# What's private, and why do we care?

DATA PRIVACY AND ANONYMIZATION IN PYTHON

**Rebeca Gonzalez**
Data engineer

datacamp

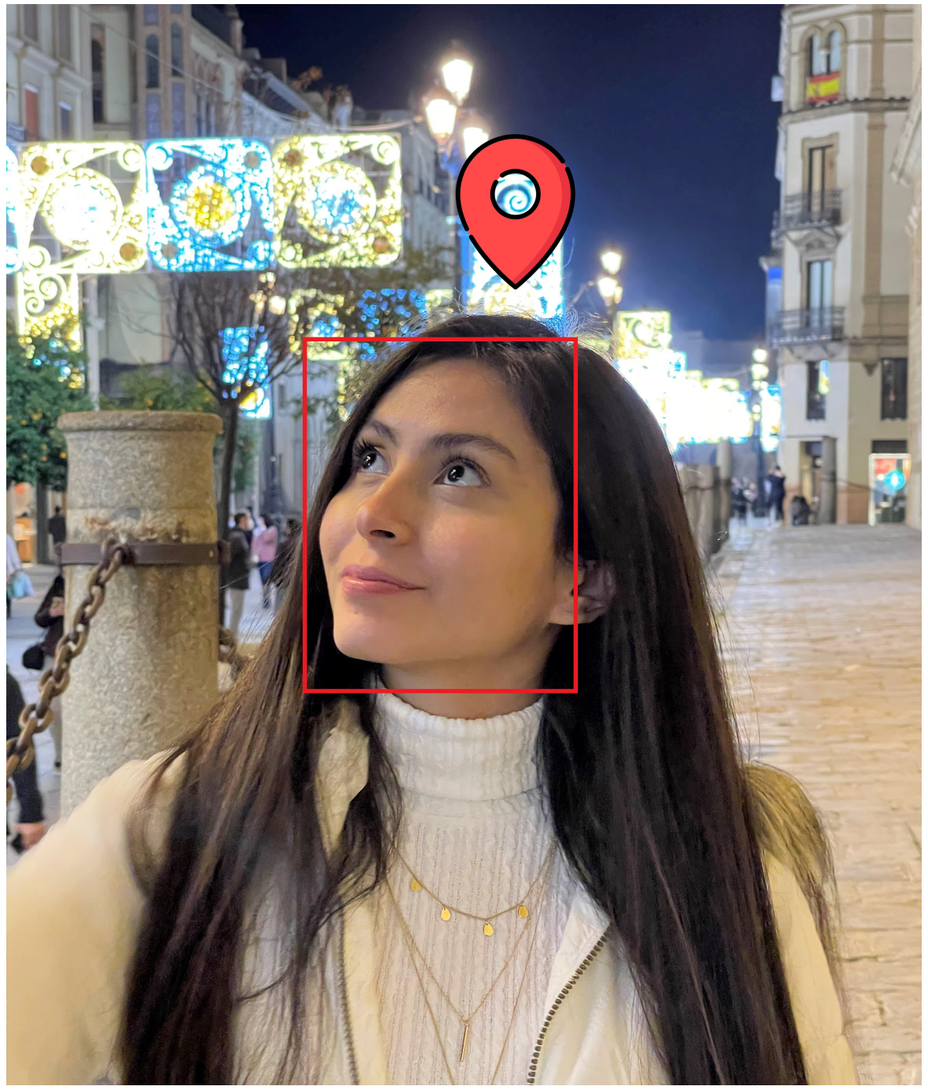
# Facebook and Cambridge Analytica scandal

## Impact of data privacy

- Unauthorized access to 87 million people's personal information
- Build psychological profiles of American voters
- Persuade them during political campaigns

