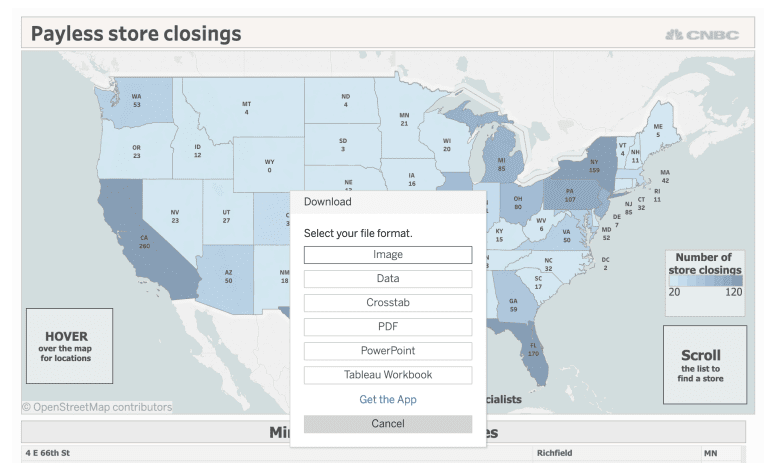
Those outside the Colonies may not know that Payless—a national chain that made footwear affordable for millions of ‘Muricans who can’t spare $100.00 USD for a pair of shoes their 7 year old will outgrow in a year— [is closing](https://www.vox.com/the-goods/2019/2/18/18229378/payless-shoesource-liqudation-stores-closing). CNBC also had [a story](https://www.cnbc.com/2019/02/19/heres-a-map-of-where-payless-shoesource-is-closing-2500-stores.html) that featured a choropleth with a tiny button at the bottom that indicated one could get the data:

[](https://rud.is/b/2019/02/20/i-just-wanted-the-data-turning-tableau-tidyverse-tears-into-smiles-with-base-r-an-encoding-detective-story/payless-get-data/)

I should have known this would turn out to be a chore since they used Tableau—the platform of choice when you want to take advantage of all the free software libraries they use to power their premier platform which, in turn, locks up all the data for you so others can’t adopt, adapt and improve. Go. Egregious. Predatory. Capitalism.

*Anyway.*

I wanted the data to do some real analysis vs produce a fairly unhelpful visualization (TLDR: layer in Census data for areas impacted, estimate job losses, compute nearest similar Payless stores to see impact on transportation-challenged homes, etc. Y’now, citizen data journalism-y things) so I pressed the button and watched for the URL in Chrome (aye, for those that remember I moved to Firefox et al in 2018, I switched back; more on that in March) and copied it to try to make this post actually reproducible (a novel concept for Tableau fanbois):

library(tibble)

library(readr)

# https://www.cnbc.com/2019/02/19/heres-a-map-of-where-payless-shoesource-is-closing-2500-stores.html

tfil <- "~/Data/Sheet\_3\_data.csv"

download.file(

"https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true",

tfil

)

## trying URL 'https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true'

## Error in download.file("https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true", :

## cannot open URL 'https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true'

## In addition: Warning message:

## In download.file("https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true", :

## cannot open URL 'https://public.tableau.com/vizql/w/PAYLESSSTORECLOSINGS/v/Dashboard2/vud/sessions/6A678928620645FF99C7EF6353426CE8-0:0/views/10625182665948828489\_7202546092381496425?csv=true&showall=true': HTTP status was '410 Gone'

WAT

Truth be told I expected a time-boxed URL of some sort (prior experience FTW). Selenium or Splash were potential alternatives but I didn’t want to research the legality of more forceful scraping (I just wanted the data) so I *manually* downloaded the file (\*the horror\*) and proceeded to read it in. Well, *try* to read it in:

read\_csv(tfil)

## Parsed with column specification:

## cols(

## A = col\_logical()

## )

## Warning: 2092 parsing failures.

## row col expected actual file

## 1 A 1/0/T/F/TRUE/FALSE '~/Data/Sheet\_3\_data.csv'

## 2 A 1/0/T/F/TRUE/FALSE '~/Data/Sheet\_3\_data.csv'

## 3 A 1/0/T/F/TRUE/FALSE '~/Data/Sheet\_3\_data.csv'

## 4 A 1/0/T/F/TRUE/FALSE '~/Data/Sheet\_3\_data.csv'

## 5 A 1/0/T/F/TRUE/FALSE '~/Data/Sheet\_3\_data.csv'

## ... ... .................. ...... .........................

## See problems(...) for more details.

##

## # A tibble: 2,090 x 1

## A

##

## 1 NA

## 2 NA

## 3 NA

## 4 NA

## 5 NA

## 6 NA

## 7 NA

## 8 NA

## 9 NA

## 10 NA

## # … with 2,080 more rows

WAT

Getting a single column back from readr::read\_[ct]sv() is (generally) a tell-tale sign that the file format is amiss. Before donning a [deerstalker](https://shop.bbc.com/deerstalker-hat-18746.html) (I just wanted the data!) I tried to just use good ol’ read.csv():

read.csv(tfil, stringsAsFactors=FALSE)

