Measuring Software Engineering

Software Engineering – CS3012



Conor Corry – MSISS – 17336016

# Measuring Software Engineering

# Abstract

This report will outline the ways in which the software engineering process can be measured and assessed. In particular, it aims to address this subject in terms of measurable data, the computational platforms available, the algorithmic approaches available and the ethics of such analytics. I will draw together my knowledge from the set of readings associated with this topic as well as the content discussed in lectures.

# Introduction

In order to consider the ways in which the software engineering process can be measured and assessed we must first understand what it meant by the term ‘software engineer’. Software engineering is defined as the process of analysing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages.

The beginning of software engineering dates back to 1968, when the term “software engineer” was first referred to at a NATO conference, on the topic of the, then current software crisis. The software crisis was the name given to the difficulties encountered in developing large, complex systems in the 1960s. It was proposed that the adoption of an engineering approach to software development would reduce the costs of software development and lead to more reliable software. It was this conference that the idea that a systematic approach, similar to that of physical engineering, should be applied to the development of software. Today, the field of software engineering continues to progress at an immense rate, and as stated by Ian Sommerville, is “critically important technology for the future of mankind.”

Since its first referral in 1968, the process of software engineering has seen a vast development. One may ask the question of, ‘Where is Software Engineering applicable in my day to day proceedings?’ The simple answer is that it determines how we interact with technology on a daily basis and in essence software engineering has helped to form the world surrounding us today.

Throughout this report I will examine each of the following topics:

* Measurable data
* Computational platforms available to perform this work
* Algorithmic approaches available
* Ethical concerns surround this kind of analytics

# Measurable Data

**Brief history of data collection**

In the early years of software engineering the major issue that developing companies faced was data collection and also the subject of collecting the correct data. Data collection was frequently carried out manually, which can be a very time-consuming endeavour. The tedious element of manual collection led to several different problems, such as **Outdated Information.** The manual data collection process takes time. That means the data will likely not be available until the next day, and it this point, it may be outdated. This delay makes it impossible to catch and fix issues while they occur. Often times, the problem does not become apparent until it’s too late to fix. This impacts the efficiency and validity of the data. Other issues such as human error and uniformed decisions can damage the validity of software engineers’ manual collected data.

The advancements of technology have allowed us to move away from the manual collection of data. Now, fortunately there is a way to improve efficiency and accuracy, thanks to automated data collection. Automated data collection provides the tools needed to eliminate all of the previous problems and enhance the overall efficiency of accumulated data.

**Data collection**

The quality of any measurement program is clearly dependent on careful data collection. Data collected can be distilled into simple charts and graphs so that the managers can understand the progress and problem of the development. Data collection is also essential for scientific investigation of relationships and trends.

Gathering software engineering data can be expensive. Large projects within organisations will spend considerable money on development costs, which involves gathering and processing data on hundreds of metrics for a number of projects. However, the cost of data collection will never be insignificant. Nonetheless, data collection and analysis, which yields intelligence about the project and the development process, is vital for business success.  Indeed, in many organizations, a tracking and data collection system is often an integral part of the software configuration or the project management system, without which the chance of success of large and complex projects will be reduced

**Why do we collect data?**

Data collection enables you to improve your understanding of who your audience is and disseminate that information throughout your organisation. It is through data collection that a business or management has the quality information they need to make informed decisions from further analysis, study, and research. Data collection instead allows them to stay on top of trends, provide answers to problems, and analyse new insights to great effect.

The existence of agile methods has brought considerable emphasis on the human element of software development. As a result, software engineers are very focused on the human effort required to develop and maintain systems. By asserting attention to this human effort involved in gathering data, it allows improvements in the software development process.

**Collecting software engineering data**

Collecting data about the software engineering process is a difficult development. Today, organisations are still pushing to find the balance between collecting data that will reward business success and efficiency, while also trying not to leave development teams investing the majority of their time in the data collection procedure. Today’s development team’s may be difficult to assess and therefore complications exist in deciding which metrics are best suited for measuring the software engineer. We must ask the question of what data are we using to study their work.

I have chosen to evaluate the productivity of development teams by looking at three popular Software engineering metrics: agile metrics, code churn and lines of code.

