NHSDataDictionaRy Workshop

NHS-R Community

Gary Hutson - Senior Data Scientist

04/10/2021

Introduction

Welcome to the NHSDataDictionaRy workshop. Today we will learn how to work with the data dictionary for NHS lookup tasks, as well as how to extend the package to work with any website. The concentration of the workshop will be broken down as such:

- Getting familiar with the NHSDataDictionaRy package and all the underlying functions
- Understanding the nhs_data_elements() elements function and how to filter on this
- Gather text from any website and perform some text cleaning operations on the text, using a combination of functions contained in the package
- Using the TableR function to retrieve HTML tables from the data dictionary site and then extending this to other websites
- Working with XPath website elements with the NHSDataDictionaRy package

Watch our previous webinar on how to use this package

This was taken from a webinar we recorded a how to guide from the launch of the NHSDataDictionaRy package. This may be useful for referring to, but most of what is covered in the video will be in today's workshop.

Loading libraries

Loading libraries in NHSDataDictionaRy package

To load the libraries needed for this session you will need to bring in the following:

```
library(NHSDataDictionaRy)
```

This will bring in the libraries needed to work with the package. The next step would be to check the dependencies in the package:

```
packrat:::recursivePackageDependencies("NHSDataDictionaRy", lib.loc = .libPaths())
```

```
##  [1] "R6"         "askpass"    "cli"        "crayon"     "curl"
##  [6] "dplyr"      "ellipsis"   "fansi"      "generics"   "glue"
## [11] "httr"       "jsonlite"   "lifecycle"  "magrittr"   "mime"
## [16] "openssl"    "pillar"     "pkgconfig"  "purrr"      "rlang"
## [21] "rvest"      "selectr"    "stringi"    "stringr"    "sys"
## [26] "tibble"     "tidyselect" "utf8"       "vctrs"      "xml2"
```

Working with the TableR, nhs_data_elements and nhs_tableR functions

Starting with the nhs_data_elements() function

To retrieve the list of currrent NHS lookups in the table finder we use the underlying function:

```
nhs_data_table <- nhs_data_elements()
```

This function will then return the full list of data elements the package has scraped from the NHS Data Dictionary website.

An example of what has been scraped is on the next slide.

Working with the scraped values

These scraped values will allow us to access the XPath elements of the website directly (don't worry if you don't know what an XPath is, we will get to that later on). To pull down a lookup table using the nhs_data_elements master reference we can use the TableR function to achieve this:

Getting the Activity Treatment Function Codes

```
# Filter the table
act_treatment_function <- nhs_data_table %>%
  dplyr::filter(link_name == "ACTIVITY TREATMENT FUNCTION CODE")
```

On the next slide you will see this has returned the lookup that I require:

```r
# Use the TableR function to pull back the data table
act_treat_lookup <- NHSDataDictionaRy::tableR(url=act_treatment_function$full_url,
                          xpath=act_treatment_function$xpath_nat_code,
                          title="NHS Hospital Activity Treatment Function Code")

print(head(act_treat_lookup,5))
```

```
## # A tibble: 5 x 4
##   Code  Description                Dict_Type                  DttmExtracted
##   <chr> <chr>                      <chr>                      <dttm>
## 1 100   General Surgery Service    NHS Hospital Activity Tr~ 2021-11-03 08:58:46
## 2 101   Urology Service            NHS Hospital Activity Tr~ 2021-11-03 08:58:46
## 3 102   Transplant Surgery Service NHS Hospital Activity Tr~ 2021-11-03 08:58:46
## 4 103   Breast Surgery Service     NHS Hospital Activity Tr~ 2021-11-03 08:58:46
## 5 104   Colorectal Surgery Service NHS Hospital Activity Tr~ 2021-11-03 08:58:46
```

This lookup will return the lookup table needed.

What if the element is not returned?

