CDM Coursework 2: Data Sharing and Anonymisation

This file contains the code and corresponding explanations for CDM coursework 2.

1. Importing Python Packages

Before we loaded in the data, we imported the following Python packages which we used later.

In [22]:
```
import pandas as pd  # Data manipulation and analysis
from datetime import datetime  # Date and time operations
import re  # Text processing
import requests
from bs4 import BeautifulSoup  # HTTP requests and web scraping
import hashlib as hl  # Hashing
import os  # File system operations
from geopy.geocoders import Nominatim  # Mapping geographical data
import openpyxl  # Enables data export to excel
```

2. Importing the Dataset

We imported the customer information dataset and explored the data to develop an initial data processing plan.

In [23]:
```
data = pd.read_csv("customer_information.csv")
data.head()
```

Out[23]:

	given_name	surname	gender	birthdate	country_of_birth	current_country	phone_number	postcode	national_insurance_
0	Lorraine	Reed	F	05/07/1984	Armenia	United Kingdom	(07700) 900876	LS5 8FN	ZZ 19
1	Edward	Williams	M	17/06/1997	Northern Mariana Islands	United Kingdom	(07700) 900 877	M0U 1RA	ZZ 7
2	Hannah	Turner	F	15/06/1990	Venezuela	United Kingdom	+447700 900148	SO1 8HZ	ZZ 9
3	Christine	Osborne	F	29/07/2000	Eritrea	United Kingdom	+447700 900112	B18 8LW	ZZ 39
4	Francesca	Yates	F	04/11/1968	Ecuador	United Kingdom	07700 900 413	TQ2 6BE	ZZ 30

We can see that the following columns contain sensitive information [1]:

'given_name', 'surname', 'birthdate', 'phone_number', 'postcode', 'national_insurance_number', 'bank_account_number'

Therefore, we needed to **delete** and **replace them** with an identifier object that does not contain sensitive information before the data is shared. We describe how we did this in the data anonymisation section.

3. Data Pre-processing

We made the following adjustments to standardise and clean the data.

3.1 Looking for missing data

Firstly, we looked for any missing data

In [24]:
```
data.isna().sum()
```

Out[24]:
```
given_name                  0
surname                     0
gender                      0
birthdate                   0
country_of_birth            0
current_country             0
phone_number                0
postcode                    0
national_insurance_number   0
bank_account_number         0
cc_status                   0
weight                      0
height                      0
blood_group                 0
avg_n_drinks_per_week       0
avg_n_cigret_per_week       0
education_level             0
n_countries_visited         0
dtype: int64
```

3.2 Standardising National Insurance Number

Within the dataset, we can see that the data format in the 'national_insurance_number' column is inconsistent: some have gaps between numbers.

In [6]:
```
data['national_insurance_number'].head()  #Read the National Insurance number
```

Out[6]:
```
0    ZZ 19 48 92 T
1      ZZ 753513 T
2      ZZ 947196 T
3    ZZ 39 69 47 T
4    ZZ 30 98 91 T
Name: national_insurance_number, dtype: object
```

We needed to standardise this value because we used it in the hashing process outlined in section 4.1.

We achieved this by **removing all gaps and spaces**.

In [21]:
```
# define a function which takes a string and returns a new string
# where all spaces of the original string have been replaced
# with no space
def delete_spaces(string):
    return string.replace(' ', '')

data['national_insurance_number'] = data['national_insurance_number'].apply(delete_spaces)  #Apply that function
data['national_insurance_number'].head()
```

Out[21]:
```
0    ZZ194892T
1    ZZ753513T
2    ZZ947196T
3    ZZ396947T
4    ZZ309891T
Name: national_insurance_number, dtype: object
```

3.3 Extracting the postcode area from the postcode

We checked if the postcodes corresponded to real addresses [2]. We did this with the code below:

In [9]:
```
geolocator = Nominatim(user_agent="postcode_validation")
```

```
def is_valid_postcode(postcode):
    location = geolocator.geocode(postcode, country_codes="GB")
    if location is not None:
        return True
    else:
        return False

data['invalid_postcode'] = data['postcode'].apply(lambda x: is_valid_postcode(x))
data['invalid_postcode'].value_counts()
```

Out[9]:
```
invalid_postcode
False    908
True      92
Name: count, dtype: int64
```

As we can see from this, most of the postcodes are invalid.

Therefore, we took only the first letters of the postcode, before any number. These correspond to a specific area (postcode area).

> e.g. We extracted 'LS' from 'LS5 8FN'.

