Precision of detecting identifying information

# Introduction

In this document, we investigate the precision of using regular expressions to identify potentially personal information in datasets. This document is preceeded by an investigation of privacy issues in open datasets ([Wicherts et al., 2022](https://doi.org/10.31234/osf.io/ybzu9)), where results indicated 1/20 open datasets contain potentially identifying information. To further promote the use of responsible open data, this report evaluates the use of automated detection tools to flag potential identifying information before it is shared.

The purpose of automated detection of identifying information is that the information can be removed or modified prior to public sharing. This is the technical precursor to more user-friendly implementations that allow for easy and large scale checking of datasets. The goal is to reduce the risk to research participants’ data, and by extension, reduce the liability of researchers and their institutions.

# Setup

### Data

# Download files if not available from filesystem  
# Download synthetic data with personal information  
if (!file.exists("synthetic-data.csv"))   
 download.file("https://www.researchequals.com/api/modules/main/wz7c-s5wc",  
 destfile = "synthetic-data.csv")  
# Download data without personal information  
if (!file.exists("real-data.csv"))   
 download.file("https://www.researchequals.com/api/modules/main/g9j4-v2gy",  
 destfile = "real-data.csv")  
if (!file.exists("summary-row-cols.csv"))  
 download.file(  
 "https://www.researchequals.com/api/modules/supporting/g9j4-v2gy/summary-rows-cols.csv",   
 destfile = "summary-row-cols.csv")  
  
principalSynthetic <- read.csv('synthetic-data.csv')  
principalSynthetic$LatLong <- sprintf('%s, %s', principalSynthetic$Latitude,  
 principalSynthetic$Longitude)  
principalSynthetic$phoneNr <- sprintf('+%s%s', principalSynthetic$TelephoneCountryCode,  
 principalSynthetic$TelephoneNumber)  
principalReal <- readLines('real-data.csv')

We reuse previously collected datasets to generate pseudo-samples of data with or without identifying information. We collected a set of real data without identifying information (x rows) based on open datasets from published articles. We also generated synthetic data (100,000 rows) for data with identifying information, based on the sorts of identifying data detected in real datasets in Wicherts et al., 2022, and other foreseeable privacy risks. These serve as the main datasets from which we generate pseudo-samples.

infoDat <- read.csv('summary-row-cols.csv')  
# Select out the ones without rows/columns  
infoSel <- infoDat[!infoDat$rows == 0 | !infoDat$columns == 0,]  
rowsP25 <- summary(infoSel$rows)[2]  
rowsP50 <- summary(infoSel$rows)[3]  
rowsP75 <- summary(infoSel$rows)[5]  
columnsP25 <- summary(infoSel$columns)[2]  
columnsP50 <- summary(infoSel$columns)[3]  
columnsP75 <- summary(infoSel$columns)[5]

The pseudo-samples are drawn in a two factorial design resulting in three conditions. Each pseudo-sample consists of a certain number of rows, based on the 25th, 50th, and 75th percentile of the open datasets we previously collected (i.e., 56, 102, and 357 respectively). Similarly, each pseudo-sample consists of a certain number of columns, based on the 25th, 50th, and 75th percentile of the open datasets (i.e., 8, 14, and 23 respectively). For simplicity of reporting, we only consider the concordant pairs (e.g., 25th percentile for rows and columns).

The general procedure for creating a pseudo-sample is as follows:

1. Randomly sample number of rows (based on P25, P50, P75; with replacement)
2. Randomly sample number of columns (based on P25, P50, P75; with replacement)

