Anonimitat de dades en Python

David Belmonte García

Resum– Avui dia les dades són una part molt important de la nostra societat i la seva privacitat encara ho és més. Aquest projecte es centra sobre un aspecte concret de la privacitat: l'anonimitat, que estudia com evitar la re-identificació dels individus en un conjunt de dades, tot i haver esborrat els camps identificadors. L'anonimitat permet que les dades es puguin compartir més fàcilment, almenys des d'un punt de vista legal, amb altres empreses o persones per al seu posterior estudi, com és el cas de la mineria de dades, que no requereix conèixer específicament a ningú sinó més aviat identificar patrons que són comuns per a un conjunt de registres de les dades. En aquest article s'explica el funcionament d'uns mètodes que permeten aplicar anonimitat a un conjunt dades, com s'han desenvolupat aquests mètodes a Python i quins han sigut els resultats dels experiments fets sobre les funcions desenvolupades, els quals han sigut comparats amb els resultats dels mateixos experiments fets sobre els mateixos mètodes provinents d'una llibreria del llenguatge de programació R.

Paraules clau– anonimització, privacitat, sdcMicro, dades, Python, pandas

Abstract– Nowadays, data is a very important part of our society and its privacy even more. This projects focuses on one specific aspect of privacy: anonymity, which studies how to avoid reidentification of individuals in a dataset, even when the identifier fields are removed. Anonymity allows data to be shared more easily, at least from a legal perspective, with other companies or people to study it, as it is the case of data mining which doesn't require knowing any specific individuals, but rather it identifies common patterns in a set of registers inside the data. In this article we explain how some methods that apply anonymity to a dataset work, how these methods were developed in Python and which were their results after running a series of experiments on the functions, which were compared to the results obtained from running the same experiments on the same functions from a library from the R programming language.

Keywords– anonymization, privacy, sdcMicro, data, Python, pandas

1 Introducció - Context del treball

Avui dia per Internet viatja una gran quantitat de dades, amb expectatives de que aquesta quantitat creixi cada any [1]. Moltes d'aquestes dades són personals, les quals són recopilades per empreses que en fan un ús específic com ara mineria de dades, però l'aparició de la llei europea de protecció de dades (GDPR) [2] al 2018 provoca que moltes empreses hagin d'adaptar-se a aquesta normativa, vigilant quin tractament fan amb les dades. No obstant, aquesta llei es pot tornar més laxa permetent que les empreses puguin guardar dades personals durant el temps que vulguin i fer-les servir per a qualsevol finalitat, sempre i quan les dades no continguin elements identificadors. És aquí on entra l'anonimitat de dades, que estudia com es pot evitar la identificació de registres específics dins dels conjunts de dades, encara que s'hagin esborrat els elements que identifiquen directament a cada registre, mantenint una certa utilitat a les dades per a que puguin ser posteriorment estudiades.

Hi ha molts tipus de mètodes d'anonimització o protecció, però bàsicament es poden classificar en mètodes pertorbatius i no pertorbatius, el que significa que o afegeixen informació errònia a les dades (soroll) o bé substitueixen la informació per una de menys específica. També hi ha mètodes que permeten avaluar quin és el nivell de protecció i d'utilitat de les dades protegides.

El propòsit d'aquest article és el d'estudiar i desenvolupar a Python 4 mètodes de protecció pertorbatius i 2 d'avaluació de la protecció que puguin fer servir científics de dades o qualsevol altre persona que necessiti anonimitzar els seus conjunts de dades. Per a poder saber si les implementacions

• E-mail de contacte: davidbelmonte.garci@gmail.com
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Guillermo Navarro-Arribas (DEIC)
• Curs 2020/21

desenvolupades són prou bones, es compararan amb els mateixos mètodes implementats a una llibreria de referència en el camp de l'anonimitat de dades, en el llenguatge de programació R: sdcMicro [3]. Aquests mètodes treballen amb un atribut o columna alhora, pel que quan s'aplica anonimitat a dades amb diversos atributs, cal aplicar els mètodes desenvolupats per a cada atribut.

En aquest article primerament s'explica quines opcions populars existeixen per a poder anonimitzar dades, quins són els objectius del projecte, la metodologia que s'ha seguit per a desenvolupar els mètodes i comparar-los, i la planificació que s'ha fet per a poder portar a terme el projecte. Seguidament s'explica el funcionament de cada mètode desenvolupat i quins resultats ha tingut cadascun, comparats amb els de la llibreria de R. Finalment, l'article acaba amb les conclusions d'aquest projecte, així com les línies de treball a seguir en un futur.

2 ESTAT DE L'ART

Actualment existeixen programes i llibreries que permeten protegir un conjunt de dades de diverses maneres, com ARX Data Anonymization Tool [4], Amnesia [5] o bé sdcMicro. En aquest projecte s'ha decidit per comparar els resultats obtinguts en aquest projecte amb els de sdcMicro, que és una llibreria important per a l'anonimització de dades implementada en R que permet també avaluar la protecció aplicada a les dades. També s'ha decidit utilitzar Python 3.7 degut a que és un llenguatge de programació molt popular.

3 OBJECTIUS

Els objectius del projecte, en l'ordre de prioritat que cal realitzar-los, són:

1. Implementar un mòdul a Python que permeti protegir i avaluar la privacitat de les dades, tractades amb la llibreria *pandas* [6]. Es pretén implementar els mètodes següents i en aquest ordre de prioritat:
   (a) Soroll additiu no correlacionat
   (b) Soroll multiplicatiu no correlacionat
   (c) *Rank swapping*
   (d) Estimació del risc d'identificació basada en la distància amb desviació estàndard
   (e) Microagregació univariant
   (f) Estimació de la pèrdua d'informació basada en la distància estandaritzada amb desviació estàndard (IL1s)
2. Utilitzar les tècniques implementades sobre un conjunt de dades no protegides i estudiar els resultats obtinguts, comprovant la utilitat i la privacitat de les noves dades generades amb els mètodes implementats. S'utilitzaran gràfiques per ajudar a veure els resultats d'una manera visual.
3. Comparar els resultats dels mètodes implementats amb els de la llibreria sdcMicro de R. També s'inclou la comparació en temps d'execució del rendiment de les funcions.

4 METODOLOGIA

En aquest projecte s'ha intentat seguir la metodologia de la programació extrema [7], ja que és una metodologia àgil que té unes certes característiques que la fan interessant: simplicitat del codi, comunicació amb el client (en aquest cas tutor) i desenvolupament iteratiu.

