7 How to anonymize columns

ℹ What will this tutorial cover?

In this tutorial we will cover six different techniques for anonymizing columns. We will find out how to use the function `fct_anon`, how to replace names with random names, how to mask values, how to group numeric variables, how to remove house numbers from street names, and how to encode and decode values.

Who do I have to thank?

For this course, I had to do some reading about anonymization. In particular, I relied on the following blog posts:

- https://www.record-evolution.de/en/blog/data-anonymization-techniques-and-best-practices-a-quick-guide/
- https://satoricyber.com/data-masking/data-anonymization-use-cases-and-6-common-techniques/
- https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

No one wants to make their bank account password or home address public. When such data breaches occur, it is costly and potentially dangerous for those affected. For example in 2020, private data of Finnish people in psychological care was hacked and published. It even went so far that the hackers forced the victims to pay money to prevent their personal data from being disclosed.

To avoid such data breaches, we as users of R need to think twice about how we handle sensitive data. Therefore, in this tutorial we will cover some techniques to anonymize or pseudoanonymize your data. We will cover these topics:

- Anonymization of a factor column with `fct_anon`
- How to replace names with random names
- Masking values in a column
- Creating groups of numeric variables (e.g. age)
- Removing house numbers from street names
- Encrypting and decrypting columns with encryptr

What is the difference between pseudonymization and anonymization?

Before we begin, we need to talk about pseudonymization and anonymization. The following techniques can be used for pseudoanonymization and anonymization. The difference between the two is that pseudonymization is reversible, while anonymization is not (at least with current technical capabilities). All of the following techniques allow pseudonymization. Not all of them are suitable for anonymization (especially encryption).

The EU defines pseudonymization as follows:

> "the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information provided that such additional information is kept separately and is subject to technical and organisational measures to ensure that the personal data are not attributed to an identified or identifiable natural person." (https://edps.europa.eu/system/files/2021-04/21-04-27_aepd-edps_anonymisation_en_5.pdf)

By this definition, pseudonymization is reversible and requires additional information to reverse the process. Thus, for the following techniques, whether your data is anomyzed or pseudoanonymized depends on your actions.

7.1 How to anonymize a factor column with `fct_anon`

Sometimes you want to make your data completely anonymous so that other people can't see sensitive information. For example, take a person's religion or gender. A simple function to anonymize such discrete data is fct_anon. The function takes two arguments. The factor you want to anonymize, and the prefix you put in front of the anonymized factor. Suppose we want to anonymize the factor levels of the `relig` column in the `gss_cat` data frame:

```r
levels(gss_cat$relig)
```

```
 [1] "No answer"               "Don't know"             
 [3] "Inter-nondenominational" "Native american"        
 [5] "Christian"               "Orthodox-christian"     
 [7] "Moslem/islam"            "Other eastern"          
 [9] "Hinduism"                "Buddhism"               
[11] "Other"                   "None"                   
[13] "Jewish"                  "Catholic"               
[15] "Protestant"              "Not applicable"         
```

With `fct_anon` we can convert these levels into numeric values and add a prefix to them:

```
gss_cat %>%
  mutate(
    relig = fct_anon(relig, prefix = "religion_")
  ) %>%
  glimpse()
```

```
Rows: 21,483
Columns: 9
$ year    <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 20~
$ marital <fct> Never married, Divorced, Widowed, Never married, Divorced, Mar~
$ age     <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51, 52, 40~
$ race    <fct> White, White, White, White, White, White, White, White, White,~
$ rincome <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not applicable, ~
$ partyid <fct> "Ind,near rep", "Not str republican", "Independent", "Ind,near~
$ relig   <fct> religion_02, religion_02, religion_02, religion_15, religion_1~
$ denom   <fct> "Southern baptist", "Baptist-dk which", "No denomination", "No~
$ tvhours <int> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, 7, NA, 3, 3~
```

Note that the numbers are generated randomly. So, each time you run this code, you will get a different set of numbers. If you need to use a factor as an id column, it is better to use other techniques that preserve the levels as unique identifiers (for example, hashing).

