Máster Interuniversitario en Ciencia de Datos

CONSTRUYENDO UNA LIBRERÍA DE PYTHON PARA ANONIMIZAR DATOS SENSIBLES
(Building a Python library for anonymizing sensitive data)

Trabajo de Fin de Master para acceder al

MÁSTER EN CIENCIA DE DATOS

Autora: Esmeralda Madrazo Quintana

Directores: Álvaro López García y Judith Sáinz-Pardo Díaz

Septiembre - 2023

Abstract

Technologies that handle large amounts of data have experienced rapid growth in recent years, thanks mainly to the easy availability of large volumes of data (big data). Problems arise when trying to maintain the balance between privacy and preserving as much information as possible. The dilemma of privacy preservation is further intensified when handling databases containing, for example, clinical patient data. The objective of this master's thesis is to address privacy issues in data science by exploring and implementing the most common anonymization techniques. More specifically, we intend to implement a Python library with the most popular anonymization models, more specifically *k*-anonymity, *l*-diversity and *t*-closeness, as well as offer some performance analysis techniques for its optimal implementation.

Key words: k-anonymity, l-diversity, t-closeness, python library, privacy, sensitive data

Resumen

Las tecnologías encargadas de manejar grandes cantidades de datos han experimentado un crecimiento rápido en los últimos años, mayoritariamente gracias a lo facilmente accesible que se han vuelto los grandes volumenes de datos (big data). Por ello han surgido problemas a la hora de tratar de mantener un equilibrio entre la privacidad y la preservación de tanta información como sea posible. Este dilema se ve intensificado cuando tratamos con bases de datos que contienen, por ejemplo, datos médicos de un paciente. El objetivo de este trabajo de fin de Master es tratar de ofrecer una solución mediante la implementación de algunas de las técnicas más comunes de anonimización. Más especificamente, nuestra intención es implentar una librería de Python que contenga algunos de los modelos de anonimización más populares, concretamente *k*-anonymity, *l*-diversity y *t*-closeness, así como ofrecer una serie de métricas de análisis para su óptima implementación.

Palabras clave: k-anonimato, l-diversidad, t-closeness, librería de python, privacidad, datos sensibles

Contents

List of Figures

Introduction

1.1 Motivation

The large amount of data that is being collected and disseminated globally raises several concerns regarding users' privacy. These challenges become even more critical when taking into account all the different professional areas that requires elevated numbers of data to operate. This rises several issues, in part due to the at times limited anonymization treatment given to such data before publishing them.

Generally speaking, it would be assumed that, in order to ensure users' privacy, there are several tools which aid corporations to satisfy those safety and privacy requirements when disseminating data. This could not be further from the truth, as only a small part of this software which is simple to use is available for the general public. As a result, Privacy-Preserving Data Publishing (PPDP) has become quite an active area of research, in which multiple techniques have been proposed, including data anonymization. On top of that, a multitude of implementations have been created with the objective of serving as convenient tools for both big and small corporations or institutions to protect their users privacy.

The main issue comes when we realize that some of these implementations are not openly available, and those that are tend to have convoluted interfaces or only implement a few number of techniques, which are only specific to certain cases of use, and that do not cater to all the multitude of unique and widely diverse fields in which data anonymization is used.

When talking about securing users' privacy, we refer to ensuring that datasets containing sensible information about them fulfill privacy guarantees, that is, preventing a potential attacker from accessing and extracting sensitive information about them. For example, if we are talking about data in the field of clinical research, this type of sensitive data could be information such as patients' medical history or even mental health records. Some subsets of this type of data, such as names, are identifying fields that put the patience privacy at risk and must be suppressed, as they allow the patient to be fully identified. These variables are known as ***identifiers*** [10] On the other hand, data like age and gender, although they seem to be non-identifying variables, can also make identification possible when combined together, that's why we call these type of parameters ***quasi-identifiers (QI)*** [10]. It is important to take into account that some fields who may belong to either of these classes might be needed in a particular study or analysis, that is why we may need to maintain them as they are or apply generalizations on them instead of suppressing them directly as should be done with identifiers. Last but not least, studies using user data have a focus or aim in mind, which can be directed to a certain number of columns from the datasets used during it, which hold the private information that must remain hidden from any attackers and can not be extracted. These fields are called ***sensitive attributes (SA)*** [10].

The aim of this project is to implement a convenient and easy-to-use Python library which can provide users with some well-known classical tools to easily anonymize their data to preserve users' privacy. In addition, the tools provided by this library also aim to be as widely usable as possible, given that data anonymization can be used in a multitude of different fields. One of the main reasons for using Python is it's huge developer community, which spans different areas and fields of work and could definitely benefit from having a handy tool for anonymization in their openly available libraries.

1.2 Objectives

Due to the aforementioned reasons, the main objectives of this project are as follows:

1. **Implementing different algorithms for *k*-anonymization and comparing them in terms of different quality metrics.** Selecting the most appropriate algorithm for a given use scenario can be quite challenging, since there are increasing number of use cases, but also of algorithms and generalization hierarchies that can be applied for each of those situations. In this case, we have selected two algorithms which use different strategies for anonymization and which can be evaluated with similar metrics and in the same framework, which allows a fairer comparison.

2. **Implementing l-diversity.** Another important aspect to take into account when talking about anonymization is how diverse the ***sensitive attributes (SA)*** are. These type of attributes can render previously anonymized data recognisable. Therefore, certain techniques have to be applied to the *k*-anonymized database to ensure a certain level of diversity in the SA that does not compromise the security of the final data relished.

   That is the main reason for the implementation of the technique known as *l*-diversity [9], to address the issue by ensuring privacy even when the publisher of such data ignores any knowledge that any possible attacker might have regarding it.

3. **Implementing t-closeness.** It is important to remember that all this techniques have is limitations and there are certain attacks that they can not protect against.

That is why, to avoid any breaches on sensitive data due to attackers inferring values due to the distribution of the resulting table after applying a technique such as l-diversity, t-closeness [7] is a helpful extension to mitigate a maelstrom of possible breaches not addressed by other techniques.

4. **Integrating all the previous techniques into a Python library.** Due to the vast number of programming languages and tools openly available and the aim for this project to be as accessible and useful as possible, we decided to make it available in one of the most commonly used languages available: Python.

   Given that in languages like Java there already exist tools that fulfill a similar function and given the modular nature and community driven focus on packages and libraries that Python has, the final decision was to encapsulate the results of this project into a library that will be available for any Python user. It's also worth mentioning that we are currently not aware of any other Python library that incorporates all the functionalities mentioned above, as most of the ones that can currently be found include just incomplete versions of one or two techniques or libraries that can check anonymity but do not apply it.

1.3 State of the art

Privacy is quite a critical aspect in any society, and is protected as such in multiple legal frameworks, specially given the immense quantity of information that is shared nowadays. But it is due to that immense quantity of information that is being generated simultaneously that getting informed consent is often impossible in practice [8]. That is why data anonymization is a prerequisite in several fields of study, and why there is a conscious effort to improve anonymization techniques and to facilitate open source tools for this purpose.

Traditionally, data anonymization has been applied manually, given the differences between structured and unstructured data, but this process is really time and resource consuming and has its limitations. In fact, these limitations are the main reason for the rapid advancement of data anonymization and its current state of the art.

1.3.1 Types of approaches

Some of the main approaches for text annonymization that are currently employed are NLP-based Textual Anonymisation and PPDP [8], which do not fully provide a satisfactory anonymization on their own, but do have their respective merits and advantages.

1. **Current NLP-based Textual Anonymisation Approaches.** Natural Language Processing (NLP) approaches focus on sequence labelling (a type of pattern recognition task that

involves algorithmically assigning a categorical label to each member of a sequence of observed values) to recognise or suppress any section of data that my contain sensitive and personal information. Some of the current NLP-based Textual Anonymisation approaches include de-identification and detection and obfuscation.

  (a) **De-identification.** Early approaches consisted on rule-base sequence labelling, be them along or with the assistant of machine learning methods. Nowadays, the state of the art of this technique has currently evolved with the help of neural networks with character-level embeddings. Although they still have some disadvantages, such as not addressing the issue of replacing identifiers with surrogate values.

  (b) **Detection and obfuscation.** The aim of this approach is to apply sequence labelling based on groups of quasi-identifiers, such as age. The current state of the art for this technique is based on adversarial learning or encryption techniques. Although it is still limited to predefined categories and ignore any other text element that may assists attackers in identifying the owner of that data.

2. **Current PPDP Approaches.** Privacy-Preserving Data Publishing (PPDP) takes a different approach by enforcing a privacy model to anonymize the text while taking into account that there will be an inevitable disclosure risk. This privacy model can consist of several types of transformations, such as generalization and suppression, each dependant on the model itself. Some of the current PPDP approaches include K-anonymity, T-plausability, C-sanitise and differential privacy.

  (a) **K-anonymity [3].** The first approaches of this technique were made for collections of datasets and focused on k-safety, which meant that all objects in those sets which didn't comply with said metric would be suppressed before publishing the datasets. This meant that the probability of an attacker uncovering these suppressed arguments would be of 1/K. But these approach had a problem, due to how restricted it is, as the datasets would need to be homogeneous for this approach to work properly.
  That's why another approach was created, which focuses on mapping input datasets to sensitive entities with the help of a multi-class classifier, so as to attempt to replicate the possible data relationships that any potential attacker might uncover.

  (b) **T-plausibility [2].** This state of the art model for document protection generalizes sensitive terms according to a certain t-plausibility property, that is, t different plausible datasets should be able to be derived from the generalized terms.

  (c) **C-sanitise [16].** This model takes a different approach, as it characterizes risk of information disclosal as an information theoretic characterisation of disclosed semantics. With this approach, it can highlight terms that might be a risk and can better protect dynamic datasets.

(d) **Differential privacy [6].** It is a framework for ensuring the privacy of datasets using randomised algorithms for computer statistics. Thanks to them, it can ensure that this statistics cannot assist any possible attacker in obtaining any individual from the anonymized dataset. It is also worth mentioning that differential privacy can be implemented locally or globally. Locally, noise is added to individual data before it is centralized in a database. Globally, noise is added to raw data after it is collected from many individuals.

1.3.2 Current tools

There are also several tools, both private and open source, that are currently being used to anonymize datasets. To mention two notable open source examples:

1. **ARX [12].** This is one of the best known open source tools for anonymizing structured data using a broad array of anonymization methods at the user's disposal, supporting user-specific privacy models and thresholds so as to mitigate any possible attack that may lead to privacy violations. It can directly remove identifying attributes such as full name.

   Some of the supported privacy models it makes available to the user include k-anonymity, l-diversity, t-closeness, δ-disclosure, β-likeness, $\delta - presence$, k-map and (ϵ, δ)-differential privacy.

   ARX also supports arbitrary combinations of data transformations, such as global and local transformations schemes, random sampling, generalization, record, attribute and cell suppression, micro-aggregation, categorization and top- and bottom-coding [12].

   It even includes data quality models and objective functions, such as cell-oriented models, attribute-oriented models, record-oriented general-purpose models and workload-aware models.

   ARX offers booth a graphical interface, that provides various visualizations, and a Java library version of its services, which counts with an API for easier access to all its functions.

2. **PyCanon [14].** It is an open source Python library and command line interface (CLI) which offers several tools to check and assess the level of anonymity of a given dataset by employing techniques such as k-anonymity, $(\alpha$,k)-anonymity, l-diversity, entropy l-diversity, recursive (c, l)-diversity, t-closeness, basic β-likeness, enhanced β-likeness and δ-disclosure privacy.

   This library returns a complete list of the parameters that each technique covers when provided with an anonymized dataset, along with its quasi-identifiers and its sensitive attributes.

As for a sample of a private tool, one that's worth mentioning is **CloverDX**, a subscription based anonymization framework which simplifies setting up and operating anonymization and pseudonymization processes. They offer APIs for AWS, Azure and Google Cloud and their annual prices for their services go higher than 5000 $. Another one that is worth mentioning due to how influential it is on the banking scene is **AirClock**, which provides an SQL interface, which can handle all types of data, including unstructured text. Both of them, do not disclose the approaches they follow for data anonymization, as do most private tools.

1.4 Project overview

This work is structured in five separate chapters as follows:

1. This first chapter, the **introduction**, presents the motivation and main objectives of this project, an abbreviate reasoning behind the techniques chosen and an overview of the current mainstream techniques and tools available.
2. The **implementation** were there is an in-depth explanation of the general concepts of anonymized data, the algorithms that this project contains and the monitoring metrics used to check their performance.
3. The third chapter, called **project design**, were there is a conscious focus to explain the ins and outs of the making of the library, be it the unit test, the documentation and the CI/CD pipeline, along with the project availability and a brief explanation of the software and hardware used during the development of this project can be found, including any auxiliary libraries or tools, be it for generating the documentation or for packaging the library, that have been employed.
4. In the **Use Cases** chapter (chapter 4), we illustrate in detail all the different task that the library is able to perform and we outline, from the user's point of view, the systems behaviour to possible queries.
5. The final chapter of this work, the **conclusions**, wraps this project up, acknowledges the achieved objectives and gives an overview of possible additions for the library.

Implementation

In this chapter, we can find some general concepts about anonymity, an in-depth discussion of the different anonymization techniques implemented and the algorithms used for their implementation. It is important to mention that in this library, our objective is to provide tools for tabular data only, which means that this methods that will be explained in the chapter will not work for other types of data, such as images or pdfs.

2.1 Fundamental concepts

To better understand the algorithms and anonymity techniques [14] implemented, there are some key concepts that must be explained regarding tabular databases:

1. **Identifier attributes (ID).** Variables in the dataset that allow to identify an individual, for example, name, surname, address or id number. In table §2.1, which is based on our of the synthetic datasets that will be used later for testing, the field called *"Name"* is an identifier attribute, as it reveals personal information about each of the 5 individuals.

