

Dataset.Anonymisation

December 14, 2022

1 Anonymisation of ‘customer_information.csv’ dataset for use by Imperial researchers and government (and calculation of K-anonymity)

Anonymisation is the practice of removing identifying information from data in order to protect individuals’ privacy. By anonymising data, we can ensure that sensitive information is kept secure. In this project, we aim to create an anonymised dataset by removing personally identifiable information from the original dataset whilst attempting to retain its utility and insights by using a combination of techniques such as pseudonymous identification and data perturbation.

1.1 Helper functions

The following helper functions are needed:

```
[ ]: # Helper functions

# The following variable countries were hard-coded to fix unmatched territory
↳errors
northern_countries = ["Svalbard & Jan Mayen Islands"]
southern_countries = ["Micronesia"]

# Parse country into shortform
def parse_country(country_name):
    return coco.convert(country_name, to='name_short', include_obsolete=True)

# Convert country of birth into Hemisphere (Northern or Southern) based on
↳latitude coordinates
def country_to_hemisphere(country_name):
    try:
        if country_name in southern_countries:
            return "Southern Hemisphere"
        elif country_name in northern_countries:
            return "Northern Hemisphere"
        else:
            return ("Southern" if Nominatim(user_agent="CDM").
↳geocode(parse_country(country_name)).latitude < 0 else "Northern") + "
↳Hemisphere"
```

```

except Exception as e:
    print(e)
    return "Error"

# SHA hash function using a key and salt
def hash(to_hash, key):
    salt = os.urandom(16)
    h = hashlib.sha256()
    h.update(key)
    h.update(salt)
    h.update(to_hash.encode())
    return to_hash, h.hexdigest(), salt.hex()

# To encrypt and save as encrypted file; specify file to encrypt, encrypted_
↳ file destination, and destination key location
def encrypt(to_encrypt, file_destination, key_location):
    key = Fernet.generate_key() # AES in CBC mode with a 128-bit key for_
↳ encryption
    fernet = Fernet(key)

    with open(key_location, 'wb') as f:
        f.write(key)

    with open(to_encrypt, 'rb') as f:
        plaintext = f.read()

    encrypted = fernet.encrypt(plaintext)
    with open(file_destination, 'wb') as e:
        e.write(encrypted)

```

1.2 Loading required data and creating the anonymised dataframe

The original dataset is loaded, an empty output Dataframe is initialised, and a postcode reference table is loaded

```

[ ]: # Read in data to be anonymised
original_data = pd.read_csv("Data/customer_information.csv")

# Reading in postcode_region.csv to map given postcode to countries in the UK -
↳ 'England' and 'Other'(includes Wales, Scotland, Northern Ireland)
postcode_dictionary = pd.read_csv('Data/postcode_region.csv')

# Create anon_data variable as initial data with unneeded direct identifiers_
↳ dropped
anon_data = pd.DataFrame()

postcode_dictionary.head()

```

```
[ ]:   Postcode   Region
      0        AB    Other
      1        AL  England
      2         B   England
      3        BA   England
      4        BB   England
```

1.3 Adding variables to the anonymised dataset

Assigning gender and case-control status as given

```
[ ]: # Assign gender
anon_data['Gender'] = original_data['gender']

# Assign case-control status
anon_data['CC.Status'] = original_data['cc_status']

anon_data.head()
```

```
[ ]:   Gender  CC.Status
      0      F         0
      1      M         0
      2      F         0
      3      F         0
      4      F         0
```

2 Anonymisation

2.1 Pseudonymisation with hashed Sample ID

Next, a unique Sample ID is created from the National Insurance Number to link the anonymised data with the reference data containing sensitive information.

```
[ ]: # Clean NIN formatting and assign Sample ID as a hashed form of the NIN
key = os.urandom(16)
original_data["national_insurance_number"], anon_data['Sample.ID'], salts = zip(
    *original_data["national_insurance_number"].apply(
        lambda x: hash(re.sub(r'(.{2})(?!$)', '\\1 ', x.replace(' ', '')) ), key)))

anon_data.head()
```

```
[ ]:   Gender  CC.Status   Sample.ID
      0      F         0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
      1      M         0  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
      2      F         0  dd3daf05200664054f9162a61dd76d64802bff56437e20...
      3      F         0  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
      4      F         0  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...
```

```
[ ]: # Create a reference table between NIN and respective hashed NIN
reference_table = pd.DataFrame()
reference_table['Hashed.NIN'] = anon_data['Sample.ID']
reference_table['Salt'] = salts
reference_table['Key'] = key.hex()
reference_table['NIN'] = original_data['national_insurance_number']

reference_table.head()
```

```
[ ]:                                     Hashed.NIN \
0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
1  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
2  dd3daf05200664054f9162a61dd76d64802bff56437e20...
3  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
4  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...

                                     Salt                                     Key \
0  ae064c90392c2d5c545da74d82c2ae52  ae823853bfce7c66904c0de363de9b2f
1  af9e92525c444b408b30a920c3e3fd1a  ae823853bfce7c66904c0de363de9b2f
2  b735f1a9a61d9716d1dbed1db632d9c4  ae823853bfce7c66904c0de363de9b2f
3  dcb27a563ea605225fc2bccdb4e7c6ea  ae823853bfce7c66904c0de363de9b2f
4  8907a93f3a4de5bdf6dc3fd1d6de8337  ae823853bfce7c66904c0de363de9b2f

                                     NIN
0  ZZ 19 48 92 T
1  ZZ 75 35 13 T
2  ZZ 94 71 96 T
3  ZZ 39 69 47 T
4  ZZ 30 98 91 T
```