# What's privacy?

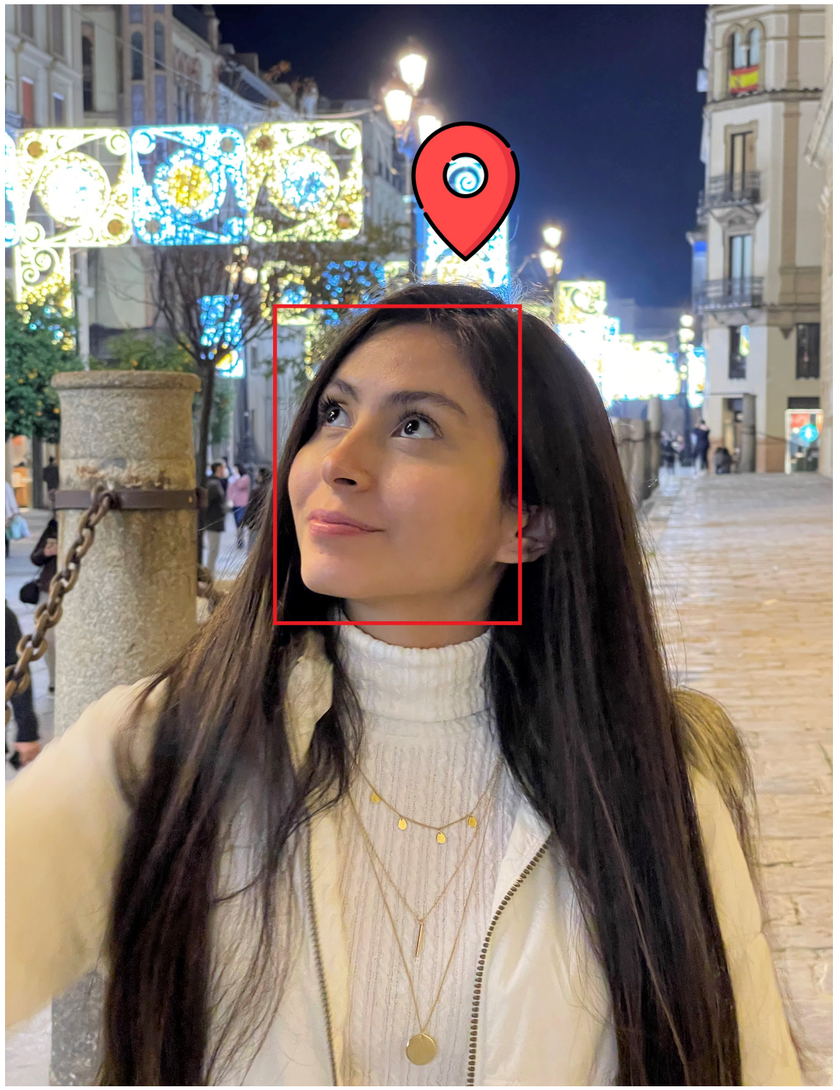
# Information flow and privacy



# Information flow and privacy

- How your personal information flows

> "The ability to ensure flows of information that satisfy social and legal norms."



# Personally identifiable information (PII)

Data that, when used alone or with other relevant data, can identify someone.

# Sensitive PII

- Clearly about someone
- Exposure can lead to harm, embarrassment or inconvenience

# Sensitive PII

- Full name
- Social Security Number (SSN)
- Financial information
- Medical records

# Non-sensitive PII

Data that cannot be used alone to trace a person

- Gender
- Occupation
- Zip code
- City of birth

# Non-sensitive PII

Data that cannot be used alone to trace a person, such as gender, occupation, zip code, or city of birth.

- Gender
- Occupation
- Zip code
- City of birth

**Still can be used with other information to identify someone!**

"Head of government in Europe"

"Born on July 17, 1954"

[1] Photo of Angela Merkel from Wikimedia Commons.

# GDPR: EU General Data Protection Regulation

Protects PII of people living, or whose data is processed, within Europe.

## Key principles of the GDPR

1. Lawfulness, fairness, and transparency
2. Purpose limitation
3. Data minimization
4. Accuracy
5. Storage limitation

**Learn more here**

# Data suppression

*Removing selected information to protect the privacy of subjects.*

## Attribute suppression

- Removing columns entirely

## Cell/record suppression

- Removing or replacing data in rows or cells

# Attribute suppression on a dataset

```
# Attribute suppression on Sensitive PII "name"
suppressed_salaries = salaries.drop('name', axis="columns")

# Explore obtained dataset
suppressed_salaries.head()
```

```
     gender    status    salary     pay_basis    position_title
0    Male    Employee    64400.0    Per Annum    DEPUTY DIRECTOR
1    Male    Employee    43600.0    Per Annum    ASSOCIATE DIRECTOR
2    Male    Employee    120000.0   Per Annum    SPECIAL ASSISTANT TO THE PRESIDENT AND DEPUTY ...
3    Male    Employee    86200.0    Per Annum    LEAD ADVANCE REPRESENTATIVE
4    Male    Employee    106000.0   Per Annum    SPECIAL ASSISTANT TO THE PRESIDENT AND DIRECTO...
```

# Record suppression on a dataset

```
# Explore the DataFrame
salaries.head()
```

```
     hours    performance    salary
0    72       51             $80,500.00
1    20       99             $2,805,000.00
3    75       62             $75,800.00
4    74       58             $60,000.00
5    70       54             $79,000.00
```

# Record suppression on a dataset

```
# Drop rows with salaries higher than 2,000,000
salaries = salaries.drop(salaries[salaries.Salary > 2000000].index)

# See reasulting DataFrame
salaries.head()
```

```
    hours    performance    salary
0     72        51             80500
2     75        62             75800
3     74        58             60000
4     70        54             79000
5     68        53             62000
```

# Suppression and linkage attacks

Attribute supression on "Name" column

| Nam | Zipcode | Age | Sex | Disease |
|---|---|---|---|---|
| li | 47677 | 29 | F | Ovarian Cancer |
| | 47602 | 22 | F | Ovarian Cancer |
| s | 47678 | 27 | M | Prostate Cancer |
| | 47905 | 43 | M | Flu |
| | 47909 | 52 | F | Heart Disease |
| Fre | 47906 | 47 | M | Heart Disease |

Voter registration data (Background knowledge)

| Name | Zipcode | Age | Sex |
|---|---|---|---|
| Alice | 47677 | 29 | F |
| Bob | 47983 | 65 | M |
| Carol | 47677 | 22 | F |
| Dan | 47532 | 23 | M |
| Ellen | 46789 | 43 | F |

# Let's practice!

DATA PRIVACY AND ANONYMIZATION IN PYTHON

# Data masking and data generation with Faker

DATA PRIVACY AND ANONYMIZATION IN PYTHON

**Rebeca Gonzalez**
Data engineer

datacamp

# Quasi-identifiers

Non-sensitive Personally identifiable information can be quasi-identifiers as well.