2. **Quasi-identifier attributes (QI).** Variables that, although seem to not reveal relevant information, when combined can assist in the identification of an individual. Some examples can be age, gender, religion, city, etc... In table §2.1, the fields called *"Age", "ZipCode" and "Marital-satus"* are considered quasi-identifier attributes, as they do not reveal personal information about each of the 5 individuals but can do so when combined.

3. **Sensitive attributes (SA).** Variables from the dataset which hold the private information that must be kept secured and should not be extracted by any attackers. In table §2.1, the critical sensitive attribute is clearly the *"crime"* committed by each of these five individual, as it is private information.

4. **Generalization [3].** Type of anonymization operation that consists in replacing the values of an attribute with a less specific version that is consistent with the original values. For example, in case of a field like age that contains singular values like *'24'*, *'27'*, *'35'* and *'39'*, the resulting fields after applying generalization could be *'20-30'*, *'20-30'*, *'30-40'* and *'30-40'* respectively. In table §2.1, the *"ZipCode"* column values have been generalized once by adding an '*' to their last digit, making what would initially be 5 different numbers

(for example, 39007, 38002, 37001, 38009, 37006) be now classified into 3 different values (3900*, 3800*, 3700*).

5. **Suppression.** Type of anonymization operation that consists in replacing some of the original values of the dataset with a special value that indicates that said data is not disclosed. For example, replacing a value like *'married'* with a character like '*'. In table §2.2, the last row which contained the data from user *"Luis"* has been eliminated by applying suppression.

6. **Equivalence class (EC).** Partition of a dataset in which all quasi-identifiers share the same value, which makes users indistinguishable with respect to the quasi-identifiers. In table §2.1, the *"Juan" & "Ana"* rows form an equivalence class, as they have exactly the same quasi-identifiers, which are *"22", "3800*" and "Married"*.

Name	**Age**	**ZipCode**	**Marital-status**	**Crime**
Lucia	21	3900*	Never-married	Murder
Juan	22	3800*	Married	Theft
David	26	3700*	Divorced	Traffic
Ana	22	3800*	Married	Assault
Luis	35	3700*	Widowed	Piracy

Table 2.1: Table based on the crime synthetic dataset which shows a sample of IDs, QI, SA and EQ.

Name	**Age**	**ZipCode**	**Marital-status**	**Crime**
Lucia	21	3900*	Never-married	Murder
Juan	22	3800*	Married	Theft
David	26	3700*	Divorced	Traffic
Ana	22	3800*	Married	Assault

Table 2.2: Table based on the crime synthetic dataset which shows an example of suppression.

7. **Common attacks on databases [14].** Not all techniques are useful against all types of attacks that databases can suffer nowadays, but they can complement each other when it comes to protecting users privacy. That is why it is of upmost importance to know some of the main types of attack [14] there are and in what they consists:

   (a) **Linkage.** Consists in mixing at least 2 anonymized datasets in order to uncover the identity of certain individuals who appear in both.

   (b) **Re-identification.** Consists in reversing the anonymization process.

   (c) **Homogeneity.** May happen when all the values for a sensitive attribute in an equivalence class are identical.

(d) **Background knowledge.** Consist on an attacker having previous knowledge about an individual's information before attacking.

(e) **Skewness.** May occur when there is an infrequent value for a sensitive attribute in the whole dataset but that is extremely frequent in a particular equivalence class.

(f) **Similarity.** Can happen when the values for a sensitive attribute in a certain equivalence class are written similarly, even thought they are different.

In table §2.3 we show a resume of the attacks prevented by each technique.

Technique	Linkage	Re-identification	Homogeneity	Background knowledge	Skewness	Similarity
k-anonymity	Yes	Yes				
l-diversity			Yes	Yes		
t-closeness					Yes	Yes

Table 2.3: Table which shows the attacks prevented by each technique [14].

2.2 *k*-ANONYMITY

The first tool that the library puts at the users disposal is k-anonymity [14], which will be verified when each equivalence class of the dataset has at least k rows, or, in other words, when given a certain row of the database, there are at least $k - 1$ indistinguishable rows with respect to the quasi-identifiers.

As stated in the introduction, anonymity techniques are limited when it comes to protecting users data from potential attackers. Different techniques offer protection from selected types of attacks, which is the main reason for the different selections of tools for the library. In the case of k-anonymity, the main attacks it protects from are *linkage* and *re-identification*.

For usability purposes, as stated before, we have decided to implement two different k-anonymization techniques, which use both generalization and suppression. The selection of these algorithms over any of the other available ones is mainly due to the fact that they can be evaluated using the same metrics, they are widely known, and their implementations are publicly available and easy to understand.

2.2.1 Data-fly

The first algorithm implemented has been data-fly [3], which is a greedy heuristic algorithm which performs full-domain generalization. This technique takes the non-generalized table and the QI list and checks whether or not the k-anonymity has been covered. If it has not been satisfied the next step is to try and apply suppression if available and re-check if k-anonymity is fulfilled. If it again is not, we generalize one of the QI columns following a given hierarchy and check its k-anonymity one more time. We continue with this procedure until k-anonymity is satisfied. Once it has been, we return the final anonymized table which covers the required k-anonymity. The following graphic illustrates the exact steps of the algorithm.

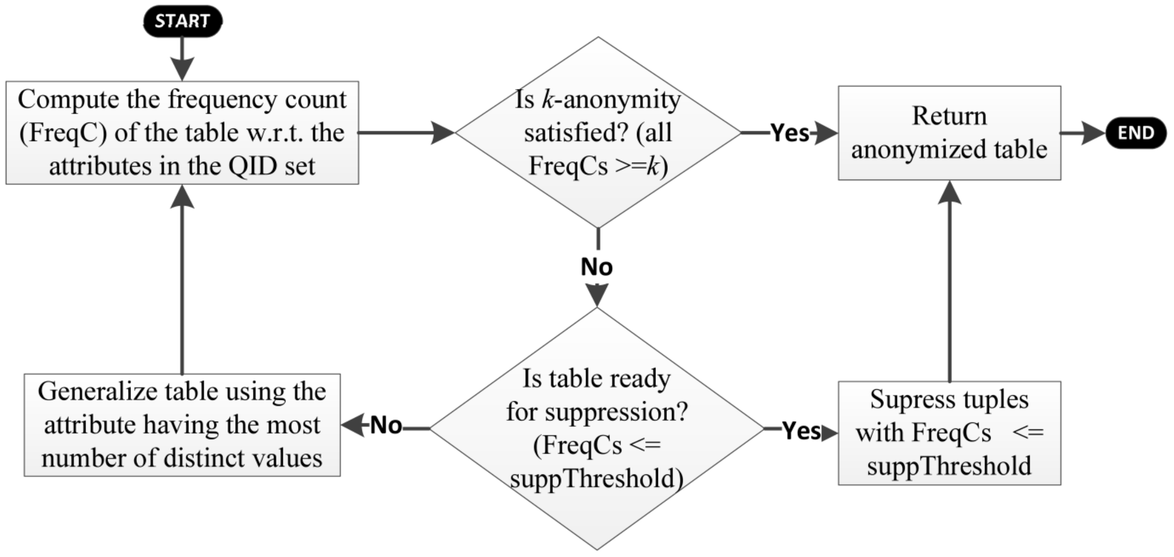

START
Compute the frequency count (FreqC) of the table w.r.t. the attributes in the QID set
Is k-anonymity satisfied? (all FreqCs >=k)
Yes
Return anonymized table
END
No
Is table ready for suppression? (FreqCs <= suppThreshold)
No
Generalize table using the attribute having the most number of distinct values
Yes
Supress tuples with FreqCs <= suppThreshold

Figure 2.1: Core process of the Data-fly algorithm [3]

This method can be considered the simplest of the two implemented in this library, as it stops as soon as *k*-anonymity is satisfied, rather than traversing all the possible options, but this also comes with the disadvantage of not being able to find the optimal solution for a particular table.

For an easier implementation of this function and all the following ones, an auxiliary module has been created. This new module contains functions which clean and treat the raw data so that the function can work properly. It also contains the generalization function which all the k-anonymity algorithms in the library use.

1. **clear_white_spaces**, which receives the raw table and returns a new table without white spaces in the column names.
2. **suppress_identifiers**, which receives a raw table and a list with the column names of the identifiers and suppresses them, returning a new table which will only contain the QI and the SA.
3. **string_to_interval**, which receives a column containing intervals as string and turn them back into interval types of data [11] so they can easily be operated with.
4. **Create_ranges**, which receives a numeric column and the step which needs to be used for the ranges. This function creates interval types for each of the values in the column,

using the given step. It returns a column which contains the original data as strings with their respective ranges.

5. **Generalization**, which receives a column that needs to be generalized, a hierarchy that will be used for its generalization and a level of generalization, which indicates the level in the hierarchy that should be used. This function has 3 separate sections depending on the type of data that needs to be generalized and it returns the generalized column.

2.2.2 Incognito

The other algorithm implemented in order to achieve *k*-anonymity ot a given value *k* has been Incognito, a single dimensional full-domain algorithm which employs a generalization lattice, a sample of which is illustrated in Figure §2.2, similar to the one illustrated below, for its generalization, traversing in from bottom to top, and looking at all the possibilities in the current level before moving up to the next one.

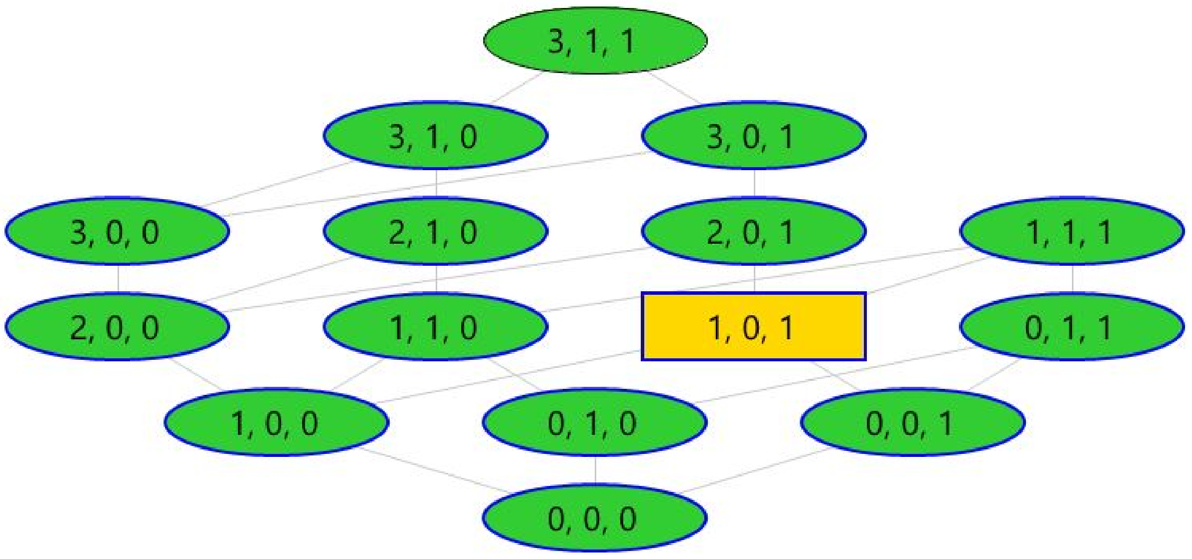


Figure 2.2: Sample lattice. Extracted from [3].

Each node of the lattice includes a value for each of the QIs of the table. The maximum value which each of them can reach is determined by the length of the generalization hierarchy set by the user. For example, if we have a class called 'Age' which has the following values, {16, 20, 22, 35, 43}, and its hierarchy goes like this 16: [15-19], [15-24], [15-49], 20: [20-24], [15-24], [15-49], 22: [20-24], [15-24], [15-49], 35: [35-39], [35-44], [15-49], 43: [40-44], [35-44], [15-49] , the minimum value for the attribute 'Age' in the lattice would be 0 and the maximum value would be 3.

While traversing the lattice, if Incognito finds a node which satisfies k-anonymity, it prunes all its direct generalizations, as they also guarantee k-anonymity. Following this, Incognito guarantees an optimal solution [3], unlike datafly.

The main steps of the algorithm, which are illustrated in figure §2.3, are as follows: a lattice is generated following all the possible permutations for each of the QIs generalizations, then we start traversing the lattice from the bottom, which is the less generalized node, to the top, checking if all the nodes have been traversed. If all the nodes have been traversed it returns the anonymized table corresponding to the optimal node (following a certain data utility metric). If not, it generalizes according to the current node, and checks if it satisfies k-anonymity, and if it does, the node is added into the list of possible nodes and all its direct generalizations are marked as already traversed. On the other hand, if it does not, suppression is tested, in case it helps satisfy k-anonymity. If it does, the node is also added to the possible node list, and if it does not satisfy k-anonymity, we mark it as traversed and move on to the next one. This process is repeated until all the lattice has been traversed.