In [8]:
```
# define a function that extracts the letter(s) before the first number in a postcode
# uses the 'match' function from the 're' package to find if there is one or more
# non-digit characters at the beginning of the string
def extract_letters_before_first_number(string):
        match = re.match(r'^\D+', string)

        if match:
            return match.group()
        else:
            return None

data['postcode'] = data['postcode'].apply(lambda x: extract_letters_before_first_number(x))
```

4. Data Anonymisation

4.1 Pseudonymisation by hashing

We decided to pseudonymise the dataset and keep the sensitive information in a private key dataset. The advantage of pseudonymisation is that we can re-identify individuals as needed [3]; this can be helpful if, for example, we gain new information about a customer that we wish to update the dataset with.

To generate a primary key which the shared datasets can use to link to the private key dataset, we created individualised hashes based on a combination of each client's full name and every other character of their national insurance number, along with a randomly generated 64-byte salt.

Using a hash as the identifier, as opposed to a random character sequence generator, has the advantage of being deterministic [4] - it would be impossible for someone with a difference anywhere in their first name, last name, halved NI number and randomly generated 64-byte salt, to have the same hash.

In [32]:
```
# define a function which defines a new string which can be encoded
# and subsequently hashed based on first name, surname and half the NI number
def get_string_for_hash(first_name, surname, national_insurance_number):
        halved_ni = ""
        for index, char in enumerate(national_insurance_number):
                halved_ni += char
        return first_name + surname + halved_ni
        # the output of the function is a concatenation of first_name, last_name and halved_ni

data['string_for_hash'] = data.apply(
    lambda row: get_string_for_hash(row['given_name'],
                                    row['surname'],
                                    row['national_insurance_number']
                                   ), axis=1)
```

```
# define a function which encodes the string which needs to be hashed into bytes using utf-8
def encode_data(data):
    encoded_data = data.encode('utf-8')  # this allows the string to be concatenated with a salt
    return encoded_data

data['encoded_data'] = data.apply(lambda row: encode_data(row['string_for_hash']), axis = 1)

# define a function that generates a unique, randomly generated 64-byte salt for each each row
def generate_salt():
    salt = os.urandom(64)
    return salt
data['salt'] = data.apply(lambda row: generate_salt(), axis=1)

# define a function that concatenates the encoded variable and the salt
def combine_data_with_salt(data, salt):
    hash_data_with_salt = data + salt
    return hash_data_with_salt

data['data_and_salt_ready'] = data.apply(
    lambda row: combine_data_with_salt(row['encoded_data'], row['salt']), axis = 1)

hash_object = hl.sha256()  # create a hash object which applies hashlib's sha256 algorithm

# define a function which updates the hash object with a value of your choosing, and return
# a hex value which displays the hash
def hash_my_data(x):
    hash_object.update(x)
    return hash_object.hexdigest()

data['id'] = data.apply(lambda row: hash_my_data(row['data_and_salt_ready']), axis = 1)
data['id'].head()
```

```
Out[32]: 0    b7d8d9b1c55af131072257d9daa73e06ca3f8b5861c93b...
         1    a67df2cc43c5384aef155987b8ceacff5f260f4378db7f...
         2    cfb54b02b9d870f9ee990080ad80ef77e02e208d8d476c...
         3    3805bf1c1b04d84e7d1e56459a1325055c84ebbdd1a51c...
         4    342d728e9458dea139eca4c251660a617eb53b5d08b235...
         Name: id, dtype: object
```

4.2 Banding data that will be shared

As well as sensitive information, our data also contain 'quasi-identifiers' [5] - variables which don't directly identify persons on their own, but in combination with other information could identify someone.

Banding is the process of making data less precise. It increases a dataset's k-anonymity [6] which refers to how difficult it is to use combinations of quasi-identifiers in any given row to re-identify individuals, at the cost of losing information. We balanced these two considerations when deciding to band the data that we shared.

4.2.1 Banding geographical data

As mentioned in section 3.2, we obtained postcode areas from full postcodes in (e.g. M from M0U 1RA) because many postcodes were invalid.

The postcode area can be given to the researchers. We mapped it to the area name for clarity (e.g. M to Manchester).

To do so, we used the information found on "https://ideal-postcodes.co.uk/guides/postcode-areas" with the following format:

Postcode Area	Postcode Area Name	Region
AB	Aberdeen	Scotland
AL	St. Albans	East of England

... ...

For the government, however, we banded this information by mapping the postcode areas directly to regions using the same dictionary table.