Given the structure of the data we created the following function to make it simpler to create these pseudo-samples:

pseudoSampler <- function (dataframe, rows, cols, messy = TRUE) {  
 if (messy) {  
 rowSampler <- sample(1:length(dataframe), size = rows, replace = TRUE)  
 sampledData <- ""  
   
 for (row in rowSampler) {  
 if (dataframe[row] != "") {  
 replaceString <- str\_replace\_all(  
 dataframe[row],   
 pattern = "[^a-zA-Z0-9 ,]",   
 replacement = "")  
   
 readRow <- read.csv(text = replaceString, header = FALSE)  
   
 colSampler <- sample(1:dim(readRow)[2], size = cols, replace = TRUE)  
 df <- readRow[1:dim(readRow)[1], colSampler]  
 names(df) <- sprintf('col\_%s', 1:cols)  
 sampledData <- rbind(sampledData, df)  
 names(sampledData) <- sprintf('col\_%s', 1:cols)  
 }  
 }  
   
 return(sampledData)  
 } else if (!messy) {  
 rowSampler <- sample(1:dim(dataframe)[1], size = rowsP25, replace = TRUE)  
 # colSampler <- sample(1:dim(dataframe)[2], size = columnsP25, replace = TRUE)  
   
 return(dataframe[rowSampler,])  
 }  
}

These can be run at later timepoints in a rather straightforward manner, like so:

pseudoSampler(principalSynthetic, rows = rowsP25, cols = columnsP25, messy = FALSE)  
## OR  
pseudoSampler(principalReal, rows = rowsP25, cols = columnsP25, messy = TRUE)

### Regular expressions

We use regular expressions to find potentially personal information in datasets. In the section above we explained how we generated pseudosamples containing (no) personal information; in this section we detail the the regular expressions used.

## Loading required package: stringr

For all applied regular expressions, we use the package stringr. Please note that R can have its own quirks regarding regular expressions, such that what would usually be ^.\*@.\*\..\*$ needs to be adjusted to escape the \ (i.e., R regex: ^.\*@.\*\\..\*$). Some regular expressions are highly specific and become rather lengthy as a result.

We cover all the regular expressions per category below. The categories included are:

1. Technical identifiers
2. Geographical identifiers
3. Direct identifiers

#### Technical identifiers

Technical identifiers are the following:

1. Email address
2. IP address (version 4, e.g., 192.0.0.1:8080, and 6, e.g., [2031:0000:130f:0000:0000:09c0:876a:130b]:8080)
3. Hardware device ID (MAC address, e.g., 38:f9:d3:4b:f5:51, IMEI, e.g., 12345678901234567)
4. Phone number ([E.164 standard](https://en.wikipedia.org/wiki/E.164), e.g., +1234567891011111)
5. Browser User Agent (e.g., Chrome/76.0.3809.100)

These are commonly used technical identifiers that can be used to uniquely identify a person. Details of the exact regular expressions can be found in the code below.

## Email  
## https://regexpattern.com/email-address/  
email <- "^.\*@.\*\\..\*$"  
  
## IPv4  
## https://regexpattern.com/ip-address-v4-port/  
ipv4 <- "^((\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])\\.){3}(\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])(?::(?:[0-9]|[1-9][0-9]{1,3}|[1-5][0-9]{4}|6[0-4][0-9]{3}|65[0-4][0-9]{2}|655[0-2][0-9]|6553[0-5]))?$"  
  
## IPv6  
## https://regexpattern.com/ip-address-v6-port/  
ipv6 <- "^(?:(?:(?:[0-9A-Fa-f]{1,4}:){7}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}:[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){5}:([0-9A-Fa-f]{1,4}:)?[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){4}:([0-9A-Fa-f]{1,4}:){0,2}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){3}:([0-9A-Fa-f]{1,4}:){0,3}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){2}:([0-9A-Fa-f]{1,4}:){0,4}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|(([0-9A-Fa-f]{1,4}:){0,5}:((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|(::([0-9A-Fa-f]{1,4}:){0,5}((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|([0-9A-Fa-f]{1,4}::([0-9A-Fa-f]{1,4}:){0,5}[0-9A-Fa-f]{1,4})|(::([0-9A-Fa-f]{1,4}:){0,6}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){1,7}:))|\\[(?:(?:(?:[0-9A-Fa-f]{1,4}:){7}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}:[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){5}:([0-9A-Fa-f]{1,4}:)?[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){4}:([0-9A-Fa-f]{1,4}:){0,2}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){3}:([0-9A-Fa-f]{1,4}:){0,3}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){2}:([0-9A-Fa-f]{1,4}:){0,4}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){6}((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|(([0-9A-Fa-f]{1,4}:){0,5}:((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|(::([0-9A-Fa-f]{1,4}:){0,5}((\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b)\\.){3}(\\b((25[0-5])|(1\\d{2})|(2[0-4]\\d)|(\\d{1,2}))\\b))|([0-9A-Fa-f]{1,4}::([0-9A-Fa-f]{1,4}:){0,5}[0-9A-Fa-f]{1,4})|(::([0-9A-Fa-f]{1,4}:){0,6}[0-9A-Fa-f]{1,4})|(([0-9A-Fa-f]{1,4}:){1,7}:))\\](?::(?:[0-9]|[1-9][0-9]{1,3}|[1-5][0-9]{4}|6[0-4][0-9]{3}|65[0-4][0-9]{2}|655[0-2][0-9]|6553[0-5]))?$"  
  