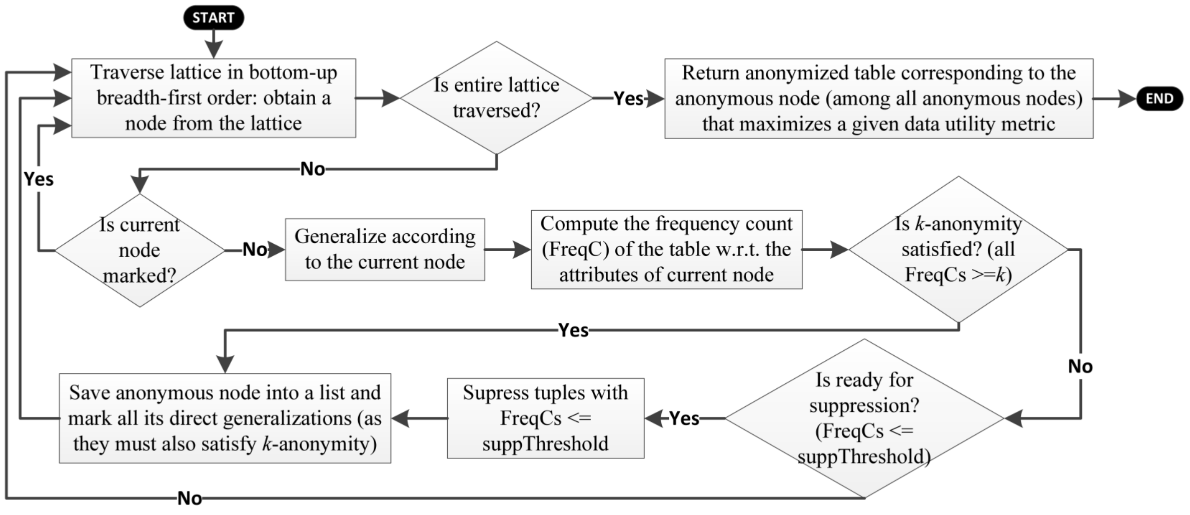

START
Traverse lattice in bottom-up breadth-first order: obtain a node from the lattice
Is entire lattice traversed?
Yes
Return anonymized table corresponding to the anonymous node (among all anonymous nodes) that maximizes a given data utility metric
END
No
Yes
Is current node marked?
No
Generalize according to the current node
Compute the frequency count (FreqC) of the table w.r.t. the attributes of current node
Is k-anonymity satisfied? (all FreqCs >=k)
Yes
No
Save anonymous node into a list and mark all its direct generalizations (as they must also satisfy k-anonymity)
Supress tuples with FreqCs <= suppThreshold
Yes
Is ready for suppression? (FreqCs <= suppThreshold)
No

Figure 2.3: Core process of the Incognito algorithm. Extracted from [3].

This implementation is closer to the one used in some of the current state of the art tools for anonymity, more specifically, ARX, which provides its users with a lattice they can use to select the exact generalization they want for each QIs. In figure §2.4 an example of the user interface of ARX sofware is shown.

2.3 ℓ-Diversity

The next tool that the library offers to its users is l-diversity [9], a technique which ensures that in each equivalence class there are at least l distinct values for a single sensitive attribute.

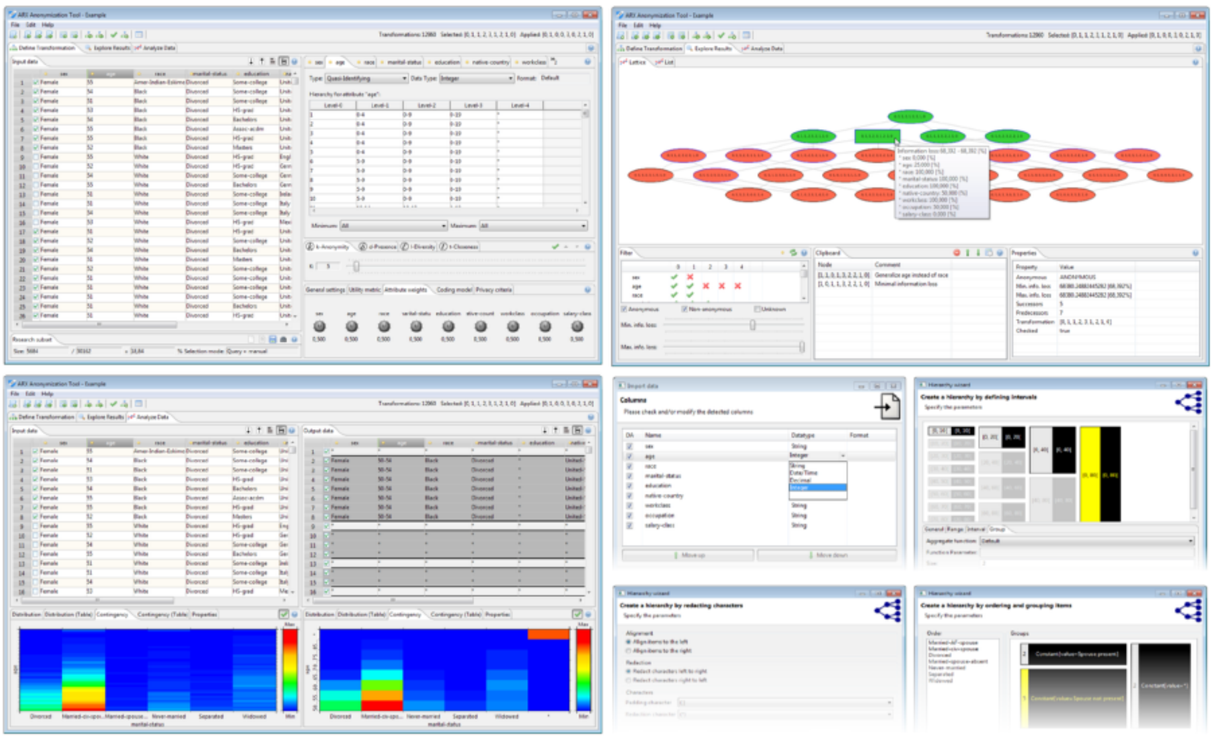


Figure 2.4: ARX interface screen-shoot.

On the security side, this technique ensures, as an extension of *k*-anonymity, an extra layer of protection on top of both k-anonymity methods, as it builds upon the security that they already offer against *linkage* and *re-identification* attacks, by protecting the dataset against *homogeneity* and *background knowledge* attacks.

This technique has a trade off, as the reduction in granularity of the data represented results in a loss of effectiveness in its use. For example, given a dataset which has two rows for each EQ, for the same set of QI, postal code = 532** and age = { 20-30 }, we will have different values of SA, disease = { flu or cancer } depending on the row, in the case of $l > 1$. Which means that , if we want to use inference with regards to the SA, the classification task will be more complex due to having different values, flu and cancer in this case, for same QI tags, 532** and 20-30 in this particular sample.

Given a more formal definition of *l*-diversity, *'let us define a q*-block as a set of tuples such that its non-sensitive values generalize to q*. A q*-block is l-diverse if it contains l "well represented" values for the sensitive attribute S. A table is l-diverse, if every q*-block in it is l-diverse.'* [9] , the implementation this technique in the library goes as follows:

1. **get_diversities**, a function which, when given the table, the SA and the QI, calculates the number of different values for each sensitive attribute in each equivalence class.
2. **get_l**, a function which, with the help of get_diversities, returns the *l*-diversity of the

whole table.

3. **apply_l_diversity**, one of the three l-diversity functions that the library offers. This version allows the user to set a single sensitive attribute as the focus of study, along with the parameters needed to apply k-anonymity, in case the table has not been previously anonymized. This function applies the l-diversity and returns an anonymized table that satisfies it.

4. **apply_l_diversity_supp**, one of the three l-diversity functions that the library offers. This version works almost exactly like the previous one but limits the number of records that can be suppressed by giving it a threshold.

5. **apply_l_diversity_multiple_sa**, is the third l-diversity function that the library offers. This function allows the user to set multiple sensitive attributes as the target of study and will attempt to satisfy l-diversity for each of them. In case of failure, it will let the user know which attribute is the one that failed.

It is important to mention that, due to how the anonymity function is implemented, this methods will not have issues running even if the introduced table has already been anonymized, as they will satisfy k-anonymity and the method will not be called.

2.4 t-Closeness

The final tool that the library provides to its users is t-closeness [7], a technique which verifies that, for each equivalence class, the distribution of values of a sensitive attribute called S are at a distance no closer than t from the distribution of said attribute in relation with the whole database.

On the security side, this technique ensures, as an extension of k-anonymity and l-diversity, an extra layer of protection on top of both methods, as it builds upon the security that they already offer against *linkage*, *re-identification*, *homogeneity* and *background knowledge*, by protecting it against *skewness* and *similarity* attacks [14].

It is important to understand that privacy is measured by the information that the user obtains from it, which can be measured by comparing the initial knowledge the user had before seeing the dataset with the new knowledge they have after. In the case of t-closeness, this *information gain* is measured in two fronts, the population as a whole and specific individuals. To illustrate this approach better we can imagine an user with some previous knowledge regarding an individual's sensitive attribute, which we will call B0. When the user sees a generalized dataset without identifiers their believed evolves due to the distribution of sensitive attributes in the table, S, and changes to B1. After that, if the user is given the released table, by knowing the values of the individual's quasi-identifiers, the observer is able to grasp the equivalence class where said individual is contained and learn the distribution of sensitive attributes, M in it, which changes the user's belief to B2 [7].

Here is where *t*-closeness comes into play, as *l*-diversity only limits the difference between B0 and B2 by requiring a certain level of diversity in M. *T*-closeness limits the difference between B1 and B2, by assuming that the distribution of the sensitive attribute in the whole dataset is public information and focusing on limiting the extent to which the user can obtain additional information about specific individuals. In other words, *t*-closeness limits the users gain from B1 to B2. It does so by limiting the distance between S and M, although a balance should be kept, as if both are too close it would limit the amount of useful information released. That is why the *t* parameter enables a balance between privacy and utility [7].

There are several function to measure that distance but the one that the library uses is *Earth Mover's distance*, which is based on the minimal amount of work needed to transform one distribution to another by moving mass between each other. In the case of this library, given that the function can receive both numerical and categorical arguments, two variants of EMD are used.

1. **EMD for numerical attributes.** Being P, the distribution of sensitive attributes in the database, and Q, a particular equivalence class, which distance we want to calculate, being p1...pi and q1...qi their elements and being $r_i = p_i - q_i, (i = 1,2,...,m)$, we can calculate the distance as follows:

$$D[P,Q] = \frac{1}{m-1}(|r_1| + |r_1 + r_2| + ... + |r_1 + r_2 + ... + r_{m-1}|) \tag{2.1}$$

2. **Equal distance for categorical attributes.** For this type of attributes we use the equal distance function, which states that the ground distance between any two values is defined by 1.

$$D[P,Q] = \frac{1}{2}\sum_{i=1}^{m}(|p_i - q_i|) \tag{2.2}$$

To help with the implementation of this technique, the following functions have been added to the library:

1. **aux_t_closeness_num**, a function that, by applying EMD for numerical attributes, returns the *t* for *t*-closeness.

2. **aux_t_closeness_str**, a function that, by applying EMD for categorical attributes, returns the *t* for *t*-closeness.

3. **get_t**, a function which returns the t value for the whole table, regardless of the type of attribute introduced, be it categorical or numerical.
4. **t_closeness**, a function that applies t-closeness to a dataset, and anonymizes it in case it already has not been anonymized.
5. **t_closeness_supp**, the other version of the t-closeness function which works almost exactly like the previous one but limits the suppression by giving it a threshold.

2.5 Monitoring metrics

Due to the complexity of these type of problems when anonymizing data and the multitude of uses cases that this library may need to assist on, we needed a way in which to measure and select the best algorithms for each technique.

The first approach was to implement general efficiency measures that should be use in any coding environment to ensure that the code is well optimized and will not put a huge strain on the implementations of future users of the library. For this purpose, the metrics that we will revise are execution time, memory consumption and cost of each of the different algorithms generalization approach, which differs from algorithm to algorithm and refers to, in the case of data-fly for example, the number of generalization operations it has performed for a single data set.

For more specific metrics, which are quite relevant to anonymization techniques, we have decided to measure how useful the remaining data is after anonymization has been performed, in the most standardized ways possible, given the clear differences between each technique. For that purpose, three general metrics that can be widely applied to anonymization algorithms have been selected.

1. **General Information Loss (GenILoss).** This first metric aims to capture the penalty incurred when generalizing the table, by quantifying the fraction of the domain values that have been generalized for each specific attribute of the original table. Given L_i, lower bound, and U_i, upper bound, the bounds of an attribute i, a cell entry for attribute i is generalized to an interval ij defined by the lower L_ij and upper bound U_ij end points. The overall information loss can be calculated by using the following function, where T is the original table, n is the number of attributes and $|T|$ is the number of records [3]:

$$GenILoss(T^{\star}) = \frac{1}{|T| \cdot n} \sum_{i=1}^{n} \sum_{j=1}^{|T|} \frac{U_{ij} - L_{ij}}{U_i - L_j} \tag{2.3}$$

2. **Discernibility metric (DM).** This second metric measures how indistinguishable the records

of the table are from one another. It does this by assigning a penalty equal the size of its equivalence class (EQ) to each record. It follows this equation:

$$DM(T^\star) = \sum_{\forall EQ s.t. |EQ| \geq k} |EQ|^2 \tag{2.4}$$

3. **Average Equivalent Class Size Metric (C_{AVG}).** This third metric checks how well the creating of EQ approaches the case where the record is generalized in a k records' EQ. It follows this equation:

$$C_{AVG}(T^\star) = \frac{|T|}{|EQs| \cdot k} \tag{2.5}$$

Project design

3.1 Project features

This library aims to be a modular expandable anonymity tools provider, that is why when planning and implementing it, we have been following a *waterfall with iterative relationships between successive processes* methodology [13], as it adapts well to the required workflow and let us develop each tool one step at a time.