S'ha anat iterant el projecte amb reunions amb el tutor a mesura que s'anaven completant les diverses parts del codi, afegint més funcionalitats o bé optimitzant les ja existents. També s'ha intentat mantenir una certa simplicitat en el codi, basant-se en les 4 regles del disseny simple [8]:

1. Es passen els tests (en aquest cas unitaris, mitjançant l'eina *pytest*)
2. Es mostra la intenció del programador (codi relativament fàcil de llegir)
3. Eliminar la duplicitat a la lògica del codi (com el principi de *Don't Repeat Yourself*)
4. Eliminar elements (classes o mètodes) que no siguin realment necessaris

En el cas d'aquest projecte, les dues primeres regles han sigut les que més s'han intentat seguir, ja que no hi ha massa complexitat d'estructura lògica en el codi. Per últim, s'ha intentat mantenir una bona comunicació amb el tutor del projecte, ja sigui per escrit o per veu, per tal d'anar adaptant el projecte a possibles modificacions no planificades inicialment.

5 PLANIFICACIÓ

La planificació d'aquest projecte s'ha fet amb un diagrama de Gantt, i utilitza un nivell de granularitat el qual ha permès ser més precís a l'hora de saber què cal fer a cada fase del projecte i per tant calcular més precisament la durada de cada fase. Els detalls del diagrama de Gantt es poden trobar a l'apèndix, però bàsicament s'ha dividit el projecte en 5 etapes que gairebé coincideixen amb les del Treball de Final de Grau: fase inicial de 3 setmanes , fase de desenvolupament del codi de 6 setmanes, fase d'obtenció de resultats de 5 setmanes, fase d'extracció de conclusions de 3 setmanes i fase de presentació del projecte de 3 setmanes. Aquesta planificació ha sigut pràcticament la que s'ha seguit durant el transcurs del projecte, només arribant a canviar d'ordre algunes tasques durant la fase de desenvolupament.

6 DESENVOLUPAMENT

S'han desenvolupat a Python tant funcions que implementen mètodes de protecció de les dades com mètodes d'avaluació de la privacitat, utilitzant la llibreria *pandas* per a manipular les dades.

6.1 Mètodes d'avaluació de la privacitat

Aquests mètodes estudien quin és el grau d'utilitat de les dades (com més s'assemblin els valors protegits als originals, més utilitat tenen), o bé quina és la probabilitat de re-identificació d'algun registre qualsevol (com més s'apropin els valors protegits als originals, més risc de re-identificació

hi ha). Per tant, abans d'aplicar aquests mètodes s'han de fer servir els mètodes de protecció per tal d'avaluar aquesta protecció, i també se'ls hi cal passar per paràmetre tant les dades protegides com les originals. Les versions estudiades en aquest projecte actuen sobre dades numèriques.

6.1.1 Estimació del risc d'identificació

La estimació del risc d'identificació consisteix en avaluar quina és la probabilitat de poder re-identificar a un registre d'un conjunt de dades protegit, és a dir, de trobar informació de l'individu que hi ha darrera de les dades protegides. El mètode estudiat en aquest projecte es basa en utilitzar la desviació estàndard de les dades protegides i un paràmetre k que representa el percentatge d'aquesta desviació que s'utilitzarà.

L'algorisme d'aquest mètode es basa en calcular un interval de risc per a cada registre de les dades protegides i mirar si el valor original d'aquell registre es troba dins d'aquest interval i si és així, es considera que aquell registre s'ha re-identificat amb un valor de 1, i si no amb un valor de 0. La mitjana d'aquests valors dona el percentatge de registres que han sigut re-identificats i és directament el valor retornat pel mètode. L'interval de risc que es calcula es pot expressar de la següent manera:

$$interval = [c - (k * \sigma), c + (k * \sigma)]$$

On c és el valor del registre actual protegit, σ és la desviació estàndard de les dades protegides i k és el percentatge de la desviació estàndard que es passa per paràmetre i que controla com de gran és l'interval dins del qual es considera que el registre original es pot re-identificar. Com més gran sigui k més fàcil serà re-identificar cada registre, i com més petit sigui més difícil serà. Aquest mètode s'ha implementat a la funció *risk* a Python.

6.1.2 Estimació de la pèrdua d'informació

La estimació de la pèrdua d'informació permet avaluar quina és la pèrdua (o inversament la utilitat) que tenen les dades protegides respecte les dades originals. En aquest mètode, la utilitat és més alta quan més propers són els valors protegits respecte els originals (és a dir, menys pèrdua) i més baixa quan més distants. En aquest projecte s'estudia una versió del mètode que es diu IL1s i es basa en la fórmula de [9]. La versió estudiada calcula la diferència entre cada registre protegit amb el seu respectiu valor original, dividida per la desviació estàndard de les dades originals. Aquestes diferències es sumen i són el resultat retornat pel mètode. L'algorisme es pot formalitzar de la següent manera:

$$IL1s = \sum_{i=1}^{n} \frac{|x_i - x'_i|}{\sqrt{2} * \sigma}$$

On n és el número de registres del conjunt de dades, x_i és l'i-èssim registre de les dades originals, x'_i és l'i-èssim registre de les dades protegides i σ és la desviació estàndard de les dades original. Per tant, com més alt sigui el valor retornat, més s'allunyen les dades protegides de les originals i per tant menys utilitat tenen les dades. Aquest mètode està implementat a la funció *il1s* a Python.

6.2 Mètodes de protecció de la privacitat

Aquests mètodes s'encarreguen d'anonimitzar les dades de tal manera que es redueix la probabilitat de re-identificació d'un registre qualsevol. La manera concreta en la que es fa pot ser molt variada, fins i tot dins de cada mètode depenent dels valors dels paràmetres que reben. En aquest projecte s'han estudiat i desenvolupat 4 mètodes pertorbatius diferents, i com els mètodes d'avaluació, les versions estudiades en aquest projecte actuen sobre dades numèriques.