2.2 Banding

2.2.1 Date of birth and education level

```
[ ]: # Banding birth date
birthyears = pd.DatetimeIndex(original_data['birthdate']).year

# Band the birth years into 20-year intervals
anon_data['Birthyear'] = pd.cut(birthyears, np.arange(birthyears.min(),
↳ birthyears.max()+20, 20), right=False)

anon_data.head()
```

```
[ ]:   Gender  CC.Status                                     Sample.ID \
0      F           0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
1      M           0  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
2      F           0  dd3daf05200664054f9162a61dd76d64802bff56437e20...
```

```

3      F      0  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
4      F      0  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...

```

```

      Birthyear
0  [1975, 1995)
1  [1995, 2015)
2  [1975, 1995)
3  [1995, 2015)
4  [1955, 1975)

```

2.2.2 Full postcode to countries within the UK

```

[ ]: # Assign UK country derived from postcode
anon_data['Postcode'] = original_data['postcode'].apply(lambda x: re.
↳search('[a-zA-Z]*', x).group(0))
anon_data = pd.merge(anon_data, postcode_dictionary, on='Postcode', how='left')
anon_data = anon_data.rename(columns={'Region': 'UK.Country'})

anon_data.head()

```

```

[ ]:   Gender  CC.Status                                     Sample.ID \
0      F      0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
1      M      0  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
2      F      0  dd3daf05200664054f9162a61dd76d64802bff56437e20...
3      F      0  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
4      F      0  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...

```

```

      Birthyear Postcode UK.Country
0  [1975, 1995)      LS  England
1  [1995, 2015)      M  England
2  [1975, 1995)      SO  England
3  [1995, 2015)      B  England
4  [1955, 1975)      TQ  England

```

2.2.3 Education level and country of birth

```

[ ]: # Assign education level as banded education level
anon_data['Education.Level'] = original_data['education_level'].map(lambda x:
↳"Higher" if x in ["bachelor", "masters", "phD"] else "BasicOther")

# Assign hemisphere of birth depending on country of birth
anon_data['Location.of.Birth'] = original_data['country_of_birth'].apply(lambda
↳x: country_to_hemisphere(x))

anon_data.head()

```

```
[ ]:  Gender  CC.Status                                     Sample.ID  \
0      F      0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
1      M      0  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
2      F      0  dd3daf05200664054f9162a61dd76d64802bff56437e20...
3      F      0  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
4      F      0  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...

      Birthyear Postcode UK.Country Education.Level  Location.of.Birth
0  [1975, 1995)      LS   England           Higher Northern Hemisphere
1  [1995, 2015)      M   England      BasicOther Northern Hemisphere
2  [1975, 1995)      SO   England           Higher Northern Hemisphere
3  [1995, 2015)      B   England      BasicOther Northern Hemisphere
4  [1955, 1975)      TQ   England      BasicOther Southern Hemisphere
```

2.3 Data perturbation (adding Gaussian noise)

```
[ ]: # Add gaussian noise to weight, height, countries visited, average number of
      ↪drinks in alcohol units per week and average cigarettes smoked per week.
weight_noise = np.random.normal(0,1,1000)*5
anon_data['Weight'] = round(original_data['weight']+weight_noise, 1)

height_noise = np.random.normal(0,1,1000)/5
anon_data['Height'] = round(original_data['height']+height_noise, 2)

countries_noise = np.random.normal(0,1,1000)*5
anon_data['Countries.Visited'] =
      ↪round(original_data['n_countries_visited']+countries_noise)

alcohol_noise = np.random.normal(0,1,1000)
anon_data['Avg.Alcohol'] =
      ↪round(original_data['avg_n_drinks_per_week']+alcohol_noise, 1)

smoking_noise = np.random.normal(0,1,1000)*20
anon_data['Avg.Cigarettes'] =
      ↪round(original_data['avg_n_cigret_per_week']+smoking_noise)

anon_data.head()
```

```
[ ]:  Gender  CC.Status                                     Sample.ID  \
0      F      0  717341b0d3455426af5043290ec12b0681175d0b2da21a...
1      M      0  ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...
2      F      0  dd3daf05200664054f9162a61dd76d64802bff56437e20...
3      F      0  a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...
4      F      0  74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...

      Birthyear Postcode UK.Country Education.Level  Location.of.Birth  \
```