How to replace names with random names

Names are also sensitive data. To anonymize names, you can simply replace them with random names. This can be done with the randomNames function from the `randomNames` package:

```
library(randomNames)

presidential %>%
  mutate(
    name = randomNames(nrow(.),
                       sample.with.replacement = FALSE)
  )
```

```
# A tibble: 11 x 4
   name                 start      end        party
   <chr>                <date>     <date>     <chr>
```

```
 1 al-Shaheen, Muammar 1953-01-20 1961-01-20 Republican
 2 Stevenson, Daniel   1961-01-20 1963-11-22 Democratic
 3 Salas, Zachary      1963-11-22 1969-01-20 Democratic
 4 Burton, Charles     1969-01-20 1974-08-09 Republican
 5 Nguyen, Joanne      1974-08-09 1977-01-20 Republican
 6 Martinez, Gabrielle 1977-01-20 1981-01-20 Democratic
 7 Whitetree, Sarina   1981-01-20 1989-01-20 Republican
 8 Whittington, Ashtyn 1989-01-20 1993-01-20 Republican
 9 Thompson, Brandon   1993-01-20 2001-01-20 Democratic
10 Velasquez, Veronica 2001-01-20 2009-01-20 Republican
11 Ahsan, Michael      2009-01-20 2017-01-20 Democratic
```

In this example we have overwritten the column name with a list of random names and told the function that no name should occur more than once (`sample.with.replacement = FALSE`). Again, you get a different set of names each time you run the function.

If we want to be more specific about how the names are generated, we can provide some additional information to the function. For example, we can specify the ethnicity of the generated names, the order of their first and last names, and a separator that separates the first name from the last:

```
presidential %>%
  mutate(
    name = randomNames(nrow(.),
                       sample.with.replacement = FALSE,
                       ethnicity = c(1, 2, 4),
                       name.order = "first.last",
                       name.sep = " ")
  )
```

```
# A tibble: 11 x 4
   name            start      end        party
   <chr>           <date>     <date>     <chr>
 1 Atsila Yates    1953-01-20 1961-01-20 Republican
 2 Yu Jin Sung     1961-01-20 1963-11-22 Democratic
 3 Blanca Manchego 1963-11-22 1969-01-20 Democratic
 4 William Duran   1969-01-20 1974-08-09 Republican
 5 Grace Lam       1974-08-09 1977-01-20 Republican
 6 Desirae Vasquez 1977-01-20 1981-01-20 Democratic
 7 Ashley Palmier  1981-01-20 1989-01-20 Republican
 8 Byunghyun Wang  1989-01-20 1993-01-20 Republican
 9 Gerardo Meza    1993-01-20 2001-01-20 Democratic
```

```
10 Matthew Zamora  2001-01-20 2009-01-20 Republican
11 Toshal Ngo      2009-01-20 2017-01-20 Democratic
```

If you are wondering what the names in the ethnicity vector mean, take a look at the documentation. Just hit `?randomNames` in your console and scroll down to ethnicity. For example, 1 stands for "American Indian or Native Alaskan".

How to mask values (e.g. credit card numbers)

Another common use case is the masking of values. Credit card numbers are a prime example. You certainly don't want to reveal a person's credit card number. Masking is a technique that hides some characters of a string. Mostly by "X"s.