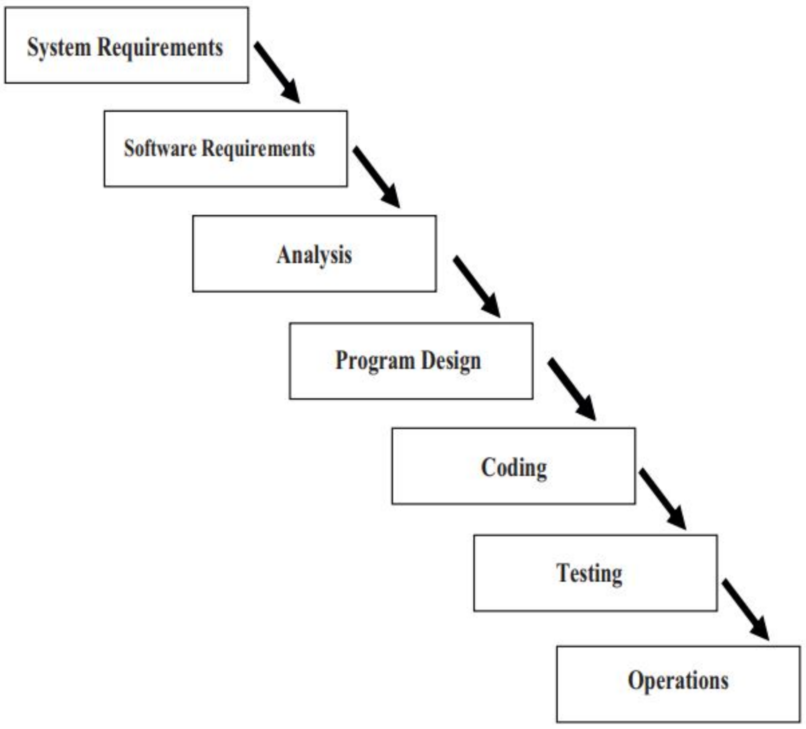


Figure 3.1: Waterfall methodology [13]

The *waterfall* methodology was firstly proposed by Winston W.Royce in 1970 [13]. This type of approach is typically used in software projects, as it conceives the work as an array of steps one after the other, as illustrated in Figure §3.1, which are executed in a direct and concrete order.

Though this approach would fit fairly well for our project, it is lacking in one key aspect, it does not allow return to an early step should the need arise, as once a task is completed it can not be returned to. For this particular reason, we opted for using its natural successor, *waterfall with iterative relationships between successive processes*, which resolves the issue by allowing the developer to go back and forth between them.

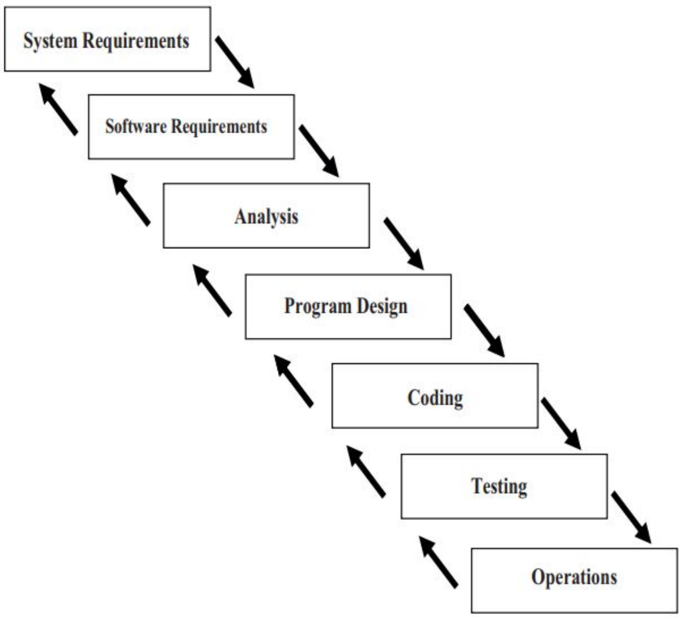


Figure 3.2: Waterfall methodology with iterative relationships between successive processes [13]

Although the Figure §3.2 shows seven phases, in practise it is usually summarise into five unique stages. The first stage is **system and software requisites analysis**, which consist on consulting the users in regards to the required services, the project restrictions and the main objectives. In this project's case this first task has been done by doing an investigation about the current state of the art of data anonymity rather than consulting any specific users, as the aim was to make a tool for everyone, not for only a selected group of individuals. The second phase, called **design**, establishes the architecture as a whole, along with a general description of the expectations regarding the final product. This has been done by stating which techniques we aim to implement, along with what language they would be programmed in, where the final library would be published and to whom it would cater to. In the third stage, **unit implementation and testing**, where the actual code is made and executed with the intention to verify its correct functionality. In this project, we have subdivided this process in little packages for each of the modules that encompass our library, so as to test each tool before implementing the next one. The forth phase is **integrating and testing the system**, which takes care of checking the working capabilities of the project at large, along with adding any necessary installation tools or interfaces. In our particular case, it would be the creation of the library itself along with the installation pipeline and documentation. The final stage is **maintenance** which is the longest phase and it is done after the project is completed, by providing updates and corrections so as to ensure that the project works as expected.

This methodology, although quite straight forward at first glance, has great advantages, as it collects the project requisites in a clear and direct manner and encloses each section of the project in a way that helps to keep the process focused and simple, avoiding any unforeseen modification or the addition of new requisites in the middle of development.

3.2 Technologies & Tools

3.2.1 Software

When it comes to software, the first step was to decide in which type of coding language the library was going to be written. As stated in the introduction and motivation of the project, the language chosen ended up being Python, due to not only how versatile and modular it is, but also the lack of complete and functional anonymization libraries there is.

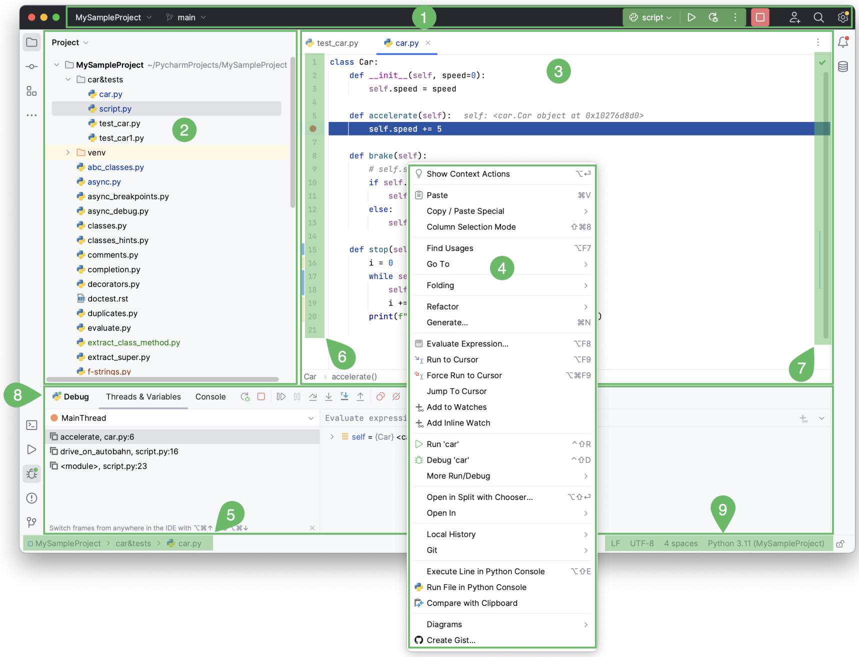
Python [17] is a high level multi-paradigm programming language, which means it offers different types of programming styles to its users, from object oriented programming to imperative programming and so on, thanks in part to extensions. Another notable advantage of Python is how modular it is, facilitating the addition of modules in even C or C++, and its dynamic name resolution, which links a method and a name to a variable during the execution time.

To be able to work with python properly, the editor selected was Pycharm, as it is quite a versatile editor native to the language which puts a lot of useful tools at the users disposal.

Pycharm [15] is an integrated development environment (IDE), specifically developed for Python (see an example of the interface in Figure §3.3). It provides assistance with code analysis, auto-completion tools, syntax and error correction, several navigation windows for easier view of the project, re-factorization to help re-organize projects more easily and an integrated debugger, among other useful tools. In the following figure §3.3 we can see the main interface of the editor, which is comprehended by the window header (1), the project tool window (2), the editor (3), the context menu (4), the navigation bar (5), the gutter (6), the scroll-bar (7), tool windows (8) and the status bar (9).

To assist during the development and be able to measure the level of anonymization our functions provided, along with checking if they worked as intended, the project employs another open source library that checks the level of anonymity of a given dataset, PyCanon.

PyCanon [14] offers tools to check the anonymity level of a dataset, along with the auxiliary



```
class Car:
    def __init__(self, speed=0):
        self.speed = speed

    def accelerate(self):   self: <car.Car object at 0x10276d8d0>
        self.speed += 5

    def brake(self):
        # self.s
        if self.
            self
        else:
            self

    def stop(sel
        i = 0
        while se
            self
            i +=
        print(f"
```

Figure 3.3: Pycharm's interface [15]

functions that make this process possible. That's why it has been an incredibly useful tool to check the correct functioning of the different functions, but it has also assisted with getting equivalence classes and calculating the current l-diversity and t-closeness of a given dataset, saving us time during development.

It is worth mentioning that this library also contains functions that measure (α, k)-anonymity, Entropy l-diversity, recursive (c, l)-diversity, basic β-likeness, enhance β-likeness and δ-disclosure privacy, which may be extremely useful for future additions to the library.

3.2.2 Hardware requirements

Given the open source nature of the library and the aim of this project to reach as many users as possible, the hardware requirements are not demanding, as the library should be able to run in any kind of equipment that has python installed.

Pycharm, the code editor used, on the other hand, can be quite resource consuming, as during operations such as indexing it can be quite draining with CPU resources. As stated in its minimal requirements [15], it is necessary to count with 8GB of RAM and at least 500MB of free disk space. In this project's case, the equipment used counts with 16 GB of RAM and 200GB of free disk space.

3.3 Continuous integration (CI) and continuous delivery (CD)

Continuous integration (CI) and continuous delivery (CD) pipelines automate much, if not all the manual labour needed to commit, build and test new code up to the deployment state.

Continuous integration is the practice of integrating all the code into the main branch of the repository and automatically testing each change before committing or merging it. It helps identify any errors or security concerns so they can be easily fixed.

Continuous delivery (CD), on the other hand, automates the infrastructure provisioning and application released process by working in conjunction with CI. Once the code is tested and build by the CI process, CD takes care of covering everything else, be it testing or deploying the application.

The pipeline for this project starts by declaring the stages it will go through to build and deploy the library, along with any test it performs in the process. The jobs for this pipeline, which are declared at the top of the script in the section called ’stages’, are ’lint’, ’build’, ’test’ and ’deploy’, and will be executed in a sequential order. After that, we declare any necessary variables that will be reused during the pipeline declaration, in this case, the ’PIP_CACHE_DIR’ which changes the cache directory so that it’s inside the project directory. We also added some cache paths so as to be able to cache the installed packages, though to properly track them we will also need to install a virtual environment and cache it as well, that’s why one of the cache paths we have declared is ’venv’.

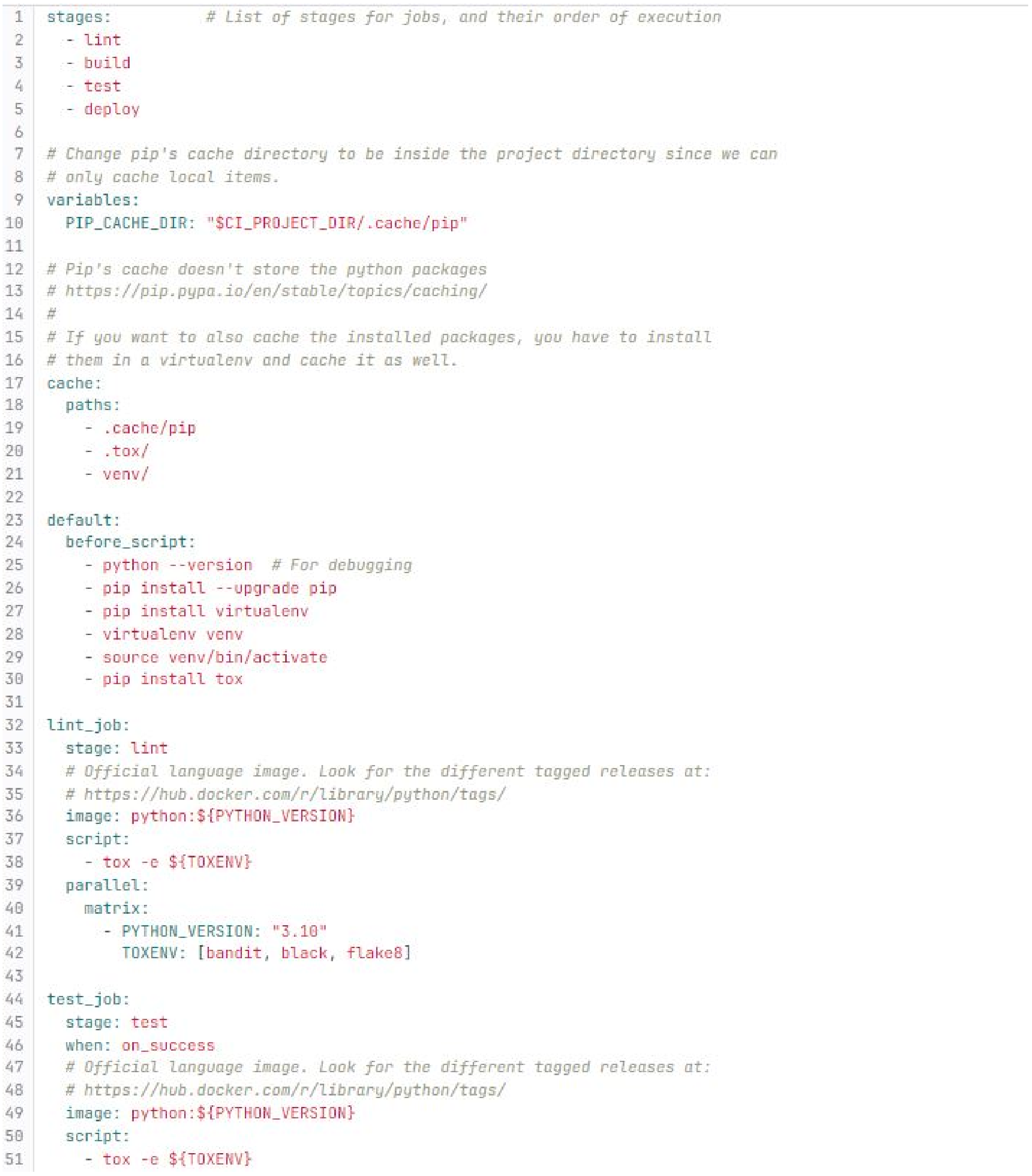
The next section of the pipeline is ’default’, which specifies configurations that will be applied to all jobs and when we want to apply them. In this particular case, we want said configurations to be applied before the script. The configurations are checking the python version, installing and running the virtual environment and installing *tox*, which is a tool that helps to automate and standardize testing.