"Head of government in Europe"

"Born on July 17, 1954"

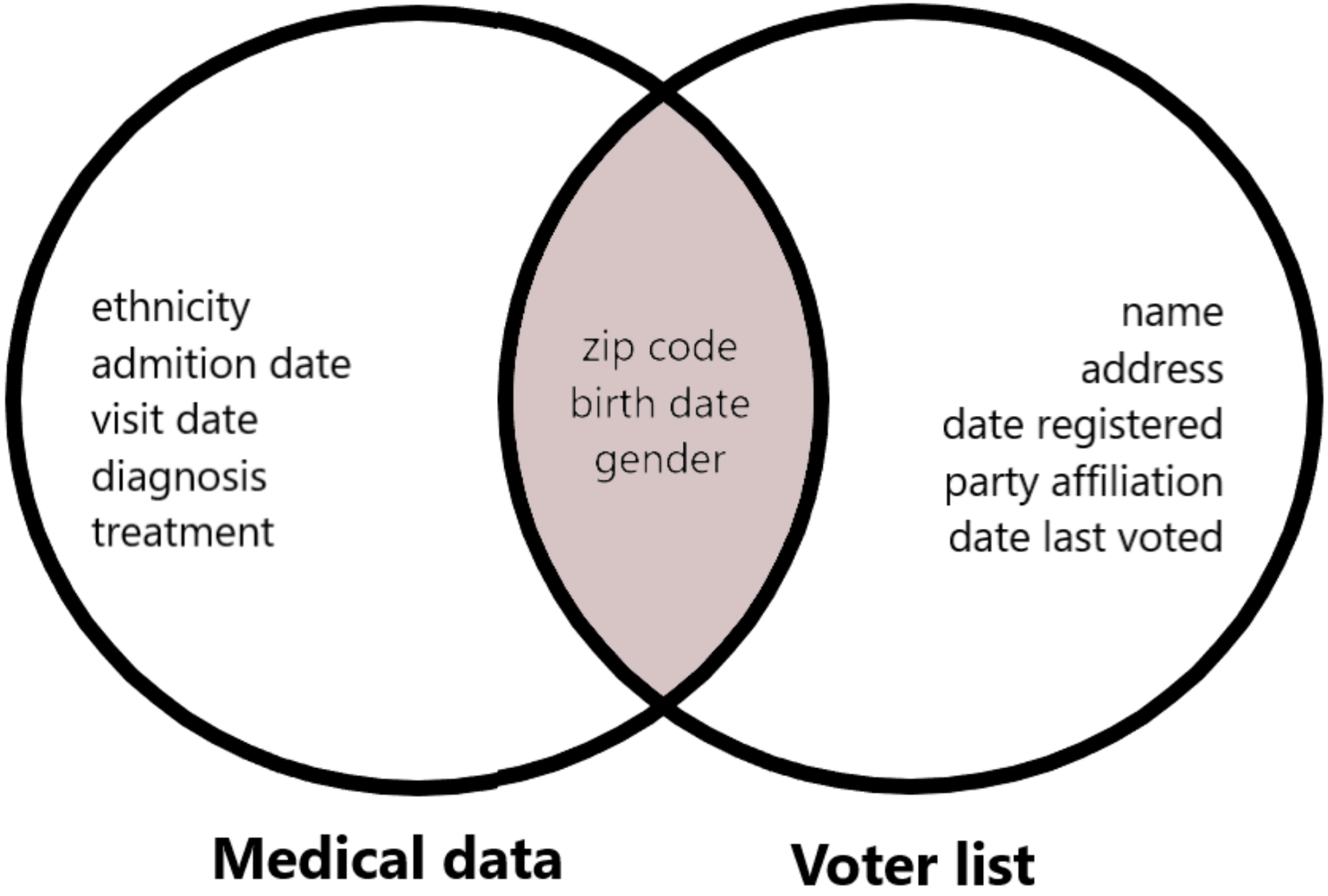
[1] Photo of Angela Merkel from Wikimedia Commons

# Quasi-identifiers

When combined with other quasi-identifiers, they become identifying.

- Age
- Gender
- Occupation
- Personal dates: birth, death, healthcare admission, visit, etc.

# Quasi-identifiers and re-identification attacks



# More anonymization techniques

Anonymization techniques for quasi-identifiers to reduce data disclosure risks.

| Name | Zipcode | Age | Sex | Disease |
|---|---|---|---|---|
| Alice S. | 47677 | 29 | F | Ovarian Cancer |
| Betty Q. | 47602 | 22 | F | Ovarian Cancer |
| Charles D. | 47678 | 27 | M | Prostate Cancer |
| David E. | 47905 | 43 | M | Flu |
| Emily J. | 47909 | 52 | F | Heart Disease |
| Fred K. | 47906 | 47 | M | Heart Disease |

Original data

| Zipcode | Age | Sex | Disease |
|---|---|---|---|
| 47677 | 29 | F | Ovarian Cancer |
| 47602 | 22 | F | Ovarian Cancer |
| 47678 | 27 | M | Prostate Cancer |
| 47905 | 43 | M | Flu |
| 47909 | 52 | F | Heart Disease |
| 47906 | 47 | M | Heart Disease |

After suppression

# Data masking

Original DataFrame

| | name | phone |
|---|---|---|
| 0 | Cassandra Nelson | 4399406975395 |
| 1 | Brian Moss | 0389407128613 |
| 2 | Melody Gill | 8283308773967 |
| 3 | Sandra Huber | 4366608954250 |
| 4 | Patricia Webster | 4466462475574 |

DataFrame with the names masked

| | name | phone |
|---|---|---|
| 0 | xxxx | 4399406975395 |
| 1 | xxxx | 0389407128613 |
| 2 | xxxx | 8283308773967 |
| 3 | xxxx | 4366608954250 |
| 4 | xxxx | 4466462475574 |

Replacing sensitive values with other characters like "x" is also known as data masking.