This preserves some geographical information within the dataset but has less of an adverse effect on k-anonymity. This data will be shared publicly and therefore lower granularity will be required.

In summary, researchers will have access to the postcode area name, and the government will have access to the region.

In [25]:
```
# Importing the dictionary from the website

# URL of the webpage containing the dictionary
url = "https://ideal-postcodes.co.uk/guides/postcode-areas"

# send a GET request to the specified URL
response = requests.get(url)

# read the content of webpage using BeautifulSoup with an HTML parser
soup = BeautifulSoup(response.content, 'html.parser')

# create empty sets
regions = {}
postcode_areas = {}

# find all 'table' in the webpage
tables = soup.find_all('table')

# for each table,
for table in tables:

    # for each row(excluding the header) in the table,
    for row in table.find_all('tr')[1:]:
```

```
        # find all 'table data cells' in the row
        cells = row.find_all('td')

        if len(cells) > 1:
            area_code = cells[0].text.strip()
            region = cells[2].text.strip()
            postcode_area = cells[1].text.strip()

            # map the area code to the area name in the 'areas' dictionary
            regions[area_code] = region

            # map the area code to the city name in the 'cities' dictionary
            postcode_areas[area_code] = postcode_area


# Defining a function that extracts the areas

def get_region_from_postcode(postcode):
    outcode = postcode.split()[0]

    # call the 'extract_letters_before_first_number' function to get the postcode area,
    # e.g. get 'LS' from 'LS5'
    first_letters = extract_letters_before_first_number(outcode)

    # use the extracted postcode area to access the corresponding area name
    # from the 'areas' dictionary, e.g. match 'LS to 'North East'
    return regions[first_letters]

# Defining a function that extracts the postcode

def get_postcode_area_from_postcode(postcode):
    outcode = postcode.split()[0]

    # call the 'extract_letters_before_first_number' function to get the postcode area,
    # e.g. get 'LS' from 'LS5'
```

```
    first_letters = extract_letters_before_first_number(outcode)

    # use the extracted postcode area to access the corresponding city name
    # from the 'cities' dictionary, e.g. match 'LS to 'Leeds'
    return postcode_areas[first_letters]
```

In [26]:
```
# Applying the functions to get areas (for the government) and cities (for the researchers)

data['region'] = data.apply(lambda row: get_region_from_postcode(row['postcode']), axis=1)
data['postcode_area'] = data.apply(
    lambda row: get_postcode_area_from_postcode(row['postcode']), axis=1)
```

```
# This is formatted as code
```

Further banding of regions (for the government)

After completing the previous steps, we now have two new columns in the general dataset, **'postcode_area_name'** and **'region'**.

However, as the UK government is going to publish the dataset, we will now band the areas into larger, more generic regions, to further increase k-anonymity and reduce the risk of an individual being re-identified.

The code below bands the regions, according to this table:

Region	Greater region
'East Midlands' 'West Midlands'	'Midlands'
'Scotland' 'Wales' 'Northern Ireland'	'Scotland, Wales & Northern Ireland'
'North West' 'North East' 'Isle of Man'	'North of England'

In [27]:
```python
# define the function that bands the region

def group_region(region):
    if region in ['East Midlands', 'West Midlands']:
        return 'Midlands'
    if region in ['Scotland', 'Wales', 'Northern Ireland']:
        return 'Scotland, Wales & Northern Ireland'
    if region in ['North West', 'North East', 'Isle of Man']:
        return 'North of England'
    if region in ['South East', 'South West', 'Channel Islands', 'East of England', 'East England']:
        return 'South of England'
    if region == 'Greater London':
        return 'Greater London'

data['region'] = data['region'].apply(group_region)
data['region'].head()
```

Out[27]:
```
0    North of England
1    North of England
2    South of England
3            Midlands
4    South of England
Name: region, dtype: object
```

4.2.2 Banding birthdate

As discussed before, the **'birthdate'** column contains sensitive information and it is a quasi-identifier[7]. Therefore, we converted it into **'age'**, which has less information[8].

