



# **Anonymisation of Transactional Data by Disassociation Techniques**

**Prajwal Nagaraja**

**26/09/2022**

**Supervisor: Dr. Jianhua Shao**

School of Mathematics,  
Cardiff University

A dissertation submitted in partial fulfilment of the  
requirements for MSc Data Science and Analytics  
by taught programme, supervised by Dr Jianhua Shao

## Executive Summary

In recent years a wide range of organisations are publishing a lot of data that's freely available to the public, but this process of publishing data can cause a lot of harm if the data being published is not anonymised. This report introduces many mathematical algorithms that's available to anonymise data and implements one of the many algorithms suggested by a Technical paper written by a Research Scientist.

The algorithm suggested, anonymises data in such a way that the data will still have some utility after the anonymisation procedure and this procedure involves Disassociating the data in such a way that terms in data that appear with each other are preserved and the terms that appear at rare instances are Disassociated as the terms that are rare can be used to pinpoint an individual to whom those rare terms are unique to. This algorithm assumes that the intruder who is trying to retrieve private information from a dataset already has background knowledge about his victim.

This report implements most of the algorithm and tests the output of the algorithm using a publicly available dataset to see how well the algorithm anonymises the dataset. To measure the performance of the algorithm, tests were carried out to measure some of the metrics that are used to estimate the performance of the anonymised and the tests reported lesser performance than the performance achieved by the tests in the technical paper where the Algorithm is from.

This drop in performance is due to the fact that the algorithm was coded in a different language and the methodology implemented was dissimilar to the methodology used in the research paper.

The paper also had some shortcomings for some of the test cases and changes to the algorithm were made to accommodate these shortcomings.

Overall most of the algorithm was implemented, the metrics were measured and improvements to the existing algorithm were suggested.

## **Abstract**

A lot of new information about a wide range of topics is being revealed with the advent of faster machines and improvements in the field of Data science which is allowing more research projects to be undertaken by research scientists, one of the main driving factors for these new discoveries is the enormous amounts of data that's either being collected or readily available for these said projects. But as they are research projects, if the insights and the data have to be published, the data has to be anonymised to ensure that privacy is preserved and there's little to no risk of a privacy breach.

Although there are many anonymisation techniques available Zhou et al.(2008)[1], this paper examines one of the Mathematical Anonymisation Techniques suggested by Terrovitis et al.(2012)[2] that can be used to Anonymise Data for these research projects or any other data in general. Some of the data used by these projects contain private information and hence has to be anonymised, without which can lead to privacy risks and sometimes even lead to monetary losses.

Incorporating evidence from other studies, personal correspondence and diaries, this study Demonstrates that with the use of Mathematical Models to Disassociate and Anonymise Data, good results can be achieved with some loss of Information. It uses a few sample Transactional Datasets to Disassociate and Anonymise data and gives an idea of the minimal losses that will be incurred when anonymising the data this way.

## **Acknowledgement**

I would like to express Gratitude towards my supervisor, Dr. Jianhua Shao for guiding, assisting and supporting me throughout this project.

# Table of Contents

<b>1. Introduction</b>	<b>5</b>
<b>1.1 Project Aim/Goal</b>	<b>5</b>
<b>1.2 Project Outcome</b>	<b>5</b>
<b>2. Background</b>	<b>5</b>
<b>2.1 Privacy Model and Attack Model</b>	<b>6</b>
<b>2.2 Anonymisation Techniques</b>	<b>7</b>
<b>2.3 K-Anonymity</b>	<b>7</b>
<b>2.4 <math>K^m</math> Anonymity</b>	<b>8</b>
<b>3. Anonymisation Algorithm</b>	<b>9</b>
<b>3.1 Horizontal Partitioning</b>	<b>9</b>
<b>3.2 Vertical Partitioning</b>	<b>10</b>
<b>3.3 Refining</b>	<b>12</b>
<b>4. Implementation</b>	<b>13</b>
<b>4.1 Horizontal Partitioning Implementation</b>	<b>14</b>
<b>4.2 Vertical Partitioning Implementation</b>	<b>15</b>
<b>5. Results</b>	<b>16</b>
<b>5.1 Processing Time</b>	<b>16</b>
<b>5.2 Top-K Deviation</b>	<b>17</b>
<b>5.3 Relative Error</b>	<b>18</b>
<b>6. Future Work</b>	<b>19</b>
<b>6.1 Refining</b>	<b>19</b>
<b>6.2 Multiple K-Value Results</b>	<b>19</b>
<b>6.3 Missing Metrics</b>	<b>19</b>
<b>7. Conclusion</b>	<b>20</b>
<b>8. References</b>	<b>20</b>

## **List of Figures**

<b>2.1 Privacy Model and Attack Model</b>	<b>6</b>
<b>2.3 K(2) - Anonymised Table</b>	<b>8</b>
<b>2.4 2<sup>2</sup> Anonymous Table</b>	<b>9</b>
<b>3.1 Horizontal Partitioning Pseudo Code</b>	<b>10</b>
<b>3.2 Vertical Partitioning Pseudo Code</b>	<b>11</b>
<b>3.2 Dissociated Output</b>	<b>12</b>
<b>3.3 Pseudo Code for Refining</b>	<b>13</b>
<b>4 Test Data Characteristics</b>	<b>13</b>
<b>4.1 Function to calculate the most common term</b>	<b>14</b>
<b>4.1 Splitting the Dataset</b>	<b>14</b>
<b>5.1 Execution Time Results</b>	<b>17</b>
<b>5.2 Top-K Deviation</b>	<b>17</b>
<b>5.2 Top-K Deviation Result</b>	<b>18</b>
<b>5.3 Relative Error</b>	<b>18</b>
<b>5.3 Relative Error Result</b>	<b>19</b>

## 1. Introduction :

With the emergence of Technology and faster processors, Data Collection by organisations such as Governments, Hospitals and Technology companies play a vital role in solving current problems and also in predicting and preparing for new problems and their solutions. The data that's collected by these organisations sometimes should be published by law in some countries. And as the data collected by these organisations contain private and sensitive information, the data published must be anonymised. Anonymisation not only preserves privacy but anonymised datasets can also be recycled and reused for other Scientific research purposes. For example, consider the article published by Abbasi, J. (Younger Adults Caught in COVID-19 Crosshairs as Demographics Shift 2020) [3] on the Demographic that was affected by the Covid-19 during the Pandemic in 2020 or consider the Election polls data which will be published during every election, these types of Datasets will contain sensitive and private information which cannot be published without changing or anonymising the data. Because if an Individual has some background information and has access to these records, he can easily Infer who the Record belongs to, which is a privacy breach. Because of all these concerns associated with the Data and Data Publishing, Privacy Preserving Data Publishing(PPDP) has become popular in the Data Science and the Research community in the past Decade and many methodologies and procedures such as Generalization, Suppression and Disassociation have been proposed and in this paper we will dive deeper into the Dissociation procedure and how this will help in preserving the privacy of the published Datasets.