6.2.1 Soroll additiu no correlacionat

El mètode de soroll additiu permet afegir error o soroll a dades numèriques mitjançant l'operació de suma o addició per tal de que els valors protegits estiguin allunyats dels valors originals. El mètode es pot formalitzar de la següent manera simplificada:

$$X' = X + \varepsilon$$

On X' és el conjunt de dades protegides, X són les dades originals i ε és el soroll o error afegit. Aquest soroll s'aplica a cada registre de les dades, i és generat de manera que segueix una distribució normal amb una mitjana μ de 0 i una desviació estàndard σ proporcional a la de les dades a protegir. Aquesta desviació es pot controlar amb un paràmetre p que representa quin percentatge d'aquesta desviació es vol utilitzar a l'hora de generar l'error, per tant com més gran sigui aquesta p més soroll s'afegeix a les dades, i com més petita sigui, menys error s'afegeix. La següent notació permet formalitzar la generació del soroll additiu:

$$N(\mu, p * \sigma) \quad per\ \mu = 0$$

On $N()$ indica que es tracta d'una distribució normal, μ és la mitjana aritmètica de la distribució, σ és la desviació estàndard i p és el percentatge de la desviació estàndard que es controla per paràmetre.

En aquest cas es treballa amb el soroll additiu no correlacionat, que vol dir que quan s'aplica el mètode a diverses variables, els errors afegits no estan correlacionats entre si (la covariància dels errors és de 0). El mètode està implementat a la funció *additiveNoise*.

6.2.2 Soroll multiplicatiu no correlacionat

El mètode de soroll multiplicatiu és molt semblant al del soroll additiu: s'afegeix un cert error a unes dades numèriques de tal manera que l'error es multiplica amb cada valor per tal de protegir-les. La seva formalització simplificada és la següent:

$$X' = X * \varepsilon$$

Anàlogament al soroll additiu, aquest error segueix una distribució normal amb una mitjana μ de 1 i una desviació estàndard σ proporcional a la de les dades a protegir. Aquesta desviació es pot controlar amb un paràmetre p que representa el percentatge d'aquesta desviació que es vol utilitzar per a generar l'error ε. La següent notació permet formalitzar la generació del soroll multiplicatiu:

$$N(\mu, p * \sigma) \quad per\ \mu = 1$$

En aquest cas, a la funció desenvolupada s'han ignorat tots els errors que són negatius, ja que al tractar-se d'una multiplicació s'arribaria a aplicar un gran error a les dades. Aquest mètode s'ha implementat a la funció *multiplicativeNoise*.

6.2.3 Rank swapping

El "rank swapping" es un altre mètode de protecció de dades el qual no afegeix soroll directament, sinó que ordena els registres de manera ascendent i intercanvia les posicions dels registres que es trobin dins d'un mateix rang. S'han desenvolupat 2 mètodes: un amb la versió "original" de l'algorisme anomenat *rankSwap* i un altre que és una variant pròpia del mateix mètode anomenat *rankSwapMk2* (nom de la funció a Python). Totes dues versions es basen en intercanviar aleatòriament registres que es trobin dins de rangs que es poden formalitzar de la següent manera:

$$rang = k * n$$

On n és el número de registres que hi ha a les dades i k és un paràmetre que determina quin percentatge de les dades formen el rang.

A les següents figures s'il·lustra el funcionament d'ambdues versions de "rank swapping" desenvolupades:

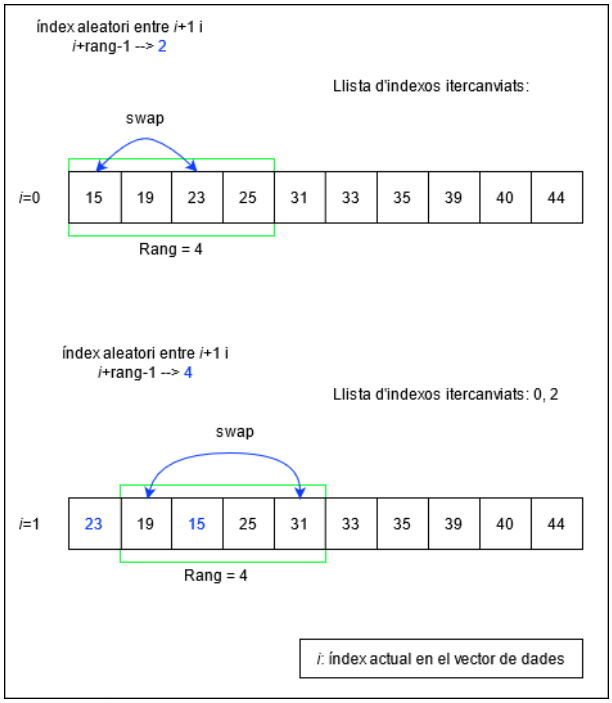


Fig. 1: Algorisme de rank swapping amb dues iteracions.

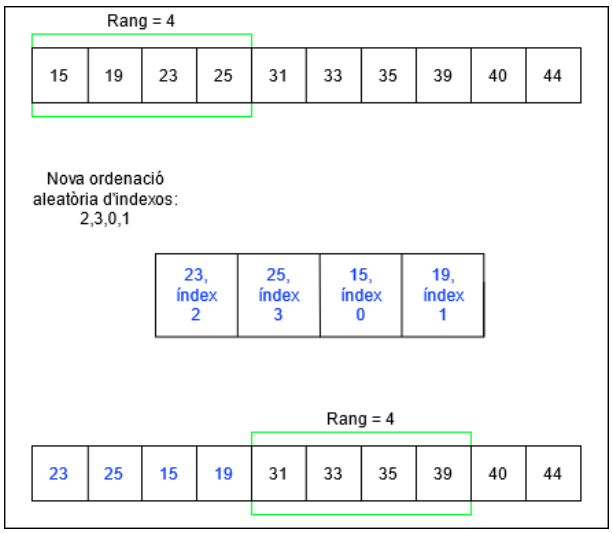


Fig. 2: Versió modificada de l'algorisme de rank swapping.

La principal diferència entre ambdós mètodes és que amb la versió original el rang es va desplaçant cada cop que s'intercanvia el registre que es troba a la primera posició del rang (s'intercanvia sempre el primer amb algun registre aleatori dins del rang), mentre que amb la versió pròpia es forma un rang, es re-ordenen tots els valors dins del rang entre ells i finalment es segueix amb el següent rang.

6.2.4 Microagregació univariant

La microagregació univariant (funció *microaggregation* a Python) consisteix en agrupar valors propers entre ells i formar clústers de k elements i substituir-los per la seva agregació, el que permet obtenir k-anonimitat, que vol dir que existeixen almenys k elements indistingibles entre ells. El mètode és univariant perquè només actua sobre 1 sol atribut o columna a la vegada. L'algorisme de microagregació que s'implementa en aquest projecte és l'algorisme MDAV degut a la seva popularitat. Aquest calcula la mitjana actual de totes les dades, busca el valor més llunyà a aquesta mitjana i d'aquest valor es busca els $k-1$ valors més propers per tal de formar un grup de k valors els quals són substituïts per una funció d'agregació. Aquest algorisme es repeteix fins que tots els elements de les dades s'hagin agrupat.