The following stage, ’lint_job’, which takes care of running the *tox* environment and checks for any syntax errors and other issues in our python module. ’Build-job’, the next stage, combines the source code and its dependencies to build a runnable instance of the library and failure in this process indicates a fundamental issue in the project’s configuration.

The ’test_job’ stage, which comes after the ’build’ stage, takes care of running all the necessary test to validate our code’s behaviors and acts as a safety net that catches any possible bugs from reaching the library users. During this step, we also run our unit test to check each part of the library in isolation too.

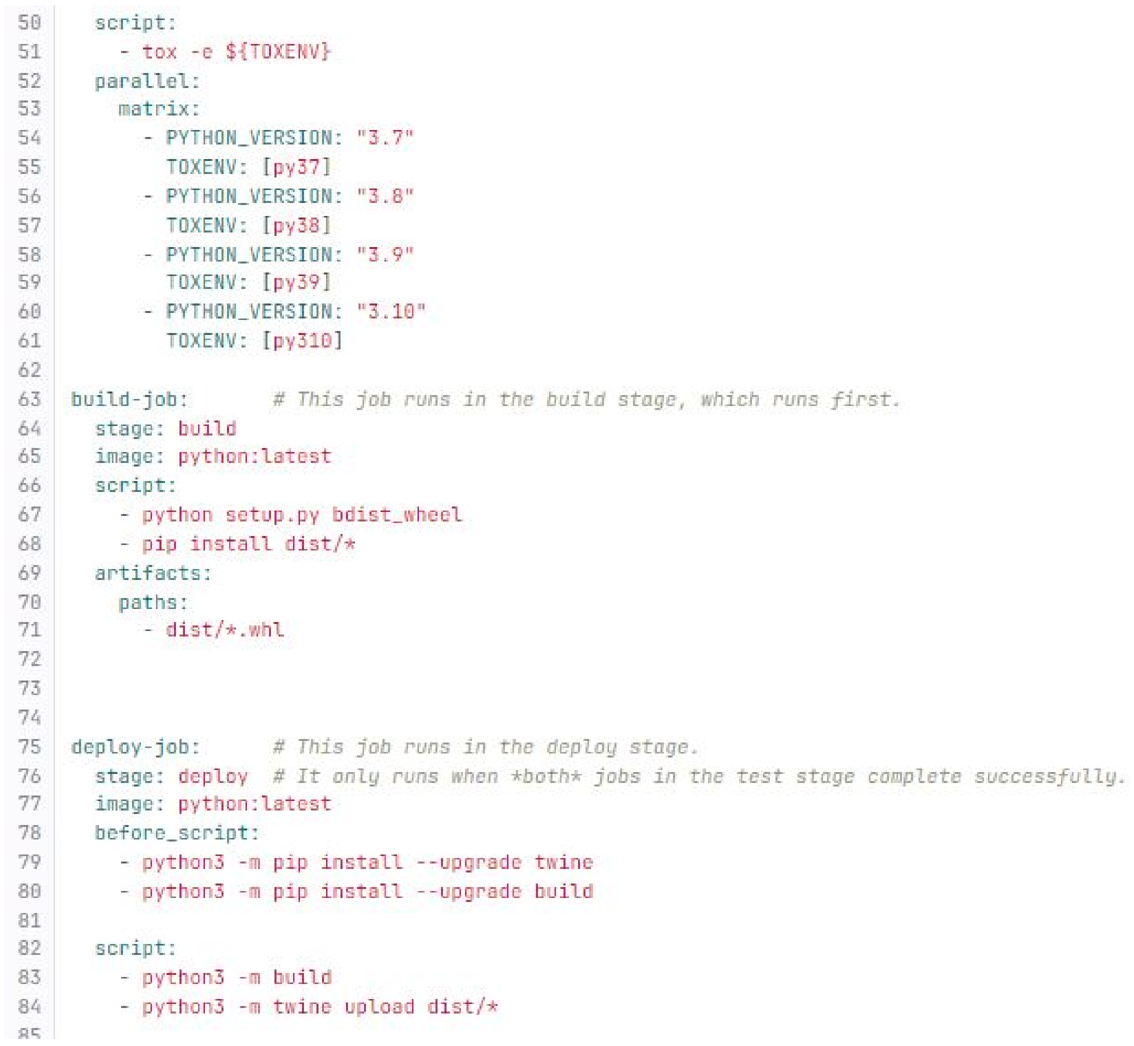
Last but not least, the final job of the pipeline, the ’deploy-job’, with which we deploy the runnable instance we have created during the ’build-job’. It’s important to mention that all this stages contain command line comments which let us know that they are working as intended.

The final pipeline is illustrated in figures §3.4 and §3.5.



```yaml
stages:          # List of stages for jobs, and their order of execution
  - lint
  - build
  - test
  - deploy

# Change pip's cache directory to be inside the project directory since we can
# only cache local items.
variables:
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

# Pip's cache doesn't store the python packages
# https://pip.pypa.io/en/stable/topics/caching/
#
# If you want to also cache the installed packages, you have to install
# them in a virtualenv and cache it as well.
cache:
  paths:
    - .cache/pip
    - .tox/
    - venv/

default:
  before_script:
    - python --version  # For debugging
    - pip install --upgrade pip
    - pip install virtualenv
    - virtualenv venv
    - source venv/bin/activate
    - pip install tox

lint_job:
  stage: lint
  # Official language image. Look for the different tagged releases at:
  # https://hub.docker.com/r/library/python/tags/
  image: python:${PYTHON_VERSION}
  script:
    - tox -e ${TOXENV}
  parallel:
    matrix:
      - PYTHON_VERSION: "3.10"
        TOXENV: [bandit, black, flake8]

test_job:
  stage: test
  when: on_success
  # Official language image. Look for the different tagged releases at:
  # https://hub.docker.com/r/library/python/tags/
  image: python:${PYTHON_VERSION}
  script:
    - tox -e ${TOXENV}
```

Figure 3.4: Pipeline of the repository



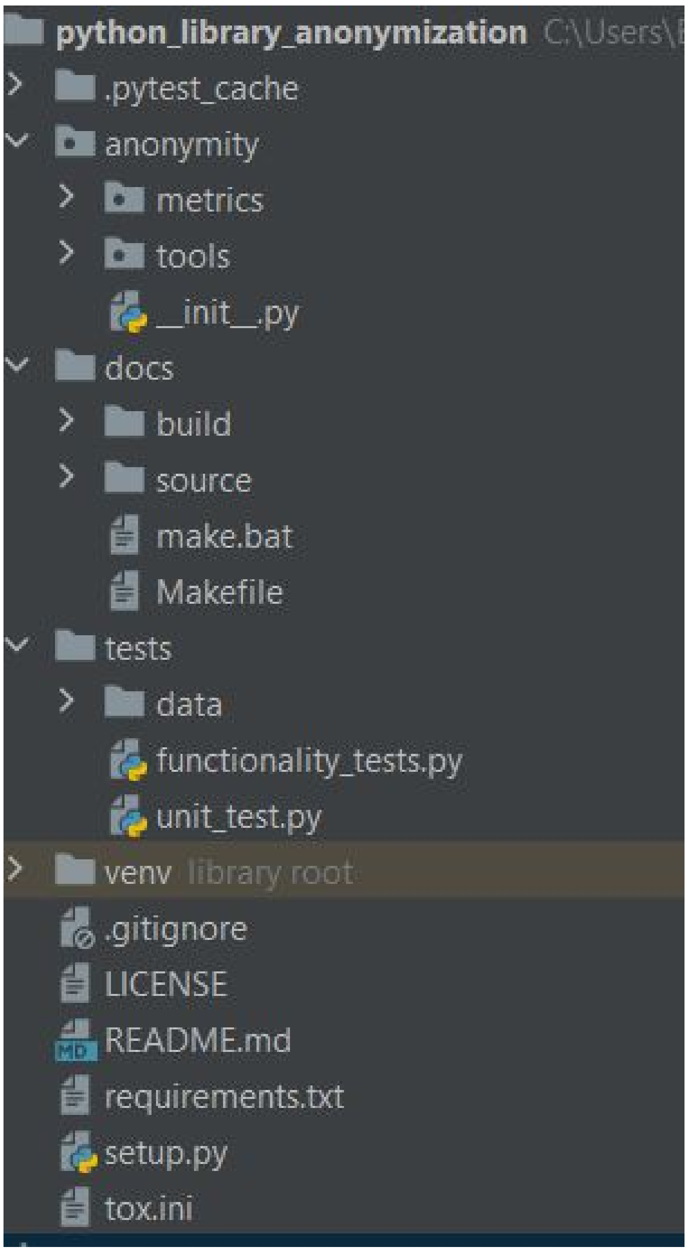
```
  script:
    - tox -e ${TOXENV}
  parallel:
    matrix:
      - PYTHON_VERSION: "3.7"
        TOXENV: [py37]
      - PYTHON_VERSION: "3.8"
        TOXENV: [py38]
      - PYTHON_VERSION: "3.9"
        TOXENV: [py39]
      - PYTHON_VERSION: "3.10"
        TOXENV: [py310]

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  image: python:latest
  script:
    - python setup.py bdist_wheel
    - pip install dist/*
  artifacts:
    paths:
      - dist/*.whl



deploy-job:      # This job runs in the deploy stage.
  stage: deploy  # It only runs when *both* jobs in the test stage complete successfully.
  image: python:latest
  before_script:
    - python3 -m pip install --upgrade twine
    - python3 -m pip install --upgrade build

  script:
    - python3 -m build
    - python3 -m twine upload dist/*
```

Figure 3.5: Pipeline of the repository

3.4 Library implementation

The first step to generate a library is to organize the project in a following a structure of folders and sub-folders, as illustrated in figure §3.6, in a way that the base folder of the project contains a *setup.py* script and a *README.md*, and that each of the modules contained in the sub-folders have a *__init__.py* script.

In our particular case, our library, called *python-anonymity* would consists on three main folders: *anonymity* which contains the scripts of the metrics and tools provided by the library, *docs* which contains the documentation of the library and the files needed to generate said documentation, and *tests* which contains the functionality and integration test described in the next chapter. Inside each of these folders there's an *__init__.py* script, thanks to which Python can interpret that said folder is part of the library. Along with these three main folders, in the base directory we can also find two key files: *README.md*, which is a file in the markdown format