### 1.1 Project Aim/Goal:

The main goal of this Thesis is to Implement a Disassociation method which separates combinations of queries or Records which could have been used to identify an Individual and Preserve Privacy and was Implemented by Terrovitis et al.(2012) [2]. I will implement this algorithm using python while also taking a closer look to propose improvements to the existing algorithm by testing extreme cases to make sure the algorithm performs efficiently and effectively in all conditions.

### 1.2 Project Outcome:

The main outcomes for this project are listed below:

- Implement, Analyse and Test one of the methods used in Privacy Preserving Data Publishing.
- Measure Performance and Loss of both the algorithms.
- Suggest Improvements to the existing algorithms.

## 2. Background:

In recent years, as technology progresses there are very few Government and Private organisations that do not employ Data mining methodologies to gain insights into their citizen's or customer's behaviour and Odlyzko, A.,(2003)[4] suggests that this has led to an increased sense of privacy intrusion within the general public. As the Organizations continue to mine data and gain further Insights, sometimes they are required by law to publish all the data collected and the insights gained from their respective research. So if a person has some background Information about his Victim such as the Victim's name or Social Security number (Explicit Identifiers - EID's) or if he has an idea of some Background Information or (Quasi Identifiers -QID's) and he also has access to publicly available Datasets that could

Potentially link some of the records in the public Dataset to the Victim's Identifiers, then he can easily narrow down the records in the public dataset to a few records and can gain access to a person's private information. This is a very big privacy risk and some Individuals consider this as an opportunity to cause harm by means of stealing Identities or even steal something of monetary value. One of the ways to effectively avoid this type of Privacy Breach is by using one of the many methods available in Privacy Preserving Data Publishing(PPDP). To further understand a few of the many concepts in PPDP, we must first understand and Investigate the Privacy Models and Attack Models.

## 2.1 Privacy Model and Attack Model:

Fung et al.(2010)[5] mainly classifies the privacy models into two types of categories; the first attack model describes a situation where an attacker is able to **Link** the Quasi Identifiers of a Record owner to a Record in the published Dataset. This type of Linking a Record to the owner of that Record mainly consists of three types of linkage, Record Linkage, Attribute Linkage and Table Linkage. In Record Linkage and Attribute Linkage, we assume that the attacker already has information about the Quasi Identifier of the Record Owner and is aware that the Victims' record can be found in the released Dataset.

The second Privacy model follows the *uninformative principle* by Machanavajjhala et al.(2006)[6] which states that "The published table should provide the attacker with little additional information beyond the background knowledge. If the attacker has a large variation between the prior and posterior beliefs, we call it the probabilistic attack". This type of Privacy model mainly focuses on how the attacker will change their probabilistic beliefs on the sensitive information of the victim after accessing the published Dataset. In other words these models would like to ensure that the attacker will gain very little information about the victim after accessing a published table.

There are many Attack Models available and different Privacy models to provide Immunity against different kinds of attacks to a varying degree. **Figure 1**(Fung et al. 2010) depicts the various kinds of Attack models and the Privacy Model used to provide immunity against the said Attack.

**Table I. Privacy Models**

Privacy Model	Attack Model			
	Record Linkage	Attribute Linkage	Table Linkage	Probabilistic Attack
$k$ -Anonymity	✓			
MultiR $k$ -Anonymity	✓			
$\ell$ -Diversity	✓	✓		
Confidence Bounding		✓		
$(\alpha, k)$ -Anonymity	✓	✓		
$(X, Y)$ -Privacy	✓	✓		
$(k, e)$ -Anonymity		✓		
$(\epsilon, m)$ -Anonymity		✓		
Personalized Privacy		✓		
$t$ -Closeness		✓		✓
$\delta$ -Presence			✓	
$(c, t)$ -Isolation	✓			✓
$\epsilon$ -Differential Privacy			✓	✓
$(d, \gamma)$ -Privacy			✓	✓
Distributional Privacy			✓	✓

**Figure 1 : Privacy and Attack Models**

## 2.2 Anonymisation Techniques:

As there are many models which an attacker can use to breach privacy from a public dataset, the data has to be anonymised before publishing. This is where the various Privacy models depicted in **Figure 1** come into picture. All the privacy models in **Figure 1** can be mapped to a few mathematical methods that are used in their anonymisation process, they are:

- Generalisation
  - Suppression
  - Anatomization
  - Perturbation
- 
- Generalisation: This method substitutes the information in a record to some generic value, this ensures that some part of the QID is hidden, generalisation technique substitutes a specific record to a more generic entry according to a given taxonomy.
  - Suppression: This anonymisation method changes some or the entire value of the QID in the record with a designated special character(& for example) to make sure that the content of the record is not revealed.
  - Anatomization: This method isolates the Quasi Identifiers(QID's) from its respective sensitive attribute. To achieve this it generates two separate tables, one for the QID and the other for the sensitive attribute with a common attribute that links both the tables.
  - Perturbation: This method changes the original dataset to some synthetically generated data entries to make sure that the statistical values of the original dataset is preserved and as the perturbed dataset does not contain any values that link to a person in the real world, privacy is preserved while maintaining the statistical value.

The Anonymization technique used in this thesis mainly prevents Record Linkage by Dissociation i.e, it uses Anatomization which separates the *terms* of the original data and hence it prevents the attacker from linking one of the records in a public Dataset back to the Record owner. To further understand Record Linkage and how to combat this type of attack we need to understand a few underlying mathematical techniques that are used to anonymise data that prevents Record Linkage. One of the Techniques that are used to anonymise data is the K-Anonymity technique.


## 2.3 K - Anonymity:

According to Sweeney(2002)[7], a dataset is said to be K- anonymous if an attacker will not be able to distinguish between K records in the published dataset for example, consider the first table in **Figure 2**[8], An attacker who has prior knowledge of the victim's QID, will not be able to pinpoint the exact record which belongs to the victim in the K-anonymised second table in **Figure 2**[8] as there are K-1 Identical records in the table that contain the same QID. The probability that the attacker will be able to identify his Victim is  $1/k$ . This model assumes that the QID is already known to the data publisher and the way to K-anonymise a table is by using Generalisation and Suppression.



