data frame) that has one row for the each of the Specialized Bike Models and columns for each of the features like model name, price, and various categories (denoted as dimensions).

**3.1 Make a Function that Converts JSON to Tibble**

This function is just a wrapper for toJSON from the jsonlite package. The only addition is converting the data frame to a tibble using as\_tibble().

# Make Function

from\_json\_to\_tibble <- function(json) {

json %>%

fromJSON() %>%

as\_tibble()

}

We can run this on the first element of the list.

json[1] %>%

from\_json\_to\_tibble() %>%

knitr::kable()

| **name** | **id** | **brand** | **price** | **currencyCode** | **position** | **variant** | **dimension1** | **dimension2** | **dimension3** | **dimension4** | **dimension5** | **dimension6** | **dimension7** | **dimension8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S-Works Roubaix – SRAM Red eTap AXS | 171042 | Specialized | 11500 | USD |  | 61 | Bikes | Road | Roubaix |  | Performance Road | S-Works |  | Men/Women |

**3.2 Iterate to All JSON Elements**

We’ll use map() to iteratively apply our from\_json\_to\_tibble() function. If we just run this, the iterative conversion error’s out – This is common in long-running iterative scripts. We can get around this using the safely() function, which isolates the errors and allows the iteration to continue (instead of grinding to a hault).

# Iterate - All JSON objects ----

bike\_data\_list <- json %>%

map(safely(from\_json\_to\_tibble))

**3.3 Inspect First Converted Element**

We can see that a list is returned with 2 elements for each item:

1. **$result** – Contains the result. If conversion succeeds, we get a tibble. If error, we get NULL.
2. **$error** – Contains the error message (if error). Otherwise, we get NULL.

# Inspect first conversion: $result & $error

bike\_data\_list[1]

## [[1]]

## [[1]]$result

## # A tibble: 1 x 15

## name id brand price currencyCode position variant dimension1

##

## 1 S-Wo… 1710… Spec… 11500 USD "" 61 Bikes

## # … with 7 more variables: dimension2 , dimension3 ,

## # dimension4 , dimension5 , dimension6 ,

## # dimension7 , dimension8

##

## [[1]]$error

## NULL

**3.4 Inspect for Errors**

We are bound to get errors in this JSON conversion process for 399 bikes. Let’s check to see where errors occurred.

error\_tbl <- bike\_data\_list %>%

# Grab just the $error elements

map(~ pluck(., "error")) %>%

# Convert from list to tibble

enframe(name = "row") %>%

# Return TRUE if element has error

mutate(is\_error = map(value, function(x) !is.null(x))) %>%

# Unnest nested list

unnest(is\_error) %>%

# Filter where error == TRUE

filter(is\_error)

error\_tbl

## # A tibble: 2 x 3

## row value is\_error

##

## 1 222 TRUE

## 2 288 TRUE

**3.5 What happened?**

We got two errors – Bike 222 and 288. We can use pluck() to grab the first error in the “value” column. It’s the result of an errant " symbol that represents inches.

error\_tbl %>% pluck("value", 1)

##

We can get around this by replacing the ". Let’s re-run the code using the str\_replace() function to replace the quote.

json[222] %>%

str\_replace('22.5\\" TT', '22.5 TT') %>%

from\_json\_to\_tibble()

## Error: lexical error: invalid char in json text.

## imension4":"","dimension5":""BMX / Dirt Jump"","dimension6":

## (right here) ------^

We get another error. There is an errant set of quotes around “BMX / Dirt Jump”. We can use str\_replace() again to resolve. Success!

json[222] %>%

str\_replace('\\"BMX / Dirt Jump\\"', 'BMX / Dirt Jump') %>%

str\_replace('22.5\\" TT', '22.5 TT') %>%

from\_json\_to\_tibble()

## # A tibble: 1 x 15

## name id brand price currencyCode position variant dimension1

##

## 1 P.Sl… 1710… Spec… 2500 USD "" 22.5 TT Bikes

## # … with 7 more variables: dimension2 , dimension3 ,

## # dimension4 , dimension5 , dimension6 ,

## # dimension7 , dimension8

**3.6 Run Again – Success – Errors Fixed!**

We can try one more time, now using the str\_replace() to remove the quotes causing conversion errors, and map\_dfr() to return a data frame stacked row-wise.

# Fix errors, re-run

bike\_features\_tbl <- json %>%

str\_replace('\\"BMX / Dirt Jump\\"', 'BMX / Dirt Jump') %>%

str\_replace('22.5\\" TT', '22.5 TT') %>%

map\_dfr(from\_json\_to\_tibble)

