Final Report

Implementation of a Data Privacy Protection Tool for Relational Data

---

Author: Benjamin G Lourence

Student No: 1111753

Supervisor: Dr. J Shao

Moderator: Dr. D Tsaneva

Module: CM3203

Date: 30/04/2015

Contents

List of Figures 4
Abstract 6
Acknowledgements 7
1 Introduction 8
1.1 Project Aims & Expectations 9
1.2 Project Audience/Beneficiaries 9
2 Background 10
2.1 Anonymisation Attribute Terminology 10
2.2 Individual Re-Identification 11
2.3 Re-identification Case Studies 12
2.4 k-Anonymity 13
3 Approach 15
3.1 Data Specifications 15
3.2 Overarching Design Concept 16
3.3 Anonymisation Structure Design 17
3.4 Result Measurements 19
4 Algorithm Design 20
4.1 Set Based Anonymisation 20
4.2 Range Based Anonymisation 21
4.3 Hierarchy Based Anonymisation 22
4.4 Set vs. Hierarchy Based Anonymisation 25
4.5 Set Based k-members Clustering 26
4.6 Hierarchy Scan and Local Replace Anonymisation 28
5 Implementation 30
5.1 Technologies Used 30
5.2 Model View ViewModel (MVVM) Architecture 31
5.3 Anonymisation Algorithms Implementation Architecture 32
5.4 Project Structure 34
5.6 Pre-Processing Tools 38
5.7 Anonymisation Algorithms Implementation 44
5.8 Post-Processing Tools 50
5.9 Testing 56
6 Results & Evaluation 59

6.1 Test Cases 59
6.3 Critical Evaluation of Results 65
7 Future Work 66
7.1 Plugin Algorithm Repository 66
7.2 Import/Export Custom Hierarchies 66
7.3 ILoss Calculation Extensions 67
7.4 Graphing of information retention metrics 68
8 Conclusions 69
9 Reflection on Learning 71
9.1 Project Management 71
9.2 Understanding of Personal Privacy Issues 72
9.3 Plugin Algorithm Architecture 73
10 Appendices 75
11 References 76

List of Figures

Figure 2.4.1 – Non-conforming vs k-conforming table (k=3) 13
Figure 2.4.2 – Suppression example: k-anonymity conformant table (k=3) 13
Figure 2.4.3 – Generalisation example: k-anonymity conformant table (k=3) 13
Figure 3.1.1 – Comma-Separated Value Format Example 15
Figure 3.2.1 – Overarching System Design Concept 16
Figure 3.3.1 – Class Diagram: Anonymisation Structure Design Concept 17
Figure 4.1.1 - Pre Set Based Anonymisation, Age Dataset 20
Figure 4.1.2 - Post Set Based Anonymisation, Age Dataset 21
Figure 4.3.1 – Pre Hierarchy Based Anonymisation, Animal Dataset 22
Figure 4.3.2 – Animal Hierarchy 22
Figure 4.3.3 – Post Hierarchy Based Anonymisation, Animal Dataset 22
Figure 4.3.4 – Pre Hierarchy Based Anonymisation, Rule Defined Postcode Dataset 23
Figure 4.3.5 – Rule Generated Anonymisation Hierarchy Extract 24
Figure 4.3.6 – Post Hierarchy Based Anonymisation, Rule Defined Postcode Dataset 24
Figure 4.6.1 – Pre Anonymisation Example 28
Figure 4.6.2 – Post Local Vs. Alternative Technique Anonymisation Example 28
Figure 5.2.1 – Graphical Representation of MVVM Architectural Pattern 31
Figure 5.3.1 – Anonymisation Algorithm Implementation Structure 32
Figure 5.4.1 – Project Structure, Visual Studio Screenshot 34
Figure 5.5.1 – Data Import: Attribute Declaration, System Screenshot 36
Figure 5.5.2 – Data Import: Successfully Loaded csv Data, System Screenshot 37
Figure 5.6.1 – Column Pre-Processing Tools, System Screenshot 38
Figure 5.6.2 - Postcode Hierarchy Construction, Rule-Based String Redaction, System Screenshot .. 39
Figure 5.6.3 – Custom Hierarchy Definition, System Screenshot 40
Figure 5.6.4 –Tree View Display Code Snippet, HierarchyTreeView.xaml, Visual Studio 42
Figure 5.7.1 – C# Generics Example, Visual Studio 44
Figure 5.7.2 – Set Based Anonymisation C# Implementation, Visual Studio 45
Figure 5.7.3 - k-members Set Based Generic Implementation, Visual Studio 46
Figure 5.7.4 – HierarchyBasedAnonymisation.cs C# Implementation, Visual Studio 47
Figure 5.7.5 – Generate Next Set of Values to be Anonymised From Hierarchy Data structure 48
Figure 5.7.6 – Dynamic Loading of Plugin Algorithms + Default Set and Hierarchy Based Anonymisations: PreprocessingColumnsVm.cs, Visual Studio 49
Figure 5.7.7 – Post Processing Results Dashboard, System Screenshot 50
Figure 5.7.9 – Task Based Evaluation Queries + Results, System Screenshot 53

Figure 5.9.1 – Unit Tests Results Explorer, Visual Studio 56
Figure 5.9.2 – Levenshtein Distance Unit Tests, Visual Studio 57
Figure 6.1.2 – Test Case 1: Output Anonymised Data Set (k=2 vs. k=3), System Screenshot 59
Figure 6.1.4 – Test Case 2: Output Anonymised Data Set (k=2 vs. k=3 vs. k=4), System Screenshot .. 60
Figure 6.1.6 – Test Case 3: Output Anonymised Data Set, System Screenshot 61
Figure 6.1.7 - Test case 3: Post-Processing, Task Based Metric Evaluation, System Screenshot 62
Figure 6.1.8 – Test Case 4: Post-Processing, Data Based Evaluation, System Screenshot 63
Figure 6.1.9 – Test Case 5: Increased Dataset Size, System Screenshot 64
Figure 7.3.1 - Postcode Anonymisation Hierarchy Possible Descendants 67

Abstract

The motivation behind this project is to help ensure the protection and privacy of individuals. An increasing number of organisations are beginning to utilise personal data for benefit. Many of these uses are pioneering and the current trend seems to indicate the demand for personal data analytics will only increase. This demand has exacerbated issues surrounding personal privacy, in particularly the assurance that an individual cannot be re-identified from such data released into the public domain. In order to provide sufficient protection for data subjects this project features k-anonymisation algorithms that protect against re-identification attacks.

This project has successfully implemented an anonymisation tool for relational data. The tool facilitates a number of various techniques that can be configured to provide custom anonymisations. The tool also supports post-processing metric generation that quantifies information retained in anonymised datasets.

The implemented tool provides an excellent base for future development in the field and could even be integrated as a component of a larger information security system. Relational anonymised data generated by the system satisfies the k-anonymity privacy component of the project and indeed would provide sufficient protection against re-identification. While no ground breaking new algorithms have been implemented within the application, the innovate way that anonymisation algorithms can be loaded and used by the system may influence similar tools in the future.

Acknowledgements

For his knowledgeable advice and helpful guidance throughout this project, special thanks must be given to my supervisor Dr. J Shao.

1 Introduction

As modern society continues to collect, store and analyse ever-increasing amounts of information, the associated value of data increases as more information is collated. This data can be used to predict trends, perform association discovery and various other data mining activities. Organisations leverage this knowledge in a wide variety of ways for example increasing efficiency or generating increased sales of targeted products. The popular rise of big data analytics has accelerated the demand for information that will impact the management and inform decisions of an organisation.

Recently the number of organisations openly releasing data to the public has dramatically increased, the underlying reasons behind this range from regulatory compliance to inviting third parties to perform specialist data mining analysis. However with these new modern trends that organisations seem to be embracing, there is a worrying and very real issue. Personal privacy has become increasingly rare within the social based and information consuming orientated web. Users willingly volunteer vast amounts of personal information in order to access 'free' services, organisations then use this knowledge to create targeted adverts specifically for each user. There are many social issues here but by participating in use of these products users exchange their personal information to gain access to services.

While users may freely volunteer data placed on social media other more sensitive information such as medical records should be protected before being passed to external parties. Therefore steps must be taken to ensure that data of this nature is suitably anonymised to ensure individuals cannot be re-identified.

At the core of this information revolution is the underlying personal data. While the increased openness and frequency of information releases to the public provide data scientists with a wealth of data to work with, it has exacerbated issues surrounding individual privacy. This project aims to design and develop a tool that will enable users to sufficiently anonymise data for privacy protection while still maintaining the usefulness of original data. The tool will leverage k-anonymisation algorithms in order to sufficiently protect individual data to a suitable level. Anonymisation is achieved by ensuring that any record in the data set is identical to at least k-1 other records (where k is some nominal value) therefore no underlying individual can be re-identified.

1.1 Project Aims & Expectations

This project aims to:
Develop a relational data anonymisation tool based on existing k-anonymisation algorithms. The system should operate on tabular data, similar in format to common relational database storage. The tool should utilise a developed k-anonymisation algorithm in order to enforce k-conformity within relational data. The system is expected to export anonymised data in a format that preserves anonymity.

The application aims to provide a series of pre-processing tools that will allow users specify anonymisation execution details such as custom anonymisation hierarchies. The tool should provide functionality to configure custom anonymisations. One vital component expected to be developed is an anonymisation hierarchy builder.

Develop post-processing tools to quantify anonymisation performance. Performance metric extraction will allows users to assess the successfulness of k-anonymisation algorithms. Users are expected to make significant use of this functionality when inspecting their anonymised relational datasets for information loss.

The tool will aim be to built as an extensible solution for future development. The solution is expected to be built as a series of modular components in order to maximise maintainability. This is vital as the project maybe included as a component of future data anonymisation projects within the schools of Computer Science and Informatics at Cardiff University.

Anonymisation algorithms are expected to be implemented using plugin architecture. Third party developers could then build new anonymisations and plug them into the platform while still being able to leverage the existing system infrastructure.

Cope with a wide range of anonymisations and data types. This means the user will be able to load relational data and invoke anonymisations after specifying a few simple criteria and minimal configuration steps.

1.2 Project Audience/Beneficiaries

As stated in the initial text there are a series of organisations releasing data into the public domain for a multitude of reasons. The resultant tool is aimed to benefit these entities. As an emerging field in the technology industry a wide array of organisations may have significant interest in harnessing such a tool when anonymising and releasing personal data. Keeping this in mind the resultant solution should be a generic implementation and not targeted too specifically at one sector of the industry, this approach will yield the largest potential audience for the tool.
In addition to the organisations that want to safely anonymise and release personal data, the tool should also be orientated towards an audience of future developers. The school of Computer Science and Informatics at Cardiff University plan to research and develop several projects associated with privacy preserving data publishing including simulating attacks on such anonymised datasets. It has been suggested the tool produced by this project could be integrated as a component of a larger systems in the future. Therefore the design and implementation stages should take extreme caution to make the resultant solution as extensible and reusable as possible.

2 Background

The emerging trend for organisations to release and leverage data has led to the creation of new careers such as data scientists. Providing analysts with the optimum quality and quantity of data has led to intense research in a wide number of fields. The field of Privacy Persevering Data Publishing (PPDP) has been the focus of intense interest recently. PPDP is required to ensure that personal data can mined and analysed for useful previously undiscovered knowledge while protecting individual's identity.

The NHS releases patient data for researchers to investigate underlying trends while also keeping regulatory compliance. Netflix the video streaming service released data about its customer's movie preferences as part of an open competition to improve its video suggestion algorithm in return for a cash prize. The rise of open data repositories such as Data.gov.uk[1] brings huge amounts of personal based information that can be easily accessed by the general public for whatever purpose they wish. These examples demonstrate the increased availability and novel use of personal data that is used to drive changes in the modern world. As openly available personal information becomes more a prevalent component of organisation operations it is pertinent to consider the impact on individual privacy. A person should not be susceptible to re-identification from an openly released dataset. For example no single person should be exposed as having a certain condition from a set of medical patient data that is openly released by a hospital. However the data must retain enough information so that it is still useful for analysis by data scientists and other professionals who will inspect the data.

2.1 Anonymisation Attribute Terminology

Explicit Identifiers: Information that explicitly identifies an individual e.g. National insurance number.

Quasi-Identifiers: An attribute that could be used in conjunction with other quasi-identifies to potentially re-identify an individual e.g. Combination of postcode and date of birth.

Sensitive Attributes: Data that relates to a specific individual within the record set e.g. Salary.

Non-Sensitive Attributes*:* Encompasses any attribute that does not fit the definition of the three terms above.

2.2 Individual Re-Identification

Latanya Sweeny reports “87% (216 million of 248 million) of the population in the United States had reported characteristics that likely made them unique based only on {5-digit ZIP, gender, date of birth}.” [2]

The statistic uncovered by the report indicates that only a partial sub-set of information about an individual is required to achieve re-identification. Datasets publicly released by organisations often have some level of privacy protection. Although the level of protection provided is usually far from adequate to assume that individuals could not be re-discovered. A theoretical example of this is could be a dataset where explicit identifiers (e.g Name, NHS number) have been removed. While providing an extremely basic level of protection, it is clearly not sufficient based on the findings of Latanya Sweeny who theorised that individuals can be identified if only a small number of key quasi-attributes remain, these may not appear to disseminate a large amount of information but could be leveraged for individual discovery. This type of pseudo-anonymised data is a major cause for concern because the organisation releasing the data and individuals featured may incorrectly feel they are protected when in actuality they have not been adequately secured and are susceptible to malicious re-identification attacks.

Leveraging attributes that can be collated together in order to identify an individual is know as a record linage attack. Record linkage attacks can be extremely easy to perform given only a minute amount of information about targeted victims.

“In the attack of record linkage, some value qid on QID [Quasi-identifier] identifies a small number of Records in the released table T, called a group. If the victim’s QID matches the value qid, the victim is vulnerable to being linked to the small number of records in the group.” [3]

As B.C. M. Fung describes how a record linkage attack utilises groups of quasi-identifiers to extract a small finite number of records from a dataset. Highly targeted attacks or relatively small datasets could even result in a single record being re-identified. Even from a minimal group of retrieved records, further investigation can easily lead to individual discovery from pseudo-anonymised data.

The real danger of this type of record linkage attack is that the quasi-identifying attributes may not appear to reveal a great deal of information about the underlying person. However when combined together these seemingly innocent pieces of information actually form a major security risk. Another point to carefully consider is the sheer number of quasi-attributes an organisation may employ in addition to the combination they are arranged within the data set. The tool produced by this project should aim to sufficiently anonymise values from a wide range of use cases in such a way that personal privacy is protected while retaining as much information as possible.

2.3 Re-identification Case Studies

William Weld is one of the highest profile victims of a record linkage attack. The former governor of Massachusetts medical records were re-identified although supposedly anonymised and released by an insurance group. The record linkage attack was performed using a combination of ZIP code, birth date and gender. A corresponding voter registration dataset revealed that these quasi-identifying values where unique to the former governor. Cross referencing this information allowed researchers to successfully link and extract the openly released medical records of William Weld. Although there have been sceptics such D. C. Barth-Jones whom investigated the feasibility of replicating this type of attack "careful re-examination of the population demographics in Cambridge indicates that Weld was most likely re-identifiable only because he was a public figure who experienced a highly publicized hospitalization rather than there being any certainty underlying his re- identification using the Cambridge voter data, which had missing data for a large proportion of the population" [4]. However this particular case has brought PPDP to the forefront of Computer Science research and highlighted the possible dangers that lay ahead.

As previously discussed the video streaming service Netflix launched a competition in 2006 to improve its video suggestion algorithm based on user movie preferences. Researchers would attempt to improve on the current algorithm and the most successful development would receive a cash prize of 1 million dollars. To support researchers the organisation publically released a set of supposedly anonymised user preference data. However two researchers from the University of Texas were able to re-identify individuals from the dataset by cross-referencing the released data with movie ratings on the popular site IMDB (Internet Movie Database). The researchers were able to achieve this re-identification with unprecedented ease. This resulted in a number of lawsuits against the movie streaming company and once again highlighted to the industry the dangers of releasing individual data publically.

2.4 k-Anonymity

Personal protection against re-identification is the fundamental concept of this project. One protection mechanism k-anonymity has been used to ensure personal privacy when publishing data. Essentially anonymity is gained by ensuring that every entry in the table has nominal value k occurrences. Since one cannot be anonymous in a crowd of one, the idea is to introduce ambiguity within the dataset to stop individual re-identification. Therefore all explicit identifiers should be removed and care must be taken to obfuscate quasi-identifiers that could be used to infer the identity of an individual.

Age
19
14
17

Age
18
18
18

Figure 2.4.1 – Non-conforming vs k-conforming table (k=3)

Fig 2.4.1 depicts two separate datasets. Given the value of k as 3, any value featured in a dataset must occur at least 3 times to be k-conformant. The left hand table does meet these criteria where as the right hand dataset is said to be k-conformant. Below two techniques have been detailed that could be applied to the left hand table in order to achieve k-anonymity.

Age
1*
1*
1*

Figure 2.4.2 – Suppression example: k-anonymity conformant table (k=3)

Suppression: Entails removing distinguishing information from the dataset, for example replacing digits of the attribute 'Age' with an asterisk '*'. This process obviously incurs a certain amount of information loss because we have lost the exact values represented by the underlying data however we can infer from Fig 2.4.2 the values must lay between 10 and 19.

Age
14-19
14-19
14-19

Figure 2.4.3 – Generalisation example: k-anonymity conformant table (k=3)

Generalisation: Requires specific values to be transformed to a boarder category that includes the original information, for example transforming age specific values in Fig 2.3.1 into a range 14-19. Similar to the suppression technique a certain amount of information loss is incurred because the original value has been obfuscated. However in this particular example more data utility is retained because the list of possible values is 14 to 19 in Fig 2.4.3, instead of the larger range in Fig 2.4.2, even though both techniques make the dataset k-anonymity compliant.

A research paper written by Ji-WonByun explains the complexity of the problem. “Although the idea of k-anonymity is conceptually straightforward, the computational complexity of finding an optimal solution for the k-anonymity problem has been shown to be NP-hard, even when one considers only cell suppression.” [5]

The paper goes on to describe the issues surrounding k-anonymisation within a dataset. NP hard (Non-deterministic polynomial time hard) essentially means there are no polynomial-time algorithms to solve such a problem. Greedy heuristic algorithms have been shown to solve such problems to an acceptable degree. Section 4; algorithm design details some of these approaches which will be used in the anonymisation tool.

This project has two fundamental concepts; to ensure the personal privacy of individuals featured within anonymised datasets and to retain the maximum amount of data utility where anonymisation is required. Both concepts are of paramount importance to create a tool that will produce results of usable quality for researchers/professionals who will probe the resultant data for underlying trends. Throughout the design and implementation stages of this project these concepts will be regarded as the highest priorities.

3 Approach

3.1 Data Specifications

Initially the system will leverage comma-separated values (csv) files to import data. These csv files are simple, easily human readable and commonly used throughout the technology sector. While there is no industry standard format there are a set of commonly understood and adhered to guidelines. Below is a typical extract of csv data, the tool will import data from files with the following syntax, some csv files use semicolons or other special characters to delimit records. However the initial implementation will only use commas.

```
Name, Age, Postcode, Condition
Dave, 21,CF24 4AR, Cancer
James, 45, BH21 1HT, HIV
Hannah, 8, GH32 3JU, Flu
Fahed, 24, MK7 7SY, Flu
```

Figure 3.1.1 – Comma-Separated Value Format Example

Future implementations of the tool could focus on developing mechanisms to import data from a relational database management system (RDBMS). Integration with industry standard database applications would make the tool far more appealing to users and would be seen as a significant benefit streamlining the import process. However due to time constraints the initial implementation of the tool will not support this feature.

Similar to the input data mechanisms initially the tool will export the anonymised data as a csv file. This file format has been selected because it is easy to implement and commonly used throughout the industry, tools in the same field as this project seem to support csv as a standard output. After investigating other output potential formats such a JavaScript Object Notation (JSON) and Extensible Mark-up Language (XML) I saw no clear advantages that would significantly affect the functionality of the tool. Therefore for the initial implementation I have decided to focus on the operation of algorithms and post-processing activities.

3.2 Overarching Design Concept

The overarching design concept separates the tool into 3 clear distinct sections.

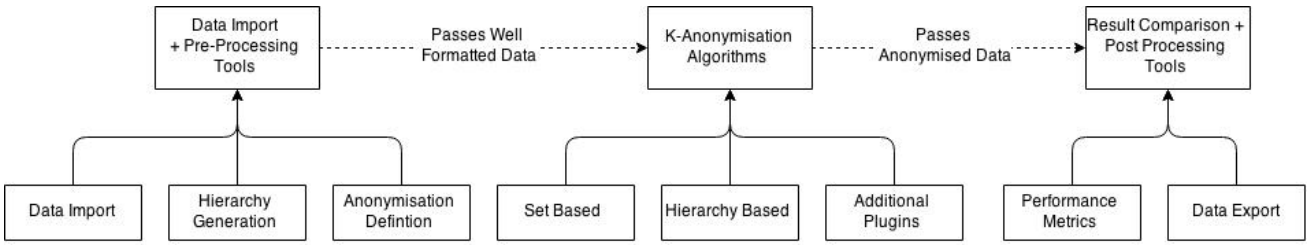


Figure 3.2.1 – Overarching System Design Concept

Data Import/Export + Pre-Processing Tools:
Throughout the design and development stages of the project a set of 'well prepared' test data will be assumed to ensure integrity and testability of results. The system will import data from a csv file. Fig 3.2.1 also indicates that pre-processing tasks includes hierarchy generation and anonymisation definition (i.e setting an attribute as a quasi identifier with k = 3). This distinct section is responsible for the preparing the data ready for the anonymisation process.