ID	Age	Zip	Disease
1	7	53715	Flu
2	9	55410	Diarrhea
3	13	52121	Flu
4	19	56421	Fever
5	29	02263	Diarrhea
6	34	02296	Fever
7	39	02278	Flue
8	33	02254	Diarrhea

Sensitive Table



ID	Age	Zip	Disease
1	0-20	5****	Flu
2	0-20	5****	Diarrhea
3	0-20	5****	Flu
4	0-20	5****	Fever
5	20-40	022**	Diarrhea
6	20-40	022**	Fever
7	20-40	022**	Flue
8	20-40	022**	Diarrhea

2 – Anonymized Table

Example of 2-Anonymization

**Figure 2 : K(2) - Anonymised Table**

From the second table in **Figure 2** we can see that the Age column is generalised by changing the original value to a range of values, this makes sure that the values are changed enough to not give away an exact number that an attacker might use to retrieve data. The Zip code column is suppressed by replacing some of the digits entirely with a symbol(\*) in this case) because there are not many zip codes that are repeating in the table.

One of the main drawbacks of Anonymising data this way is the utility of the anonymised tables is low because there is a lot of information that is lost during the anonymisation process and hence the anonymised tables cannot be relied upon to produce good results for Machine Learning applications. Along with low utility, this method is vulnerable to Background Knowledge Attacks wherein if an attacker knows that the target is in the anonymised table he can easily narrow down the records as he has some background knowledge about the victim.

## 2.4 $K^m$ Anonymity:

$K^m$  Anonymity is a special model derived from the K-Anonymity model and was proposed by Terrovitis et al.(2012) [2] et al.(2012)[2] et al.(2012) and is mainly used to anonymise Transactional Datasets. A Dataset is said to be  $K^m$  Anonymous if an attacker who has background knowledge of upto m items in a record, will only be able to identify less than K items in the same record. The rationale behind this algorithm is that the attacker is unlikely to have or know all the items of a user.

My Implementation of the  $K^m$  anonymity uses the anatomization Technique. Contrary to the Datasets used in K Anonymity model,  $K^m$  anonymity uses datasets that do not have a fixed or established set of QID's and sensitive data plus the transactional data do not have a fixed length as opposed to the fixed length of data in the K anonymity model.

**Figure 3** depicts a  $2^2$  Anonymous table where if an attacker has Background Information about 2(m) records, he will not be able to distinguish between the 1(k-1) items, as there are at least 2(k) records that contain them.

Rec#	2 <sup>2</sup> -anonymity
1	{LA}
2	{LA, Seattle}
3	{West US}
4	{West US}
5	{LA, Seattle, West US}
6	{LA, Seattle, West US}
7	{LA, Seattle, West US}

**Figure 3 : A 2<sup>2</sup> Anonymous Table**

My methodology mainly revolves around using the  $K^m$  anonymity model to Disassociate data but before we directly Disassociate data there's another algorithm which generates clusters of the dataset by partitioning the data according to a simple heuristic algorithm called Horizontal Partitioning.

### 3 Anonymisation Algorithm:

This thesis implements the algorithm described by Terrovitis et al.(2012) [2] et al(2012) using Python and the entire algorithm is divided into 3 parts:

- Horizontal Partitioning
- Vertical Partitioning
- Refining

The first algorithm, Horizontal Partitioning is a simple Heuristic which produces clusters of the original dataset by splitting the data by grouping them into clusters that contain the same or a similar record. The second algorithm, Vertical Partitioning is the part where the actual Disassociation of the data takes place and the clusters that the Horizontal Partitioning produced will be Disassociated based on the  $K^m$  Anonymity logic. Although the algorithm has 3 parts, I will only be implementing the first two parts of the algorithm which will Dissociate the data into sub clusters.

#### 3.1 Horizontal Partitioning:

This is the first algorithm and is based on a simple Heuristic and will split the data into different clusters based on their similarity. The algorithm takes in two inputs, the Dataset itself and a list called *Ignore* that will store the most common element in each iteration and will initially be empty. The output of this algorithm will be a set of clusters that will be grouped together based on their similarity.

This Algorithm Iteratively Divides the Dataset into multiple clusters based on the similarity of the elements in the dataset. The algorithm first checks if the length of the dataset is lesser than the parameter *maxClusterSize* which is a parameter used to define the size of every cluster that the parameter outputs. If the size of the dataset is lesser than the *maxClusterSize* then the algorithm does not process the dataset further as the original dataset itself can be considered as a cluster and be VerticallyPartitioned.

**Algorithm: HORPART****Input** : Dataset  $D$ , set of terms *ignore* (initially empty)**Output** : A HORIZONTAL PARTITIONING of  $D$ **Param.** : The maximum cluster size *maxClusterSize*

```

1 if  $|D| < \text{maxClusterSize}$  then return  $\{D\}$ ;
2 Let  $T$  be the set of terms of  $D$ ;
3 Find the most frequent term  $a$  in  $T - \text{ignore}$ ;
4  $D_1 =$  all records of  $D$  having term  $a$ ;
5  $D_2 = D - D_1$ ;
6 return HORPART( $D_1, \text{ignore} \cup a$ )  $\cup$  HORPART( $D_2, \text{ignore}$ )

```

**Figure 4 : Horizontal Partitioning Pseudo Code**

If the length of the Dataset is greater than the *maxClusterSize* parameter, then the algorithm in the first iteration computes the most frequent term ‘a’ in the entire dataset and adds this term ‘a’ to the ignore list which ensures that the term will not be used in further splitting of the dataset. Once the Ignore list is updated the algorithm splits the dataset into two parts  $D_1$  and  $D_2$ .  $D_1$  contains all the records of the dataset that have the most frequent term ‘a’ in it and  $D_2$  contains the remaining records of the dataset that does not have the most frequent term ‘a’ in it. In the next iteration the most frequent term in the partition  $D_1$  is calculated, let’s call this term ‘b’ and ‘b’ is appended to the ignore list and  $D_1$  is further split into two subsets, one subset having the term ‘b’ and the other subset that does not have the term ‘b’. This operation is iteratively applied till a dataset which is smaller than the *maxClusterSize* is produced. The subsets of the operation that will not have the most frequent term of that splitting operation are all accumulated and the same splitting operation is applied to that accumulated set of records.