La k és un paràmetre que se li passa al mètode i la funció d'agregació també pot ser un paràmetre però en el cas d'aquest projecte s'ha decidit implementar la microagregació de manera que utilitzi per defecte com a funció d'agregació la mitjana, és a dir, per a cada grup de valors es calcula la seva mitjana i tots els valors del grup es substitueixen amb aquesta mitjana.

Per tal d'entendre millor el funcionament d'aquest mètode, la següent figura il·lustra una iteració de l'algorisme de microagregació:

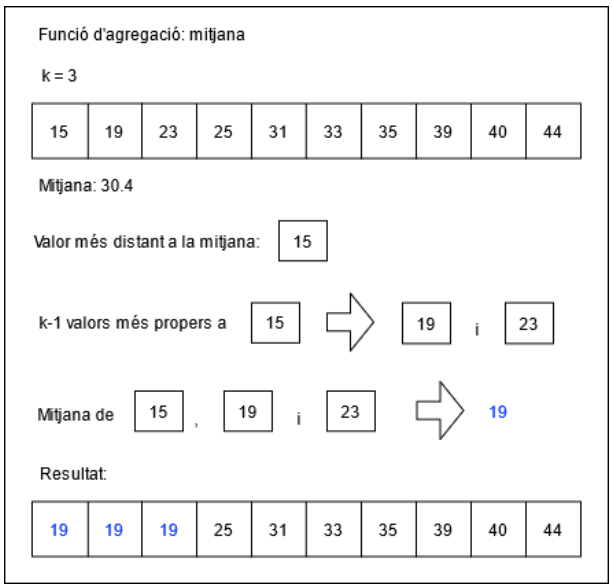


Fig. 3: Algorisme de microagregació simplificat.

7 Resultats

Les funcions desenvolupades s'han aplicat sobre un conjunt de dades de 148651 registres i un altre de 714 registres, provinents tots 2 de *Kaggle* [10], una pàgina web en la que es poden pujar conjunts de dades per a que altres usuaris en facin ús. El conjunt petit tracta amb dades dels passatgers del Titanic i el conjunt gran tracta amb dades salarials de diferents treballadors a San Francisco. Del conjunt petit s'ha protegit el camp *Age* (edat) i del conjunt gran el camp *TotalSalary* (salari), i als conjunts protegits resultants se'ls ha aplicat les funcions d'avaluació de la protecció desenvolupades a Python. També s'han realitzat tests de rendiment per a saber el temps d'execució mitjà de cada funció, així com també s'han extret la mitjana aritmètica i la desviació estàndard de les dades protegides per a poder comparar-les amb les de les dades originals.

Per a obtenir resultats més representatius s'han realitzat 100 iteracions sobre cada mètode, els quals s'han comparat amb els respectius resultats obtinguts a les funcions de sdcMicro. Aquestes comparacions s'han fet amb taules comparatives amb els valors promig de les 100 iteracions, histogrames de les dades protegides i gràfiques que mostren l'error ordenat ascendentment que s'ha afegit a cada mètode (en aquest cas, l'error s'ha considerat com la diferència entre els valors originals i els valors protegits). També cal dir que totes les funcions que reben un paràmetre que les controla, s'han executat amb valors que no siguin molt extrems i facin que els resultats d'executar les funcions d'avaluació siguin valors que no permetin comparar bé les funcions de protecció amb les de sdcMicro.

Cal dir que les funcions encara que implementin els mateixos algorismes no necessàriament produiran els mateixos resultats, doncs poden haver components aleatoris que facin que els resultats difereixin, així com diferències entre els llenguatges de programació i en les llibreries utilitzades, que no necessàriament s'implementen de la mateixa manera.

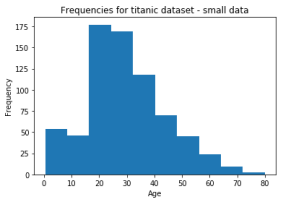
Primer de tot, cal conèixer quins histogrames i estadístiques tenen les dades sense protegir per tal de tenir una referència sobre la qual comparar els resultats:

TAULA 1: Taula resum amb els resultats estadístics del conjunt de dades petit

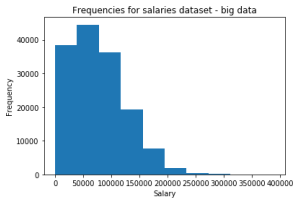
	Conjunt de dades petit (Edat)
número de registres	714
mitjana aritmètica	29.6991
desviació estàndard	14.5265

TAULA 2: Taula resum amb els resultats estadístics del conjunt de dades gran

	Conjunt de dades gran (Salari)
número de registres	148651
mitjana aritmètica	74759
desviació estàndard	50476



(a) Histograma del conjunt de dades petit



(b) Histograma del conjunt de dades gran

Fig. 4: Histogrames dels conjunts de dades estudiats

Un cop especificades les bases dels experiments, ja es poden mostrar els resultats de cada funció. Per tal d'evitar confusions amb els noms de les funcions, a continuació es presenta una taula amb els diferents noms de les funcions que s'han desenvolupat a Python, junt amb les respectives funcions homologues a sdcMicro:

TAULA 3: Taula amb els noms de cada funció a Python i a sdcMicro

Python	**sdcMicro**
risk	*dRisk*
il1s	*dUtility*
additiveNoise	*addNoise*
multiplicativeNoise	-
rankSwap, rankSwapMk2	*rankSwap*
microaggregation	*microaggregation*

7.1 *risk* i *dRisk*

La funció anomenada *risk* és comparada amb *dRisk* de sdcMicro i s'han executat ambdues funcions amb un paràmetre $k = 0.2$. El valor de retorn és un valor decimal entre 0 i 1, que representa un percentatge de risc de re-identificació. A continuació es mostren dues taules que comparen els resultats de les dues funcions:

TAULA 4: TAULA AMB ELS RESULTATS DE LES FUNCIONS RISK I DRISK SOBRE EL CONJUNT DE DADES PETIT

	risk	*dRisk*
Resultat promig	0.6695	0.6762
Rendiment promig (en segons)	0.0012	0.0013

TAULA 5: TAULA AMB ELS RESULTATS DE LES FUNCIONS RISK I DRISK SOBRE EL CONJUNT DE DADES GRAN

	risk	*dRisk*
Resultat promig	0.6931	0.7006
Rendiment promig (en segons)	0.0070	0.1097

Quan les funcions tracten amb el conjunt de dades petit, tenen un rendiment i donen uns resultats similars (difereixen al voltant d'un 1%), mentre que amb el conjunt de dades gran el rendiment de *dRisk* és unes 15 vegades pitjor que el de la funció desenvolupada. Aquestes diferències poden ser degudes al propi llenguatge de programació, ja que el codi desenvolupat és molt similar al codi de la funció de sdcMicro.