## Error in make.names(col.names, unique = TRUE) :

## invalid multibyte string at 'A'

## In addition: Warning messages:

## 1: In read.table(file = file, header = header, sep = sep, quote = quote, :

## line 1 appears to contain embedded nulls

## 2: In read.table(file = file, header = header, sep = sep, quote = quote, :

## line 2 appears to contain embedded nulls

## 3: In read.table(file = file, header = header, sep = sep, quote = quote, :

## line 3 appears to contain embedded nulls

## 4: In read.table(file = file, header = header, sep = sep, quote = quote, :

## line 4 appears to contain embedded nulls

## 5: In read.table(file = file, header = header, sep = sep, quote = quote, :

## line 5 appears to contain embedded nulls

WAT

Actually the “WAT” isn’t *really* warranted since read.csv() gave us some super-valuable info via invalid multibyte string at 'A'. FF FE is a *big* signal[1](https://en.wikipedia.org/wiki/UTF-16)[2](http://unicode.org/faq/utf_bom.html) we’re working with a file in another encoding as that’s a common “magic” sequence at the start of such files.

But, I didn’t want to delve into my Columbo persona… *I. Just. Wanted. The. Data*. So, I tried the mind-bendingly fast and flexible helper from data.table:

data.table::fread(tfil)

## Error in data.table::fread(tfil) :

## File is encoded in UTF-16, this encoding is not supported by fread(). Please recode the file to UTF-8.

*AHA*. UTF-16 (*maybe*). Let’s poke at the raw file:

x <- readBin(tfil, "raw", file.size(tfil)) ## also: read\_file\_raw(tfil)

x[1:100]

## [1] ff fe 41 00 64 00 64 00 72 00 65 00 73 00 73 00 09 00 43 00

## [21] 69 00 74 00 79 00 09 00 43 00 6f 00 75 00 6e 00 74 00 72 00

## [41] 79 00 09 00 49 00 6e 00 64 00 65 00 78 00 09 00 4c 00 61 00

## [61] 62 00 65 00 6c 00 09 00 4c 00 61 00 74 00 69 00 74 00 75 00

## [81] 64 00 65 00 09 00 4c 00 6f 00 6e 00 67 00 69 00 74 00 75 00

There’s our ff fe (which is the beginning of the possibility it’s UTF-16) but that 41 00 harkens back to UTF-16’s older sibling UCS-2. The 0x00‘s are embedded nuls (likely to get bytes aligned). And, there are *alot* of 09s. Y’know what *they* are? They’re s. That’s right. Tableau named file full of TSV records in an unnecessary elaborate encoding CSV. Perhaps they broke the “T” on all their keyboards typing their product name so much.

**Living A Boy’s [Data] Adventure Tale**

At this point we have:

* no way to support an automated, reproducible workflow
* an ill-named file for what it contains
* an overly-encoded file for what it contains
* many wasted minutes (which is likely by design to have us give up and just use Tableau. No. Way.)

At this point I’m in full-on Rockford Files (pun intended) mode and delved down to the command line to use a old, trusted sidekick [enca[](https://github.com/nijel/enca/)](https://github.com/nijel/enca/):

$ enca -L none Sheet\_3\_data.csv

## Universal character set 2 bytes; UCS-2; BMP

## LF line terminators

## Byte order reversed in pairs (1,2 -> 2,1)

Now, all we have to do is specify the encoding!

read\_tsv(tfil, locale = locale(encoding = "UCS-2LE"))

## Error in guess\_header\_(datasource, tokenizer, locale) :

## Incomplete multibyte sequence

WAT

Unlike the other 99% of the time (mebbe 99.9%) you use it, the tidyverse doesn’t have your back in this situation (but it *does* have your [backlog](https://github.com/tidyverse/readr/milestone/5) in that it’s on the TODO).