# Data masking

```
# Explore the DataFrame
df.head()
```

```
     country         card_number          email
0    Finland         3546746666030419     fisherlarry@gmail.com
1    Belarus         4303032415762821     matthewfritz@gmail.com
2    Turkmenistan    4536883671157        alexanderlong@gmail.com
3    Puerto Rico     3568819286614160     kristenwheeler@gmail.com
4    Angola          2514167462583016     dhenson@gmail.com
```

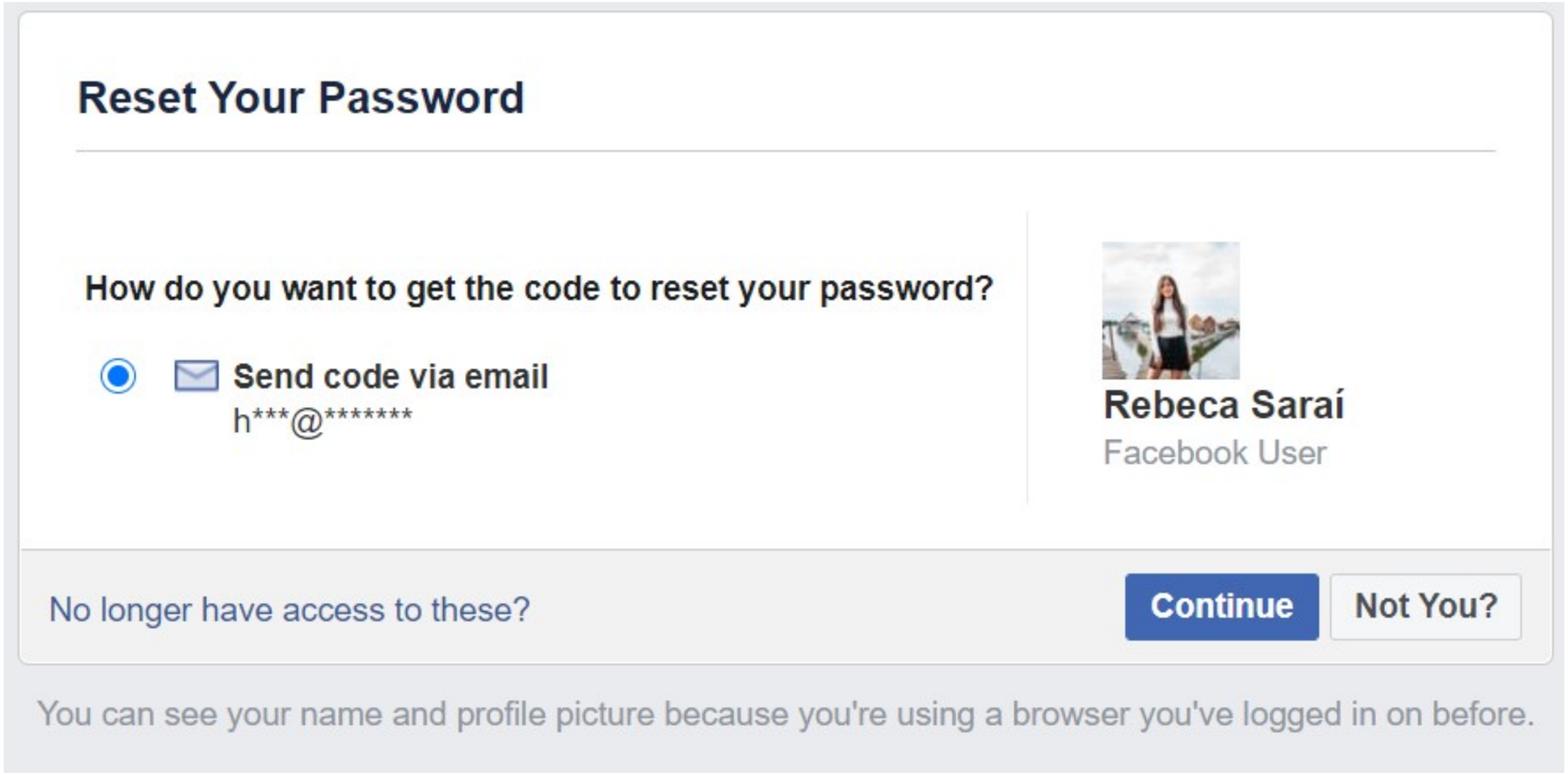
# Data masking

```
# Uniformly mask the card number colum
df['card_number'] = '****'

# See resulting DataFrame
df.head()
```

```
     country          card_number     email
0    Finland          ****            fisherlarry@gmail.com
1    Belarus          ****            matthewfritz@gmail.com
2    Turkmenistan     ****            alexanderlong@gmail.com
3    Puerto Rico      ****            kristenwheeler@gmail.com
4    Angola           ****            dhenson@gmail.com
```

# Partial data masking



# Partial data masking

```
# Mask username from email
df2['email'] = df2['email'].apply(lambda s: s[0] + '****' + s[s.find('@'):] )

# See the resulting pseudonymized data
df2.head()
```

```
     country          card_number          email
0    Finland          3546746666030419     f****@gmail.com
1    Belarus          4303032415762821     m****@gmail.com
2    Turkmenistan     45368836711 57       a****@gmail.com
3    Puerto Rico      3568819286614160     k****@gmail.com
4    Angola           2514167462583016     d****@gmail.com
```