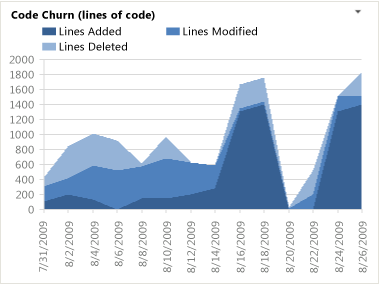
**Agile Metrics**

Agile metrics are a crucial part of an agile software development process. They help software teams monitor productivity across workflow stages, access software quality, as well as introduce more clarity to the development process. Agile metrics include lead time, cycle time and velocity. Lead time measures the total time from the moment a story enters the system (in the backlog), until it is completed as part of a sprint, or released to customers. It is obvious that management will endeavour to minimise lead time in projects, and by having a shorter lead time it allows reduced expenses that may have been endured with a longer project, as well as gaining a competitive advantage in the market by being able to outpace competitors.

**Code Churn**

There are many ways of protecting the quality of the software and devising various techniques to ensure the right test plan for each software release. Code churn-based analysis and testing are very effective in uncovering issues and protecting software quality. Code churn is nothing but code changes made for the software to achieve a certain functionality. It is the percentage of a software engineer’s own code that represents an edit to their own recent work. It’s typically measured in Lines of Code (LOC) that were modified, added and deleted over a short period of time. Spikes in code churn can help identify problems early on in the lifetime of a project, as it describes how much time a developer spends on a feature while making limited progress. Several factors could lead to high code churn in your project, two of the more common are:

* Prototyping
* Allows the developer to explore more ideas and test multiple solutions. The code is intended only to learn something new and then be thrown away, this is called a code spike. Code spike leads to an increase in code churn levels.
* Indecisive developers
* In any workplace several different solutions are available to complete the task in hand. By not committing to one solution, many branches of options exist, and levels of code tallies amongst all the approaches of arriving at the solution, resulting in high code churn.



(Source: Microsoft – <https://docs.microsoft.com/en-us/azure/devops/report/excel/code-churn-excel-report?view=tfs-2017>)

**Lines of code**

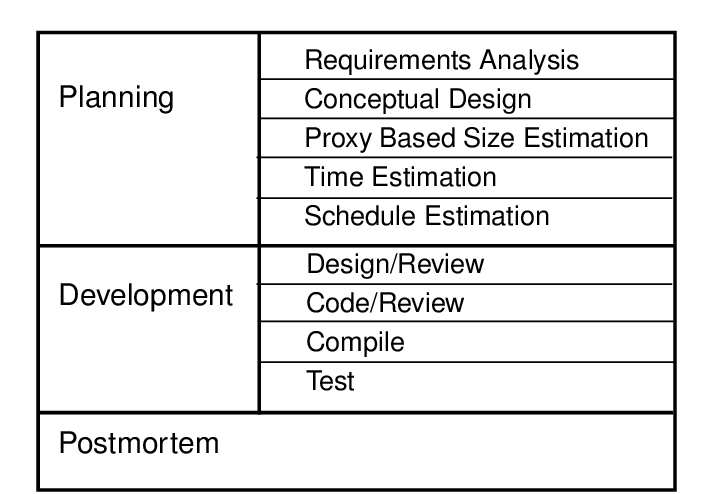
Measuring lines of code is one way in which programmers can be universally compared. However, this procedure for measurement has too many blemishes, in essence it’s too simplistic of a measure. Drawing comparisons between engineers having more lines of code than another and therefore being more productive, is a very simplistic assessment of one’s programming skills, and draws the question, ‘When, if ever is the ‘number of lines’ a useful metric?’ The ever-increasing diversity of programming languages also contributes to the problem of increasing lines of code, as high-level languages’ complexity and functionality is immeasurable to that of assembly language code.

# Computational platforms available

After evaluating the various different metrics and approaches to measuring data collected, we will look at the computational platforms available to the user to both collect and analyse data. There are several platforms that exist, each with different built in functionalities to suit different metrics. In this section I will explore the multitude of ways in which data can be collected and analysed.

**Personal Software Process**

The Personal Software Process (PSP) is a structured software development process that is designed to help software engineers better understand and improve their performance by bringing discipline to the way they develop software and tracking their predicted and actual development of the code. Watts Humphry wrote a book in 1990’s in relation to PSP. In his book, he believed that a structured development process as well as constant tracking of progress would allow the engineer in question to streamline their process, making it more efficient.