In [28]:
```
# define the function 'calculate_age'

def calculate_age(birth_date):
    now = pd.Timestamp('now')
    now_year, now_month, now_day = now.year, now.month, now.day

    # Convert the birth_date to a datetime object, so that we can perform manipulation
    birth_date = pd.to_datetime(birth_date, dayfirst=True)

    # Extract year, month, and day from the birth date
    birth_year, birth_month, birth_day = birth_date.year, birth_date.month, birth_date.day

    # Give an initial value for age by subtracting the birth year from the current year
    age = now_year - birth_year - 1
    # Check if the current month and day are greater than or equal to the birth month and day
    if now_month >= birth_month:
        if now_day >= birth_day:
            age = now_year - birth_year + 1
    return(age)

data['age'] = data['birthdate'].apply(calculate_age)
```

```
data['age'].head()
```

Out[28]:
```
0    40
1    25
2    32
3    22
4    56
Name: age, dtype: int64
```

We shared age with the researchers. However, for the dataset we shared with the government, we banded **age** into **age groups** to boost k-anonymity and reduce the risk of an individual being identified.

In [29]:
```
print(data['age'].describe())  # min age is 19, max age is 68

# defining a function for banding age

def categorise_age(age):
    if 18 <= age < 30:
        return '[18, 30)'
    if 30 <= age < 40:
        return '[30, 40)'
    if 40<= age < 50:
        return '[40, 50)'
    if 50<= age < 60:
        return '[50, 60)'
    return '[60, 70)'


data['age_categories'] = data['age'].apply(categorise_age)
```

```
count    1000.000000
mean       44.220000
std        14.032053
min        19.000000
25%        32.000000
50%        44.000000
75%        56.000000
max        68.000000
Name: age, dtype: float64
```

4.2.3 Banding Education Level

Since education level is a quasi-identifier, we banded this information. Those with primary or secondary education have been grouped into the "primary or secondary" category. Those with 'others' listed as their education level were also grouped into the "primary or secondary" category as we assumed that they would have at least completed primary education. Those with tertiary education or higher were grouped into the category "bachelor or higher".

In [30]:
```
def categorise_education_level(education_level):
    if education_level == 'primary' or education_level == 'secondary' or education_level == 'other' :
        return 'primary or secondary'
    return 'bachelor or higher'

# The categorise_education_level function is applied to the education level of each individual
#within the dataset and the outputs are stored in a new column titled 'education_level_categories'.

data['education_level_categories'] = data['education_level'].apply(categorise_education_level)
data['education_level_categories'].head()
```

```
Out[30]: 0      bachelor or higher
         1    primary or secondary
         2      bachelor or higher
         3    primary or secondary
         4    primary or secondary
         Name: education_level_categories, dtype: object
```

5. Creating our relations/tables

5.1 Creating the government table

For this table, we emphasised anonymity (as opposed to utility), because this dataset will be published. We dropped all sensitive identifiers, direct identifiers, and non-banded quasi-identifiers (except for gender, which cannot be banded beyond 2 groups for it to be an informative variable), to increase k-anonymity.

We preserved the hash for customer re-identification, and all other columns that can inform customer characteristics.

We therefore selected the columns below:

- id
- gender
- age_categories
- region
- education_level_categories
- height
- weight
- avg_n_drinks_per_week
- avg_n_cigret_per_week

- n_countries_visited
- cc_status

In [33]:
```
gov_dataset = data[['id',
                    'gender',
                    'age_categories',
                    'region',
                    'education_level_categories',
                    'height',
                    'weight',
                    'avg_n_drinks_per_week',
                    'avg_n_cigret_per_week',
                    'n_countries_visited',
                    'cc_status'
                   ]]
```

5.2 Creating the researchers table

For this table, we sacrificed k-anonymity to maximise utility, as researchers are trusted collaborators who are not at liberty to publish the data. We nonetheless dropped sensitive and direct identifiers as they provide no utility for their stated research aim. Quasi-identifiers which do not inform lifestyle habits, such as blood type, are likewise dropped for the same reason. However, alongside the hash identifier, we retain quasi-identifiers which could inform lifestyle habits in maximum granularity, to empower the researchers to perform the most precise and informative research possible.

We therefore selected the columns below:

- id
- gender
- age
- height

- weight
- postcode_area
- country_of_birth
- education_level
- avg_n_drinks_per_week
- avg_n_cigret_per_week
- n_countries_visited
- cc_status

In [34]:
```
researchers_dataset = data[['id',
                            'gender',
                            'age',
                            'postcode_area',
                            'height',
                            'weight',
                            'country_of_birth',
                            'education_level' ,
                            'avg_n_drinks_per_week',
                            'avg_n_cigret_per_week',
                            'n_countries_visited',
                            'cc_status'
                           ]]
```

5.3 Creating the key table

A table with the hash identifier, alongside the sensitive and direct identifiers dropped in the aforementioned tables, can be used to re-identify customers as needed. Only we will have the authority to do this. To create this table, we dropped all columns except the hash identifier and all sensitive and direct identifiers other than bank account number whose utility was not apparent.