# Generating synthetic data

| | Country | Card_number | Email |
|---|---|---|---|
| 0 | Finland | 2299019158659850 | patrick28@gmail.com |
| 1 | Belarus | 4849530471252505 | trivera@gmail.com |
| 2 | Turkmenistan | 2298943538908227 | matthew32@gmail.com |
| 3 | Puerto Rico | 30357901334084 | boltonrichard@gmail.com |
| 4 | Angola | 5315364087008412 | jorge37@gmail.com |

| | Country | Card_number | Email |
|---|---|---|---|
| 0 | Finland | 3565986561740748 | patrick28@gmail.co |
| 1 | Belarus | 3572135942035995 | trivera@gmail.com |
| 2 | Turkmenistan | 2274632826606183 | matthew32@gmail.c |
| 3 | Puerto Rico | 373508449581876 | boltonrichard@gmai |
| 4 | Angola | 2710445991443049 | jorge37@gmail.com |

- Replacing sensitive information with newly generated data, similar to the original.

# Generating synthetic data

```
# Import Faker class
from faker import Faker

# Create fake data generator
fake_data = Faker()

# Generate a credit card number
fake_data.credit_card_number()
```

```
3542216874440804
```

# Generating synthetic data

```
# Mask card number with new generated data using a lambda function
df['card_number'] = df['card_number'].apply(lambda x: fake_data.credit_card_number())

# See the resulting pseudonymized data
df.head()
```

```
     country         card_number          email
0    Finland         3596625386355448     fisherlarry@gmail.com
1    Belarus         376297265347524      matthewfritz@gmail.com
2    Turkmenistan    4377494880888682     alexanderlong@gmail.com
3    Puerto Rico     30553931809810       kristenwheeler@gmail.com
4    Angola          42417357483 82       dhenson@gmail.com
```

# Generating other types of data

```
fake_data.name()
```

```
'Kelly Clark'
```

```
fake_data.name_male()
```

```
'Antonio Henderson'
```

```
fake_data.name_female()
```

```
'Jennifer Ortega'
```

# Time to practice!

DATA PRIVACY AND ANONYMIZATION IN PYTHON

# Anonymizing with data generalization

DATA PRIVACY AND ANONYMIZATION IN PYTHON

**Rebeca Gonzalez**
Data engineer

datacamp

# Data generalization

Technique that allows you to replace data values with less precise ones.

The purpose is to eliminate some of the identifiers while retaining data utility for analysis.

Allows you to replace a data value with a less precise one: "Dancer" -> "Artist"

```
     name              location
0    Amanda Hooper     146 Rodgers Field\nGregoryview, MS 71630
1    Sarah Smith       817 Garcia Shoal\nJonesville, AR 30299
2    Sean Boyd III     1938 90th St\nDallas, TX 59715
```

```
     name              location
0    Amanda Hooper     998 Boone Estate\nReedborough, MS 71630
1    Sarah Smith       7255 Shelby Rapids Apt. 455\nKarenland, AK 30299
2    Sean Boyd III     791 Crist Parks\nGreenton, TX 59715
```

# Data generalization

```
# Original data
df_employees.head()
```

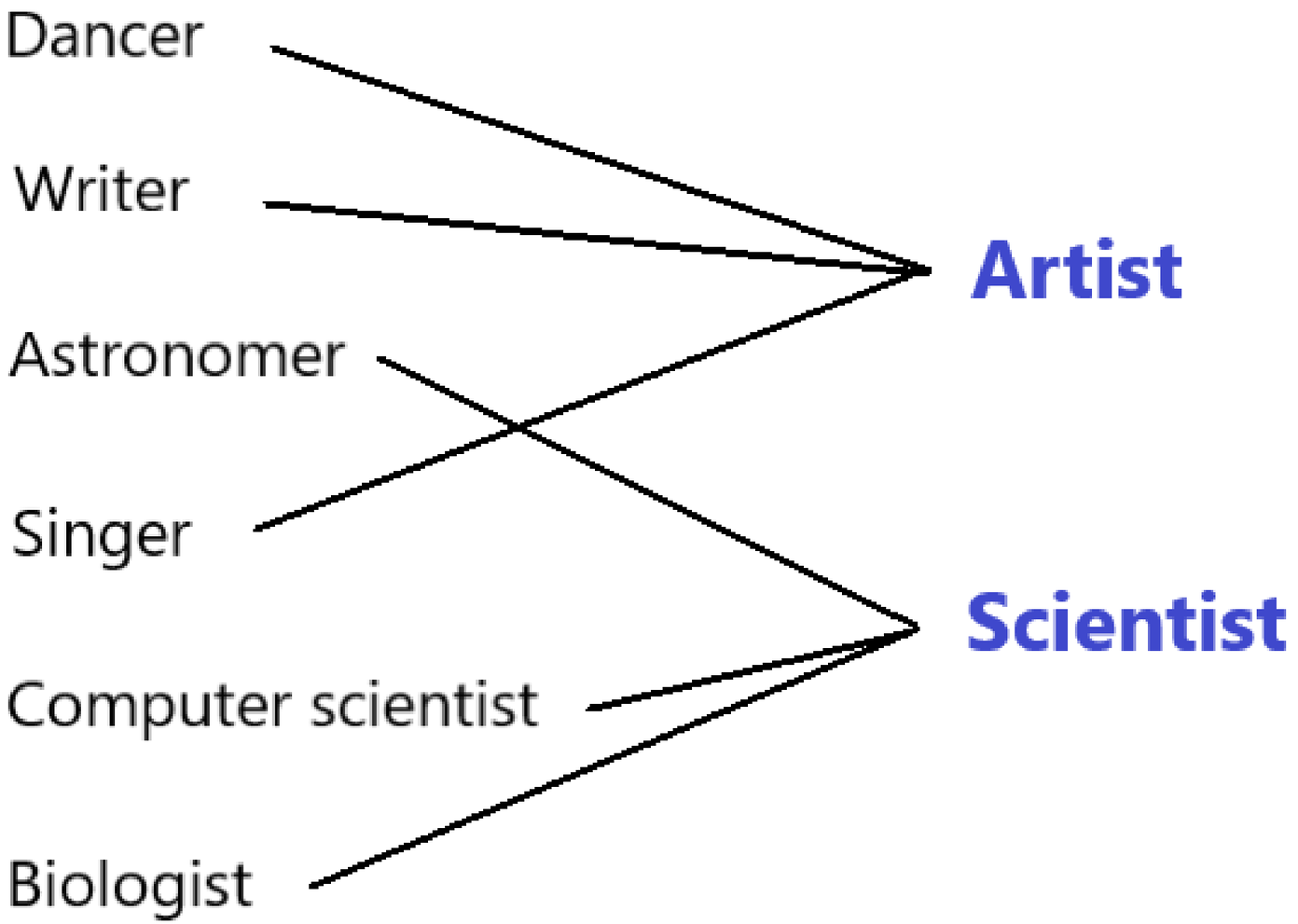
| | first name | last name | age | ssn |
|---|---|---|---|---|
| 0 | Amber | Brown | 91 | 798-29-4785 |
| 1 | William | Gibson | 34 | 431-66-8381 |
| 2 | Daniel | Lee | 92 | 825-91-5550 |
| 3 | Andrea | Stevenson | 64 | 188-59-3544 |
| 4 | Julie | Horn | 35 | 020-60-6388 |

```
# Generalized data: Age in intervals and masked SSN
generalized_df.head()
```