(Source: ResearchGate - <https://www.researchgate.net/figure/The-Personal-Software-Process_fig4_220542225>)

PSP is constructed on the idea that plans should be based on the engineers own personal data as every engineer is different and each should feel responsible for the output of their product quality. PSP’S adopts the concept of manual input, where the engineer fills out a variety of forms which require personal judgement. This has the obvious disadvantage of high potential for human error.

**GitHub**

GitHub is a repository hosting service for Git that also has a web-based graphical interface. It lets you and others work together on projects from anywhere and tracks developers commits on a project. GitHub lets you see how frequently someone commits and how many commits a developer adds, as well as how many lines they commit. GitHub can be divided into the Git and the Hub. The Git implies the version control system; a tool which allows developers to keep track of the constant revisions to their code. The Hub is the community of like-minded individuals who participate. The major advantage of GitHub is that it’s all about the collaborative effort of the community, in reviewing, improving, and deriving new ideas from the uploaded code.

**GitPrime**

GitPrime is an organizational tool, pioneering a different way of measuring and communicating about productivity in software engineering. GitPrime aggregates historical git data into easy to understand insights and reports, to help make engineering teams more successful. It collects data from git repos, ticketing systems and pull requests and presents this data as reports. Rather than relying on subjective metrics, GitPrime mines data in Git to measure productivity and collaboration.

**Leap**

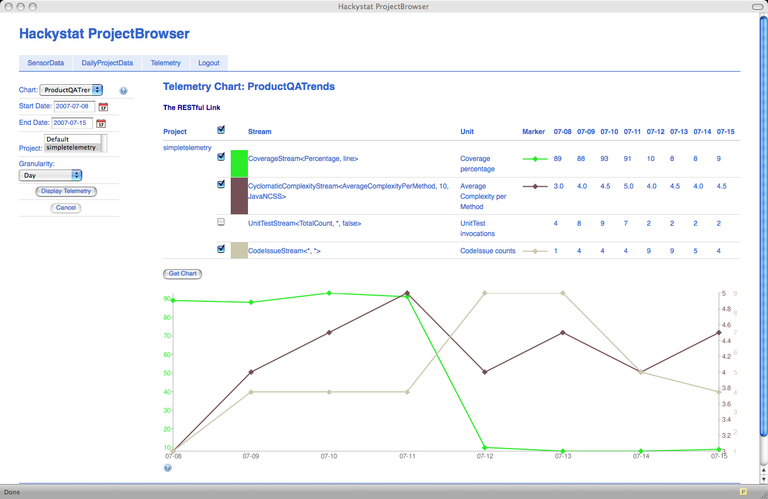
Leap toolkit is aimed to deal with the data quality issues presented by PSP. LEAP creates a portable repository of personal process data that developers can bring with them as they move from projects to projects. It performs many forms of analysis on the dataset automatically and can then be used to calculate the size of the product, defects found, time estimations along with checklists and more. This technique still requires sufficient manual input meaning that it’s prone to the element of human error.

**Code Climate**

Code climate is a web-hosted software which provides organisations and corporations with the ability to take charge of their code quality by incorporating fully configurable test coverage as well as data maintainability throughout the workflow development. Code climate provides an open and extensible platform for ensuring code health. The software allows the user to manage large organisations using the teams feature. Code climate also makes it easier for programmers to identify problems with their coding solutions, by identifying subtle issues which at first wouldn’t be obvious. It helps developers trap issues by analysing each pull request before it is integrated.

**Hackystat**

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data.  Hackystat users typically attach software ‘sensors’ to their development tools, which unobtrusively collect and send “raw” data about development to a web service, this essentially eradicates the overhead of metrics collection. With Hackystat collecting data without the user’s knowledge, a dispute was raised from developers surrounding the privacy of work, as they didn’t want their data being obtained without them being aware of it. However, the minute by minute collection of data is beneficial, as it allows managers to gain deep understanding of work being completed.



(Source: CSDL - <http://csdl.ics.hawaii.edu/research/hackystat/>)

# Algorithmic approaches available

After evaluation of the measurable data and computational platforms used in determining the software engineer, I will now explore the algorithmic approaches available in the world of software engineering.

There are numerous algorithmic approaches to analysing data. The technological advancements in software engineering has led to the increase in the use of machine learning techniques, introducing the concept of computer intelligence and the applications that evolved as a result, such as artificial intelligence (AI). In this section I will discuss computational intelligence and investigate machine learning approach to data analysis, including the supervised and unsupervised methods.