7.2 *il1s* i *dUtility*

La funció anomenada *il1s* és comparada amb *dUtility* de sdcMicro i retornen la distància entre els valors originals i els de les dades protegides (per tant retornen un valor entre 0 i n). A continuació es mostren dues taules que comparen els resultats de les dues funcions:

TAULA 6: TAULA AMB ELS RESULTATS DE LES FUNCIONS IL1S I DUTILITY SOBRE EL CONJUNT DE DADES PETIT

	il1s	*dUtility*
Resultat promig	82.8266	83.65
Rendiment promig (en segons)	0.0008	0.0005

TAULA 7: TAULA AMB ELS RESULTATS DE LES FUNCIONS IL1S I DUTILITY SOBRE EL CONJUNT DE DADES GRAN

	il1s	*dUtility*
Resultat promig	16754	16914
Rendiment promig (en segons)	0.0076	0.0416

Ambdues funcions donen resultats similars per al conjunt de dades petit (difereixen en menys d'un 1%), però amb el conjunt de dades gran la funció *dUtility* és unes 5 vegades pitjor que la funció desenvolupada a Python. Aquestes diferències poden ser degudes al propi llenguatge de programació i les llibreries utilitzades.

7.3 *additiveNoise* i *addNoise*

El soroll additiu vé implementat a la funció *additiveNoise*, que és comparada amb *addNoise* de sdcMicro. S'han executat ambdues funcions amb un paràmetre de soroll $p = 0.2$. A continuació es mostren dues taules que comparen els resultats de les dues funcions:

TAULA 8: TAULA AMB ELS RESULTATS DE LES FUNCIONS ADDITIVENOISE I ADDNOISE SOBRE EL CONJUNT DE DADES PETIT

	additiveNoise	*addNoise*
Pèrdua d'informació promig	80.7796	81.3743
Risc d'identificació promig	0.6910	0.7010
Mitjana d'edat promig	29.7152	30
Desviació estàndard promig	14.8088	14.96
Rendiment promig (en segons)	0.0003	0.0025

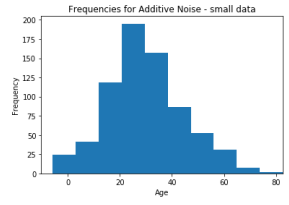
TAULA 9: TAULA AMB ELS RESULTATS DE LES FUNCIONS ADDITIVENOISE I ADDNOISE SOBRE EL CONJUNT DE DADES GRAN

	additiveNoise	*addNoise*
Resultat *il1s* promig	16772	16943
Resultat *risk* promig	0.6923	0.6990
Mitjana de salari promig	74766	75506
Desviació estàndard promig	51477	51994
Rendiment promig (en segons)	0.0078	0.0919

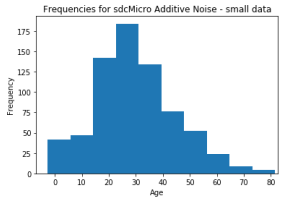
Ambdues funcions obtenen uns resultats similars, tot i que el rendiment de les funcions de Python és millor i els resultats estadístics s'acosten una mica més als originals que els de sdcMicro. Les diferències en rendiment poden ser degudes a diferències en el propi llenguatge de programació i llibreries utilitzades.

Respecte a la mitjana i la desviació estàndard, ambdues funcions s'allunyen menys del 5% dels respectius valors originals.

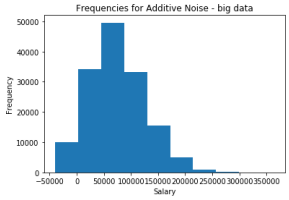
A continuació es mostren els histogrames corresponents als 2 conjunts de dades protegides per cada funció, així com les gràfiques que comparen l'error afegit per les dues funcions:



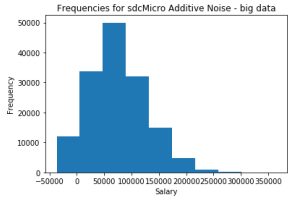
(a) Histograma del conjunt de dades petit protegit amb *additiveNoise*



(b) Histograma del conjunt de dades petit protegit amb *addNoise*

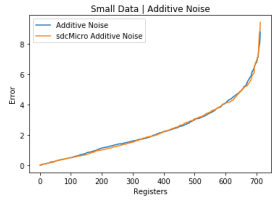


(c) Histograma del conjunt de dades gran protegit amb *additiveNoise*

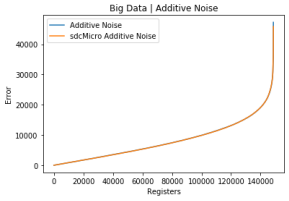


(d) Histograma del conjunt de dades gran petit protegit amb *addNoise*

Fig. 5: Histogrames del soroll additiu



(a) Gràfica comparativa de l'error afegit al conjunt de dades petit pels 2 mètodes



(b) Gràfica comparativa de l'error afegit al conjunt de dades gran pels 2 mètodes

Fig. 6: Gràfiques comparatives del soroll additiu

Els histogrames mostren que les freqüències són molt similars entre elles, i respecte a les dades originals apareixen valors negatius que abans no hi eren en ambdós conjunts, el que és degut a que s'ha afegit un cert error que ha fet que valors que eren petits passin a ser negatius. A les gràfiques que mostren l'error afegit es pot veure com l'error és pràcticament el mateix, sent les diferències amb el conjunt de dades gran gairebé imperceptibles i més evidents amb el conjunt de dades petit. Els últims registres tenen més error perquè precisament el mètode ha generat un error més alt per a aquests registres. Aquestes gràfiques demostren que el mètode implementat a Python té uns resultats molt similars als del seu anàleg a R.