| | First name | Last name | Age | SSN |
|---|---|---|---|---|
| 0 | Amber | Brown | (80, 99] | 798-**-**** |
| 1 | William | Gibson | (30, 50] | 431-**-**** |
| 2 | Daniel | Lee | (80, 99] | 825-**-**** |
| 3 | Andrea | Stevenson | (60, 80] | 188-**-**** |
| 4 | Julie | Horn | (30, 50] | 020-**-**** |

Those who are 34 years old will be placed in a range of 30 to 50. This is also known as binning.

# Data aggregation



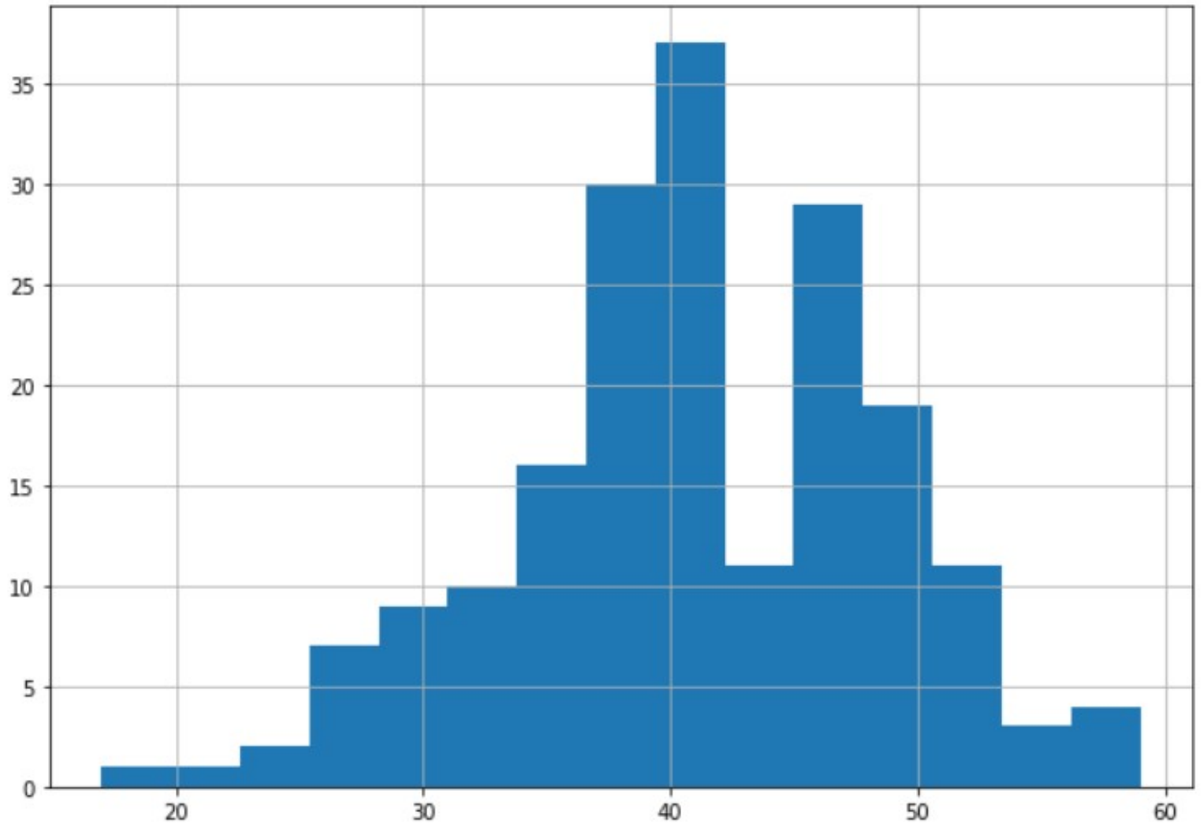
# Medical dataset

```
# Explore the dataset
df_medical.head()
```

```
    age    gender    department    condition
0    30    F         Finance       Anxiety disorders
1    42    M         Production    Bronchitis
2    35    F         Marketing     Dysthymia
3    39    F         Production    Dysthymia
4    40    M         Marketing     Flu
```