**Computational Intelligence**

Computational Intelligence (CI) is the theory, design, application and development of biologically and linguistically motivated computational paradigms. Many modern algorithmic approaches to data measurement are focused on computer intelligence and traditionally, the three main pillars of CI have been [Neural Networks](https://cis.ieee.org/about/what-is-ci/22-publications/tnnls), [Fuzzy Systems](https://cis.ieee.org/publications/t-fuzzy-systems) and [Evolutionary Computation](https://cis.ieee.org/about/what-is-ci/23-publications/tevc).

Neural networks

Neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input.

Fuzzy systems

A fuzzy control system is a [control system](https://en.wikipedia.org/wiki/Control_system) based on [fuzzy logic](https://en.wikipedia.org/wiki/Fuzzy_logic) —a [mathematical](https://en.wikipedia.org/wiki/Mathematics) system that analyses [analogue](https://en.wikipedia.org/wiki/Analog_signal) input values in terms of [logical](https://en.wikipedia.org/wiki/Mathematical_logic) variables that take on continuous values between 0 and 1. Fuzzy logic is measurements and process modelling made for real world problems, dealing with incompleteness. For this reason, fuzzy systems have a wide range of applications such as decision making.

Evolutionary Computation

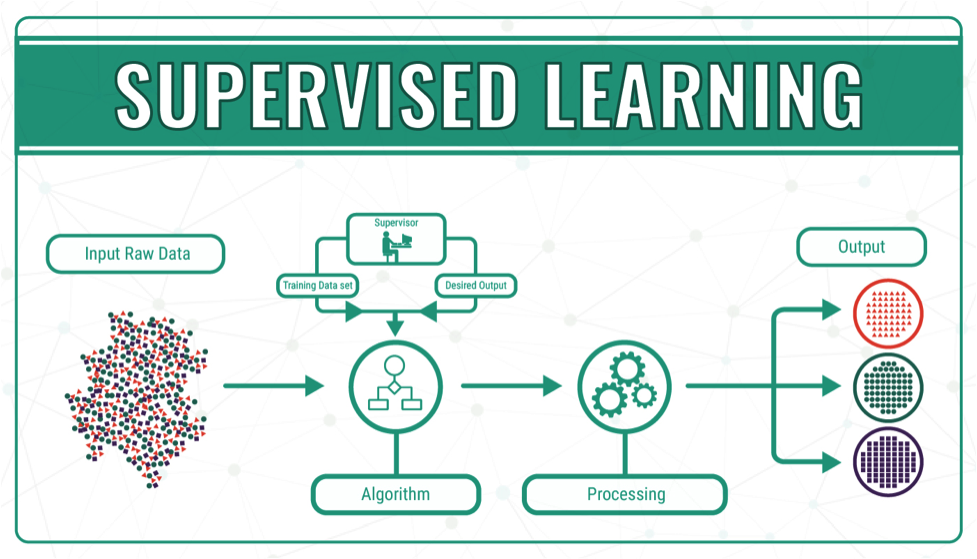
In evolutionary computation, the process of natural evolution is used as a role model for a strategy for finding optimal or near-optimal solutions for a given problem. It generates, evaluates and modifies a series of potential solutions and uses these processes to overcome optimisation problems that were not solvable using a mathematical approach.

**Machine Learning**

Machine learning is a method of data analysis that automates analytical model building and it divides into two subsections: supervised and unsupervised.

Supervised Learning

Supervised learning is the [machine learning](https://en.wikipedia.org/wiki/Machine_learning) task of learning a function that maps an input to an output based on example input-output pairs. They assume a given structure within the data and in supervised learning methods any particular input will have a known output and it is therefore easier to identify prediction errors.



(Source: Datafloq - https://datafloq.com/read/machine-learning-explained-understanding-learning/4478)

Examples:

* K-Nearest Neighbours Classification

K nearest neighbours is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure. KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. The k-nearest neighbour classification is a non-parametric in that it makes no assumption on the spread of the data within each class, making it a very simple method. After a positive integer value for k is specified, the class assignment of the k closest neighbours of each new point is considered. The new observation is then classified as belonging to the most prevalent classification of these entries.