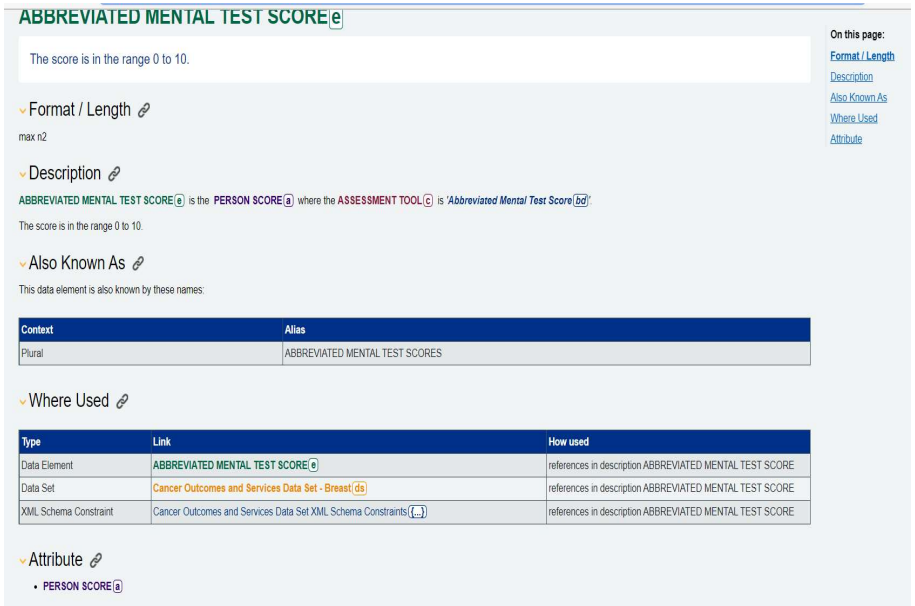
I include an example of when an element is not returned, as not all elements from the list have corresponding HTML data tables to extract. A worked example is below:

```
# Filter the table
abb_ment_test_score <- nhs_data_table %>%
  dplyr::filter(link_name == "ABBREVIATED MENTAL TEST SCORE")

# This will show a NULL return, as there is no HTML table present
null_output <- NHSDataDictionaRy::tableR(abb_ment_test_score$full_url,
                          abb_ment_test_score$xpath_nat_code)
```

```
## HTML Table does not exist for this URL.
## Check the URL online at:
## https://datadictionary.nhs.uk/data_elements/abbreviated_mental_test_score.html.
```

This shows a ***NULL*** return, as there is no HTML present for the national code specified. This can be highlighted by using the NHS Data Dictionary website to highlight this.

Checking the NHS Data Dictionary website



ABBREVIATED MENTAL TEST SCORE e

The score is in the range 0 to 10.

On this page:
Format / Length
Description
Also Known As
Where Used
Attribute

Format / Length

max n2

Description

ABBREVIATED MENTAL TEST SCORE e is the PERSON SCORE a where the ASSESSMENT TOOL c is 'Abbreviated Mental Test Score bd'.

The score is in the range 0 to 10.

Also Known As

This data element is also known by these names:

Context	Alias
Plural	ABBREVIATED MENTAL TEST SCORES

Where Used

Type	Link	How used
Data Element	ABBREVIATED MENTAL TEST SCORE e	references in description ABBREVIATED MENTAL TEST SCORE
Data Set	Cancer Outcomes and Services Data Set - Breast ds	references in description ABBREVIATED MENTAL TEST SCORE
XML Schema Constraint	Cancer Outcomes and Services Data Set XML Schema Constraints {...}	references in description ABBREVIATED MENTAL TEST SCORE

Attribute

- PERSON SCORE a

You can see that the website contains no national code table, thus the message.

You will not be able to retrieve these elements from the site, so the package presents you with a warning that these HTML table tables do not exist.

This is supposed to be informative and will allow you then to quickly inspect other items of interest.

Practice time - 10 minutes

Have a go at extracting elements from the nhs_data_elements() and TableR functions.

Introducing a quicker way to retrieve elements

Until now we have been working long hand and finding with retrieving the list of lookups and then using the TableR function. There is a more simple function in the package to use to do this, but I wanted to get you familiar with using the returned elements such as the URL and XPath, because we will come on to this again later.

nhs_table_findeR to the rescue

```
tfc <- NHSDataDictionaRy::nhs_table_findeR("ACTIVITY TREATMENT FUNCTION CODE",
                                           title="Treatment Function Code")

glimpse(tfc)
```

```
## Rows: 184
## Columns: 4
## $ Code          <chr> "100", "101", "102", "103", "104", "105", "106", "107", ~
## $ Description   <chr> "General Surgery Service", "Urology Service", "Transplan~
## $ Dict_Type     <chr> "Treatment Function Code", "Treatment Function Code", "T~
## $ DttmExtracted <dttm> 2021-11-03 08:58:46, 2021-11-03 08:58:46, 2021-11-03 08~
```

This function replaces the convoluted code we used earlier.