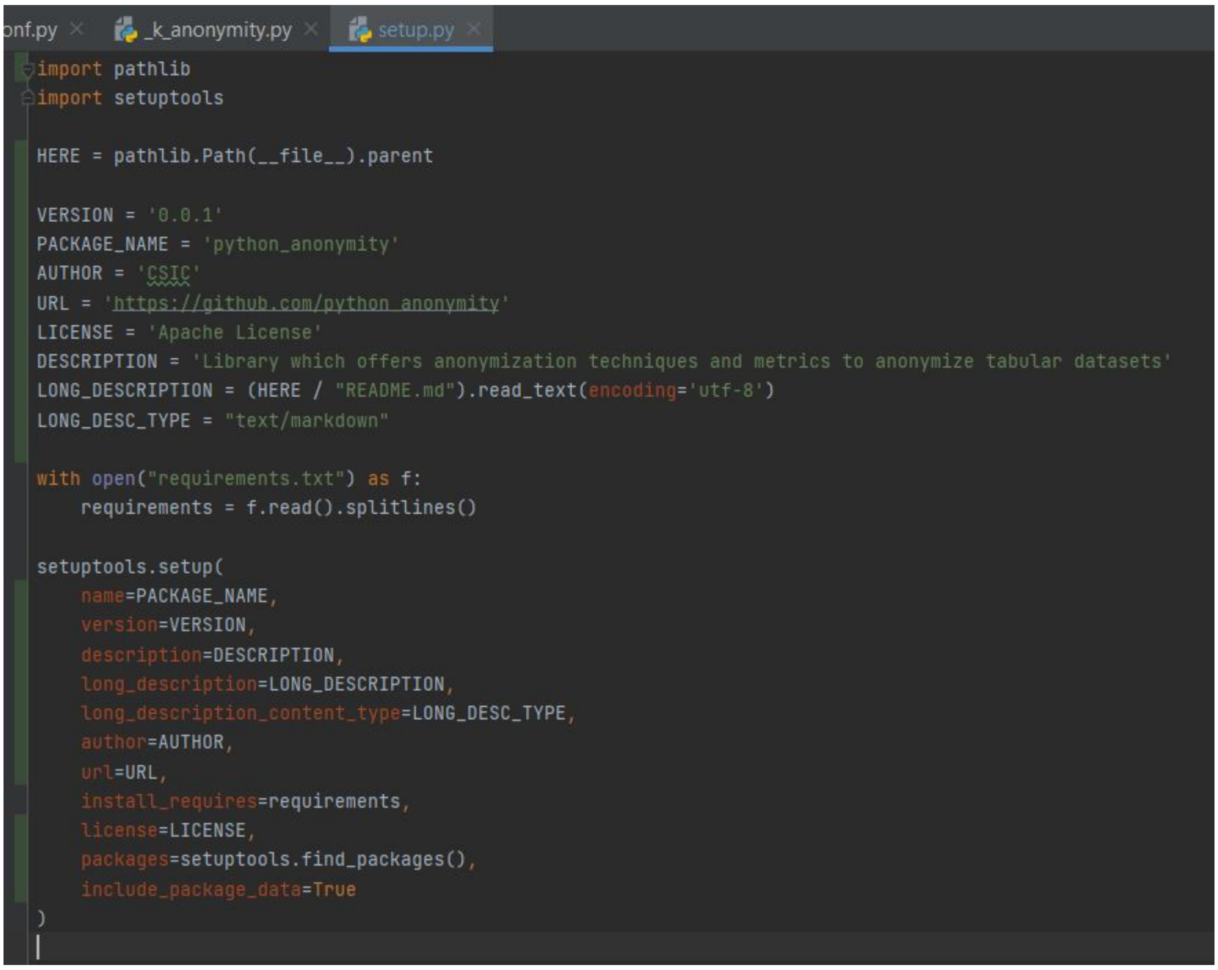


```
python_library_anonymization C:\Users\
> .pytest_cache
v anonymity
    > metrics
    > tools
      __init__.py
v docs
    > build
    > source
      make.bat
      Makefile
v tests
    > data
      functionality_tests.py
      unit_test.py
> venv library root
  .gitignore
  LICENSE
  README.md
  requirements.txt
  setup.py
  tox.ini
```

Figure 3.6: Structure of the project folder

which gives information regarding the library, and *setup.py*, which is a script that includes the necessary instructions to be able to install the library, such as the requirements. In our case, the requirement are located in a separate file for easier editing called *requirements.txt* which can also be found in the base folder.

The *setup.py* contains the current version of the library, the installation requirements, though in our case they are called into the script from a separate file, the name of the library, the license, the description, the repository which contains it and the authors' names. It's structure is shown in figure §3.7.



```python
import pathlib
import setuptools

HERE = pathlib.Path(__file__).parent

VERSION = '0.0.1'
PACKAGE_NAME = 'python_anonymity'
AUTHOR = 'CSIC'
URL = 'https://github.com/python_anonymity'
LICENSE = 'Apache License'
DESCRIPTION = 'Library which offers anonymization techniques and metrics to anonymize tabular datasets'
LONG_DESCRIPTION = (HERE / "README.md").read_text(encoding='utf-8')
LONG_DESC_TYPE = "text/markdown"

with open("requirements.txt") as f:
    requirements = f.read().splitlines()

setuptools.setup(
    name=PACKAGE_NAME,
    version=VERSION,
    description=DESCRIPTION,
    long_description=LONG_DESCRIPTION,
    long_description_content_type=LONG_DESC_TYPE,
    author=AUTHOR,
    url=URL,
    install_requires=requirements,
    license=LICENSE,
    packages=setuptools.find_packages(),
    include_package_data=True
)
```

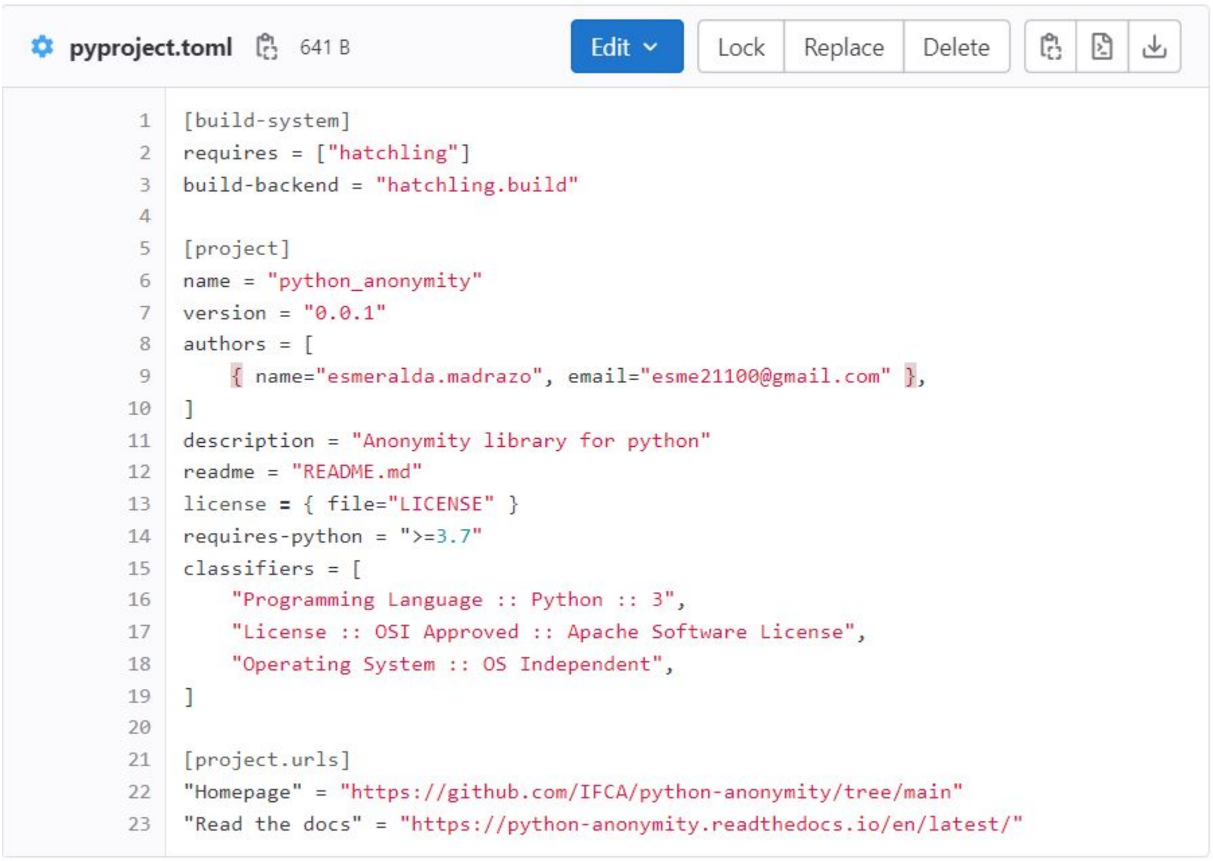
Figure 3.7: Structure of the setup.py file

Another file that we need to create to be able to properly prepare our library is the *tox.ini* file, which helps us create and configure virtual environments and their dependencies for the purpose of testing the package. In our particular case, *tox.ini* contains the configuration for the *Python 3* environment, the *flake8, black and bandit*, which are all tools for code styling and syntax errors, environments, the *mypy*, which is an optional type of Python checker, environment and the *Pypy* environment.

Once all the files are ready, the last step is to generate the actual package that we will submit to Pypi, which is a repository of software for the Python language [1], the functional models of the library and the readme file. The first step is to run the *setup.py* script on a command line by using the following command: *python setup.py sdist bdist_wheel*. If everything is working as intended, it generates a folder called "dist" which contains the packaged library.

Lastly, it is time to upload the package to Pypi so it can be accessed by other users. For that, we need to create a Pypi account and generate a token which will enable us to submit things to Pypi. To do so, the account need to have two factor authentication though any external apps, such as google authentication. Once we have the token at our disposal we need to go back to the project and create a file called *pyproject.toml*, which takes care of the storing configuration of the project. It should contain its name, the requirements for the build, some general descriptors, including the readme and license file names, the author's name in Pypi, the project and docs urls and some classifiers which will help categorise the project, which in our case are Python 3, Apache License and Operating System Independent.

The file in our library looks as shown in figure §3.8.



```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "python_anonymity"
version = "0.0.1"
authors = [
    { name="esmeralda.madrazo", email="esme21100@gmail.com" },
]
description = "Anonymity library for python"
readme = "README.md"
license = { file="LICENSE" }
requires-python = ">=3.7"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: Apache Software License",
    "Operating System :: OS Independent",
]

[project.urls]
"Homepage" = "https://github.com/IFCA/python-anonymity/tree/main"
"Read the docs" = "https://python-anonymity.readthedocs.io/en/latest/"
```

Figure 3.8: Structure of the pyproject.toml file

Once all the necessary files are ready, we can go ahead and use the command *python -m twine upload –repository pypi dist/** to submit the packaged library to Pypi. This command prompts us to introduce a user, which should be *__token__*, and a password, which is the token we had previously generated in our Pypi account. And with that, the library is available in Pypi at the following url: `https://pypi.org/project/python-anonymity/#modal-close` and can be installed directly from a command line by using the command *pip install python-anonymity*.

3.5 Library documentation

The documentation for the library, which can be found in the docs folder inside the projects repository that is included in the next section, has been generated by using Sphinx [5], a powerful documentation generator which can create from printable PDFs to web pages and more, and can also use markdown to write the documentation. In addition to this, it also highlights code samples. In our particular case, we have generated HTML files from .rst files.

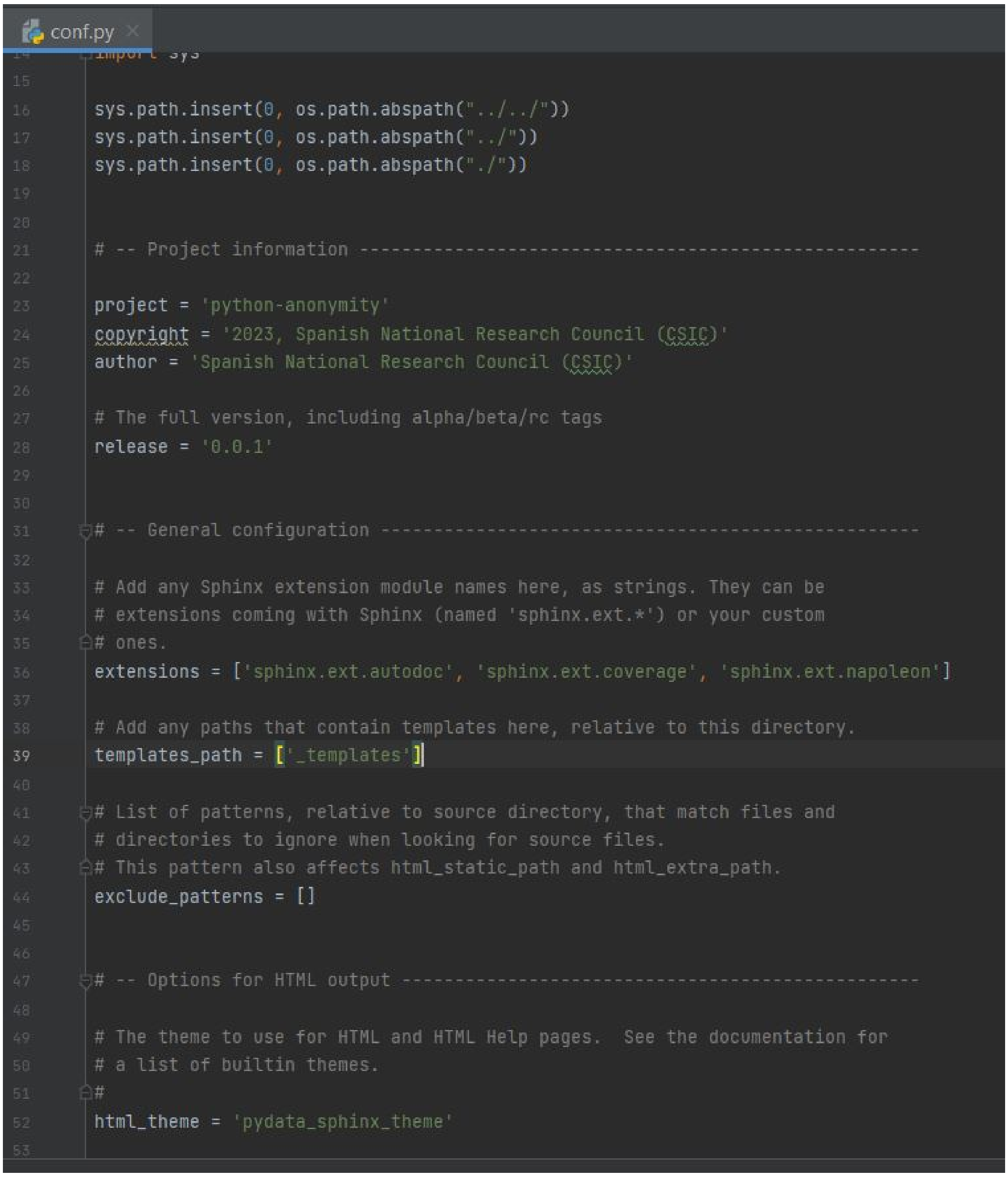
To generate the documentation we first need to install sphinx, which can easily be done in any terminal by using the command *pip install sphinx*. In our particular case, we have also installed a customization packaged called *pydata_sphinx_theme* to change the visual aspect of the documentation, but this step is not necessary for their correct functionality.