Anonymisation:
The fundamental operation of the system is to anonymise data. One of the project concepts is to create a platform that will allow developers to implement new algorithms and while leveraging the existing system infrastructure. The tool will utilise techniques to enforce k-anonymity within a dataset. Initially the k-members algorithm will be developed to provide the necessary functionality. Anatomy has been suggested as a suitable secondary k-anonymisation algorithm to be developed if time constraints allow. In the future other developers should be able to design and code their own anonymisation algorithms that will then feed the next comparison stage of the system. The anonymisation process can be regarded as a black box; raw data is passed in, some form of anonymisation is applied, k-anonymity conformant data is passed out.

Result Comparison + Post-Processing Tools:
The proposed tool includes functionality for post-processing activities once an anonymisation algorithm has been applied. This section should allow users to review and compare performance of their designed algorithms. Comparison between anonymised data sets is an essential requirement of the tool because it will allow data scientists/developers to test, review and improve anonymisation techniques. The comparison elements of the system should support querying of processed data, for example extracting all the associated age data that has been anonymised by the system. Here a line-by-line comparison against the original data would be extremely useful to develop an in-depth understanding of these anonymisation techniques. These post-processing tools will allow users to quantify the performance of the application and the quality of resultant datasets.

3.3 Anonymisation Structure Design

This section of the report details the design concept of k-anonymisation algorithm structure and how the pluggable architecture will be achieved. For more detail on the underlying processes please see section 4; algorithm design. As depicted in Fig 3.2.1 the anonymisation component of the system sits as a stand-alone component of the wider system. The application will simply call to a common interface and invoke anonymisation process on a particular dataset.

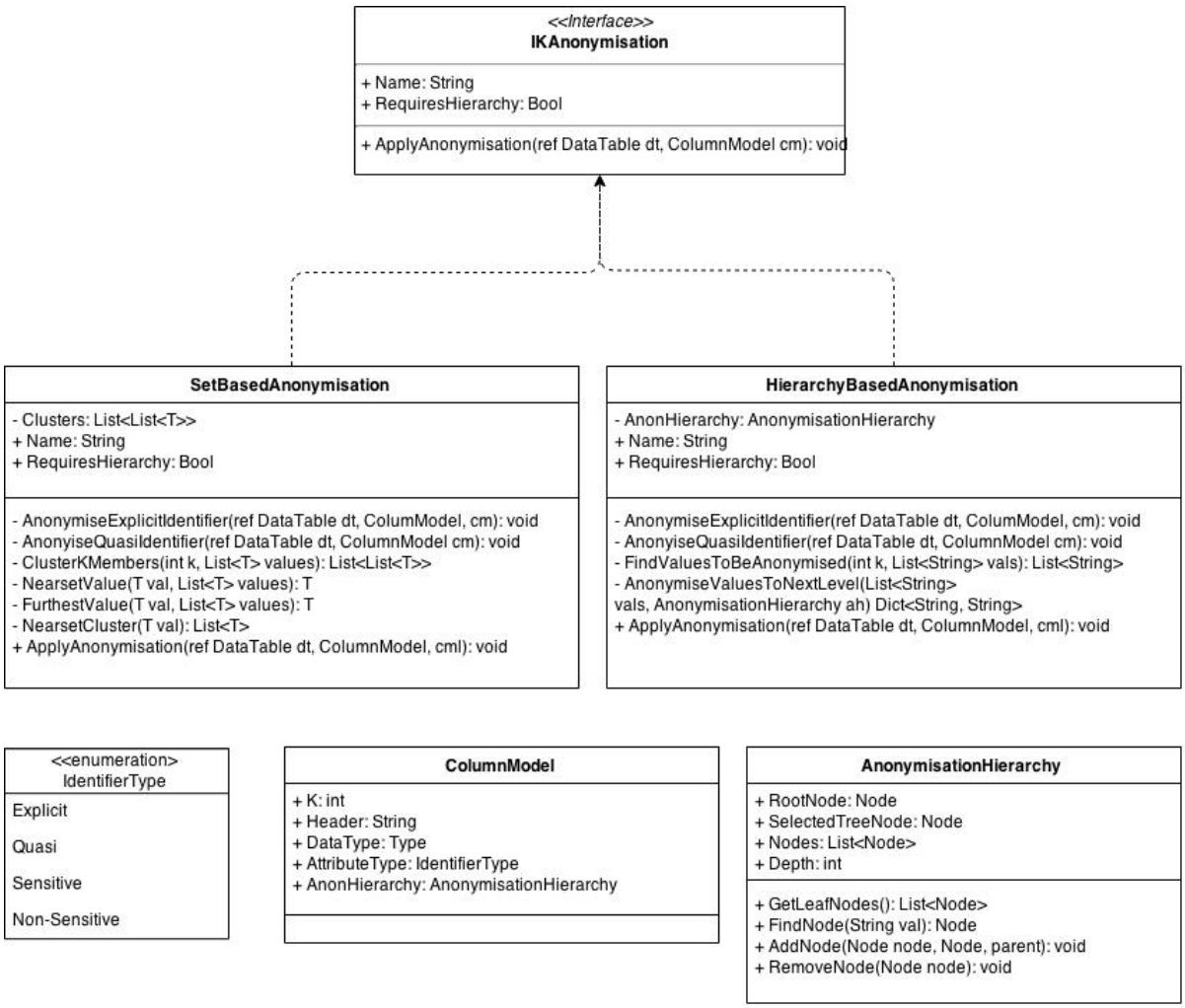


Figure 3.3.1 – Class Diagram: Anonymisation Structure Design Concept

The class diagram (Fig 3.3.1) depicts the common interface IKAnonymisation that all algorithms must implement to be compatible with the system. The underlying implementation of each algorithm is very different (information is detailed within section 4 Algorithm Design) but the common interface means that each algorithm can be implemented in vastly different manners while still being invoked in a common way. The required hierarchy Boolean variable indicates to the system if the algorithm will need an anonymisation hierarchy to operate. The anonymisation hierarchy tree is passed into the application as a component of the column model class.

For each attribute in a dataset there will be a corresponding column model class that retains information required for the anonymisation process. Each individual k level is set independently here this means that a user could opt to set different k-levels of anonymity for various attributes in the same anonymisation operation. This ensures the system will be flexible and highly configurable to the user needs. Column model classes hold an identifier type enumeration for each column that indicates whether the selected attribute is explicit, quasi, sensitive or non-sensitive. Once these properties have been defined it will be up to the discretion of an anonymisation algorithm to perform corresponding operations.

The anonymisation hierarchy class featured is used for any type of hierarchy-based anonymisation. Essentially the class contains a tree like data structure where each original unique value in the dataset forms a leaf node, as anonymisation is applied each value to be anonymised is replaced by it's parent element in the tree until the dataset becomes k-conformant. This is why the hierarchy based anonymisation class features an 'AnonymiseValuesToNextLevel' method. The anonymisation hierarchy class also exposes a number of methods for adding, removing and interacting with the data structure. These methods are then used freely by algorithms within the system and will be made available to external plugins as well.

3.4 Result Measurements

Once a dataset has been successfully anonymised by the system, the system will provide a suite of tools that will enable users to extract relevant metrics in order to judge the performance. There are two differing aspects that could be measured.
The first, information retention describes the utility of anonymised versus original data. Given a postcode BH21 1SY that is anonymised to BH21 1** in order to form a k-compliant table, a portion of the original underlying data has been truncated. Extracting data metrics for information retention would quantify the usability of the newly anonymised value.

Benjamin Fung states, “A data metric measures the data quality in the entire anonymous table with respect to the data quality in the original table.” [6]

One overarching goal of the tool is to retain the maximum amount of data utility, researchers/professionals examining the resultant data will form better solutions given the most information and an in depth understanding of the data context. A medical researcher investigating the spread of diseases would be able to form far superior conclusions given high data utility as appose to working with a dataset where large amounts of the original information had been changed to a generic disease name in order to become k-compliant. The platform will enable users to apply various anonymisation algorithms, each of these will generate different levels of information retention based on their implementation, therefore the system must provide mechanisms to measure these important data metrics which helps to quantify algorithm performance.

Another technique to provide data metrics is finding the number of occurrences of a value from both the original and anonymised data set. Where the anonymisation process may have altered certain values occurrences in the anonymised set maybe more frequent, such an increase would indicate a loss in precision of the underlying data values.

There are a number of data metric calculations that can be performed to judge information loss based on a hierarchy of anonymisation values. These types of calculations take into account a selected value with respect to any possible original values from a tree structure that forms the incremental anonymisation levels.

The tool will include functionality to extract data metrics from the two techniques detailed above. The implementation stage of the report will detail exactly how this will be achieved.

The other aspect of result measurement concerns privacy perseveration. Often researches will not deal with privacy measurement because if k-level anonymity has been satisfied than it is assumed that privacy is adequate. Obviously the nominal value k can be selected as any integer in accordance with user preference, this opens the discussion that privacy perseveration if only as strong as specified. Without imagining and simulating every attack that could be performed on the anonymised data it is hard to quantify how much privacy protection has been implemented.
Due to the time constraints I will only be initially implementing a series of data metrics to evaluate information retention. However future iterations of this project should consider adding some metrics to quantify privacy perseveration levels.

4 Algorithm Design

Throughout the design and creation of this project I will attempt to implement two types of anonymisation algorithms. Both will attempt to anonymise data with vastly different approaches however each algorithm will be aiming to attain a pre-defined level of k-anonymity in the resultant data set.

In order to ensure k-anonymity within a dataset, values within attributes must be counted to determine their number of occurrences. Both algorithms will implement a set of functionality to calculate and collate values in order to establish what data does not meet the threshold k value and therefore must be anonymised. After generating a list of values to be anonymised, each algorithm will then perform actions to collate values into groups, the resultant groups are then analysed to see if they satisfy the required k-anonymity threshold. This process is repeated until the dataset has attainted the specified level of k-anonymity.

The two different approaches discussed are Set Based anonymisation and Hierarchy Based Anonymisation.

4.1 Set Based Anonymisation

Arguably the simplest approach to group items that have not satisfied k-anonymity threshold values is to preform a union of items. Grouping the resultant items means that you can join the number of occurrences together to form a group that satisfies the k threshold. The table below (Fig 1.1) details a list of 6 ages. Given a specified level of k=3. The value 17 has 3 occurrences and would satisfy the criteria; therefore these values can remain unchanged in the table. However the remaining values in the table will not meet the threshold, each with only 1 occurrence.

Age
17
17
17
41
67
23

Figure 4.1.1 - Pre Set Based Anonymisation, Age Dataset

K Anonymity Threshold = 3
17: 3 occurrences
Values to be anonymised:
23: 1 occurrence
41: 1 occurrence
67: 1 occurrence

Aggregating the values to be anonymised into a single set means that the number of occurrences can be added together. Then replacing each occurrence of a value contained within the set with the resultant set means the level of k-anonymity can be satisfied (Fig 4.1.2).

Age
17
17
17
{23, 41, 67}
{23, 41, 67}
{23, 41, 67}

Figure 4.1.2 - Post Set Based Anonymisation, Age Dataset

K Anonymity Threshold = 3
17: 3 occurrences
Resultant Set - {23, 41, 67}: 3 occurrences

Deciding which items to group together in a set based anonymisation is an important task. Ideally grouping values that are closely related to other values in the set would minimise data loss while still achieving k-anonymity status. However the hardest part is discerning what makes values 'close'. Numeric values 'closeness' can be easily quantified, measuring the difference between separate values. Although this task is inherently more difficult given a category such as patient condition, evaluating and grouping conditions such as Cold, Flu and Fractured Leg require pre-existing knowledge. Anonymising data to form the set {Cold, Flu} would make far more sense to a user evaluating the data than {Cold, Fractured Leg}. While both sets may satisfy the k-anonymity threshold required for the dataset, the initial Cold and Flu set would make far more semantic sense. However when grouping values within categories that require specialist knowledge it would be my recommendation to use hierarchy based anonymisation with a custom defined hierarchy, doing so would result in a far more coherent and sensible output. Hierarchy based anonymisation is discussed below. One significant benefit implementing a Set Based Anonymisation approach is that the user can load the data and clicks go to kick off the anonymisation process. No pre-processing work is required to define sets or format values. This makes the process of anonymising data sets simple, quick and easy to perform from the users perspective. However as previously discussed, Set Base anonymisation is not a 'one-size fits all' approach that can be applied to all types of data.

4.2 Range Based Anonymisation

Another similar method to group data items is to utilise range based anonymisation. Therefore in the example above the resultant set would instead become the range [23-67]. This encompasses the value 41 implicitly within the range. However this resultant range represents a significant amount of information loss in comparison to the set featured above. While the set based anonymisation alludes that the original value is either 23, 41 or 67, 3 possible values. The range based anonymisation is interpreted to mean the original value could lie anywhere between and including 23 to 67, 45 possible values. One of the overarching concepts of this project is to maintain the highest level of data utility possible. Applying a range-based anonymisation in the situation decimates large amounts of information about the underlying data, therefore I have not considered it appropriate to implement within the project.
Another point of interest, ranges can only be calculated and applied to numeric values. Given the category 'Condition' within a medical context the range [Fractured Leg - Flu] makes no sense and would not be an appropriate anonymisation to apply. On the other hand set based anonymisation can be very easily applied to non-numeric values.

4.3 Hierarchy Based Anonymisation

This approach requires some pre-processing criteria to be defined. Obviously in order to apply a hierarchy based anonymisation a hierarchy must first be generated. Unlike set based anonymisations where the values can only be anonymised to sets which are formed by creating a union of values already present in the data, a hierarchy based approach could alter the original value to something completely different. However this process of altering the value is done so that other values in the table could also be anonymised and changed to the new values. The new value that supersedes several items in the original dataset is then measured to see if it has achieved the threshold value for k-anonymity. The process is repeated until the dataset has achieved k-anonymity status.

Animal Type
Dog
Dog
Dog
Snake
Snake
Lizard

Figure 4.3.1 – Pre Hierarchy Based Anonymisation, Animal Dataset

Given k=3 the values Snake and Lizard need to be anonymised in order to satisfy the threshold value. However unlike set based anonymisation which would generate {Snake, Lizard}, this approach requires a defined hierarchy for this example Fig2.2 depicts the generated hierarchy.

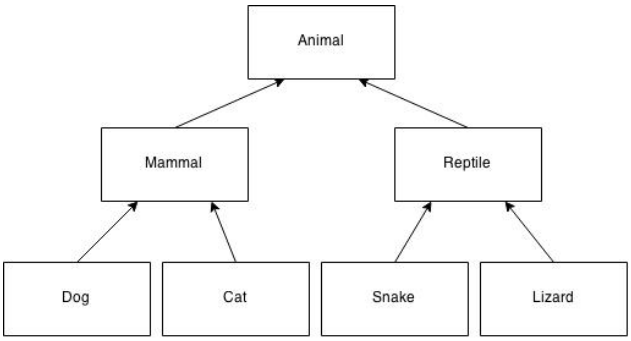


Figure 4.3.2 – Animal Hierarchy

Animal Type
Dog
Dog
Dog
Reptile
Reptile
Reptile

Figure 4.3.3 – Post Hierarchy Based Anonymisation, Animal Dataset

The resulting anonymisation inspects the nodes Snake and Lizard; replacing these values with the common parent Reptile satisfies the k anonymity threshold. Therefore the occurrences of Snake and Lizard become their parent value Reptile. This approach traverses the hierarchy tree in order to find a value that is common enough to meet the pre-defined k number of occurrences.

Hierarchy based anonymisations offer a far more flexible and configurable way of anonymising data. Unlike a set based approach, hierarchies allow users to generate a custom set of values at various levels within anonymisation hierarchies. The advantage of this is that anonymised values are more likely to retain their semantic meaning, altering Flu -> Virus, retains more information utility about the original data compared to forming a set {Flu, Fractured Leg} from other values in the data set. The approach to anonymising values and their results can be more tightly controlled with a user-generated hierarch. However this requires significantly more effort by the user to create and define a custom hierarchy for each unique value in the tree.

An interesting idea that has previously been suggested would be to use a third party data source to automatically generate hierarchies, such as a medical taxonomy of diseases. Using the data from such as source could facilitate the generation of a hierarchy of diseases that could be utilised by the anonymisation tool. This has the advantage of leveraging a pre-created and maintained data source to build what would be an extremely complex hierarchy with minimal effort required by the user.

Rule Defined Hierarchies – One approach to reduce the effort required to generate hierarchies is to employ a rule that automatically builds a tree of nodes based on a list of values fed to the system. A good example of this is string redaction, where characters of a string are anonymised using a character such an asterisk. The underlying rule could be if the k threshold is not met, replace the furthest right valid character to a * and check to see if k occurrences have been created.

Postcode
MK7 7SU
MK7 7ST
MK7 7UY

Figure 4.3.4 – Pre Hierarchy Based Anonymisation, Rule Defined Postcode Dataset

Given k=3 all the values in Fig 4.3.4 are required to be anonymised. Using the string redaction rule described above a custom hierarchy could be defined. Fig 4.3.5 displays a small extract of the rule-generated hierarchy that has been used to anonymise the dataset

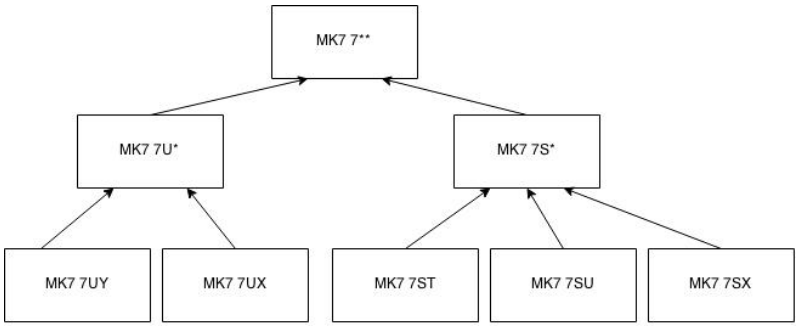


Figure 4.3.5 – Rule Generated Anonymisation Hierarchy Extract

Postcode
MK7 7**
MK7 7**
MK7 7**

Figure 4.3.6 – Post Hierarchy Based Anonymisation, Rule Defined Postcode Dataset

The resultant table displays 3 occurrences of the anonymised Postcode MK7 7** therefore the table satisfies the k=3 level anonymity threshold. Fig 4.3.5 depicts part of a hierarchy that is generated via the string redaction rule. This process means that rule based hierarchies can be automatically generated without the need to build a custom tree every time an anonymisation is performed. This is a convenient cross between the flexibility of hierarchy anonymisation, which retains more semantic meaning, and the ease of use of a set based anonymisation, which requires no pre-processing operation to be performed. Rule based hierarchies can be customised to pad and anonymise strings in different ways left to right or right to left and other options such as changing the character used to anonymise parts of the value.

4.4 Set vs. Hierarchy Based Anonymisation

The advantage to set based anonymisation is that it can be simply run on the input data without much pre-processing setup. Ideally the optimum solution for this entire project is to produce a tool that requires as little user input as possible and encapsulates the complexities of data anonymisation. On the other hand the absence of pre-defined groups means that unrelated data maybe be grouped together in a way that is of no use to the data analysts who want to manipulate and explore the information. Set based grouping may for example collate the conditions {foot pain, headache} to sufficiently meet the k anonymisation criteria. The two conditions may have no commonality and therefore it would be illogical to group the two items together but with set based anonymisation there is not always a clear way to define this.

The other technique that will be implemented in the initial tool is a hierarchy-based anonymisation. However the user will create a hierarchy that will apply a suitable level of anonymisation to the data. This advantage of this approach is that anonymised data will be sensibly grouped together in a way that still maintains utility of the original data. Although this requires the extra step, pre-defining a hierarchy, which can be then applied to the data.

The approaches discussed have both their individual merits and drawbacks. The current preference of many researchers would be to implement a hierarchical approach to data anonymisation. However I am attempting to build a tool that will help users understand the associated processes and differences in techniques, therefore I see fit to include to include both types as a starting point of the platform.

There are many different algorithms that can be implemented to achieve K-Anonymity compliant datasets. I have selected two different algorithms, one to achieve set based anonymisation and the second is to enable hierarchy-based anonymisations.

4.5 Set Based k-members Clustering

This particular approach is an extension of the popular data-mining algorithm k-means clustering with a few alterations. Similar to k-means the initial value is randomly selected from the dataset. One difference is that with k-means the user defines k as the amount of resultant clusters the algorithm should produce, where as in k-members clustering k represents the minimum number of items that should be present in a cluster. As such an obvious difference is that the user does not define how many clusters they want to produce with k-members, instead that is dependant on the data that is fed to the algorithm.

After conducting research I found the following greedy k-members algorithm from a research paper [7]. This pseudocode has been used as the basis for the algorithm I will implement within the data anonymisation tool.

[Greedy k-members co-clustering algorithm]
1. Let S be a set of objects. Choose the anonymity level k and randomly select an object r.
2. Let a cluster index t be 0. Repeat the following process while |S | > k.
(a) Replace r with its furthest object and remove r from S .
(b) t = t + 1. Generate cluster Gt with a single element r.
c) Repeat the following process while |Gt| < k.
i. Find the best neighbour object r of cluster Gt, which has the largest within-cluster aggregation of Eq.(7) after merging.
ii. Add r to cluster Gt and remove r from S .
3. Repeat the following process while |S | > 0.
(a) Randomly select an object r from S .
(b) Find the best neighbour cluster Gt of r, where the object has the largest within-cluster aggregation of Eq.(7) after merging.
(c) Add r to cluster Gt and remove r from S .