Once the entire dataset is divided into clusters that are smaller than the *maxClusterSize* parameter, the algorithm will output a set of clusters that will be used by the Vertical Partitioning algorithm which will further process these clusters using the  $K^m$  anonymity technique. One of the cases that the authors Terrovitis et al.(2012) [2] et al.() does not take into consideration in his Horizontal Partitioning Algorithm is when one of the Partitions is smaller than the  $k$  value that’s used in Vertical Partitioning. This is a problem, because when a partition is smaller than the  $k$  value, then that particular cluster will become infeasible as there’s no way that particular partition can form a  $K^m$  anonymous partition as there are no  $K$  items in the cluster. My solution to overcome extreme cases like this is to collect all the clusters that have elements smaller than the  $K$  value in them and apply the Horizontal Partition algorithm to that collection separately. If even this operation results in clusters with only 1 element then divide the sample of rows from the dataset to form clusters that are smaller than the *maxClusterSize* parameter this way makes sure that the rows are not lost and retains utility.

### 3.2 Vertical Partitioning

After the Horizontal Partitioning has produced all the clusters, the Vertical Partitioning Algorithm will take these clusters as input and Disassociate them individually.

**Output** : A  $k^m$ -anonymous VERTICAL PARTITIONING of  $P$

```

1 Let  $T^P$  be the set of terms of  $P$ ;
2 for every term  $t \in T^P$  do
3    $\lfloor$  Compute the number of appearances  $s(t)$ ;
4 Sort  $T^P$  with decreasing  $s(t)$ ;
5 Move all terms with  $s(t) < k$  into  $T_T$ ; //  $T_T$  is finalized
6  $i = 0$ ;
7  $T_{\text{remain}} = T^P - T_T$ ; //  $T_{\text{remain}}$  has the ordering of  $T^P$ 
8 while  $T_{\text{remain}} \neq \emptyset$  do
9    $T_{\text{cur}} = \emptyset$ ;
10  for every term  $t \in T_{\text{remain}}$  do
11    Create a chunk  $C_{\text{test}}$  by projecting to  $T_{\text{cur}} \cup \{t\}$ ;
12    if  $C_{\text{test}}$  is  $k^m$ -anonymous then  $T_{\text{cur}} = T_{\text{cur}} \cup \{t\}$ ;
13   $i++$ ;
14   $T_i = T_{\text{cur}}$ ;
15   $T_{\text{remain}} = T_{\text{remain}} - T_{\text{cur}}$ ;
16 Create record chunks  $C_1, \dots, C_v$  by projecting to  $T_1, \dots, T_v$ ;
17 Create term chunk  $C_T$  using  $T_T$ ;
18 return  $C_1, \dots, C_v, C_T$ 

```

### Figure 5 : Vertical Partitioning Pseudo Code

**Figure 5** illustrates the Pseudo Code for the Vertical Partitioning Algorithm and although this algorithm may seem complicated, it is actually quite straightforward to Implement. The only section where this algorithm becomes complex is when trying to generate all the term combinations to check if they are  $K^m$  Anonymous.

The main idea behind this algorithm is that the algorithm preserves the term combinations that appear frequently but dissociates the terms that do not appear frequently as these terms stand out from the other records and can be used to identify the record owner.

This algorithm takes  $k$  and  $m$  values along with the Clusters P as inputs and processes each cluster produced by the Horizontal Partitioning algorithm separately. The output of this algorithm is two kinds of chunks: Record Chunk which is a  $K^m$  anonymous subset of the cluster and the Term Chunk which contains the remaining terms in the cluster. Initially the term frequency also called as  $\text{support}(s(t))$  for every term in the cluster is calculated and sorted in descending order of their frequency. The terms that have support less than  $k$  value are moved to the Term Chunk  $T_T$  because they appear occasionally and cannot be formed into a  $K^m$  Anonymous set. This operation corresponds to the lines 1-5 in **Figure 5**.

After the Term Chunk  $T_T$  for the cluster is finalised the algorithm then proceeds to process another term  $T_{remain}$  which is nothing but the terms that are not assigned to the Term Chunk  $T_T$  is also sorted in the descending order of their frequency. All the terms in  $T_{remain}$  will be processed iteratively with the use of a While loop (Lines 8-16 in the PseudoCode) and two terms  $T_{cur}$  (T-current) and  $C_{test}$  (test Chunk) will be initialised.  $T_{curr}$  will initially be empty during the first Iteration of the loop and the term  $C_{test}$  will be used to test all the combinations of the term in  $T_{remain}$  and hence will be initialised during every Iteration of the loop as long as the  $T_{remains}$  is not empty.

For every term 't' in  $T_{\text{remain}}$  the test chunk  $C_{\text{test}}$  will be initialised with the union or the addition of the terms  $T_{\text{curr}}$  and term 't' ( $T_{\text{curr}} \cup t$ ). If the term  $C_{\text{test}}$  turns out to be  $K^m$  Anonymous then the current term  $T_{\text{curr}}$  will take the value of  $C_{\text{test}}$  i.e,  $T_{\text{curr}}$  will take the value of  $T_{\text{curr}} + 't' (T_{\text{curr}} \cup t)$  for that particular iteration. To check if  $C_{\text{test}}$  is  $K^m$  Anonymous, the algorithm will need to compute all the possible combinations for the terms in  $T_{\text{remain}}$  up to the value of m. So if the value of m is 2, then the algorithm will have to compute all the combinations of the term  $T_{\text{remain}}$  containing two terms. After creating all the combinations and checking if they are  $K^m$  anonymous, the term  $T_i$  will take the value of  $T_{\text{curr}}$  and then the loop increments by one and repeats the entire process which will return the following terms  $T_1, T_2, T_3, \dots, T_v$  which are all  $K^m$  Anonymous.