Once the package is installed, we need to ensure that the project structure goes as illustrated in §3.6 and in the previous section. What we need to ensure for a smooth document generation are two key components: a main folder which contains the code for the library functions, the docs folder which will hold both the HTML files generated and the .rst files that allow us to generate the documentation and other folders and files which are less relevant for documentation kept outside of these two.

Now that the structure is adequate, we start generating the documentation by using the command *sphinx-quickstart* in the docs folder, which will generate the complete documentation directory, along with a file called make.bat, which will help us generate the HTML docs.

After generating the documentation directory we locate the *conf.py* file inside it and change the *os.path.abspath()* to indicate where the code is being stored and that it is outside of the docs folder. In this folder, we also need to add the project information, which consists on mainly the name of the project, its copyright, the current released version and the author. The next thing we need to add to the file are the general configurations, that is the extensions that we are using, the templates and any patters we may want to exclude, which in our case were none. Last but not least, we can add some optional output configurations, which we added, as we are using a theme for the documentation, as previously stated. The whole structure of the *conf.py* can be seen in figure §3.9

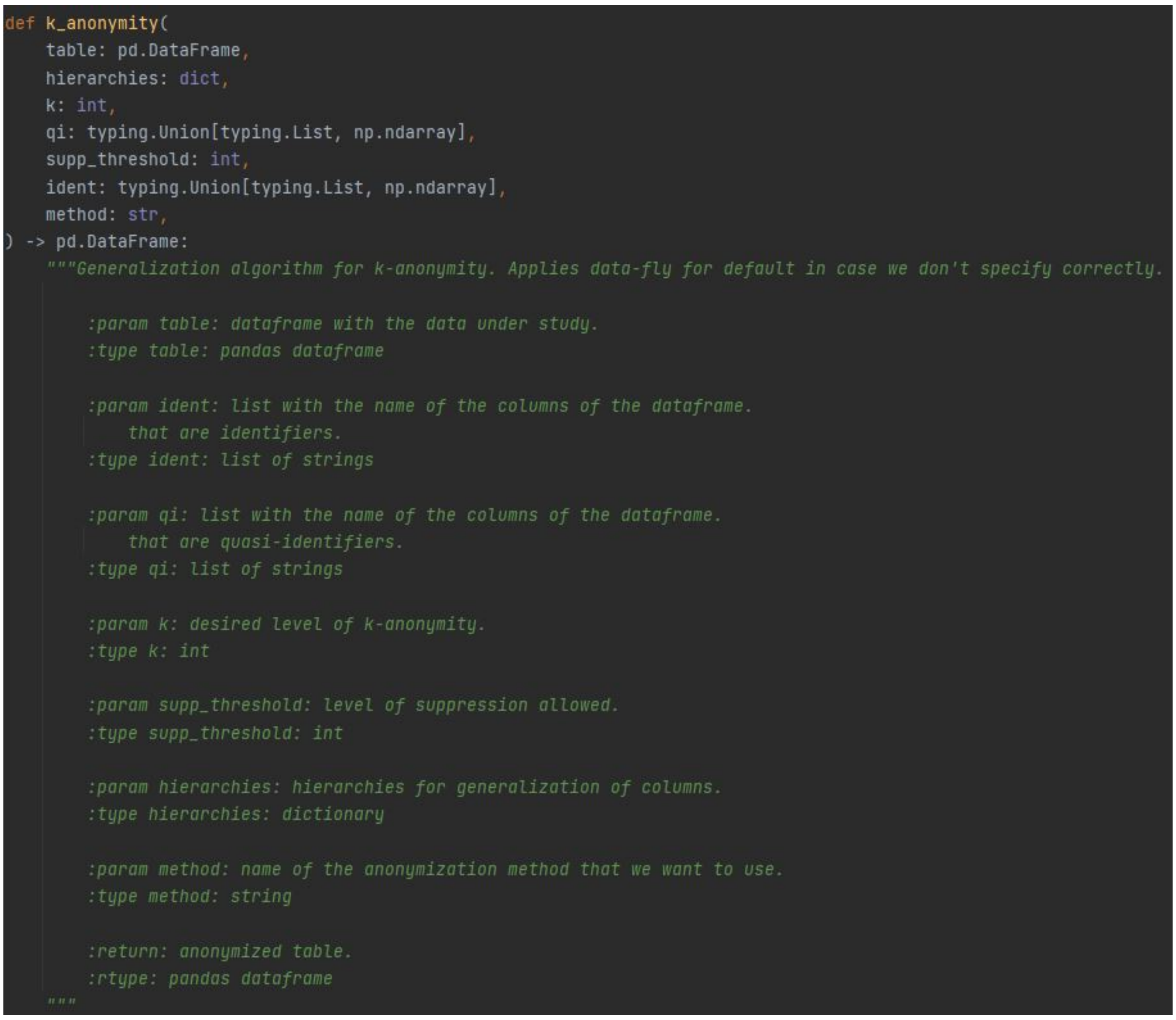
With the *conf.py* set up, the next step is to check if the *index.rst* file has been correctly generated in the source folder and to generate the rest of the .rst files needed to create the final HTML documentation. To do so, we need to use the following command, *sphinx-apidoc -o docssource anonymity* which will check our code located on the anonymity folder and will generate the corresponding output on the .rst files in the docssource folder. After those have been successfully generated we need to create a modules.rst file in that same folder, which should contain a reference to the folder that contains the modules and the maximum depth in the directory that sphinx is allowed to check.



```python

sys.path.insert(0, os.path.abspath("../../"))
sys.path.insert(0, os.path.abspath("../"))
sys.path.insert(0, os.path.abspath("./"))


# -- Project information -----------------------------------------------------

project = 'python-anonymity'
copyright = '2023, Spanish National Research Council (CSIC)'
author = 'Spanish National Research Council (CSIC)'

# The full version, including alpha/beta/rc tags
release = '0.0.1'


# -- General configuration ---------------------------------------------------

# Add any Sphinx extension module names here, as strings. They can be
# extensions coming with Sphinx (named 'sphinx.ext.*') or your custom
# ones.
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.coverage', 'sphinx.ext.napoleon']

# Add any paths that contain templates here, relative to this directory.
templates_path = ['_templates']

# List of patterns, relative to source directory, that match files and
# directories to ignore when looking for source files.
# This pattern also affects html_static_path and html_extra_path.
exclude_patterns = []


# -- Options for HTML output -------------------------------------------------

# The theme to use for HTML and HTML Help pages.  See the documentation for
# a list of builtin themes.
#
html_theme = 'pydata_sphinx_theme'

```

Figure 3.9: Structure of the configuration script for generating the documentation of the library

The last step is to generate the HTML, and for it we just simply need to execute the command *make html*, which will generate the final documentation inside the docsbuild folder.

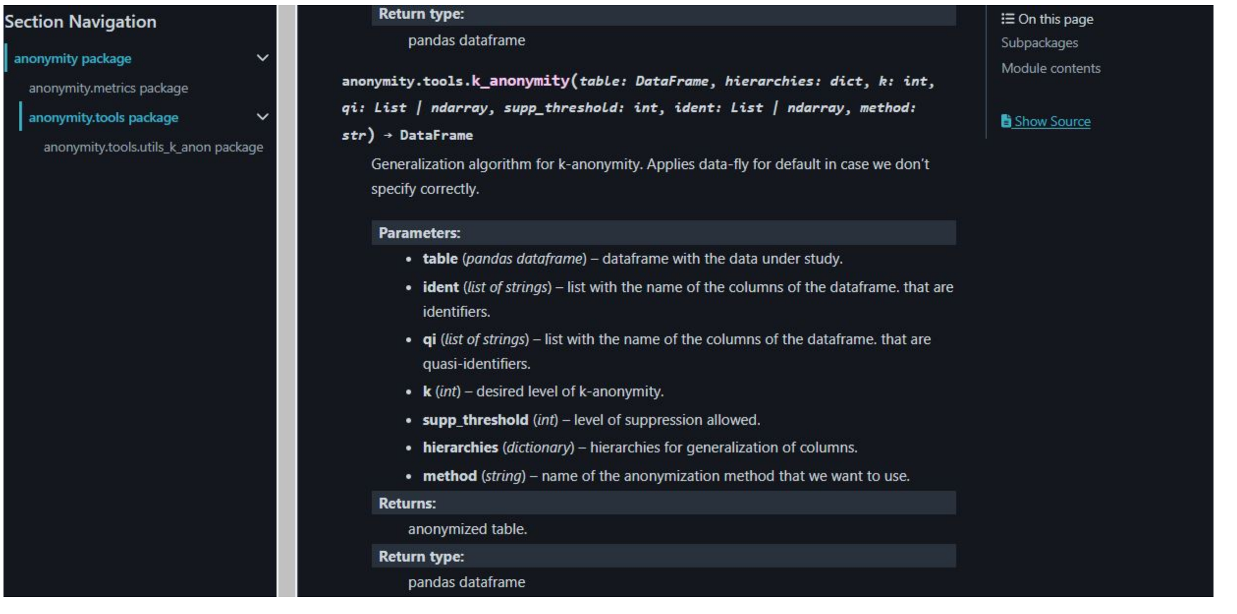


```
def k_anonymity(
    table: pd.DataFrame,
    hierarchies: dict,
    k: int,
    qi: typing.Union[typing.List, np.ndarray],
    supp_threshold: int,
    ident: typing.Union[typing.List, np.ndarray],
    method: str,
) -> pd.DataFrame:
    """Generalization algorithm for k-anonymity. Applies data-fly for default in case we don't specify correctly.

        :param table: dataframe with the data under study.
        :type table: pandas dataframe

        :param ident: list with the name of the columns of the dataframe.
            that are identifiers.
        :type ident: list of strings

        :param qi: list with the name of the columns of the dataframe.
            that are quasi-identifiers.
        :type qi: list of strings

        :param k: desired level of k-anonymity.
        :type k: int

        :param supp_threshold: level of suppression allowed.
        :type supp_threshold: int

        :param hierarchies: hierarchies for generalization of columns.
        :type hierarchies: dictionary

        :param method: name of the anonymization method that we want to use.
        :type method: string

        :return: anonymized table.
        :rtype: pandas dataframe
    """
```

Figure 3.10: Sample of comment structure in code [5]

Also, it is important to mention that all of these has been possible by structuring the comments on our code as illustrated in §3.10, which will help Sphinx render them automatically in our documentation as in §3.11.

The final library documentation has been uploaded to *Read the docs*. To do so, we had to create a small pipeline called *.readthedocs.yml*, which serves to specify the specific requirements to generate the docs. The final documentation can be found in the following link: `https://python-anonymity.readthedocs.io/en/latest/`



Return type:
pandas dataframe

```
anonymity.tools.k_anonymity(table: DataFrame, hierarchies: dict, k: int,
qi: List | ndarray, supp_threshold: int, ident: List | ndarray, method:
str) → DataFrame
```

Generalization algorithm for k-anonymity. Applies data-fly for default in case we don't specify correctly.

Parameters:
- **table** (*pandas dataframe*) – dataframe with the data under study.
- **ident** (*list of strings*) – list with the name of the columns of the dataframe. that are identifiers.
- **qi** (*list of strings*) – list with the name of the columns of the dataframe. that are quasi-identifiers.
- **k** (*int*) – desired level of k-anonymity.
- **supp_threshold** (*int*) – level of suppression allowed.
- **hierarchies** (*dictionary*) – hierarchies for generalization of columns.
- **method** (*string*) – name of the anonymization method that we want to use.

Returns:
anonymized table.

Return type:
pandas dataframe

Figure 3.11: Sample of comment structure in documentation [5]

3.6 Project availability

This project will be available, along with its code, documentation and compliance tests, in a GitHub repository, `https://github.com/IFCA/python-anonymity/tree/main`.

Due to it being open source and with the possible additions of other features, techniques and tools, the project is subject to updates and changes.

It's last update, before the publication of this work, was done in 19th of September of 2023 and it is version 0.0.1.post1.

This library can be directly installed from the command prompt by using the *pip install python-anonymity* command and can be found on Pypi by following this url `https://pypi.org/project/python-anonymity/0.0.1/`.

The documentation, which has been submitted to *Read the docs*, can also be found by accessing the following url `https://python-anonymity.readthedocs.io/en/latest/`.

Use cases & Functionality tests

4.1 Use cases

Once everything has been implemented the next step is to test out each module to see if it works as intended or if it contains any errors, be it during development or in future updates. To help us with this process we have created several functionality and integration test using three different datasets: *adult.csv* [4], an dataset created by Barry Becker from the 1994 Census database which prediction task is to determine whether a person makes over 50K a year that contains more than 30.000 rows, *hospital_extended.csv*, a small artificial dataset which prediction task is to determine the illness of each patience, and *crime*, another small artificial dataset directly generated from a Python script which prediction task is to determine the type of crime committed by each individual.

As stated before, the main reason for this particular selection of datasets is that, in the case of the two synthetic ones, their parameters are easier to track and the number of rows helps to predict borderline use cases which might cause errors if the functions do not take them into account or have any kind or error prevention measures. In the case, for example, of the *crime* dataset, its QI, which are *"marital stat"*, *"age"*, *"ZIP code"*, are conformed by short and simple values which let us test both categorical and numerical versions of each method without having to create extensive hierarchies. Its SA, *"crime"* is also a straight forward set of values which do not hinder the testing process. Same thing can be applied to the *hospital_extended.csv* dataset, though a bit bugger, its QI, which are *"age"*, *"gender"*, *"city"*, also make it fairly simple to create hierarchies for them and, with *disease* as its SA, allow us to test the functions on a slightly bigger scale but still in a controlled environment.