During initialisation a list of values is fed to the algorithm, the level of anonymity threshold k is also defined. While the number of values contained in the list is greater than the level of k, the algorithm proceeds to build a set of clusters that contain k number of items (after adding each value to a cluster it is subsequently removed from list of values). Each cluster looks at the remaining items in the list and selects the closest value to be added.

Once the number of list items has been reduced to less than the level of k. The next item is randomly selected from the remaining values. The resulting value is then assessed to find the closest cluster. There are a number of ways you could calculate the closest value. Group average would find the average value of all current values in the cluster, while min/max link look at the nearest/furthest value present in the cluster respectively. Since group average makes use of all the values in the cluster and would average out any outlying values I have opted for this implemenation. Once initial the randomly selected value is added to the closest cluster it is then removed from the list of values. This process is repeated until the initial list of values is empty. An important point to note with this implementation is that once the number of items in the list reaches the level of k, no new clusters are created. Instead the remaining items are appended to pre-existing clusters, which means that clusters may contain more values than the minimum number of k-level items.

K-Members Hierarchical Clustering Issues - One issue that has arisen is the classification of values for the ‘closest cluster’ selection element of the k-members clustering algorithm. This process is fairly

simple for numeric values or structured strings such as postcodes because various measurements can be employed (Euclidean and Levenshtein distances respectively) to find the comparative difference between two values. This calculation would then allow a nearest neighbour to be found. However other attributes may not be so easy to measure, for example in a medical set of records patient condition. Finding a metric to measure the difference between ‘Cancer’ and ‘Flu’ would be necessary in order to ascertain what is the closest cluster. Without defining a custom measurement for every single type of attribute it would be hard to allow the tool to automatically infer what values are ‘near’ to each other, considering near as near in meaning not similarity of the characters of corresponding data. One technique that has been suggested is to utilise the pre-existing hierarchy in order to understand what values are related and therefore devise a set of definitions that specific which values are close to others in terms of meaning.

4.6 Hierarchy Scan and Local Replace Anonymisation

The next algorithm will leverage hierarchies in an effort to maintain data utility. The design and generation of hierarchies will be covered in other aspects of this report. This algorithm possesses two main functions; scan and replace. The scan function similar to the set based implementation will iterate through dataset in order to ascertain and collate a list of values for anonymisation.
Once a value has surpassed the predefined k-level then no further operation will be performed on the item. The second replace element, details the way in which 'values to be anonymised' is transformed. Simply the algorithm looks at the parent node of the value in hierarchy. An example would be Flu -> Virus, if Virus were the parent node of Flu. This process is repeated until the table has satisfied the level of k required.

The root of the hierarchy should be a completely anonymised value which reveals no information such a '*'. Therefore if a value does not meet the k-threshold then it should be fully anonymised as specified and updated within the dataset.

Postcode
MK7 7SU
MK7 7SU
MK7 7SU
MK7 7TY

Figure 4.6.1 – Pre Anonymisation Example

Postcode
MK7 7SU
MK7 7SU
MK7 7SU
*

Postcode
MK7 7**
MK7 7**
MK7 7**
MK7 7**

Figure 4.6.2 – Post Local Vs. Alternative Technique Anonymisation Example

The local anonymisation aspect refers to the fact that if a value meets the threshold it is exempt from any further action. Given a k=3 threshold anonymity level. Fig 4.6.1 displays a set of postcode values to anonymise. Subsequently Fig 4.6.2 depicts two differing techniques used to anonymise these values, local and alternative anonymisation left to right respectively.

Local anonymisation identifies the 3 occurrences of MK7 7SU and because the threshold value of k has been met, no further action will be performed on these values. The remaining value MK7 7TY has a single reaming occurrence and has no opportunity to meet the threshold, therefore it is anonymised to the root of the hierarchy in this case *. The anonymisation is localised to occurrences that have not been discarded by the algorithm because they have surpassed the threshold value. The alternative anonymisation on the other hand would identify the 3 occurrences of MK7 7SU that meet the threshold. The algorithm would then move onto the single occurrence of MK7 7TY and attempt to anonymise this value, the key difference being the alternative scan of the data is performed at this point to find a suitable set of values the occurrence can be anonymised with. In this case the postcode MK7 7SU is found and all the resulting values are converted to MK7 7**. Both techniques produce k-anonymity conformant table. Interestingly preforming a post-processing query on the tables from Fig.4.6.2 searching for possible matches to 'MK7 7SU' would return 4

possibilities from both tables. However the information loss between the two results does differ quite considerably. The local anonymisation maintains maximum data utility for 3 out of the 4 entries in the table. However for the final entry, the original information has been completely decimated as a result of the local anonymisation. Compared to the alternative anonymisation where each entry in the table suffers some degree of information loss. Although no information is fully decimated, as is the case with local anonymisation, the resultant approach does cause a fair amount of information loss across the dataset. I have elected to perform local anonymisation process in the algorithm I have selected because I believe it will maintain greater utility of the original data.

[Hierarchy Scan and Local Replace Anonymisation]
1. Let S be a set of objects. Let H be the anonymisation hierarchy. Choose the anonymity level k.
2. Repeat until all values occur k times or value is anonymised to root of the hierarchy.
a. Scan through s and return a list of values to be anonymised.
b. For each value to be anonymised
i. Anonymise value to parent node in H

The intended end product of this project is a data anonymisation tool that implements both set based k-members clustering in addition to the hierarchy scan and local replace algorithms. Both of these will form the default algorithms coded and built into the system. Should future developers see fit to tweak these default processes, then by adjusting the source code and building a new version of the application the hard coded algorithms can be updated.
However during my initial research and report, I discussed the idea of pluggable algorithm architecture, where a developer makes their anonymisation algorithm available in the system via a plugin. The advantage with this approach is that the system would not have to be rebuilt and instead transforms the final product from a stand-alone tool to a platform that can be used as a test bed to alter and update algorithms. While two default algorithms have been designed and documented within this section of the report, the system architecture means that in reality the tool could feature and leverage many more algorithms.

5 Implementation

5.1 Technologies Used

Implementation Language: C# .NET
C# has been selected as the programming language to implement the data anonymisation tool. The language is powerful, mature and versatile. The language is focused on the object-orientated programming paradigm with c style syntax similar to Java. The language also facilitates access to Microsoft's .NET framework, which itself holds a wealth of useful libraries and other functionality. Language-integrated query (LINQ) allows developers to perform various operations on relational datasets which is an ideal feature given the aims of this project. Last year I undertook an industrial placement where I developed several applications using C# and the .NET framework, this exposure has given me an excellent understanding and strong base to develop the application.

Graphical User Interface: WPF
To create a graphical user interface (GUI) I will be using Windows Presentation Format (WPF) another component of the .NET programming framework. Personally I have just over a years experience in developing and creating interfaces with this set of tools, during which time I found the process to be incredibly simple and intuitive. WPF uses XAML; XML based set of statements to declare and arrange graphical components

Integrated Development Environment:
I have opted to implement the tool using the C#.NET platform. I will develop the system using Visual Studio; Microsoft's accompanying IDE. The software itself is stable and mature; it also features several tools that maybe useful throughout the application implementation such as in built unit testing and code profiling to test speed of execution.

Version Control:
Throughout the implementation process I will be using Git & GitHub to provide version control for the code base of my system. Version control is an important tool used to backup the implementation of a system. Although it also provides other handy features such as visualising changes that have been committed and also the ability to roll back a version if bugs are accidentally introduced to the application.

5.2 Model View ViewModel (MVVM) Architecture

WPF relies on MVVM architectural design pattern in order to separate application logic from GUI implementation. Eventually this means that the layout and information displayed by the user interface can change completely without having to alter the underlying logic code. MVVM is an extension on the popular architectural pattern Model View Controller (MVC), although it has been adapted specially to facilitate data binding which is WPF's technique to populate GUI components with application data. The pattern separates implementation into 3 distinct sections.

Model:
The model section is primarily concerned with retrieving data from data stores as a data access layer (DAL). Data for the anonymisation tool, will be stored within CSV files however in future implementations of the software, the model classes could be extended to retrieve information from a Relational Database Management Systems (RDBMS). The tool will employee this model in order to retrieve and load raw data from CSV files.

View:
While the view refers the graphical implementation presented to the user. Views can be nested or combined to form other views. A view hooks into exposed properties from the ViewModel with a binding expression in the XAML. The underlying business layers and application logic are completely isolated from any GUI implementation. This separation is what gives the MVVM pattern the flexibility required to update the underlying logic or user interface implementations independently.

ViewModel:
The most significant component in comparison to other similar patterns is the ViewModel. The ViewModel provides the bridge from application logic to graphical interface. The ViewModel exposes a set of public properties that can be bound to the view via a binding expression in the xaml. Once again the key point is that the ViewModel has no relational knowledge to the View, therefore it can be independently altered and the View still binds to the resulting ViewModel in the exact same way.

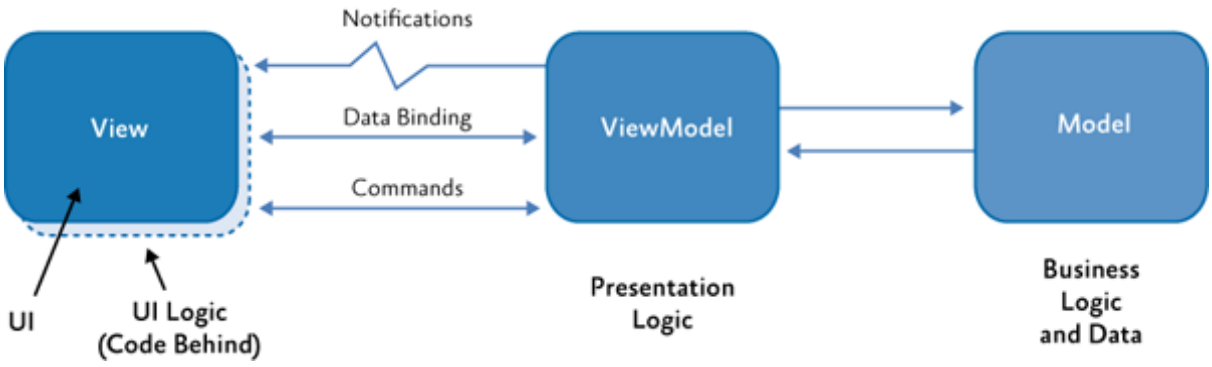


Figure 5.2.1 – Graphical Representation of MVVM Architectural Pattern

The diagram above taken from Microsoft's documentation [8] graphically illustrates the separation of application logic from graphical presentation. The MVVM design approach will allow me to create a modular and very maintainable system. The tool maybe integrated into larger systems so it is key to design a system that is extensible and could be easily incorporated within other applications.

5.3 Anonymisation Algorithms Implementation Architecture

This section is aimed at explaining the implementation of anonymisation algorithms within the tool. Primarily discussing the complexities of implementing plugin algorithm architecture.

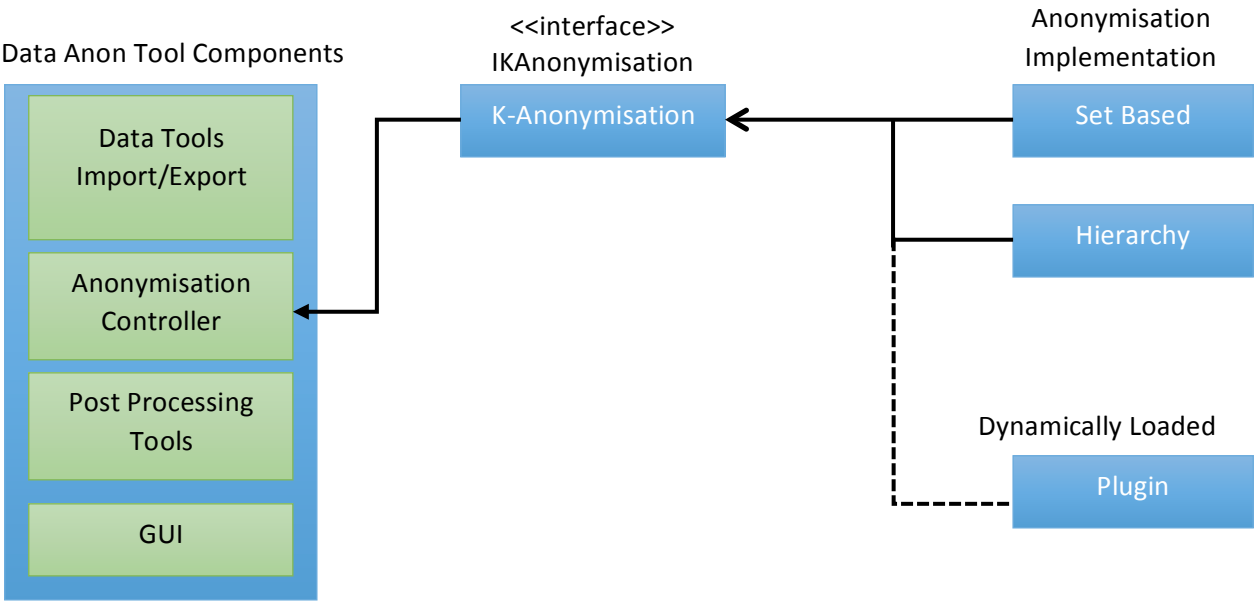


Figure 5.3.1 – Anonymisation Algorithm Implementation Structure

The diagram splits the data anonymisation tool into a set of distinct components, for the purposes of this explanation I will assume that a series of well-formatted data has been loaded into the system and any other necessary pre-processing tasks have been completed.

The anonymisation controller component is responsible for invoking the algorithms that will anonymise data. A dataset is passed into the controller along with another object that contains reference to what attribute is to be anonymised and a reference to an interface; IKAnonymisation. The important point here is that the anonymisation component understands how to interact with the interface IKAnonymisation. Regardless of the underlying implementation the controller simply invokes the algorithm calling a method 'InvokeAnonymisation'. The default set and hierarchy based algorithms both implement the same IKAnonymisation interface. The controller does not care which algorithm is being implemented as long as it can execute the common method, which is guaranteed by implementing the interface. The interface acts as a bridge between anonymisation implementation and the data tool structure.

The IKAnonymisation contains a boolean field 'RequiresHierarchy'. Obviously this indicates as to whether a valid hierarchy should be defined. Once defined a hierarchy is passed to the controller as part of an object that contains other information about attributes being anonymised. From here the algorithm can utilise a hierarchy if required.

One of the requirements for the system is to implement a plugin architecture so that developers can include algorithms without rebuilding the entire system. Each of these algorithms must implement the same IKAnonymisation interface, the same as the default set and hierarchy algorithms. Once again the Anonymisation controller does retain any knowledge about how the process is achieved, that logic is encapsulated within each class that implements the interface.

Plugin algorithms can be built separately from the data anonymisation tool. The output should be built as a dynamic-link library (.dll) file.

"A dynamic-link library (DLL) is a module that contains functions and data that can be used by another module (application or DLL)…. DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily." [9]

Once built an algorithm should be added to a specified plugin directory. Classes that contain the corresponding interface will be dynamically instantiated and appended to a list of algorithms available for use in the system. Should one of these plugin algorithms be selected for anonymisation then the anonymisation controller will execute the method 'InvokeAnonymisation' on the dynamically created object. Regardless of whether the underlying anonymisation technique requires sets, hierarchies or another type of implementation they are all initialised and invoked in exactly the same way. This means that developers can implement their own anonymisation algorithms and plug them into the system, utilising the pre-existing infrastructure and functionality.

5.4 Project Structure

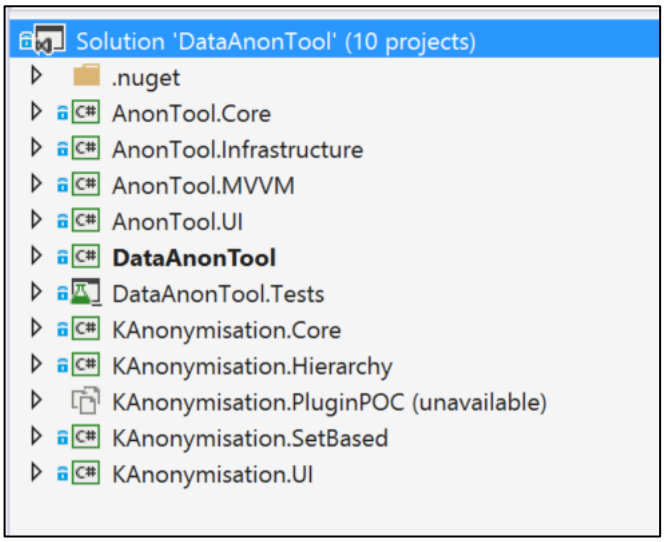


Figure 5.4.1 – Project Structure, Visual Studio Screenshot

The graphic above depicts the project structure of the system. The final product makes use of 10 separate projects within a single solution. I have taken a great deal of care when implementing the anonymisation tool to ensure that various aspects of the system are independent modules, they are combined together to form a maintainable and easily expandable platform. In order to achieve this each individual project has a very targeted function

NuGet [10] is Microsoft's package manager for developers is used within the project to import and makes use of third party software libraries. Specifically a library to handle reading and writing CSV files, this process will be detailed within the section 5.6.

DataAnonTool is the entry point of the application and contains only the shell window of the system. The shell is used as a host for components in other projects. Hosting items from other projects in a shell means that should the required components of the system change the old code can be thrown away and replaced easily by changing what is contained within the shell view. This implementation detail was aimed at designing and implementing an easily maintainable system.

DataAnonTool.Tests is a UnitTesting project. It contains a number of hard coded tests that ensure the functionality of components in the solution. Every time the project is built the set of unit tests are run to ensure that changes to the code have not impacted the functionality of the system. Specifically I have made use of unit tests to ensure the validity of anonymisation techniques applied by the system. Unit and further testing details of the implementation have been detailed in section 5.10 of this report.

AnonTool.Core is one of the most important projects of the system. This library contains a number of classes the core to the functionality of the application. Application logic for menu bars, hierarchy generation and pre-processing operations are all contained within this core library. AnonTool.UI contains the graphical interface implementations of these components.

AnonTool.Infrastructure currently the infrastructure library is responsible for controlling the flow of data into and out of the application. Future developers could extend this project adding functionality to import/export data from a RDBMS. Although this feature has not currently been added to the tool

the system has been designed with extensibility in mind, which is why an infrastructure project has been created.

AnonTool.MVVM is responsible for implementing the controls that enable the application to utilise the Model View ViewModel architectural pattern. One such example is a stub of code "RaisePropertyChanged" that notifies the user interface to update when a publically exposed property is altered.

Projects starting with tag KAnonymisation relate to the implementation of k-anonymity algorithms within the system. KAnonymisation.Core contains the interface IKAnonymisation that anonymisation algorithms must implement, in addition to other key classes. One such is ColumnModel that retains information about what attributes are to be anonymised and holds a reference to the hierarchy that will be applied during the process. Any project that is responsible for the implementation of an anonymisation algorithm makes use of these core common classes. The UI component of KAnonymisation contains a set of views that facilitate post-processing tasks on anonymised data. This project contains strictly graphical components only (the same as AnonTool.UI); the logic to populate these interfaces is situated in the other projects.

There are 3 implementations of anonymisation algorithms in the final system. The default set and hierarchy based anonymisations in addition to KAnonymisation.PluginPOC a proof of concept project that demonstrates the process of injecting a custom plugin anonymisation algorithm. The project is secluded (hence the unavailable warning in Fig 5.4.1) from the rest of the solution to ensure that it functions correctly as a plugin. To further ensure its validity, plugin proof of concept is built independently from the other projects of the solution.

Design and implementation of this solution structure was one of the hardest challenges I faced throughout this project. Careful thought has been given when separating implementation components into isolated projects.

5.5 Data Loader Importing/Exporting

As previously discussed in section 5.4 project structure, DataAnonTool.Infrastructure is currently responsible for importing/exporting data in the application. The tool currently supports both of these operations using csv files.

I have created the a DataLoader class which acts a wrapper class for a third party open sourced library 'Csvhelper' [11]. To import data using the CsvHelper library a file stream is passed into the class 'CsvReader' , from here each line can be processed and broken down into separate attributes within each line. The first line of the csv file is reserved for the list of attribute names (see report section 3.1 data formats).