0	[1975, 1995)	LS	England	Higher	Northern Hemisphere
1	[1995, 2015)	M	England	BasicOther	Northern Hemisphere
2	[1975, 1995)	SO	England	Higher	Northern Hemisphere
3	[1995, 2015)	B	England	BasicOther	Northern Hemisphere
4	[1955, 1975)	TQ	England	BasicOther	Southern Hemisphere

	Weight	Height	Countries.Visited	Avg.Alcohol	Avg.Cigarettes
0	73.5	1.58	39.0	4.3	208.0
1	66.9	1.52	43.0	1.5	50.0
2	92.0	1.84	7.0	6.3	61.0
3	58.9	1.84	33.0	4.4	261.0
4	101.9	1.74	29.0	4.9	348.0

3 Calculating K-anonymity using quasi-identifiers

The following code groups the quasi-identifiers specified and returns a count of the “unique” rows.

```
[ ]: # Checking k-anonymity for quasi-identifiers
df_count = anon_data.groupby(['Gender', 'Birthyear', 'Location.of.Birth',
                              'UK.Country', 'Education.Level']).size().
    ↪reset_index(name = 'Count')

# Print rows where k-anonymity is 1
print("There are " + str(len(df_count[df_count['Count']==1])) + " unique_
    ↪quasi-identifier permutations.\n")

# Printing the final grouped output
print("Least-frequent quasi-identifier permutations, in ascending order:")
df_count.sort_values("Count")[df_count.sort_values("Count")['Count'] > 0].head()
```

There are 0 unique quasi-identifier permutations.

Least-frequent quasi-identifier permutations, in ascending order:

	Gender	Birthyear	Location.of.Birth	UK.Country	Education.Level	Count
47	M	[1995, 2015)	Southern Hemisphere	Other	Higher	2
31	M	[1955, 1975)	Southern Hemisphere	Other	Higher	2
46	M	[1995, 2015)	Southern Hemisphere	Other	BasicOther	2
19	F	[1995, 2015)	Northern Hemisphere	Other	Higher	2
39	M	[1975, 1995)	Southern Hemisphere	Other	Higher	3

4 Viewing and saving the anonymised dataset

```
[ ]: # Re-order columns
anon_data = anon_data[['Sample.ID', 'Gender', 'Birthyear', 'Location.of.Birth', 'UK.Country', 'Weight',
                        'Height', 'Education.Level', 'Avg.Alcohol', 'Avg.Cigarettes', 'Countries.Visited', 'CC.Status']]

# View the anonymised dataset
anon_data.head()
```

```
[ ]:
```

	Sample.ID	Gender	Birthyear	\
0	717341b0d3455426af5043290ec12b0681175d0b2da21a...	F	[1975, 1995)	
1	ff878150b526edf6a15e3bcc58e6a30bd2852f5dbef10c...	M	[1995, 2015)	
2	dd3daf05200664054f9162a61dd76d64802bff56437e20...	F	[1975, 1995)	
3	a17d9326ef0ada45cfbc51fdf59c70129813f7c4bf2421...	F	[1995, 2015)	
4	74e8b816673182819be3ba63b609ff22fb9b8f8a28426f...	F	[1955, 1975)	

	Location.of.Birth	UK.Country	Weight	Height	Education.Level	\
0	Northern Hemisphere	England	73.5	1.58	Higher	
1	Northern Hemisphere	England	66.9	1.52	BasicOther	
2	Northern Hemisphere	England	92.0	1.84	Higher	
3	Northern Hemisphere	England	58.9	1.84	BasicOther	
4	Southern Hemisphere	England	101.9	1.74	BasicOther	

	Avg.Alcohol	Avg.Cigarettes	Countries.Visited	CC.Status
0	4.3	208.0	39.0	0
1	1.5	50.0	43.0	0
2	6.3	61.0	7.0	0
3	4.4	261.0	33.0	0
4	4.9	348.0	29.0	0

4.1 Creating CSV files for the anonymised data and the reference table

```
[ ]: # Output the files into .csv format
output_name = "anon_dataset"
anon_data.to_csv(output_name + ".csv", sep=";", index=None)

reference_table.to_csv("reference_table.csv", sep=";", index=None)
```

4.2 Encrypting the dataset

```
[ ]: # Encrypt csv and delete original file
encrypt(output_name + ".csv", output_name + "_encrypted.csv", "key.key")
os.remove(output_name + ".csv")
```



```
[ ]: pip freeze > requirements.txt
```

Note: you may need to restart the kernel to use updated packages.