Using the lookup to join on to hospital data

We have this lookup and now we will generate some hospital data to match with our lookups:

```
set.seed(123)
spec_code <- rep(as.character(c(101,102,103,104,105,106,107, 108)), 12)
attends <- round(rnorm(length(spec_code), 500, 50))
breaches <- round(rnorm(length(spec_code), 40, 3))
admits <- round(rnorm(length(spec_code), 300, 20))
month <- rep(c("Jan", "Feb", "March", "April", "May", "Jun", "July", "Aug", "Sept",
               "Oct", "Nov", "Dec"),8)
# Combine these atomic vectors into tibble
ed_act_by_month <- tibble(spec_code, attends, breaches, admits, month)
glimpse(ed_act_by_month)
```

```
## Rows: 96
## Columns: 5
## $ spec_code <chr> "101", "102", "103", "104", "105", "106", "107", "108", "101~
## $ attends   <dbl> 472, 488, 578, 504, 506, 586, 523, 437, 466, 478, 561, 518, ~
## $ breaches  <dbl> 47, 45, 39, 37, 38, 41, 39, 39, 37, 40, 38, 35, 39, 43, 38, ~
## $ admits    <dbl> 302, 282, 274, 340, 312, 275, 288, 276, 344, 326, 295, 311, ~
## $ month     <chr> "Jan", "Feb", "March", "April", "May", "Jun", "July", "Aug",~
```

Using our custom data to join our lookups

Now we have some data to match, we can join our dynamic lookup from the web on to the lookup:

```
# Get our tfc and we are going to join this to specialty code
joined_activity = ed_act_by_month %>%
  dplyr::left_join(tfc, by = c("spec_code"="Code")) %>%
  dplyr::select(everything(), -c(Dict_Type, DttmExtracted))

glimpse(joined_activity)
```

```
## Rows: 96
## Columns: 6
## $ spec_code   <chr> "101", "102", "103", "104", "105", "106", "107", "108", "1~
## $ attends     <dbl> 472, 488, 578, 504, 506, 586, 523, 437, 466, 478, 561, 518~
## $ breaches    <dbl> 47, 45, 39, 37, 38, 41, 39, 39, 37, 40, 38, 35, 39, 43, 38~
## $ admits      <dbl> 302, 282, 274, 340, 312, 275, 288, 276, 344, 326, 295, 311~
## $ month       <chr> "Jan", "Feb", "March", "April", "May", "Jun", "July", "Aug~
## $ Description <chr> "Urology Service", "Transplant Surgery Service", "Breast S~
```

Practice time - 10 minutes

Try joining the main specialty code to this data

Practice solution

This is pretty simple really - I just need to use the nhs_table_findeR function to get my lookup, if you are not sure what the string is to pass, use the nhs_data_elements() function to get all the strings.