We therefore selected the columns below:

- id
- given_name
- surname
- phone_number
- national_insurance_number

As stated above, the datasets that we are sharing with the researchers and the government also contain the hash identifier referencing this table.

In [35]:
```
private_dataset = data[['id',
                        'given_name',
                        'surname',
                        'phone_number',
                        'national_insurance_number'
                       ]]
```

6. k-anonymity

k-anonymity represents how easily re-identifiable subjects in the dataset are. If a dataset has a k-anonymity of k, then each subject in the dataset has the same combination of quasi-identifiers as at least k-1 other subjects.

k-anonymity is obtained by identifying all distinct combinations of quasi-identifiers in the dataset and taking the number of occurrences of the combination that appears the least[9].

We calculated k-anonymity separately for the government table and for the researchers table.

We excluded from both calculations all biometric data that can change between measurements and all self-reported information that cannot be easily replicated and will vary over time. Hence, we didn't consider the following variables to be quasi-identifiers:

- height

- weight
- avg_n_drinks_per_week
- avg_n_cigret_per_week
- n_countries_visited

In [36]:
```
# defining a function that analyses combination occurrences
def calculate_k_anonymity(data, quasi_identifiers):
    # grouping combination of quasi-identifiers
    combinations_occurrences = data.groupby(quasi_identifiers).size()
    df = pd.DataFrame(combinations_occurrences, columns=['Count'])

    value_counts = df['Count'].value_counts().sort_index(ascending=True)

    # exporting both k (minimum number of occurrences)
    # and the number of times each number of occurrences repeats
    return df['Count'].min(), value_counts

researchers_quasi_identifiers = ['gender',
                                 'age',
                                 'postcode_area',
                                 'country_of_birth',
                                 'education_level'
                                ]
gov_quasi_identifiers = ['gender',
                         'age_categories',
                         'region',
                         'education_level_categories'
                        ]

# calculating k-anonimity for the researchers table

researchers_k_anon, researchers_value_counts = calculate_k_anonymity(
    data, researchers_quasi_identifiers)
print('The k-anonymity for the dataset shared with the researchers is', researchers_k_anon, '.')
```

```
# print(researchers_value_counts)
# for col in researchers_quasi_identifiers:
#     print(data[col].value_counts())  # used in development to define the best banding strategies

# calculating k-anonimity for the government table

gov_k_anon, gov_value_counts = calculate_k_anonymity(
    data, gov_quasi_identifiers)
print('The k-anonymity for the dataset shared with the government is', gov_k_anon, '.')
# print(gov_value_counts)
# for col in gov_quasi_identifiers:
#     print(data[col].value_counts())  # used in development to define the best banding strategies
```

```
The k-anonymity for the dataset shared with the researchers is 1 .
The k-anonymity for the dataset shared with the government is 3 .
```