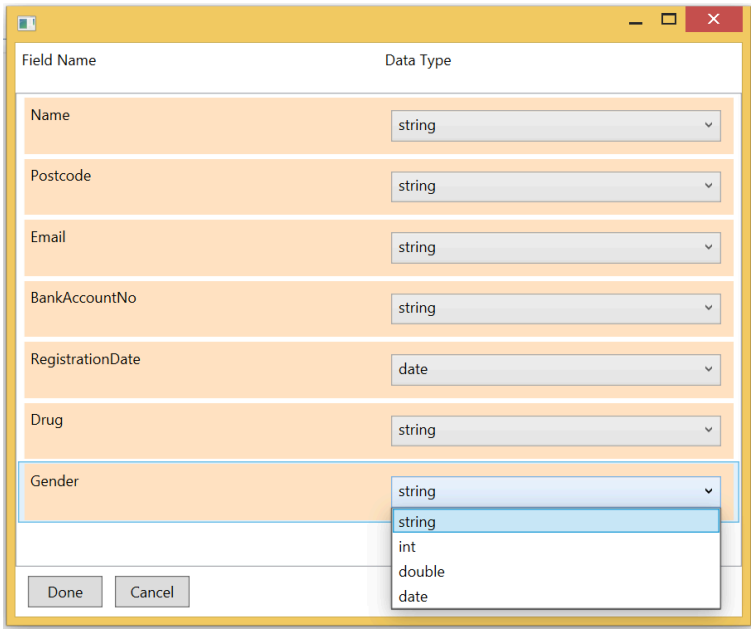
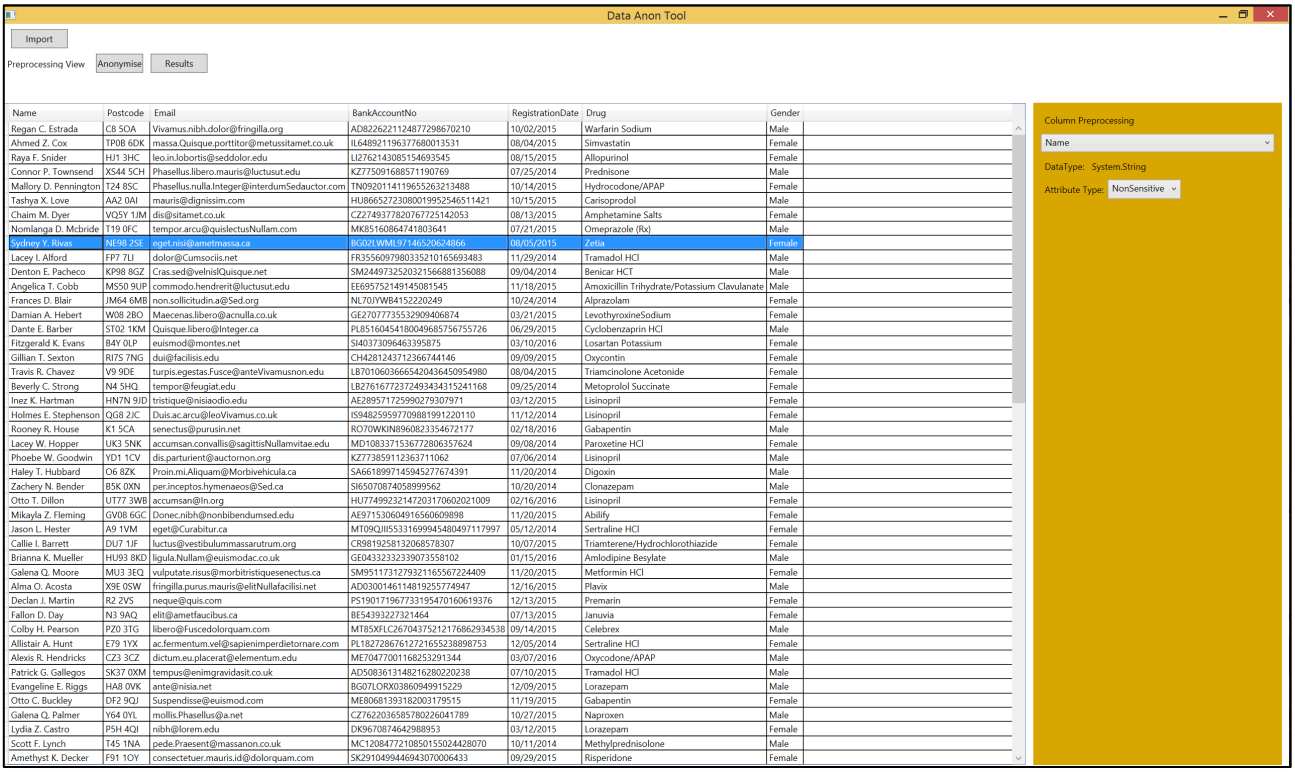


Figure 5.5.1 – Data Import: Attribute Declaration, System Screenshot

Once a list of attribute names has been retrieved from the first line of the csv file, the system presents the window depicted in Fig 5.5.1 This view allows the user to specify the data types of each attribute. It is important to specify correct data types so that the anonymisation algorithms can find semantically 'close' values. Running a set based anonymisation on the attribute age should be done with the assigned data type int, the algorithm can then calculate close integers to form the cluster and restrict information loss. Otherwise age would be defined as a string then algorithm would automatically attempt to match based on characters of strings, two values such as 102 and 10 maybe grouped together because they are fairly similar strings but vastly different values in the context of integer age.



Name	Postcode	Email	BankAccountNo	RegistrationDate	Drug	Gender
Regan C. Estrada	C8 5OA	Vivamus.nibh.dolor@fringilla.org	AD822622112487729867021O	10/02/2015	Warfarin Sodium	Male
Ahmed Z. Cox	TP0B 6DK	massa.Quisque.porttitor@metussitamet.co.uk	IL648921196377680013531	08/04/2015	Simvastatin	Female
Raya F. Snider	HJ1 3HC	leo.in.lobortis@seddolor.edu	LI276214308551469354S	08/15/2015	Allopurinol	Female
Connor P. Townsend	XS44 5CH	Phasellus.libero.mauris@luctusut.edu	KZ7750916885711907G9	07/25/2014	Prednisone	Male
Mallory D. Pennington	T24 8SC	Phasellus.nulla.Integer@interdumSedauctor.com	TN092011411965526321348B	10/14/2015	Hydrocodone/APAP	Female
Tashya X. Love	AA2 0AI	mauris@dignissim.com	HU866527230800199525465114Z1	10/15/2015	Carisoprodol	Male
Chaim M. Dyer	VQ5Y 1JM	dis@sitamet.co.uk	CZ2749377820767725142053	08/13/2015	Amphetamine Salts	Female
Nomlanga D. Mcbride	T19 0FC	tempor.arcu@quisilectusNullam.com	MK8516086474180364I	07/21/2015	Omeprazole (Rx)	Male
Sydney Y. Rivas	NI'98 2SI	eget.nisi@ametmassa.ca	BLG2LWML9714652062486G	08/05/2015	Zetia	Female
Lacey I. Alford	FP7 7LI	dolor@Cumsociis.net	FR3556097980335210165693483	11/29/2014	Tramadol HCl	Male
Denton E. Pacheco	KP98 8GZ	Cras.sed@velnislQuisque.net	SM2449732520321566881356088	09/04/2014	Benicar HCT	Male
Angelica T. Cobb	MS50 9UP	commodo.hendrerit@luctusut.edu	EE695752149145081545	11/18/2015	Amoxicillin Trihydrate/Potassium Clavulanate	Male
Frances D. Blair	JM64 6MB	non.sollicitudin.a@Sed.org	NL70JYWB4152220249	10/24/2014	Alprazolam	Female
Damian A. Hebert	W08 2BO	Maecenas.libero@acnulla.co.uk	GE270777355329094068I4	03/21/2015	LevothyroxineSodium	Female
Dante E. Barber	ST02 1KM	Quisque.libero@Integer.ca	PL8516045418004968575675572G	06/29/2015	Cyclobenzaprin HCl	Male
Fitzgerald K. Evans	B4Y 0LP	euismod@montes.net	SI403730964633958I5	03/10/2016	Losartan Potassium	Female
Gillian T. Sexton	RI7S 7NG	dui@facilisis.edu	CH4281243712366744146	09/09/2015	Oxycontin	Female
Travis R. Chavez	V9 9DE	turpis.egestas.Fusce@anteVivamusnon.edu	LB701060366654204364505O4980	08/04/2015	Triamcinolone Acetonide	Female
Beverly C. Strong	N4 5HQ	tempor@feugiat.edu	LB2761677237249343431524116B	09/25/2014	Metoprolol Succinate	Female
Inez K. Hartman	HN7N 9JD	tristique@nisiaodio.edu	AE2895717259902793079I1	03/12/2015	Lisinopril	Female
Holmes E. Stephenson	QG8 2JC	Duis.ac.arcu@leoVivamus.co.uk	IS9482595977098819912201I0	11/12/2014	Lisinopril	Female
Rooney R. House	K1 5CA	senectus@purusin.net	RO70WKIN8960823354672177	02/18/2016	Gabapentin	Male
Lacey W. Hopper	UK3 5NK	accumsan.convallis@sagittisNullamvitae.edu	MD1083371536772806357624	09/08/2014	Paroxetine HCl	Female
Phoebe W. Goodwin	YD1 1CV	dis.parturient@auctornon.org	KZ7738591123637110G2	07/06/2014	Lisinopril	Male
Haley T. Hubbard	O6 8ZK	Proin.mi.Aliquam@Morbivehicula.ca	SA661899714594527767439I	11/20/2014	Digoxin	Male
Zachery N. Bender	B5K 0XN	per.inceptos.hymenaeos@Sed.ca	SI65070874058999562	10/20/2014	Clonazepam	Male
Otto T. Dillon	UT77 3WB	accumsan@In.org	HU774992321472031706020210O9	02/16/2016	Lisinopril	Female
Mikayla Z. Fleming	GV08 6GC	Donec.nibh@nonbibendumsed.edu	AE971530604916560609898	11/20/2015	Abilify	Female
Jason L. Hester	A9 1VM	eget@Curabitur.ca	MT09QJII553316999454804971179I7	05/12/2014	Sertraline HCl	Female
Callie I. Barrett	DU7 1JF	luctus@vestibulummassarutrum.org	CR98192581320685783O7	10/07/2015	Triamterene/Hydrochlorothiazide	Female
Brianna K. Mueller	HU93 8KD	ligula.Nullam@euismodac.co.uk	GE043323323390735581O2	01/15/2016	Amlodipine Besylate	Male
Galena Q. Moore	MU3 3EQ	vulputate.risus@morbitristiquesenectus.ca	SM951173127932116556722440I	11/20/2015	Metformin HCl	Female
Alma O. Acosta	X9E 0SW	fringilla.purus.mauris@elitNullafacilisi.net	AD03001461148192557749471	12/16/2015	Plavix	Male
Declan J. Martin	R2 2VS	neque@quis.com	PS190171967733195470160193760	12/13/2015	Premarin	Female
Fallon D. Day	N3 9AQ	elit@ametfaucibus.ca	BE54393227321464	07/13/2015	Januvia	Female
Colby H. Pearson	PZ0 3TG	libero@Fuscedolorquam.com	MT85XFLC26704375212176862934538	09/14/2015	Celebrex	Male
Alllistair A. Hunt	E79 1YX	ac.fermentum.vel@sapienimperdietornare.com	PL182728676127216555238898753	12/05/2014	Sertraline HCl	Female
Alexis R. Hendricks	CZ3 3CZ	dictum.eu.placerat@elementum.edu	ME70477001168253291344	03/07/2016	Oxycodone/APAP	Male
Patrick G. Gallegos	SK37 0XM	tempus@enimgravidasit.co.uk	AD5083613148216280220238	07/10/2015	Tramadol HCl	Male
Evangeline E. Riggs	HA8 0VK	ante@nisia.net	BG07LORX03860949915229	12/09/2015	Lorazepam	Male
Otto C. Buckley	DF2 9QJ	Suspendisse@euismod.com	ME80681393182003179515	11/19/2015	Gabapentin	Female
Galena Q. Palmer	Y64 0YL	mollis.Phasellus@a.net	CZ7622036585780226041789	10/27/2015	Naproxen	Male
Lydia Z. Castro	P5H 4QI	nibh@lorem.edu	DK9670874642988953	03/12/2015	Lorazepam	Female
Scott F. Lynch	T45 1NA	pede.Praesent@massanon.co.uk	MC1208477210850155024428070	10/11/2014	Methylprednisolone	Male
Amethyst K. Decker	F91 1OY	consectetuer.mauris.id@dolorquam.com	SK2910499446943070006433	09/29/2015	Risperidone	Female

Figure 5.5.2 – Data Import: Successfully Loaded csv Data, System Screenshot

Exporting data from the application is a fairly simple process. After invoking the export process the user is prompted with a dialog window where they select the location of soon to be generated csv file. The output data table is then passed to the DataLoader wrapper class along with the location of the output file. The DataLoader class once again makes use of CsvHelper open source library, iterating over each row of the dataset writing it to the csv file.

5.6 Pre-Processing Tools

Once the data has been successfully been loaded into the application (Fig 8.2) the user is required to specify which attributes should be anonymised and to what extent they should be anonymised.

Building a set of controls that would allow a user to quickly and intuitively specify these requirements was not an easy task. Throughout the implementation I created several iterations of layout and constantly tested them to assess their ease of use. Eventually I finalised a design that featured a right hand pane easily readable and available directly adjacent to the input data. From here the user selects a value from a list of corresponding attributes. The right hand pane then populates anonymisation information for the attribute; this includes the associated anonymisation type (Explicit, Quasi, Sensitive or Insensitive).

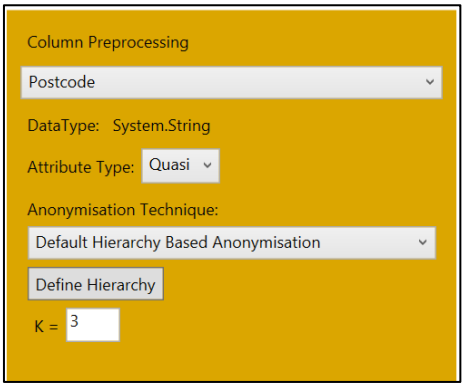


Figure 5.6.1 – Column Pre-Processing Tools, System Screenshot

Once an attribute is marked as a quasi identifier a dropdown menu listing available anonymisations and an accompanying field to specify the required k-level appear. The user proceeds to select an appropriate algorithm from the anonymisation technique drop down menu, if the resulting algorithm requires an anonymisation hierarchy, the 'Define Hierarchy' button is displayed. The resultant user interface (Fig 5.6.1) is a simple, clean and intuitive experience for users. Exposing corresponding choices such as define hierarchy only when hierarchies are a pre-requisite for the algorithm simplifies the selection process, users are not bombarded with a series of choices that are superfluous to the functionality being implemented.

The hierarchy definition tool is one of the most important features of the application. The tool facilitates two separate ways to define a hierarchy. Both methods generate hierarchies that can be used in conjunction with hierarchy-based anonymisation algorithms. Although differing in implementation both techniques have some common features; all unique values of an attribute form a set of leaf nodes in the hierarchy and the root node should be a fully anonymised value.

Deciding the most suitable format to visualise generated hierarchies was a difficult challenge during the implementation. The data structure should be displayed in such a way that users can drill down into particular levels of the hierarchy to find particular anonymisation values. Initially I attempted to display this information as a linked list of items each adjacent to the previous anonymisation value. However this didn't appear to visual the structure of the hierarchy particular well. After some experimenting I developed a tree view structure (seen in Fig.5.6.2), users can expand particular levels to discover anonymisation values and the general presentation of the resultant display is very clean and self-explanatory.

Rule Based Hierarchy Generation:
As previously discussed in section 4.6 Hierarchy Based Anonymisation of this report, hierarchies can be automatically generated by a rule. String redaction is the process where strings are anonymised by replacing the last valid character with an anonymised value such as '*'. A hierarchy of nodes can be built by anonymising each character of every unique value in the attribute. I have implemented this rule based automatic hierarchy generation approach in the anonymisation tool. The advantage of generating a hierarchy automatically is that the users are not required design and implement a new custom hierarchy each time they want to perform a hierarchy based anonymisation.
To accommodate this functionality I crated a static utility class 'StringRedactionHierarchyGenerator' that exposes a static method 'Generate'. Passing a list of strings that correspond unique values in the attribute to this method invokes the construction of an anonymisation hierarchy. The process pads the original strings to the same length and then begins the redaction process building up a list of parent nodes, eventually the root node is constructed that can represent any attribute value fully anonymised. Each unique value is retained as a leaf node. Once the hierarchy has been constructed it is retuned from the generating method.

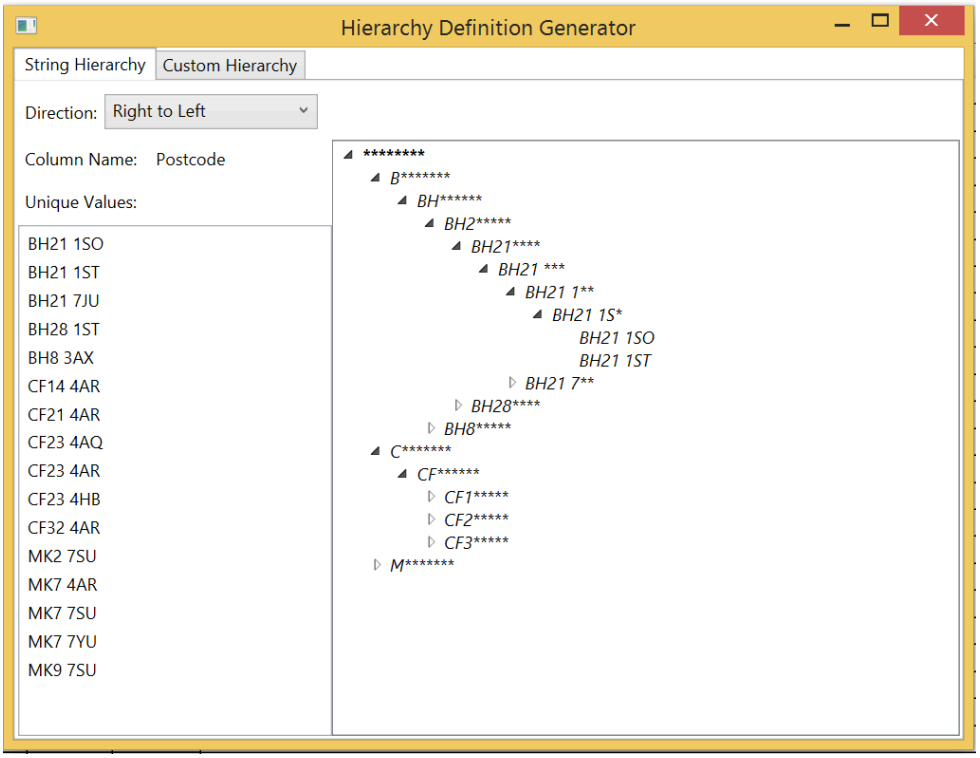


Figure 5.6.2 - Postcode Hierarchy Construction, Rule-Based String Redaction, System Screenshot

When a user opens the hierarchy definition tool they are initially presented with this rule-based construction from the corresponding attribute values. Fig 5.6.2 provides a visualisation of the rule-constructed hierarchy that is displayed to the user when opening the hierarchy definition tool. The value BH21 1S* has two child nodes BH21 1ST and BH21 1SO, both of these children will be anonymised to their parent node value if they do not satisfy the k-level threshold.

In an effort to make rule based hierarchy generation functionality flexible the user is given the choice to perform the string redaction right to left: MK7 7SU -> MK7 7S*, this option is also the default selection or left to right: MK7 7SU -> *K7 7SU. Initially this feature was not planned for the first version of the tool. However when reflecting on the functionality I came to the conclusion that the quick rule based hierarchy generation would be a very appealing feature of the system, to help users apply this process for as wide a range of situations as possible it was important to ensure this process could be tailored for specific operations. Future iterations of the tool should contain more rule based configuration options because it is a convenient trade off between reducing information loss and sufficiently anonymising relational datasets while the process to set-up such hierarchies is minimal in comparison to custom anonymisations.

Custom Hierarchy Generation:
Rule based hierarchy generation is perfectly suited for certain types of attributes. Postcode is a prime example of this; the process of obscuring data by removing the last trailing value to satisfy k-anonymity conformity attempts to retain the maximum amount of information utility. However other categories such as disease do fit this model as well. Strings close in value i.e Canavan and Cancer diseases may be completely unrelated therefore it makes no sense to group these items close together in a hierarchy. To accommodate grouping of items based on underlying meaning the tool supports the generation of custom hierarchies. This requires the user to explicitly define nodes of hierarchies. Once again common rules are imposed on the hierarchy, leaf nodes must constitute of unique values in the attribute and the root node must be a fully anonymised value. However the structure between root and leaf nodes is fully customisable and can be constructed in any way the user requires.

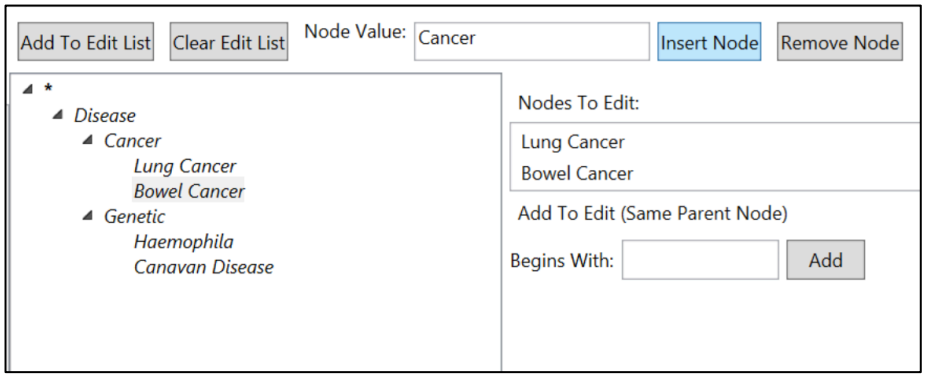


Figure 5.6.3 – Custom Hierarchy Definition, System Screenshot

The example above depicts the creation of a simple custom hierarchy. The fully anonymised root node '*' will replace any value that cannot satisfy k-level anonymity of any other node. The simplified hierarchy features 4 leaf nodes that are the original unique values of attribute 'Condition'. The user selects nodes they wish to group adding the corresponding nodes to the edit list. The example above has collated lung and bowel, inserting a new node 'Cancer' produces the result depicted in the left hand tree view pane. The edited nodes to edit become the children of the newly inserted node. 'Disease' that was originally the parent of both lung and bowel becomes the parent of the inserted node. An important point to note is that once an item has been marked for edit, only other nodes with the same parent maybe added to the list. This stops users attempting to create an illogical parent of two items that are not currently adjacent in the hierarchy and therefore should not have the same parent node.