* Linear Discriminant Analysis

LDA is a dimensionality reduction technique which is commonly used for the supervised classification problems.  It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. This method is widely used with machine learning and artificial intelligence.

Unsupervised Learning

Unsupervised machine learning algorithms infer patterns from a dataset without reference to known, or labelled, outcomes. Learning is similar to supervised learning in the sense that the goal is to create the most accurate model possible to complete an assigned task. However, unlike [supervised machine learning](https://www.datarobot.com/wiki/supervised-machine-learning/), unsupervised machine learning methods cannot be directly applied to a [regression](https://www.datarobot.com/wiki/regression/) or a [classification](https://www.datarobot.com/wiki/classification/) problem because you have no idea what the values for the output data might be, making it impossible for you to [train](https://www.datarobot.com/wiki/training-validation-holdout/) the algorithm the way you normally would.

Examples:

* K-Means Clustering

K-means clustering is seen as one of the simplest and popular unsupervised machine learning algorithms. It works by dividing data into clusters relevant to their characteristics. K-Means clustering may be seen as more popular to other clustering techniques such as hierarchical clustering as if variables are large, k-means is usually computationally faster. However, difficulties can exist when trying to predict k-values.

* Principal Component Analysis

Principal Component Analysis is a statistical procedure that uses an [orthogonal transformation](https://en.wikipedia.org/wiki/Orthogonal_transformation) to convert a set of observations of possibly correlated into a set of values of [linearly uncorrelated](https://en.wikipedia.org/wiki/Correlation_and_dependence) variables called principal components. PCA finds linear combinations of variables in the data which capture most of the data’s variation. Principal components are found by performing an eigen-decomposition of the data’s covariance matrix.

# Ethical Concerns

Ethics is defined as the moral principles that govern a person’s behaviour or the conducting of an activity. Everyday across the globe, an array of organisation’s handle large amounts of data on a daily basis, whether this is for analysis of one’s performance or the entire company. Today, there is ready data availability, cheap storage capacity, and powerful tools for extracting information from data have the potential to significantly enhance the human condition. However, as with all advanced technologies, this comes with the potential for misuse. The legislation that exist surrounding the capture of data is complex. In this section I will outline the current legislation that is in place and the different measures corporations take to protect their employees from undesirable data practice

**Legislation**

As data becomes more readily available with the development of technology, legislation therefore becomes stricter as more laws are put in place to accommodate this large storage of data. The [European Union](https://en.wikipedia.org/wiki/European_Union) has the [General Data Protection Regulation](https://en.wikipedia.org/wiki/General_Data_Protection_Regulation), in force since May 25, 2018. This outlines that Controllers of personal data must put in place appropriate technical and organizational measures to implement the data protection principles. Examples of a brief outline of the laws put in place by this legislation state that:

1. Fairly and lawfully processed.
2. Processed for limited purposes.
3. Adequate, relevant and not excessive.
4. Accurate.
5. Kept no longer than necessary.
6. Processed in accordance with the data subject's rights.
7. Secure.
8. Transferred only to countries with adequate protection.

**Ethics of data collection**

The best modern example for the subject of ethics regarding data collection is social media platforms. “Facebook changes its policies frequently, like a child switching up the rules to a game of his own making: a sly update when it’s personally beneficial, a knee-jerk pivot when in trouble. Everyone else—the platform’s users and advertisers—is left scampering and contorting to comply” (Sarah Steimer).

We have seen with the recent Cambridge Analytica scandal that there is strict legislation in place today when one’s data is being collected, and also that big corporations such as Facebook may be mistreating your personal data.

(Source: Facebook- https://www.facebook.com/)

Unobtrusive data collection is a major issue, not just with social media sites but also within the workplace. Referring back to Hackystat, developers may be uncomfortable with data being heedlessly collected. It is important that developers are aware of the protection of their personal data and access to the way their information is collected, to reduce the potential for conflict in the workplace. It is also of key importance that the workplace is only collecting data directly relevant to the work they’re doing.

Today in Ireland, reported data breaches have nearly tripled in the last 5 years, reinforcing the prevalence of the ethical considerations involved in data measurement. The ever-evolving environment surrounding data collection and lawful data usage, makes this is an issue that organisations need to target to minimise to see a reduction in cases of illicit data use.