7. Previewing each dataset

In [37]:
```
# Run the code below to preview each dataset.
print('researchers_dataset')
print(researchers_dataset.head())
print('government_dataset')
print(gov_dataset.head())
print('private_dataset')
print(private_dataset.head())
```

```
researchers_dataset
                                                  id gender  age  \
0  b7d8d9b1c55af131072257d9daa73e06ca3f8b5861c93b...      F   40
1  a67df2cc43c5384aef155987b8ceacff5f260f4378db7f...      M   25
2  cfb54b02b9d870f9ee990080ad80ef77e02e208d8d476c...      F   32
3  3805bf1c1b04d84e7d1e56459a1325055c84ebbdd1a51c...      F   22
4  342d728e9458dea139eca4c251660a617eb53b5d08b235...      F   56

  postcode_area  height  weight          country_of_birth education_level  \
```

```
0         Leeds    1.73    74.2                  Armenia             phD
1    Manchester    1.74    69.4  Northern Mariana Islands         primary
2   Southampton    1.88    98.6                Venezuela        bachelor
3    Birmingham    1.56    62.0                  Eritrea         primary
4       Torquay    1.81    96.3                  Ecuador       secondary

   avg_n_drinks_per_week  avg_n_cigret_per_week  n_countries_visited  \
0                    6.5                  218.8                   48
1                    0.7                   43.6                   42
2                    7.8                   59.1                    9
3                    4.6                  284.2                   32
4                    4.4                  348.8                   34

   cc_status
0          0
1          0
2          0
3          0
4          0
government_dataset
                                                  id gender age_categories  \
0  b7d8d9b1c55af131072257d9daa73e06ca3f8b5861c93b...      F       [40, 50)
1  a67df2cc43c5384aef155987b8ceacff5f260f4378db7f...      M       [18, 30)
2  cfb54b02b9d870f9ee990080ad80ef77e02e208d8d476c...      F       [30, 40)
3  3805bf1c1b04d84e7d1e56459a1325055c84ebbdd1a51c...      F       [18, 30)
4  342d728e9458dea139eca4c251660a617eb53b5d08b235...      F       [50, 60)

             region education_level_categories  height  weight  \
0  North of England         bachelor or higher    1.73    74.2
1  North of England       primary or secondary    1.74    69.4
2  South of England         bachelor or higher    1.88    98.6
3         Midlands       primary or secondary    1.56    62.0
4  South of England       primary or secondary    1.81    96.3

   avg_n_drinks_per_week  avg_n_cigret_per_week  n_countries_visited  \
```

```
0                   6.5                  218.8                   48
1                   0.7                   43.6                   42
2                   7.8                   59.1                    9
3                   4.6                  284.2                   32
4                   4.4                  348.8                   34

   cc_status
0          0
1          0
2          0
3          0
4          0
private_dataset
                                                 id given_name   surname  \
0  b7d8d9b1c55af131072257d9daa73e06ca3f8b5861c93b...   Lorraine      Reed
1  a67df2cc43c5384aef155987b8ceacff5f260f4378db7f...     Edward  Williams
2  cfb54b02b9d870f9ee990080ad80ef77e02e208d8d476c...     Hannah    Turner
3  3805bf1c1b04d84e7d1e56459a1325055c84ebbdd1a51c...  Christine   Osborne
4  342d728e9458dea139eca4c251660a617eb53b5d08b235...  Francesca     Yates

      phone_number national_insurance_number
0   (07700) 900876             ZZ 19 48 92 T
1  (07700) 900 877               ZZ 753513 T
2   +447700 900148               ZZ 947196 T
3   +447700 900112             ZZ 39 69 47 T
4    07700 900 413             ZZ 30 98 91 T
```

8. Exporting the datasets

We exported each dataset for sharing into .xlsx files because these can be password-encrypted. The data users can then convert these to csv files for loading into statistical software as needed.

We exported our key dataframe directly into a csv file.

In [38]:
```
researchers_dataset.to_excel('researchers_dataset.xlsx', index=False)
gov_dataset.to_excel('governament_dataset.xlsx', index=False)
private_dataset.to_csv('private_dataset.csv', index=False)
```

9. References

1. Oaic. What is personal information? [Internet]. 2023 [cited 2023 Dec 14]. Available from: https://www.oaic.gov.au/privacy/your-privacy-rights/your-personal-information/what-is-personal-information#:~:text=Sensitive%20information%20is%20personal%20information,religious%20or%20philosophical%20beliefs
2. Welcome to GeoPy's documentation![Internet]. [cited 2023 Dec 14]. Available from: https://geopy.readthedocs.io/en/stable/#nominatim
3. Monash. PRIVACY – KEY DEFINITIONS[Internet]. [cited 2023 Dec 14]. Available from: https://www.monash.edu/privacy-monash?a=1358299
4. Stec A. Deep dive into hashing [Internet]. 2023 [cited 2023 Dec 14]. Available from: https://www.baeldung.com/cs/hashing
5. Chapter 3: pseudonymisation [Internet]. [cited 2023 Dec 14]. Available from: https://ico.org.uk/media/about-the-ico/consultations/4019579/chapter-3-anonymisation-guidance.pdf
6. Devane H. Everything you need to know about K-anonymity [Internet]. 2023 [cited 2023 Dec 14]. Available from: https://www.immuta.com/blog/k-anonymity-everything-you-need-to-know-2021-guide/
7. van Tilborg HCA, de Capitani di Vimercati S, Foresti S. Quasi-Identifier. In: Encyclopedia of cryptography and security. 2nd ed. New York: Springer; 2011. p. 1010–1.
8. Tasdelen I. Current age calculator by date of birth with python [Internet]. Medium; 2023 [cited 2023 Dec 14]. Available from: https://ismailtasdelen.medium.com/current-age-calculator-by-date-of-birth-with-python-23c4b845cef4
9. L. Sweeney. k-anonymity: a model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5), 2002; 557-570.

In [ ]:
```

```