## MAC address  
## https://regexpattern.com/mac-address/  
macAddress <- "^((([a-f0-9]{2}:){5})|(([a-f0-9]{2}-){5}))[a-f0-9]{2}$"  
  
## IMEI  
## https://regexpattern.com/international-mobile-equipment-identity/  
# AA-BBBBBB-CCCCCC-D  
# AA-BBBBBB-CCCCCC-EE  
#   
# \\d{2}-?\\d{6}-?\\d{6}-?\\d{1}  
imei <- "^\\d{2}-?\\d{6}-?\\d{6}-?\\d{1,2}$"  
  
## Phone number  
## https://ihateregex.io/expr/phone/  
# phoneNr <- "^[\\+]?[(]?[0-9]{3}[)]?[-\\s\\.]?[0-9]{3}[-\\s\\.]?[0-9]{4,6}$"  
phoneNr <- "\\+\\d{1,4}?[-.\\s]?\\(?\\d{1,3}?\\)?[-.\\s]?\\d{1,4}[-.\\s]?\\d{1,4}[-.\\s]?\\d{1,9}$"  
  
## Browser user agent  
## https://regex101.com/r/2McsiK/1  
browserUA <- "(firefox|msie|chrome|safari)[/\\s]([\\d.]+)"

#### Geographical identifiers

We only use country and latitude-longitude for identifying geographical information. We initially identified city and country as potential options but this is not feasible in the scope of the current project. This would require creating a list of relevant options for an extensive regular expression.

These are commonly used geographical identifiers that may be used to identify someone in a dataset. Details of the exact regular expressions can be found in the code below.

## street Address  
streetAddress <- "((\\w+\\s){1,3}\\d+)|(\\d+(\\w+\\s){1,3})"  
## Latitude / longitude  
## https://regexpattern.com/latitude-longitude/  
latitudeLongitude <- "^((\\-?|\\+?)?\\d+(\\.\\d+)?),\\s\*((\\-?|\\+?)?\\d+(\\.\\d+)?)$"

#### Direct identifiers

As identifiers of direct risk to the identification of individuals we included:

1. Gender
2. Social security number
3. Birthday (<2015; format 18/08/1999 and Aug 18 1999)
4. Bloodtype
5. IBAN bank number (e.g., NL1234563234234212345)
6. Creditcard number (e.g., 424242424242)
7. MTurk numbers (e.g., A123456asdfgh123)

## Gender  
## https://regexlib.com/REDetails.aspx?regexp\_id=2441  
gender <- "^(transgender|two-spirit|non[-\\s]?binary|cisgender|gender[-\\s]?fluid|trans)$"  
  
## SSN  
## https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s12.html  
ssn <- "^(?!000|666)[0-8][0-9]{2}-(?!00)[0-9]{2}-(?!0000)[0-9]{4}$"  
  