7.4 *multiplicativeNoise*

El soroll multiplicatiu està implementat a la funció *multiplicativeNoise*, la qual no es pot comparar amb una funció anàloga de sdcMicro perquè no existeix cap. No obstant, es mostraran els resultats obtinguts per aquesta funció. S'ha executat la funció amb un paràmetre de soroll $p = 0.3$. A continuació es mostren dues taules que mostren els resultats de la funció:

TAULA 10: TAULA AMB ELS RESULTATS DE LA FUNCIÓ MULTIPLICATIVENOISE SOBRE EL CONJUNT DE DADES PETIT

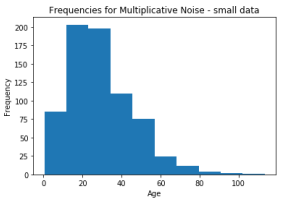
	multiplicativeNoise
Resultat *il1s* promig	247.7623
Resultat *risk* promig	0.3765
Mitjana d'edat promig	29.7151
Desviació estàndard promig	17.5809
Rendiment promig (en segons)	0.0002

TAULA 11: TAULA AMB ELS RESULTATS DE LA FUNCIÓ MULTIPLICATIVENOISE SOBRE EL CONJUNT DE DADES GRAN

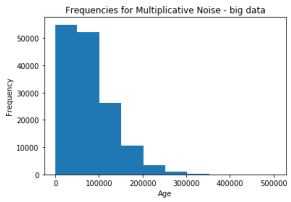
	multiplicativeNoise
Resultat *il1s* promig	37265
Resultat *risk* promig	0.5059
Mitjana de salari promig	74753
Desviació estàndard promig	57264
Rendiment promig (en segons)	0.0067

La mitjana està a una distància petita respecte al valor original (<1% per a les dades petites i un 17% per a les dades grans), i similarment amb la desviació estàndard (<1% i un 12%).

A continuació es mostren els histogrames associats a les dades protegides per aquesta funció:



(a) Histograma del conjunt de dades petit protegit amb *multiplicativeNoise*



(b) Histograma del conjunt de dades gran protegit amb *multiplicativeNoise*

Fig. 7: Histogrames del soroll multiplicatiu

Aquestes gràfiques mostren que les freqüències respecte les dades originals són similars, però apareixen valors en rangs en els que a les dades originals no hi havia cap, com ara edats per sobre dels 80 i salaris per sobre dels 400000, el que redueix la utilitat de les dades.

7.5 *rankSwap* i *rankSwapMk2*

El "rank swapping" vé implementat a la funció *rankSwap*, que és comparada amb la funció del mateix nom de sdcMicro. No obstant, com ja s'ha mencionat anteriorment, també s'ha desenvolupat una variant de la funció anomenada *rankSwapMk2*, la qual és comparada a la vegada amb la versió original del "rank swap". S'han executat les funcions amb un paràmetre $k = 0.2$. Seguidament es mostren dues taules que comparen els resultats de les funcions:

TAULA 12: TAULA AMB ELS RESULTATS DE LES FUNCIONS RANKSWAP, RANKSWAPMK2 I SDCMICRO RANKSWAP SOBRE EL CONJUNT DE DADES PETIT

	rankSwap	*rankSwapMk2*	*sdcMicro rankSwap*
Pèrdua d'informació promig	264.8834	157.7202	559.0935
Risc d'identificació promig	0.2225	0.4490	0.1068
Mitjana d'edat promig	29.6991	29.6991	29.6991
Desviació estàndard promig	14.5265	14.5265	14.67
Rendiment promig (en segons)	0.1652	0.0888	0.0014

TAULA 13: TAULA AMB ELS RESULTATS DE LES FUNCIONS RANKSWAP, RANKSWAPMK2 I SDCMICRO RANKSWAP SOBRE EL CONJUNT DE DADES GRAN

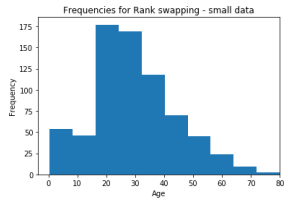
	rankSwap	*rankSwapMk2*	*sdcMicro rankSwap*
Resultat *il1s* promig	51686	29014	119211
Resultat *risk* promig	0.2847	0.5327	0.1269
Mitjana de salari promig	74759	74759	74759
Desviació estàndard promig	50476	50476	50476
Rendiment promig (en segons)	217	11.9252	18.35

El rendiment de la versió original de "rank swapping" és molt pitjor que el de la versió de sdcMicro, ja que triga uns 217 segons per al conjunt gran de dades en comparació als 18 segons de la funció de sdcMicro. Aquesta diferència de rendiment podria deure's a que la funció de sdcMicro està realment implementada en C, però importada cap a R (C al ser un llenguatge de programació compilat té un millor rendiment que no Python, que és un llenguatge interpretat).

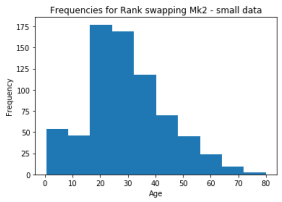
També es pot veure com la funció *rankSwapMk2* destaca amb el seu rendiment, que és el millor, així com els seus

resultats d'utilitat que indiquen que són més propers als valors originals respecte les altres funcions a canvi de tenir un major risc de re-identificació. A més, la funció de sdcMicro introdueix molt més error que no pas la seva homòloga de Python (*rankSwap*), el que fa pensar que la funció no s'implementa de la mateixa manera. En totes les funcions, la mitja i la desviació estàndard es manté, el que és normal perquè el "rank swap" no afegeix soroll que modifiqui les dades, només intercanvia valors.