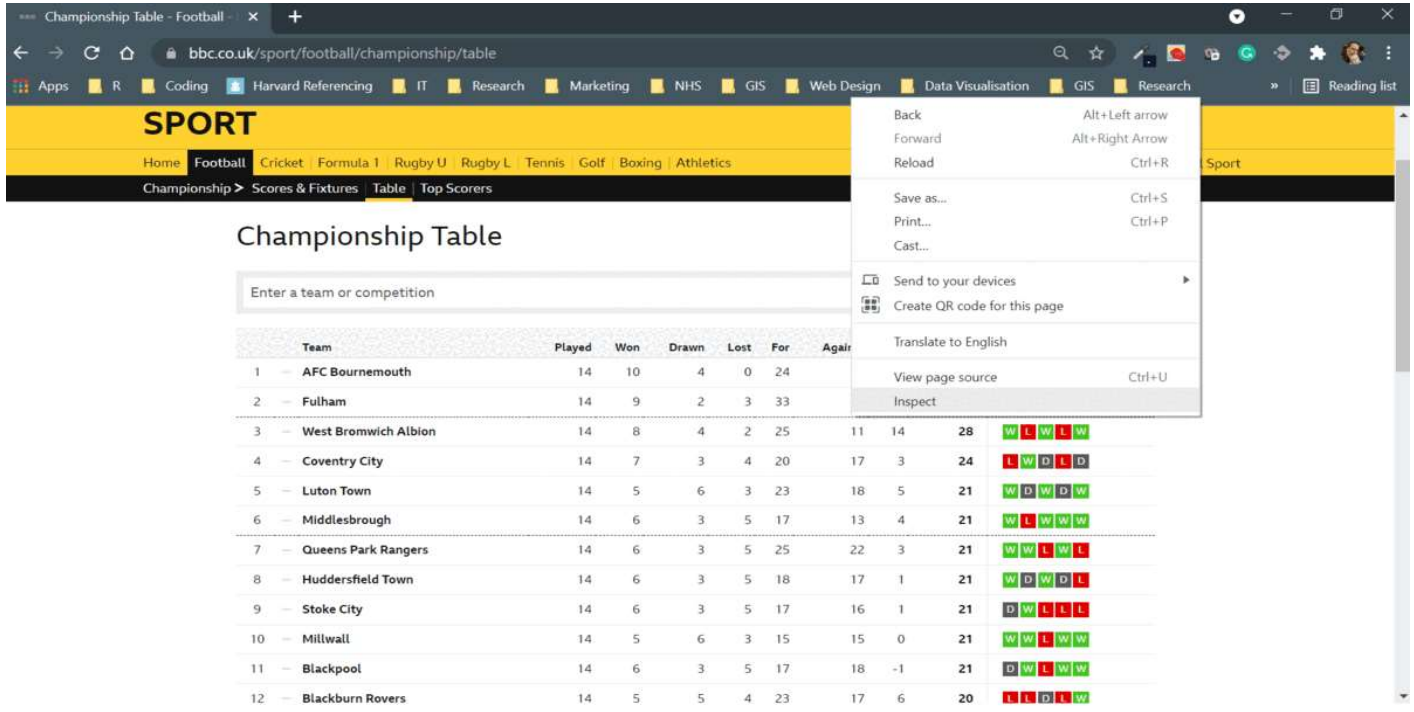
```
main_spec <- NHSDataDictionaRy::nhs_table_findeR("CARE PROFESSIONAL MAIN SPECIALTY CODE"
                                    title="Main Specialty")

# Join our main specialty on to the ed_act_by_month tibble
main_spec_with_act <- ed_act_by_month %>%
  dplyr::left_join(main_spec, by= c("spec_code"="Code")) %>%
  dplyr::select(everything(), -c(Dict_Type, DttmExtracted))

glimpse(main_spec_with_act)
```

```
## Rows: 96
## Columns: 6
## $ spec_code   <chr> "101", "102", "103", "104", "105", "106", "107", "108", "1~
## $ attends     <dbl> 472, 488, 578, 504, 506, 586, 523, 437, 466, 478, 561, 518~
## $ breaches    <dbl> 47, 45, 39, 37, 38, 41, 39, 39, 37, 40, 38, 35, 39, 43, 38~
## $ admits      <dbl> 302, 282, 274, 340, 312, 275, 288, 276, 344, 326, 295, 311~
## $ month       <chr> "Jan", "Feb", "March", "April", "May", "Jun", "July", "Aug~
## $ Description <chr> "Urology", NA, NA, NA, NA, NA, "Vascular Surgery", NA, "Ur~
```

Wider use cases of the web scraping potential

Scraping the Championship Football table

We could use the tableR function to scrape the results of the Championship table, here we will have to do some text processing as well to clean this up. I want to see how good or bad my team Nottingham Forest are doing.To get the Xpath I will need to use the Inspect button in google to copy this:



Using the retrieved Xpath from Google

The next step would be to copy the xpath and url into a variable, so we can differentiate them later.

```
football_xpath <- '//*[@id="u16945876197938725"]/div/div[2]/div/div/div[2]/div/div/table
url <- "https://www.bbc.co.uk/sport/football/championship/table"
```

Now let's get the HTML table from the website. Important to note that if it does not end in **/table** at the end of the url then it cannot be used with the TableR function, as it is not an HTML table object.

```
football_table <- NHSDataDictionaRy::tableR(url=url,
                          xpath=football_xpath,
                          title="English Championship League Standings")

head(football_table, 3)
```

```
## # A tibble: 3 x 14
##   V1    V2     Team   P     W     D     L     F     A     GD    Pts   Form
##   <chr> <chr>  <chr>  <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 1     team ~ Bourn~ 15    11    4     0     26    8     18    37    WWon 2 - ~
## 2 2     team ~ Fulham 15    10    2     3     36    14    22    32    LLost 1 -~
## 3 3     team ~ West ~ 15    8     4     3     25    14    11    28    LLost 0 -~
## # ... with 2 more variables: Dict_Type <chr>, DttmExtracted <dttm>
```