But all these cases still do not resemble a real use of the library outside of an academic or learning setting. That is why we have carefully selected the *adult.csv* [4], as it being one of the most well known datasets for anonymization and being composed by real data and having more than 30.000 rows, which is a closer measure to the type of workload this library expects to handle. This dataset is obviously more complex than the rest and it contains several QI, which are *"age"*, *"education"*, *"occupation"*, *"marital-status"*, *"race"*, *"sex"* and that we have selected between all the columns by discarding the other less related values as ID (*"workclass"*, *"fnlwgt"*, *"education-num"*, *"relationship"*, *"capital-gain"*, *"capital-loss"*,*"hours-per-week"*, *"native-country"*). All these QI require more complex hierarchies to properly be anonymized, and they contain both categorical and numerical values for a more through out testing, but the SA attribute, *"salary-class"* makes the process a bit simpler, as it does not require a complex hierarchy by only having two possible values. A small sample of how this dataset looks once it's ready to be anonymized can be found in table §4.1.

Age	**Education**	**Occupation**	**Marital-status**	**Race**	**Sex**	**Salary-classs**
21	Some-college	Priv-house-serv	Never-married	Asian-Pac-Islander	Male	<=50K
79	Bachelors	Tech-support	Widowed	Black	Female	>50K
37	Masters	Machine-op-inspct	Divorced	Other	Male	<=50K

Table 4.1: Sample version of the adult.csv data.

A usability sample of the library would go as follows:

1. First, we create a virtual environment.

```
virtualenv .venv -p python3
source .venv/bin/activate
```

2. Then, we install the library by using the command:

```
pip install python-anonymity
```

And finally, we create a python script and try one of the functions. Here is a quick sample for the function *k*-anonymity.

```
import pandas as pd
import pycanon
from anonymity import tools
from anonymity.tools.utils_k_anon import utils_k_anonymity as utils

d = {
        "name": ["Joe", "Jill", "Sue", "Abe", "Bob", "Amy"],
        "marital stat": [
           "Separated",
           "Single",
           "Widowed",
           "Separated",
           "Widowed",
           "Single",
        ],
        "age": [29, 20, 24, 28, 25, 23],
        "ZIP code": ["32042", "32021", "32024", "32046", "32045", "32027"],
        "crime": ["Murder", "Theft", "Traffic", "Assault", "Piracy", "Indecency"↩
            ],
    }

data = pd.DataFrame(data=d)
ID = ["name"]
QI = ["marital stat", "age", "ZIP code"]
SA = ["crime"]
age_hierarchy = {"age": [0, 2, 5, 10]}
hierarchy = {
```

```
            "marital stat": [
                ["Single", "Not married", "*"],
                ["Separated", "Not married", "*"],
                ["Divorce", "Not married", "*"],
                ["Widowed", "Not married", "*"],
                ["Married", "Married", "*"],
                ["Re-married", "Married", "*"],
            ],
            "ZIP code": [
                ["32042", "3204*", "*"],
                ["32021", "3202*", "*"],
                ["32024", "3202*", "*"],
                ["32046", "3204*", "*"],
                ["32045", "3204*", "*"],
                ["32027", "3202*", "*"],
            ],
        }

mix_hierarchy = dict(hierarchy, **utils.create_ranges(data, age_hierarchy))

k = 2
supp_threshold = 0
new_data = tools.k_anonymity(data, ID, QI, k, supp_threshold, mix_hierarchy, ↩
    method =  "datafly")
```

The initial table looked like figure §4.2 and the now anonymized one returned by the function is the one shown in figure §4.3, which satisfies k-anonymity with $k = 2$.

Name	Marital stat	Age	ZIP Code	Crime
Joe	Separated	29	32042	Murder
Jill	Single	20	32021	Theft
Sue	Widowed	24	32024	Traffic
Abe	Separated	28	32046	Assault
Bob	Widowed	25	32045	Piracy
Amy	Single	23	32027	Indecency

Table 4.2: Initial table sample for the crime dataset

Name	Marital stat	Age	ZIP Code	Crime
*	Not married	[20.0, 30.0)	3204*	Murder
*	Not married	[20.0, 30.0)	3202*	Theft
*	Not married	[20.0, 30.0)	3202*	Traffic
*	Not married	[20.0, 30.0)	3204*	Assault
*	Not married	[20.0, 30.0)	3204*	Piracy
*	Not married	[20.0, 30.0)	3202*	Indecency

Table 4.3: Anonymized table sample for the crime dataset return by the library function

4.2 Functionality tests

For this particular library, each test case checks each one of the functions of each module in isolation, to ensure that all of them can be called without issues. We have also used the *pycanon* [14] to assist during the assertion process, as it can calculate the real values for k, l and t of the anonymized datasets.

The first module that we tested is the *_k-anonymity*, which contains both the *datafly* and *incognito* methods. To test both of them we have settled on 6 relevant use cases for each of them, making it 12 totals test for the *_k-anonymity* module, that will test for any possible failures during implementation. The first case takes care of checking if, given a high value of k that is bigger than the amount of rows in the dataset, the function will not be able to anonymize the dataset to satisfy that k, be it with the help of suppression, as in case 2, or not, as in case 1. The second testing case checks if, given a value of k which can realistically be satisfied, the function returns a value of k which is equal or higher to the k we introduced as an input, that is a value of k which satisfies the k-anonymity required by the user, be it by using the assistance or suppression, as in case 4, or by not allowing suppression at all, case 3. The last case for each of the methods was that, given a value of k fairly small in comparison with the actual k that can be achieve with anonymization, the methods would return a value of k equal or higher to the one used as an input.

For each of these test, both the incognito and datafly functions and the main function k-anonymity are called by following these three structures:

1. *data_fly(table, ident, qi, k = 2, supp_threshold = 2, hierarchies)*
2. *incognito(table, ident, qi, k = 2, supp_threshold = 2, hierarchies)*
3. *k_anonymity(table, ident, qi, k = 2, supp_threshold = 2, hierarchies, method = "incognito")*

For the next two modules, *_l_diversity* and *_t_closeness*, we have first tested them in a similar manner to the last module, by employing 24 functions in total, 12 for each, to ensure that given values of l and t that are higher, realistic or lower the methods will still return proper results when given a not anonymized dataset that they will have to anonymize before applying each particular technique, be it with *datafly* or *incognito*. For example, one of the test checks that, given a value of l realistic to the number of rows and value of the attributes in the dataset, and the raw *crime* dataset, it returns a value of l equal or higher to the one used as an input so that it satisfies l-diversity.

For each of these test, the l-diversity, t-closeness and t-closeness-supp functions are called by following these three structures:

1. *l_diversity(table, SA, QI, k_method = "datafly", l = 2, ID, supp_threshold = 3, hierarchies, k = 2)*
2. *t_closeness(table, SA, QI, t = 0.7, k_method = "incognito", ID, supp_threshold = 2, hierarchies)*
3. *t_closeness_supp(table, SA, QI, t = 0.7, supp_limit = 0.6)*

For these two modules we also included integration testing, also called thread testing, that is a type of testing which checks the interaction between modules. In this case, we check the interactions between the *_k_anonymity* module and each of these two so as to see if, when using one of the two *_k_anonymity* functions to anonymize a dataset before passing it to the *l*-diversity and *t*-closeness methods they still return the proper results. This is due to the fact that in this library, as in most software products, modules can be used in isolation but they also can be used to complement each other.

The last modules tested are *data_utility_metrics* and *efficiency_metrics* which we test by having them compare two datasets, the raw dataset which has not been anonymized yet, and a new dataset, which is a version of the latter which has already passed through an anonymization process. Passing these two values as inputs, we can check whether the function is working properly or not by checking that the metric it returns is higher than 0, as returning 0 is the default value and would mean that the function has not been executed at all.

For each of these test we have called the l-diversity, t-closeness and t-closeness-supp functions are called by following these three structures:

1. *generalized_information_loss(hierarchies, table, new_table, QI)*
2. *discernibility(table, new_table, QI*
3. *avr_equiv_class_size(table, new_table, QI*

After executing all the functionality tests and checking that they pass properly and that there are not any major issues or errors, the next step is to check whether the functions work as intended with more realistic datasets. We achieved this by creating a second testing script which checks that each of the functions fulfills the initial requirements established at the beginning of the project, that is that each function is able to apply the implemented technique when given a relatively big dataset. The functions that we check with this one do not tackle all the possible cases, just a general use case to check if they work properly with a bigger and more realistic dataset. In this case, the functions we check are *data-fly*, *incognito*, *l-diversity with datafly*, *l-diversity with incognito*, *t-closeness with datafly* and *t-closeness with incognito*.

It's also important to mention that, in the future, we will implement some unit test which is a type of software test which aims to ensure that each unit, be it a function, method, module or object, of the code works as expected, returns the expected outputs for all relevant inputs and has no errors, to check each function is working properly, but the more pressing matter is to check whether or not the main functions work properly and return results that make sense. For that purpose, the first type of testing implemented has been the functionality tests. We have run several of these tests with our synthetic dataset, as it was easier to predict and control which results they should return.

Conclusions

5.1 Achieved objectives

In the previous sections of the project four techniques have been explained and implemented. Along with them, the creation of the final library and its documentation have been presented. The motivation and research behind their implementation and the decision to add them in the first place have also been justified. The measuring metrics and tests used to evaluate is correct execution have also been described in detail. And the first version library, which is still in a production state, has been published and is available for general use.

The theory behind each technique and the attacks which protects user data from, have been discussed, and some of the most important concepts of anonymity have been explained, so as to illustrate a general idea of what anonymizing data entails for any newcomer.

In addition, the tools used during the development are also included in previous sections of this work, along with the documentation for the library, which has been manually generated and which creation process has been dissected in Chapter 3. We have also explained step-by-step the structure of a Python library and which files are necessary for its correct creation, along with the implementation of each of them. A pipeline has also been added to the repository so as to speed up the process for future updates and releases.

Obviously, it also satisfies the main objective of the project, that is, to create an openly available tool for all Python users, expert or not, so that they have a simple and accessible way to anonymize their data before realising it to the public. This has been tested by applying different types of testing approaches, such as functionality and integration testing, each consisting of several tests that have evaluated various use cases with different types of data sets.

Last but not least, the project has been done under the scope of a JAE Intro internship, which objectives have also been satisfied. These objectives are are the integration of the pupil to the different CSIC investigation groups and their collaboration to the labour developed in them.

5.2 Future work

As the library has been implemented as a set of modules, each containing an anonymity tool, it leaves the door open for future additions of other techniques, be it the ones listed in PyCanon [14], like (α, k)-anonymity, Entropy l-diversity, recursive (c, l)-diversity, basic β-likeness, enhance β-likeness and δ-disclosure privacy, so as to make it fully compatible with that library, or other techniques that may become state of the art in the near future.

This modular structure also leaves the door open for future users to build upon this library with extra models, as the code and the auxiliary functions and metrics are available in GitHub. It will also let users compare the efficiency of the functions for themselves and conduct their own research on top of it.

Some of our future objectives included the implementation of unit tests, for example using Pytest, for each of auxiliary functions that conform our main modules so as to make it easier for future updates to check if any of the changes made ends up breaking a previously working component due to an issue with these auxiliary sections. We also intend to add an an application programming interface API or a web service that makes the library more accessible to the users.

BIBLIOGRAPHY

[1] Python package index - pypi. URL `https://pypi.org/`.

[2] B. Anandan, C. Clifton, W. Jiang, M. Murugesan, P. Pastrana-Camacho, and L. Si. t-plausibility: Generalizing words to desensitize text. *Transactions on Data Privacy*, 5, 12 2012.

[3] V. Ayala-Rivera, P. Mcdonagh, T. Cerqueus, and L. Murphy. A systematic comparison and evaluation of k-anonymization algorithms for practitioners. *Transactions on Data Privacy*, 7:337–370, 12 2014.

[4] B. Becker and R. Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[5] G. Brandl. Sphinx documentation. *URL http://sphinx-doc. org/sphinx. pdf*, 2021.

[6] C. Dwork. *Differential Privacy*, pages 338–340. Springer US, Boston, MA, 2011. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_752. URL `https://doi.org/10.1007/978-1-4419-5906-5_752`.

[7] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007. doi: 10.1109/ICDE.2007.367856.

[8] P. Lison, I. Pilán, D. Sanchez, M. Batet, and L. Øvrelid. Anonymisation models for text data: State of the art, challenges and future directions. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4188–4203, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.323. URL `https://aclanthology.org/2021.acl-long.323`.

[9] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. L-diversity: privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 24–24, 2006. doi: 10.1109/ICDE.2006.1.

[10] W. Mahanan, W. Chaovalitwongse, and J. Natwichai. Data privacy preservation algorithm with k-anonymity. *World Wide Web*, 24, 09 2021. doi: 10.1007/s11280-021-00922-2.

[11] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL `https://doi.org/10.5281/zenodo.3509134`.

[12] F. Prasser, J. Eicher, H. Spengler, R. Bild, and K. A. Kuhn. Flexible data anonymization using arx - current status and challenges ahead. *Software: Practice and Experience*, 20(7): 1304, 2020. doi: 10.1002/spe.2812.

[13] W. W. Royce. Managing the development of large software systems. *IEEE WESCON*, pages 328–338, 1970.

[14] J. Sáinz-Pardo Díaz and A. López García. A python library to check the level of anonymity of a dataset. *Scientific Data*, 9(1):785, 2022.

[15] J. s.r.o. Pycharm, getting started, aug 2023. URL `https://www.jetbrains.com/help/pycharm/getting-started.html`.

[16] D. Sánchez, Batet, and Montserrat. C-sanitized: A privacy model for document redaction and sanitization: C-sanitized: A privacy model for document redaction and sanitization. *Journal of the Association for Information Science and Technology*, 67, 06 2014. doi: 10.1002/asi.23363.

[17] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.