Y’know who *does* have your back? Base R!:

read.csv(tfil, sep="\t", fileEncoding = "UCS-2LE", stringsAsFactors=FALSE) %>%

as\_tibble()

## # A tibble: 2,089 x 14

## Address City Country Index Label Latitude Longitude

##

## 1 1627 O… Aubu… United… 1 Payl… 32.6 -85.4

## 2 900 Co… Doth… United… 2 Payl… 31.3 -85.4

## 3 301 Co… Flor… United… 3 Payl… 34.8 -87.6

## 4 304 Ox… Home… United… 4 Payl… 33.5 -86.8

## 5 2000 R… Hoov… United… 5 Payl… 33.4 -86.8

## 6 6140 U… Hunt… United… 6 Payl… 34.7 -86.7

## 7 312 Sc… Mobi… United… 7 Payl… 30.7 -88.2

## 8 3402 B… Mobi… United… 8 Payl… 30.7 -88.1

## 9 5300 H… Mobi… United… 9 Payl… 30.6 -88.2

## 10 6641 A… Mont… United… 10 Payl… 32.4 -86.2

## # … with 2,079 more rows, and 7 more variables:

## # Number.of.Records , State , Store.Number ,

## # Store.count , Zip.code , State.Usps ,

## # statename

~~WAT~~ WOOT!

Note that read.csv(tfil, sep="\t", fileEncoding = "UTF-16LE", stringsAsFactors=FALSE) would have worked equally as well.

**The Road Not [Originally] Taken**

Since this activity decimated productivity, for giggles I turned to another trusted R sidekick, the stringi package, to see what it said:

library(stringi)

stri\_enc\_detect(x)

## [[1]]

## Encoding Language Confidence

## 1 UTF-16LE 1.00

## 2 ISO-8859-1 pt 0.61

## 3 ISO-8859-2 cs 0.39

## 4 UTF-16BE 0.10

## 5 Shift\_JIS ja 0.10

## 6 GB18030 zh 0.10

## 7 EUC-JP ja 0.10

## 8 EUC-KR ko 0.10

## 9 Big5 zh 0.10

## 10 ISO-8859-9 tr 0.01

And, just so it’s primed in the Google caches for future searchers, another way to get this data (and other data that’s even gnarlier but similar in form) into R would have been:

stri\_read\_lines(tfil) %>%

paste0(collapse="\n") %>%

read.csv(text=., sep="\t", stringsAsFactors=FALSE) %>%

as\_tibble()

## # A tibble: 2,089 x 14

## Address City Country Index Label Latitude Longitude

##

## 1 1627 O… Aubu… United… 1 Payl… 32.6 -85.4

## 2 900 Co… Doth… United… 2 Payl… 31.3 -85.4

## 3 301 Co… Flor… United… 3 Payl… 34.8 -87.6

## 4 304 Ox… Home… United… 4 Payl… 33.5 -86.8

## 5 2000 R… Hoov… United… 5 Payl… 33.4 -86.8

## 6 6140 U… Hunt… United… 6 Payl… 34.7 -86.7

## 7 312 Sc… Mobi… United… 7 Payl… 30.7 -88.2

## 8 3402 B… Mobi… United… 8 Payl… 30.7 -88.1

## 9 5300 H… Mobi… United… 9 Payl… 30.6 -88.2

## 10 6641 A… Mont… United… 10 Payl… 32.4 -86.2

## # … with 2,079 more rows, and 7 more variables: `Number of

## # Records` , State , `Store Number` , `Store

## # count` , `Zip code` , `State Usps` ,

## # statename

(with similar dances to use read\_csv() or fread()).

**FIN**

The night’s quest to do some real work with the data was DoS’d by what I’ll brazenly call a deliberate attempt to dissuade doing exactly that in anything but a commercial program. But, understanding the impact of yet-another massive retail store closing is super-important and it looks like it may be up to us (since the media is too distracted by incompetent leaders and inexperienced junior NY representatives) to do the work.

Folks who’d like to do the same can grab the UTF-8 encoded actual CSV from [this site](http://rud.is/dl/2019-payless-store-closings.csv) which has also been run through janitor::clean\_names() so there’s proper column types and names to work with.

Speaking of which, here’s the cols spec for that CSV:

cols(

address = col\_character(),

city = col\_character(),

country = col\_character(),

index = col\_double(),

label = col\_character(),

latitude = col\_double(),

longitude = col\_double(),

number\_of\_records = col\_double(),

state = col\_character(),

store\_number = col\_double(),

store\_count = col\_double(),

zip\_code = col\_character(),

state\_usps = col\_character(),

statename = col\_character()

)

If you do anything with the data blog about it and post a link in the comments so I and others can learn from what you’ve discovered! It’s already kinda scary that one doesn’t even need a basemap to see just how much apart of ‘Murica Payless was:

[](https://rud.is/b/2019/02/20/i-just-wanted-the-data-turning-tableau-tidyverse-tears-into-smiles-with-base-r-an-encoding-detective-story/payless-map-01/)