## Birthdays  
birthday <- "^([0123]?[0-9][-/][0123]?[0-9][-/][0-9]{2,4})|([0-9]{2,4}[-./][0123]?[0-9][-/][0123]?[0-9])|((jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec).\*?\\s\\d+.\*?\\s((([19]{2})?([1-9]{2}))|((20)([01][0-5]))))$"  
  
## Blood type  
## https://www.regexlib.com/REDetails.aspx?regexp\_id=2671  
bloodType <- "(a|b|ab|o)[+-]"  
  
# Financials  
## IBAN  
## http://blog.marketto.it/en/2018/06/validate-any-country-iban/  
## https://regex101.com/r/mSqBqC/1  
iban <- "^(?:(?:it|sm)\\d{2}[a-z]\\d{22}|cy\\d{2}[a-z]\\d{23}|nl\\d{2}[a-z]{4}\\d{10}|lv\\d{2}[a-z]{4}\\d{13}|(?:bg|bh|gb|ie)\\d{2}[a-z]{4}\\d{14}|gi\\d{2}[a-z]{4}\\d{15}|ro\\d{2}[a-z]{4}\\d{16}|kw\\d{2}[a-z]{4}\\d{22}|mt\\d{2}[a-z]{4}\\d{23}|no\\d{13}|(?:dk|fi|gl|fo)\\d{16}|mk\\d{17}|(?:at|ee|kz|lu|xk)\\d{18}|(?:ba|hr|li|ch|cr)\\d{19}|(?:ge|de|lt|me|rs)\\d{20}|il\\d{21}|(?:ad|cz|es|md|sa)\\d{22}|pt\\d{23}|(?:be|is)\\d{24}|(?:fr|mr|mc)\\d{25}|(?:al|do|lb|pl)\\d{26}|(?:az|hu)\\d{27}|(?:gr|mu)\\d{28})$"  
  
## Creditcard  
## https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s20.html  
# creditcard <- "^(?:(4[0-9]{12}(?:[0-9]{3})?)|(5[1-5][0-9]{14})|(6(?:011|5[0-9]{2})[0-9]{12})|(3[47][0-9]{13})|(3(?:0[0-5]|[68][0-9])[0-9]{11})|((?:2131|1800|35[0-9]{3})[0-9]{11}))$"  
# creditcard <- "(^4[0-9]{12}(?:[0-9]{3})?$)|(^(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}$)|(3[47][0-9]{13})|(^3(?:0[0-5]|[68][0-9])[0-9]{11}$)|(^6(?:011|5[0-9]{2})[0-9]{12}$)|(^(?:2131|1800|35\\d{3})\\d{11}$)"  
  
creditcard <- "^(?:4[0-9]{12}(?:[0-9]{3})?|(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|6(?:011|5[0-9]{2})[0-9]{12}| (?:2131|1800|35\\d{3})\\d{11})$"  
  
## MTurk  
mturk <- "^a[a-zA-Z0-9]{12,15}$"

## Precision

In this section we present the precision results for the different regular expression per category. This means that we display the precision of for example gender as a direct identifier, IPv4 as a technical identifier. This allows us to identify potential weak points in the identification process.

### Technical identifiers

for (currentRegex in c("email", "ipv4", "ipv6", "macAddress", "imei", "phoneNr", "browserUA",  
 "streetAddress", "latitudeLongitude",  
 "gender", "ssn", "birthday", "bloodType", "iban", "creditcard", "mturk")){  
 for (p in c(25, 50, 75)) {  
 # Scaffolding  
 absentObject <- sprintf("absent\_%s\_identified\_p%s", currentRegex, p)  
 presentObject <- sprintf("present\_%s\_identified\_p%s", currentRegex, p)  
 assign(absentObject, c())  
 assign(presentObject, c())  
   
 for (i in 1:iter) {  
 pseudoSynthetic <- pseudoSampler(  
 principalSynthetic,   
 rows = get(sprintf("rowsP%s", p)),  
 cols = get(sprintf("columnsP%s", p)),  
 messy = FALSE)  
   
 pseudoReal <- pseudoSampler(  
 principalReal,   
 rows = get(sprintf("rowsP%s", p)),  
 cols = get(sprintf("columnsP%s", p)),  
 messy = TRUE)  
   