Let's clean the table up

The first stage would be to drop some of the less informative columns:

```
# Convert to integer
ftball_reduced <- football_table %>%
  dplyr::select(-c(V1, V2, Form, Dict_Type,
                   DttmExtracted)) %>% #Purge the none informative columns
  dplyr::slice(1:nrow(football_table)-1) %>% # Get rid of the last row
  dplyr::add_tally()#Filter out team

#Remove team
minus_team <- ftball_reduced %>%
  dplyr::select(!contains("Team"))

# Convert all fields to integer
minus_team <- data.frame(sapply(minus_team, as.integer))
# Bind back together

football_table <- cbind(Team=ftball_reduced$Team, minus_team)
```

Viewing the data on a scatter chart to explore key metrics

```
plot <- ggplot(football_table,
       aes(x=W, y=Pts, color=factor(Team))) + geom_point() +
  theme_minimal()

ggplotly(plot)
```

Practice time - 15 minutes

Have a go at retrieving some Xpath data from any website. Find a data table and bring it into R with the TableR function.

Scraping raw text from websites with xpathTextR function

Extracting text from the NHS Data Dictionary website

```
acc_type <- nhs_data_table %>%
  filter(link_name=='ACCOMMODATION TYPE')
#browseURL(acc_type$full_url)

# Specify the custom element obtained from the inspect function
xpath <- '//*[@id="element_accommodation_type"]/div/p'
result_list <- NHSDataDictionaRy::xpathTextR(url=acc_type$full_url,
                                xpath = xpath)

str(result_list)
```

```
## List of 6
##  $ result          : chr "ACCOMMODATION STATUS CODE will be replaced w
##  $ website_passed  : chr "https://datadictionary.nhs.uk/data_elements/accommodation_t
##  $ xpath_passed    : chr "//*[@id=\"element_accommodation_type\"]/div/p"
##  $ html_node_result:List of 2
##   ..$ node:<externalptr>
##   ..$ doc :<externalptr>
##   ..- attr(*, "class")= chr [1:2] "xml_document" "xml_node"
##  $ datetime_access : POSIXct[1:1], format: "2021-11-03 08:58:47"
##  $ person_accessed : chr "GARY.HUTSON - ASCENT"
```

Cleaning the extracted text

The next stage is to clean the extracted text from the list element **result**:

```
clean_txt <- trimws(unlist(result_list$result)) %>%
  gsub("[\r\n]", "",.)

print(clean_txt)
```

```
## [1] "ACCOMMODATION STATUS CODE will be replaced with ACCOMMODATION TYP
```

Now you could use the inbuilt string functions in the package to strip the text further:

```
# Extracted string
NHSDataDictionaRy::left_xl(clean_txt, 20)
```

```
## [1] "ACCOMMODATION STATUS"
```

Practice time - 10 minutes

Have a go at finding a component from the NHS Data Dictionary, or other website, to test out this functionality.

Scraping a list of links from a website

The NHSDataDictionaRy tool provides functionality for scraping a webpage to extract all the urls from the page. This is how it can be implemented:

```
url <- "https://datadictionary.nhs.uk/data_sets_overview.html"
results <- NHSDataDictionaRy::linkScrapeR(url, SSL_needed = FALSE)
head(results %>%
       dplyr::select(url), 10)
```

```
## # A tibble: 10 x 1
##    url
##    <chr>
##  1 #wh_topic_body
##  2 https://www.nhs.uk
##  3 index.html
##  4 about/about.html
##  5 help/help_and_guidance.html
##  6 data_sets_overview.html
##  7 data_elements_overview.html#dataElement_overview
##  8 attributes_overview.html#attribute_overview
##  9 classes_overview.html#class_overview
## 10 nhs_business_definitions_overview.html#business_definitions_overview
```

Capstone Project - Working through the whole solution

We will now utilise the tool to scrape the levels of COVID-19 by country, as I know it has been a hard year and we are getting there:

Data Preparation

```
xpath <- '//*[@id="main_table_countries_today"]'
url <- "https://www.worldometers.info/coronavirus/"

covid19 <- NHSDataDictionaRy::tableR(url=url, xpath=xpath) %>%
  select(-c("#"))

c19_reduced <- covid19 %>%
  select(TotalCases, TotalRecovered, TotalDeaths, TotalTests, NewCases, `Country,Other`,
# Map through the data frame and strip out the comman from the text and make numeric
```

Data Preparation (continued)

The data returns character strings, so I will use the first 4 columns from my selection above:

```
c19_numerics <- c19_reduced[,1:5] %>%
  purrr::map_dfr(~ as.numeric(gsub(",", "", .))) %>% #Get rid of
  purrr::map_dfr(~as.numeric(gsub("+", "", .))) %>% # Get rid of + symbol
  #Fill in NAs if they are zero
  purrr::map_dfr(~ ifelse(is.na(.),0, .))
```

```
## Warning in .f(.x[[i]], ...): NAs introduced by coercion
```

We then ***purrr::map_dfr*** over the data frame to convert to numeric and strip out the commas and then to fill in the NA values with a zero if they are missing.

Data Preparation (continued)

Finally, I will join the data back on to the original frame to bring back in the country:

```
c19_final <- cbind(
  c19_reduced <- c19_reduced %>% dplyr::select(`Country,Other`, Continent),
  c19_numerics
) %>%
  dplyr::rename(Country=`Country,Other`) %>%
  dplyr::filter(Country!="",
                !str_detect(Country, "Total"),
                Country!="World")
```

Here we:

- Used cbind base R function to combine the reduced dataset and the numerical dataframe
- We renamed the Country field
- We then filtered out the blank countries and those where there is a total column

The final dataset

This is the final dataset ready to be worked on with R:

```
print(head(c19_final,5))
```

```
##         Country     Continent TotalCases TotalRecovered TotalDeaths TotalTests
## 1 North America North America   56441733       45210670     1151451          0
## 2          Asia          Asia   79591311       76678696     1174438          0
## 3 South America South America   38442213       36577168     1171024          0
## 4        Europe        Europe   64958366       58371504     1309941          0
## 5        Africa        Africa    8581186        7920450      219105          0
##   NewCases
## 1     3687
## 2    24275
## 3      138
## 4    37922
## 5        0
```

With this dataset we will create some visuals. We could even store these in a data table with a time stamp over time to get the relevant days COVID-19 results.

Visualising the results

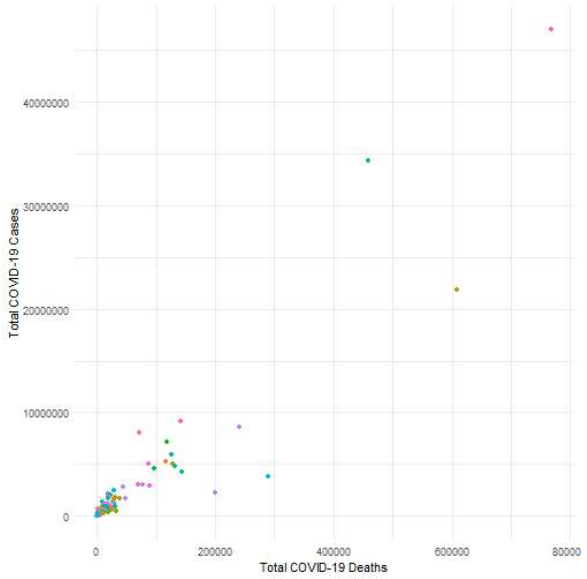
We'll use our final data frame that we generated to create some visuals of the COVID rates, perhaps a scatter chart would be good for this purpose to see the ratio of total cases to total deaths.

```
highest_cases <- c19_final %>%
  dplyr::arrange(desc(TotalCases)) %>%
  dplyr::filter(Country!=Continent)
```

On the next slide we will visualise the results:

Visualising the results

```
options(scipen=999)
ggplot(highest_cases, aes(TotalDeaths,TotalCases)) +
 geom_point(aes(color=factor(Country))) + theme_minimal() +
  theme(legend.position = "none") +
  labs(x="Total COVID-19 Deaths", y="Total COVID-19 Cases")
```



Questions?