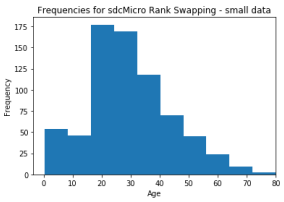
A continuació es mostren els histogrames corresponents als 2 conjunts de dades protegides per cada funció, així com la gràfica que compara l'error afegit per les funcions:



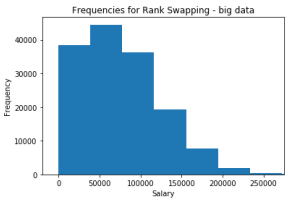
(a) Histograma del conjunt de dades petit protegit amb *rankSwap*



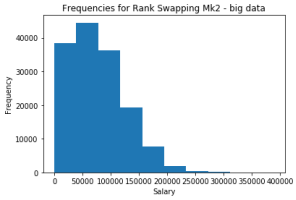
(b) Histograma del conjunt de dades petit protegit amb *rankSwapMk2*



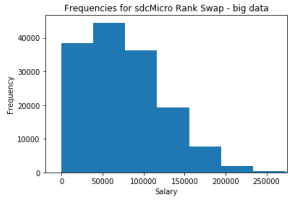
(c) Histograma del conjunt de dades petit protegit amb *sdcMicro rankSwap*



(d) Histograma del conjunt de dades gran protegit amb *rankSwap*

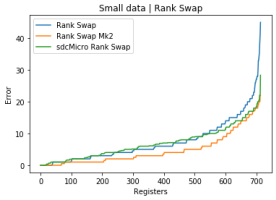


(e) Histograma del conjunt de dades gran protegit amb *rankSwapMk2*

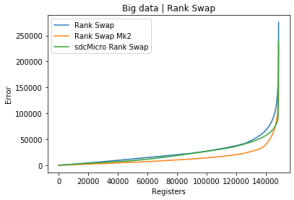


(f) Histograma del conjunt de dades gran protegit amb *sdcMicro rankSwap*

Fig. 8: Histogrames del rank swap



(a) Gràfica comparativa de l'error afegit al conjunt de dades petit pels 3 mètodes



(b) Gràfica comparativa de l'error afegit al conjunt de dades gran pels 3 mètodes

Fig. 9: Gràfiques comparatives del rank swap

Els histogrames de la versió original de "rank swap" són iguals entre ells, i en comparació del conjunt de dades gran original, valors que abans estaven entre 300000 i 400000 passen a estar en un altre rang, el que redueix el seu risc de re-identificació. En canvi l'histograma de la variant de "rank swap" es manté pràcticament igual en tots 2 conjunts de dades.

Respecte les gràfiques comparatives de l'error, es pot veure com en general la funció *rankSwapMk2* és la que introdueix menys error, seguit per la funció de sdcMicro i per últim la seva funció homòloga de Python. Els últims registres tenen més error perquè s'intercanvien valors molt distants però dins del mateix rang, el que provoca que s'afegeixi més error.

Aquestes gràfiques i resultats demostren que la versió variant de "rank swap" desenvolupada és una opció que pot arribar a ser interessant per a poder tenir un millor rendiment i major utilitat a les dades que la versió original, a canvi d'un major risc de re-identificació.

7.6 *microaggregation*

La microagregació univariant vé implementada a la funció *microaggregation*, que és comparada amb la funció del mateix nom de sdcMicro. S'han executat ambdues funcions amb un paràmetre $k = 5$ i s'ha especificat a la funció de sdcMicro que s'utilitzi el mètode "mdav" amb una funció d'agregació de "mean" (mitjana aritmètica) per tal de que ambdues funcions treballin en igualtat de condicions. A continuació es mostren dues taules que comparen els resultats de les dues funcions:

TAULA 14: TAULA AMB ELS RESULTATS DE LES FUNCIONS MICROAGGREGATION I SDCMICRO MICROAGGREGATION SOBRE EL CONJUNT DE DADES PETIT

	microaggregation	*sdcMicro microaggregation*
Pèrdua d'informació promig	5.9121	5.9680
Risc d'identificació promig	0.9985	1.0
Mitjana d'edat promig	29.6991	29.6991
Desviació estàndard promig	14.5197	14.66
Rendiment promig (en segons)	0.4645	0.0116

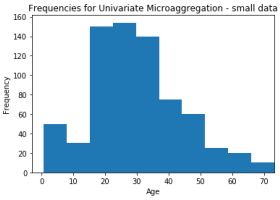
TAULA 15: TAULA AMB ELS RESULTATS DE LES FUNCIONS MICROAGGREGATION I SDCMICRO MICROAGGREGATION SOBRE EL CONJUNT DE DADES GRAN

	microaggregation	*sdcMicro microaggregation*
Resultat *ilIs* promig	7.3139	7.4579
Resultat *risk* promig	1.0	1.0
Mitjana de salari promig	74759	74759
Desviació estàndard promig	50476	50476
Rendiment promig (en segons)	373	0.3235

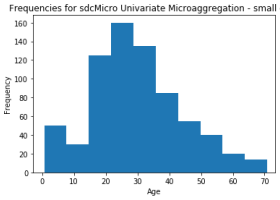
El rendiment de la funció de Python és molt pitjor que la versió de sdcMicro, ja que triga uns 373 segons per al conjunt gran de dades, en comparació als 0.3 segons de la funció de sdcMicro. Igual que a la versió de Python de "rank swapping", aquesta diferència de rendiment podria deure's a que la funció de sdcMicro està realment implementada en C, però importada cap a R.

També es pot observar com les dues funcions mantenen pràcticament la mateixa mitjana i la desviació estàndard de les dades originals, i com els resultats d'avaluació de privacitat són molt similars, el que fa pensar que les dues funcions estan seguint el mateix algorisme.

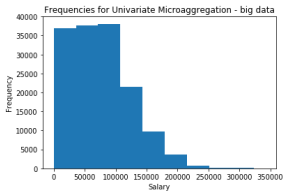
A continuació es mostren els histogrames corresponents als 2 conjunts de dades protegides per cada funció, així com la gràfica que compara l'error afegit per les dues funcions:



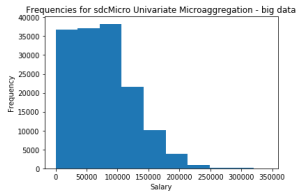
(a) Histograma del conjunt de dades petit protegit amb *microaggregation*



(b) Histograma del conjunt de dades petit protegit amb *sdcMicro microaggregation*

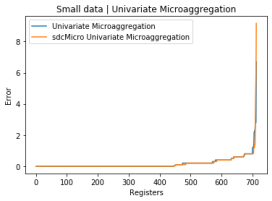


(c) Histograma del conjunt de dades gran protegit amb *microaggregation*

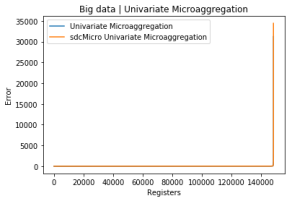


(d) Histograma del conjunt de dades gran protegit amb *sdcMicro microaggregation*

Fig. 10: Histogrames de la microagregació univariant



(a) Gràfica comparativa de l'error afegit al conjunt de dades petit pels 2 mètodes



(b) Gràfica comparativa de l'error afegit al conjunt de dades gran pels 2 mètodes

Fig. 11: Gràfiques comparatives de la microagregació univariant

Els histogrames mostren uns resultats molt similars entre ells, que són a la vegada similars als de les dades originals. Es pot destacar que respecte les dades originals, els valors més grans i amb menys freqüència s'han ajuntat amb altres valors més petits, el que ha reduït la seva probabilitat d'identificació (per exemple respecte el conjunt de dades petit els valors entre 70 i 80 anys s'han ajuntat amb els valors entre 60 i 70) .