		Record chunks		Term chunk
		$C_1$	$C_2$	$C_T$
Cluster $P_1$ $ P_1  = 5$	$r_1$	{itunes, flu, madonna}		ikea, viagra, ruby
	$r_2$	{madonna, flu}	{audi a4, sony tv}	
	$r_3$	{itunes, madonna}	{audi a4, sony tv}	
	$r_4$	{itunes, flu}		
	$r_5$	{itunes, flu, madonna}	{audi a4, sony tv}	
		Record chunk		Term chunk
		$C_1$		$C_T$
Cluster $P_2$ $ P_2  = 5$	$r_6$	{madonna, digital camera}		panic disorder, playboy, ikea, ruby
	$r_7$	{iphone sdk, madonna}		
	$r_8$	{iphone sdk, digital camera, madonna}		
	$r_9$	{iphone sdk, digital camera}		
	$r_{10}$	{iphone sdk, digital camera, madonna}		

(b) Anonymized dataset  $D^A$

Figure 6 : Dissociated Output

### 3.3 Refining:

This is the final step of the Anonymisation process and it does not involve dissociating the data anymore, the Dissociating is already complete and this step mainly focuses on improving the the quality of the anonymised data while still maintaining the anonymised property. This step focuses the term chunk output of the anonymised data. Consider the **Figure 6**, from the term chunk we can see that the terms Ikea and Ruby appear in both the clusters and although the support  $s(t)$  for these two terms are low in their respective clusters, when both the clusters are combined, their support becomes high enough to threaten user privacy and this lowers the quality of the output. To overcome this Terrovitis et al.(2012) [2] et al.(2012) suggested *joint clusters* which allowed different clusters to share the TermChunks where terms appear multiple times in two or more consecutive terms. This method ensures higher flexibility and quality by Iteratively constructing more *joint clusters* until no more increase in quality can be achieved.

Terrovitis et al.(2012) [2] et al.(2012) considered implementing the Refining process by calculating the Information loss for all the possible steps and choosing the step that results in the maximum increase in quality, in other words the step that produces the least loss and has the highest increase in data quality. But proceeding to Implement the Refining step this way would be very inefficient as there are a lot of possible scenarios that could be possible and calculating loss for every single step is very cumbersome. Because this is very inefficient, Terrovitis et al.(2012) [2] et al.(2012) defined a criterion where the join cluster would only be formed if the support of the terms in the term chunk is greater than the support of the terms in their individual clusters. But even this criterion was very computationally exhausting to process, so he came up with a simple heuristic described in **Figure 7** which can be used in the Refining process.

**Algorithm:** REFINES

**Input** : A set  $\mathcal{P}$  of  $k^m$ -anonymous clusters

**Output** : A refinement of  $\mathcal{P}$

```
1 repeat
2   Add to every joint cluster a virtual term chunk as the union of the
   term chunks of its simple clusters;
3   Order (joint) clusters in  $\mathcal{P}$  according to the contents of their
   (virtual) term chunks;
4   Modify  $\mathcal{P}$  by joining adjacent pairs of clusters (simple or joint)
   based on Equation 1;
5 until there are no modifications in  $\mathcal{P}$ ;
6 return  $\mathcal{P}$ 
```

**Figure 7 : Pseudo Code for Refining**

## 4. Implementation:

The aim of the Implementation is to demonstrate the performance and verify the results of the Horizontal Partitioning and Vertical Partitioning Algorithms, this is achieved by Implementing the algorithm using Python running on Google Colab. Three real world Datasets and 1 constructed dataset were used to implement the algorithms and the datasets *WV1* and *WV2* contained clickstream data from an Electronic Retailer and the *POS* dataset contained Transaction log from another E-Commerce website and the constructed dataset was made by me to verify the results. **Figure 8** illustrates the number of rows of the dataset (  $|D|$  ), number of unique terms of the dataset (  $|T|$  ) and the maximum record size and the average record size of each of the datasets respectively.

Dataset	$ D $	$ T $	max rec. size	avg rec. size
<i>POS</i>	515,597	1,657	164	6.5
<i>WV1</i>	59,602	497	267	2.5
<i>WV2</i>	77,512	3,340	161	5.0

**Figure 8: Test Data Characteristics**

Before we could start implementing the algorithms, a little data pre-processing had to be carried out for the *WV1* and the *WV2* datasets, as the format of these datasets are not conventional. The data in these two datasets were separated by ‘-1’ instead of the conventional comma and the end of every row of data had a ‘-2’ instead of the more traditional fullstop. To convert the file into a more conventional Comma separated value(CSV) file, a simple find and replace method was carried out in a text editor. All the ‘-1’ values were replaced by ‘,’ and all the ‘-2’ entries were removed. This find and replace operation converted the text files to a csv file which was then later uploaded to Google Colab and the column name was changed to ‘Records’ to be further processed.



## 4.1 Horizontal Partitioning Implementation:

This is the one of the two algorithms that will be implemented and is based on a simple heuristic which splits the dataset into two parts. Initially the most common term 'a' in the entire dataset is calculated and then all the rows in the dataset that contain the term 'a' will form the first split of the dataset and all the rows that do not contain the term 'a' will form the second split of the dataset. If the first split or the first subset is bigger than the *maxClusterSize* then the first split is again divided by finding out the most common term in that respective subset. This operation is Iteratively repeated till the clusters that emerge are smaller than the *maxClusterSize*. As this process is an Iterative process, I wrote a function shown in **Figure 9** that calculates the most common term and returns a list that contains the most common terms along with its frequency that can be used in splitting. This function first converts the 'Record' column of the dataset into a list of strings and then cleans the list by removing all the characters that are not Alphanumeric using Python's Regular Expressions. After the terms are cleaned it uses Python's 'Counter' function from the Collections library to count the occurrence of each element in that list. Although the 'Counter' Function is time consuming, it provides accurate results and a similar methodology can be used for Vertical Partitioning.

```
[ ] def most_common(Dataframe):  
    Queries = (str(list(Dataframe['Records'])))  
    PureQueries = re.sub('[^A-Za-z0-9]+', ' ', Queries)  
    frequency = Counter(PureQueries.split(' ')).most_common()  
    return frequency
```

**Figure 9: Function to calculate the most common term**

After finding the most common term in the dataset, the algorithm will have to iteratively split the dataset to get clusters that are smaller than the *maxClusterSize*. To split the dataset, the algorithm will use the Pandas function '.contains' which will return all the rows of the dataset that contain the most common term 'a'. **Figure 10** depicts the first split of the dataset, where WV1D1 is the subset of the *WV1* dataset that contain all the rows of the dataset that contain the term 'a' and WV1D2 is the subset of the *WV1* dataset that does not contain the term 'a'. After the dataset is split into two parts, the WV1Ignore list gets updated with the term 'a'.

```
▶ WV1Ignore = []  
  
WV1D1 = WV1[WV1['Records'].str.contains(most_common(WV1[0][0]))]  
WV1D2 = WV1[~WV1['Records'].str.contains(most_common(WV1[0][0]))]  
BMS1Ignore.append(most_common(BMS1)[0][0])
```

**Figure 10 : Splitting the Dataset**

If the partition WV1D1 is larger than the *maxClusterSize* parameter, then the algorithm will again partition WV1D1 into two parts by finding the most common terms in WV1D1, this process keeps repeating iteratively until the partition WV1D1 is smaller than the *maxClusterSize* and for every Iteration the WV1Ignore list gets updated with the term used to partition for that particular loop. Once WV1D1 is smaller than the *maxClusterSize* that particular instance of

WV1D1 will get put into a list containing other clusters and will be removed from the WV1 dataset, so that the entire dataset can be partitioned again.