# Conclusion

The software engineering process can be measured in the several different ways I have outlined in this report. Measurable data, computational and algorithmic platforms available and ethics are just some of the parameters for measuring a software engineer. The continuing development of technologies and computational intelligence will change how we obtain and analyse data. It is likely that the process of data collection and analysis will become quicker and data more readily available, however, the workplace needs to be in sync with developers in relation to the ethics in an ever-advancing workplace.

# Bibliography

* Stephen H. Kan. 2002. Metrics and Models in Software Quality Engineering (2nd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
* Fenton, N. E., and Martin, N. (1999) "Software metrics: successes, failures and new directions." Journal of Systems and Software 47.2 pp. 149-157.
* P.M. Johnson et al., “Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined,” Proc. 25th Int’l Conf. Software Eng. (ICSE 03), IEEE CS, 2003, pp. 641–646.
* Ian Somerville, 2008, “Software Engineering History” [online] Available at:

<https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/index.html>

* 2019, “What are the methods of data collection?” [online] Available at : <https://www.lotame.com/what-are-the-methods-of-data-collection/>
* 2015, “What is data, and why is it important?” [online] Available at: <https://www.import.io/post/what-is-data-and-why-is-it-important/>
* Stephen H. Kan, 2002, “Software Quality Metrics Overview” [online] Available at: <http://www.informit.com/articles/article.aspx?p=30306&seqNum=5>
* Paul Ventura, “Struggles of manual data collection” [online] Available at: <https://www.shoptech.com/blog/manual-data-collection-struggles/>
* “Software Quality Metrics” [online] Available at: <https://www.tutorialspoint.com/software_quality_management/software_quality_management_metrics.htm>
* Lokesh Raj, 2018, “Code Churn – A Magical Metric for Software Quality” [online] Available at: <https://dzone.com/articles/code-churn-a-magical-metric-for-software-quality>
* 2018, “15+ Useful Agile Metrics in Scrum & Kanban: Measure Quality, Productivity & Performance” [online] Available at: <https://www.intellectsoft.net/blog/agile-metrics/>
* “10 Powerful Agile Metrics – and 1 Missing Metric” [online] Available at: <https://www.sealights.io/software-development-metrics/10-powerful-agile-metrics-and-1-missing-metric/>
* 2019, “What is Code Churn and how to reduce it” [online] Available at: <https://textexpander.com/entry/what-is-code-churn-and-how-to-reduce-it/>
* 2019, “Personal Software Process” [online] Available at: <https://en.wikipedia.org/wiki/Personal_software_process>
* E.Novoseltseva, 2017, “Benefits of using GitHub” [online] Available at: <https://apiumhub.com/tech-blog-barcelona/using-github/>
* 2018, “Code Climate” [online] Available at: <https://www.blissfully.com/code-climate/>
* Phillip Johnson, “Hackystat” [online] Available at: <http://csdl.ics.hawaii.edu/research/hackystat/>
* 2018, “What is Computational Intelligence?” [online] Available at: <https://cis.ieee.org/about/what-is-ci>
* Achim G. Hoffmann, 2015, “Artificial and Natural Computation” [online] Available at: <https://www.sciencedirect.com/topics/computer-science/evolutionary-computation>
* “Supervised Learning” [online] Available at: <https://en.wikipedia.org/wiki/Supervised_learning>
* “K Nearest Neighbors” [online] Available at: <https://www.saedsayad.com/k_nearest_neighbors.htm>
* Tavish Srivastava, 2014, “Introduction to k-Nearest Neighbors: A powerful machine Learning Algorithm” [online] Available at: <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>
* Raman\_257, “ML/Linear Discriminant Analysis” [online] Available at: <https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/>
* “Unsupervised Machine Learning” [online] Available at: <https://www.datarobot.com/wiki/unsupervised-machine-learning/>
* “Principal component analysis” [online] Available at: <https://en.wikipedia.org/wiki/Principal_component_analysis>
* David J.Hand, 2018, “Aspects of data ethics in a changing world: Where are we now?” [online] Available at: <https://www.liebertpub.com/doi/full/10.1089/big.2018.0083>
* “General Data Protection Regulation” [online] Available at: <https://en.wikipedia.org/wiki/General_Data_Protection_Regulation>
* Sarah Steimer, 2018, “The murky ethics of data gathering in a post-Cambridge Analytica world” [online] Available at: <https://www.ama.org/marketing-news/the-murky-ethics-of-data-gathering-in-a-post-cambridge-analytica-world/>