Suppose we want to mask the last digit of the height of Starwars characters. This is how we would do it in Tidyverse:

```
starwars %>%
  mutate(
    height = map_chr(height, ~ str_replace(.x, ".$", "X"))
  )
```

```
# A tibble: 87 x 14
   name     height  mass hair_color skin_color eye_color birth_year sex   gender
   <chr>    <chr>  <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
 1 Luke Sk~ 17X       77 blond      fair       blue            19   male  mascu~
 2 C-3PO    16X       75 <NA>       gold       yellow         112   none  mascu~
 3 R2-D2    9X        32 <NA>       white, bl~ red             33   none  mascu~
 4 Darth V~ 20X      136 none       white      yellow          41.9 male  mascu~
 5 Leia Or~ 15X       49 brown      light      brown           19   fema~ femin~
 6 Owen La~ 17X      120 brown, gr~ light      blue            52   male  mascu~
 7 Beru Wh~ 16X       75 brown      light      blue            47   fema~ femin~
 8 R5-D4    9X        32 <NA>       white, red red             NA   none  mascu~
 9 Biggs D~ 18X       84 black      light      brown           24   male  mascu~
10 Obi-Wan~ 18X       77 auburn, w~ fair       blue-gray       57   male  mascu~
# ... with 77 more rows, and 5 more variables: homeworld <chr>, species <chr>,
#   films <list>, vehicles <list>, starships <list>
```

The trick here lies in the `str_replace` function. The `.x` stands for the piped variable (in this case `height`). Then I provide a regular expression that searches for the last character of the string (`.$`). This character should then be replaced by an `X`.

A more complicated example arises when we want to mask more than one character. So let's return to our credit card example. Here is a data frame with credit card numbers:

```
ccards <- tibble(
  creditcards = c(
    36555224524299,
    36350489667466,
    36002887965170,
    5447552069207504,
    2221002654361034,
    5127699386148536)
)
```

Let's convert the first 10 characters of these credit card numbers to "X"s:

```
ccards %>%
  mutate(
    creditcards_masked = map_chr(creditcards, ~ str_replace(.x, "^.{10}",
                                                  replacement = strrep("X", 10)))
  )
```

```
# A tibble: 6 x 2
  creditcards creditcards_masked
        <dbl> <chr>
1     3.66e13 XXXXXXXXXX4299
2     3.64e13 XXXXXXXXXX7466
3     3.60e13 XXXXXXXXXX5170
4     5.45e15 XXXXXXXXXX207504
5     2.22e15 XXXXXXXXXX361034
6     5.13e15 XXXXXXXXXX148536
```

This code is a little more complicated than the first one. The regular expression `^.{10}` indicates that we are looking for the first 10 characters of the string. We replace this pattern with 10 "X"s, specified by `strrep("X", 10)`. The function `strrep` is a basic function of R, which simply repeats a series of characters:

```
strrep("X", 10)
```

```
[1] "XXXXXXXXXX"
```

Similarly, we could replace the last 5 characters with "X"s:

```
ccards %>%
  mutate(
    creditcars = map_chr(creditcards, ~ str_replace(.x, "\\d{5}$",
                                                    replacement = strrep("X", 5)))
  )
```

```
# A tibble: 6 x 2
  creditcards creditcars
        <dbl> <chr>
1     3.66e13 365552245XXXXX
2     3.64e13 363504896XXXXX
3     3.60e13 360028879XXXXX
4     5.45e15 54475520692XXXXX
5     2.22e15 22210026543XXXXX
6     5.13e15 51276993861XXXXX
```

In the regular expression `\\d{5}$` we look for the last five digits of a string.

How to turn ages into groups

Another common technique for anonymizing data is to divide it into groups. Suppose we want to divide the ages of people into groups.

Let's first calculate the age of our Starwars characters again:

```
(age_starwars <- starwars %>%
  mutate(age = as.integer(format(Sys.Date(), "%Y")) - birth_year) %>%
  select(name, age) %>%
  drop_na(age))
```

```
# A tibble: 43 x 2
   name                 age
   <chr>              <dbl>
 1 Luke Skywalker     2003
 2 C-3PO              1910
 3 R2-D2              1989
 4 Darth Vader        1980.
 5 Leia Organa        2003
 6 Owen Lars          1970
 7 Beru Whitesun lars 1975
 8 Biggs Darklighter  1998
 9 Obi-Wan Kenobi     1965
10 Anakin Skywalker   1980.
# ... with 33 more rows
```