But not every row of the dataset can be assigned to WV1D1, the rows of the dataset that do not contain the most common element will be assigned to WV1D2, and WV1D2 will also be partitioned in a similar fashion to how WV1D1 was partitioned i.e, WV1D2 will also be partitioned again into two parts where one part will contain the most common term in WV1D2 and the other part will not contain the most common term of WV1D2, this process too will be iteratively repeated till the size of the WV1D2 instance is smaller than the *maxClusterSize* and the list WV1Ignore will again be updated with the term that will be used for partitioning in every Iteration. Once the WV1D2 instance is smaller than the *maxClusterSize* this particular instance will be removed from the original dataset WV1 which will then be partitioned again.

But since WV1D2 originally will not have the most common terms of the dataset because of how the algorithm is designed, the further iterations of WV1D2 will have terms that appear very rarely in that particular Instance, this will lead to clusters being smaller than the K value which implies that they won't be able to form  $K^m$  anonymous chunks in the Vertical Partitioning because they will not have enough support  $s(t)$ . My solution to this problem is to separate all the terms that have support smaller than the K value and divide them equally to form clusters that are greater than the K value but smaller than the *maxClusterSize*.

After splitting the dataset into clusters that are smaller than the *maxClusterSize* but not smaller than the K value, all the clusters will then be used as input to the Vertical Partitioning Algorithm which will Dissociate every cluster Individually based on the logic provided in the Pseudo Code.

## 4.2 Vertical Partitioning Implementation:

After the Horizontal Partitioning function has finished processing each of the datasets it will produce a list of clusters which will then be used as input to the vertical partitioning algorithm along with the terms K and *m*. This algorithm also makes use of the `most_common` function **Figure 9** used during the Horizontal Partitioning to calculate support for all the terms in a cluster. Support is the frequency of every term that appears in the said cluster; this is done to filter out the terms that appear less than the K value, because it is not possible for these terms to form a chunk of the dataset that will be  $K^m$  anonymous.

For every cluster of the dataset, this algorithm first calculates the  $support(s(t))$  of every term in the cluster using the `most_common` function, which uses a Counter from the python's library Collections. This method returns a list with the respective terms and the frequency of that corresponding term.

After calculating the support for every term in the cluster, terms that have their support smaller than the K value are separated by putting them into a chunk called the *termChunk*, once the *termChunk* is set, the terms that are not in the *termChunk* will then be iteratively assigned to another temporary chunk to be used for testing all the possible combinations.

To test all the possible combinations for the remaining terms that are not assigned to *termChunk*, the algorithm uses the *combinations()* function from the *itertools* library that will output all the possible combinations for a given list. The function *combinations(iterable, r)* returns all the possible combinations upto the length 'r'. For this case the 'r' value in the *combinations* function will be the same as 'm' because we will only need to test combinations of terms of upto size 'm' to generate terms that are  $K^m$  Anonymous.



The remaining terms that are not in the *termChunk* will be iteratively assigned to the test chunk and the test chunk will use the *combinations()* function to check for all the possible combinations. Because the terms are being added iteratively, there is no need to produce combinations for single terms as there is only one possible combination that can be produced for a single term. This method saves computation power and will be quicker for execution.

After generating all the combinations, the algorithm will check if there are at least  $K$  other combinations with the same terms, once the algorithm has found the  $K$  other combinations that contain the same terms, it then creates the *recordChunk* containing all the terms that are  $K^m$  Anonymous.

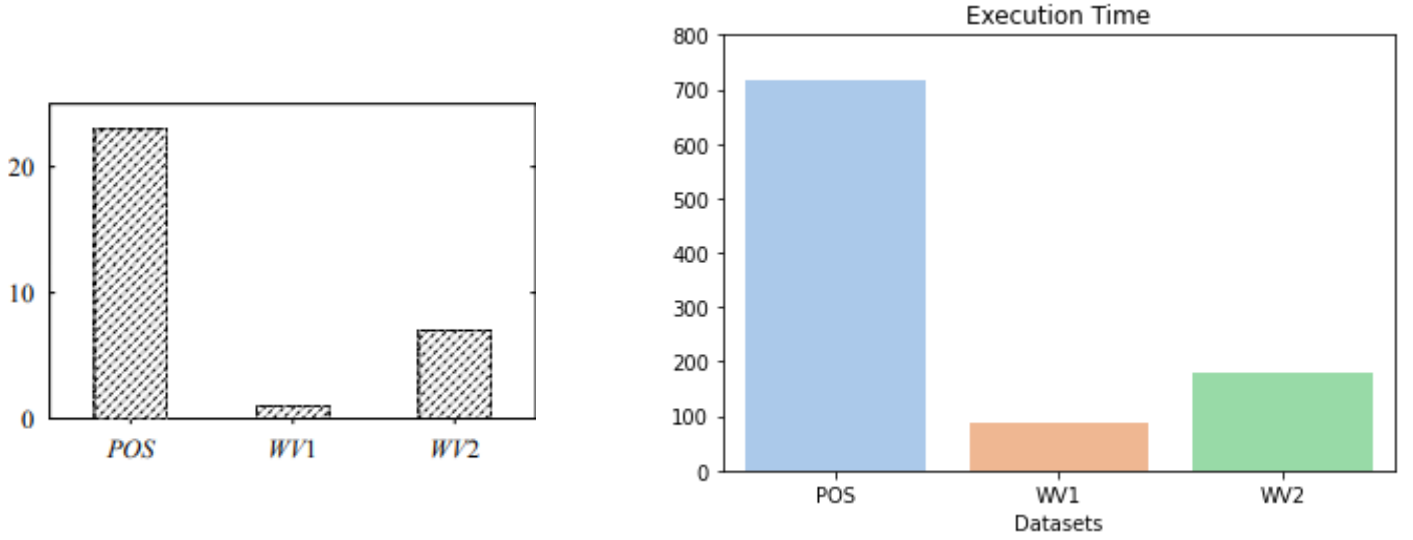
Once the *recordChunk* and the *termChunk* is finalised, it then forms a Disassociated cluster with the  $K^m$  anonymous *recordChunk* and the *termChunk* that will be common to all the rows of the *recordChunk*. This ensures that an attacker who has background knowledge of upto 'm' terms will only be able to distinguish between 'k' records.