The hierarchy definition tool will not allow users to remove leaves or the root node of the hierarchy. This has been implemented to ensure a valid hierarchy will be produced that can successfully anonymise all the original unique values. However in Fig 9.3 inserted nodes between the root and leaves can easily be removed. Should the user elect to remove the node 'Cancer' from the hierarchy both lung and bowel nodes would become the children of 'Disease'. The tool has been designed to be simple, intuitive and highly configurable, any user can easily create targeted and complex hierarchies with relatively little effort.

An additional feature of the tool that has been implemented to streamline the generation process is the 'begins with add node component'. Say the condition list contained several blood diseases (where each value would be structured 'Blood X' with X representing the second part of the disease name), to quickly group all of these items together the user would simply type 'Blood' and click the adjacent add button, at this point the system would appended all values beginning with the word blood to the edit list. Although an initial item must be added to the edit list which allows the system to work out what the targeted parent node is, because all the items in the edit list must have the same parent node.

While features of the hierarchy generation tool streamline the construction process there are a few more additions I would have liked to made however owing to time constraints I was not able to implement them. These additional features would have included an 'Ending with' equivalent to 'Begin with' that would quickly detect nodes ending with a matching string, this maybe particularly useful for values that have common trailing names i.e Disease, Heart Disease, Canavan Disease etc. The ability to import and export custom hierarchies from the anonymisation tool is another important feature that would ideally be implemented. Inclusion of this functionality would greatly enhance the appeal to potential users. Allowing users to build and create their own custom hierarchies is a highly configurable and flexible approach. I have discussed this process in greater detail within section future work.

Hierarchical Tree View UI:
The underlying UI code to present both rule based and custom hierarchies is exactly the same. A singular tree view component has been used to visualise the levels of the structure and allows the nesting of child elements inside nodes (Fig 5.6.2 & Fig 5.6.3). At the beginning of the implementation I discussed the different technologies I would be leveraging in order to create the anonymisation tool. WPF was the favoured method to arrange and present graphical components. As previously mentioned WPF uses Xaml an extension of xml to arrange UI controls in a view, a C# ViewModel is then used to populate data in these controls via data binding. The code snippet below demonstrates the elegance and flexibility of the platform. In order to display potentially thousands of nested hierarchy items, 3 simple Xaml statements have been written.

```
<HierarchicalDataTemplate x:Key="ChildTemplate" ItemsSource="{Binding Path=ChildNodes}">
    <TextBlock FontStyle="Italic" Text="{Binding Path=Value}" />
</HierarchicalDataTemplate>
<HierarchicalDataTemplate x:Key="RootTemplate" ItemsSource="{Binding Path=ChildNodes}" ItemTemplate="{StaticResource ChildTemplate}">
    <StackPanel Orientation="Vertical">
        <TextBlock Text="{Binding Path=Value}" FontWeight="Bold" />
    </StackPanel>
</HierarchicalDataTemplate>

</Grid.Resources>

<TreeView ItemsSource="{Binding Nodes}" ItemTemplate="{StaticResource RootTemplate}"
          SelectedItemChanged="hierarchyTreeView_SelectedItemChanged" x:Name="hierarchyTreeView" />
```

Figure 5.6.4 –Tree View Display Code Snippet, HierarchyTreeView.xaml, Visual Studio

The first 2 statements are hierarchical data templates. Data templates are used in WPF to arrange a UI component when the hosting control is populated with items. Therefore when the tree view control is populated, both of the Hierarchical templates are utilised to display the bound data.

RootTemplate – This template is applied to the root node of the tree. As the data template is hierarchical it sets the item source of this node as ChildNodes, which means that the children of the current node in the corresponding ViewModel will populate child nodes.

ChildTemplate – This template is used to display all items in the tree view expect the root node of the tree. Similar to the root template children of the selected node in the corresponding ViewModel will populate further hierarchical child nodes.

The final statement defines the tree view control that hosts the hierarchical data templates. ‘ItemsSource’ refers to list of objects that have been linked to the component through data binding via a ViewModel, the attached ViewModel is expected to expose a list of ‘Nodes’ which will be used to populate the tree. An event handler is attached using the SelectedItemChanged attribute, this notifies a section of code when different items are selected from the tree view and is used to construct custom hierarchies.

In total 8 lines of UI code has been written that will automatically display up to thousands of nodes when populated with data. I personally believe this example demonstrates the power of WPF and vindicates my choice to use the technology for the implementation of the anonymisation tool UI. As the component was created as a modular ‘HierarichalTreeView’ control I have reused the same component to display rule based and custom hierarchies. Implementing the control this way stopped me having to write the same snippet of code twice and future developers can alter both tree view layouts be easily refactoring the singular common component.

Once a user has generated the hierarchy they want to utilise of the anonymisation process, they ensure the corresponding tab item has been selected when closing the hierarchy definition tool. To select the rule based anonymisation close the tool with the ‘String Hierarchy’ tab selected. Alternatively selecting the ‘Custom Hierarchy’ tab before closing the tool can enable the custom defined hierarchy.

5.7 Anonymisation Algorithms Implementation

As detailed in section 5.4 project structure the solution features a series of C# projects named with the precursor KAnonymisation. Each of these projects contains some form of anonymisation implementation.

Set Based k-members Clustering Anonymisation:
Section 4.5 of the report details the design of the set-based k-members clustering algorithm which has been implemented. Anonymisation of explicit identifiers replaces each value with an asterisk in order to obfuscate the original value. The tool allows each algorithm to apply it's own explicit identifier anonymisation; this has been done in order to make the platform highly configurable. The pseudocode from Co-Clustering paper by A. Kawano [7] has been used as a basis for my C# implementation in the tool.

One difficulty encountered when implementing the set based algorithm was attempting to design a singular piece of code that could handle multiple data types. Initial implementations featured correctly performing processes but they system would only execute with integer data types. Obviously the tool should be able to handle multiple data types, which adds flexibility that is important to the audience of this project. After research I decided to employ the use generics to help solve this particular problem. The use of C# generics is obvious from looking at the source code (See Fig 5.7.1). Generics have been employed to allow re-use of methods by a range of data types, such a process avoids repeated method implementations for every single of data type that maybe passed into the algorithm.

Microsoft developer notes explains "using a generic type parameter T you can write a single class that other client code can use without incurring the cost or risk of runtime casts or boxing operations…Use generic types to maximize code reuse, type safety, and performance". [12]

```
class GenericsExample
{
    private List<int> _listOfIntegers = new List<int>() { 3, 5, 1, 7 };
    private List<string> _listOfStrings = new List<string>() { "foo", "bar", "thump" };

    private T ReturnFirstGenericValue<T>(List<T> genericListofItems)
    {
        return genericListofItems.First();
    }


    private void GenericExampleMethod()
    {
        var intResult = ReturnFirstGenericValue<int>(_listOfIntegers);
        var strResult = ReturnFirstGenericValue<string>(_listOfStrings);
    }
}
```

Figure 5.7.1 – C# Generics Example, Visual Studio

The generic example shows that one singular method can be used to perform operations on both integers and strings. Without the use of generics the same method would have to be rewritten separately for each data type.

The clustering algorithm requires the operations nearest/furthest value given an initial value and a list of items. The data anonymisation tool was designed to anonymise many different data types, without the use of generics every data type would require a separate nearest/furthest functions (NearsetString/FurthestString, NearsetInt/FurthestInt, etc). However I was successfully able to implement these as generic methods, which keeps the implementation code clean, minimal and ideal for maintenance. Generics where also utilised to return the nearest cluster to a specified value. Once again because of the implementation any generic data type can be used and the function will return a generic list, which represents a cluster of that data type.

```
namespace KAnonymisation.SetBased
{
    public class SetBasedAnonymisation : IKAnonymisation
    {
        public string Name...
        public bool RequiresHierarchy ...

        public void ApplyAnonymisation(ref DataTable dataTable, ColumnModel columnModel)...

        private void AnonymiseExplicitIdentifier(ColumnModel columnModel, ref DataTable dataTable)...
        private void AnonymiseQuasiIdentifier(ColumnModel columnModel, ref DataTable dataTable)...
        private void EditRow(DataRow row, string header,string toUpdate)...

        private List<List<T>> ClusterKMembers<T>(int k, List<T> values) where T : IComparable...
        private T FurthestVal<T>(T r, ref List<T> shuffledArray) where T : IComparable ...
        private T NearestVal<T>(T r, ref List<T> shuffledArray) where T : IComparable...
        private Dictionary<T, double> CalcSimilairtyDict<T>(T r, List<T> shuffledArray)...

        private Dictionary<T, double> CalcDoubleVals<T>(object r, List<T> shuffledArray)...
        private Dictionary<T, double> CalcIntVals<T>(T r, List<T> shuffledArray)...
        private Dictionary<T, double> CalcLevenstheinVals<T>(T r, List<T> shuffledArray)...
        private Dictionary<T, double> CalcDateVals<T>(T r, List<T> shuffledArray)...
        private List<T> NearestCluster<T>(T r, ref List<List<T>> clusters) where T : IComparable ...
    }
}
```

Figure 5.7.2 – Set Based Anonymisation C# Implementation, Visual Studio

While generic methods can be used to reduce the level of code repetition they are not appropriate to perform the calculations that quantify the closeness of two values (although this process is wrapped by generic methods e.g nearest/furthest value). Different data types require specific calculations to measure distance, for example numeric values can be quantified by Euclidean distance of two values. Where as strings can be measured using Levenshtein distance that returns a number based on the closeness of two strings. The Levenshtein distance calculation featured in this algorithm has been taken from a third party source [13] and verified as correctly functioning (see unit testing Fig 5.9.2). Support for date data types has also been added; closeness is quantified as the number of days between respective dates.

This varied difference in calculation techniques means it is unreasonable to try and use generic methods to calculate the distance between all different data types. However in order to facilitate a data type that is not currently supported, a single calculation method would need to be added plus a small tweak to the code to notify the algorithm that the new data type is available for use. Then the generic methods would operate as before returning closest and furthest values of the new data type based on the calculation function. This simple extensibility means that future development would easily be able to accommodate more obscure data types that I haven't had time to implement throughout the project.

The edit row helper method is used to alter the contents of the dataset. To edit a data table entry a series of operations; begin edit, update and accept changes are required to be performed. Therefore this process has been wrapped into a helper method that in turn is invoked by the clustering algorithm at the appropriate time.

```
private List<List<T>> ClusterKMembers<T>(int k, List<T> values) where T : IComparable
{
    var clusters = new List<List<T>>();
    var rand = new Random();

    //shuffled values via LINQ statment
    var shuffledList = values.OrderBy(val => rand.Next()).ToList();

    if (shuffledList.Count < 1)
        return clusters;

    var r = shuffledList.First();

    while (shuffledList.Count >= k)
    {
        r = FurthestVal<T>(r, ref shuffledList);
        shuffledList.Remove(r);

        var cluster = new List<T>();
        //cluster container used to count number of items, can't expliclty count occurences of numbers
        //in case the same number is added to the cluster i.e {2, 2, 4} -> {2, 4}
        int clusterContainer = 1;
        cluster.Add(r);

        while (clusterContainer < k)
        {
            r = NearestVal<T>(r, ref shuffledList);
            if (!cluster.Contains(r))
                cluster.Add(r);
            shuffledList.Remove(r);
            clusterContainer++;
        }
        clusters.Add(cluster);
    }

    while (shuffledList.Count > 0)
    {
        r = shuffledList.First();
        var nearestCluster = NearestCluster<T>(r, ref clusters);

        if (!nearestCluster.Contains(r))
          nearestCluster.Add(r);

        shuffledList.Remove(r);
    }

    return clusters;
}
```

Figure 5.7.3 - k-members Set Based Generic Implementation, Visual Studio

The implementation (Fig 5.7.3) follows very closely to the designed algorithm within section 4. A few changes have been made to ensure that the process can handle multiple data types by implementing a generic approach that has been discussed above. A list of values and a nominal integer k value are fed to the algorithm, it then builds a number of clusters each of size k related to closeness of the items. This is repeated until less than k items are left so no more clusters can be formed. Then the algorithm uses the closest cluster function to iteratively add each remaining item to its closest related cluster. Once the series of clusters have been built then algorithm then passes over the data table updating singular values within a replacement set where appropriate. As the data set is passed by reference the anonymisation process is applied to the dataset that is passed from the client side of the anonymisation tool that invokes the algorithm therefore no value needs to be returned by the method ‘ApplyAnonymisation’.

Hierarchy Based Scan and Replace k-Anonymisation:
The other default anonymisation algorithm that has been implemented within the system is a hierarchy-based scan and replace. Once again the implementation is contained within a single C# file which implements the IKAnonymisation interface. The specific design details of the technique are listed within section 4; algorithm design. The process requires a predefined anonymisation hierarchy. This is essentially a tree structure where the leaf nodes are formed by unique values in the dataset and the root is a fully anonymised value, therefore the levels between leaves and root represents the different levels of anonymisation.

Unlike the set based anonymisation this algorithm relies on a hierarchy to anonymise values. The anonymisation levels within this data structure therefore there is no need to group items related on a concept of 'closeness' at run time. This means that all the occurrences of values can be counted after casting all values to a string, so there is no need to implement generic methods to handle various data types.

```
namespace KAnonymisation.Hierarchy
{
    public class HierarchyBasedAnonymisation : IKAnonymisation
    {
        public string Name...
        public bool RequiresHierarchy...

        public void ApplyAnonymisation(ref DataTable dataTable, ColumnModel columnModel)...

        private void EditRow(DataRow row, string header, string toUpdate)...

        private void AnonymiseExplicitIdentifier(ColumnModel columnModel, ref DataTable dataTable)...
        private void AnonymiseQuasiIdentifier(ColumnModel columnModel, ref DataTable dataTable)...
        private void ApplyAnonymisedValues(Dictionary<string, string> newAnonValues, ref DataTable dataTable, ColumnModel columnModel)...

        private List<string> FindValuesToBeAnonymised(Dictionary<string, int> valueLookup, int k, AnonymisationHierarchy anonHierarchy)...

        private Dictionary<string, int> CountOccurances(DataTable dataTable, ColumnModel columnModel)...
        private Dictionary<string, string> FindNextLevelAnonymisation(List<string> valsToBeAnonymised, AnonymisationHierarchy anonymisationHi
    }
}
```

Figure 5.7.4 – HierarchyBasedAnonymisation.cs C# Implementation, Visual Studio

The algorithm iterates over the dataset counting the frequency of each item, results are added to a dictionary data structure and returned from the 'CountOccurances' method. The find values to be anonymised function inspects this data to identify values that should be anonymised and discards values that have met the k-level threshold value. Also worth noting that any value, which is the root of the anonymistation tree, is also discarded as it has been fully anonymised therefore no more action is required.

```
private Dictionary<string, string> FindNextLevelAnonymisation(List<string> valsToBeAnonymised,
                                                              AnonymisationHierarchy anonymisationHierarchy)
{
    // Produces a dictionary where the item to be anonymised is the key and
    // the newly anonymised object is the value. New value is generated by navigating
    // to the parent node in the anonymisation hierarchy.

    var result = new Dictionary<string, string>();
    var nodes = new List<Node>();

    if (valsToBeAnonymised == null || anonymisationHierarchy == null)
        throw new Exception("Error Applying Anonymisations Levels");

    //finds corresponding nodes in the anonymisation hierarchy
    foreach(var val in valsToBeAnonymised)
        nodes.Add(anonymisationHierarchy.FindNode(val));


    foreach(var node in nodes)
    {
        if (result != null && !result.ContainsKey(node.Value))
        {
            // Ternary statement is essentailly a compacted if statement
            var parentValue = (node.ParentNode == null) ? null : node.ParentNode.Value;
            result.Add(node.Value, parentValue);
        }
    }

    return result;
}
```

Figure 5.7.5 – Generate Next Set of Values to be Anonymised From Hierarchy Data structure

Next the method 'FindNextLevelAnonymisation' (Fig 5.7.5) iterates through the list of values to be anonymised generating the next level of anonymisation values from the corresponding hierarchy. This method generates a dictionary that contains items to be anonymised as the key and the new level of anonymisation as the value. Once this dictionary data structure is returned to the calling method the information is used to alter the contents of the dataset by iterating over items and replacing with the new corresponding value where appropriate.

The same helper method to edit data table rows from the set based algorithm is present in this implementation and once again provides an encapsulated easy way for the algorithm to alter the information stored. As described in section 4; algorithm design, once a value meets the threshold it is discarded and no further action is performed, as appose to finding close values and then grouping information together in an effort to satisfy the threshold. This was a conscious design decision that has been previously justified in the design section and been implemented successfully. The algorithm repeats the process of counting occurrences and applying the next level of anonymisation until the all values satisfy the threshold and the dataset is k-complaint or the remaining values have been fully anonymised with the root level of the anonymisation hierarchy. The original dataset is passed by reference to the algorithm so once anonymisation has been applied it is returned to the calling client code as the anonymised dataset.

Anonymisation Algorithm Loading:
Once the algorithms had been implemented they must be loaded into the tool to ensure they are available for use. The observable collection 'AvailableKAnonymisations' holds an instance of each algorithm; the list populates the client UI ready for user selection. One of the main goals of the tool was to be able to dynamically load plugin algorithms at run time that would allow users to access techniques that were not hard coded into the original system. The source code below is the final implementation of this process that achieved this specification goal.

```
private void LoadAvailableKAnonymisations()
{
    //Defaults hard coded to load
    IKAnonymisation defaultSetBasedAnon = new SetBasedAnonymisation();
    IKAnonymisation defaultHierarchyBasedAnon = new HierarchyBasedAnonymisation();
    AvailableKAnonymisations = new ObservableCollection<IKAnonymisation>() { defaultSetBasedAnon, defaultHierarchyBasedAnon };

    if (!_loadPluginAnonymisations)
        return;

    try
    {
        //Dynamically load anonymisation plugins
        List<IKAnonymisation> pluginAnons = DynamicallyLoadAnonymisationPlugins();
        foreach (var iKAnon in pluginAnons)
            AvailableKAnonymisations.Add(iKAnon);
    }
    catch(Exception ex)
    {
        var msgBox = MessageBox.Show(ex.Message, "Error Dynamically Loading Anonymisation Plugins");
    }
}
private List<IKAnonymisation> DynamicallyLoadAnonymisationPlugins()
{
    var result = new List<IKAnonymisation>();

    var curDir = Directory.GetCurrentDirectory();
    var baseDir = Application.StartupPath;
    var pPath = "\\Plugins";
    var fullPath = string.Format("{0}{1}", baseDir, pPath);

    string[] plugins = Directory.GetFiles(fullPath, "*.DLL");

    foreach(var plugin in plugins)
    {
        var assembly = Assembly.LoadFile(plugin);
        if(assembly != null)
        {
            var type = typeof(IKAnonymisation);
            var types = assembly.GetTypes();
            foreach(var t in  types)
            {
                if (type.IsAssignableFrom(t))
                    result.Add((IKAnonymisation)Activator.CreateInstance(t));
            }
        }
    }
    return result;
}
```

Figure 5.7.6 – Dynamic Loading of Plugin Algorithms + Default Set and Hierarchy Based Anonymisations: PreprocessingColumnsVm.cs, Visual Studio

The method 'LoadAvaibleKAnonymisation' initially instantiates the default set and hierarchy based anonymisation algorithms detailed in this section of the report. If the flag to dynamically load plugin algorithms is set to true then the code inspects a local directory labelled 'Plugins' to search for any files that implement the IKAnonymisation interface. If any matching files are found they are loaded dynamically (see activator, create instance statement) and appended to a list of plugin algorithms available to the system. This list is returned to the calling function and any found plugins algorithms are added to the observable collection and subsequently populate the user interface ready for selection.
The source code for each algorithm can be inspected line-by-line in the report appendix, which details every implementation file used to build the system.

5.8 Post-Processing Tools

The two objects of PPDP are to maintain information utility and provide a sufficient level of anonymisation to protect individual privacy. Once a dataset has been successfully anonymised the user may want to query the level of retained information utility compared to original data. To facilitate this the tool implements a range of post processing tools that allow users to extract information utility metrics.

This report primarily implements metrics used to measure information loss, however an important point to consider is the secureness of anonymised data generated by the tool. Metrics providing details about the secureness of anonymisation are rarely extracted from the dataset. However one opinion is that this process is superfluous as the anonymisation criteria have been fulfilled d by ensuring the table achieves k-anonymity conformity. While implementing the system I shared this opinion that achieving k-anonymity within a dataset ensures the personal privacy of an individual. The system should then support post-processing operations to measure the retained data utility. In this section I have detailed two approaches to calculating metrics on anonymised data. However in the future other developers may wish to expand the functionality to facilitate personal privacy metric calculations, but for now privacy protection is assumed if k level anonymity has been satisfied.