# Medical dataset

```
# Explore the histogram of the age variable
df_medical['age'].hist(bins=15)
```



# Generalization

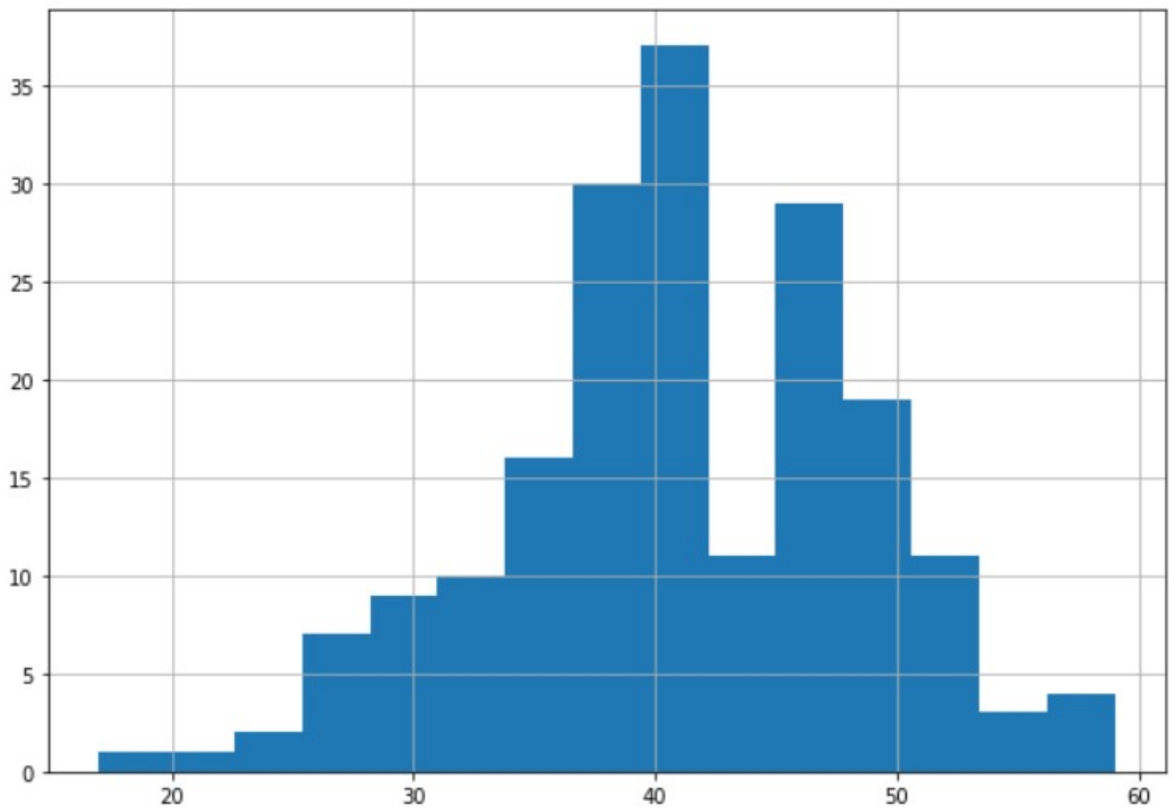
```
# Apply generalization by transforming to binary data
df_medical['age'] = df_medical['age'].apply(lambda x:">=40" if x>=40 else "<40" )

# See results
df_medical.head()
```

```
    age    gender    department    condition
0    <40   F         Finance       Bronchitis
1    >=40  M         Production    Bronquitis
2    <40   F         Finance       Dysthymia
3    <40   F         Production    Dysthymia
4    >=40  M         Marketing     Flu
```

# Top and bottom coding

```
# Explore the histogram of the age variable
df_medical['age'].hist(bins=15)
```




- Few people are younger than 25 and few are older than 55
- Apply bounds to reduce the risk of re-identification for these people in outliers
- Best when there are very few observations within a category, especially at the tails of the distribution

# Top coding

```
# Filter to see rows affected
df_medical[df_medical['age'] >= 55]
```

```
     age   gender   department     condition
26   56    F        Production     Flu
65   55    M        Finance        Dysthymia
126  59    F        Production     Anxiety disorders
139  58    F        Finance        Dysthymia
142  59    M        Marketing      Flu
145  57    M        Marketing      Anxiety disorders
```