## 5. Results:

To measure the performance of the algorithms implemented, we used the WV1, WV2 and the POS dataset and the Processing time, Top-K Deviation and the Relative Error was measured by varying the parameters  $m$  and the *maxClusterSize*. The  $m$  value is set to 2 because anything greater than 2 will result in a very long execution of the Vertical Partitioning algorithm as it takes a very long time to generate every combination of length 3 for every cluster.

### 5.1 Processing Time:

To compare my results with the results achieved in the paper by Terrovitis et al.(2012) [2] et al.(2012), I ran the algorithm with the same parameters and the same number of times as the paper i.e, the values of the parameters as  $k=5$ ,  $m=2$  and the *maxClusterSize* value was set to 11 and ran the program 5 times. With the algorithm running under the same conditions as the paper, my algorithm took much longer to process the datasets when compared to the paper. **Figure** Illustrates the results achieved by our algorithm on the right when compared to the results achieved by Terrovitis et al.(2012) [2] et al.(2012) on the left, I expected this result because the the code in the paper is written in C++ which is a compiled language and executes programs much quicker when compared to python which my program was written in and my program uses the Counter function which is significantly slower to execute when compared to the other methods, but I chose this method as it returned the most accurate results. And the size of the dataset was also directly proportional to the execution time i.e, the larger dataset POS took longer to execute than the relatively smaller datasets WV1 and WV2.



**Figure 11 : Execution Time Results**

And if you look at the second graph in **Figure 11**, we can see that the execution time between the datasets WV1 and WV2 is also different, this is because of the WV2 dataset has about six times more Unique terms in the dataset when compared to the WV1 dataset which implies that the algorithm will take longer to find and sort the most common term in the WV2 dataset when compared to the WV1 dataset. This change in the execution time is also reflected in the results achieved by Terrovitis et al.(2012) [2] et al.(2012).

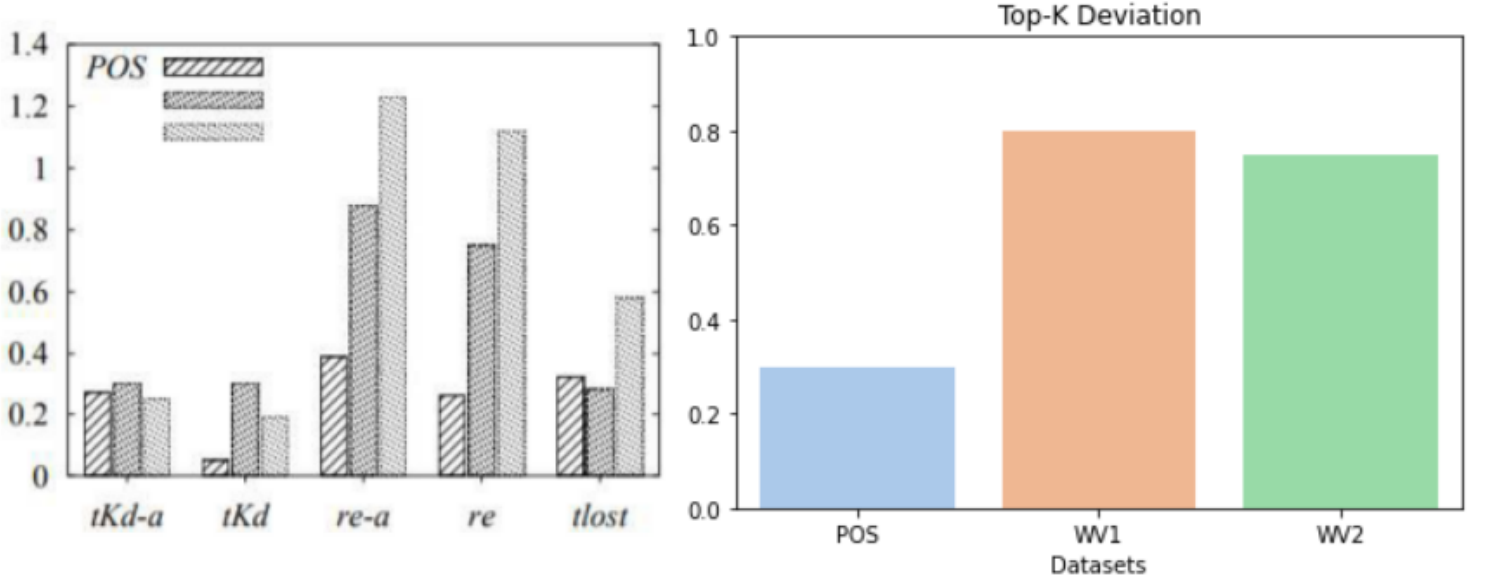
## 5.2 Top-K Deviation:

This metric measures how the top-k frequent items in the original dataset differs from the reconstructed dataset. The formula to measure this dataset is shown in Figure 12 where  $FI$  and  $FI'$  are the top-k frequent terms in the original and the reconstructed dataset respectively. This metric is a ratio of the top-k terms that appear in the reconstructed dataset and also in the original dataset and is used to measure how well the algorithm preserves the terms from the original dataset to the reconstructed dataset.

$$tKd = 1 - \frac{|FI \cap FI'|}{|FI|}$$

**Figure 12 : Top-K Deviation**

For our case, to find the Top-K deviation, I used an Apriori function that extracts frequent itemsets in a dataset from the python library mlexend. One issue during testing this metric is that the function took a long time to process large datasets like the ones used in this project. The tKd values was relatively higher for the WV1 and the WV2 dataset when compared to the POS dataset and this is because the POS dataset is large compared to the other two and the average length of terms in the POS dataset is also higher than the other two datasets. This implies that there were many record chunks created for the POS dataset. **Figure 13** indicates the tKd values for the WV1, WV2 and the POS dataset along with the results obtained by Terrovitis et al.(2012) [2] et al.(2012)