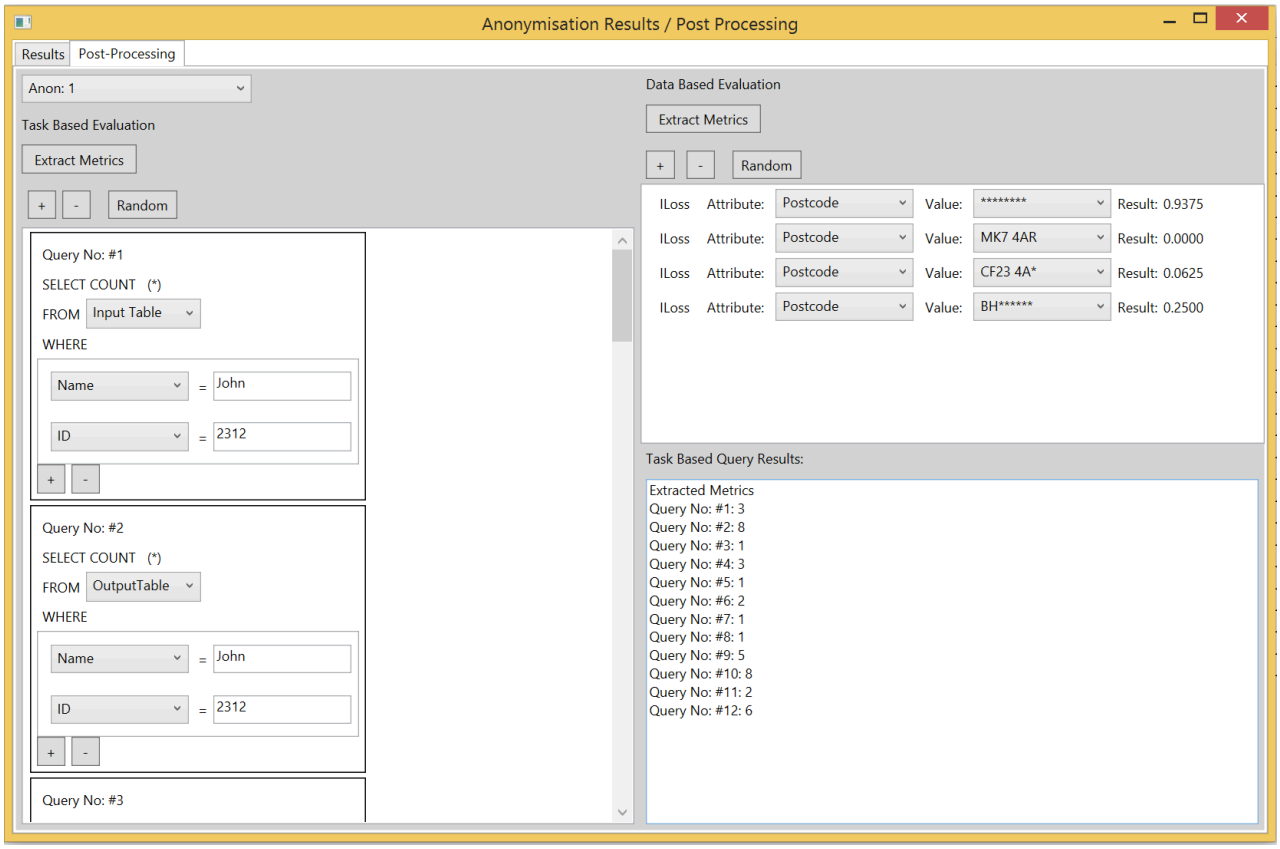


Figure 5.7.7 – Post Processing Results Dashboard, System Screenshot

The system implements two approaches to calculate information loss metrics. Task and data based measurements are utilised to understand how the information is altered and whether it retains its original meaning. The system supports a post-processing dashboard (Fig 5.7.7) for each dataset anonymisation. Here the user can design and implement a set of custom post processing calculations. This functionality is particularly useful if the user wants to drill down to a specific set of values and see how the anonymisation algorithm affects the information utility. Alternatively the system can automatically generate a set of random calculations for both task and data based metrics.

Data Based Measurements:
Benjamin Fung discusses information loss calculations in a research paper referenced below. One calculation explored was the ILoss general-purpose metric. This calculation assumes a hierarchy of anonymisation values. Essentially the calculation is formed between the numbers of leaf node descendants of vg minus 1 divided by the total number of leaf nodes for a given attribute.

$$ILoss(v_g) = \frac{|v_g|-1}{|D_A|}$$

"Where $|v_g|$ is the number of domain values that are descendants of v_g, and $|D_A|$ is the number of domain values in the attribute A of v_g. This data metric requires all original data values to be at the leaves in the taxonomy. ILoss(v_g) = 0 if v_g is an original data value in the table. In words, ILoss(v_g) measures the fraction of domain values generalized by v_g." [6]

The ILoss calculation is an incredibly simple and effective measurement of information loss. The larger the result the more distorted the anonymised value has become. It is important to extract these types of metrics from the output data because it informs users on the level of data utility that has been retained during the anonymisation process. Obviously if a dataset achieves k-anonymity conformity but has to throw away huge amounts of information the output may not be suitable for practical application, therefore it is important to calculation information loss metrics.

The system leverages the defined hierarchies used to anonymise the data to form the basis for these calculations. The example in Fig 5.7.8 shows the data anonymisation tool calculating ILoss metrics for the attribute Postcode. The value combo box has been populated from the nodes of the anonymisation hierarchy that was automatically generated by rule-based string redaction of the unique values, see Fig 5.6.2. The application allows users to specify custom calculations and configure them as they wish, to add or remove a calculation select the corresponding + or - button. Alternatively the random button populates 4 different ILoss calculations for each attribute that has a valid hierarchy defined.

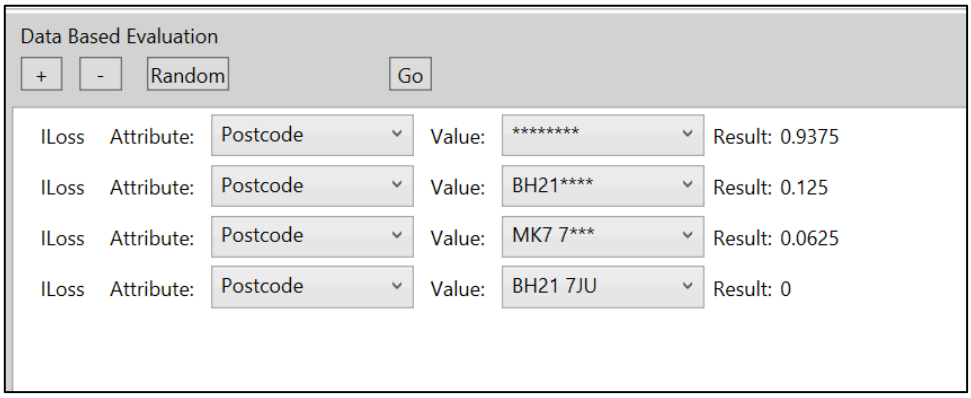


Figure 5.7.8 – Post Processing ILoss Calculation, System Screenshot

The anonymisation hierarchy featured has been automatically generated by a rule-based string redaction of unique postcode values displayed in the left hand pane of the screen shot. At this point it is worth noting the differences between the actual leaf nodes and the nodes that could be part of the hierarchy. As stated the actual leaf nodes are derived from a set of unique values, however the hierarchy does not feature all the possible values. BH21 1S* features two actual leaf nodes that are present in the original dataset. Although once the string has been anonymised the original value could have been BH21 1S[A-Z], 1 of 26 possibilities. Once the information has been exported from the anonymisation tool then anyone reviewing the information should not have access to the original data and therefore no way of discerning the set of items that anonymised values have been formed with. Generating this extended hierarchy is not required for the anonymisation process and would needlessly increase the amount of resource required by the application. When calculating the information loss with the ILoss calculation I have opted to use the anonymisation hierarchy as above, instead of the range of possible values. This decision was made because I want the system to convey the amount of information loss in comparison to the original data set. Therefore ILoss calculations will extract metrics based on nodes present in the anonymisation hierarchy. However the system could be easily altered to calculate this metric against a set of all possible values.

A series of unit tests have been added to the application build process. This ensures that ILoss calculations are being performed correctly and returning the correct pre-calculated value. This process gives the developer confidence that any changes that have been made, will not impact the functionality of the code. When the project is being built each unit test run and if it passes the test and gives the expected result the unit of code is preforming as intended. However for the purposes of demonstration I will show a worked example of an ILoss metric calculation from Fig 5.7.8.

Worked ILoss Metric Examples:
Total Number of Leaves in hierarchy: 16

BH21****
Descendant Leaf Nodes:
BH21 1ST
BH21 1SO
BH21 7JU
Total Descendant leaf nodes: 3
ILoss(BH21****) = (3 – 1) / 16
= 0.125

BH21 7JU,
Total nodes: 0
ILoss(BH21 7JU) = (0 – 1)/ 16
= -0.2 --> 0
cannot have information gain therefore result
= 0

Task Based Measurements:
Information loss metrics can be calculated by performing a common task on both the input and output datasets. The system allows users to design and run multiple tailored search queries to assess differences in information utility. The search queries are extremely flexible, users configure the arrangement of criteria to be fulfilled, any number of constraints can be easily added or removed from the query.

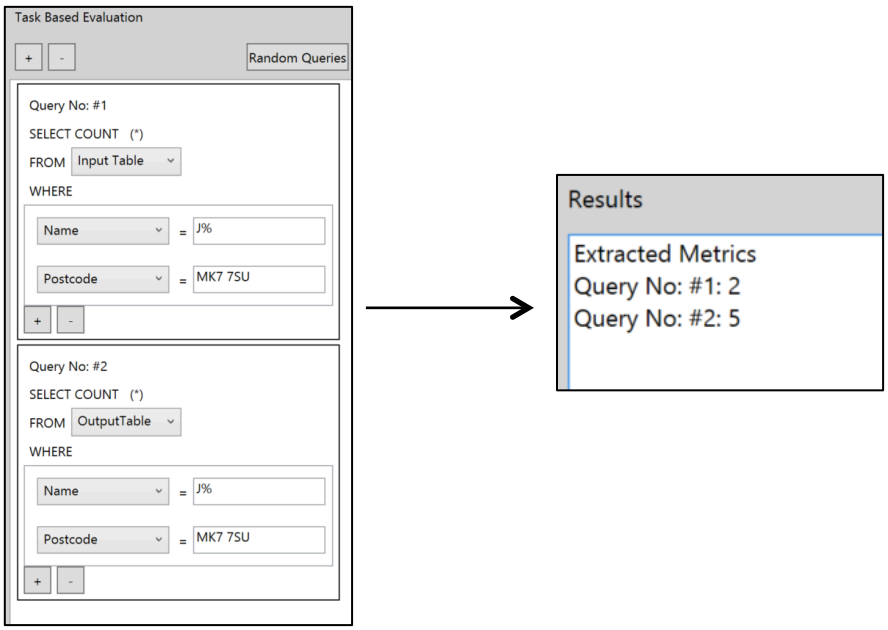


Figure 5.7.9 – Task Based Evaluation Queries + Results, System Screenshot

The two queries featured above search the input and output datasets respectively for a set of pre-defined criteria. Additional constrains could be added to each individual query using the corresponding + and – buttons nested within each query window. Entirely new queries can be appended or removed from the query planner by once again using the + and – buttons featured at the top of the view situated under the title 'Task Based Evaluation'. The flexibility and customisable nature of the query planner allows users to define and run highly targeted analysis on the data. This was considered a key feature when implementing the post-processing tools and so great care has been taken to ensure that users can change and detail precise criteria of task based queries. The task based query solution includes complex features such as wildcard searching, set based matching and anonymised data matches that are necessary to extract the appropriate information loss metrics from the datasets.

Wildcard Search – The use of special character % indicates to the system that the user would like to return data that fits the criteria preceding the percentage symbol, the criteria after the symbol matches regardless.

Criteria: J%
Possible Matches: J, James, Justin, Joanne

The excerpt above details a few possible matches to a wild card search criteria. As discussed only information preceding the % character is used to identify possible matches. Wildcard searching is an important feature of the query planner functionality because it allows users to easily select groups of data without having to reference each member of the group individually.

Set Based Matching – Previously detailed within this report the tool implements both set and hierarchy based anonymisation techniques. Values maybe anonymised as a set to satisfy k level anonymity threshold, this result of this means that one anonymised item may contain several values.

Criteria: 19
Possible Matches 19, {19, 22}, {19 ,41, 67}

The specified criteria 19 will be matched exactly if queried against the original data table. However queried against an anonymised table that contains sets the system should match wherever an exact or possible match occurs. The use of curly braces { } indicates to the system that a set has been formed containing a number of values. Querying the original criteria 19 against a set causes the system to evaluate if the value; 19 occurs in any of sets. Without access to the original table there is no way that the application can identify if a 19 was the original value of a set, therefore possible matches are returned. The increased number of possible matches indicates the amount of information incurred by the anonymisation process. List of possible matches above depict this possible set based matching process.

Anonymised Matches – The core functionality of the system is intended to anonymise data in such a way that it cannot be re-identified and linked back to its individual owner. Querying a dataset to find possible occurrences of information therefore requires some thought when performed on anonymised data.

Criteria: MK7 7SU
Possible Matches: MK7 7SU, MK7 7**, *

The example above specifies a postcode that will be used to identify matches within the data. When performed on the original dataset, only exact matches to the string MK7 7SU will be retuned as occurrences of the data. However MK7 7** may feature in an anonymised data in order to satisfy the defined k-level anonymity threshold. Looking only at the output anonymised data there is no way of discerning what the original value was, the anonymised characters may well have been ‘SU’ and would therefore meet the search criteria defined. So requiring that anonymised values may have been derived from the search criteria a possible match should be identified. This holds true for both the value MK7 7** and the fully anonymised value * and therefore are both possible matches of the original search criteria MK7 7SU.

Also featured in Fig 9 are the corresponding extracted metrics. The results indicate that the query #1 has found 2 occurrences of individuals who’s name begins with the letter J and live in the postcode MK7 7SU from the original input table. While the second query performed on the anonymised dataset returned 5 possible occurrences. The difference between these two values informs the user about the information loss between the input and output datasets. Clearly some information utility degradation has occurred within the data because more possible matches have been identified comparing non-anonymised to anonymised data. Running a number of these task based queries over the input and output datasets would allow users to build up a picture of how the data has been altered in order to satisfy k anonymity conformity.

Functionality to display information loss metrics with graphing was discussed in my initial project plan. Unfortunately due to time constraints I have been unable to implement these features. Implementing the post processing metric extraction was another significant challenge throughout the creation of the anonymisation tool. Developing these aspects took significantly longer than I had first anticipated, especially to produce the behaviour that extracts metrics for set based anonymisation and wild card support. While the graphing features have not been the underlying infrastructure to extract these metrics has been built, simply extending the system would enable users to access graphing of these metrics. This is one of the major work items I would like to add as future work because I feel it would be an extremely useful addition to the system. Although taking longer than initially expected, the metric extraction features work extremely well and provide a solid base for future development in this area.

5.9 Testing

Throughout the implementation of this system I have leveraged several different types of testing to ensure system functionality and validity of results during the development phase.

Unit Testing – These tests are designed by the developer and used to ensure the functionality of internal application components. Unit tests are coded into the project solution and can be part of the application build process to ensure the components being compiled into the system are performing as expected. Unit tests often function by running a set of pre-defined test conditions leveraging a unit of functionality from the system, returned results are then evaluated for correctness using assertion statements. If a value is not asserted as the expected result the test fails and the developer is. Unit tests are especially useful if a developer is refactoring a set of functionality. Ensuring no unintended side affects have been introduced during a refactor of the code base can be achieved by ensuring operational unit tests are passing during build of the application, therefore the system is still performing as expected.

Black Box Testing – Entails testing the application functionality without attempting to understand the internal logic. Sets of values are passed into the black box that encapsulates application logic; the tester then evaluates the result that returned. If the returned result is expected the system has passed the test. Otherwise the application has failed to produce an expected result and the cause of this should be investigated and fixed.

As previously discussed in section 5.4 Project Structure I have implemented a unit-testing library that is run every time I build the application from Visual Studio. The project 'DataAnonTool.Tests' contains 5 separate test classes, which between themselves contain 17 individual test methods. The majority of these test methods contain multiple assertion statements.

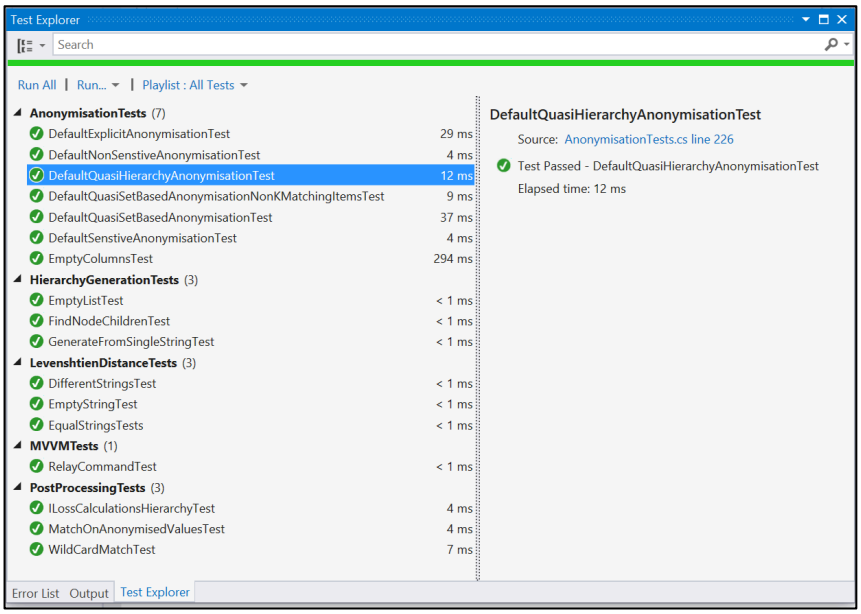


Figure 5.9.1 – Unit Tests Results Explorer, Visual Studio

As depicted in the screenshot above I have tested units of functionality from varying areas of the system. Ranging from Levenshtein string difference calculations to Model View ViewModel infrastructure tests. The named tests classes are self-explanatory to their content and intended testing purpose. An important point to note is that unit tests cannot provide complete code coverage to ensure that every single possible execution of the code unit functions as expected. It would be impractical to think of minutely possible but plausible edge cases that unit tests should accommodate, in addition to this the time it would take implement these tests would take an unacceptably long period of time. There would be no point spending 1 week developing functionality to then spend 6 weeks exhaustively testing every possible outcome, there has to be a reasonable cost/benefit pay off to justify implementing a sensible number of unit tests. Due to these reasons I have implemented a series of key unit tests which evaluate the core components of the anonymisation tool. The tests that have been performed on these components are also general targeted cases to indicate if the system was functioning as intended.

```
[TestClass]
public class LevenshteinDistanceTests
{
    [TestMethod]
    public void EqualStringsTests()
    {
        var str1 = "foo";
        var str2 = "foo";

        var result = LevenshteinDistance.Compute(str1, str2);
        Assert.IsTrue(result == 0.0);
    }

    [TestMethod]
    public void DifferentStringsTest()
    {
        var str1 = "foo";
        var str2 = "bar";

        var result = LevenshteinDistance.Compute(str1, str2);
        Assert.IsTrue(result == 3.0);
    }
```

Figure 5.9.2 – Levenshtein Distance Unit Tests, Visual Studio

The screenshot above shows two unit test methods that have been used to evaluate results produced from the utility class ‘LevenshteinDistance’. This class returns an integer that varies in size depending on the similarity between two strings passed into the compute method. Therefore I have implemented these two test methods (other methods have also been written) to evaluate the unit functionality of the Levenshtein distance calculation unit of code.

Unit test classes and methods must be decorated with the appropriate attribute [TestClass] and [TestsMethod] respectively, this enables Visual Studio to detect and run the corresponding tests. As with all the unit tests I have implemented, both test methods have been descriptively named. I have selected these two unit tests because they are simple examples and demonstrate the general structure of a unit test. After the utility class calculates the difference between the two strings an assertion is used to evaluate the expected result against the actual result. Should a developer refactor how the utility class calculates Levenshtein distance these two unit tests should remain in the project. Once the code has been refactored running the unit test will allow the developer to confirm no bugs have been introduced in the code if the units are still preforming as expected. Placing unit test around the key components of the system increases the confidence that the developed system is working as intended.

I have leveraged black box testing throughout the implementation of the application. This approach has been especially useful when evaluating the functionality of anonymisations on datasets. Given a small-restricted data sample (see examples in section 4, algorithm design) passed into the anonymisation algorithm. I would then observe the result that was returned from the algorithm, which acted as a black box. When the algorithm generated results I was expecting I knew the internal logic was functioning as intended. On the other hand when the actual result did not match the expected result I knew the algorithm had a bug and I then proceeded to find and fix the issue.

For example defining a set based anonymisation with k threshold value equal to 3. If the dataset contains 3 separate individual values, I would expect the algorithm to replace all instances in the table with a set that comprised of all 3 values. Passing my known data into the black box anonymisation algorithm, I could then evaluate if my set based process was functioning correctly. An important thing to note about black box testing is that as the tester I was required to define exactly what was expected as output, if these assumptions were incorrect I could have missed a bug in the system. This meant I needed to develop in depth knowledge about the different algorithms I was implementing, and understand the differing impact each of these would have on my controlled dataset. The input to a black box test must also be strictly controlled, I used a few different input datasets to ensure validity and test different scenarios the algorithms would have to deal with.

Testing has been a significant aspect of implementation. The combination of unit and black box testing has ensured that the functionality of the system remained of paramount importance. The anonymisation algorithms are the core functionality of the tool, therefore ensuring correct valid results has been the focus of my development. I feel this approach has contributed towards a final system that produces accurate and reliable anonymisations.

6 Results & Evaluation

This section of the report aims to demonstrate and critically assess the performance of the resultant tool built throughout the project. In order to achieve this several test cases will be used exhibit system behaviour. Once the test case result have been inspected a critical evaluation of results and explanation of the selection of tests will also be justified. The figures displayed below are screenshots taken from the tool before/after the anonymisation has been applied. The test cases feature a series of fictional data that I have created for purposes of testing the system.