Let us now go through four techniques we can use to group this age variable. With the function `cut_width` we can create groups of arbitrary width from a numeric variable. So let's group age into groups of 10 years:

```
age_starwars %>%
  mutate(
    age_groups = cut_width(age, 10)
  )
```

```
# A tibble: 43 x 3
   name                 age age_groups
   <chr>              <dbl> <fct>
 1 Luke Skywalker     2003  (1995,2005]
 2 C-3PO              1910  (1905,1915]
 3 R2-D2              1989  (1985,1995]
 4 Darth Vader        1980. (1975,1985]
 5 Leia Organa        2003  (1995,2005]
 6 Owen Lars          1970  (1965,1975]
 7 Beru Whitesun lars 1975  (1965,1975]
 8 Biggs Darklighter  1998  (1995,2005]
 9 Obi-Wan Kenobi     1965  (1955,1965]
10 Anakin Skywalker   1980. (1975,1985]
# ... with 33 more rows
```

The round bracket means that a number is not included in the set. The square bracket means that a number is included in the set.

The function `cut_number` creates a certain number of sets. For example, we could say that the age column should be grouped into 10 groups:

```
age_starwars %>%
  mutate(
    age_groups = cut_number(age, 10)
  )
```

```
# A tibble: 43 x 3
   name                 age age_groups
   <chr>              <dbl> <fct>
 1 Luke Skywalker     2003  (2001,2014]
 2 C-3PO              1910  [1126,1922]
 3 R2-D2              1989  (1981,1991]
 4 Darth Vader        1980. (1976,1981]
 5 Leia Organa        2003  (2001,2014]
 6 Owen Lars          1970  (1965,1970]
 7 Beru Whitesun lars 1975  (1970,1976]
 8 Biggs Darklighter  1998  (1991,2001]
 9 Obi-Wan Kenobi     1965  (1965,1970]
10 Anakin Skywalker   1980. (1976,1981]
# ... with 33 more rows
```

Note, however, that the width of each group varies. For example, Luke Skywalker is in the set between 2011 and 2014 (a difference of 13 years) and Darth Vader is in the set between 1976 and 1981 (a difference of 5 years).

Then we can convert the age to millennia or decades. For example, Luke Skywalker is 2003 years old (at least in this data frame; I honestly don't know much about Starwars, so be gentle :)), so his age should fall in the millennia 2000. That's how we would do it with David Robinson's decade trick:

```
age_starwars %>%
  mutate(
    millenium = 1000 * (age %/% 1000)
  )
```

```
# A tibble: 43 x 3
   name                 age millenium
   <chr>              <dbl>     <dbl>
 1 Luke Skywalker     2003       2000
 2 C-3PO              1910       1000
 3 R2-D2              1989       1000
 4 Darth Vader        1980.      1000
 5 Leia Organa        2003       2000
 6 Owen Lars          1970       1000
 7 Beru Whitesun lars 1975       1000
 8 Biggs Darklighter  1998       1000
 9 Obi-Wan Kenobi     1965       1000
10 Anakin Skywalker   1980.      1000
```

```
# ... with 33 more rows
```

In a similar way, we could convert ages into decades:

```
age_starwars %>%
  mutate(
    decade = 10 * (age %/% 10)
  )
```

```
# A tibble: 43 x 3
   name                 age decade
   <chr>              <dbl>  <dbl>
 1 Luke Skywalker      2003   2000
 2 C-3PO               1910   1910
 3 R2-D2               1989   1980
 4 Darth Vader         1980.  1980
 5 Leia Organa         2003   2000
 6 Owen Lars           1970   1970
 7 Beru Whitesun lars 1975   1970
 8 Biggs Darklighter   1998   1990
 9 Obi-Wan Kenobi      1965   1960
10 Anakin Skywalker    1980.  1980
# ... with 33 more rows
```