# Show first 6 rows

bike\_features\_tbl %>%

head() %>%

kable()

| **name** | **id** | **brand** | **price** | **currencyCode** | **position** | **variant** | **dimension1** | **dimension2** | **dimension3** | **dimension4** | **dimension5** | **dimension6** | **dimension7** | **dimension8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S-Works Roubaix – SRAM Red eTap AXS | 171042 | Specialized | 11500 | USD |  | 61 | Bikes | Road | Roubaix |  | Performance Road | S-Works |  | Men/Women |
| S-Works Roubaix – Shimano Dura-Ace Di2 | 170241 | Specialized | 11000 | USD |  | 56 | Bikes | Road | Roubaix |  | Performance Road | S-Works |  | Men/Women |
| S-Works Epic AXS | 171229 | Specialized | 11020 | USD |  | S | Bikes | Mountain | Epic FSR/Epic |  | Cross Country | S-Works |  | Men/Women |
| Stumpjumper EVO Comp Carbon 29 | 173494 | Specialized | 4520 | USD |  | S3 | Bikes | Mountain | Stumpjumper EVO |  | Trail |  |  | Men/Women |
| Stumpjumper EVO Comp Carbon 27.5 | 173495 | Specialized | 4520 | USD |  | S2 | Bikes | Mountain | Stumpjumper EVO |  | Trail |  |  | Men/Women |
| Fuse Expert 29 | 171068 | Specialized | 2150 | USD |  | XS | Bikes | Mountain | Fuse |  | Trail |  |  | Men/Women |

**4. Explore Bike Models**

I want to understand how price depends on various features like model, type of bike (electric, mountain, road), and other features that will eventually be used in my XGBoost Machine Learning model inside of my Shiny Web App.

**4.1 Most and Least Expensive Bike Models**

There’s a clear relationship between price and “Dimension 3” (bike model). We can see this visually.

bike\_features\_tbl %>%

select(dimension3, price) %>%

mutate(dimension3 = as\_factor(dimension3) %>%

fct\_reorder(price, .fun = median)) %>%

# Plot

ggplot(aes(dimension3, price)) +

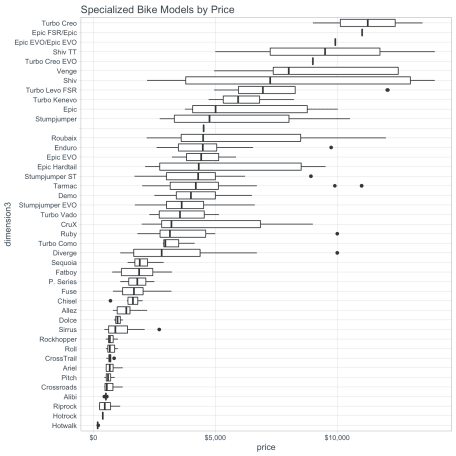
geom\_boxplot() +

coord\_flip() +

theme\_tq() +

scale\_y\_continuous(labels = scales::dollar\_format()) +

labs(title = "Specialized Bike Models by Price")



**4.2 S-Works Effect**

I also noticed that “S-Works” is Specialized’s Premium Brand. We can update the ggplot2 visualization to segment bikes with “S-Works” in the model name to visually compare the “S-Works Effect”. I see that the S-Works bikes tend to have a higher median price than “non-S-Works”.

bike\_features\_tbl %>%

select(name, price, dimension3) %>%

mutate(s\_works = ifelse(str\_detect(name, "S-Works"), "S-Works", "Not S-Works")) %>%

mutate(dimension3 = as\_factor(dimension3) %>%

fct\_reorder(price, .fun = median)) %>%

# Plot

ggplot(aes(dimension3, price, color = s\_works)) +

geom\_boxplot() +

coord\_flip() +

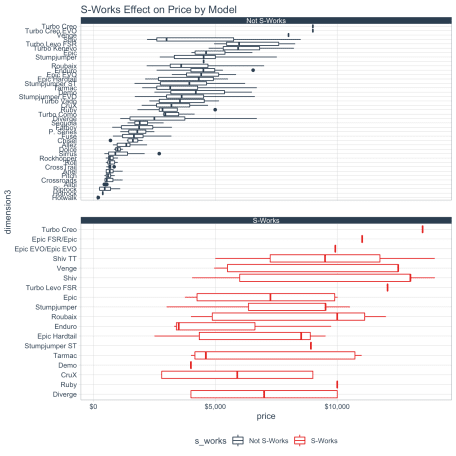
facet\_wrap(~ s\_works, ncol = 1, scales = "free\_y") +

theme\_tq() +

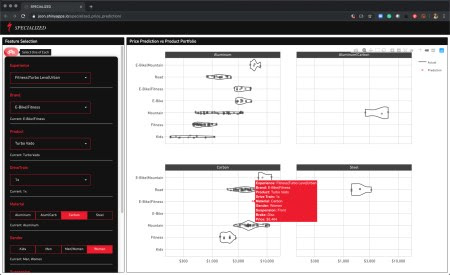
scale\_color\_tq() +

scale\_y\_continuous(labels = scales::dollar\_format()) +

labs(title = "S-Works Effect on Price by Model")



**5. Predictive Web Application**

[](https://joon.shinyapps.io/specialized_price_prediction/" \t "_blank)

**Parting Thoughts**

Web-scraping with rvest has fundamentally changed the way I understand the Internet. Once I realized that the entire Internet (well, most of it) is basically just one big database, it rocked my world. I highly encourage you to sign up.

Using the data, I was able to make and deploy a Shiny web application that uses an XGBoost Machine Learning model to predict and recommend bicycle prices. This is just one way that businesses can use the strategic database. If you want to learn shiny, I highly recommend the Shiny Web Applications Course by Business Science.

**Other Student Articles You Might Enjoy**

Here are two more Student Success Tutorials related to scraping data and building shiny applications.