6.1 Test Cases

NoOfTreatments	Letter
1	A
2	B
5	C
6	D
9	E
10	F

Figure 6.1.1 – Test Case 1: Input Data Set, System Screenshot

This test case demonstrates the set-based anonymisation algorithm behaviour. Specifically the difference in operation when opting for various k-level thresholds. The initial input data set is an extremely concise table, an integer attribute 'NoOfTreatments' that will be used to assess the performance of the default set-based algorithm which has been hard coded into the system.

NoOfTreatments	Letter
{1, 2}	A
{1, 2}	B
{6, 5}	C
{6, 5}	D
{10, 9}	E
{10, 9}	F

NoOfTreatments	Letter
{1, 2, 5}	A
{1, 2, 5}	B
{1, 2, 5}	C
{10, 9, 6}	D
{10, 9, 6}	E
{10, 9, 6}	F

Figure 6.1.2 – Test Case 1: Output Anonymised Data Set (k=2 vs. k=3), System Screenshot

The output datasets have been successfully anonymised, the left hand table k=2 where as the right hand table k=3. The key function of this algorithm is to generate a series of clusters, each must contain at least k elements. Once the clusters have been built they are applied to the original dataset replacing singular values that have not met the k-level threshold. Examining both test cases it is clear to see the appropriate number of items has been added to each cluster, 2 and 3 respectively left to right. Clusters that have been built also contain values that are 'close' in meaning to each other.

For example grouping the two lowest integers into a set {1, 2} for the k=2 anonymisation process is exactly the result that was expected. The greedy set based algorithm evaluates the list of values to be anonymised and groups them into respective sets in an effort to efficiently optimise the process. When the k-level threshold is set to 3 the algorithm correctly splits data set into clusters containing 3 items. The algorithm also partitions the data into higher and lower integer values within each cluster

{1,2,5} and {10, 9, 6}. The ordering of the values within the higher integer cluster is not in ascending order. Ideally these values would be sorted accordingly to make it easier for the user to read however it appears that this has not been implemented from the test case results.

Animal	Letter
Snake	A
Snake	B
Lizard	C
Cat	D
Dog	E
Dog	F
Dog	G

Figure 6.1.3 – Test Case 2: Input Data Set + Anonymisation Hierarchy

The next test case illustrates the performance of a hierarchy-based anonymisation. This particular anonymisation leverages a simple custom hierarchy that is specified above. A key point to remember is the algorithm has been coded to count the frequency of item appearances, once the k-level threshold has been satisfied that value is discarded and no further processing will occur. Even if similar items are required to be anonymised they will not be grouped with values that have already satisfied the threshold.

Animal	Letter
Snake	A
Snake	B
Animal	C
Animal	D
Dog	E
Dog	F
Dog	G

Animal	Letter
Reptile	A
Reptile	B
Reptile	C
*	D
Dog	E
Dog	F
Dog	G

Animal	Letter
*	A
*	B
*	C
Mammal	D
Mammal	E
Mammal	F
Mammal	G

Figure 6.1.4 – Test Case 2: Output Anonymised Data Set (k=2 vs. k=3 vs. k=4), System Screenshot

From left to right the figure above displays the anonymised data sets that have resulted from the hierarchy-based algorithm with respective k-levels: k=2, k=3 and k=4. Each anonymisation produces very different set of results.
The anonymisation on the left hand side identifies that the values dog and snake both meet the specified criteria k=2 threshold. The algorithm discards these values and is left with a remaining items cat and lizard. Due to the hierarchy both the values converge at the animal level of the data structure, both animals items combined meet the threshold and thus each original value is replaced by the newly formed animal value.
K=3 anonymisation still retains each original dog value because it satisfies the threshold value. However snake by it self no longer meets this criteria, ascending the hierarchy it can be combined with lizard to form 3 objects reptile. This is the expected behaviour and reptile replaces all the

corresponding values in the original dataset. Cat once again does not meet any criteria and thus ascends to the fully anonymised root value of the hierarchy.
The final anonymisation again presents a stark difference in output simply by updating the threshold k=4. The 3 instances of dog must be combined with cat to from mammal that meets the required 4 values. Where as snake and lizard form 3 occurrences which cannot meet any level of k=4 therefore each item is fully anonymised to the root value '*'.
While each k value generates very different output datasets, the processing of each anonymisation is exactly as intended. This would seem to indicate the user has a lot of control over the output dataset. Tweaking the threshold values may produce more favourable results; therefore it is up to the user to ensure the optimum format of anonymised data.

Name	EnrollmentDate	Postcode	Gender	Treatment	NumberOfTreatments
Sarah	12/03/2014	MK7 7SU	F	Drug A	3
Fahed	09/03/2012	MK7 7TR	M	Drug A	6
Hannah	05/09/2013	MK7 YHE	F	Drug C	7
Dave	06/07/2013	BH21 1SY	M	Drug D	9
James	12/03/2014	BH21 1SY	M	Drug B	2
Roy	12/03/2014	BH21 1SY	M	Drug A	2
Connie	02/11/2011	CF24 4AR	F	Drug C	2
Kim	09/01/2012	CF25 4RE	F	Drug A	1
Faye	01/05/2013	MK7 7GH	F	Drug C	7

Figure 6.1.5 – Test Case 3: Input Data Set, System Screenshot

Test case 3 features a typical extract of relational data. This set was created to test the functionality of the tool and also to show case the ability the system has to create highly configurable anonymisations.

Name	EnrollmentDate	Postcode	Gender	Treatment	NumberOfTreatments
*	12/03/2014	MK7 7***	{M, F}	Drug A	{1, 3}
*	{02/11/2011, 09/01/2012, 09/03/2012}	MK7 7***	{M, F}	Drug A	{9, 6}
*	{05/09/2013, 06/07/2013, 01/05/2013}	********	{M, F}	Drug C	7
*	{05/09/2013, 06/07/2013, 01/05/2013}	BH21 1SY	{M, F}	Drug D	{9, 6}
*	12/03/2014	BH21 1SY	{M, F}	Drug B	2
*	12/03/2014	BH21 1SY	{M, F}	Drug A	2
*	{02/11/2011, 09/01/2012, 09/03/2012}	********	{M, F}	Drug C	2
*	{02/11/2011, 09/01/2012, 09/03/2012}	********	{M, F}	Drug A	{1, 3}
*	{05/09/2013, 06/07/2013, 01/05/2013}	MK7 7***	{M, F}	Drug C	7

Figure 6.1.6 – Test Case 3: Output Anonymised Data Set, System Screenshot

The name attribute has been fully anonymised as an explicit identifier therefore all underlying information is removed at every row.
The enrollment date attribute has been anonymised to level k=3 using a set based anonymisation. Defining this attribute as a date data type has successfully resulted in closer dates being grouped together in corresponding sets. The set based anonymisation has also correctly identified the date 12/03/2104 occurs 3 times and therefore shouldn't be anonymised.

Postcode data has been anonymised using an automatically generated string redaction hierarchy. Once again setting k-level threshold to 3 has resulted in a mixture of anonymised values; some items being totally anonymised, others left as their original values. The automatic hierarchy generation performs as intended obscuring a relevant amount of characters for values that should be partially anonymised. This process provides a balance between personal privacy and information retention. Gender has been anonymised into a set that includes both female and male indicators. The mock treatment attribute has not been anonymised; this is a simplified example of data that researchers/professionals examining the data would use to find underlying trends when linked to personal data. Ensuring the rest of the relational data is anonymised sufficiently should mean this type of information would not need to be obfuscated.
The final attribute number of treatments was imported into the tool as an integer data type. Performing a set based anonymisation k=2, it is clear to see the algorithm has grouped the anonymised data sets effectively with close numbers added to the corresponding groups.

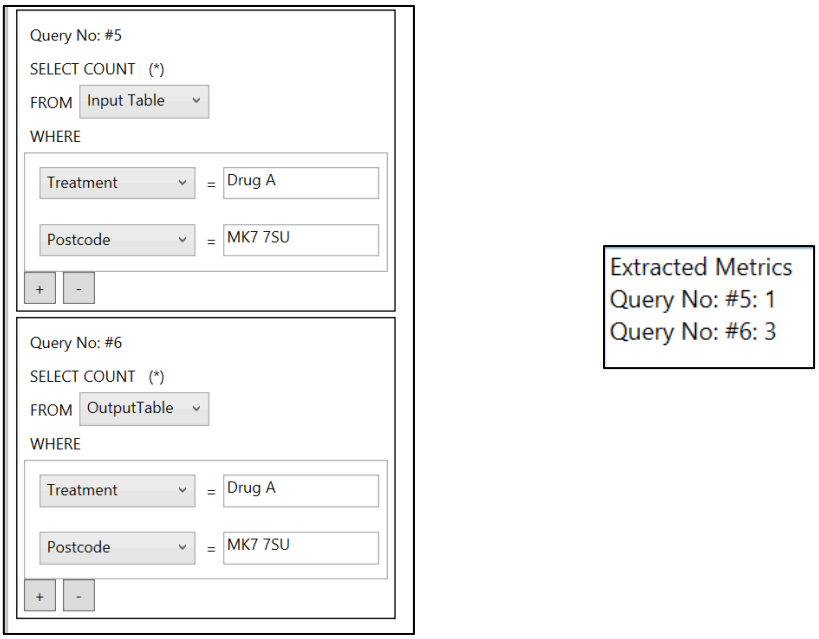


Figure 6.1.7 - Test case 3: Post-Processing, Task Based Metric Evaluation, System Screenshot

The post-processing query example uses the input and output dataset from Fig 6.1.5 & Fig 6.1.6 respectively. This particular test case illustrates that more potential matches may occur within an anonymised dataset due to the information loss that has been enforced to produce k-conformant tables. The initial query correctly identifies that only one possible match is available from the input dataset, the second row Fahed lives within such a postcode and was on treatment from drug A. The secondary query is structured as the first however it is searching through the output dataset. Due to the anonymisation process and resultant information loss 3 potential matches have been identified. Rows 1, 2 and 8 within Fig 6.1.6 could all possibly match as the postcode ‘MK7 7SU’ and each has been subjected to drug A treatment. The post-processing task based metric extractions have performed exactly as expected and would allow users to gain an insight into the retained information within anonymised datasets.

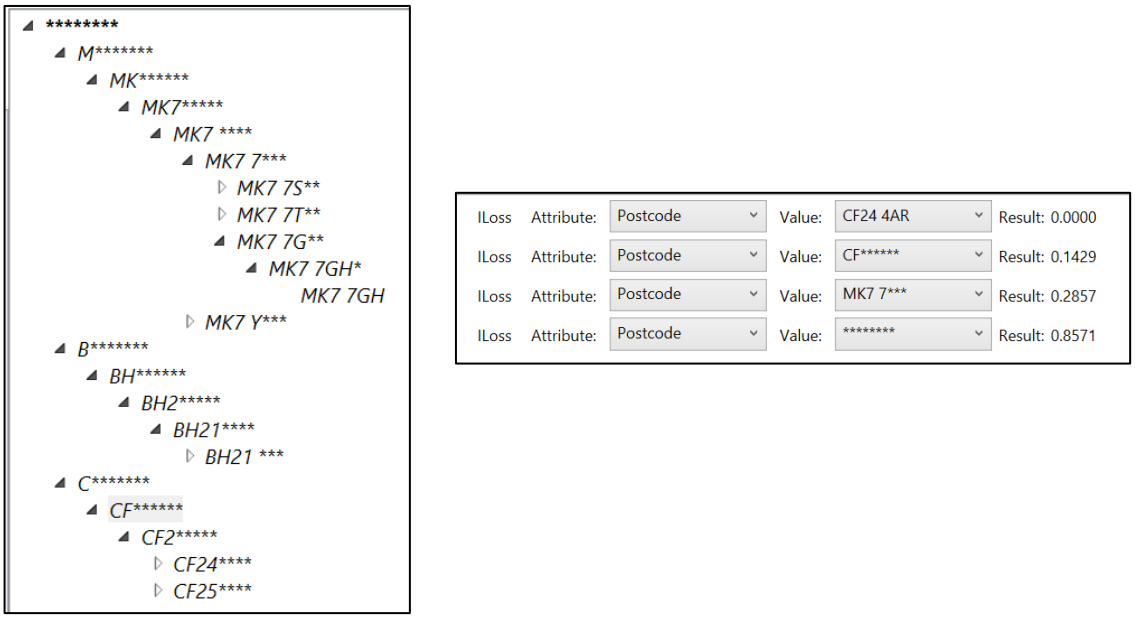


Figure 6.1.8 – Test Case 4: Post-Processing, Data Based Evaluation, System Screenshot

This test case features the tools data based metric extraction. The anonymisation hierarchy depicted above was automatically generated using the applications rule based string redaction hierarchy generator. The resultant structure was used to anonymise the postcode attribute in Fig 6.1.8. The post –processing tool allows users to examine the associated information loss when transforming values to parent nodes in order to achieve k-level conformity. As expected the original leaf node value 'CF24 4AR' produces a result of 0 because no information loss has been incurred. Parent nodes above this value such as 'CF******' incur some loss, however because there are only two possible CF postcode values the information loss isn't as high as a value closer to the leaf node of an MK postcode. At this point it is worth reiterating that these calculations work solely based on parent and leaf values specified in the anonymisation hierarchy. The alternative would be to perform a calculation on possible values from example CF24 4A* -> CF24 4A[A-R] 26 possible alternatives. However given the current functionality the test case produces sensible results that appear to sufficiently reflect the level of information loss imposed on values that maybe utilised by the anonymisation process.

Name	Postcode	DOB	Requests	Department	MaritialStatus
*	D*******	{29/12/2010, 31/12/2010, 10/01/2011}	{90, 89}	Quality Assurance	Single
*	********	{23/09/2010, 25/09/2010, 10/10/2010}	{1, 2, 4}	Customer Relations	Common-Law
*	********	{19/10/2010, 04/11/2010, 04/12/2010}	{38, 40}	Customer Relations	Single
*	M*******	{21/02/2012, 18/02/2012, 14/02/2012}	{41, 42}	Payroll	Single
*	********	{03/03/2011, 08/03/2011, 17/03/2011}	{50, 49, 47}	Finances	Common-Law
*	G*******	{21/02/2012, 18/02/2012, 14/02/2012}	{15, 16}	Research and Development	Single
*	X*******	{21/02/2012, 18/02/2012, 14/02/2012}	{50, 49, 47}	Public Relations	Divorced
*	B*******	{26/04/2012, 22/04/2012, 21/04/2012}	85	Payroll	Married
*	R*******	{25/04/2011, 29/04/2011, 17/05/2011, 19/05/2011}	{57, 53, 51}	Media Relations	Common-Law
*	G*******	{09/01/2012, 27/12/2011, 24/12/2011}	{15, 16}	Research and Development	Married
*	********	{28/04/2010, 05/05/2010, 09/05/2010}	{96, 95, 94}	Finances	Single
*	H*******	{13/01/2012, 11/01/2012, 10/01/2012}	{36, 37, 38}	Finances	Divorced
*	H*******	{21/04/2012, 18/04/2012, 16/04/2012}	100	Legal Department	Common-Law
*	********	{07/03/2012, 27/02/2012}	{31, 33}	Asset Management	Common-Law
*	M*******	{21/04/2012, 18/04/2012, 16/04/2012}	{16, 18, 20}	Human Resources	Married
*	T*******	{10/05/2010, 16/05/2010, 01/06/2010}	{57, 53, 51}	Payroll	Single
*	S*******	{07/03/2012, 27/02/2012}	{81, 80}	Quality Assurance	Common-Law
*	CF24 4AR	{09/01/2012, 27/12/2011, 24/12/2011}	{27, 28, 29}	Customer Service	Single
*	OO5 7***	{25/04/2011, 29/04/2011, 17/05/2011, 19/05/2011}	{9, 10}	Sales and Marketing	Divorced
*	MC17 5Q*	{11/11/2011, 01/11/2011, 24/10/2011}	{43, 46}	Finances	Common-Law
*	QM2 3Y**	{11/06/2011, 09/06/2011, 05/06/2011}	{71, 68, 67}	Research and Development	Single
*	I2 3****	{07/03/2012, 27/02/2012}	19	Asset Management	Single
*	IX3Z 5OB	{12/01/2011, 23/01/2011, 26/01/2011}	{31, 33}	Tech Support	Divorced
*	U97 5MG	{15/04/2012, 12/04/2012, 10/04/2012}	54	Payroll	Married
*	CF24 4AR	{08/09/2011, 23/08/2011, 26/07/2011}	{90, 89}	Public Relations	Married
*	IX3Z 6**	{31/01/2012, 17/01/2012, 14/01/2012}	45	Legal Department	Single
*	XF5T 1M*	{07/06/2010, 24/06/2010}	{7, 8}	Advertising	Single
*	H*******	{30/03/2011, 01/04/2011, 07/04/2011}	{64, 63, 61}	Customer Service	Divorced
*	********	{18/07/2010, 26/07/2010, 20/08/2010}	{96, 95, 94}	Accounting	Single
*	D*******	{15/04/2012, 12/04/2012, 10/04/2012}	{64, 63, 61}	Public Relations	Married
*	W*******	{20/03/2012, 13/03/2012, 10/03/2012}	100	Advertising	Single
*	X*******	{12/01/2011, 23/01/2011, 26/01/2011}	{88, 86, 83}	Quality Assurance	Single
*	********	{25/07/2011, 08/07/2011, 07/07/2011}	{21, 25}	Accounting	Common-Law
*	OO5 7***	{11/11/2011, 01/11/2011, 24/10/2011}	{71, 68, 67}	Public Relations	Divorced
*	MC17 5Q*	{17/12/2010, 26/12/2010}	85	Media Relations	Married
*	QM2 3Y**	{27/02/2011, 28/02/2011, 02/03/2011}	{16, 18, 20}	Sales and Marketing	Single
*	I2 3****	{05/04/2012, 30/03/2012, 24/03/2012}	{34, 36}	Human Resources	Common-Law
*	IX3Z 5OB	{11/06/2011, 09/06/2011, 05/06/2011}	{41, 42}	Payroll	Single
*	U97 5MG	{28/04/2010, 05/05/2010, 09/05/2010}	100	Research and Development	Common-Law
*	CF24 4AR	{30/03/2011, 01/04/2011, 07/04/2011}	19	Payroll	Common-Law
*	IX3Z 6**	{07/09/2010, 12/09/2010, 17/09/2010}	{88, 86, 83}	Customer Relations	Divorced
*	XF5T 1M*	{31/01/2012, 17/01/2012, 14/01/2012}	{64, 63, 61}	Asset Management	Common-Law
*	H*******	{15/04/2012, 12/04/2012, 10/04/2012}	{71, 68, 67}	Payroll	Divorced

Figure 6.1.9 – Test Case 5: Increased Dataset Size, System Screenshot

Test 5 was targeted at stressing the system further than the previous cases. The input dataset had 100 records, far beyond the numbers tested above. The system handled the increased size exceptionally well and the execution time of the anonymisation process was not distinguishable from the smaller sets. For the sake of brevity a figure of the input data has not been shown but will be listed within the appendix. Despite the mixed anonymisation techniques applied to this larger relational dataset the tool appears to still function at an acceptable rate. Although with much larger datasets it may have been appropriate to implement process indicators to inform the user how long the operation is expected to take and the it's current stage of competition.

6.3 Critical Evaluation of Results

Evaluating the performance of hierarchy based anonymisations within the tool showed the process works as designed and intended. However critically assessing the difference in results from Fig 6.1.4 the behaviour to instantly discard values from any further processing once the k-level is satisfied may cause vast contrasts in results depending on the threshold. The advantage of this approach is that a high level of information is retained on values that satisfy k-conformity. However the contrast is that that values that fall short of this are more likely to suffer high information loss because they have to navigate to high levels of the hierarchy in order to combine with other values to meet the criteria. This is depicted in Fig 6.1.4 because values that fall short tend to become fully anonymised. The alternative to this approach is to group values that do not meet the k-level with other values that have already satisfied the criteria. Overall this would mean that some information loss is incurred by more values although the remaining data would be more evenly anonymised. In hindsight perhaps an algorithm that provides more evenly distributed anonymisation across a whole dataset would be favourable to one that contrasts between stark differences of high and low information loss.

Iloss post-processing data-based evaluation produces sensible metrics based on the amount of information loss that is incurred by transforming original values to others anonymised values within the hierarchy. While these metrics make perfect sense within the data anonymisation tool because the user has access to the corresponding hierarchy. The information maybe far less pertinent once it has left the context of the application. Attackers looking to uncover the original values and re-identify individuals would have no prior knowledge of the hierarchy used to anonymise the values. Therefore to external third parties the Iloss calculation maps more appropriately to every possible value that could occur i.e (CF24 4A* -> CF24 4A[A-Z], 26 possibilities) instead of the actual descendants of that node in a hierarchy. On the other hand without programming specific data types such as postcode into the tool the application would not know what could logically fit into such a gaps. The domain of potential values that could take the place of anonymised characters would have to be defined every time such a calculation was performed. As such defining behaviour for the tool to understand that the anonymised character within the redacted postcode ‘CF24 4A*’ could only be a letter would be a very complex problem. The system could not include every single possible character in this space as many would produce wildly invalid results e.g ‘CF24 4A$’. Without such mechanisms the application would require a huge amount of work to add tailored data types. Unless this behaviour could be incorporated into the tool it may actually be fundamentally incorrect to base metric calculations upon possible descendants. If the system doesn’t understand that the redacted postcode could only have possible characters missing then calculating a metric that uses characters, numbers and symbols as possible descendants would generate a metric that is simply wrong and does not reflect the true information loss. On balance I believe the way these post-processing evaluations have been implemented is correct based on the project length and time constraints. Although I can see the appeal and benefits of a system identifies possible decedent matches, this is something I have specified later to be discussed within section 7; Future Work.