 # Always present is NA  
 # Length > 1 to check whether there are any other matches than NA  
 absentIdentified <- length(unique(str\_extract(tolower(unlist(pseudoReal)),   
 pattern = get(currentRegex)))) > 1  
 presentIdentified <- length(unique(str\_extract(tolower(unlist(pseudoSynthetic)),   
 pattern = get(currentRegex)))) > 1  
   
 # Write out result  
 assign(absentObject, c(  
 get(absentObject), absentIdentified  
 ))  
 assign(presentObject, c(  
 get(presentObject),   
 presentIdentified  
 ))  
   
 }  
 }  
}

#### Email

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### IPv4

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### IPv6

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### MAC Address

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### IMEI

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 0.145 | 0.012 | 0 | 0 |
| Yes | 0.855 | 0.988 | 1 | 1 |

#### Phone number

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### Browser User Agent

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

### Geographical identifiers

#### Street address

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 0.132 | 0.01 | 0 | 0 |
| Yes | 0.868 | 0.99 | 1 | 1 |

#### Latitude/longitude

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

### Direct identifiers

#### Gender

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0.015 |
| Yes | 0 | 0 | 0 | 0.985 |

#### Social Security Number

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0.177 |
| Yes | 0 | 0 | 0 | 0.823 |

#### Birthday

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 0.997 | 0.987 | 0.965 | 0 |
| Yes | 0.003 | 0.013 | 0.035 | 1 |

#### Blood type

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### IBAN

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 1 | 1 | 1 | 0 |
| Yes | 0 | 0 | 0 | 1 |

#### Credit card

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 0.944 | 0.843 | 0.484 | 0 |
| Yes | 0.056 | 0.157 | 0.516 | 1 |

#### MTurk ID

| Detected? | Absent (P25) | Absent (P50) | Absent (P75) | Present |
| --- | --- | --- | --- | --- |
| No | 0.958 | 0.915 | 0.707 | 0 |
| Yes | 0.042 | 0.085 | 0.293 | 1 |

# Executive summary

In this precision report, we provide information on the specificity and sensitivity of using regular expressions to retrieve identifying information. Overall, technical identifiers are highly specific and sensitive (e.g., email, IP addresses), except for IMEI numbers (hardware numbers for phones). Phone numbers can be detected if reported in their international form (e.g., +1 555 55 555). For location information, regular expressions are highly precise in the case of latitude/longitude combinations, but not for street addresses (sensitive but not specific). For direct identifiers, we see that gender is hardest to detect with high specificity, but given the risk of disclosing marginalized gender identities, we consider this important to check for nonetheless (specificity gets worse as the dataset is larger in size). Similar issues exist for credit card information, where the risk is high if disclosed. As a result, we recommend scanning for all identifying information looked into in this report, except for IMEI and street addresses, as they underperform.

# Environment info

sessionInfo()

## R version 4.2.1 (2022-06-23)  
## Platform: x86\_64-apple-darwin21.5.0 (64-bit)  
## Running under: macOS Ventura 13.0.1  
##   
## Matrix products: default  
## BLAS: /usr/local/Cellar/openblas/0.3.20/lib/libopenblasp-r0.3.20.dylib  
## LAPACK: /usr/local/Cellar/r/4.2.1/lib/R/lib/libRlapack.dylib  
##   
## locale:  
## [1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8  
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] stringr\_1.4.1  
##   
## loaded via a namespace (and not attached):  
## [1] compiler\_4.2.1 magrittr\_2.0.3 fastmap\_1.1.0 cli\_3.4.1   
## [5] tools\_4.2.1 htmltools\_0.5.3 rstudioapi\_0.14 yaml\_2.3.5   
## [9] stringi\_1.7.8 rmarkdown\_2.17 knitr\_1.40 xfun\_0.31   
## [13] digest\_0.6.29 rlang\_1.0.4 evaluate\_0.15