**Figure 13 : Top-K Deviation Result**

### 5.3 Relative Error:

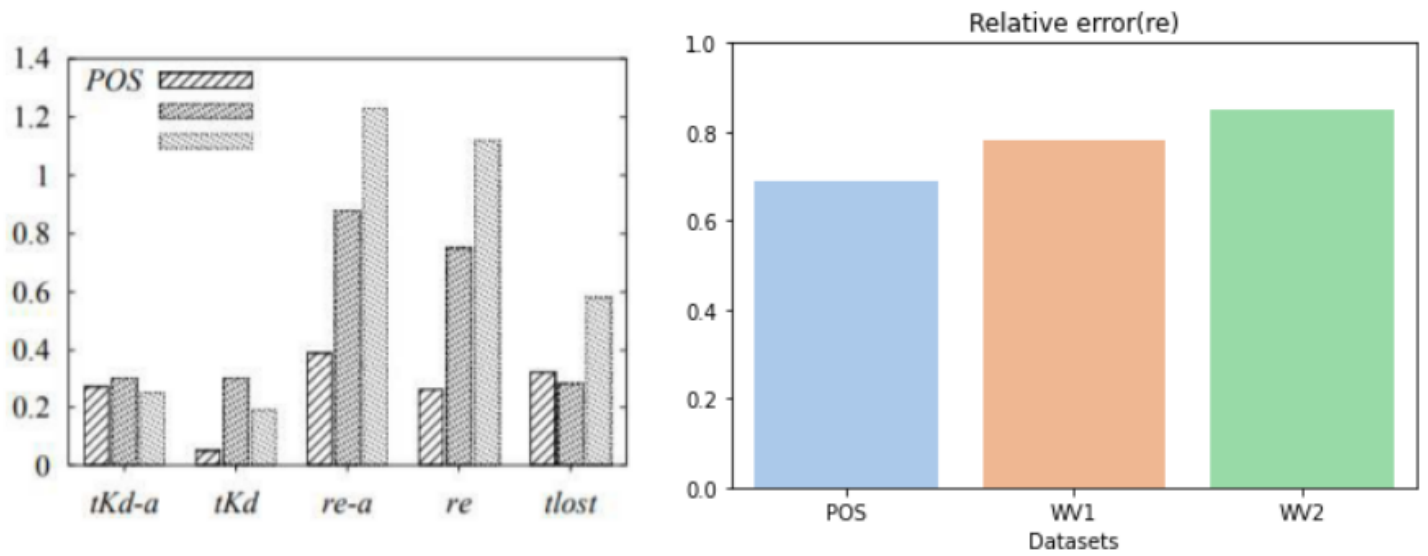
Relative error is the measure of the error between the support of the term combinations in the reconstructed dataset to the support of the term combinations in the original dataset. This metric is used to measure how effective the algorithm is at reconstructing the dataset and is defined as shown in Figure 14.

$$re = \frac{|s_o(a, b) - s_p(a, b)|}{AVG(s_o(a, b), s_p(a, b))},$$

**Figure 14 : Relative Error**

The term  $s_o(a, b)$  is the support of the terms  $(a, b)$  in the original dataset and  $s_p(a, b)$  is the support of the terms  $(a, b)$  in the reconstructed dataset. The denominator of the formula uses both  $s_o(a, b)$  and  $s_p(a, b)$  but averages both the terms' value because if we consider only one of the two terms in the denominator, then there will be many cases where the metric will raise an error because terms that might appear in the original dataset might not appear in the reconstructed dataset and terms that appear in the reconstructed dataset might not appear in the original dataset. So calculating the Average of both the terms ensures that there is no instance of division by 0 anywhere.

To measure this metric accurately, the support of the terms are ordered in descending order and a small set of sequential terms are used to return the relative error. This is done to make sure that the result is not skewed as it does for large datasets with multiple domains. **Figure 15** depicts the relative error measured for the 3 datasets used in my testing and from this figure we can see that the relative error achieved by me for the POS dataset is much larger than the error that was achieved by Terrovitis et al.(2012) [2] et al(2012). but the relative error is relatively small when compared to the relative error of all the three datasets I tested.



**Figure 15 : Relative Error Result**

## 6 Future Work:

### 6.1 Refining:

This paper does not implement the Refining stage of the Dissociation which leads to skewed results. Therefore the current result is different from the results that was obtained from the paper where the algorithm was sourced from. I would like to finish implementing the Refining algorithm to see if my results are equivalent to the results achieved by Terrovitis et al.(2012) [2] et al.

### 6.2 Multiple K-value results:

The paper only measured results with the K value set as 5 and I would like to measure all the metrics again by varying the K value within a range just like how Terrovitis et al.(2012) [2] et al did it.

### 6.3 Missing Metrics:

There are several metrics like the Multi Level Mining loss that are missing from the paper that I would like to measure in order to see how effective my algorithm was at dissociating datasets by comparing my results to the results from the paper.

## 7 Conclusion:

The current research was aimed at implementing algorithms that would anonymise datasets suggested by Terrovitis et al.(2012) [2] et al. which are used in Privacy Preserving Data Publishing(PPDP). The main goal was to implement the Horizontal Partitioning and the Vertical Partitioning algorithms which make up two of the three algorithms suggested. Both the algorithms were successfully implemented and tested using real world datasets and the anonymity for the respective datasets were achieved successfully. However, because only two of the three algorithms were implemented and also because the method of implementation was different, there was variation in the expected results and the achieved results.

This research has shown that data could be published while maintaining the privacy of the individuals in the data, but there are a few methods that can still infer information and further research has to be carried out to ensure that the privacy of the people in published data will be preserved to the best possible degree.

## 8 References:

- [1] Zhou, B., Pei, J. and Luk, W., 2008. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM Sigkdd Explorations Newsletter*, 10(2), pp.12-22.
- [2] Terrovitis, M., Liagouris, J., Mamoulis, N. and Skiadopoulos, S., 2012. Privacy preservation by disassociation. *arXiv preprint arXiv:1207.0135*
- [3] Abbasi, J., 2020. Younger adults caught in COVID-19 crosshairs as demographics shift. *JAMA*, 324(21), pp.2141-2143.
- [4] Odlyzko, A., 2003, July. The unsolvable privacy problem and its implications for security technologies. In *Australasian Conference on Information Security and Privacy* (pp. 51-54). Springer, Berlin, Heidelberg.
- [5] Fung, B.C., Wang, K., Chen, R. and Yu, P.S., 2010. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (Csur)*, 42(4), pp.1-53.
- [6] Machanavajjhala, A., Kifer, D., Gehrke, J. and Venkitasubramaniam, M., 2007. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1), pp.3-es.
- [7] Samarati, P. and Sweeney, L., 1998. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression.
- [8] Aluthwala T. 2020. K-Anonymity Privacy Preservation in Data Mining.  
<https://thamindur.medium.com/k-anonymity-privacy-preservation-in-data-mining-8d5b5ad19d45>