A les gràfiques comparatives de l'error s'observa com les funcions afegeixen pràcticament el mateix error, i com al final es dona un pic en l'error afegit, el que és degut a que hi ha valors outliers que s'han agrupat amb valors que cauen més a dins de la distribució normal de les dades, el que ha provocat que s'afegeixi un gran error només amb aquests valors.

8 Conclusions

S'han pogut assolir tots els objectius dels projecte dins dels límits temporals especificats a la planificació del projecte i sense ser necessari l'ús de més recursos materials dels especificats, pel que es pot dir que el projecte ha pogut concloure satisfactòriament.

Respecte els resultats obtinguts, les funcions d'avaluació de la privacitat desenvolupades són fins i tot millors que les de sdcMicro per a poder ser utilitzades per professionals que només necessitin avaluar 1 sola variable. També cal destacar que la funció de soroll additiu és una bona alternativa de la versió de R, ja que obté resultats molt similars. Les funcions de microagregació univariant i "rank swapping" original no són tan interessants pel fet que tenen un rendiment bastant pitjor que les versions de sdcMicro. No obstant, la versió alternativa de "rank swapping" obté uns resultats que podrien ser molt interessants per als usuaris del mòdul desenvolupat de Python, doncs el rendiment és similar al de la versió original de sdcMicro.

Les línies de continuació d'aquest projecte són molt clares:

- Adaptar les funcions per a que acceptin una quantitat de variables arbitrària i no només 1 variable.
- Optimitzar les funcions de microagregació i "rank swapping" versió original, important-les des d'un altre llenguatge de programació més eficient, com ara C/C++.
- Afegir més opcions a les funcions, com ara poder passar per paràmetre a la microagregació el mètode d'agregació que es vol.

La realització d'aquestes millores faria que el mòdul de Python desenvolupat fos molt més interessant per als professionals que necessitessin aplicar anonimitat a les seves dades.

Agraïments

Vull agraïr al meu tutor de projecte Guillermo Navarro-Arribas per la seva ajuda i consells durant el desenvolupament del projecte. També agraeixo a tots els meus familiars i amics pel seu suport incondicional des de que vaig començar a fer la carrera d'Enginyeria Informàtica.

Referències

[1] Cisco Systems, "Cisco Annual Internet Report (2018–2023) White Paper" accessed: 2021-06-03. [Online]. Available: https://cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] "General Data Protection Regulation (GDPR) – Official Legal Text" accessed: 2021-05-31. [Online]. Available: https://gdpr-info.eu/

[3] M. Templ, A. Kowarik, and B. Meindl, "Statistical disclosure control methods for anonymization of microdata and risk estimation," accessed: 2021-03-10. [Online]. Available: http://sdctools.github.io/sdcMicro/

[4] ARX, "ARX - Data Anonymization Tool," accessed: 2021-06-03. [Online]. Available: https://arx.deidentifier.org/

[5] OpenAIRE, "Amnesia Anonymization Tool - Data anonymization made easy," accessed: 2021-06-03. [Online]. Available: https://amnesia.openaire.eu/

[6] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4572994

[7] D. Wells, "Extreme programming: A gentle introduction," accessed: 2021-02-25. [Online]. Available: http://www.extremeprogramming.org

[8] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999

[9] J. M. Mateo-Sanz, F. Sebé, and J. Domingo-Ferrer, "Outlier protection in continuous microdata masking," in *Privacy in Statistical Databases - PSD 2004, In Lecture Notes in Computer Science*, J. Domingo-Ferrer and V. Torra, Eds., vol. 3050. Berlin, Heidelberg:Springer, Jun 2004, pp. 201–215

[10] Kaggle, "Kaggle: Your Machine Learning and Data Science Community," accessed: 2021-03-10. [Online]. Available: https://www.kaggle.com/

APÈNDIX

A.1 Diagrama de Gantt

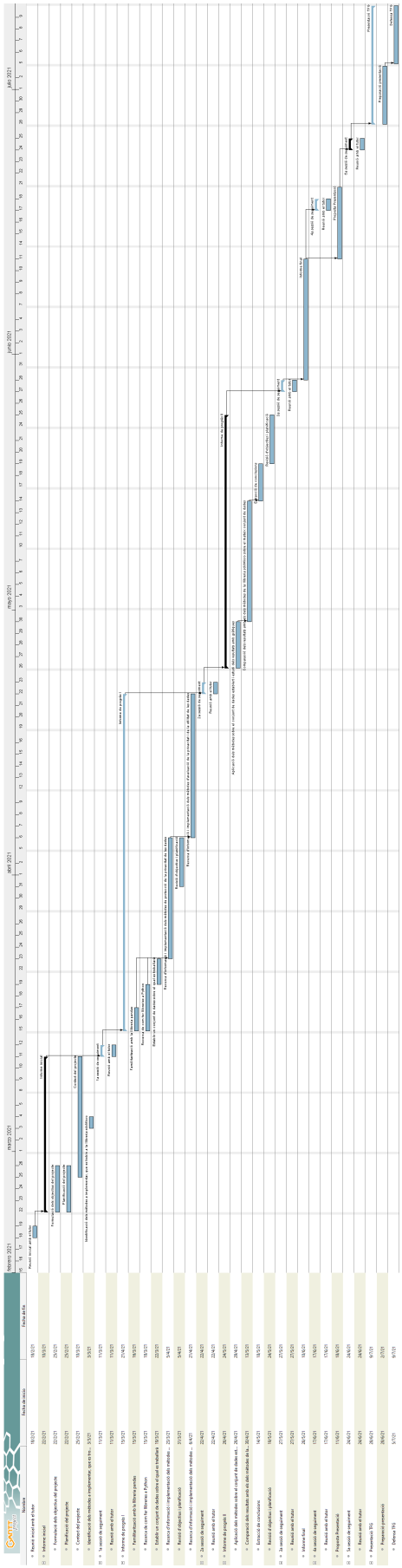


Fig. 12: Diagrama de Gantt de les tasques del projecte.