# Top code

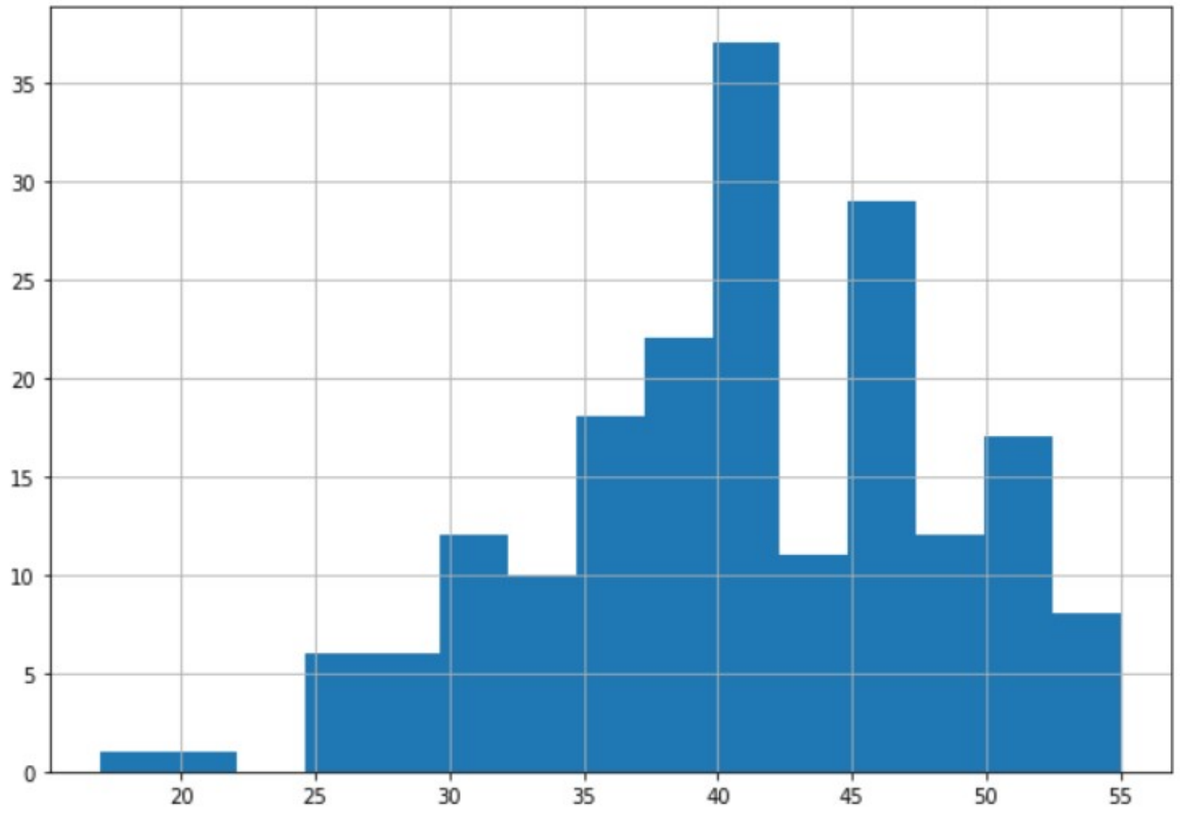
```
# Top code the age to 55
df_medical.loc[df_medical['age'] > 55, 'age'] = 55

# Filter to see rows affected
df_medical[df_medical['age'] >= 55]
```

| | age | gender | department | condition |
|---|---|---|---|---|
| 26 | 55 | F | Production | Flu |
| 65 | 55 | M | Finance | Dysthymia |
| 126 | 55 | F | Production | Anxiety disorders |
| 139 | 55 | F | Finance | Dysthymia |
| 142 | 55 | M | Marketing | Flu |
| 145 | 55 | M | Marketing | Anxiety disorders |

# Bottom coding

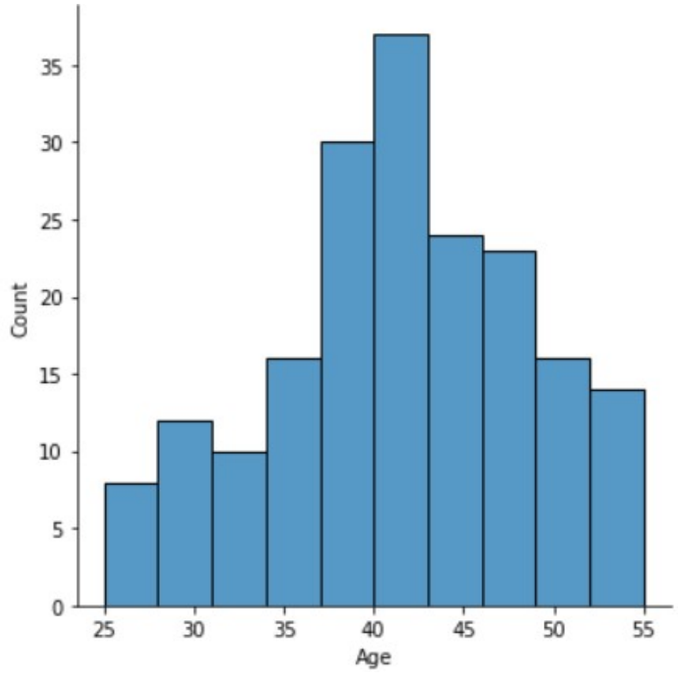
```
# Explore the histogram of the age variable
df_medical['age'].hist(bins=15)
```



# Bottom code

```
# Bottom code the age to 25
df_medical.loc[df['age'] < 25, 'age'] = 25

# Explore the histogram of the age variable
df_medical['age'].hist(bins=15)
```



# Data generalization and privacy models

Better when used with Suppression and Masking following a **Privacy Model** like K-anonymity.

Specify conditions that the dataset must satisfy to keep disclosure risk under control.

We'll learn how to implement this privacy model in the next chapter!

# Time to practice!

DATA PRIVACY AND ANONYMIZATION IN PYTHON