How to remove house numbers from street names

Then we have street names and especially street numbers that sometimes need to be anonymized. Suppose we want to remove the street numbers of these streets:

```
street_names <- tibble(
  street_name = c("Bromley Lanes 34",
                  "Woodsgate Avenue 12",
                  "Ardconnel Terrace 99",
                  "Gipsy Birches 45",
                  "Legate Close 8",
                  "Stevenson Oval 9",
                  "St Leonard's Boulevard 112",
                  "Copper Chare 435",
                  "Glastonbury Glebe 82",
                  "Southern Way 91")
)
```

To remove them, we can use the `str_remove_all` function from the stringr package:

```
street_names %>%
  mutate(
    street_names_no_number = str_remove_all(street_name, "\\d")
  )
```

```
# A tibble: 10 x 2
   street_name                 street_names_no_number
   <chr>                       <chr>
 1 Bromley Lanes 34            "Bromley Lanes "
 2 Woodsgate Avenue 12         "Woodsgate Avenue "
 3 Ardconnel Terrace 99        "Ardconnel Terrace "
 4 Gipsy Birches 45            "Gipsy Birches "
 5 Legate Close 8              "Legate Close "
 6 Stevenson Oval 9            "Stevenson Oval "
 7 St Leonard's Boulevard 112 "St Leonard's Boulevard "
 8 Copper Chare 435            "Copper Chare "
 9 Glastonbury Glebe 82        "Glastonbury Glebe "
10 Southern Way 91             "Southern Way "
```

With the regular expression `\\d` we can remove all digits from a string.

How to encrypt and decript columns

Finally, we can anonymize each column by encrypting it. Encryption is a complicated and vast topic. I can't and won't go into detail here, but let me give you a brief introduction to how it works in R.

When we encrypt a column, we convert the values of a column into another form, which we call ciphertext. The ciphertext is not readable by humans, but it can be converted back to the original value. There are two forms of encryption. Symmetric encryption, where a single key is used to encrypt and decrypt a value, and asymmetric encryption, where two keys are used to encrypt and decrypt a value. A key is plaintext that translates between the two representations. Once you have the key in symmetric encryption, you can decrypt values. To decrypt values in asymmetric encryption, you need the public key and the private key.

The public key is as it says public, so open to anyone. The private key is a key you should not share with anyone. Only when you have both, can you decrypt a value. Also, private key cannot be guessed from the public key. To add another level of security, the private key also sometimes has a passphrase (or password) to it.

So much for the short theory, let's encrypt a column in R. For example, suppose you have this table of usernames and passwords:

```r
users <- tibble(
  name = c("Alexander", "Marie", "John"),
  password = c(12345, "8$43_45*", "becker23#")
)
```

We can encrypt this data with the package encryptr. First we need to load the package and create the private and public keys using the `genkeys` function:

```r
library(encryptr)

# genkeys() generates a public and private key pair
# Passphrase: 456#7
genkeys()
```

The function prompted us to provide a passphrase for the private key. This passphrase and the private key should not be shared with anyone!

Once we have the passphrase, we can encrypt our columns. Let's encrypt the password column:

```r
users_encrypted <- users %>%
  encrypt(password)
```

```r
users_encrypted %>%
  glimpse()
```

```
Rows: 3
Columns: 2
$ name     <chr> "Alexander", "Marie", "John"
$ password <chr> "5b084bd19feef7b413b7a5ec4417f4c28b06e1f6ba8c91…
```

As you can see, the passwords have been converted into a long string. It is not possible to decrypt the passwords from this string. To decrypt the column, we simply use the decrypt function:

```r
users_encrypted %>% decrypt(password)
```

You must provide the passphrase to decrypt the column. Also, this works only if the R file is in the same directory as the public and private keys.

ℹ Summary

Here is what you can take from this tutorial.

- Anonymization and pseudoanonymization are two different techniques. Anonymization is not reversible, pseudoanonymization is.
- Whether your data is anonymized or pseudoanonymized depends on how you handle the data.
- With asymmetric encryption, you have a public key and a private key. Only with both keys you can decrypt a message.