7 Future Work

7.1 Plugin Algorithm Repository

The most novel feature of the tool is the ability to inject and leverage custom anonymisation algorithms. The plugin architecture that provides this functionality is an original concept and something I have not seen in similar targeted applications. These tools appear to force users to leverage built in algorithms or refactor the algorithms and then rebuild the whole system from source code. Obviously the ability to plugin custom algorithms (even while the system is running) gives the tool far more flexibility and transforms the product from an isolated anonymisation application into a platform for algorithm development. An interesting addition to the system would be the inclusion of an algorithm repository. Users could upload their developed anonymisation techniques to a centralised store. Users could also make their algorithms available to other users of the application via this repository. Additional features such as user ratings and comments would make a plugin store very attractive to potential users. Similar in concept to many programming IDE plugin managers that allow developers to download and use third party functionality. Given sufficient community and demand, users developing algorithms could look at charging a fee to access certain anonymisation plugins. This would be an interesting approach to revenue generation, especially if the eventual system source code was open sourced at the end of the project.
While this feature would not add more functionality to the current system, it would open the possibilities for users of the system and is a natural extension to the plugin architecture.

7.2 Import/Export Custom Hierarchies

The inclusion of functionality to import/export custom created hierarchies is a feature that would require careful thought before implementation. The obvious benefits of saving time, sharing and reusing previously built hierarchies are great reasons to implement import/export functionality. The addition of hierarchy import/export would add a substantial level of complexity to the system. Defining behaviour to handle previously built hierarchies that no longer contain an identical input set of data is just one example of many problematical scenarios. The list of unique values would not be the same this would mean some leaf nodes may not be present in the anonymisation tree. Options that adapt the tree to accommodate values that are not present could be implemented, but then the application is changing the hierarchy and therefore may not apply the anonymisation in the users intended way. Alternatively a hierarchy-based anonymisation cannot be applied if a required leaf value is not featured in the anonymisation hierarchy. A combination of these difficulties and time constraints mean that I have not been able to implement this particular feature as the system stands. However the addition of this functionality would be a natural evolution of the tool and should rank among the highest priorities for future work.

7.3 ILoss Calculation Extensions

The inclusion of ILoss calculations allows user to extract information loss metrics from anonymisation hierarchies. This post-processing operation gives users insight into the level of information utility that is retained by values in comparison to other items in the hierarchy. However as previously discussed in section 5.8 Post Processing Tools an important interpretation has been made when calculating these metrics.
As the system has been implemented ILoss is calculated as the number of leaf descendants from the specified node minus 1 divided by the total number of leaf node descendants for the attribute. This assumes that only values within the hierarchy are used to define the possible leaf descendants. For example Fig 3.2 (Algorithm Design section) depicts that the value MK7 7S* has 2 leaf values; therefore the current ILoss method would calculate the descendant number of level nodes as 2.

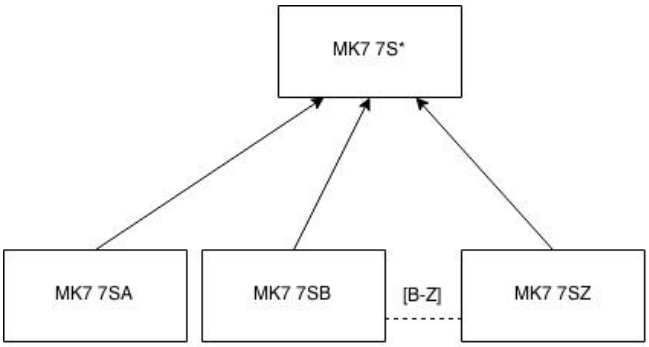


Figure 7.3.1 - Postcode Anonymisation Hierarchy Possible Descendants

However another technique (depicted in Fig 7.3.1) is to assume no prior knowledge of the structure, therefore the node MK7 7S* could have 26 possibilities MK7 7S[A-Z]. The node would then have 26 descendant leaf nodes. Originally this method was not implemented because the system calculates information loss metrics based on the known hierarchy, therefore I interpreted that the actual leaf descendants should be used instead of the full range of possible values.

An extension to the post-processing tool would allow users to switch between these two methods of defining how leaf node descendants are generated. Then it would become the users discretion that influences how the calculation is implemented, instead of forcing them to work upon my personal interpretation. This project objective would not expand the functionality of the system, however it would give the user more control to configure the tool. The ability to easily configure the tool would be a major benefit of the system and ultimately means that as new techniques and best practice are established within the field, users can tweak configuration to reflect these changes.

7.4 Graphing of information retention metrics

The system features a series of post processing tools. They allow the user to extract a series of metrics pertaining to the level of information loss that has been incurred by the anonymisation process. An appropriate extension to this set of tools is a graphing suite that visually depicts the extracted metrics. These graphs would allow users to group and display information from a number of different anonymisation sources in one easily readable format. Users would be able to clearly compare how various anonymisations have impacted the original data set and hopefully allow them to draw effective conclusions when developing and using anonymisation algorithms. Visually displaying these metrics would be far more efficient compared to the current method of printing calculated metrics on screen for the user to examine individually. The graphing suite could leverage line and bar graphs to depict this information. Graphing metrics was discussed within the initial report, however due to time constraints it has not been included in the initial solution.
While disappointing that this functionality does not currently feature within the tool the infrastructure to calculate the metrics has been implemented. Adding graphing functionality would simply be an extension to visualise these values.

8 Conclusions

The overarching objective of the project was to build a data anonymisation tool that featured a pre-existing k-anonymisation algorithm to help protect relational datasets. Other sub-objectives were derived from this project goal. Inclusion of pre/post-processing tools, use of existing k-anonymisation techniques, the ability to cope with a wide range of data types and an extensible project structure were all conceived from the original notion of a data anonymisation tool.

The resultant solution is an anonymisation tool that can be used to generate k-anonymity conformant datasets. The tool currently supports a default set and hierarchy based anonymisation in addition to a plugin architecture that allows users to run third party algorithms. The plugin functionality is perhaps the most innovate feature of the project and elevates the solution from a stand-alone tool into an anonymisation platform that supports algorithm design and development. While conducting initial research I discovered a few tools with similar anonymisation functionality such as ARX – Data Anonymisation Tool [14]. However these tools featured one anonymisation method that was hard coded into the system. This would be sufficient for general users that simply want to anonymise relational datasets. However as best practice and new techniques emerge the user is stuck using the same anonymisations until the system is re-built. Although there is relatively little support for super users who may want to tweak their own custom algorithms, the inclusion of a plugin architecture within the tool allows researchers/professionals to work in such a way that will hopefully accelerate the discovery and implementation of information security best practice.

Pre-processing tools have been created that allow users to generate highly configurable and tailored anonymised datasets. Custom hierarchies are a prime example, they facilitates configurable anonymisations built exactly to the user specifications. While post-processing tools extract data metrics that can illustrate information loss incurred by anonymisation. Metrics will be utilised by users to quantify the quality of results. The inclusion of these tools was vital to ensure the resultant solution could be effectively used to configure anonymisations and then to evaluate output quality.

The objective to include a wide range of data types within the anonymisation tool was not fully achieved. In hindsight this was an ambitious objective, the implementation of custom data types would have required a substantial amount of implementation time for each individual type. Instead an attitude to handle varying data as a series of generic types appeared to be the most logical approach. While string, integer, double and date generic data types have all been implemented in the future more specialised data types would be a logical step forward for example a specific postcode type. Unfortunately this aspect of implementation never came to fruition due to added complexity layers to the tool and the associated time constraints.

On the other hand the objective to create an extensible tool suitable for future development has been a resounding success. The plugin architecture gives anyone with access to the software platform the ability to perform immediate updates with the inclusion of third party custom algorithms. In addition the structure of the system has been meticulously designed and implemented which has resulted in a modular solution. Thanks to this modular structure coupled with an MVVM architecture individual components can be easily refactored or even completely replaced without needing to alter large sections of the system to achieve such updates.

The output data sets such as the one produced in Fig 6.1.6 are k-conformant and provide the necessary levels of personal protection against record linkage attacks and subsequent re-identification. The tool facilities a range of ways that k-anonymity can be enforced, set based, rule-based (string redaction) and custom hierarchies. To this end I believe the project has been very

successful delivering a tool that supersedes some of the required functionality. In particular I feel it is a strong base for future development, as discussed with my supervisor the school of Computer Science and Informatics has plans to develop further anonymisation projects to model attacks on relational data. This tool would be an ideal candidate for inclusion as a component of a larger information security project. Alternatively the tool is suitable for use by one of the many organisations that want to ensure they are anonymising data sufficiently for public release.

Throughout this project I have learnt and applied various techniques in the field of privacy persevering data publishing. Having no prior experience with k-anonymity there was a steep learning curve at the design stage of the project, an acknowledgement must go to my supervisor for working through numerous examples helping me to understand the underpinning theory. One thing that has become apparent to myself throughout this project is that despite the best anonymisation tools in the world, the safety of personal privacy must be ensured by knowledgeable humans with careful forethought. Anonymising a series of relational attributes is of no useful protection if the person applying the anonymisation has misunderstood a fundamental piece of information that could result in the exposure and re-identification of individuals from that dataset. Information security is constantly evolving field and individuals within it should be prepared to change as best practice and new threats emerge within the sector. This project has created an anonymisation tool that can safely protect individual privacy but it must be used correctly with caution urged every step of the way before publically releasing such data. While continued research and support for increasingly clever and secure tools should be a priority for the technology industry, an equal if not greater emphasis should be placed on the teaching and training of appropriate individuals to better understand the fundamental concepts of privacy persevering data publishing.
I believe certain organisations such as the NHS are indeed doing such a task, through giving their member's information governance training [15] in order to sufficiently prepare them to handle personal data in the modern world. The policy places emphasis on holding data securely while maintaining confidentially in addition to sharing/disclosing data lawfully and appropriately. Organisations have the responsibility to define and implement clear and effective policies to handle and release personal information. Educational institutions should also be equipping the next generation of workers to protect personal data.

The project has indeed attempted to tackle each of these objectives and for the most part it would be fair to say has fulfilled the concept idea underpinning its inception. Some particular areas such as custom hierarchy generation have exceeded my initial expectations where as others have not quite seen the same progress i.e graphing of resultant data metrics. However the final solution has produced a tool that can theoretically leverage a huge number of varying k-anonymisation algorithms. A large amount of design and development has gone into the tool and I would be hard pressed to implement any more features due to time constraints. The features that have been implemented work well and I believe the whole application has a general smooth and intuitive feel that will allow users to quickly generate anonymised datasets with a small amount of application familiarisation.

9 Reflection on Learning

9.1 Project Management

This process has resulted in the largest academic project I have ever produced. A good approach and style of management has been of paramount importance at every stage of the project. Planning work items, organising supervisor meetings in addition to effectively communicating problems and solutions have been required throughout the process. Without a sustainable style of project management I would not have been able to successfully implement or document the creation of the data anonymisation tool.

Working on a project of such magnitude was an overwhelming prospect initially. I felt it would be hard to keep track of the tasks that had been completed and what was still required. This was especially prevalent at the start of the project as I attempted to design/implement and document the project simultaneously. Quickly I felt this was not an appropriate approach and altered my project management technique accordingly. To alleviate this situation I began to allocate blocks of time to specific tasks e.g 2 days to program functionality x. This allowed my approach to become more focused on smaller targeted objectives. Once this had changed I began to feel far more in control of the project and actually enjoyed the freedom I allowed myself to work with on different aspects of the project.

Evaluating my management I can identify that by the end the project I was able to effectively control, track and allocate work to myself. This was all done while maintaining a positive relationship with my supervisor, arranging meetings, demonstrating my latest work and communicating other issues that surrounded design or implementation of the tool. However while I was able to develop personal work management skills there has been no scope to expand such skills in a group context. However this has been an individual project so intrinsically there have been no opportunities for group development. Although the personal management skills I have developed can be utilised in future projects where I will have to work strictly towards tight deadlines.

The change in approach that occurred near the beginning of the project that meant I stopped attempting to design/implement and document the tool simultaneously. I believe I can multi-task well however this was not sustainable and in hindsight an unrealistic approach. An important aspect of project management throughout this process was not to underestimate the time it would take to complete tasks.

Hofstadter’s law [16] states, “It always takes longer than you expect, even when you take into account Hofstadter's Law”. Reflecting on this statement I can see that when planning and managing tasks it is important to assign ample time for completion even if the task has been identified as fairly non-intensive. Looking back to my initial report project plan, I can easily identify some areas that needed more time to be completed effectively. Many of the tasks I estimated would take a significant time to implement still took even longer than initially anticipated. In future projects I will remember this adage as a guide to ensure that I do not allocate myself too many tasks with not enough time to successfully implement them.

In the final stages of the project I began to use a whiteboard to note down activities for the week. This included objectives that needed to be implemented, work items in progress and tasks that had been completed. I found this approach very effective as it immediately visualised my progress. Given its success I should have adopted this approach sooner in the project.

In the future projects I will attempt to manage my expectations of how much work can be successfully implemented with given time constraints. I will also attempt to visualise my progress in some manner, this could take the form of the previously discussed whiteboard work items technique or even something more elaborate such as Trello [17] which functions as a smart notice board with specific tools designed to track and visualise work items. Another important piece of knowledge I have gained from this process is that I work most effectively when focusing on specific targeted tasks. After initially attempting to design/implement and document the tool I found that I was more effective when applying myself to a singular task with a smaller objective. In my future career I will remember to give myself sufficient time per task and rotate my schedule to achieve them, instead of attempting to perform all my tasks simultaneously.

9.2 Understanding of Personal Privacy Issues

This project has been centralised around the implementation of a privacy protection tool. Having little prior experience and understanding of these issues I was required to constantly learn throughout the design and development stages. While having a small amount of prior experience within data protection, I had never studied or implemented anything related to k-anonymity. Throughout the project I have been furthering my understanding of these issues and in particular record linkage attacks that can mounted to re-identify individuals within relational datasets.

Previously I have not been required to read, understand and apply large quantities of research-based information. Due to this I found the initial research stages of the project fairly demanding, although in hindsight this was exacerbated by a fairly inexperienced knowledge of personal privacy issues. As the project has progressed I have felt more comfortable with my understanding of these issues and the solutions I have produced to solve the associated problems.

Implementation of the tool has significantly benefited my personal knowledge and understanding. To this extent the project has been a positive learning experience. On the other hand while a positive learning experience this has shown me at times the relatively insecure privacy methods that organisations (some of which hold personal data about myself) use to insufficiently protect their data. In the future I will be far more aware regarding the personal information organisations store about myself.

Reflecting on the process of understanding and applying research level content I would spend longer ensuring I had substantial core knowledge of the underlying concepts. After developing an understanding of the high level issues I would be far more equipped to under the research level detailed information. I believe my approach to applying the gained knowledge was suitable and actually very effective. In the future it would be vitally important to gain an understanding of how to apply the research level content. However once the process had been completed I would approach any future iterations of work that require a significant research component in the same manner.

9.3 Plugin Algorithm Architecture

From the initial research stage of the project several possible system architectures were conceived that aimed to make the tool extensible while providing easy access to implemented algorithms. Eventually the final decision was made to implement a plugin algorithm architecture because it fulfilled the project aim to provide a simple extensible platform in addition to supporting more than the initial hard coded default anonymisations. While conducting research on tools that offered similar functionality I could not find any projects that supported such a way of allowing users to leverage third party algorithms. The significant benefits associated with this approach meant that the plugin solution was favoured to other possible architectures.

Having never implemented a similar system before witnessing each step of plugin functionality being added to the tool was an exciting process. Reflecting on this aspect of the project I felt the implementation of a working plugin system was one of my greatest achievements that demonstrates a well designed and executed final solution. Although during the initial design concepts I felt this particular goal may have been too ambitious because it took me significantly longer than expected to research, plan and build a working prototype that exhibited this type of behaviour. However once this prototype snippet of functionality was operating I recognised that the objective could easily be incorporated into the resultant solution.

The main advantage to the architecture is that future developers can build and leverage new anonymisations by dynamically plugging in their code (without even having to restart the application). This transforms the system from a tool that features a couple of hard coded anonymisations, into a platform for algorithm research and development. Overall this feature adds significant functionality, while the design and implementation process although at times was difficult, it has actually been a hugely positive experience.
On the other hand by implementing a plugin architecture I have made a series of assumptions that eventually may constrict the behaviour of the system. One such example is that a plugin algorithm assumes one dataset as input and one anonymised dataset as output, this assumption has been fine for the length of the project however other anonymisations such as k-anonymity via anatomy may utilise multiple tables. Obviously the restriction of the plugin algorithm to use one data table cannot sufficiently accommodate this process. Attempting to design a solution that could solve these issues was an extremely difficult problem and was certainly a very demanding aspect of the project.

In hindsight the innovative design and implementation of an anonymisation plugin architecture has been a rewarding learning experience. Having no previous experience there was a large portion of research and prototype development that accompanied this objective. The resultant solution operates exactly as intended and does indeed provide an easy flexible way for future developers to extend the tool with their own custom functionality. As a result of this implementation I have built a comprehensive base of knowledge related to algorithm plugin structure. I have no doubt the skills I have gained will benefit me far beyond this project; I would now confidently suggest this approach as a possible structure for future solutions. I have also built sufficient knowledge to recognise in which situations this implementation would be appropriate.

Reflecting on this objective I should have spent more time developing a system that could cope with a wide range of plugins without enforcing strict restrictions such as 1 relational dataset in and 1 relational anonymised dataset as output. On the other hand due to the nature of plugin implementations some form of commonality (i.e assumptions of behaviour) must be applied.

Given the knowledge I have gained from this objective I would approach this problem in a slightly different manner. In the future I will aim to spend slightly more time designing and thinking about the range of possibilities I would like a plugin system to cope with. Once I have settled on a list of requirements I would then move to develop an initial prototype proof of concept. Spending more time at this design stage would hopefully lead to the most flexible plugin system possible within future projects.

10 Appendices

i. Data Anon Tool Complete Project Code Listing
ii. Test Cases Input & Output
    a. TestCase1Input.csv (Fig 6.1.1)
    b. TestCase1OutputK2.csv (Fig 6.1.2, k=2)
    c. TestCase1OutputK3.csv (Fig 6.1.2, k=3)
    d. TestCase2Input.csv (Fig 6.1.3)
    e. TestCase2OutputK2.csv (Fig 6.1.4, k=2)
    f. TestCase2OutputK3.csv (Fig 6.1.4, k=3)
    g. TestCase2OutputK4.csv (Fig 6.1.4, k=4)
    h. TestCase3InputData.csv (Fig 6.1.5)
    i. TestCase3OutputData.csv (Fig 6.1.6)
    j. TestCase5InputData.csv
    k. TestCase5OutputData.csv (Fig 6.1.9)

11 References

[1] UK Government. About Data.Gov. Available: http://data.gov.uk/about. Last accessed 19th April 2015

[2] Sweeny L. (2002). K-ANONYMITY: A Model for Protecting Privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems. 10 (5), p2.

[3] B. C. M. Fung. (2010). Privacy-Preserving Data Publishing: A Survey of Recent Developments. ACM Computing Surveys. 42 (4), p7

[4] D. C. Barth-Jones. (2012). The "Re-identification" of Governor William Weld's Medical Information: A Critical Re-examination of Health Data Identification Risks and Privacy Protections, Then and Now. Analysis and Commentary: "Re-identification" of Governor William Weld. 1 (1), p.1.

[5] Ji-Won Byun. (2007). Efficient k-Anonymization Using Clustering Techniques. DASFAA, LNCS. 4443 , p188.

[6] B. C. M. Fung. (2010). Privacy-Preserving Data Publishing: A Survey of Recent Developments. ACM Computing Surveys. 42 (4), p22.

[7] A. Kawano. (2013). A Greedy Algorithm for k-Member Co-clustering and Its Applicability to Collaborative Filtering. Procedia Computer Science. 22 (477-484), 480.

[8] Microsoft. Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF. Available: https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx. Last accessed 26th Feb 2015.

[9] Windows Dev Center. Dynamic-Link Libraries. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx. Last accessed 29th Mar 2015.

[10] Microsoft. NuGet Gallery, What is NuGet? . Available: https://www.nuget.org. Last accessed 17th April 2015.

[11] John Close. (2009). CsvHelper. Available: https://github.com/JoshClose/CsvHelper. Last accessed 30th Mar 2015.

[12] Microsoft. Generics (C# Programming Guide). Available: https://msdn.microsoft.com/en-us/library/512aeb7t.aspx. Last accessed 12th April 2015.

[13] Dot Net Perls. Levenshtein Distance Computations. Available: http://www.dotnetperls.com/levenshtein. Last accessed 13th April 2015

[14] ARX – Data Anonymisation Tool . Available: http://arx.deidentifier.org/. Last accessed 30th Jan 2015.

[15] NHS. (2014). Information Governance Policy . Available: http://www.england.nhs.uk/wp-content/uploads/2013/06/ig-policy-1.1.pdf. Last accessed 22nd April 2015.

[16] D. R. Hofstadter. (1999). Gödel, Escher, Bach: An Eternal Golden Braid. 20th anniversary ed. New York: Basic Books, Inc. p.152.

[17] Trello. Available: https://trello.com. Last accessed 25th April 2015.